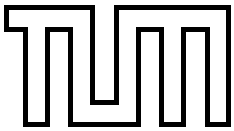
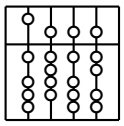


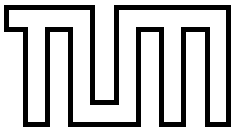
Technische Universität München
Fakultät für Informatik



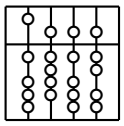
Diploma Thesis in Computer Science

Automatic Feature Computation for Endoscopic Image Classification

Ulrich Friedrich Klank



Technische Universität München
Fakultät für Informatik



Diploma Thesis in Computer Science

Automatic Feature Computation for Endoscopic Image Classification

Ulrich Friedrich Klank

Advisor:	Prof. Nassir Navab
Supervisor:	Nicolas Padoy
Date:	May 14, 2007

Ich versichere, dass ich diese Diplomarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 21. Mai 2007 Autor

Contents

Part I

Preface

Abstract

This work addresses the recognition of phases in minimally invasive surgeries. These phases are sequential in time and should be recognized using observations which are acquired automatically. The endoscopic view of the surgeon provides information about the surgical field. This work focuses on the recognition of surgical phases using image features computed from these endoscopic images. It is a complex task to select efficient features and few literature exists about features that discriminate surgical phases in endoscopic views.

In this work, a new Genetic Programming approach is proposed to automate the search for efficient features. A feature is modeled as a program and those programs are evolved to improve the recognition rate. For the representation of the programs a programming language was defined, specialized on computation of image features. Programs of this programming language are evaluated by executing them on a virtual machine with labeled sample images as input. Once the programs are evaluated and achieved a so called fitness, the best programs are selected as parents for a slightly changed and probably improved new generation of programs.

Finally, the resulting features are compared with several standard image features, to show their performance in distinguishing between two phases using an image. With a selection of the best features, a multi-class classifier is built. It is compared with an early approach which is based on a neural network fed with a set of standard image features.

entia non sunt multiplicanda praeter necessitatem
(entities should not be multiplied beyond necessity)

– William of Ockham, 14th century (Ockham's razor)

Acknowledgement

I want to thank especially my supervisor Nicolas Padoy for organizing the whole project, looking up everything and always supporting me. I also want to thank my advisor Professor Nassir Navab for giving me the chance of accept all wishes and still including me in his projects. I want to thank Martin Horn for all my hardware issues. I want to thank the Workflow group for all their ideas. Also I want to thank the Klinikum Rechts der Isar, for allowing us to film the surgeries. A special thank is granted to the Magrit group at the LORIA in Nancy, France, for integrating me during the final phase of my thesis.

Part II

Automatic Feature Computation for Endoscopic Image Classification

Chapter 1

Introduction

Image features are a central element of image processing. They represent properties of an image. Whenever a problem can be solved using information in an image, adequate features are one way to extract this information. Images contain a large amount of information, but not all of it is needed. Here, features are the key to extract the relevant information. But since only some features are relevant, the features selection has to be adapted to the problem. This specialization of the feature set can be automated and this work shows an approach to perform this automatization.

1.1 Motivation

We wanted to reach the following objective: we want to recognize phases in minimally invasive surgeries. Based on what the surgeon is doing, different phases of a surgery should be recognized. Most minimally invasive surgeries have a strict scheme of actions for the surgeons. This will be called the surgical workflow, for more details see section 3.5. The workflow can be extracted by observing surgeries, for instance in order to provide phase dependant interfaces to the surgeon. Also other applications like the support of training systems or data acquisition for such surgeries would benefit from it.

A very useful observing tool in a minimally invasive surgery is the endoscope. The endoscope image of a patient shows the visual development during the surgery. Also the action of a surgeon can often be seen directly. So, we want to recognize a surgical phase in an endoscopic image. It can be hardly said what distinguishes the phases, but there are differences in the images of the phases. These differences should be representable by features. This leads to the topic of this work, which is the automatic feature computation for endoscopic image classification. The next three sections give a short introduction of what stands behind the parts of this title.

1.2 Classes of endoscopic images

As surgery type, this work uses cholecystectomies, which means gallbladder resection. It is a standard surgery that is frequently performed. The duration of it is about one and a

Table 1.1: The surgical phases, that are used as classification labels

Label	**Phase name**	**Remarks**
1	CO2 inflation	No endoscope
2	Trocar insertion	
3	Dissection phase 1	
4	Clipping/ cutting 1	
5	Dissection phase 2	
6	Clipping/ cutting 2	
7	Gallbladder detaching	
8	Liver bed coagulation 1	
9	Packaging of gallbladder	
10	External gb. retraction	
11	External cleaning	
12	Liver bed coagulation 2	
13	Trocar retraction	
14	Abdominal suturing	Endoscope outside

half hours.

The classes we want to recognize in the endoscopic images are phases of such a surgery. The phase definition of [Ahmadi, 2003] will be used as ground truth. The different phases are listed in Table 1.1. Every name describes the main action of the surgeon during this phase. To illustrate the image material Figure 1.1(a), Figure 1.1(c) and Figure 1.1(e) show endoscopic images from the second cutting and clipping phase. These images should be separated automatically from images like Figure 1.1(b), Figure 1.1(d) and Figure 1.1(f). These three images are from the external gallbladder retraction phase. The differences in theses images can be recognized. An experienced observer can distinguish most of the phases visually.

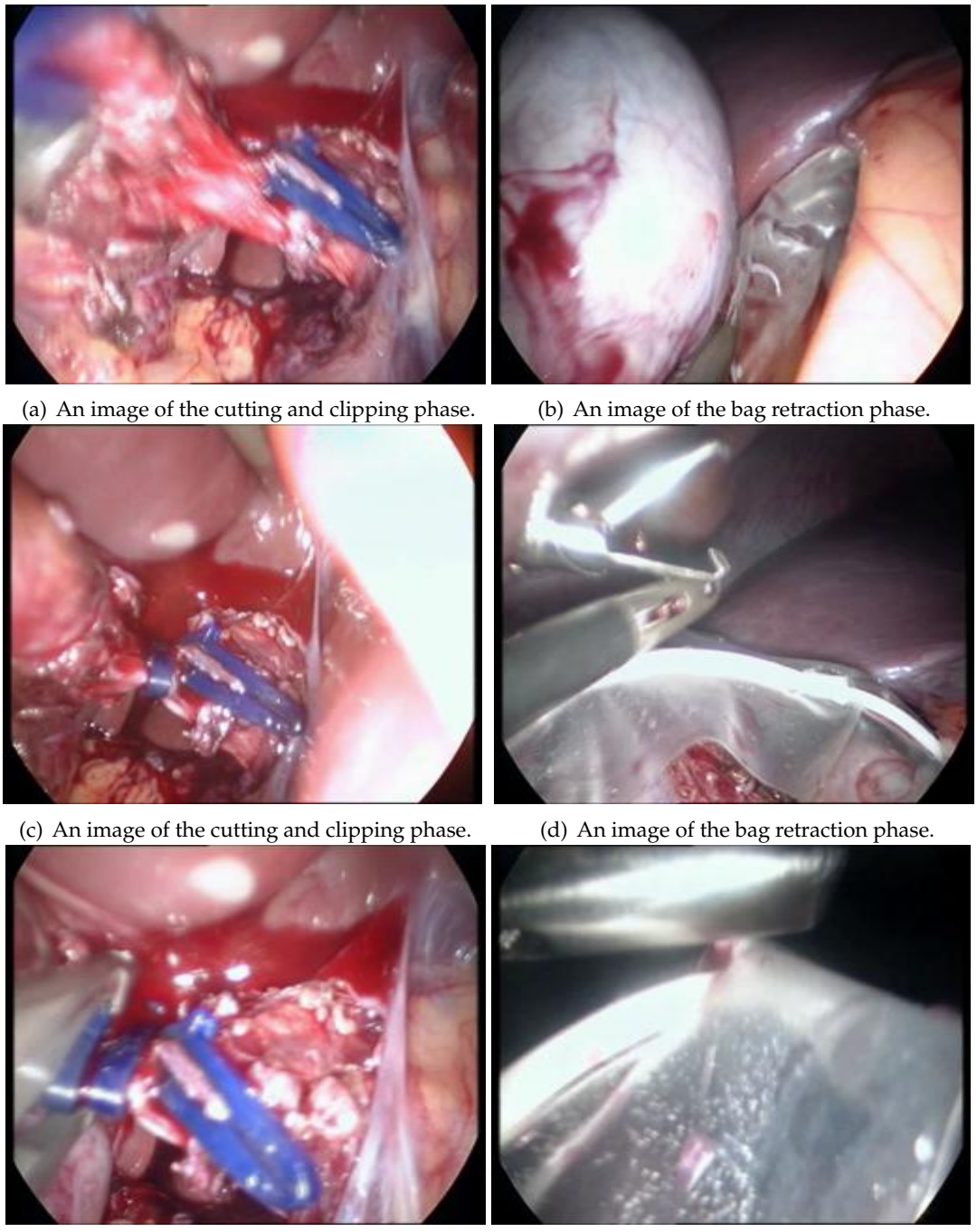
1.3 Automatic Feature Computation

For endoscopic images in general, there are few features in the literature, for details see section 2.2 and 3.3. These features are mostly used for pathology detection, while this work observes the surgeon's interaction in the minimally invasive surgery. So, there is few information about a good features and few about a good feature set.

To solve this lack of knowledge, a Genetic Programming system was implemented. This system searches for features, that extract the differences between the different surgical phases. Given a set of labeled images, and some programs implementing basic features, this Genetic Programming system will evolve new programs, implementing new features. Their performance to distinguish the phases of the generated features is compared with several basic images features that are better fulfilling the needs to distinguish the phases than earlier programs. See section 5.3.3 for more details. An introduction to Genetic Programming can be found in section 2.1, related work over Genetic Programming can be found in section 3.2.2 and 3.2.2. The details of the Genetic Programming system will be described in chapter 4.

1.4 Image Classification

To estimate the current phase a classifier was built with the generated features. This classifier consists of two levels. A feature is generated in order to distinguish between two phases. Therefore a 2-class Support Vector Machine was the choice to build the first level classifier. See section 2.3 for a discussion of different classifiers. We generated a feature and a corresponding classifier for every pair of phases, and used their classification results for the final classification. This final classifier is a voting decision, and will be introduced in section 4.5.2. Its results can be seen in section 5.2.2. We compare this approach with a direct classification using a Neural Network on a combination of image features.



(a) An image of the cutting and clipping phase.
(b) An image of the bag retraction phase.
(c) An image of the cutting and clipping phase.
(d) An image of the bag retraction phase.
(e) An image of the cutting and clipping phase.
(f) An image of the bag retraction phase.

Figure 1.1: Several samples of endoscopic images in a gallbladder resection.

Chapter 2

Theoretical Background

Since this work unites several techniques from different fields of computer science, this chapter gives an overview over these techniques. First, Genetic Programming will be presented, which can be seen as the central part of this work. Image features will be introduced in the following section. We will also give a definition of how the word "feature" is used in this work and provide a selection of basic features. The last section discusses some classification methods and their problems.

2.1 Genetic Programming

Genetic Programming (GP), following the ideas of [Koza, 1990], is a machine learning technique. In general, a GP algorithm adapts a program to achieve a defined goal. In Figure 2.1, the principle of GP can be seen: the process starts with programs and their code. A so called fitness function performs the evaluation of these programs. After a selection of the best programs, a reproduction step produces new code for new programs. This loop leads to an increasing quality of the programs under certain conditions. The three most important requirements therefore are: a correct fitness function, a good selection strategy and a working reproduction technique. These are the basic elements of GP, which will be explained in further detail in the next sections.

2.1.1 Fitness function

The fitness function models all aspects of the goal. This model represent the goals in a non-discrete way, to allow a smooth approximation towards the goal. Using this model, the behavior or the results of a program can be evaluated. This evaluation is called the fitness of this special program. The fitness function is always problem specific, so there is no universal strategy for the design of it. Naturally, its complexity will increase with the complexity of the goal. But in most cases a simple fitness function will lead to better results, following to Ockham's razor. This is obvious, since a GP system could abuse any side effects in the fitness function. A common sample is an undefined behavior near singularities of any function.

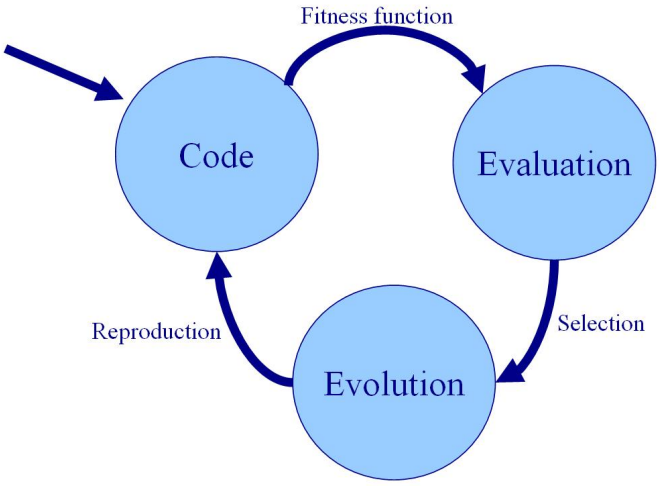


Figure 2.1: The basic components of Genetic Programming

A fitness function often consists of two parts, a static part and a dynamic part. The static part performs an analysis of the program itself by semantic criteria. The dynamic part executes the program in the target environment. This is obviously the most time consuming part. So, any fitness function should include as many static tests as possible. Knowing some fitness criteria before a new generated program is executed for the first time can save time, especially when the tests are expensive.

However the fitness is calculated, it will serve for the selection.

