NTNU – Trondheim
Norwegian University of
Science and Technology

Automatic Fish Classification

Using Image Processing and Case-Based Reasoning

Lars Moland Eliassen

Master of Science in Computer Science

Submission date:	June 2012
Supervisor:	Theoharis Theoharis, IDI
Co-supervisor:	Agnar Aamodt, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

Counting and classifying fish moving upstream in rivers to spawn is a useful way of monitoring the population of different species. Today, there exist some commercial solutions, along with some research that addresses the area. Case-based reasoning is a process that can be used to solve new problems based on previous problems. This thesis studies the possibilities of combining image processing techniques and case-based reasoning to classify species of fish which are similar to each other in both shape, size and color. Methods for image preprocessing are discussed, and tested. Methods for feature extraction and a case-based reasoning prototype are proposed, implemented and tested with promising results.

Sammendrag

Telling og klassifisering av fisk som svømmer oppstrøms i elver for å gyte er en nyttig måte å overvåke bestanden av ulike arter på. I dag eksisterer det noen kommersielle løsninger, sammen med noe forskning som tar for seg problemområdet. Case-based reasoning er en prosess som kan brukes til å løse nye problemer basert på tidligere problemer. Denne oppgaven studerer mulighetene for å kombinere bildebehandlingsteknikker og case-based reasoning for å klassifisere arter av fisk som er lik hverandre i både form, størrelse og farge. Metoder for bildebehandling diskuteres, og testes. Metoder for egenskapsuttrekking og en prototype for case-based reasoning blir foreslått, implementert og testet med lovende resultater.

Preface

This master thesis is the result of work performed from January 15th to June 10th 2012. It has been carried out at the Department of Computer and Information Science (IDI), at the Norwegian University of Science and Technology (NTNU) in collaboration with Trollhetta AS.

I would like to thank my supervisors Theoharis Theoharis and Agnar Aamodt for guidance and good advice. I would also like to thank Ketil Bø at Trollhetta AS for useful input and valuable domain knowledge.

I would also like to thank my loyal Mac for not dying on me this year either, and my coffeemaker for helping me through the early mornings and late nights (of which there has been few). And last but not least, i would like to thank my dad for proof-reading and correcting equations he did not even understand, and my mom for everything else.

Lars Moland Eliassen

Trondheim, June 5, 2012

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Counting and classifying fish can be useful for several settings. One of which is the monitoring of volume and species of fish going upriver to spawn during spring and summer. This monitoring can provide useful overview of the population of each species in the area.

Today, there exist few commercial solutions to automatically count and classify species. The VAKI Riverwatcher [1] is the market leader. The Riverwatcher uses infrared scanning to count fish but the classification has to be done manually based on images that are saved for each fish counted. The system depends on very specialized and expensive equipment in the form of infrared sensors and digital video.

The goal for this thesis is to study the use of image processing and case-based reasoning for automatically determining species of fish based solely on digital video. The three species addressed for classification are among the most common species in Norwegian rivers and streams: Arctic Salmon, Brown Trout and Arctic Char.

A set of fish videos has been provided by the fisherman's association of the Laksådal river in Gildeskål, Norway. These videos are captured in a closed environment, where the fish swim trough a magnetic gate which activates a camera, recording a clip of the passing fish. As can be seen in Figure 1.3, the quality of the videos is not sufficient separate fish which are similar in shape and color. Regardless, these videos has been used for testing the preprocessing, but for testing the classification an ad-hoc dataset is used. However, the

[1]http://www.vaki.is/Products/RiverwatcherFishCounter/

closed-environment in which these videos are captured is used to define a set of constraints for the type of videos on which the classification should be able to operate:

- There should be only one fish in the image.
- The entire fish should be visible in the image.
- The fish should be captured from the side.
- The vertical rotation of the fish should be limited to $\sim$ 45 degrees.
- The distance from the camera to the fish should be no greater that one meter.
- The background should be uniform and static.

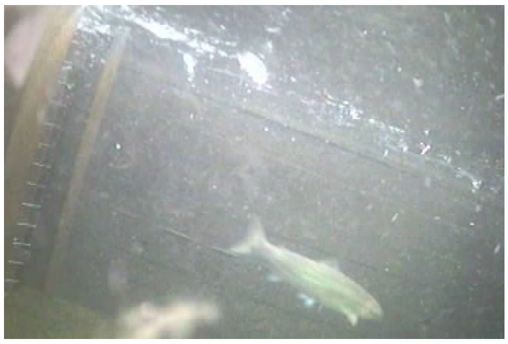


Figure 1.1: Poor quality image.

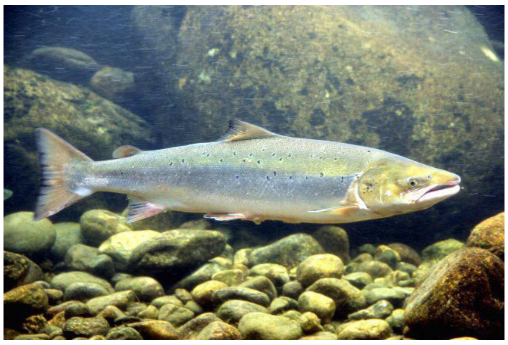


Figure 1.2: Good quality image [a].

[a]Image by Morten Harangen (reprinted with permission)

Figure 1.3: A poor quality image from the set of fish videos, and a good quality image for reference (except for the background not being uniform).

Separating the three species from each other is not always trivial, even for a trained human eye. Char is the easiest to distinguish as its pattern is inverted compared to the pattern on the other two species, with spots that are brighter than the rest of the body. Additionally, its belly turns red during the spawning period. Salmon and trout are not as easy to separate from each other, but there exist features that can be used to distinguish them. The caudal peduncle tends to be fuller on the trout, while it is more lean on the salmon. The caudal fin is often concave in salmon, while it is usually straight or convex in trout. Also, while the salmon and trout are quite similar in colors, the trout often has dark spots all the way down to the abdominal region, while the salmon's spots end at the lateral line.

A human expert is likely to use a combination of features to distinguish between the species, based on knowledge gained from previous experience with the species. One way to replicate this expert behavior is to use case-based reasoning (CBR). CBR uses past problems to solve new problems, much like the way the human expert will use previous experience to distinguish the species.

Case-based reasoning has the advantage of adapting to the available data and learning from previous experience, as opposed to e.g. a decision tree that relies only on the knowledge of the programmer. While the decision tree will always use the same rules, case-based reasoning will adapt over time to make use of the features that are most important when separating classes from each other.

The remainder of this thesis is structured as follows. In Chapter 2, a survey of related research in the field of automatic fish classification and other relevant fields is given. Chapter 3 present the tools used, while Chapter 4 explains the methods used for the proposed system. Chapter 4 is split in three parts, preprocessing, feature extraction and case-based reasoning. In Chapter 5 the results are presented, and Chapter 6 discusses them. Chapter 7 is devoted to future work.

Chapter 2

Related research

No previous work in the area of fish classification using case based reasoning has been found. Therefore, this chapter has been separated in two sections, Section 2.1 addresses fish classification in general, while Section 2.2 addresses image interpretation with case-based reasoning.

2.1 Fish classification

There are several existing works which address the problem of automatic fish classification in general. Common for most of them are that they extract symbolic or numerical features, which are then used in a reasoning process to classify the fish. Following is a description of some of the previous work done in this area.

Saue [2003] proposes a way to separate salmon and trout using the euclidean distance from a set of quantified features. The features are the width of the caudal peduncle, curve of the caudal fin and the pattern below the lateral line. How the features are extracted is only vaguely described in the report. The extracted feature values are compared to the mean value for each species, calculated by a training set, and the species with the smallest euclidean distance is the found class. The proposed solution is only tested on the 10 images in the training set, along with one salmon, but all images tested are classified correctly.

Akgül [2003] propose a system based on eigenspaces. The proposed system is invariant to rotation, and is able to correctly separate nine different species with a success rate of 76% on a dataset containing five images per fish.

Rodrigues et al. [2010] propose a system based on two robust feature extraction techniques: Scale-Invariant Feature Transform (SIFT) and Principal Component Analysis (PCA), and two immunological algorithms: Artificial Immune Network and Adaptive Radius Immune Algorithm. The system achieves a 92% success rate for six species of dead fish. The dataset used contains three fish per species with six images per fish. The system is also tested on four live fish where it achieves a 92% with a dataset containing one fish per species and twelve images per fish.

Lee et al. [2003] propose a classification technique based on six shape descriptors. These are the adipose fin, anal fin, caudal fin, head and body shape, size and length/depth ratio of the body. Testing shows a 100% success rate for seven species, but the test data is very limited with only 22 images that together represent nine species.

Zion et.al. present a shape based classification system for separating three species, in Zion et al. [2000] the system is tested on dead fish, while it is tested on live fish in Zion [1999]. For dead fish, the accuracy is 96-100% for three species represented by a dataset containing 143 images, almost evenly distributed among the species. For live fish, the accuracy is 91-100% for three datasets containing 96, 140 and 146 images. The method used in both papers is based on Hu invariant moments for the whole body, the head and the tail.

In Zion et al. [2007] the same authors propose a continuation of the system, as the original system proved to be to sensitive to water opaqueness and fish motion. The improved system uses contour landmarks, which are significant points along the contour, and the relation between them. The improved system shows a success rate of 98-99% with two datasets of 1701 and 2164 images.

Rova et al. [2007] propose a system based on deformable templates for separating two species with similar shape. The system uses shape contexts (Belongie et al. [2002]), distance transforms, iterative warping (P.F.Felzenszwalb and Huttenlocher [2005]) and a support vector machine(Cortes and Vapnik [1995]) for texture classification to classify the two species. A success rate of 90% is achieved with cross-validation testing on a dataset containing 320 images, 160 for each species.

What differentiates the specific problem addressed in this thesis from most of the problems above, is the visual similarity between the species to classify. Two of the problems does resemble the problem at hand, Saue [2003] proposes a way to separate salmon and trout, which are two of the species addressed in this thesis. Rova et al. [2007] separate two other species, that have similar shapes to eachother. Both achieve good results, but none of them have been thoroughly tested.

2.2 Case-based reasoning and image interpretaion

Perner [2001] argues for the use of case-based reasoning in image interpretation. The use of CBR in image interpretation is far from widespread despite that, according to this paper, it has several advantages over traditional image interpretation techniques. Among the advantages is the ability overcome problems regarding limited number of observations, environmental influence and noise. As CBR does not rely deeply on a well-formulated domain theory, it can be a good approach for image interpretation systems.

In Perner et al. [2006], a summary of the use of case-based reasoning in image processing and interpretation is made. Below is a short description of some of the papers, which are considered most relevant for this thesis.

Grimnes and Aamodt [1996] propose a system for medical image interpretation containing two case-based reasoners. One for segment identification, the *Segment ImageCreek* (SICT). The other for interpretation of the entire image, the *Wholistic ImageCreek* (WICT). The cases in the segment case-base SICT are used as indexes for the image interpreting reasoner WICT.

Micharelli et al. [2000] use wavelet transforms (Charles [1992]) to find image properties, which are stored as cases in a case-base. The proposed architecture is applied to mobile robots, making them capable of classifying objects in order to navigate.

Perner and Bühring [2004] propose a case-based object recognition for detecting objects of interest in a complex background where several objects are present. Because the object to be recognized can vary greatly in appearance, a generalization of the object is impossible, a problem which can be overcome with a large amount of cases for each object.

Aasen [2006] proposes a CBR surveillance system that interprets video and raises an alarm if necessary. The system consists of two separate modules, one for image processing, including feature extraction, and one case-based reasoning module. It's programmed in C++ for the surveillance system TrollEye by

Trollhetta AS[1]. For the case-based reasoner proposed in this thesis, all of Aasens functionality has been used as a staring point, based on the source code and pseudo code from his thesis.

[1]http://www.trollhetta.com

Chapter 3

Tools

OpenCV 2.0 with Python bindings is used for the feature extraction along with most of the preprocessing methods and the CBR prototype. The background subtraction is implemented in OpenCV with C++ bindings.

3.0.1 Python

Python [1] is an open-source, high-level programming language. It allows for use of multiple programming paradigms, ranging from object-oriented to functional programming. In addition to OpenCV, two libraries for Python have been utilized, NumPy and Matplotlib. NumPy [2] is an open-source math library. It includes support for arrays, matrices and a large set of mathematical functions for these. Matplotlib is a 2D/3D plotting library [3].

The reason for using Python is its large standard library and, because it is a scripting language, it is very well suited for rapid prototyping. It is not as fast as C++ and Java, but the advantage of quick developing is considered more important the disadvantage of slower processing.

[1]http://www.python.org/
[2]http://numpy.scipy.org
[3]http://matplotlib.sourceforge.net/

3.0.2 OpenCV

The Open Source Computer Vision Library (OpenCV) [4] is an image processing library originally developed by Intel, now maintained by Willow Garage [5]. It is open-source, multi-platform and while originally supported only by C and C++, there are wrappers available for several languages including: C#, Python, Ruby and Java.

[4]http://opencv.willowgarage.com
[5]http://willowgarage.com

Chapter 4

Method

To interpret images using case-based reasoning, a three stage process as seen in Figure 4.1 is proposed. First, the images have to preprocessed and segmented to separate the fish to be classified from the rest of the image. The preprocessing and segmentation stage is discussed in Section 4.1. When the fish has been separated, features that can be used to classify the fish have to be extracted. The feature extraction stage is discussed in Section 4.2. The last stage is the classification itself, through case-based reasoning. The CBR is discussed in Section 4.3.

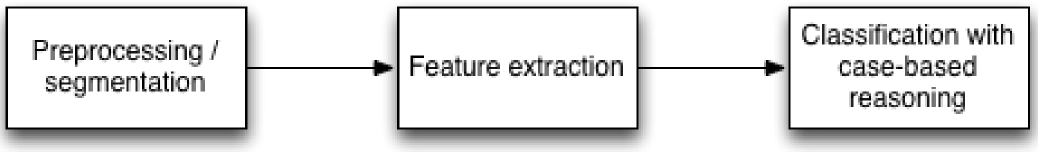
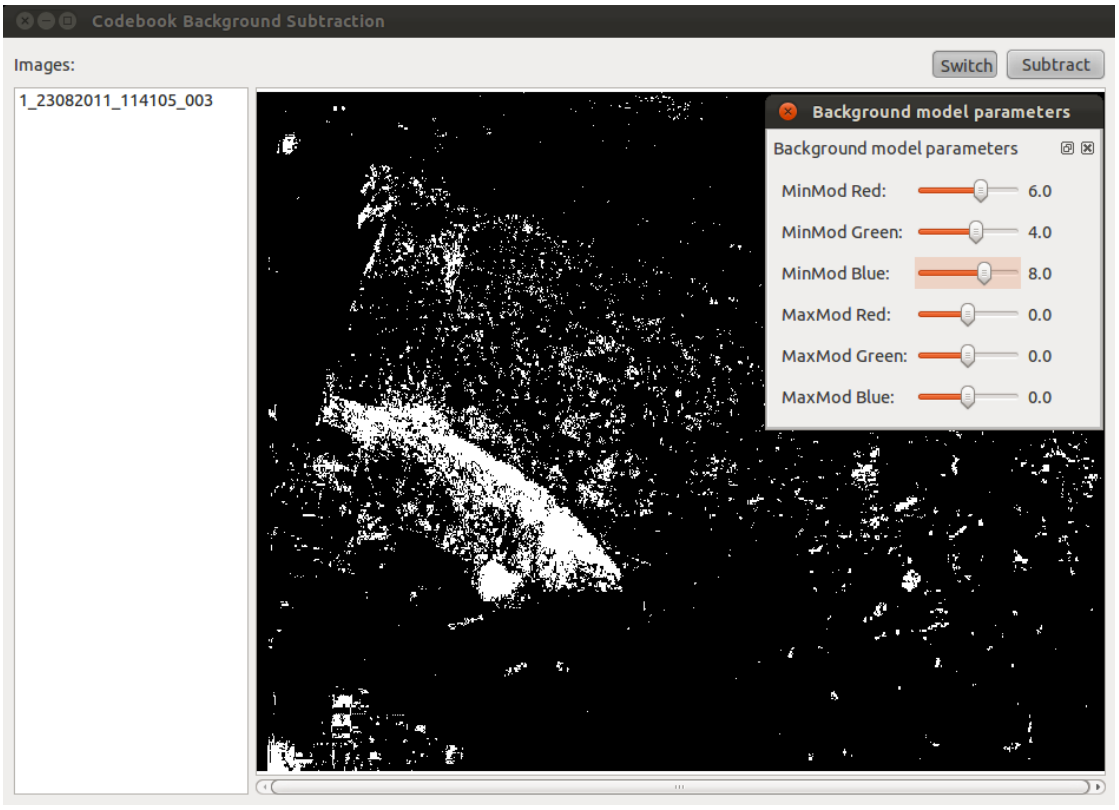

Preprocessing / segmentation
Feature extraction
Classification with case-based reasoning

Figure 4.1: Proposed three stage process for classifying fish.