2.1.2 Selection

One main idea in the evolution theory of Charles Darwin, [Darwin, 1859], is that natural selection takes place in a world with limited resources. Meaning, only the fittest individuals of a population will survive. Transferring this to the world of GP, a program will evolve if it is confronted with selection. The GP algorithm has to decide which programs will be kept and used as a base for further generations. There are several strategies for selection. Two examples will be presented here, which are representative for most of the ideas in these strategies. First, the tournament strategy: The fitness of two programs is compared directly, and only the one with the higher fitness will survive. Second, the limited population strategy: A program can be selected for breeding as long as it is in the population. The programs with the lowest fitness will be taken out of the population. Figuratively spoken, these programs will starve. Once some programs are selected, they will be used to build the next generation.

2.1.3 Reproduction

Another main issue in the modern evolution theory is the reproduction. Every individual has as secondary goal, beyond surviving the selection, to generate offspring. This offspring should be at least as successful as the parental generation. In order to fulfill this principle

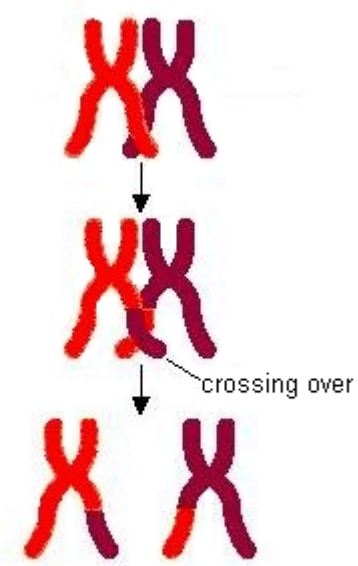


Figure 2.2: Crossing over as the most important recombination strategy, here a schema of the biological inspiration: The crossing over of chromosomes.

in GP, the parental programs have to give their abilities to their offspring. But, to challenge increasing concurrency, the next generation has to be better. This is tried to reach by slightly changing the next generation of programs using two strategies. The first strategy is the point mutation. For programs, this can be easily performed by reordering commands, changing commands, by inserting or removing commands or changing parameters of the program.

More problematic is the combination of two or more parental programs, which is the second strategy for program adaptation. The standard recombination method is the so called crossing over. Generally spoken, this strategy combines blocks out of the parental genetic materials.

Figure 2.2 illustrates the biologic observable crossing over of DNA chromosomes, that serves as model for the strategy used in GP. The relevant parameters of a crossing over are the number of crossings and the desired length of a block.

Using crossing over can cause several problems. The syntactic correctness must be granted. This is normally solved by specifying the programming language adequately, more on this, see 2.1.4. But the semantic sense of such a combination is a serious matter. To solve this problems there are many different approaches. Most of these approaches work with an intermediate representation of the code. This allows easy recombination. The intermediate representation has to be transformed. This transformation is done in different ways, including complex ontogenies, like it was done in [Josh C. Bongard, 2003]. But the easiest way is to just use a linear representable programming language, that allows recombination without syntactical problems. This approach does not consider the semantic problems, hoping the fitness function will filter out all senseless programs.

2.1.4 Code

The selected code representation depends first of all on the problem statement. The questions are, what should be computable with the used programming language and what performance criteria exist. On the other side the recombination and mutation capabilities, that should be possible, will also limit the selection of the code representation.

In the literature there two basic types of programming languages which are used in GP and have to be distinguished. Programming languages can be linear or tree based. An example for a linear programming language would be an imperative language like c and for a tree based language a functional language like LISP, which was basically designed for artificial intelligence applications. The tree based variant is favored by the standard books about Genetic Programming, [Koza, 1990] and [Banzhaf *et al.* , 1998].

Normally a custom operator set is provided. On the one side, the number of commands must be held low, to reduce the combination possibilities. On the other side, the basic capabilities of a programming language have to be preserved. The capabilities of a programming language can be ordered in theoretical hierarchies based on Chomskys work about grammars. For the GP especially the primitive recursive functions and the μ-recursion are interesting. For a formal definition of these two see [Kleene, 1952]. Normally one of these two classes of calculability is chosen. This depends on the chosen operator set. The capabilities of loop and jump operators condition the class.

To speed up the evolution, higher level functions can be inserted as operator. The selection of these functions is task specific. The insertion of a function will precondition the solution in a direction, that can be against the current interest. High level constructs are in advantage of low level constructs, that still have to evolve. So, the selection of the operator set is a delicate task.

2.2 Image features

An image feature consists of information extracted from an image. It describes the properties of this specific image. This section defines an image feature in a formal way and gives examples for basic features. We will use several of these examples later on.

Formal definition

To formalize a feature, we will define it as a mapping between an image and an corresponding output:

$$f : I -> F,$$

with a dimensionality for I as the product of image height, image width and color depth. The dimensionality of the feature F is less or equal than the dimensionality of I.

$$Dim(I) := N_I,$$

$$N_I = height \times width \times colordepth,$$

$$Dim(F) \in [1...N_I]\,.$$

Both dimensionalities refer to a binary vector. Based on this, the number of possible features for an image type can be counted. The possible number of different images is:

$$N_{images} = 2^{N_I}.$$

For an image, the number of possible results a feature can calculate is the sum over the possible bit combination with a length less or equal to the image size:

$$N_{results} = \sum_{i=1}^{N_I} 2^i.$$

This expression is equivalent to

$$N_{results} = 2^{N_I+1}.$$

Using N_{images} and $N_{results}$, the number of possible matches can be limited to:

$$N_{features} \approx N_{images}^{N_{results}},$$

$$N_{features} \approx \left(2^{N_I}\right)^{2^{(N_I+1)}}.$$

The result of this calculation is, that there are nearly infinite different features and it would be problematic to claim to know all relevant or even the most important.

Even so, there are several standard features for image understanding. The next section will give a short overview, what kind of features are commonly used.

2.2.1 Color features

The most common color based feature is the color histogram. Principally, a color histogram counts how often a specific color can be found in an image. A color histogram normally does not cover the whole used color spectrum. It reduces the depth by accumulating similar colors. The definition of similarity of colors depends on the used color space. The most common color space is the additive Red Green Blue (RGB) color space. A similarity measure defined over geometric error, will measure the color difference depending always on the intensity. Since color has influence in all three dimension, a similarity measure will always take the intensity and the color equally into account. The human perception separates intensities and colors.

To separate the intensities from the colors, the Hue Saturation Value (HSV) color space can be used. Here the color has an own dimension. So a simple similarity measure will easily combine similar colors into histogram bins. So, a HSV color histogram preserves similarities between images, that corresponds more to the human perception. See Figure 2.3 for a sketch of the three dimension of the HSV color space. The Hue dimension defines the color spectrum, the value defines the brightness.

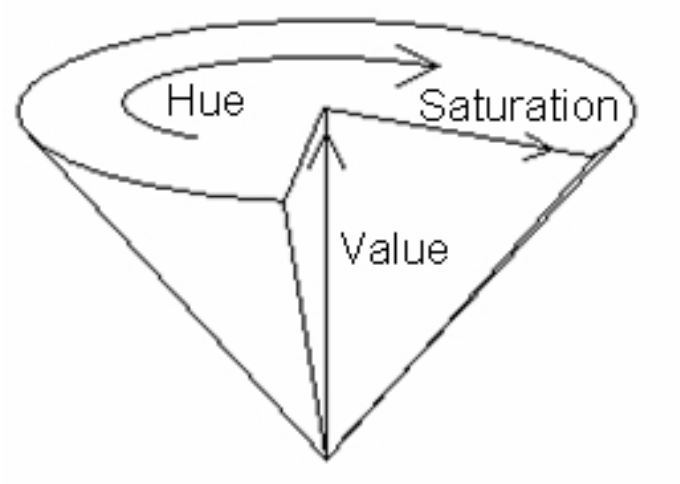


Figure 2.3: A schema of how the 3 dimensions of the HSV-Colorspace are connected.

2.2.2 Gradient features

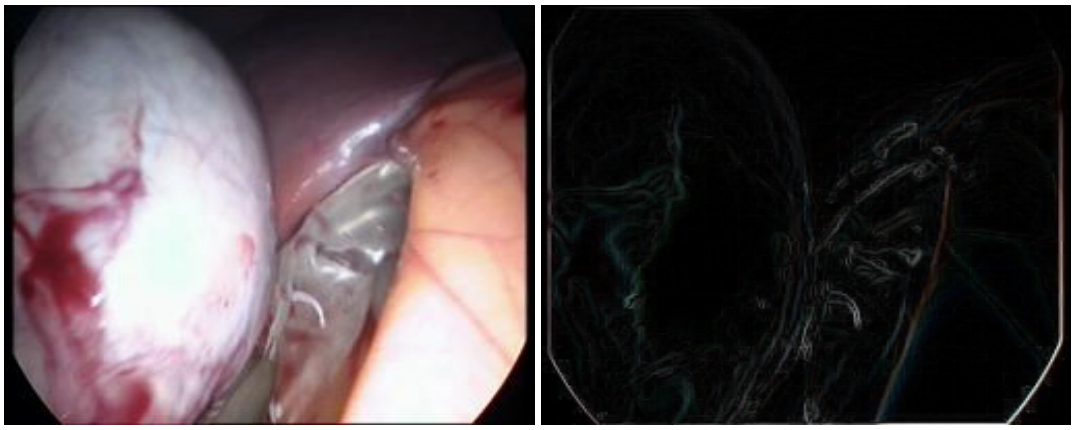
Gradient features are based on the gradient map of an image. Similar to the gradient are edge extractions, but an edge extraction usually includes several steps besides a simple gradient filter. We will not discuss the differences in the method, but since the mayor effects are similar, we will treat them as one category. In Figure 2.4 the effects of edge detection can be seen.

Gradient histogram

The easiest example for a gradient based feature is the gradient histogram. This feature just counts the number of edges with a certain strength in a specific direction. But there are more complex features based on edges. Gradients can be applied with certain preferred direction. This means that only edges with a perpendicular direction will be detected. Therefore even for such a simple feature like gradient histograms, there is a great variability.

Water filling feature

A more complex example for a gradient based feature is the water filling feature. Here shape based statistics are calculated. Or more specific, the water filling algorithm is applied on a canny edge map. All edge points in this edge map will be marked as filled starting with any not marked point and marking all of its 4-neighborhood. During this filling, four statistical values are observed. One of them is the maximal number of pixels in one connected edge with the corresponding number of forks. A fork exists if the number of connected and unmarked edge pixels is greater than one. Second, a histogram is created over these numbers of pixels, containing also the average fork count per bin. The third statistical value is the maximum fork count with the corresponding number of connected pixels and the forth is



(a) An image of the cutting and clipping phase. (b) The edge detection.

Figure 2.4: A sample image and its gradient image

a comparable histogram over these numbers, this time containing the average number of connected pixels. These features are rotation and scale invariant.

2.2.3 Motion features

The motion is defined as the direction in that the main components in an image move towards in a second image from a series. One feature to measure it is the Optical Flow, published by [Horn & Schunck, 1980]. The optical flow can be visualized by so called flow pattern, see Figure 2.5 for an example. This image show the relative movement of all points in the image. A direction and a strength defines the movement, these are visualized with the small lines in the image. Over these information histograms, cluster or maps with a lower dimension than the image can be used as features.

2.2.4 Texture features

A texture can be defined as the relations of a pixel to the pixels in its neighborhood. A texture based feature calculation tries to categorize these relations. Those categories of textures can be used for postprocessing. Possible postprocessing are a clustering and counting of pixels with the same value for a histogram. Normally texture features are based on a 8-pixel neighborhood, here are two examples for such texture features.

Local binary pattern

The LBP is a gray-scale invariant texture measure in the local neighborhood, it is used by [Wang *et al.* , 2001]. Figure 2.6 shows the calculation of a basic LBP, based in this case on a the 8-neighborhood. First, by using a threshold, the neighborhood of a pixel is transformed into a binary pattern, by replacing them by 1 for a grater value and 0 for a smaller. The threshold can be based on the value of the pixel. The binary pattern is used to generate a

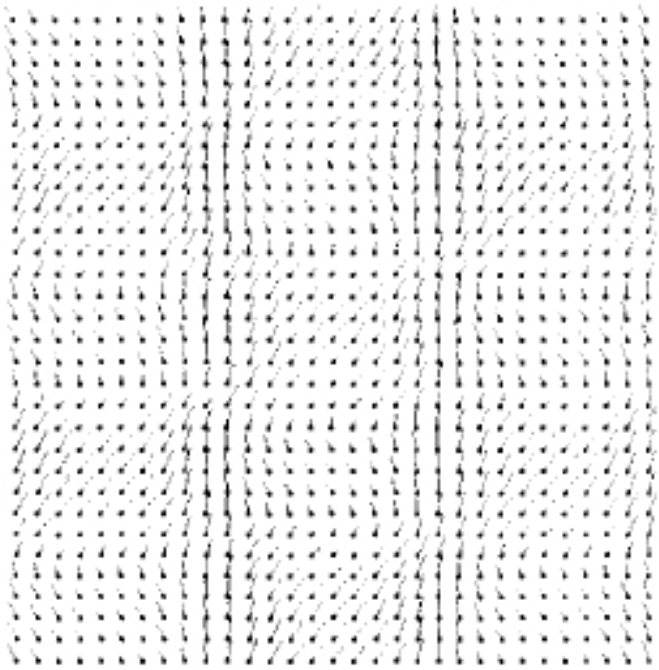


Figure 2.5: A flow pattern.

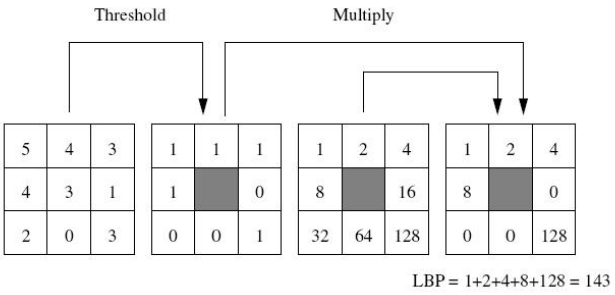


Figure 2.6: Calculation of a LBP code and its contrast measure

code, just reading the zeros and ones of the 8 neighbored pixels in a certain order as a binary number. The LBP can be used to calculate histograms and to combine intensity values with their pattern in the neighborhood. The maximal size is 256 for the LBP value multiplied by the color depth. The LPB can be mapped into smaller histogram sizes, down to 8, without loosing too much information.

Texture spectrum map

A texture feature similar to the LBP is the texture spectrum map. It also refers to the local 8-neighborhood. It allows only 3^8 different texture units. The basis of 3 instead of a 2 in the LBP is caused by the three basic relations two pixels can have: brighter, similar or darker. So for a region class a representative subset of these 3^8 different texture types is selected, in order to segment regions of interest.

2.2.5 Principal Component Analysis

The Principal Component Analysis (PCA) is a more widely used algebraic technique than an image feature. But with the definition of an image feature in section 2.2, PCA can also be seen as an image feature. Beyond on images, it also can be applied on features or feature sets for general dimension reduction.

PCA analyzes a data set using the principal components, see [Bishop, n.d.]. It tries to generate representing vectors, the eigenvectors, for the data set. Any vector of the data set can be expressed as a linear combination of eigenvectors. This property can be used to reduce the dimensions.

The principal components are calculated by decomposing a matrix consisting of a set of training vectors. The decomposition is done with a singular value decomposition. Any singular value decomposition decomposes a Matrix I:

$$I_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^*_{n\times n}$$

where Σ contains the singular values, also known as eigenvalues of I. U contains the left eigenvectors and V the right eigenvectors.

After sorting the eigenvectors by the corresponding eigenvalues, a specific number of eigenvectors is taken. Here for, only the vectors with the highest corresponding eigenvalues are chosen. With these eigenvectors, a matrix can be build that transforms a vector from its own high dimensional space to a lower dimensional space. The own high dimensional space is the vector length. The low dimensional space is defined by the number of eigenvectors selected before. The entries of the new low dimensional vector are the coefficients needed to for the linear combination of the eigenvectors to represent the old vector.

Since a PCA based transformation preserves geometric position properties, the Mahalanobis distance can be used to determine distances between the resulting vectors transforming it into the eigenvector space. The Mahalanobis distance is defined as:

$$k(x,y) = \sqrt{(x-y)^T \Sigma^{-1} (x-y)}$$

with Σ, the covariance matrix of x's and y's distribution.

2.3 Classification

This section gives an overview over different classification methods. Classifiers are mostly trained by a data set. If the classifiers are not trained, they rely on a model, that is validated by earlier experiences. To validate a classifier, the classification rate for every class must be evaluated. These rates have to be handled with respect to a priori probabilities of the classes. Also a cross validation of the data set has to be performed, in order to get a reliable quality measure. Cross validation means that the data set is divided into different sets, and complementary combinations of these sets are used for training and testing. This validation method tries to detect both great problems of classifiers: overfitting and underfitting. Overfitting denotes that too many singularities of the training set are used to classify it. Underfitting implies that not even the training data can be classified correctly.

The first presented method is the k-means algorithm. This is a simple method and it also works for very little training sets. The Support Vector Machine(SVM) is a more advanced approach, which needs larger training sets, but provides higher classifications rates. The third presented approach is the Backpropagation Neural Network. Which is a theoretical more powerful method than the SVM, but tends to have application problems. Last, model based multi-class decisions will be discussed.

2.3.1 k-Means

The k-Means algorithm was first published by [Forgy, 1965] and provides an easy linear classification method, based on a geometric distance. It tries to separate a data set into k clusters, by assigning every point of the data set to the nearest cluster. By doing this iteratively, updating every round the centroid of every cluster, and reassigning all points, a stable linear classification will be reached after several iterations. The algorithm can be started with a random initialization, to examine the existence of various clusters. If the existence of different classes is provided by the data set properties, even the initialization of the whole data set into one cluster will give a result.

If a training set exists, the convergence of the algorithm can be speeded up. The centroides of the different classes of the training set can be used as initial configuration of the k clusters.

2.3.2 Support Vector Machine

An advanced classification method is the Support Vector Machine (SVM). See [Burges, 1998]. The SVM tries to find a hyperplane separating different classes of vectors. Since it is a maximal margin classifier, this classifier does not only separate the training data, but it also searches for an optimal separation.

Figure 2.7 illustrates a hyperplane and the corresponding margin. The margin is determined summing up the distance of every support vector to the hyperplane. The support vectors are a subset of the training set. A vector is selected as a support vector by its distance to the borders of its class. The distance measurement takes place in a kernel space. The vectors are first transformed in the kernel space, and there a algebraic distance is calculated. Several different standard kernels are available.

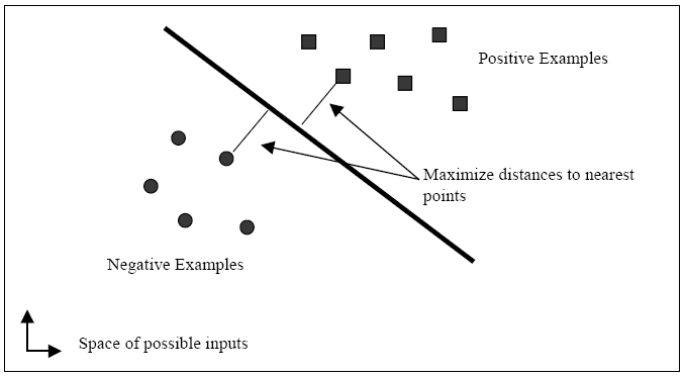


Figure 2.7: An example for a linear support vector machine.

Distance measure in kernel space

The kernel function defines the kernel space. It calculates the distance between two vectors. The easiest way to calculate this distance is the linear kernel. It takes the geometric error as distance. An example for a non-linear kernel is the RBF kernel. This kernel allows a non-linear separation of classes with the SVM. A kernel is defined by a distance function, which returns the distance between two points:

$$k(x, y) = e^{\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)}, \sigma > 0$$

Where k is the distance function, x and y are vectors and σ is the RBF kernel size. Alternatively a polynomial kernel can be used:

$$k(x, y) = (xy)^d, d \in I\!N$$

The RBF kernel returns values different from zero only inside the area specified by σ and it has a well defined behavior at the borders of the definition space. This is not given by the polynomial kernel, which can have unexpected behaviors at the borders.

Training methods

The standard SVM training method assumes the separability of the classes. There are training methods for complex problems that allow a certain number of wrong classified training samples, by penalizing them with a negative distance. One of these training methods is the ν-Support Vector Classification (ν-SVC), published by [Schölkopf *et al.* , 2000]. Another training method for example is the Sequential Minimal Optimization published by [Platt, 1999].

A crucial part of the training is the regression of the decision boundary depending on the selected support vectors. The ν-Support Vector Regression (ν-SVR) is part of the corresponding classification and tries to minimize three values: the training error, the model complexity and a tolerance corridor. The training error is the number of misclassified vectors. The model

complexity is the degree of the polynomial function that is used for the regression. The tolerance corridor can be imagined as the size of the tube that allows support vectors to be in the part of the regression or not.

The direct approach minimizes the following function:

$$e(SV) = \sum^{x_i \in SV} |y - f_w(x_i)|.$$

When a tolerance margin is added, the error function will be:

$$e(SV) = \sum^{x_i \in SV} |y - f_w(x_i)| - \epsilon,$$

with f as the decision function in both cases, defined as follows:

$$f_w(\mathbf{x}) = (\mathbf{wx}) + b,$$

These two functions are minimized regarding the error e itself and the model complexity $\|w\|$. The ν-SVR will minimize additionally the ϵ. Another parameter ν defines the wished fraction of the error and the fraction of the SV. The algorithms approximates these two values to ν, by minimizing ϵ. This simplifies the training of the SVM, since ν depends less on the given data as it would depend on with a directly given margin.

Abilities and Problems

The ν-SVC provides a fast and easy-to-use possibility to train SVMs. It only tends to perform very slowly for bad conditioned data sets. Also the corridor between over- and underfitting is narrow. Still, it can be seen as one of the best possibilities to classify high dimensional data sets.

2.3.3 Backpropagation Neural Network

Neural networks are another machine learning approach and can have the same capabilities as a SVM, but there are more parameters to learn. Due to the great number of learned parameters, their main problem is that the reason for working or not on a special class can hardly be understood.

Feed forward network

Feed forward networks after [Lenze, 1997] consist of a set of neurons. These are organized in an input layer, at least one hidden layer and an output layer. More hidden layers improve the learning capacity, but delay or even hinder a learning effect.

Figure 2.8 shows a scheme of a feed forward network with one hidden layer. Every neuron has an activation function, which defines how the incoming connection changes the neuron activation, that will be passed to the output. This activation function can be seen as

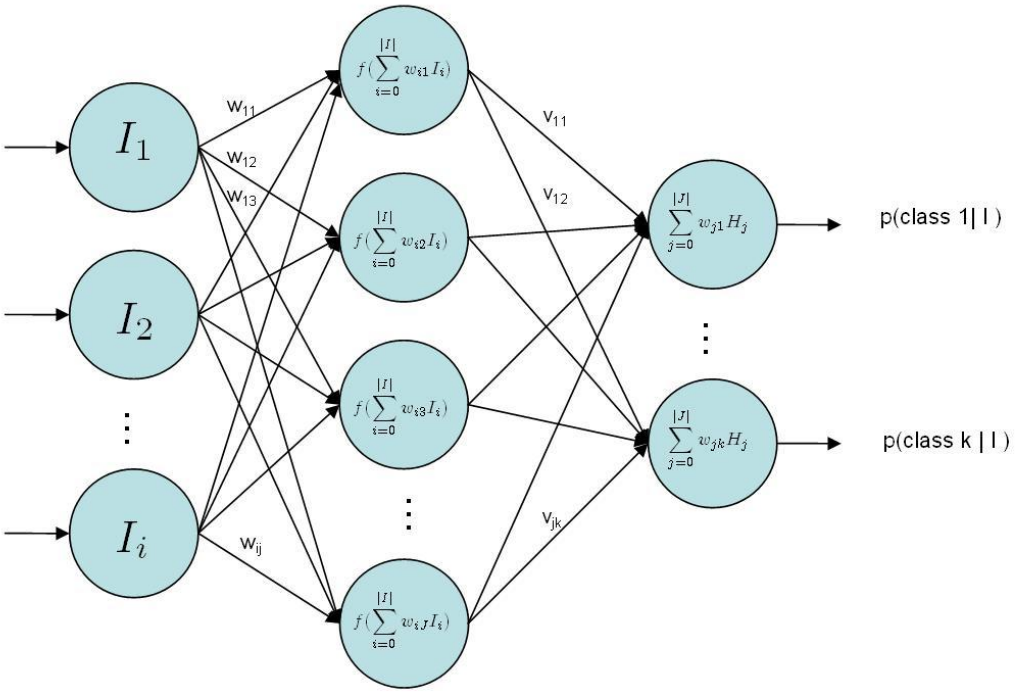


Figure 2.8: A scheme for a feed forward network. The input is the vector I with its entries $I_1..I_i$ and the length I. The results are the estimated probabilities for the special class.
s

equivalent to the kernel function of the SVM. For the input and output layers is no further need for an activation function. Most times the identity is used. The hidden layer requires an activation function. Since it has a simple derivative, the sigmoid function is a common choice:

$$f_\sigma(x) = \frac{1}{1 + e^{-(x-\sigma)}}$$

The sigmoid function has a parameter σ, that defines the value for x where the result of the function is 0.5. Using the activation function, the activation of the hidden layer can be calculated as followed:

$$J_j = f_{t_j}\left(\sum_{i=1}^{|I|} w_{i,j} I_i\right)$$

Where J_j is the activation of the j-th hidden layer neuron. The input vector is I, and the i-th entry is I_i. The weights $w_{i,j}$ have to be trained before. How to train a weight is described in the next section. t_j is a parameter for the activation function of a hidden layer neuron which has to be trained too.

The results are the activation output neurons. These are calculated from the hidden layer activations as follows:

$$A_o = \sum_{j=1}^{|J|} v_{j,o} J_j$$

The weights w_{o_j} have to be trained before, too. The activation A_o represents the calculated probability of the input vector belonging to the class o.

Learning algorithm

The Backpropagation algorithm is a form of supervised learning for a neural network. It requires enough learning samples that represent a class. Since the Backpropagation algorithm provides a method to learn patterns in the information relatively fast and with a good stability of already learned facts, it is presented here. During the learning process, several parameters are adjusted. This depends on the error of the feed forward calculation. The idea of Backpropagation is to use the error at the output neuron in all the layers to adapt all parameters. The influence will decrease every layer. The next evaluation of the same input will result in a smaller error.

The errors at the output neurons are defined as follows:

$$\epsilon_o = \begin{cases} 1 - A_o & \text{if class of } I = o \\ \frac{-1}{|O|} - A_o & \text{otherwise} \end{cases}$$

The weights between the hidden and the output layer are adapted using the formula:

$$v_{o_j}^{new} = v_{o,j}^{old} + 2\delta J_j \epsilon_o;$$

δ is an variable with a value below 1 that decides about the learning velocity. The lower the value of δ is, the more stable results obtained are. The higher it is, the faster is the learning process. J_j is the activation of the hidden neuron j as described in the section before.

The weights $w_{i,j}^{new}$ connecting the input neurons i and the hidden layer neuron j will be changed according to the following formula:

$$w_{i,j}^{new} = w_{i,j}^{old} + 2\delta v_{input}(i) f'_{t_j}(J_j) \epsilon w_{out_j}^{old};$$

with the derivative of the activation function $f'_{t_j}(J_j)$ would be $J_j(1 - J_j)$ in the case of the sigmoid function. The threshold $t_{hidden_j^{new}}$ of the hidden neurons is also changed:

$$t_{hidden_j^{new}} = t_{hidden_j}^{old} - 2\delta f'_{t_j}(J_j) \epsilon w_{out_j}^{old};$$

Once all weights updated, the error for the current input will be smaller than before and new training data can be passed as input, to adapt the weights again.

Abilities and Problems

The backpropagation learning algorithm can only be applied with a high risk of over fitting. The usage of PCA on the training data can reduce this risk, but PCA weights the information in a way that can cause problems, like removing the interesting part of the information. The learning time is high, compared to the SVM times. Anyway, it was shown that for some cases the classification results are be very good.