4.1 Preprocessing

The preprocessing stages proposed has not been implemented from scratch for this thesis. For the background subtraction algorithms, the implementations from Fauske et al. [2009] have been used. The implementation of the codebook algorithm (Kim et al. [2005]) has been adapted to create a graphical application using Qt [1] for the graphical user interface and for reading and writing images. With this application, the parameters can be tuned with sliders, and the result is shown in real-time. A screenshot from the application can be seen in Figure 4.2.


Codebook Background Subtraction
Images:
Switch
Subtract
1_23082011_114105_003
Background model parameters
Background model parameters

MinMod Red: 6.0
MinMod Green: 4.0
MinMod Blue: 8.0
MaxMod Red: 0.0
MaxMod Green: 0.0
MaxMod Blue: 0.0

Figure 4.2: Screenshot from the codebook application.

The rest of preprocessing methods are parts of the OpenCV library, so this section describes the general theory behind them. Active contour models, border following and Hu moments are based on the original papers. For the morphology and the raw and central moments, Gonzalez and Woods [2008] is used for reference.

4.1.1 Background subtraction

In background subtraction, the foreground and background of an image are separated from each other using a background model. Everything in the image

[1]http://qt.nokia.com

that cannot be classified as background in accordance to the background model is classified as foreground. Fauske et al. [2009] compare three learning based background subtraction techniques with respect to shadow and noise. Two of these three techniques have been tested on the available video data to see how well they perform in an underwater environment.

Codebook algorithm The background model in the Codebook algorithm (Kim et al. [2005]) consists of a codebook for each pixel, each codebook has a number of codewords. A codeword is a valid color-area for the corresponding pixel being part of the background. If a pixel in the subtraction is inside one of its codewords boundaries it is part of the background. Each codeword consists of the following values:

- The mean of the R,G and B values: $\bar{R}$,$\bar{G}$, $\bar{B}$.
- The minimum and maximum brightness: $\hat{I}$,$\check{I}$
- The longest interval between two occurrences: λ
- The first and last access times: p, q

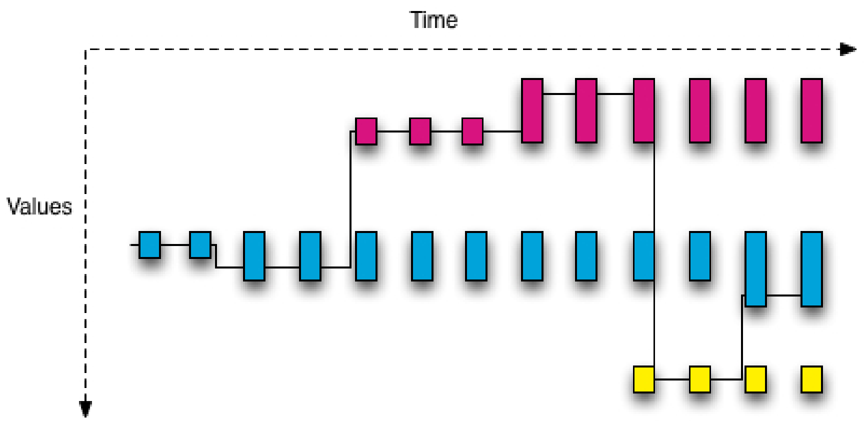


Figure 4.3: Codebook formation. The colored boxes represent codewords, the black line represent the pixel value at a given time. As a given pixel changes value over time the codewords grow, and new codewords are created.

During learning, a codeword is updated if its corresponding pixel value in the current frame is within its interval. If no matching codeword is found, a new codeword is created (Figure 4.3).

After the learning process, all stale entries are removed. A stale entry is a codeword that is accessed for a given period of time, e.g. half of the learning runtime. When the stale entries has been removed, the model is complete, and the subtraction can begin. The subtraction works in the same way as the learning, but instead of making a new codeword when none is found, the pixel is marked as foreground.

A advantage with the codebook model for underwater images is its adaptability to noise. The fact that it can have different intervals of color values for the background helps coping with underwater noise, such as air bubbles.

Horprasert et al.'s statistical approach. Horprasert et al. [2000] introduce a color model in the three-dimensional RGB color space. Each pixel in a frame is classified as either "Foreground", "Background", "Shadowed Background" or "Highlighted Background" depending on where it is located in the color space.

For each pixel, the following values are stored in the background model:

- The expected value for the i'th pixel: $E_i = \{\mu_i^R, \mu_i^G, \mu_i^B\}$.
- The brightness distortion: α.
- The cromaticity distortion: CD.

The expected pixel value E_i is a point in the 3D-space, as shown in Figure 4.4. The brightness distortion $\alpha E_i$is the intersection between the cromaticity line OE and the orthogonal from new pixel value I_i to the line. Chromaticity distortion is the distance from I_i to OE. The values are calculated for each frame in the training process, and then normalized to find a single threshold value. The threshold is found by building a histogram of the normalized values.

For the subtraction itself, new normalized brightness and chromaticity distortion values are calculated for each pixel in the captured frame. A pixel is classified by the following taxonomy:

- Background if both $\hat{\alpha}_i$ and $\hat{CD}_i$ are within a threshold compared to the background model.

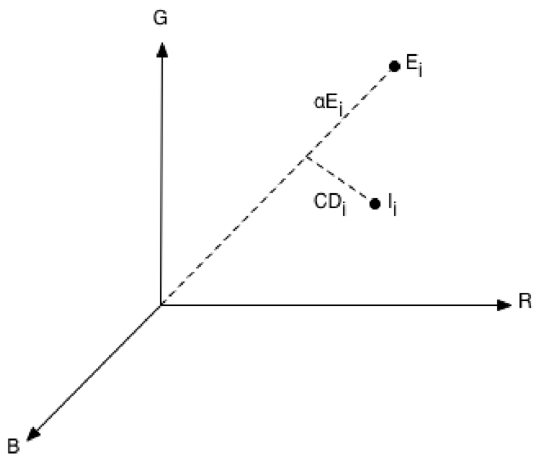


Figure 4.4: The RGB color model proposed by Horprasert et al.

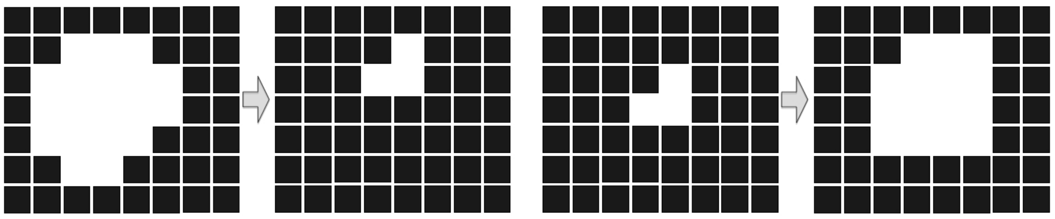
- Shaded background if $\hat{CD}_i$ is within the threshold and $\hat{\alpha}_i$ is below.
- Highlighted background if $\hat{CD}_i$ is within the threshold and $\hat{\alpha}_i$ is above.
- Foreground if $\hat{CD}_i$ is outside the threshold.

4.1.2 Morphological cleanup

Morphological transformations have several applications in image processing, two of which are noise removal and joining or separating image elements. Morphology contains two basic transformations, erosion and dilation. Erosion will, as the name implies, erode bright areas in the image, making them smaller, while dilation does the opposite, dilate bright areas, making them larger.

Both use kernels with an anchor point. The kernels can be any geometrical shape and the anchor point does not have to be the center, it can be anywhere in the kernel. Erosion will replace the anchor value with the minimum intensity value under the kernel, while dilation replaces the anchor value with the maximum intensity value under the kernel. The effects of erosion and dilation on a binary image can be seen in Figure 4.5.

Morphological opening (Equation 4.1) and closing (Equation 4.2) are composite transformations using the erosion and dilation operators. Opening will remove small objects, which usually represent noise, from the image. Closing will fill



(a) Erosion (b) Dilation

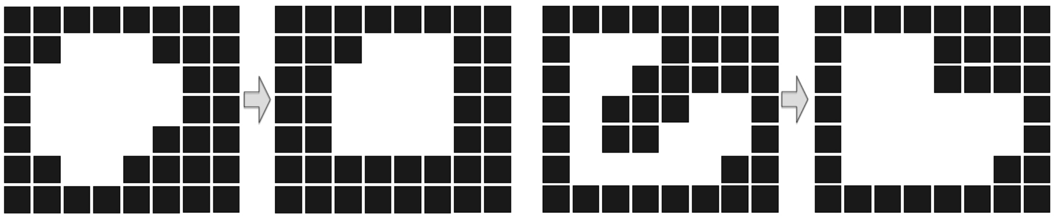
Figure 4.5: The effects of applying erosion and dilation to a binary image.

holes in the foreground objects, making them more complete. The effects of performing opening and closing can be seen in Figure 4.6

$$A \circ B = (A \ominus B) \oplus B \tag{4.1}$$

$$A \bullet B = (A \oplus B) \ominus B \tag{4.2}$$

Where A is the image, B is the kernel, $\ominus$ is the erosion and and $\oplus$ the dilation.



(a) Opening (b) Closing

Figure 4.6: The effects of applying opening and closing to a binary image.

4.1.3 Finding contours by border following

To extract contours from a binary image, a border following algorithm is used. The algorithm which is part of OpenCV's image processing algorithms, is an implementation of Suzuki and Abe [1985]. There are two versions of the algorithm, one for topological analysis, and one for extracting only the outermost contour. Both exist in OpenCV, but only the latter is utilized in this thesis. The

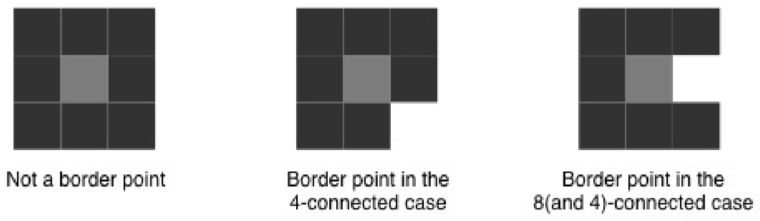

Not a border point
Border point in the 4-connected case
Border point in the 8(and 4)-connected case

Figure 4.7: Border points

algorithm works by raster scanning the image until an outer border point (x, y) is found. A border point is, in the 4-connected case, defined as a 1-pixel having a 0-pixel in its 8-connected neighborhood. In the 8-connected case, it is defined as a 1-pixel having a 0-pixel in its 4-connected neighborhood (See Figure 4.7 for examples). A border point (i, j) belongs to the outer border if all the pixels $(i, 1), (i, 2), ..., (i, j - 1)$ are 0-pixels, or the most recently scanned border point (i, h) is on the outer border and $(i, h + 1)$ is part of the background. The background of the image are all pixels (4-/8-)connected to the frame, where the frame is defined as the top-/bottom rows, and left-/right columns of the image.

4.1.4 Active contour models

Active contour models, also known as Snakes (Kass et al. [1988]), is a technique that can be used to find or improve contours in an image. The model needs an external constraint as a starting point. It then attempts to minimize the sum of internal and external energy to fit the contour to an image feature as illustrated in Figure 4.8. The energy is minimized by iterative gradient descent. The image energy moves the model towards intensity changes (edges) using a regularized gradient, while the internal constraints keep the model smooth and continuous, and reduce the presence of sharp corners. The total energy of the snake can be defined as:

$$E_{total} = E_{image} + E_{internal} \tag{4.3}$$

If the image intensity in (x, y) is denoted $I(x, y)$, the image energy E_{image} can, in its simplest form, be defined as the gradient intensity at each contour point:

$$E_{image} = -||\nabla I(x_i, y_i)|| \tag{4.4}$$

$E_{internal}$ is the internal energy of the spline due to bending and E_{image} is the image forces on the spline. If the snake contour is represented parametrically

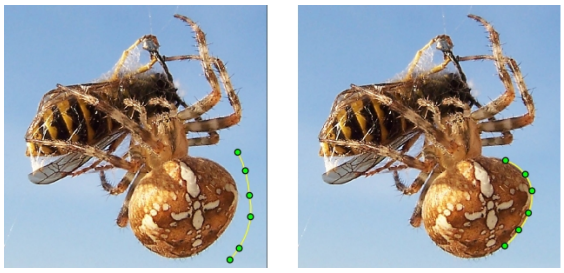


Figure 4.8: Applying a snake to an image. [2].

as $c(s) = (x(s), y(s))$, where x(s) and y(s) are coordinates along the contour, the internal energy $E_{internal}$, can be defined as:

$$E_{internal} = E_{cont} + E_{curv} \tag{4.5}$$

where E_{cont} is the continuity energy, defined as the first derivative:

$$E_{cont} = ||\frac{dc}{ds}||^2 \tag{4.6}$$

or, in discrete form, where i is the i't contour point, as:

$$E_{cont} = \sum_{i=0}^{n} (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2 \tag{4.7}$$

and E_{curv} is the curvature energy, defined as the second derivative:

$$E_{curv} = ||\frac{d^2c}{d^2s}||^2 \tag{4.8}$$

or, in discrete form, where i is the i't contour point, as:

$$E_{curv} = \sum_{i=0}^{n} (x_{i-1} - 2x_i + x_{i+1})^2 + (y_{i-1} - 2y_i + y_{i+1})^2 \tag{4.9}$$

[2]Original image by Jürgen Howaldt, via Wikimedia Commons

4.1.5 Image moments

4.1.5.1 Raw Moments - Finding the centroid of a binary image

For binary images the area and center of gravity (centroid) can be found using raw moments. Raw moments in a digital image ,where $I(x, y)$ is the image function, are defined as :

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y) \tag{4.10}$$

If the object pixels in a binary image are represented by 1, it can be observed from Equation 4.10 that the raw moment M_{00} will represent the area of the object, as $i = 0$ and $j = 0$ will simply sum the object pixels. The raw moment M_{10} represents the sum of all the object's x-values, and the moment M_{01} the sum of all the objects y-values. The mean x and y, which give the centroid of the object, can be calculated by dividing M_{10} and M_{01} by the area M_{00} (Equation 4.11).

$$Centroid : (\bar{x}, \bar{y}) = (M10/M00, M01/M00) \tag{4.11}$$

4.1.5.2 Central Moments - Finding the rotation of a binary image

Raw moments are not translation invariant, so the location of the image objects will affect the values of the raw moments. Central moments on the other hand are translation invariant, so the location of the image objects will not affect the moments values. Central moments in a digital image, where $I(x, y)$ is the image function, are defined as:

$$\mu_{ij} = \sum_x \sum_y (x - \bar{x})^i (y - \bar{y})^j I(x, y) \tag{4.12}$$

As can be observed from Equation 4.12, the central moment μ_{00} for digital images represent the area of the object as it will ,similar to M_{00}, simply sum the object pixels. The central moments μ_{20} and μ_{02}, which represent variance, can be calculated from the raw moments (Equations 4.13 and 4.14).

$$\mu 20 = M_{20} - \bar{x} M_{10} \tag{4.13}$$

$$\mu 02 = M_{02} - \bar{y} M_{01} \tag{4.14}$$

The rotation of the binary shape (ϕ) can be found using the formula:

$$\phi = \frac{1}{2} \arctan \left(\frac{2\mu'_{11}}{\mu'_{20} - \mu'_{02}} \right) \tag{4.15}$$

where: $\mu'_{11} = \frac{\mu_{11}}{\mu_{00}}$, $\mu'_{20} = \frac{\mu_{20}}{\mu_{00}}$ and $\mu'_{02} = \frac{\mu_{02}}{\mu_{00}}$

4.1.5.3 Hu Set of Invariant Moments

The Hu set of invariant moments (Hu [1962]) is invariant to translation,rotation and scale. Hu moments are therefore well suited to extract shape information and coarsely separate different shapes.