2.3.4 Multi-class extension

An approach to solve multi-class decision problems can use several 2-class classifiers. 2-class Classifiers are faster to train, due to a smaller training set size, that is required. Two simple methods to combine 2-class classifiers to a multi-class classifier are the one vs all and the pairwise method. The *one-vs-all* method uses classifiers that are trained on one class and samples from each other, to decide if a sample is in the specific class or not. The pairwise method trains classifiers on every pair of class, to decide in which class a sample is expected with a higher probability. Also [Hsu & Lin, 2000] comes to the solution that for SVM the pairwise method performs better than the one versus all, while [Rifkin & Klautau, 2004] in a more general context come to the conclusion that one vs all performs as good as any other technique. More sophisticated techniques, like proposed by [Dietterich & Bakiri, 1996, Allwein *et al.* , 2000], need an error model for the classified data, which we did not have for the generated features.

Chapter 3

Related Work

This chapter leads through the literature that is in some aspects related to this work. First, we present methods, systems and several application of Genetic Programming. Then, a list of applications based on image features is given, all applied on endoscopic images. As next point, techniques for feature selection will be explained. The last section presents works over surgical workflow and related topics.

3.1 Genetic Programming methods

Genetic programming is a very time consuming technique, since very large search spaces have to be explored. Also the evaluation of quality and correctness is not always easy. So, there are several methods in this area to improve the performance. [Johnson, 2002] run through a static analysis before execution and [Motsinger *et al.* , 2006, McConaghy & Gielen, 2006, Thomason & Soule, 2006] propose improvements to the reproduction. [Wang & Soule, 2004] provides a method for comparison of different function sets.

3.1.1 Fitness

To improve semantic tests, [Johnson, 2002] derives static analyzes over programs automatically, to supervise constraints on variables before an execution can be done. The work proposes static analysis of a program, in order to find out the general abilities of this program. Or namely:

- relationships of the values of variables and their extreme values
- references information that is crucial for the solution or not
- complexity information
- performance information

The work postulates that the optimal case for Genetic Programming would consist in a pure static fitness analysis, for time and failure safety reasons.

3.1.2 Reproduction

An area where a lot of specialization can be done is the reproduction. Improvements of the reproduction can increase the probability that newly combined programs are correct and useful. Several crossing over strategies are discussed in [Motsinger *et al.* , 2006] and a new technique is presented. This work uses a Neural Network design pattern as programming language. Another technique to supervise new programs was defined by [McConaghy & Gielen, 2006]. They use a a canonical form for programs. This form makes it possible to correct logical errors in the parametrization of commands. Since the parametrization can be random, this is a very useful method for the generation of new programs. [Thomason & Soule, 2006] proposes redundant genes for a higher robustness of the evolution. Their results show, that evolution exerts pressure on the individuals of a population towards robustness of their genom.

3.1.3 Function set comparison

[Wang & Soule, 2004] runs tests over different operator sets, and shows that the optimal operator set can be searched in limited function spaces by comparison of the evolution speed. He uses this fact, to divide the operator set into functionality groups. The final result of the work is, that only one operator of such a functionality group is needed for the optimal operator set.

3.2 Genetic Programming with images

Genetic Programming (GP) has a large area of applications. This section will focus on applications in the area of image understanding. We will look at two different ways to use GP. First, directly accessing the images and second using the images indirectly by a fixed feature set, which is extracted from the image. The first serves more for segmentation of the image or object recognition, like in [Szymanski *et al.* , 2002, Perkins *et al.* , 2000, Ghosh & Mitchell, 2006]. The second method is often applied only on regions of images, and mostly serves for classification, like in [Zhang *et al.* , 2003, Nandi *et al.* , 2006].

3.2.1 Direct approach

An approach can be called direct, when the created program has direct access to an image. Based a set of image operators, a generated program tries to calculate the results.

An example for such a GP environment is the Genetic imagery exploitation software (short GENIE) of the Los Alamos Laboratories, published in [Perkins *et al.* , 2000]. The GENIE software tries to assemble an algorithm based on basic image operators. The generated programs can be used for segmentation of an image in regions. GENIE uses a linear programming language. An application example can be found in [Szymanski *et al.* , 2002]. Here a segmentation of satellite images is performed. Another application example of GENIE in the medical field is published in [Ghosh & Mitchell, 2006]. Herein the goal of segmenting a prostate in CT images is reached with the Genetic Programming abilities of GENIE.

3.2.2 Indirect approach

The indirect approach combines lower level arithmetic operators and fixed image features. The image is only accessed by a fixed number of features, and only the results of these features are given as input to the program.

Such an approach is used by [Zhang *et al.* , 2003]. In this work uses just parts of images as input for a program. Instead, a region around a pixel is taken to calculate a limited and fixed set of features. These features are combined with arithmetic operators by a genetic programming system. The target of the project is to classify parts of the image into one of several classes. A similar approach is used by [Nandi *et al.* , 2006]. There, s detection of carcinoma malignity is performed, based on texture features, edge sharpness features and shape features. They also use Genetic Programming to combine the features, in order to classify finally a region containing a tumor as malign or not in four different levels.

3.3 Features for endoscopic images

In literature, many features for general images can be found. This section is restricted to applications that work with endoscopic images. Examples will be given for different types of features. All of them are used to automatically recognize pathologies. Applications or methods that are observing human interaction during a minimally invasive surgery, could not be found. So there is no comparable method, but different features that work with special cases of endoscopic images. Later on we will use some of these features for comparison. [Wang *et al.* , 2001, Karkanis *et al.* , 1991, Karkanis *et al.* , 2003, Iakovidis *et al.* , 2005] are pathology recognitions based on texture features, while [Majewski & Jedruch, 2005] uses shape based features for the same task. These two types of features are combined in an approach presented by [Nandi *et al.* , 2006].

3.3.1 Texture based features

Texture features were introduced in section 2.2.4. Representative examples have been selected to show applications of them. The first is a histogram based feature, and the second a region based clustering.

Texture features like in [Wang *et al.* , 2001] are used for tumor recognition. In the case of recognition ulcers and Melanosis coli in colonoscopic data, a local binary pattern (LBP, see section 2.2.4) histogram was used. This LBP was used to calculate a histogram with 256 bins for the LPB values and a second dimension of size 8 for the intensity. This histogram was calculated for every region of interest. If there was more than one region of interest in the image, the image was split into two regions, or the regions of interest were merged. A neural network was trained with the resulting histograms. The results were classes of regions. The neural network was trained unsupervised, and nevertheless it was able to classify ulcers and Melanosis coli regions in the images.

Also for tumor recognition, in [Karkanis *et al.* , 1991] a texture spectrum is calculated. See section 2.2.4. Here it serves for lung endoscopic images to decide about different cancer stages.

Not only texture but also color based features and histograms of colors are used. For example in [Karkanis *et al.* , 2003] and [Iakovidis *et al.* , 2005]. Here a Second-Order Color Wavelet Covariance feature in combination with various similar texture features is used for computer aided tumor detection. The calculation consists of transforming the image into a multi scale wavelet space, that represents its texture for every color channel differently. This is also used in tumor recognition.

3.3.2 Shape based features

Shape based features measure statistics over gradient maps. For example, they can help to decide if a special object is in an image. Or they represent special classes of objects. We will present two samples for usage of shape based features in endoscopic images.

A different approach to classify malignity of tumors is presented in [Majewski & Jedruch, 2005]. This method is based on the water filling feature. See section 2.2.2. The water filling algorithm is applied on the edge map for a previously selected region and the statistics are given to a classifier. This classifier decides over the malignity of tumors seen in endoscopic images.

Also [Nandi *et al.* , 2006] works with shape based features, which are the normalized compactness, the Fourier Descriptor, the spiculation index and the fractional concavity. These features are the base for a GP framework that evolves programs to classify the malignity of tumors in four classes. See section 3.2.2.

3.3.3 PCA as a feature

Images in general have a high dimension, which is why features are used to reduce it. But even features can have high dimensional result. PCA can be applied directly on the image or on features and feature sets like in [Tjoa & Krishnan, 2003]. In [Keil *et al.* , 2006], PCA is used directly on the image to determine if a CT slice shows the feet or the head of a patient, in order to get information about the position of this patient. This approach uses the PCA directly on the images. For both classes of images a centroid is calculated from a training set. A new image is classified using the Mahalanobis distance to the centroids.

An adapted version PCA is used for face recognition in [Turk & Pentland, 1991]. By defining several undecomposable dimensions an so called independent component analysis is used to extract more independent

3.4 Classification

Image classification based on features of medical images, or more specific on endoscopic images, is performed by [Majewski & Jedruch, 2005, Iakovidis *et al.* , 2005]. They use SVMs directly. The pairwise decision to solve a multi-class problem, is used by [Kudo & Matsumoto, 2001]. Another used classification method are the neural networks, which for example is utilized [Tjoa & Krishnan, 2003, Wang *et al.* , 2001].

3.4.1 Applications of SVM

In [Majewski & Jedruch, 2005] a two class decision is performed in order to decide if a tumor is malign or not. It is based on a feature set, described before in 3.3.2. The method uses a least square support vector machine. Least square is the name for the used kernel space. Similar, [Iakovidis *et al.* , 2005] classifies, via a long list of features also using a support vector machine, the malignity of tumors. [Li *et al.* , 2003] present a comparison between different classifiers. The performance of classifying natural textures is measured based on translation-invariant feature sets. As A result that the SVM comes out on top, better than the Bayes classifier and learning vector quantization. They use Gaussian kernel for error measurement.

3.4.2 Application of NN

[Tjoa & Krishnan, 2003] use a neural network with a backpropagation learning algorithm for feature set classification. On the feature set, containing texture and color based features, a PCA is also applied to reduce the dimension of the data. This leads to lower training times of the neural network. With the reduced training time, the risk of overfitting the training data is diminished, risking to loose information in the processed data.

The work of [Wang *et al.* , 2001], cited before in section 3.3.1, uses a self organizing map. The work was mentioned before, since it extracts texture features to classify regions of endoscopic images. Self organizing maps are a special kind of neural networks. It uses an unsupervised learning strategy and has an activation strategy that includes backward links.

Feature Selection with a NN

Like the PCA, there are more dimension reduction techniques. More specialized towards features for example is the work of [Jain & Zongker, 1997]. This work discusses several methods of feature reduction, including a method with neural networks. It compares a NN with statistical pattern recognition methods, divided into categories. As categories for the statistical methods the work uses optimal and suboptimal search strategies. As winner of this comparison results a suboptimal statistical method, the so called floating search method. The approach presented by [Farmer *et al.* , 2004] can be seen as extension to this work. It is based on random mutation hill climbing, and works for large scale problems. It is an approach, similar to the principle of a genetic algorithm. It selects several feature sets and evaluates the fitness of these features sets. The best feature set is slightly mutated by removing or adding features. The underlaying image material was taken inside a car, to decide if the airbag should stay activated or not. A benchmark system for image features was proposed by [Ma & Zhang, 1998] and applied on interesting images features that can be used to combine extrinsic information with the right image features. In [Vailaya *et al.* , 2001] this idea is applied to classify vacation images.

3.4.3 Pairwise voting

The pairwise decision with SVM, followed by voting to get a multi-class decision is done by [Kudo & Matsumoto, 2001]. This work is about the analysis of English phrase parts, so

called chunks. The words are divided into five classes, and the used classification system choose the correct class for a word. They provide a method for chunking in general, based on SVM and a weighted voting algorithm.

3.5 Surgical workflow recovery

The analysis of surgical workflow is a rising topic in the medical engineering area. In order to be able to train new surgeons in simulated environment or to use supporting systems like surgical robots, the necessary actions for a surgery first have to be recognized, analyzed, standardized and modeled. If a model exist, this has to be recognized in the reality to have influence on reactions of any system, like it is done in [Lo *et al.* , 2003, Lin *et al.* , 2005, Sielhorst *et al.* , 2006, Padoy *et al.* , 2007, Ahmadi *et al.* , 2006].

3.5.1 Analysis of surgical endoscope sequences

[Lo *et al.* , 2003] presents a framework for the analysis of the endoscopic view of minimally invasive surgeries. The work uses several tools, including:

- instrument segmentation
- instrument tracking
- tissue deformation detection
- change of specular highlight detection
- suture and suturing detection

All these tools were used to track movements of instruments in video sequences of different types of minimally invasive operations. The approach is tested on several short video sequences. The test case is the classification of frames and episodes of frames into types of movements.

3.5.2 Movement recognition

The work of [Lin *et al.* , 2005] analyzes movements of robotic surgical systems like Intuitive Surgical's da Vinci. These movements are mapped on a list of different actions a surgeon does with this system. This recognition is part of a surgeon training system, an example to examine of the skill of the person using the robot.

The work of [Sielhorst *et al.* , 2006] is about the synchronization of 3D movement trajectories, embedded in several contexts. It also includes the idea of a training application for surgeons, that provides a trainee with the possibility to follow movements of an expert surgeon. The movement trajectories are captured via outside in tracking which is part of the used RAMP augmented reality system.

3.5.3 Surgery synchronization

[Ahmadi *et al.* , 2006, Padoy *et al.* , 2007] synchronize operations based on the use of instruments in surgeries. One is based on dynamic time warping and the other on hidden markov models. The time synchronization serves to find corresponding instrument changes in different surgeries. In case of the DTW, by warping a surgery to each other, a new surgery can be divided into phases. In case of the HMM approach, the states of the HMM provide this division. This allows any kind of phase dependant feedback in a training application or a supporting system, that could directly support a surgery.

Chapter 4

Method

For this project a Genetic Programming (GP) framework has been implemented. It included a new virtual machine for image processing, which simulates the execution of feature programs. The feature programs are written a linear programming language, specially defined for this purpose. First the basics of the virtual machine and the programming languages will be presented. As a next step the evaluation mechanism of the GP framework will be described. The last section of the chapter describes, how the resulting features are used to build a multi-class classifier.

4.1 Virtual machine

Semantic problems in a generated program are normal and have to be considered. See section 2.1. To avoid infinite running times, memory leak or stack overflows, a generated program should run on a Virtual Machine (VM). The VM will supervise a safe execution and provide an interface for state changes, that can be used to implement the custom commands.

4.1.1 Command interface

A command for the VM consists of the following items:

- Operation Code (OpCode)
- Function that defines the command
- Operation name
- Parameter sets (3 integer values)

By defining these 4 items a command can be added to the operator set of the VM. A program will consist of a sequence of OpCodes and a Parameter lists. The functionality of a command is defined in a callback function. These callback functions have access to the VM interface. This interface allows the commands to also change the state of the VM, like for example the instruction pointer. The instruction pointer specifies the next line that will be executed from a program. The other states of the VM are described in the following sections.

4.1.2 Memory environment

The memory the VM provides to the program for execution consists of two parts. First, a stack is provided where the program can access only the upper element. Second, a heap structure can be directly addressed by the program. Both are restricted in size and initialized with 0. The lower part of the heap will automatically be saved on a sub procedure call and reloaded on a return.

4.1.3 Input handling

The virtual machine provides to the program for access to an input image. It offers also the possibility to change a working copy of this image with image operators. A stack of such working copies can be managed using special push and pop commands.

4.2 Programming language

The used programming language is a linear. It is assembler-like, since the number of parameters is limited to three. In the following sections we will present the basic structure of a program and the used operator sets.

4.2.1 Programs

Programs are the genetic material in the Genetic Programming system, they are specified by an ID, an corresponding operator set, a program length and the list of commands. Every command has a line number, an OpCode and three parameters. The programs will be managed using an ID, to distinguish them from the outside. The operator set limits the maximal opcode, and is used by the virtual machine to execute the program correctly.

Manually implement feature

Appendix A lists several sample programs. Here the structure can be seen. There is one program that was manually created to implement the water filling feature. See section 2.2.2. Manually created programs tend to be very long and complicate, compared to the same program in a high level programming language like for example C++. This is caused by the linear style, meaning that all kind of arguments are atomic. So every complex expression has to be solved on the stack.

Computed feature

To illustrate how a program looks like we will have a look at a generated program. If we compare the water filling feature with the second example: the generated program for to distinction between phases 2 and 6. It is used later in section 5.1. The code can be found in Appendix A.2, the operators that are used for the programs are explained in section 4.2.2.

The computed feature is much shorter, and easily structured. The first thing that can be seen when analyzing the used commands, is the large number of unnecessary commands. Half of the program's commands can be removed without changing its semantic. Also there are repeated structures in the short program. But on a closer look, the most repeated structure is a double extraction of the maximum and the minimum of the current working image on the stack. This structure can be seen as reaction to the fitness criteria, see section 4.3, that there must be an output on the stack and the input image must be referred. By the recombination technique crossing over blocks are preferred to stay together. See section 4.4.2.

To conclude these observation, the structure given by the programming language and the fitness criteria will strongly influence the resulting programs. So these elements have to be chosen carefully. Until a complex feature, comparable to the water filling, can evolve, many generations have to be run.

4.2.2 Operator sets

The values that are accepted by an operator as a parameter are normally limited to a certain interval. Parameter values will be forced to the closest border if they exceed this interval.

Low level operator set

Table 4.1 shows the list of low level operators. The operator set represents a programming language powerful enough to express μ-recursive functions. The operator set is not minimal, since there are operators that can be replaced by others. These operators were introduced in order to allow easier addressing. Since the VM has limited memory and allowed running time, not all μ-recursive functions will be computable by the system.

High level operator set

Table 4.2 show the list of the additional high level operators with a short description of their functionality. The selection is based on several sources, including [Perkins *et al.* , 2000], but most of it was chosen intuitionally. A part of the operators work directly on the input image. The functionality of these commands is most times implemented using OpenCV. They represent a selection of image operators and try to provide an interface for easy feature implementation. Combined with the low level operators, they also represent a μ-recursive programming language.

4.3 Evaluation

The evaluation system consists of three elements. First semantic tests are performed to preselect the programs. They are then executed in the VM for several inputs. The last step of the evaluation is then the calculation of the fitness function using the results of the execution.

Table 4.1: Low level operators, also part of the the high level operator set

Command name	Parameters	Functionality
FOR	Start, End, BlockLength	Repeating execution of the next block, defined by the *BlockLength* parameter, putting every turn an increasing value starting from *Start* up to *End* on the stack
IF	BlockLength	If the topmost stack value is equal to 0 the next block, defined by the *BlockLength* parameter, is not executed
POP	*None*	Deletes the last value from the stack
PUSH	Value	Pushes the *Value* parameter on the stack
PLUS	*None*	Takes the last two values from the stack and adds them
MINUS	*None*	Takes the last two values from the stack and subtracts them
MULTIPLY	*None*	Takes the last two values from the stack and multiplies them
DIVIDE	*None*	Takes the last two values from the stack and divides them
GREATER	*None*	Compares the last two values from the stack
NOT	*None*	Inverts the last value on the stack
EQUALS	*None*	Compares the last two values from the stack
SMALLER	*None*	Compares the last two values from the stack
DUPLICATE	*None*	Duplicates the last value on the stack
CALL	TargetLine	Calls a subroutine in line *TargetLine*
RETURN	*None*	Returns to the last call position
SAVE	Address, Value	Saves the value referenced by *Value* to the memory position *Address*. The value can be read from the stack
LOAD	Address	Calls a subroutine in line *TargetLine*
READINPUT	X, Y, Z	Reads the pixel of an image at position *X Y Z*
INPUTDIMENSION	Dimension	Puts the size in the dimension *Dimension* on the stack
SUBTRACT	Address, Value	Subtracts the value referenced by *Value* from the memory position *Address*. *Value* can be read from the stack
ADD	Address, Value	Adds the value referenced by *Value* to the memory position *Address*. *Value* can be read from the stack
JUMP	TargetLine	Jumps to the line *TargetLine*

Table 4.2: High level operators

HREADINPUT	*None*	Loads the input image as the current working image
GRADIENT	X, Y	Calculates a gradient in x direction if the *X* parameter is different to 0 else in y direction
HISTOGRAM	Size	Calculates a histogram from the current working image over the values with a resolution given by the *Size* parameter
EDGES	ThresX, ThresY	Applies a canny edge detection on the working image
GAUSS	KernelX, KernelY	Applies a gaussian filter on the working image
DOWNSCALE	Factor	Reduces the dimensionality of the image by the *Factor* parameter
EXTRACTMAX	Channel	Puts the maximal value of the working image on the stack
EXTRACTMIN	Channel	Puts the minimal value of the working image on the stack
EXTRACTMAXLOC	Channel	Puts the location of the maximal value of the working image on the stack
EXTRACTMINLOC	Channel	Puts the location of the minimal value of the working image on the stack
DILATE	Size, Iterations	Applies a dilation on the working image
ERODE	Size, Iterations	Applies a erosion on the working image
PUSHIMAGE	*None*	Puts a copy of the working image on the image stack
POPIMAGE	*None*	Deletes the topmost image of the image stack
ADDIMAGES	*None*	Takes the last image from the image stack and adds it to the working image
SUBTRACTIMAGES	*None*	Takes the last image from the image stack and subtracts it to the working image
MOTION	Direction	Takes the last image from the image stack and calculates the optical flow to the working image, *Direction* says if it is in x or y direction
THRESHOLD	Thres, Value	Sets all pixel in the images to *Value*, if their former value was over *Thres*. If it was smaller it will be set to 0
SETPIXEL	X Y Z	Writes the last stack value to the image at the specified location

4.3.1 Semantic test

Two semantic checks are performed. First, the program code is checked for commands referring to the input. A program that does not fulfill this criterion gets an evaluation result of zero and will not be included for further tests. The same will happen to programs that fail on the second semantic test, which checks all unconditioned jump statements for an obvious infinite loop. Or in more detail, it checks if the target line of a jump operator is lower than the line number of the statement itself. In case it jumps backwards, all the lines between jump target and jump statement are checked for another jump command.

4.3.2 Test runs

All programs, that passed the semantic test, will be executed with randomly selected images. These images are selected well distributed over the phases of interest, normally two phases. The random selection divides the phase duration according to the number of images needed and randomly selects one image in every part. Due to the different length of the phases, this has to be handled with care. A longer phase will have a greater variance of the selected images than a smaller phase. We used 64 frames per phase and video for the fitness function.

If an error occurs during the execution, the execution is stopped and the program will get assigned an evaluation result of zero. Possible errors are a stack overflow or exceeding of the time limit. Otherwise, for all images an output vector will be calculated and an average execution time will be measured. The output vectors and the average execution time are then passed to the fitness function.

4.3.3 Fitness function

The fitness function will take the results of the tests and calculate a value between 0 and 1, that will be assigned to the program as its fitness.

For a first test of a fitness function the output vectors were just taken half and half and assigned to the test or the training set. The fitness was just the classification rate of the test set.

For a second attempt, the fitness function was changed to reach a better generalization between the different surgeries. Therefore all resulting output vectors from three of four surgeries are taken as training set to initialize the clusters of the same k-means. The testing set consists of the resulting output vectors of the forth surgery. This is done for all possible combinations. The average classification rate defines 90 percent of the fitness of the program. The remaining 10 percent are defined by a percentage of the maximum execution time that was not used by an average execution of the program. Here the final fitness function:

$$f = \frac{9\sum_{i=1}^{N} c_i}{10N} + \frac{1 - \frac{t_{avg}}{t_{max}}}{10}$$

with c_i as the classification rate for the i-th surgery initialization of the k-means from all others surgeries, N as the number of surgeries available, t_{max} as the maximal allowed execution time and t_{avg} as the average execution time of a program.

4.4 Evolution

The evolutionary part of the system consists of the selection and the generation of new programs.

4.4.1 Selection

We use three kinds of selection in our system. There is a population of programs. To enter this population the initiation has to be passed. Once in the population, a program can be selected as parental program. Only parental programs generate offspring. And last, if the population exceeds a certain number of programs, a starving event will deselect a number of programs.

Initiation

To enter the population, a program has to have fitness higher than the population average. The average fitness is updated every turn. A turn ends when all new children have passed or failed their initiation. The definition of a turn is based on Figure 2.1 in section 2.1.

Choice of parents

The selection of the programs that are used for reproduction is based on the fitness plus a random component. First the two best programs are determined. These two best programs will be mutated each once using the method from section 4.4.2. This mutation will result two new programs. Then these two programs are combined with the crossing over recombination. See section 4.4.2.

For all other programs in the population a random event determines if they will serve as parental program. For every program we want a chance to generate offspring that depends on the fitness of the program. The following criteria will decide over the selection of a program, to provide a decreasing probability with increasing population size and an increasing probability with an increasing fitness:

$$f > \frac{x f_{max}(N_{population} - 2)}{N_{population}}$$

where f is the fitness of the program. f_{max} is the mean of the fitness of the best and second best program. $N_{population}$ is the current population size and x is a random value between 0 and 1. With this method pairs of programs are selected to create a new program by crossing over. The number of pairs depends on the maximal population size. With every selected program a direct offspring will also be generated by mutation.

Starving

If the population exceeds the maximal population size, a starving event will remove part of the programs. Programs with fitness below the average fitness of the population are re-

moved. When the average is updated and the population size is still over the maximum, the step is repeated.

4.4.2 Recombination

The recombination is done in two different ways, the point mutation and the crossing over. Point mutation does slight changes to a single program. Crossing over combines the codes of two programs.

Point mutation

Point mutation will will choose for every line of a program for one of these six actions:

- Let the current line unchanged
- Change the OpCode
- Change a parameter
- Change the order of the next two lines
- Skip the line
- Add a random command in front of this line

For each of those actions exists a certain probability to be chosen. While the first action will have the highest, and all the others a lower. We defined the probability for an unchanged line with $\frac{16}{20}$, and $\frac{4}{20}$ for a changing event.

Crossing over combination

The crossing over combination takes the code of two programs and merges it into a new program. Figure 4.1 tries to sketch the principle used for crossing over. A counter will parallel run through the lines of both programs. At every line a decision if there is a crossing point takes place. This decision depends on a random variable and a changing probability. The probability depends on the number of lines passed since the last crossing point.

In order to keep blocks complete which are semantically connected, a size of a preferred block must be defined. By analyzing programs we defined which size a semantically connected block has, later on called $blocksize$. So, if a part of a program is taken, we want to also take the next $blocksize$ commands with a decreasing probability for a new crossing point until a block of length $blocksize$ is reached. The maximum length that is taken from one program is two times the $blocksize$, meaning that the probability for a crossing point will increase for every new line in the block greater than $blocksize$.

Whenever a crossing point occurs, there is a small probability of inserting a random command at this crossing point.

We estimated the maximal size of a strongly connected block to 24 commands, and chose 12 commands as block size.