The seven Hu moments are calculated as follows:

1. $I_1 = \eta_{20} + \eta_{02}$
2. $I_2 = (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2$
3. $I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta 03)^2$
4. $I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$
5. $I_5 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03}^2] +$

   $(3\eta 21 - \eta 03)(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$
6. $I_6 = (\eta_{20} - \eta 02)[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta 03)^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})$
7. $I_7 = (3\eta_{21} - \eta 03)(\eta_{30} + \eta 12)[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta 03)^2] -$

   $(\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$

The scale invariant moment of order ij , η_{ij} is defined as:

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{(1+\frac{i+j}{2})}} \tag{4.16}$$

where μ_{ij} is the central moment of order ij.

For comparing the moments of two shapes, Equation 4.17 is used. The equation is found in the *MatchShapes* function in OpenCV [3].

$$D(A,B) = \sum_{i=1...7} \frac{|m_i^A - m_i^B|}{|m_i^A|} \tag{4.17}$$

where $m_i^A = sign(I_i^A) \cdot log\ h_i^A$ and $m_i^B = sign(I_i^B) \cdot log\ I_i^B$, and I_i^A, I_i^B is the i'th Hu moment for shapes A and B.

4.2 Feature extraction

In order to use case-based reasoning for image classification, a set of numerical features have to be extracted. Five features of the fish are quantified, and their numerical values are used in the case-based reasoning to separate the three species. The selected features are based on shape, pattern and color information.

4.2.1 Locating and measuring the caudal peduncle

The ratio between the width of the caudal peduncle and the length of the fish is the first feature extracted. According to Penthon [2005], the salmon has a thinner caudal peduncle than trout and char (Figure 4.12).

To locate the caudal peduncle(Figure 4.13), the binary fish image acquired through preprocessing is rotated to a horizontal position using the rotation angle acquired trough central image moments (Equation 4.15).

[3]The OpenCV 2.3 Python Reference Manual, page 104

[4]Image by Morten Harangen (reprinted with permission)

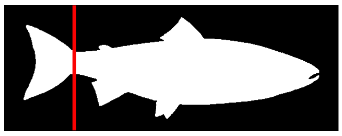


Figure 4.9: Salmon

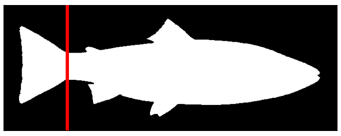


Figure 4.10: Trout

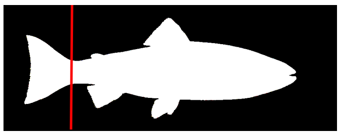


Figure 4.11: Char

Figure 4.12: A salmon, a trout and a char with their caudal peduncle indicated by a red line

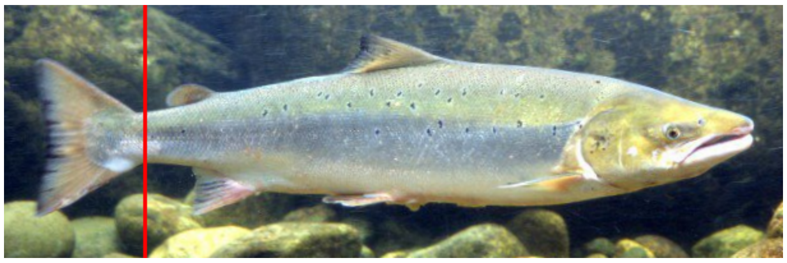


Figure 4.13: Position of the caudal peduncle for a salmon [4].

After rotating the binary fish-shape, the caudal peduncle is found by starting at $x = 0$ and iterating to $x = x_{centroid}$, then for each x, iterate from $y = 0$ and $y = y_{max}$ to $y_{centroid}$. This results in the distance from the top of the image to the uppermost object pixel, and the distance from the bottom of the image to the lowermost object pixel for the current x value (Figure 4.16). The complete function for locating and measuring the caudal peduncle can be seen in Algorithm 1.

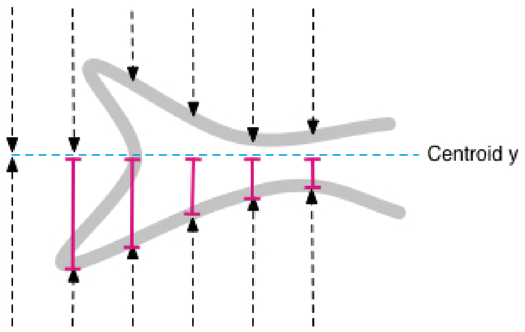


Figure 4.14: Locating the caudal peduncle

Algorithm 1 Locating the caudal peduncle

```
for $x$ from 0 to $x_{centroid}$ do

    $y_{up} \leftarrow image.height$
    $y_{down} \leftarrow image.height$

    for $y$ from 0 to $y_{centroid}$ do
        if $binaryimage[y, x]! = 0$ then
            $y_{up} \leftarrow x$
        end if
    end for

    for $z$ from $y_{max}$ to $y_{centroid}$ do
        if $binaryimage[z, x]! = 0$ then
            $y_{down} \leftarrow x.$
        end if
    end for

    if $2max(y_{down}, y_{up}) < min.height$ then
        $min.height \leftarrow 2max(y_{down}, y_{up})$
        $min.x \leftarrow x$
        $min.down \leftarrow y_{down}$
        $min.up \leftarrow y_{up}$
    end if
end for
$min.height \leftarrow min.down + min.up$
```

4.2.2 Estimating the curve of the caudal fin

Salmon tend to have a more concave caudal fin than both trout and char. To estimate the curvature, the caudal fin is sampled from the binary image of the fish. In the same way as for finding the width of the caudal peduncle, the fish is rotated to a horizontal position before sampling. The sampling is done similarly as for the caudal peduncle. Starting at $x = 0$ $y = 0$ and iterating in the x-direction until a object pixel is found or the location of the peduncle if no object pixel is found (see Figure4.15 and Algorithm 2). The sampled values are fitted to a second-degree polynomial, which will represent the caudal fin in the case-based reasoning. The fitting is done by ordinary least squares (Rao [1973]).

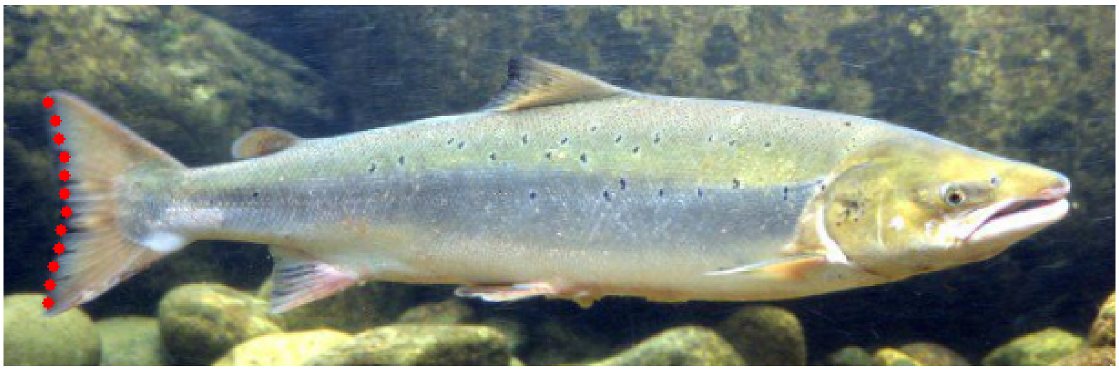


Figure 4.15: Sampled caudal fin for a salmon[5].

By sampling and fitting the curvature to a polynomial, the representation of the shape is not dependent on the shape's size or the caudal fin in the binary image being complete.

Algorithm 2 Sample caudal fin

```
for y from 0 to image.height do
    for x from 0 to peduncle.x do
        if $binaryimage[y, x] != (0, 0, 0)$ then
            $sampledpoints \leftarrow x.$
    end for
end for
```

[5]Image by Morten Harangen (reprinted with permission)

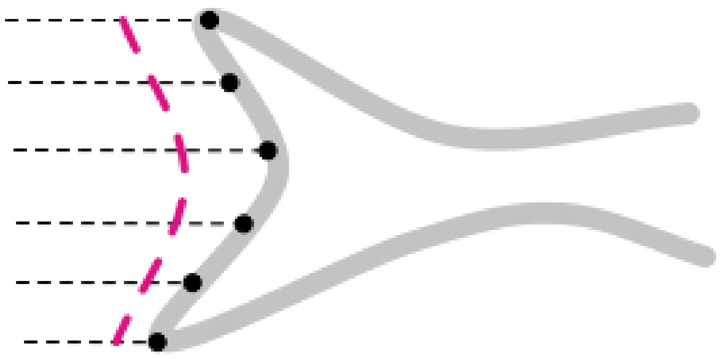


Figure 4.16: Sampling the caudal fin. Sampled points and iteration lines (black), estimated curve (red).

4.2.3 Quantifying the belly pattern

One feature that distinguishes trout from salmon is the presence of dark spots below the lateral line. Saue [2003] use standard deviation in an isolated area below the lateral line to measure the amount of spots. The presence of spots will produce a high standard deviation in the area. The method proposed by Saue is used for quantifying the pattern, but with some modifications regarding isolating the measure area.

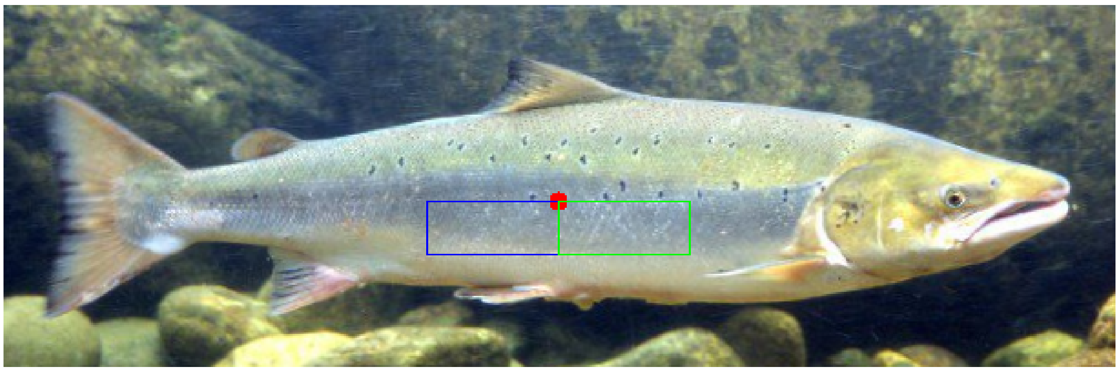


Figure 4.17: Pattern area. The green and blue squares indicates the two areas used for the histograms, the red dot is the center of gravity. [6]

Using the centroid, as found by Equation 4.15, as anchor point, two rectangular

[6]Original image by: Morten Harangen

areas are isolated from below the lateral line of the fish (Figure 4.17). The areas are defined as:

1. Left area:

   Bottom left corner:$x_{centroid} - (\frac{w}{2}), y_{centroid} - w\frac{w}{2}$
   Top right corner: $x_{centroid}, y_{centroid}$

2. Right area:

   Bottom right corner:$x_{centroid} + (\frac{w}{2}), y_{centroid} - w\frac{w}{2}$
   Top left corner: $x_{centroid}, y_{centroid}$

4.2.4 Quantifying the redness of the fish

As stated in Chapter 1 one of the easiest ways to separate char from salmon and trout is the redness of the fish, especially below the lateral line. A simple approach for quantification of the redness is to count the number of red pixels and the total number of pixels, to get a relation between red and non-red pixels (Figure 4.18.

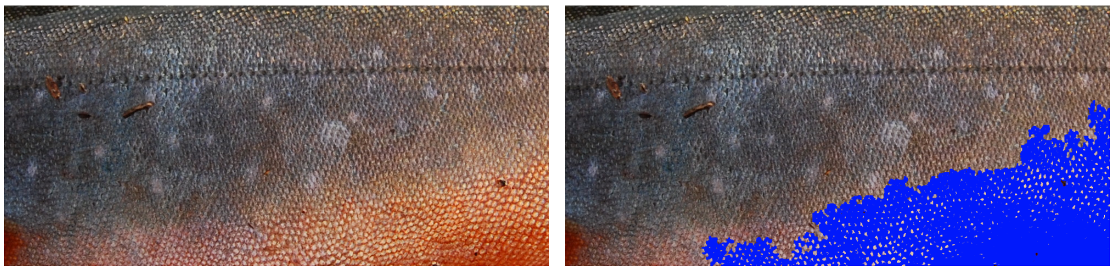

(a) Original (b) Red areas

Figure 4.18: The red area in a partial image of a char [7]. The blue area in 4.18b show the pixels from 4.18a that are defined as red.

To count the number of red pixels the following definition for a red pixel is used:

[7]Original image by: Petr Brož, via Wikimedia Commons

Definition Let P be a pixel in the RGB color-space. For a pixel to be considered red, it's R component's value has to be more than 10% greater than the averaged value of the B and G components.

When the pixels are counted, the binary image acquired trough segmentation is used as a mask so that no background pixels are counted. A pixel in the color image is only counted if the corresponding pixel in the mask image is present. Algorithm 3 shows the redness quantification.