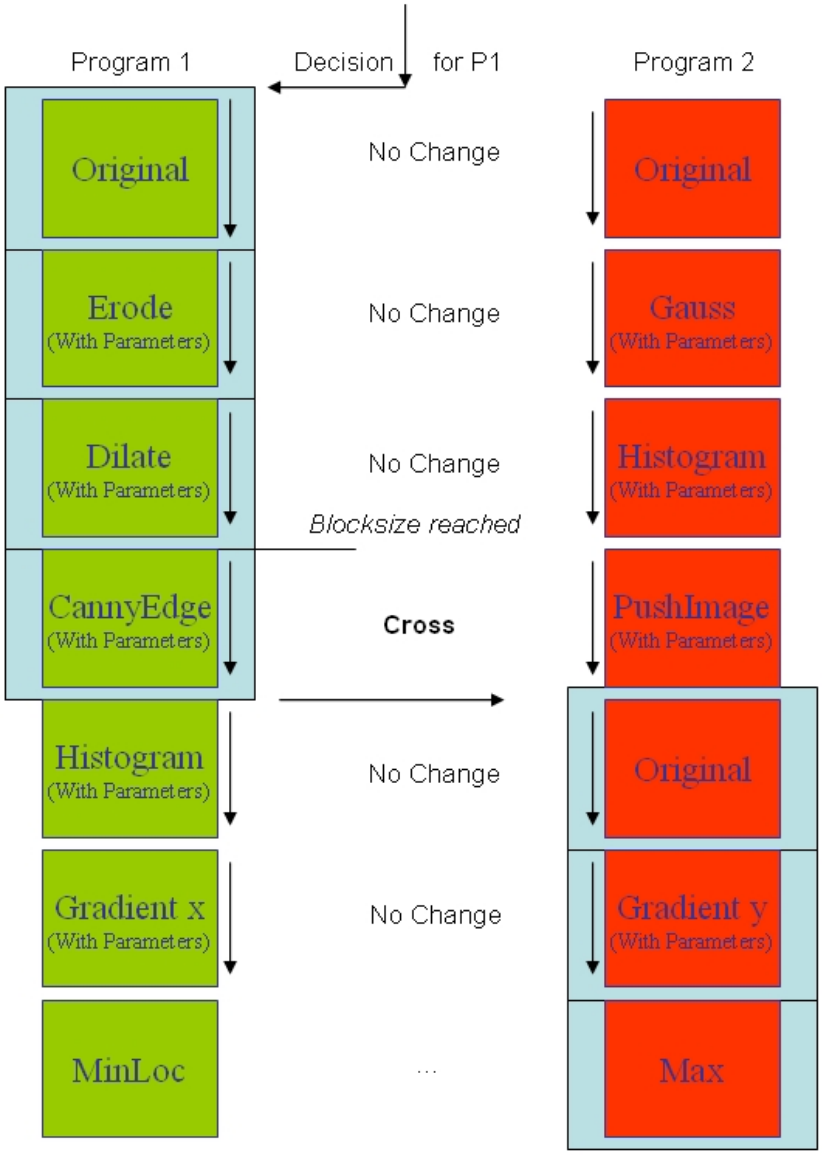


Figure 4.1: The crossing over in the programming language

4.4.3 Remarks

The system has a variable running time depending on the starting population, the problem complexity and the computing power.

Starting population

The first evolution was started including several hand written programs. These programs only consist of a few line of code each, one extracting an x gradient, another extracting the minimum and the maximum from an image, another using the histogram command on an edge map and a last one performing an opening and a closing followed by an extraction of the maximum value. None of theses programs performed well. But the structures used there could be found again in programs generated later on. Normally, those structures were used in new contexts and combined.

Problem complexity

The problem complexity varies for each phase combination. Phases near to each other tend to cause more problems than phases with larger distances. But this is only a tendency and can not be measured by the running time of the system which depends on to many additional random factors. The performance of standard features was a better indicator for the complexity.

Distributed execution of the application

Since we run a large number of tests on each program, this method needs a lot computing power. We used a distributed system to perform the test runs and semantic tests. A server sends programs to several clients that run all necessary test and return messages with the calculated fitness to the server. The server selects the new generation of programs and sends them to the clients again. This allows a reduction of the calculation time nearly by the factor of used computers, since the selection and reproduction was much faster than a single evaluation and had only to be done once for each generation.

4.5 Classification

The final classification consists of two steps. First a binary classification is described based on one feature for every pair of classes. The final classification into one of the phases is performed through an voting system using several improvements.

4.5.1 Binary classifiers

Using the ν-SVC training with a fixed parameter set, for all programs that got assigned a fitness over average of the final population. Here the best of each population is chosen. The

criterion was the average classification rate on four surgeries, with the three others as training data. The test set size here was 512 frames from every tested phase and video, meaning 3072 training images and 1024 test images. The selected classifier was chosen to be one of the final programs.

The parameters of the ν-SVC were optimized for two generated features, and then fixed.

The frame selection of the training frames was similar to the selection in the fitness function. The phase is divided into 512 parts and a frame is randomly chosen from every part.

4.5.2 Voting based decision

The final programs resulting from an evolution were the base for voting. Every final program will be executed over a new image. With the result of the program the image is classified into one of two phases. The classification result is part of a symmetric matrix that contains the boolean decisions between all pairs of phases. Here we can start a simple vote, and choose the phase with the most positive decisions.

Problems

The problem hereby is the high risk of a prediction error. For n classes and $\frac{n+(n-1)}{2}$ decisions, only $n-1$ decisions are related with the correct phase, and $\frac{(n-1)(n-2)}{2}$ are not. If only one decision between the correct class and any other will result incorrect, there is a risk to choose another class, even with a very good result for the correct class.

Improvements

This risk can be lowered by using the average matrix over several frames. Figure 4.2 shows the system we used to improve the stability of the voting decision. From every frame in the video the features selected for a phase decision are extracted again. A 2-class SVM decides the class that is more probable for every feature. These decisions for every phase pair are accumulated for 25 subsequent images. Now the voting is done again, by counting the positive decisions for a phase. Still, with 25 frames as a base, the results contain many outliers . Here we used a median filter over 25 decisions which were calculated as mentioned before. This method requires 2 seconds of a video for every decision.

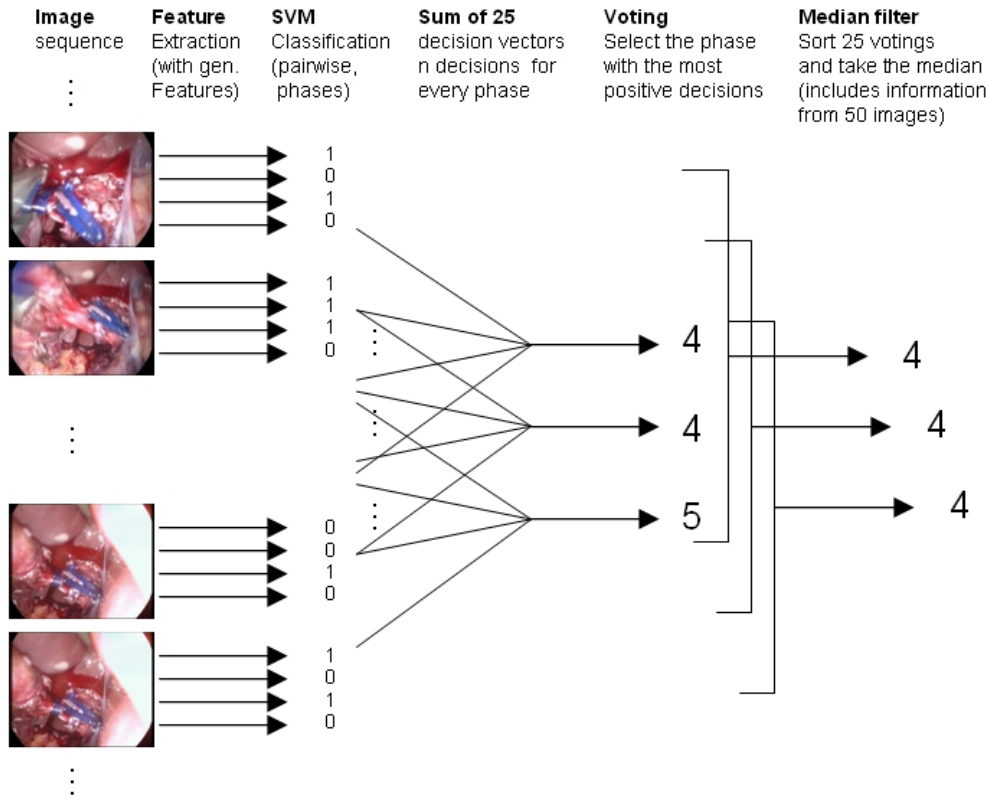


Figure 4.2: The voting based decision process, started from the image to the median filter.

Chapter 5

Results

The method is validated using the following steps:

- statistical results over the Genetic Programming system
- validation of the fitness function
- presentation evolution results
- classification rates of the generated features
- comparison of our features with standard features
- comparison of the multi-class approach with a Neural Network based method

First, we want to give a reasoning for the final selection of the operator set, and also give an impression on how the system works. We want to show and explain the final results in numbers and figures. Finally we compare these results with different features and methods, in order to show how our results compete.

Remarks on the image material

All Results are acquired using a set of videos of cholecystectomies. See Table 5.2, for the number of frames that are in the 6 videos used, to get an overview of the statistical base. Generally, not many videos were available, since it takes great effort to acquire the videos. So, we needed a method that also works on a small training set. We limited the number to a fraction of the available videos for time reasons, since most of the calculations take a lot of time. Also The phases are defined by [Ahmadi, 2003], as mentioned in the introduction.

5.1 Results of the evolution

To show how the Genetic Programming framework operates, the key data of the evolution process will be presented. First by giving an overview of the differences between the low level and high level operator sets, to show why all tests were done with the high level

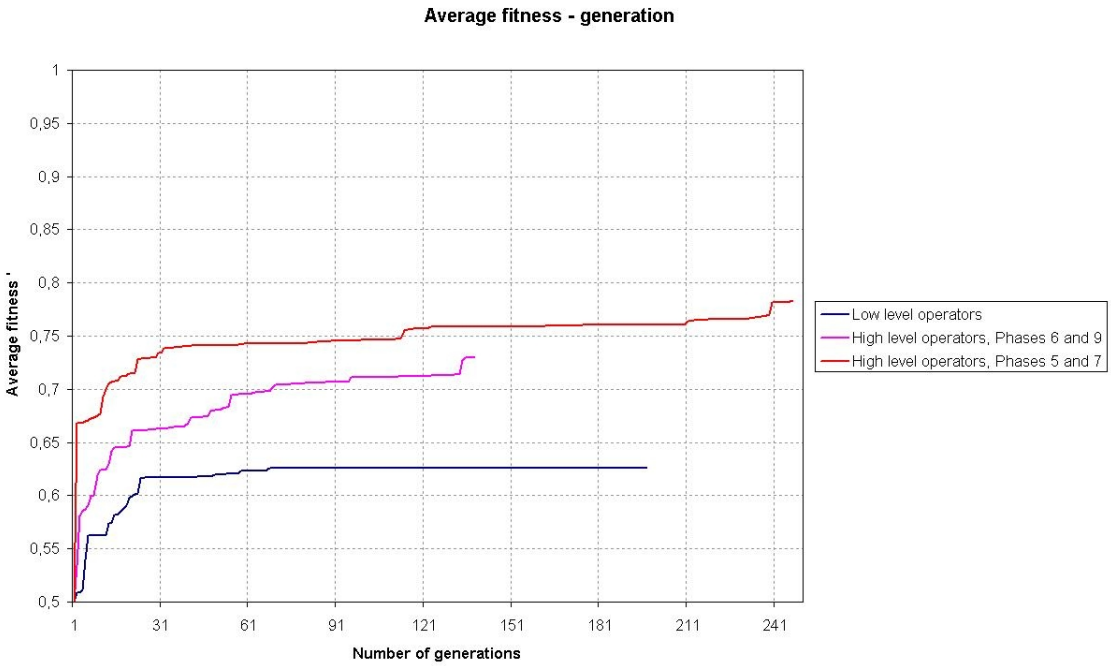


Figure 5.1: The evolution of the average classification error (in y direction) of the population of a generation (in x direction)

operator set. Mainly, there are two reasons for the better performance of the high level operator set: the evolution speed and the error rate. Second, the algorithms of two of the resulting programs will be analyzed.

5.1.1 Comparison of operator sets

The project was started with the idea of generating image features on the level of assembler commands. This idea was implemented with the low level operator set. But due to a lack of performance, increasing the evolution speed was necessary to reach acceptable results in time. So, by reducing generality, we used the high level operator set, where we want to show the gain of performance with the high level operator set. The comparison is based on the fitness evaluation, that is described in section 4.3.

Evolution speed

To show the evolution speed of the high level and low level operator set, the resulting fitness values are plotted relative to the number of generations in Figure 5.1. Since only programs with a fitness higher than the average can be part of the next generation, average fitness will not fall. The number of programs that is accepted is falling faster than the average fitness of the newly generated programs is rising. So the evolution has a decreasing growth. If the population increases over a certain size, an starving event will remove the programs

	High level	Low level
Average execution time	13,04 ms	148,07 ms
Average fitness	0.547	0.508
Statistical base	11 902	9 108

Table 5.1: Average execution time and average fitness of a set of programs from the low level and a set of programs from the high level operator set.

with the lowest fitness. This event can be seen in the graph as a sudden jump of the average fitness.

With the low level operator set, the evolution is slower than for the high level operator set, since the number of senseless combinations, that can not be filtered out by the semantic tests, is higher and so the average fitness of newly generated programs is lower. So, passing the initiation to the population is less probable for a new program.

Error rates

The low level operator set does not contain image related commands, it treats the input as an array of integers, and does not perform action on the array, but only on a single element. Therefore the number of basic operators that is necessary to analyze an image is higher. This also leads to higher evaluation times, since the most costly part of testing is the simulation of the commands.

In Table 5.1 there can be seen the average execution time of low and high level programs. In particular is shows that an average low level program needs more time, even if the operators are less complex. In both cases the statistic contains approximately the first 100 generations of a newly started evolution for the same phase pair.

Figure 5.2 shows the quote of rejected programs, and the different reasons for rejection. While for low level programs the percentage of rejected programs is 89%, most of the high level programs reach a final fitness. While the numbers of time related errors are nearly the same, the memory errors are far less for the high level programs, since there are more possibilities for a high level program to generate a good output without an extensive use of the stack. Semantic checks are less for the high level programs because there are more input referring functions, which increases the probability that one of them occurs in a program.

5.1.2 Program discussion

This section will analyze two of the programs closer. The first is printed in Appendix A.2. The longer programs are very hard to analyze for a human observer, thus we have chosen a short one.

The detailed algorithm that is performed by the program *GEN72-04.HPRG* can be reviewed in Appendix A.2. The following is a short summary of the Appendix. At first the program returns the resulting maximum and minimum gradient strengths of a gradients filtering. Then it adds the histogram of size 64 of an edge map that was extracted from the

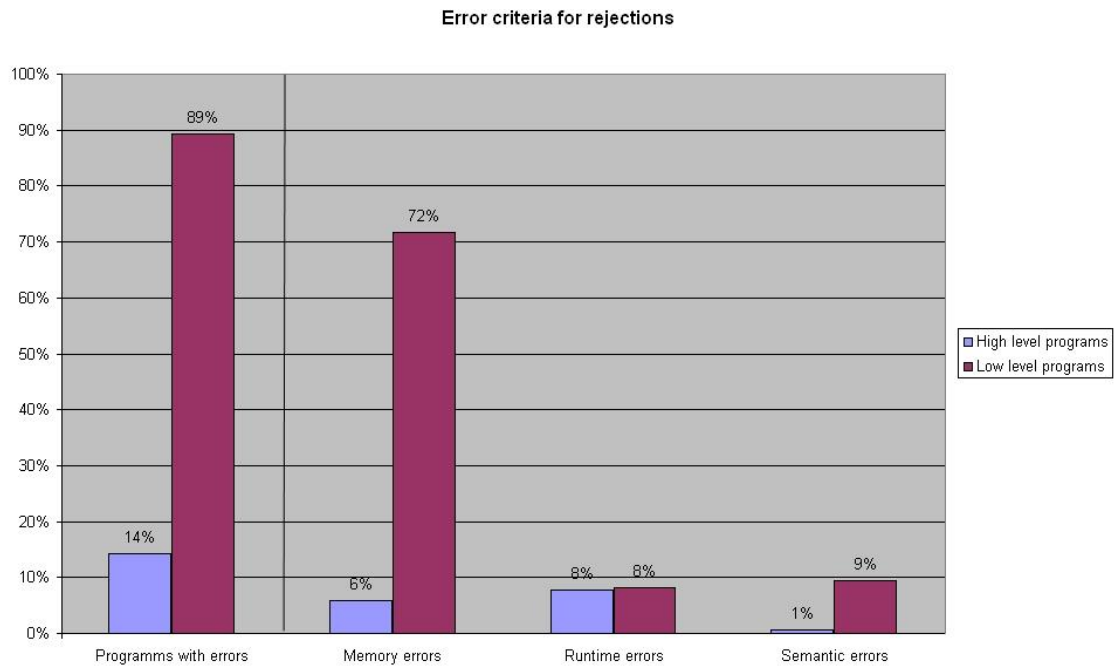


Figure 5.2: The percentage of rejected programs on the left side and the percentage of different reasons for the rejection on the right side, separately shown for the low and the high level operator set.

down sampled gradient images. The edge map was closed before the histogram was extracted by a morphological dilate and an erode operator.

Without a too detailed analysis, we look at the algorithm of the program *GEN6-12.HPRG*. Due to a longer evolution, the resulting structure is further away from the simple starting programs than the program before. The program takes an morphological opened image, subtracts it from the original image and analyzes the resulting difference by maximum extraction and a histogram. The it calculates an edge histogram. Among others these two actions are the main resources that lead to a good classification result.

5.2 Classification results

The result of this work is the features, that were generated with our framework. The classification rates that were reached to distinguish two phases are presented below. These rates will be compared with several image features, that were presented in the theoretical background. We will also show the results for the multi-class decision.

5.2.1 2-Class Classification

Using a statistical overview, the performance of the single programs will be shown. The programs in Table 5.3 are the programs that were performing best in the last generation of the genetic programming framework. The number of generation to reach a good result differs from phase pair to phase pair between 5 and over 1000.

Table 5.4 shows the performance we reached for the programs with the training data from the videos 1 to 4. This was the criteria for the final selection. These 4 videos were taken for the fitness evaluation during the evolution as well. More interesting are the classification rates for the test video, that was not used for evaluation before.

Table 5.5 presents these classification rates for the fifth surgery. As expected, most of the classification rates are falling, some of them down to a useless rate, like 3-4, 3-5, 3-7, 4-7 or 5-7. While other programs performed nearly as well as on the training set, like 2-3, 2-6, 2-7, 3-6, 4-6 or 6-7, and the program 5-6 even performed better on the test than on the training set.

Seeing these numbers, the question comes up, whether the similarity between the videos is strong enough to be detected by only having 4 videos as training data. We are positive about, since for every phase we found another phase that we could distinguish well. Even if we did not find a criteria for every phase pair, that was general enough to also work in the test video, there are enough similarities in the videos. Of course a larger training set will lead to better results, at the cost of running time.

5.2.2 Multi-class Classification

Using the generated features, the multi-class classification is performed. The phase with the most positive decisions in 25 frames is selected. See Figure 5.3, to see the results for the simple voting, using the accumulated decisions over 25 images. All positive decisions for a

Phase	2	3	4	5	6	7	8	9	10	11	12	13
OP1	3 325	4 725	2 125	975	1 625	13 175	2 175	1 475	15 200	1 225	4 975	3 325
OP2	3 200	9 050	2 925	3 075	1 575	22 900	5 550	5 300	15 175	5 075	4 600	825
OP3	825	16 575	750	3 850	1 725	21 975	5 050	2 425	1 600	450	5 600	725
OP4	5 825	12 900	3 600	14 950	2 800	33 925	12 250	8 400	2 325	8 875	4 500	2 425
OP5	5 475	9 150	5 675	2 925	2 525	4 550	4 600	1 325	4 075	325	5 975	4 350
OP6	3 825	8 750	1 950	625	3 275	10 675	3 100	3 225	1 800	300	3 000	625
Sum	18 650	52 400	15 075	25 775	10 250	96 525	29 625	18 925	38 375	15 950	25 650	11 650
Avg	3 730	10 480	3 015	5 155	2 050	19 305	5 925	3 785	7 675	3 190	5 130	2 330

Table 5.2: Number of available frames per phase in the used videos.

Phases	ID	Avg. exec. time	Output length	Prg. length
2-3	GEN73-01.HPRG	72 ms	75	39
2-4	GEN53-02.HPRG	166 ms	79	34
2-5	GEN4-03.HPRG	126 ms	155	30
2-6	GEN72-04.HPRG	126 ms	77	38
2-7	GEN78-05.HPRG	105 ms	388	37
3-4	GEN6-12.HPRG	133 ms	1 091	307
3-5	GEN190-13.HPRG	251 ms	76	50
3-6	GEN14-14.HPRG	60 ms	316	226
3-7	GEN37-15.HPRG	206 ms	32	877
4-5	GEN111-23.HPRG	10 ms	258	1 525
4-6	GEN509-24.HPRG	117 ms	531	263
4-7	GEN533-25.HPRG	62 ms	25	15
5-6	GEN187-34.HPRG	10 ms	257	677
5-7	GEN447-35.HPRG	93 ms	9	20
6-7	GEN207-45.HPRG	245 ms	16	23

Table 5.3: The average execution time of the resulting programs. The output vector length and the program length are in the last column. Only for all pairs of the phases 2 to 7 the program is analyzed.

Phases	class. rate	true neg.	true pos.	num. neg	num pos
2-3	0,831	0,668	0,953	7 167	7 167
2-4	0,907	0,839	0,962	6 893	6 893
2-5	0,936	0,872	0,985	7 118	7 118
2-6	0,903	0,778	0,997	7 167	7 167
2-7	0,851	0,828	0,869	7 167	7 167
3-4	0,666	0,616	0,720	7 918	7 918
3-5	0,705	0,726	0,683	8 143	8 143
3-6	0,752	0,830	0,673	5 120	5 120
3-7	0,726	0,906	0,545	8 192	8 192
4-5	0,724	0,810	0,634	8 942	9 167
4-6	0,626	0,488	0,755	7 918	7 918
4-7	0,766	0,908	0,634	7 918	7 918
5-6	0,593	0,651	0,535	8 143	8 143
5-7	0,720	0,839	0,603	8 143	8 143
6-7	0,825	0,949	0,702	8 192	8 192

Table 5.4: The classification rate of the resulting programs for phases 2 to 7. These are the results on randomly selected frames from the training videos (1-4). The SVM classifiers are trained on three of them(1-3).

Phases	class. rate	true neg.	true pos.	num. neg	num pos
2-3	0,760	0,834	0,686	1 024	1 024
2-4	0,642	0,845	0,438	1 024	1 024
2-5	0,693	0,925	0,461	1 024	1 024
2-6	0,868	0,741	0,994	1 024	1 024
2-7	0,769	0,795	0,742	1 024	1 024
3-4	0,519	0,835	0,203	1 024	1 024
3-5	0,534	0,973	0,095	1 024	1 024
3-6	0,758	0,975	0,543	1 024	1 024
3-7	0,531	0,671	0,391	1 024	1 024
4-5	0,566	0,603	0,530	2 048	2 048
4-6	0,790	0,684	0,896	1 024	1 024
4-7	0,511	0,548	0,474	1 024	1 024
5-6	0,684	0,479	0,889	1 024	1 024
5-7	0,544	0,722	0,367	1 024	1 024
6-7	0,756	1,000	0,512	1 024	1 024

Table 5.5: The classification rate of the resulting programs for phases 2 to 7. These are the results on randomly selected frames of the test video (5). The SVM classifiers are trained on three of them(1-3).

Phase	2	3	4	5	6	7	All (2-7)
Video 5							
Simple Voting	0,197	0,350	0,967	0,897	0,519	0,815	0,558
Median 50	0,183	0,239	0,990	0,852	0,211	0,192	0,489
Video 6							
Simple Voting	0,360	0,548	0,842	0,938	0,995	0,403	0,546
Median	0,360	0,562	0,855	1,000	0,992	0,248	0,506

Table 5.6: The prediction error in case of the voting system. The statistical base are all frames from the phases. See 5.2.

phase in the pairwise classifications were counted. The phase with the highest value for 25 frames was used as a first prediction. The median 25 of these first prediction was used as the final prediction, based on 50 images, or 2 seconds of the video.

The overall correctness in the video of the fifth surgery was 49,9% on the five phases (a random guess would have a 16.7% chance, a fixed guess on the largest phase 32,1%). On the training data a successful prediction of up to 75% could be achieved with the same method. See Table 5.6 for the error per phases, and compare with the yellow dots and the light blue dots in Figure 5.3(b). This figure shows with a thick magenta line the original phase where the images where taken of, the yellow dots are showing the prediction using the voting system with the median filter.

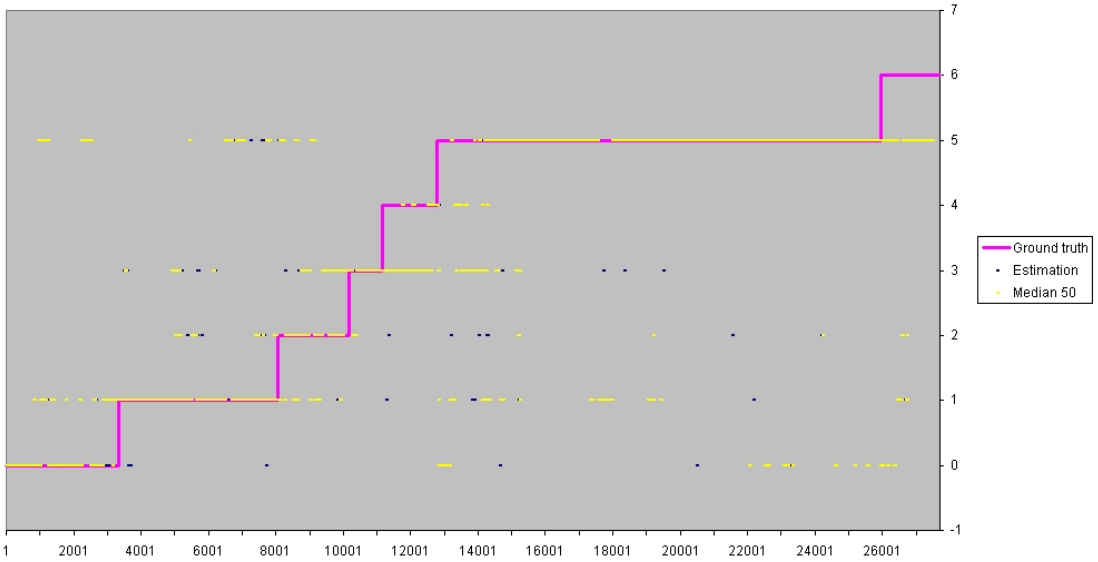
5.3 Comparison

In this section we compare the fitness function, the results of the pair wise phase classification and the multi-class classification with different methods. The fitness function will be qualified by comparing there average results with a standard classifier. The results for the the final programs will be compared with other image features used on endoscopic images. The multi-class classification will be compared with an early approach, a neural network classification on the same data set, but directly, without voting on a reasonable feature set like color, edge and motion components of two images.

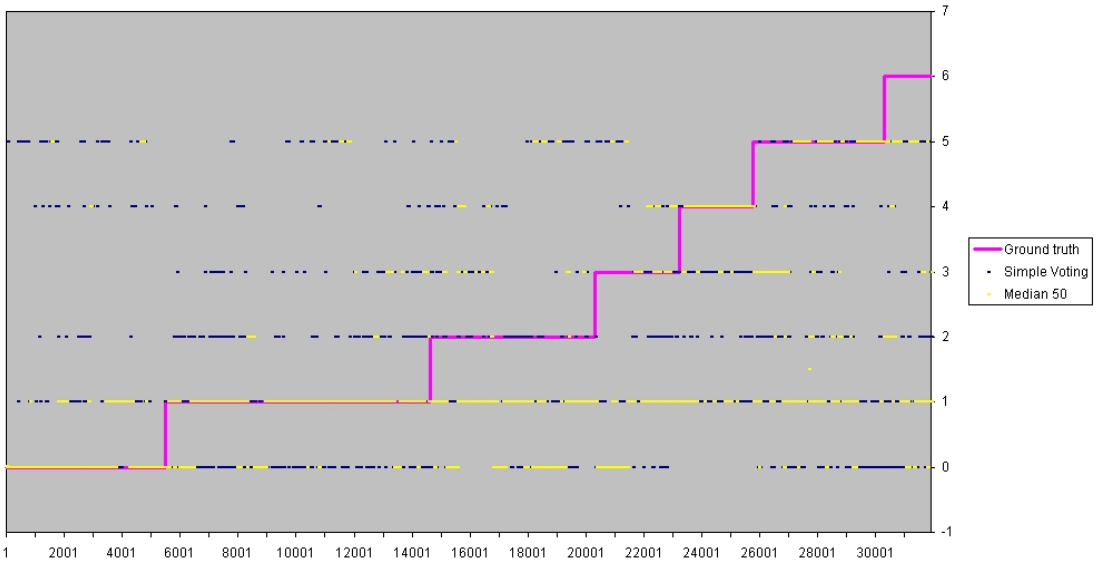
5.3.1 Fitness function and classifiers

To verify the quality of the fitness function, the results of several evaluations are compared with a standard classifier, in this case a SVM with a ν-SVC training algorithm using a RBF kernel. The expected result was that a high fitness value of a program would indicate a high classification rate with a SVM training on the output vectors of the program.