Algorithm 3 Find redness

```
for $(x, y)$ in $image$ do
    if $mask\ (x, y)\ != \ 0$ then
        if $red > 1.1 * avg(green, blue)$ then
            $red \leftarrow +1$
    $num \leftarrow +1$
end for
$redness = red/num$
```

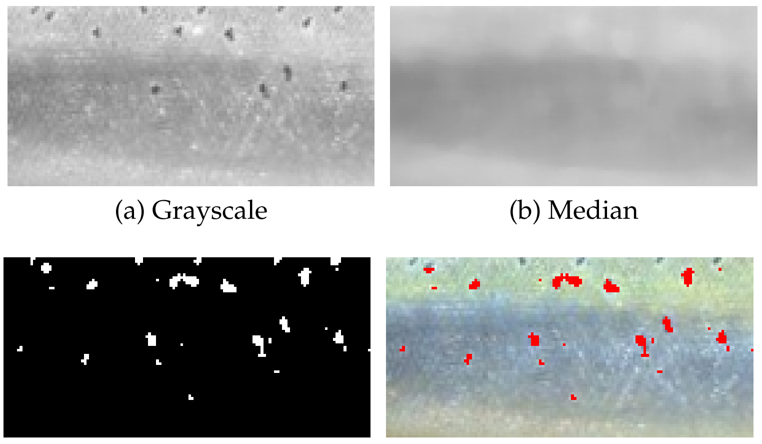
4.2.5 Quantifying spots

In addition to the red color below the lateral line, the color of the spots can be used to separate char from salmon and trout. The char has spots that are brighter than the rest of the body, while salmon and trout have spots that is darker than the rest of the body. One way to quantify this feature is by using median filtering, difference image, and Otsu thresholding to locate the spots, then use the result as a mask to find the color of the spots.

Median filtering. The median filter replaces each pixel's value with the median of the neighboring pixels. The neighborhood can be of any shape (e.g square, circle or cross) and size.

$$y[m, n] = median\{y[i, j], (i, j) \in w\} \tag{4.18}$$

where w represents a neighborhood centered around location (m, n) in the image y.


(a) Grayscale (b) Median

(c) Differenced image (d) Spots

Figure 4.19: The three stages of locating spots in a image of a salmon.[8]

Otsus method Otsu's method (Otsu [1979]) is a method for finding an optimal threshold value assuming the image to be thresholded contains two classes of pixels. It does this by minimizing intra-class variance in two classes of pixels, making their combined spread minimal. The intra-class variance is defined as a weighted sum of variances of the two classes.

$$\sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \tag{4.19}$$

Where ω_n is the probabilities and σ_n^2 the variances of the two classes separated by the threshold t.

Finding spot colors. Differencing the median filtered image with the original image will result in the spots represented by the largest difference values. This is a result of the median filter removing them as they have more non-spot neighbors than spot neighbors due to the neighborhood being larger than the spots. Also, as the median filter uses the "new" values during processing, the spots will be eroded, having their edge removed as the filter processes the image.

The median image will have the spots removed, unless they are considerably larger than the median kernel. When the original image is differenced from

[8]Original image by: Morten Harangen (reprinted with permission).

Algorithm 4 Otsu's method.

```
1. Compute histogram and probabilities of each intensity level
2. Set up initial $\omega_i(0)$ and $\mu_i(0)$
3. Step through all possible thresholds $t = 1...$ maximum intensity
       Update $\omega_i$ and $\mu_i$
       Compute $\sigma_b^2(t)$
4. Desired threshold corresponds to the maximum $\sigma_b^2(t)$
```

the median image, the spots in the original image will have a relatively large difference value, as their values are distant from the "background" color.

Algorithm 5 Quantify spots

```
1. Smooth image using median filtering($kernel\ size = 5 \times 5$).
2. Difference smoothed image with original image.
3. Threshold difference image using Otsu's method.
4. Find spot and background color in original image using thresholded im-
   age as a mask.
```

4.3 Case-based reasoning

Case-based reasoning reuses and adapts solutions from previously experienced problems to solve new problems (Aamodt and Plaza [1994]). In contrast to other AI approaches for reasoning, it relies not only on domain specific knowledge, but also case specific knowledge from earlier experiences. The problems and solutions are stored in data structures called cases, where each case consists of a set of features representing the situation or problem. When a new problem is experienced, the most similar cases from the case-base are used to propose a solution.

The case-base can vary in both size and how cases are created and indexed. For some domains, it can be beneficial to have a large case base which can include a number of similar cases. The cases can have only a slight variance in feature values, where each case represent a very specific situation or problem, or each situation or problem is represented by a number of cases. For other domains it may be better generalizing the cases and having a fewer number of cases with larger ranges for their values.

Building the case-base can be a automated process, or it may be highly dependent on user interaction. There are several ways of indexing the case-base. One way is to use the feature considered most important as the index, only comparing the problem with the cases which have a similar value for the most important feature. The goal of indexing is to make the case-base able to locate the previous cases which are most similar to the problem it is trying to solve.

The features are very domain specific, and can be represented symbolically or numerically. The only requirement is that they are comparable either directly or indirectly. Case-based reasoning can be described as a four-step cycle (Figure 4.20):

1. **Retrieve** relevant cases from the case-base. Relevant cases can be cases that are similar to the target problem, or in other ways useful for solving it.

2. **Reuse** the information in the retrieved cases to propose a solution to the target problem. The information can be used directly, or might need to be adapted to fit the new problem.

3. **Revise** the solution by applying it to the target problem. This can be done by testing it on the problem itself, a simulation of the problem, or by supervision of an human expert.

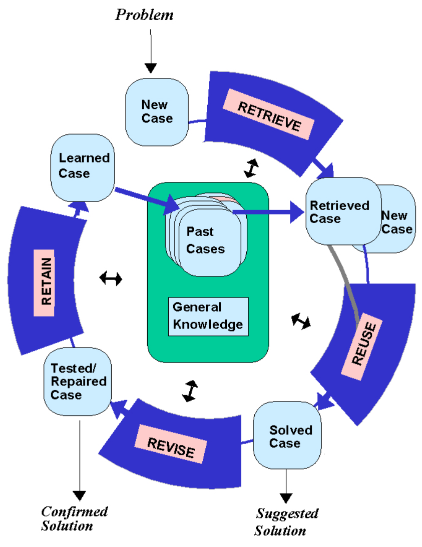


Figure 4.20: The CBR cycle [9].

4. **Retain** the solution to the case-database, either as a new case, or by including it to case representing the proposed solution.

One of the things that separate case-based reasoning and other learning based methods, from more traditional reasoning methods like decision trees, is the fact that it makes and adapts its own rules for how to reason. In decision trees, the developer has the full responsibility to manage how the system will reason. This is done by rules that are final and does not change throughout the run-time of the reasoner. In case-based reasoning, the rules for reasoning are dynamic, and will change over time, making it able to cope with changes in the basis of the problem it reasons about.

Perner [2001] argues for the use of case-based reasoning in image interpretation. Among the disadvantages of more traditional image interpretation techniques is the lack of robustness, accuracy and flexibility, all of which can be overcome with case-based reasoning strategies.

[9]Figure from: Aamodt and Plaza [1994]

4.3.1 Case-based learning

Aha [1991] describes a subset of CBR algorithms referred to as case-based learning (CBL). Aamodt and Plaza [1994] refer to the same subset as instance-based reasoning. What differentiates CBL from more general CBR algorithms is that they do not perform case adaption, limits the features to values [10] and does not necessarily use indexing schemes for the case base. Two of the most well-known CBL-systems are Protos (Bareiss and Porter [1987]) and MBRtalk (Stanfill [1987]). The CBR system proposed by Aasen [2006] can be said to fall into this subset. Only numerical feature values are used, the cases in the case-base does not use indexing schemes and as no case adaption is performed, the solution is used directly.

Case-based learning seems well suited for the classification of fish. The distinguishing features are can be quantified, and represented with numerical values, and as it is a classification problem, the solution needs no adaption as it can only be one of three species.

4.3.2 Implementation

There exist frameworks that enable fast prototyping of CBR solutions such as jCOLIBRI [11] and myCBR [12]. The advantage of using such pre-made systems is that most general CBR related methods are pre-implemented, making it possible to test different approaches quickly. Both the jCOLIBRI framework and myCBR was considered for the CBR part of this thesis.

There were several reasons for implementing a CBR module in Python instead of using a pre-made framework. The main reason is the integration with the feature extraction modules. As these are prototyped in Python, transferring data from them to a stand-alone CBR system like myCBR will require extra work and has to be done every time the test-data or feature extraction methods are changed. As the CBR system proposed by Aasen [2006] is made for image interpretation, reimplementing and adapting it is considered as a better solution than using myCBR or jCOLIBRI.

As stated in Section 2.2, Aasens CBR system is made as a module for the Troll-

[10]Symbolic, binary or numerical values
[11]http://gaia.fdi.ucm.es/research/colibri/jcolibri
[12]http://www.mycbr-project.net/

Eye software by Trollhetta AS. All the source code is therefore in C++. As the feature extraction methods for this thesis have been implemented in Python, the CBR has been reimplemented in Python as well. This is considered a better solution than reimplementing the feature extraction methods to C++, as Aasens CBR system can not be used directly, but need a large amount of modification, as it is not very generalized. Aasens system is used as a skeleton for the implemented CBR, and most methods have been modified.

4.3.3 Features

Features are implemented as classes, with value and weight as variables, and a method for comparing with other features of the same type. As all the features are numerical, a general feature class has been implemented. The feature class consists of: "raw" input value, normalized value, mean, standard deviation, weight and name. The name is used to give a warning if two different features are being compared. Table 4.1 shows an example feature.

Table 4.1: A feature instance

Variable	value	raw value	mean	stdev	weight	name
Value	0.555	0.153	0.145	0.015	1.0	"peduncle"

4.3.4 Cases

Cases are also implemented as classes, consisting of a set of features, the species and the filename for the corresponding color image. The features are organized in an array, the species is added when the case has been classified. There is also an extra variable called solution used for automatic revision during testing, which is found from the file-name of the image used to create the case. It is not to be confused with the species variable which is the ordinary solution for the case. Table 4.2 shows an example case.

Table 4.2: A case instance

Variable	features	species	solution	color image
Value	[F1,F2,F3,F4]	"salmon"	"salmon"	"salmon1.jpg"

4.3.5 Retrieve

As all features are numerical values, the retrieval is done is by using a similarity measure for calculating the numerical difference between the input case and each case in the case base. Before the retrieve stage can be executed, the problem has to be represented as a case. A case consists of a set of extracted features and a solution. A flowchart for the retrieve function can be seen in Figure 4.21.

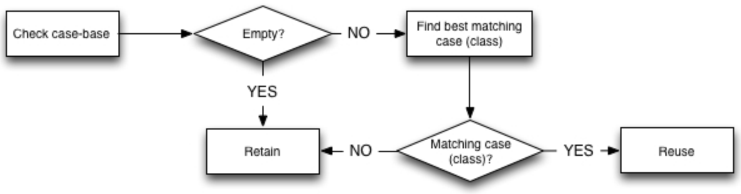


Figure 4.21: Flowchart for the retrieve method for best matching-case and k-nearest neighbor(in parentheses).

4.3.5.1 Filter out noisy feature values

Aasens CBR system does not address problems regarding feature values with values far from the expected range. These "extreme" values can be caused by irregularities in the captured images that are not taken care of by the preprocessing and feature extraction. To reduce the influence of these "extremes" noisy feature values are reduced and the range of all features are normalized. This is done by using standard score and hyperbolic tangent on all features. The standard score normalizes their range, while the hyperbolic tangent removes values that are far from the feature's normal range.

Standard score To equalize the ranges of the different features, standard score is used (Equation 4.20). It indicates the number of standard deviations the feature's value is above or below the mean value for that feature. Standard score is not only to reduce the effect of noisy feature values. If will also ensure that all features has the same influence in the similarity measurement.

$$z = \frac{x - \mu}{\sigma} \tag{4.20}$$

where: x is a raw value, μ is the mean value and σ is the standard deviation.

Hyperbolic tangent The hyperbolic tangent (Equation 4.21) is used to limit feature values to the range -1,1.

$$tanh\ x = \frac{sinh\ x}{cosh\ x} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{4.21}$$

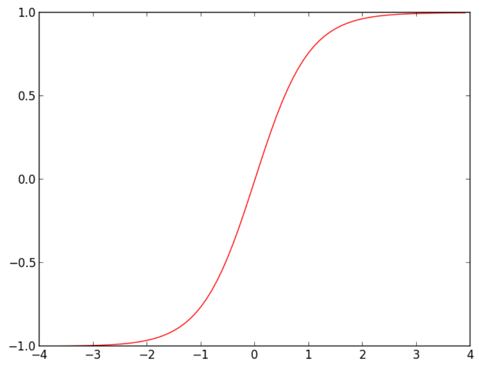


Figure 4.22: A graph showing the hyperbolic tangent

4.3.5.2 Initial matching

Usually, the case matching in CBR is done in two phases. The initial matching process, which selects a number of candidate cases, and then a more elaborate process of selecting the best match. As no abstract of hierarchical indexing is done on the case-base, the difference from the problem and all cases in the database is calculated.

Similarity measures The difference between features are found by a comparison method. As all features are numerical, euclidean difference is used. For one-dimension, this is the absolute value of their numerical difference (Equation 4.22).

$$|a - b| = \sqrt{(a-b)^2} \tag{4.22}$$

4.3.5.3 Select

Two methods for selecting the best matching class have been implemented. The first selects the best matching case, based on similarity measures between features and the feature weights. The second finds the *n* best matching cases and selects the class most frequent among these cases. Both methods use the Algorithm 6 for computing the difference between two cases.

Algorithm 6 Computing difference between two cases. (new and old)

```
1: $Difference = 0$
2: for $i = 0$ to $i = number\ of\ features$ do
3:     $Difference += \sqrt{old[i].value - new[i].value}^2 \cdot old[i].weight$
```

Best matching case As the difference from the problem to all cases in the database is calculated in the initial matching, this method simply propose the case with the lowest difference to the problem as the solution. The best matching case is the selection method used in Aasen [2006].

k-nearest neighbour The k-nearest neighbor algorithm is used for classification of problems based on closest examples with regards to features. k denotes the number of neighbors used for classifying a new problem. If $k = 1$, the algorithm will give the same result as the best-matching case as the nearest neighbor will be the best matching case.

The neighbors are scored based on their difference from the case to classify, this means that the closest neighbors will be weighed higher than neighbors further away. The class most frequent in the k most similar cases is selected as the class of the new problem. A larger k will generally reduce the amount of noise in the selection, but if the value for k is set too high, it will result in classes with few instances in the test-set never being selected as the best match.

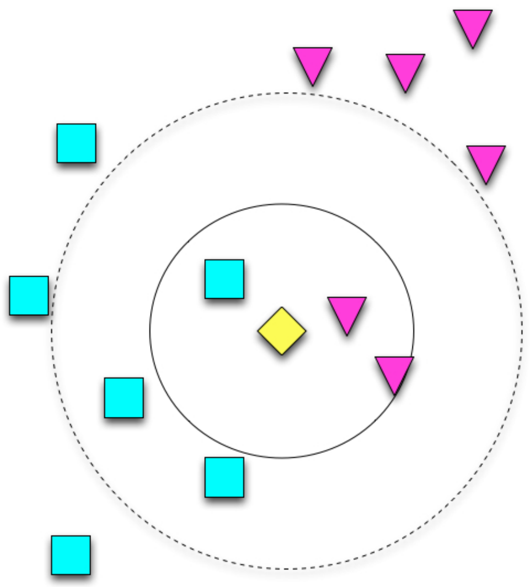


Figure 4.23: k-NN classification of a new case. The new case is indicated by the diamond. If k=3, the new case is classified as class 1 (triangle). If k=5, it is classified as class 2 (square).

4.3.6 Reuse

Aamodt and Plaza [1994] describe two ways of doing case reuse. By copying the solution or by adapting the solution. Copying is done by abstracting away all differences, and using the solution of the retrieved case as a solution to the new case. Adaption is not as trivial and can be done in two ways, by transforming the solution by transformational operators (transformational reuse) or by using the method that constructed the solution (derivational reuse). As only solution copying is used, case adaption will not be explained in more detail.

As stated in Section 4.3.1, no case-adaption is made due to the nature of the problem. The reuse stage is used for deciding if the solution proposed is close enough to the problem to be automatically approved, or if revision is needed. To decide if it is close enough, different measures is used for the two selection methods. For the best-matching case, the weighted difference between the problem case and the solution case is used. For k-nearest neighbor, the score for the proposed class is used. A flowchart of the reuse method can be seen in Figure 4.24.

Algorithm 7 k-NN scoring.

```
1: Forall $case$ in $cases_{sorted}$
2:     if $species = salmon$ then
3:         $salmon \leftarrow \frac{1.0}{(difference+1)}$
4:     else if $species = trout$ then
5:         $trout \leftarrow \frac{1.0}{(difference+1)}$
6:     else if $species = char$ then
7:         $char \leftarrow \frac{1.0}{(difference+1)}$
8:     end if
```

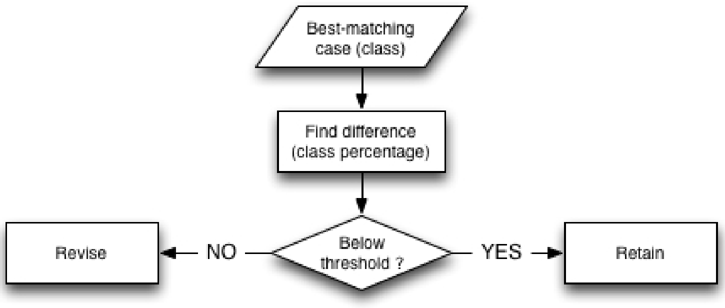

Best-matching case (class)
Find difference (class percentage)
Below threshold ?
NO
Revise
YES
Retain

Figure 4.24: Flowchart for the reuse method for best matching-case and k-nearest neighbor(in parentheses).

The reuse method is a is based on the method proposed by Aasen [2006] with one exception. In Aasens method the feature weights is not used for determining if the case can be automatically reused, meaning that features considered unimportant by the system is given as much weight as features considered important. For the reimplemented method, the same case difference function is the same as the one used for comparing cases in Retrieval (Algorithm 6).

4.3.7 Revise

Revise is the third stage of the CBR-cycle, where the found solution is evaluated. This can be done automatically by applying it to the problem (or a simulation of the problem) or it can be done with the assistance of a human supervisor. The revise step is only triggered if the best matching case is not automatically reused. Both an automatic and a user-interaction based revision method have been implemented.

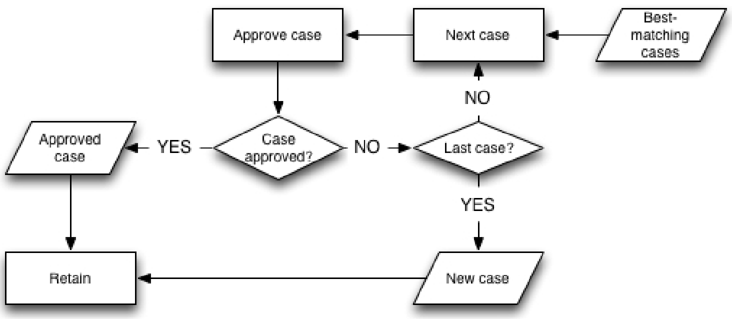


Figure 4.25: Flowchart for the revise method.

The automatic revision method is intended for testing purposes, as it depends on the species being included in the filename. The supervised method is based on simple user interaction. Except for the difference in how they approve a solution they work similarly. What they do is to approve or disapprove a suggested solution from the system, the automatic one compares the suggested solution to the correct answer, found by trimming the filename of the new case. The user interaction version displays the image for the new case along with the suggested solution, and waits for the user to approve or disapprove the solution via the command line. This procedure is repeated for the three best matching cases or until a suggested solution is approved (Figure 4.25).

4.3.8 Retain

Retain is the last stage. In this stage the new case is retained as a case in the case base, in this way it can be used to solve later problems. Another way to retain can be to merge it with an existing case in the case-base, or not to modify the case-base at all if the new case is very close to the best matching case.

The implemented retain method can be seen in Figure 4.26. There are three outcomes of the retain stage: (1) there are no similar cases in the case-base - the new case is added to the case-base, (2) a solution has been found that has been automatically accepted - weights of the closest cases are adjusted, (3) the solution found has been revised - weights of the closest cases are adjusted and the new case is added to the case-base.

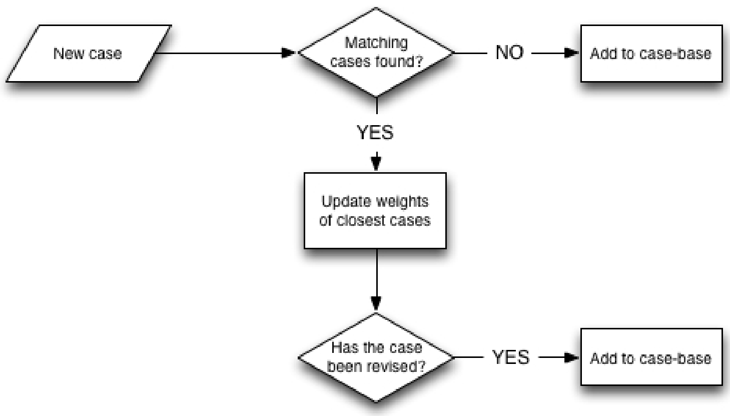


Figure 4.26: Flowchart for the retain method.

4.3.8.1 Weight adjustment

Individual weights are applied to each feature in a case. These weights are updated in the retain phase. The purpose of these weights is to learn the importance of the feature and tolerate irrelevant features (Aha [1991]). Weight adjustment is performed on the three best matching cases regardless of any of them being accepted as a solution. For cases which are of the correct class (the same class as the new case), feature weights are increased according to the distance to feature values of the new case. For cases of the incorrect class the weights are decreased by the same criteria.

The weight adjustment is based on the one from Aasen [2006], with some modifications. In Aasens system, the implemented weights were adjusted based on a threshold. This gives a very static behavior, as two features with very similar differences can have their weights adjusted in opposite ways if one happens to be directly above and one directly below the threshold. The weight adjustment in Aasens system is shown in Algorithm 8.

To achieve a more dynamic behavior, the weight adjustment uses the feature differences directly when modifying the weights. The weights adjustment can be seen in Algorithm 9. The modifier can be any floating point number, and controls the general amount of weight-change. The weights are normalized after the adjustment to make sure that all the cases are compared equally in the retrieve stage.

Algorithm 8 Aasens weight adjustment

```
1: if $correct = true$ then
2:     if $difference < threshold$ then
3:         $weight \leftarrow weight \cdot modifier$
4:     else
5:         $weight \leftarrow \frac{weight}{modifier}$
6:     end if
7: else if $correct = false$ then
8:     if $difference > threshold$ then
9:         $weight \leftarrow \frac{weight}{modifier}$
10:    else
11:        $weight \leftarrow weight \cdot modifier$
12:    end if
13: end if
```

Algorithm 9 New weight adjustment

```
1: if $correct = true$ then
2:     $weight \leftarrow weight \cdot modifier(1 - |difference|)$
3: else if $correct = false$ then
4:     $weight \leftarrow \frac{weight}{modifier(1-|difference|)}$
5: end if
```

4.3.9 Implementing Database Support

To enable monitoring of the case-base, database support has been implemented with sqlite [13]. The case-base is not saved to the database automatically, but methods for writing and reading the database have been implemented. The methods writes all cases and their features to the database. This way, a footprint of the case-base can be viewed when needed. This also provides the possibility of saving the case-base to disk, and retaining it at a later time. Also, cases that have to be revised can be saved in a database, and revised by a human supervisor at any time.

[13]www.sqlite.org

Chapter 5

Testing and results

5.1 Leave-one-out cross-validation

Cross-validation is usually applied to estimate how accurately a predictive model will perform in practice. It involves separating subsets in a larger test set. The model is built with one subset (training set) and validated with another (validation set).

Leave-one-out cross-validation(LOOCV) is one form of cross-validation. As the name implies, one of the instances in a test-set is left out of the set during training. Then the instance left out is used to validate the training. By rotating which instance is left out, the testing will give a reasonable estimation of the models predictive accuracy.

Since all but one instance are used for training, LOOCV is well suited for validating small datasets compared to e.g. 2-fold validation. In 2-fold validation the test-set is split in two equally sized sets and one is used for training and one for validation. With few cases in the test set, it is probable that there will not be enough cases representing each class to produce an accurate model.

The downside with LOOCV is that it is computationally expensive. The model has to be re-trained for each instance, which for case-based reasoning may be an extensive process.

5.2 Goals of testing

As stated in the introduction, the main focus of this thesis has been feature extraction and classification of fish. The preprocessing has been tested, but not to the same extent as the other parts of the solution. Thus, this chapter contains only a short overview of the results of the preprocessing. The testing of the preprocessing stage shows what can be achieved for relatively poor data, to give an indication on how good the data has to be in order for the feature extraction and case-based reasoning to perform properly.

For feature extraction and classification, the goals for testing is to see if the case-based reasoning is a feasible approach to fish classification. The reliability of the CBR is also tested to see if it produces the same results over time. This is done by shuffling the training data and observing if the CBR reasons similarly for a large number of test runs.

5.3 Preprocessing

The preprocessing has not been thoroughly tested as there was a lack of appropriate test data. A few of the best frames from the existing videos were used to see what the preprocessing stage could accomplish.

As the results from Horprasert et al. [2000] algorithm were very poor, they are not included. It seems that the algorithm does not handle the noise present in underwater images very well. The codebook algorithm Kim et al. [2005] on the other hand gave much better results, and was therefore chosen for testing.

The resulting binary images from the background subtraction were cleaned up using morphological closing, before the contours were found. Contours of considerable size were then improved using snakes on a greyscale version of the image. Then each contour was compared to a template contour using Hu's moments to find its degree of resemblance to a fish.

As can be seen in Figure 5.3 the results of the preprocessing are decent, but far from perfect. The fish are recognized, but the low quality of the images does not make the results suitable for testing on the CBR stage.

One challenge using snakes is to find a good external constraint in the form of a starting snake. For testing, the contour found by Suzuki and Abe [1985] is used as this constraint. As the snake will not expand, only retract, the resulting contour will lie inside the external constraint. This can be seen in Figure 5.2e where the tail fin is left out of the contour. The tail fin is disconnected from the rest of the fish during background subtraction, and is not reconnected trough morphological closing. This leads it to be excluded from the initial contour, and therefore its outside the external constraint when the snake is applied.

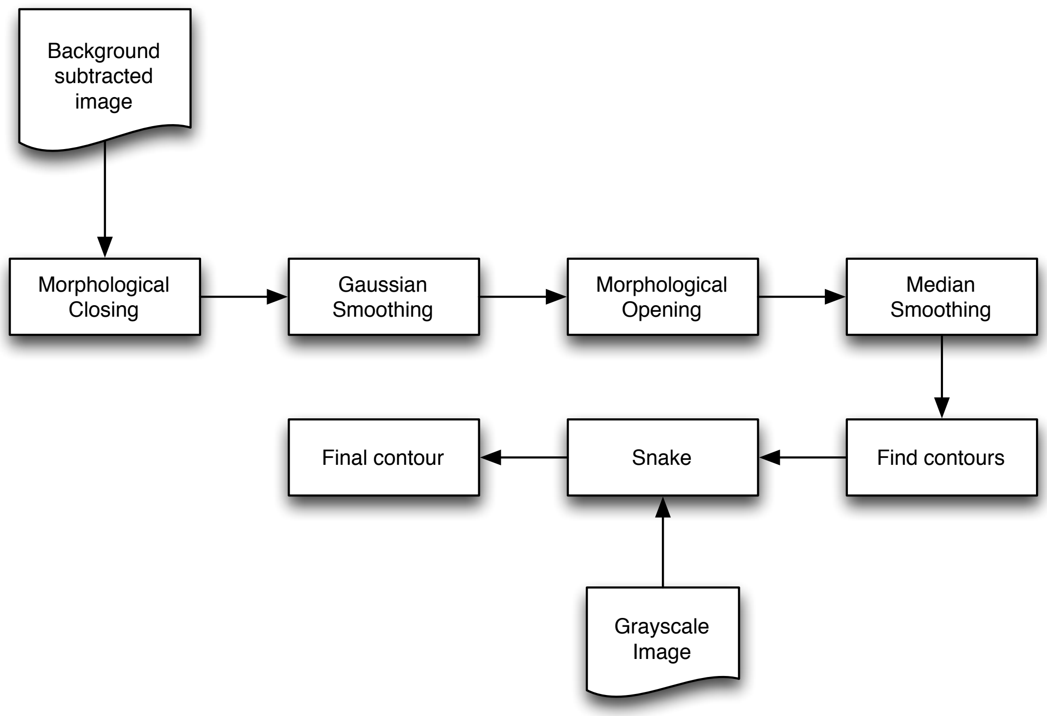
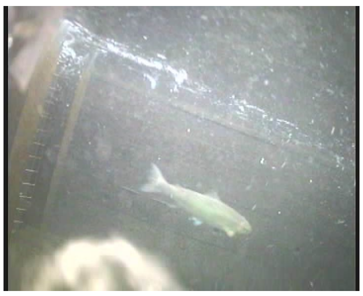
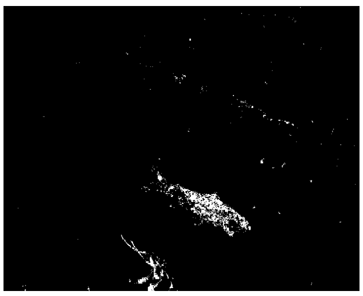
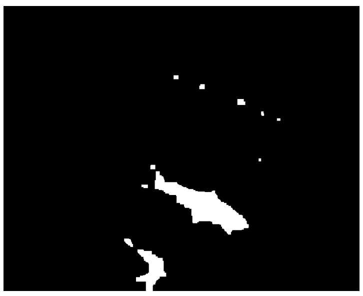
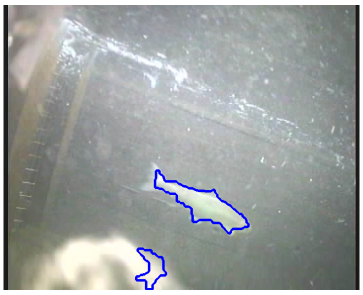


Figure 5.1: The preprocessing procedure.


(a) Original


(b) Subtracted


(c) Cleaned


(d) Contours

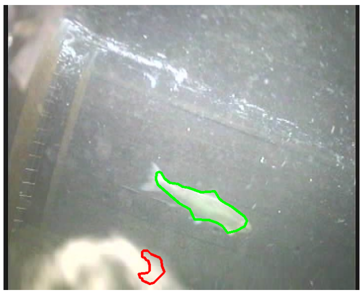
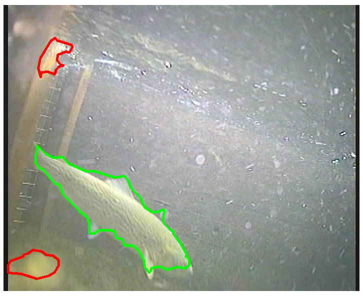
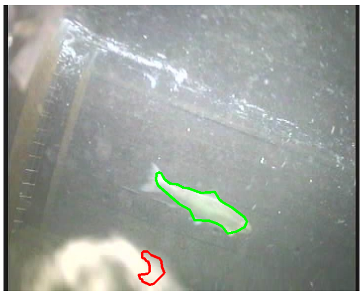

(e) Snaked contours

Figure 5.2: The stages of preprocessing a frame. Green contours indicate fish.


(a) Frame1


(b) Frame2

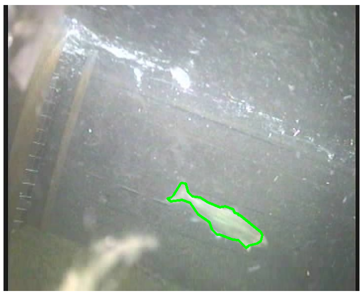

(c) Frame3

Figure 5.3: Three preprocessed frames.

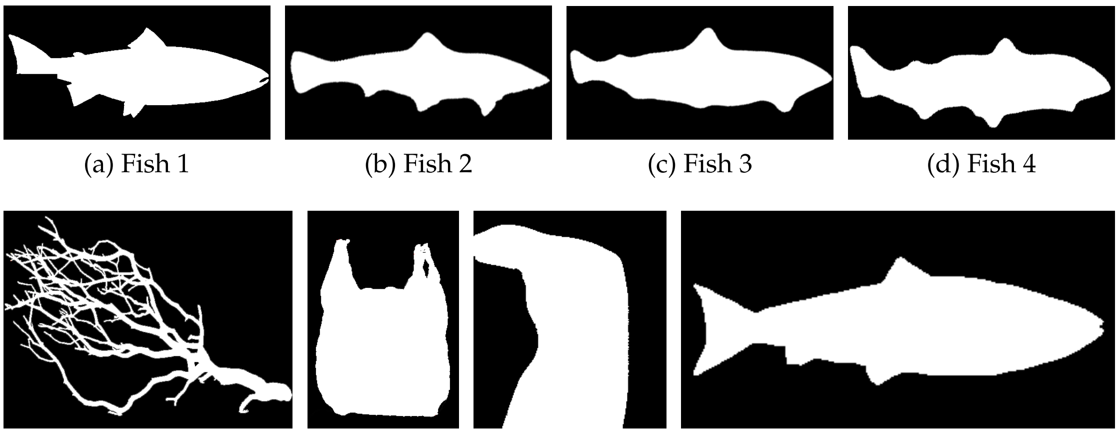
5.3.1 Coarse sorting with Hu moments

The results of the testing of fish-shape recognition using Hu moments can be seen in Table 5.1 and Figure 5.4. As can be observed, the sorting works well for separating fish-like shapes from other objects. The fish shapes are segmented with the proposed preprocessing routine, except for Image which is the template shape with half of its caudal fin removed.

The non fish objects are far from the fish with regards to shape, but the results shown in Figure 5.2 and 5.3 indicate that the methods works also less defined shapes.

Table 5.1: The calculated distances from the shapes to the template shape using Equation. 4.17 .

Image	Fish 1	Fish 2	Fish 3	Fish 4	Branch	Bag	Blob
Diff.	0.0744	0.1514	0.1271	0.1234	0.5132	0.7111	1.6432


(a) Fish 1 (b) Fish 2 (c) Fish 3 (d) Fish 4

(e) Branch (f) Bag (g) Blob (h) Template

Figure 5.4: Binary shapes used for testing comparison using Hu's moments and Equation. 4.17

5.4 Feature extraction

The largest challenge in this thesis has been the lack of good test data, as the few available videos from the existing system was the only data available and of very low quality. This was solved by using regular images found on the internet for testing the case-based module. Most of the images are not taken under water, but comparing the ones that are with those that are not, colors and patterns are equally good in both types of images. The images taken on land are therefore considered to be adequate replacements for underwater images. A total of 28 fish images: 11 salmon, 9 char and 8 trout is used during testing.

For testing, the supervision for the case-based reasoning was made automatic by using the file-names of the cases as guidance. The test script takes a folder containing the test images as input. The images have to be arranged in two folders. One containing the original images as jpg, and one containing the mask images as png. The mask images were drawn manually. The matching files need to have the same file-name and be of the same spatial resolution.

The results of the feature extraction in seen in Tables 5.2, 5.3 and 5.4.

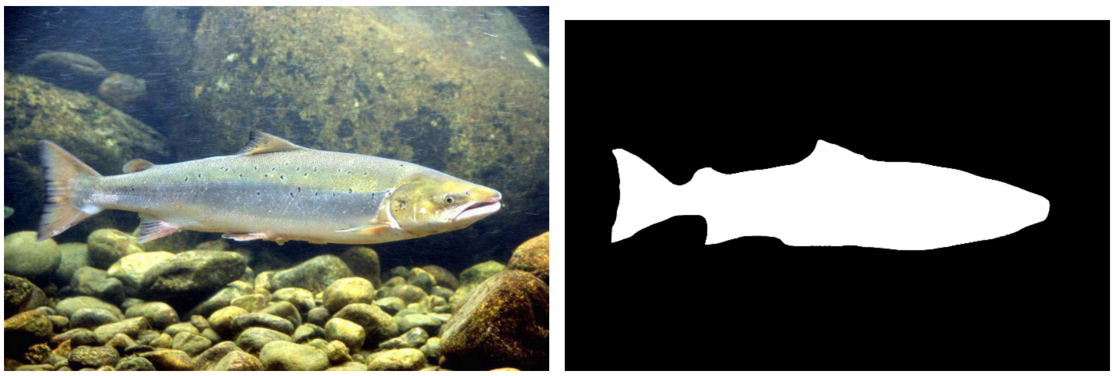

(a) Color image (b) Mask image

Figure 5.5: A test image and its corresponding mask image.

Table 5.2: The average values of the five features in salmon.

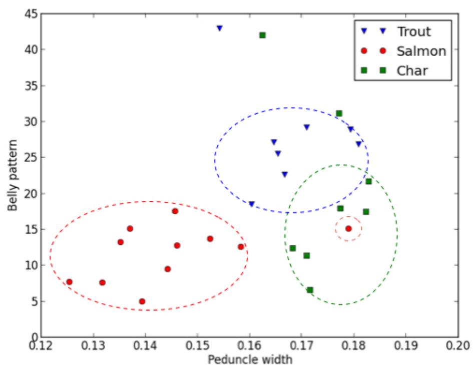
	Average	Std. dev	Min	Max	Range
Peduncle	0.145	0.014	0.125	0.179	0.053
Tail fin	-0.020	0.051	-0.168	0.000	0.168
Belly pattern	11.785	3.850	4.981	17.536	12.554
Redness	0.020	0.012	0.010	0.052	0.041
Spots	0.812	0.097	0.645	0.990	0.345

Table 5.3: The average values of the five features in trout.

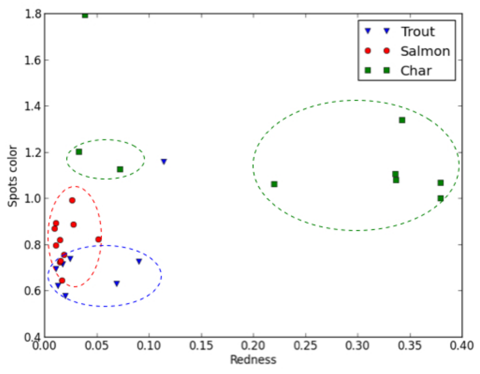
	Average	Std. dev	Min	Max	Range
Peduncle	0.167	0.009	0.154	0.180	0.027
Tail fin	-0.006	0.037	-0.047	0.068	0.114
Belly pattern	27.088	7.103	18.441	42.920	24.479
Redness	0.045	0.040	0.011	0.113	0.102
Spots	0.733	0.182	0.576	1.159	0.582

Table 5.4: The average values of the five features in char.

	Average	Std. dev	Min	Max	Range
Peduncle	0.177	0.010	0.163	0.197	0.034
Tail fin	-0.006	0.013	-0.029	0.019	0.049
Belly pattern	22.063	12.350	6.597	42.020	35.42
Redness	0.0237	0.150	0.033	0.380	0.346
Spots	1.198	0.244	1.002	1.795	0.793



(a) Belly pattern / Peduncle width



(b) Spots / Redness

Figure 5.6: Clustering for some of the feature values.

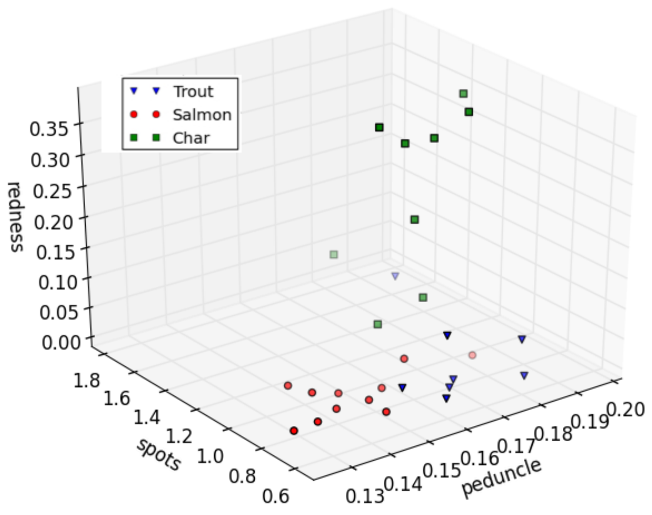


Figure 5.7: 3D plot showing clustering for three of the features.

5.4.1 Peduncle width

The peduncle width extraction has been tested on the same test-set as the CBR. The results can be seen in Table 5.5 and Figure 5.9. As expected, salmon has the thinnest average peduncle width 16% thinner than the average Trout, and 22% thinner than the char. There is especially one value that is significantly larger than the average, 0.179. The fish with these values can be seen in Figure 5.8a. As can be seen in the image, the fish is lying in a curved posture, leading the peduncle to be measured as thicker than it actually is. This posture is unnatural, and very unlikely to be present in underwater images, as the fish moves its tail fin from side to side when swimming, not up and down. The problem with the peduncle being measured incorrectly can occur in other situations, as when the fish is moving towards or away from the camera. This is not likely to occur in the very controlled environment in which the fish are to be captured, and is also difficult to avoid using only one camera, the problem is not addressed in this thesis.

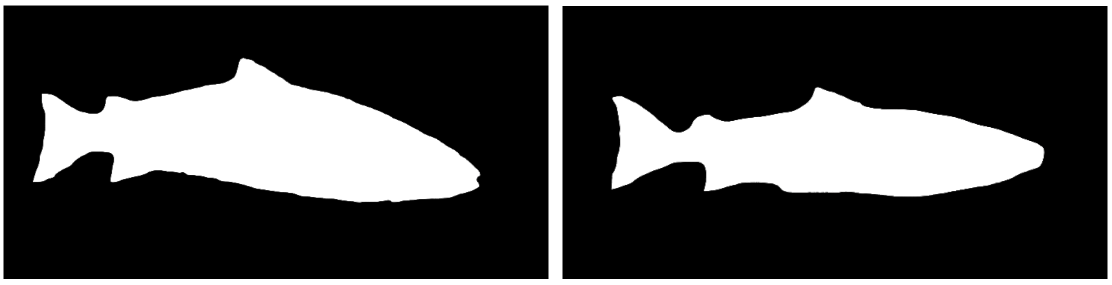

(a) Curved (b) Normal

Figure 5.8: 5.8a shows a salmon in a curved posure, while 5.8b shows a salmon in "normal" posure.

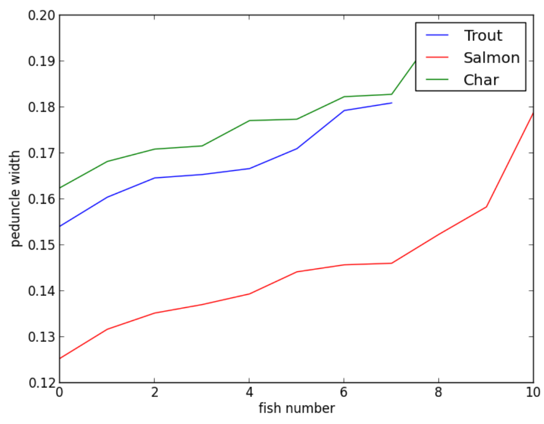


Figure 5.9: Extracted peduncle values

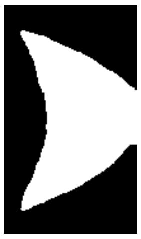
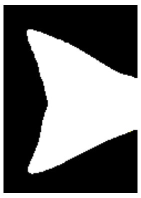
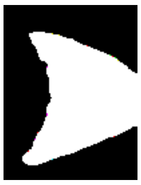
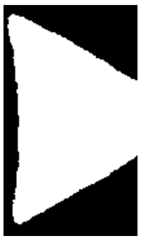
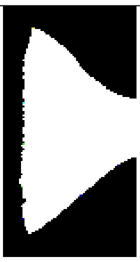
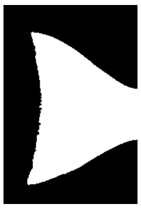
Table 5.5: Extracted peduncle values

Species	Mean	Std.Dev.
Salmon	0.1449	0.0146
Trout	0.1678	0.009
Char	0.1766	0.0101

5.4.2 Curvature of the caudal fin

Of the features extracted, the caudal fin is the least reliable, while it is possible to detect some trends in the data, it is questionable if it is consistent enough to use for the CBR. The results from the extraction can be seen in Table 5.6 and Figure 5.10. There are a few "extreme" values that seemingly disturb the measuring, and by removing these, the data becomes somewhat more consistent. Salmon has an average curve that is twice the size of the trout but only 10% larger than char. This indicates that the tail fin curve can be used to separate trout from salmon and char. On the other hand, the standard deviation is very large, indicating that there is a large amount of variation. In the images used for testing, the tail fin is the feature that is most likely to be obscured as the fish is lying on a rock or in grass. Table 5.7 show extracted values for some fish not in the training set, but with more normal tailfins.

Table 5.7: Some caudal fins and their extracted values.

Img	Salmon 1	Salmon 2	Salmon 3	Trout 1	Trout 2	Char
x^2	-4.60e-3	-4.40e-3	-8.13e-3	-1.07e-3	0.97e-3	-1.67e-3
x	0.62	0.56	0.70	0.19	-0.13	0.29
c	-0.93	-1.66	4.93	-0.96	6.25	4.26
						

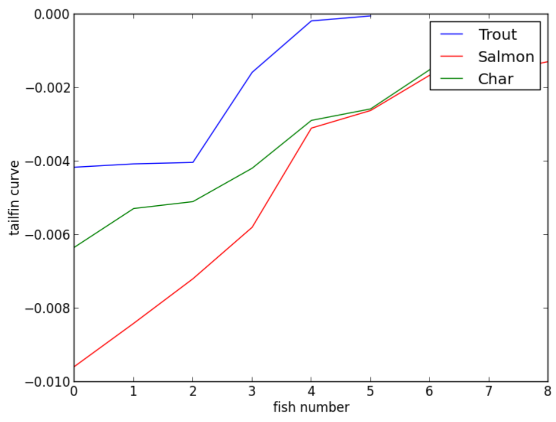


Figure 5.10: Extracted caudal fin values

Table 5.6: Extracted caudal fin values

Species	Mean	Std. Dev.
Salmon	-0.0045	0.0032
Trout	-0.002	0.0020
Char	-0.0039	0.0017

5.4.3 Belly pattern

As can be seen in Table 5.8 and Figure 5.11, the extraction of the belly pattern as standard deviation works well. The salmon is separable from trout with a simple threshold value, and the difference between the two classes is 33%, this indicates, that at least for this test set, the belly pattern is a useful feature to separate salmon and trout. Char has an average value significantly higher than the salmon (21.42 vs 11.90), but the standard deviation is very high (9.83) so difference between the two species in general is too low to conclude if this feature is good for distinguishing.

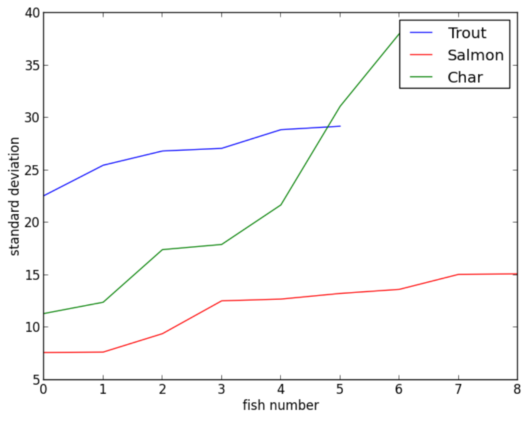


Figure 5.11: Extracted belly pattern values

Table 5.8: Extracted belly pattern values

Species	Mean	Std. Dev.
Salmon	11.90	2.93
Trout	26.67	2.43
Char	21.42	9.83

5.4.4 Redness

The extracted values for redness are shown in Table 5.9 and Figure 5.12. From the results it appears to be a good feature of separating trout and char. There are only three char with less than 20% red pixels, and only three trout with more than 5% red pixels. All salmon have less than 5% red pixels. So more than 20% red pixels will be a very good indication of a char, less that 5% is a good indication of a salmon or trout, while between 5% and 20% red pixels will leave a degree of uncertainty.

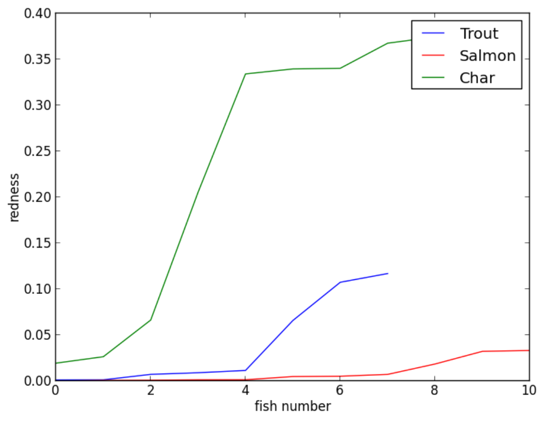


Figure 5.12: Extracted redness values

Table 5.9: Extracted redness values

Species	Mean	Std. Dev.
Salmon	0.0096	0.0125
Trout	0.0400	0.0492
Char	0.2304	0.1532

5.4.5 Spots

As explained in Section 4.2.5, the spots are quantified as the ratio between spot color and the color of the rest of the fish. From Table 5.10 and Figure 5.13 it can be observed that the extracted value can separate the char from the trout and salmon. Except for one trout, the ratio for all trout and salmon is below 1.0,the ratio for all char is above 1.0. This indicates that this is a good feature for distinguishing char from the two other fish. Also it shows that the feature extraction works well for the test set.

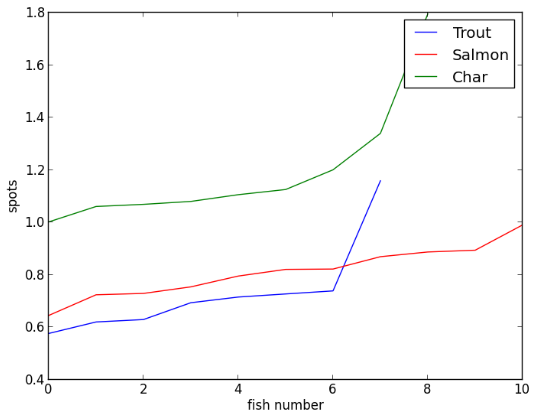


Figure 5.13: Extracted spots values

Table 5.10: Extracted spots values

Species	Mean	Std. Dev.
Salmon	0.8122	0.0968
Trout	0.7328	0.1816
Char	1.1978	0.244

5.5 Classification

As stated in Section 5.4, fish images found on the internet have been used for testing both the feature extraction and the case-based reasoning. The prototype CBR use the feature extraction methods to extract features and make cases from these images.

The case base is trained trough a number of training iterations. For each iteration all cases, except the one that should be left out for validation, is retrieved by the CBR. The cases outside the threshold for automatic reuse are revised using the automatic revise method described in Section 4.3.7. A flowchart of the training process can be seen in Figure 5.14.

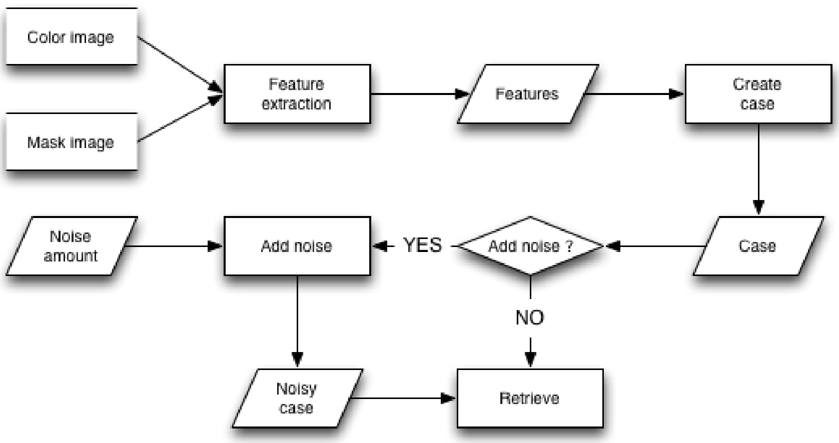


Figure 5.14: A flowchart for the training process.

5.5.1 Training noise

To increase the number of training cases, noise was added to the training cases. By adding a random amount of $0 - 10\%$ noise, with a probability of 10%, the amount of training cases was increased drastically. The noise simulates a larger amount of training data, and is therefore considered to give a more realistic indication of how the classification will perform in real-life. The noise affects the best-matching case method negatively, and the k-nearest neighbor algorithm positively (see Table 5.11). This indicate that the k-nearest neighbor algorithm works better for large dataset, and is more robust to noise. If the random amounts is large enough, it will make some features which have normal values for species move out of the natural range. This will introduce some noisy cases into the training set which are better handled by the nature of the k-nearest neighbor. Training noise is applied for most of the tests below (if stated in the caption).

Table 5.11: Table showing the effects of training noise.

Training noise	Best-matching case	k-nearest neighbor	Avg. num. cases
0%	0.854	0.802	26
10%	0.792	0.854	80

5.5.2 Influence of parameters

Various values have been tested for the parameters in the CBR. This is to give an indication on how they influence the behavior of the reasoning. The parameters tested is the weight adjustment modifier, threshold for automatic reuse and number of nearest neighbors.

5.5.2.1 Weight adjustment modifier

The modifier value used in the weight adjustment influences the rate of learning. The higher the value, the more the weights are adjusted and the faster the reasoner learns. Figure 5.15 shows the results with different values for the weight modifier. As can be observed from the figure, the system is more robust with a lower learning rate. If the weight modification is too rapid, the first cases will be of much higher significance than later cases, as the weight adjustment is not linear. Therefore, the higher the modifier, the less cases the reasoner uses for learning.

5.5.2.2 Reuse threshold

The reuse threshold value for deciding when a case should be automatically classified, and when it should be revised. Automatically classified cases are not added to the case base. The value will therefore control the number of cases in the case-base, as can be seen in Table 5.12, the table shows the total number of

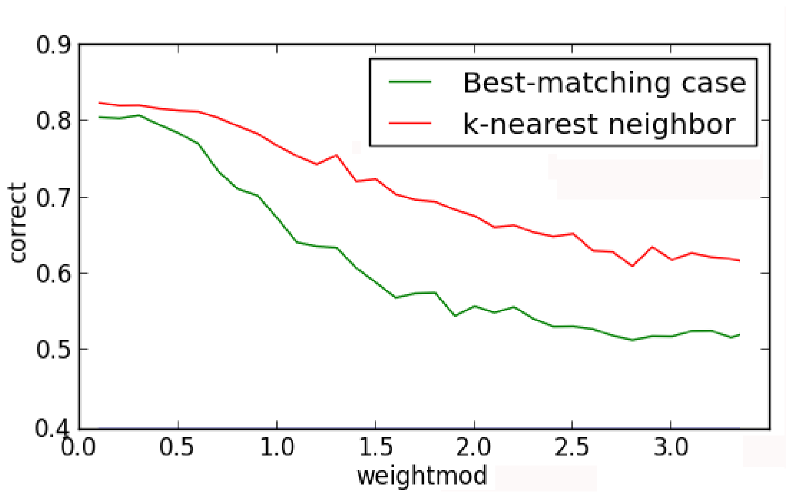


Figure 5.15: Plot of results with various weight modifier values (100 runs, number of neighbors = 4, reuse threshold = 0.2, training noise = 0.1).

Table 5.12: Average number of cases in the case-base for various reuse threshold values. (50 runs, weight modifier = 0.2)

Reuse threshold	Average number of cases
0.050	25.6
0.075	24.5
0.100	20.9
0.150	14.3

cases in the case-base after five iterations of training. Low values will result in a large amount of cases in the case-base, while a high value will result in few cases in the case-base. The results for various threshold values can be seen in Figure 5.16.

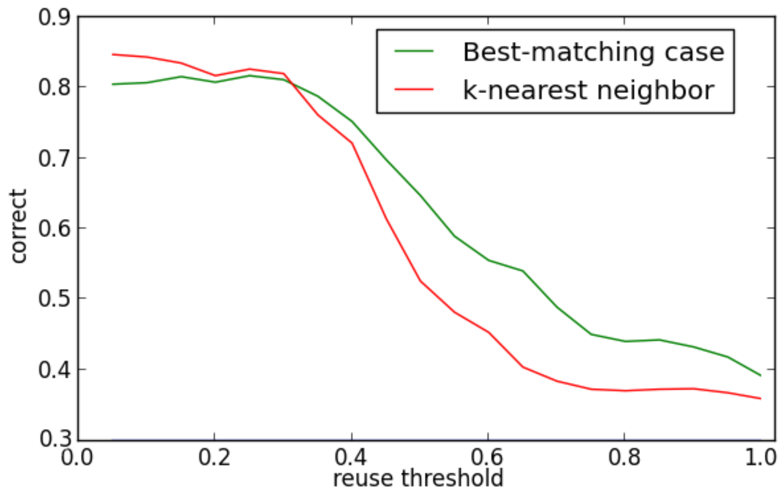


Figure 5.16: Plot of results with various reuse threshold values (50 runs, number of neighbors = 4, weight modifier = 0.2, training noise = 0.1).

5.5.2.3 Number of nearest neighbors

The number of neighbors is used for the k-nearest neighbors (kNN) selection of the retrieve stage. As can be seen in Figure 5.17, the parameter has a good amount of influence on the precision of the classification. The figure show the best-matching case despite the fact that it is not affected by the parameter. It is merely show as an indication of the general performance of the trained case-base, as it is trained differently for every run as the cases are shuffled.

The two methods perform equally for one and two neighbors, this is no surprise as the nearest neighbor will be chosen as a solution for both methods in for these values of k. For three neighbors, the kNN starts to perform better, this is because it is able to correctly classify species where the very nearest neighbor is not of the correct class, but the majority of the neighbors are. The performance of the kNN starts to drop at around seven neighbors. This is likely caused by the correct class not being represented by enough cases in the case base, and the amount of a wrong class getting so high that it equalizes the difference scoring, a parameter for the kNN can be introduced to adjust the scoring, but this has not been done, as selecting a reasonable number of neighbors yields good results.

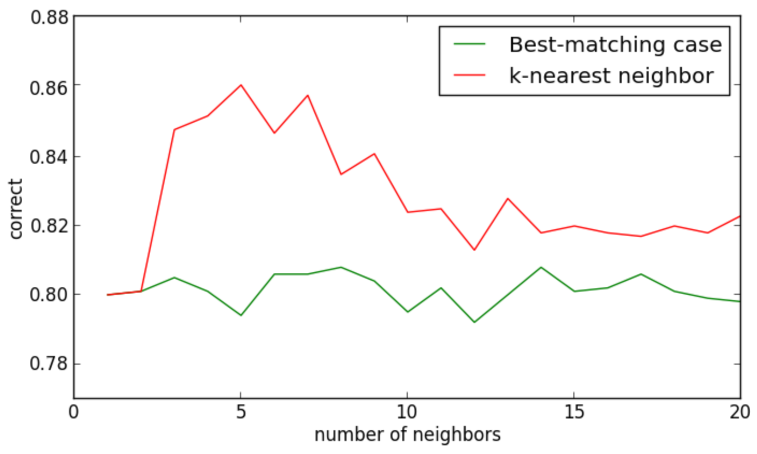


Figure 5.17: Plot of results with various number of nearest neighbors (50 runs, reuse threshold = 0.05, weight modifier = 0.2, training noise = 0.1).

5.5.3 Comparing with a decision tree

The case-based reasoning system has been tested on the same data as the feature extraction. The caudal fin is left out as the extracted values are very ambiguous. A simple decision tree is used as reference for the testing. The implemented decision tree can be seen in Figure 5.18.

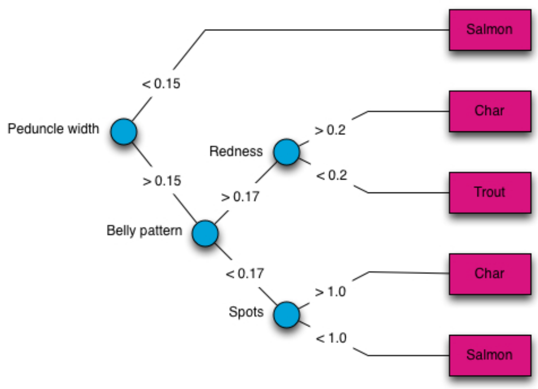


Figure 5.18: Decision tree used for comparison.

5.5.3.1 First comparison

The results from a test run with relatively optimal parameters can be seen in Table 5.13. The results show the amount of correct classification for each image in the test-set. For salmon and trout, the decision tree performs best, classifying all images correctly. For char, the both CBR methods perform 10% better than the decision tree.

5.5.3.2 Second comparison

One of the advantages for the CBR in image interpretation is it's ability to handle noise (Perner [2001]). The CBR and the decision tree has therefore been tested with different amounts of noise. The amount of noise consist of two factors, the size of the noise in accordance to its feature and the probability of the noise being present.

The noise is introduced gradually; for each noise level a random amount of noise between $0 - 100\%$ is added with a probability increasing by 2,5% for each level, this means that all probabilities from $0 - 50\%$ have been covered in 2.5% intervals. The noise is applied to the features before the reasoning, so all three methods have exactly the same noisy features to reason with.

For the case-based reasoning, the case order is randomized for this first test. The parameters are set to: reuse threshold = 0.2, weight modificator = 0.2, number of nearest neighbors = 4. As can be seen from Figure 5.19, the decision tree starts out with $\sim$ 92% correct classification while the k-nearest neighbor and best-matching case starts with $\sim$ 82.5% and $\sim$ 80% respectively. This shows that with absolutely no noise present, the simple decision tree performs better than the CBR.

As the noise level rises, the robustness of the CBR regarding noise is starting to show. At 20% noise probability, the decision tree accuracy drops from $\sim$ 92% to $\sim$ 80%. while the k-nearest neighbor and best-matching case only drop a few percent to $\sim$ 81% and $\sim$ 78% respectively. This means that with a 20% probability for random amounts of noise in a feature, the k-nearest neighbor classification is performing as well as the decision tree. At 40% noise probability, both CBR methods are still getting $\sim$ 75% correct classification while the decision tree drops down to $< 65\%$.

Table 5.13: Results for the 28 test images. Parameter values: reuse threshold = 0.05, weight modifier = 0.2, nearest neighbors = 5, training noise amount = 0.1, training noise probability = 0.1

Image	Best-matching case	k-nearest neighbor	Decision tree
Trout1	48%	100%	100%
Trout2	0%	8%	100%
Trout3	100%	100%	100%
Trout4	0 %	0%	100%
Trout5	100%	100%	100%
Trout6	100%	100%	100%
Trout7	100%	100%	100%
Trout8	100%	100%	100%
Trout total	**68.5%**	**76%**	**100%**
Salmon1	100%	100%	100%
Salmon2	98%	100%	100%
Salmon3	98%	100%	100%
Salmon4	54%	100%	100%
Salmon5	100%	100%	100%
Salmon6	0 %	16%	100%
Salmon7	86%	100%	100%
Salmon8	100%	100%	100%
Salmon9	100%	100%	100%
Salmon10	80%	98%	100%
Salmon11	100%	100%	100%
Salmon total	**83.3%**	**92.2%**	**100%**
Char1	100%	100%	100%
Char2	100%	100%	100%
Char3	84%	86%	100%
Char4	100%	90%	0%
Char5	0%	12%	0%
Char6	100%	100%	100%
Char7	100%	100%	100%
Char8	100%	100%	100%
Char9	100%	100 %	100%
Char total	**87.1%**	**87.6%**	**77.1%**
Total	**80.3%**	**86.1%**	**92.9%**

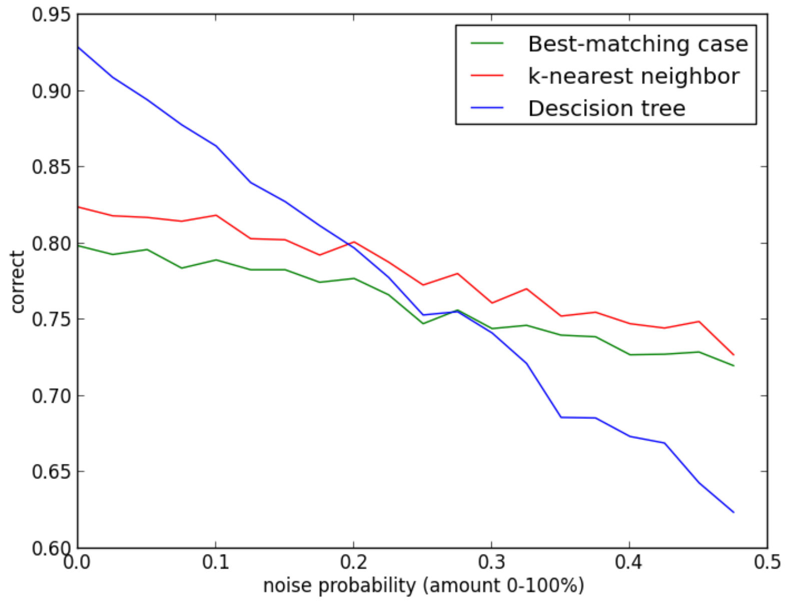


Figure 5.19: Results of the second comparison (100 runs, number of neighbors = 4, weight modifier = 0.2, reuse threshold = 0.2, training noise = 0.1)

5.5.3.3 Third comparison

The previous test runs force the system to classify each test image regardless of how secure the classification is. For this test run, threshold values are used for classification using the k-nearest neighbor algorithm. The threshold indicates how many percent the classification has to score to be classified. The images below this value are arranged in two categories, correct and unclassified and incorrect and unclassified. All other parameters is identical to the first test run. The results of this run can be seen in Table 5.14.

With the threshold value set to 0.7, the number of incorrectly classified cases is 10.4%, this is $\sim$ 8% less than when classification is forced. The total amount of unclassified cases is 14.3%. At 0.75, 8.8% of the images are incorrectly classified, but the amount of unclassified cases is 20.6%, meaning only 4/5 images are classified by the reasoner. At 0.8, only 4% of the cases are classified incorrectly, but only half (44.1%) of the images are classified.

Table 5.14: Results for third comparison.(100 runs, number of neighbors = 4, weight modifier = 0.2, reuse threshold = 0.2, training noise = 0.1)

	Unclassified		Classified	
Threshold	Correct	Incorrect	Correct	Incorrect
0.7	6.4%	7.9%	75.3%	10.4%
0.75	10.7%	9.9%	71.4%	8.8%
0.8	30.1%	14.0%	51.9%	4.0%

5.6 Processing time

Neither the feature extraction nor the case-based reasoning have been implemented with consideration to processing time. Regardless, a few notes on the performance will me made. Extracting features for the 28 test cases takes approximately 16 seconds all together, this gives an feature extraction time of 0.6 seconds for each case. The full testing routine takes six minutes for 100 runs. One run involves five training iterations per case in the test-set, or 140 training iterations all together. So the retrieval of 14000 cases takes six minutes, or 25 milliseconds per case.

For the testing, a MacBook Pro notebook computer has been used. The computer has the following specifications:

- CPU: 2,66 GHz Intel Core 2 Duo
- Memory: 8 GB 1333 MHz DDR3
- Hard drive: solid state drive

Chapter 6

Discussion

The goal for this thesis has been to study the combination of image processing methods and CBR for classifying fish in digital images. This involves preprocessing, feature extraction and reasoning. The main focus has been on the feature extraction and reasoning parts. The study has resulted in a experimental prototype case-based reasoner, able to classify fish with reasonably high accuracy. The prototype does not involve the preprocessing stage, but segmentation methods have been tested.

6.1 Preprocessing and segmentation

Due to the lack of appropriate test data and limited time, preprocessing and segmentation has not been integrated in the prototype. The methods have been tested separately, resulting in an indication as to what might work for underwater footage. Of the two background subtraction techniques tested, only the Codebook algorithm appears to work well under water. The results from this technique show promising results, even for the poor quality video used for testing.

Using the Hu set of variant moments for separating fish from other objects also shows good results. By using it to coarsely sort the segments of the image, processing time can be saved as only the segments containing fish have to be processed by the reasoner.

6.2 Feature extraction

As stated in Section 5.4, the results from preprocessing and segmentation have not been used for the feature extraction. The combination of limited amount of time and low quality of the available video was the reason for this. The testing data used is of much better quality, as this was considered necessary in order to be able to properly test the extraction methods. The gap between the results will have to be lessened in order for the preprocessing to be integrated into the case-based reasoning in order to create a fully automatic solution.

The feature extraction methods show very good results. The exception is the caudal fin extraction which does not give unambiguous values, but the nature of the test images is likely to be the cause. The method itself gives a good estimation of the curve, but as most of the images are taken on land with unnatural postures for the fish the caudal fin is often obscured.

The remaining four features give good values with significant clustering. This makes it possible to separate the species with more than 90% accuracy using only threshold values as shown in the tests with the decision tree. The extraction methods have been tested on still images only, not live video. But with good quality capturing equipment it should be possible to achieve similar resolution and clarity from video. Also, most of the images used for testing not taken underwater. The main problem with underwater footage is the loss of colors, but by placing the camera at low depths this can be avoided.

6.3 Case-based reasoner

Case-based reasoning has been applied to the extracted features to successfully classify the species. The CBR does not perform as well as a decision tree using the same feature values. The CBR is on the other hand much more robust to noise, and at 20% noise probability the two reasoning methods perform equally well. The implemented CBR system is not very advanced, and more advanced methods for the reasoner might increase the accuracy while retaining the robustness to noise.

The parameters of the CBR have been tested thoroughly, but not all combinations of them as testing takes an extensive amount of time. Untested combinations of the values might still improve the performance of the system, but based on the tests done this unlikely to improve the results radically.

During testing, all images are classified, regardless of how good the classification is. For the k-nearest neighbor method, a percentage is given to each species, the higher it is, the more likely that the classification is correct. For the tests where images are not classified if the certainty is below a given threshold, the amount of incorrectly classified cases decrease. On the other hand, the amount of images which are correctly classified also decrease, as some of these will also be below the threshold. The threshold should therefore be chosen based on correct classification versus the amount of supervision. A higher amount of automatically classified images will result in a higher amount of wrongly classified images.

Overall, case-based reasoning seems to be an interesting approach to the problem of fish classification. The results of this study are not directly comparable to all the previous approaches discussed in Section 2.1. Rova et al. [2007] address the most similar problem, where two species with similar shapes are classified with a success rate of 90%. As the case-based reasoning achieves at best 86% during testing, the results must be said to be approximately equal.

Chapter 7

Future work

There are many possible improvements and extensions that can be made to the solution proposed in this thesis. The preprocessing stage has large room for improvement, and need to be properly tested on appropriate data. Other preprocessing and segmentation methods can also be studied. The goal here should be to produce results that are similar to the testing data used for the feature extraction.

One part of the preprocessing which has not been addressed at all is how to select the best image frame to use, as the fish is likely to be captured on video. A way to do this, is to use the proposed preprocessing on all frames, and then use the frame which has the object with most resemblance to a fish in it. It can also be possible to extract features from several frames, and calculate mean values for the feature values from each frame.

As goal of this thesis has not been to create a fully functional system for fish classification, the case-based reasoning is only a prototype. A natural continuation will be to further test and improve it, and to implement it in a programming language with better processing capabilities. No user interface exist, except for in the revision method, where a image of the fish to classify is shown, along with input and output provided from the command line.

The redness feature depends highly on color information in the image. This puts some limitations on the capturing equipment, black and white cameras can not be used and neither can infrared cameras. This can be solved with controlling the capture environment by adding strobe lights, or by finding alternative features to use for distinguishing the char.

The case-based reasoning uses only numeric feature values; replacing them with fuzzy-sets is another extension that can be interesting. Fuzzy sets allows elements to have degrees of membership to classes, e.g. the peduncle can be *very thin*, *thin*, *normal*, *thick* or *very thick*.

Lack of test-data has been a large challenge in this study. The creation of a proper data-set for fish is something that would be very useful for comparing fish-classification methods. Naturally, most classification methods will be specialized for some species, and the preliminaries for the images might be different. But a online repository for fish data would make the acquiring of testing data far easier.

Bibliography

Aamodt, A. and Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaces. In *AI Communications*, volume 7:1, pages 39–59. IOS Press.

Aasen, T. A. (2006). Case based surveillance system. Master's thesis, Norwegian University of Science and Technology, Department of Computer and Information Science.

Aha, D. W. (1991). Case-based learning algoritms. In *Proceedings of the 1991 DARPA Case-Based Reasoning Workshop*.

Akgül, C. B. (2003). Automatic fish classification from underwater images. Technical report, Bogazici University, Istanbul.

Bareiss, R. and Porter, B. (1987). Protos - an exemplar-based learning apprentice. In *Proc. 4th International Workshop on Machine Learning*, pages 12–23.

Belongie, S., Malik, J., and Puzicha, J. (2002). Shape matching and object recognition using shape contexts. In *IEEE Trans. PAMI*, volume 24(4), pages 509–522.

Charles, C. K. (1992). *An Introduction to Wavelets*. Academic Press.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. In *Machine Learning*, volume 20:3, pages 273–297.

Fauske, E., Eliassen, L. M., and Bakken, R. H. (2009). A comparison of learning based background subtraction techniques implemented in cuda. In *Proceedings of the First Norwegian Artificial Intelligence Symposium*, pages 181–192.

Gonzalez, R. C. and Woods, R. E. (2008). *Digital Image Processing*. Pearson International, 3rd edition.

Grimnes, M. and Aamodt, A. (1996). A two layer case-based reasoning architecture for medical image understanding. *Lecture Notes in Computer Science*, 1168(Proceedings of the Third European Workshop on Advances in Case-Based Reasoning):164–178.

Horprasert, T., Harwood, D., and Davis, L. (2000). A robust background subtraction and shadow detection. In *Proc. ACCV*.

Hu, M. K. (1962). Visual pattern recognition by moment invariants. In *IRE Trans. Info. Theory*, volume IT-8, pages 179–187.

Kass, M., Witkin, A., and Terzopoulos, D. (1988). Snakes: Active contour models. In *International Journal of Computer Vision*, volume 1(4), pages 321–331.

Kim, K., Chalidabhongse, T., Harwood, D., and Davis, L. (2005). Real-time foreground background segmentation using codebook model. In *Science Direct, Real Time Imaging*, volume 11, pages 172–185.

Lee, D., Redd, S., Schoenberger, R., Xu, X., and Zhan, P. (2003). An automated fish species classification and migration monitoring system. In *Proceedings of The 29th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, pages 1080–1085.

Micharelli, A., Neri, A., and Sansonetti, G. (2000). *A case-based approach to image recognition*, pages 443–454. Springer Verlag.

Otsu, N. (1979). A threshold selection methon from gray-level histograms. In *IEEE Trans. Sys., Man., Cyber*, volume 9 (1), pages 62–66.

Penthon, P. (2005). *Aschehougs Store Fiskebok*. Aschehoug.

Perner, P. (2001). Why case-based reasoning is attractive for image interpretation. *CaseBased Reasoning Research and Development Proceedings of the Fourth International Conference on CaseBased Reasoning ICCBR01*, pages 27–43.

Perner, P. and Bühring, A. (2004). *Case-Based Object Recognition (Lecture Notes in artificial intelligence 3155)*, pages 375–388. Springer Verlag.

Perner, P., Holt, A., and Richter, M. (2006). Image processing in case-based reasoning. In *The Knowledge Engineering Review*, volume 20:3, pages 311–314.

P.F.Felzenszwalb and Huttenlocher, D. (2005). Pictorial structures for object recognition. In *IJCV*, volume 61(1), pages 55–79.

Rao, C. R. (1973). *Linear Statistical Inference and its Applications*, page 220.

Rodrigues, M. T. A., Pádua, F. L. C., Gomes, R. M., and Soares, G. E. (2010). Automatic fish species classification based on robust feature extraction techniques and artificial immune systems. In *Bio-Inspired Computing: Theories and Applications (BIC-TA), 2010 IEEE Fifth International Conference on*, pages 1518 –1525.

Rova, A., Mori, G., and Dill, L. M. (2007). One fish, two fish, butterfish, trumpeter: Recognizing fish in underwater video. In *IAPR Conference on Machine Vision Applications*.

Saue, T. D. (2003). Klassifikasjon av levende fisk. Technical report, Norwegian University of Science and Technology, Department of Computer and Information Science.

Stanfill, C. (1987). Memory-based reasoning applied to english pronunciation. In *Proceedings of the Sixth National Conference on Artificial Intelligence.*, pages 577–581.

Suzuki, S. and Abe, K. (1985). Topological structural analysis of digitized binary images by border following. In *Computer Vision, Graphics, and image processing*, volume 30, pages 32–46.

Zion, B. (1999). Sorting fish by computer vision. *Computers and Electronics in Agriculture*, 23(3):175–187.

Zion, B., Alchanatis, V., Ostrovsky, V., Barki, A., and Karplus, I. (2007). Real-time underwater sorting of edible fish species. *Computers and Electronics in Agriculture*, 56(1):34 – 45.

Zion, B., Shklyar, A., and Karplus, I. (2000). In-vivo fish sorting by computer vision. *Aquacultural Engineering*, 22(3):165 – 179.