Table 5.7 displays the fitness values and the classification rates of the final programs. For each of these programs the fitness function was executed 30 times, since it depends on random events for selection of the input images. Thus the variance for the resulting fitness of every program is plotted as well. The SVM was trained on 2048 images, from four different surgeries and from two different surgical phases. The test set had the same size, and contained 2048 different images from the same surgeries and the same surgical phases.



(a) The predicted phase for the phases 2 - 7 in a video of the training data.



(b) The predicted phase for the phases 2 - 7 in a test video.

Figure 5.3: Compare the results for the stateless prediction of the current phase using 25 images (blue) and 50(yellow)

Phases	Average fitness	std. deviation	Op5 class rate
2-3	0,801	0,012	0,760
2-4	0,839	0,033	0,642
2-5	0,932	0,014	0,693
2-6	0,932	0,008	0,868
2-7	0,757	0,025	0,769
3-4	0,572	0,035	0,519
3-5	0,640	0,018	0,534
3-6	0,734	0,012	0,759
3-7	0,662	0,071	0,531
4-5	0,667	0,027	0,566
4-6	0,728	0,019	0,790
4-7	0,742	0,029	0,511
5-6	0,664	0,012	0,684
5-7	0,712	0,025	0,544
6-7	0,813	0,017	0,756

Table 5.7: The average fitness with the standard deviation, in comparison with the classification rate for the test video

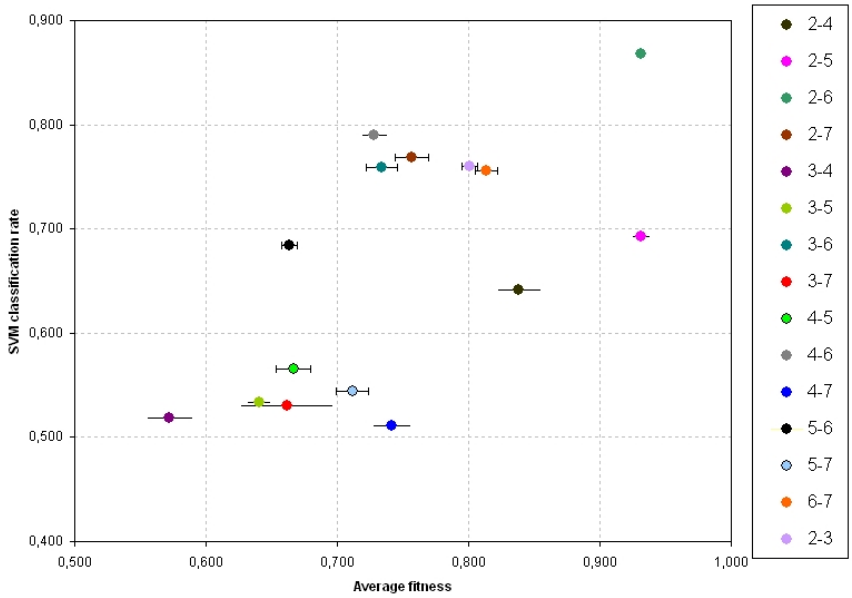
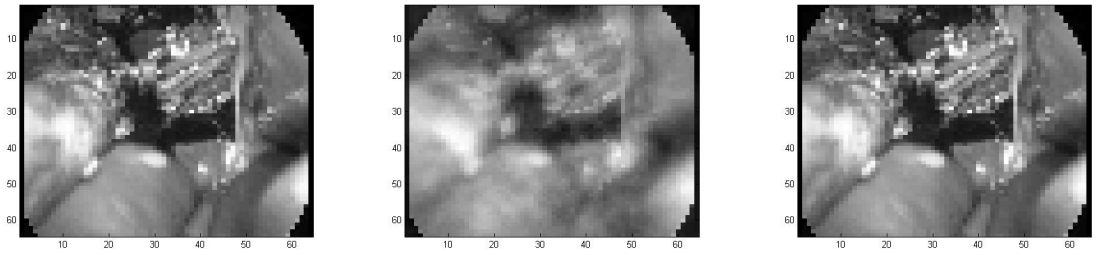


Figure 5.4: Comparison between the average fitness of a program in 30 runs of the fitness function with the standard deviation in the error bars , and a result for a SVM classification

Data	PCA	3-6
Video 1-4	0,889	0,752
Video 5	0,500	0,758

Table 5.8: The classification rate for the PCA based classification for the phases 3 and 6, compared with the corresponding feature 3-6.



(a) Original endoscopic image.

(b) Reconstruction using 100 eigenvectors, 100 is the average output length of the generated programs.

(c) Using 200 eigenvectors.

Figure 5.5: Reconstruction error using different numbers of eigenvectors (image size 64x64, first color channel).

In Figure 5.4 a weak correlation can be seen. So the upper left and the lower right quadrant are empty, and program with a relative high fitness has a relative high classification rate. And for low fitness the classification rate was relatively low.

5.3.2 Program and PCA

On the data a simple PCA based approach of [Keil *et al.* , 2006] was tested to compare it with the resulting programs. The 100 most relevant eigenvectors were used, decomposed from 2000 images, down scaled to a 64x64-resolution. Plotting the cumulative sum of the eigenvalues, the resulting curve represent the percentage of the data that can be reconstructed using the corresponding eigenvectors. The reconstruction of an image using this technique is visualized in Figure 5.5. The first reconstruction shows an image reconstructed using only 100 Eigenvectors and the second a reconstruction using 200. We transformed the training images from two phases into this eigenvector space, and build a cluster for every phase. Taking the centroid for 2 phases, the Mahalanobis distance is a measurement, whether an image is closer to the first or the second phase.

Table 5.8 presents the classification results for the PCA. While we reached a classification rate up to 88.9% on test images from the same surgeries as the training images, we did not reach classification rate over 50,0% for another surgery. This is probably caused by the common features of the phases not being locally invariant. Even if the images seem similar, a slight shift of the relevant features in the images from one video to another removes the similarity in the PCA space.

Test domain	HSV Color Histogram	To compare: 2-7	Stat. base
Video 1-4	0,753	0,903	17 212
Phase 2	0,480	0,778	8 191
Phase 7	1,000	0,997	9 021
Video 4	0,650	0,778	4 096
Phase 2	0,300	0,561	2 048
Phase 7	1,000	0,994	2 048
Video 5	0,614	0,868	4 096
Class 2	0,229	0,741	2 048
Class 7	1,000	0,994	2 048
Avg. exec. time	239 ms	105 ms	21 308

Table 5.9: The HSV Color Histogram and the generated feature for the phase combinations 2 - 7.

5.3.3 Program and standard image features

We compared some programs with several standard image features. The first test was with a HSV-color histogram. The second test was done with Water Filling features. See in section 3.3.2. The third test was done with a combination of HSV-color histogram, a Gradient Histogram and a OpticalFlow map.

HSV-color histogram

The HSV-color histogram was chosen, because some phases can be recognized due to colors. For example in the clipping phase, the surgeon uses colored clips. Using a color histogram, the presence of such clips should be easy detectable. The expected problems are the color variation between the clips in different videos and the images were no clips are in view. For performing this test, we used two elements that were not available as operator in our Genetic Programming framework: the color transformation to the HSV space and a color normalization for the image. We performed the test for three different phase pairs. First, an early phase without clips against a later phase were all clips are set, and coagulation is performed or with number phase 2 against 7.

See the classification results in Table 5.9. Analyzing these results, it reveals that the HSV color histogram performs worse than the generated feature for this phase pair, but still relatively good. Taking the more homogenous images from phase 4 instead of phase 2, the classification rates are even better than the rates from the comparable generated feature. This is in Table 5.10 in detail.

For the other tested phase pairs, there was no classification at all, the training error were about 50 percent and there was no generalization possible with the classifiers. These tested phase pairs were 2-3 and 4-5.

Test domain	HSV Color Histogram	To compare: 4-7	Stat. base
Video 1-5	0,770	0,754	19 182
Phase 4	0,980	0,903	8 942
Phase 7	0,588	0,625	10 240
Video 4	0,732	0,753	4 096
Phase 4	0,992	0,983	2 048
Phase 7	0,473	0,523	2 048
Video 5	0,689	0,653	4 096
Phase 4	0,929	0,818	2 048
Phase 7	0,450	0,487	2 048
Avg. exec. time	239 ms	192 ms	19 182

Table 5.10: The classification results HSV Color Histogram and the generated feature for the phase combinations 4 - 7.

Test domain	LBP	To compare: 3-6	Stat base
Video 1-4	0,541	0,785	17 212
Video 4	0,536	0,707	4 096
Phase 3	0,890	0,799	2 048
Phase 6	0,182	0,614	2 048
Video 5	0,516	0,759	4 096
Phase 3	0,995	0,975	2 048
Phase 5 cl 6	0,036	0,543	2 048
Avg. exec. time	372 ms	60 ms	21 308

Table 5.11: The classification rates for the LBP Feature in comparison with the generated feature for the phase combinations 3-6.

Local Binary Pattern

We also started some tests with the local binary pattern (LBP) histogram. On the first sight texture feature shows interesting. The texture of the tissue in the endoscopic view changes during the surgery, therefore it seemed possible to distinguish the phases using a texture feature.

Table 5.11 shows the result for the phase pair 3 and 6, it demonstrates the poor results we got for the LBP feature. Similar result we got for other phase pairs, 2-6, 4-5, 2-4 and 2-5. This shows, that this texture feature is not a very discriminant features. Perhaps, another texture feature that works can be found for some phases, but searching for it is not as interesting as for other types of feature. Therefore, it can be seen as correct, that by analyzing the generated features no calculation similar to a texture analysis could be found.

Water-Filling Feature

The most interesting feature we use for comparison is the water filling (WF) feature, which was already presented in 2.2.2. The statistics it measure about the shape structure of the image could for example distinguish

Test domain	WFFeature	To compare: 3-6	Stat base
Video 1-4	0,752	0,785	17 212
Video 4	0,770	0,707	4 096
Phase 3	0,820	0,799	2 048
Phase 6	0,720	0,614	2 048
Video 5	0,955	0,759	4 096
Phase 3	0,995	0,975	2 048
Phase 5 cl 6	0,915	0,543	2 048
Avg. exec. time	3s542 ms	60 ms	21 308

Table 5.12: The classification rates for the WF Feature in comparison with the generated feature for the phase combinations 3-6.

Test domain	WFFeature	To compare: 4-5	Stat. base
Video 1-4	0,494	0,724	17 212
Video 4	0,500	0,627	4 096
Phase 3	1,000	0,903	2 048
Phase 6	0,000	0,351	2 048
Video 5	0,500	0,566	4 096
Phase 3	1,000	0,603	2 048
Phase 6	0,500	0,530	2 048
Avg. exec. time	5s542 ms	10 ms	21 308

Table 5.13: The classification rates for the WF feature in comparison with the generated feature for the phase combinations 4-5.

Test error	Training error
0,705	0,445
0,644	0,355
0,625	0,298
0,593	0,249
0,580	0,237
0,644	0,116
0,735	0,062

Table 5.14: Excerpts of the test and training error evolution of a NN, to demonstrate the danger of overfitting

Phase	2	3	4	5	6	7	All (2-7)
NN with combined features	0,366	0,000	0,994	1,000	1,000	0,121	0,580

Table 5.15: Prediction error with a neural network

The execution times of the WF feature was higher than the execution time allowed by the VM. But the execution time can not be seen as an absolute value, since there are better ways to execute it, than in our virtual machine. This shows the complexity of WF feature compared to the generated program.

Table 5.12 presents are the classification rate reached with the water filling feature. It performed very well for the class pair 3 and 6. It performed better than the generated feature, especially with a higher grade of generalization to the test videos. For the phase pair 4-5 we got no results, like for the HSV feature, see Table 5.13.

Combined basic features

As a last test, we combined several features and fed them as input to a neural network. The desired output was one of the phases between 2 - 8. The features used were an HSV-color histogram with 512 bins, a gradient histogram with 64 bins and a motion map with 16 entries. The gradient histogram counted the accumulated strength of the gradients in a certain direction.

We used a resolution of the direction of 64. The motion was calculated using 2 images, parting the two in 16 subimages and giving the direction and strength of the motion in a subimage as a value to the neural network. We used a feed forward network with a back-propagation learning algorithm, with .

As training data served those three features extracted from four videos, 512 frames per phase , and over all the phases 2 to 7. For the same phases of a fifth video, we extracted the features as before and used them as testing images.

The Table 5.14 shows the generalization of different learning states of the neural network. The random guess would be around 0,84 error, the decreasing training error leads to a decreased testing error, this shows that the generalization works, and overfitting is not to strong. But the data from the different video do not seem really comparable, so the resulting error differs strongly. Decreasing the training error too much, will result in an overfitting,

as can be seen in the last steps of the lines, where the test error increases again. The neural network finally stopped at a level oscillating around the final result and a 10 percent worse error value.

See Table 5.15 for the phase wise results in training and testing data of this final training state. The classification rate is around 55

5.4 Summary and possible improvements

The results show that an identification of a surgical phase from an endoscopic image with this method is possible up to a chance of about 50% in contrast to a chance of a random pick of 17% and a maximal class size of 30%.

For a 2-class problem the rates range up to 86%, but they can also go down to approximately 50%. For any standard feature we tested, we found several generated features that performed better for a phase pair. In special cases, when the standard image feature nearly worked perfect, we could only reach a comparable result.

Generally, by investing more time in the selection of operators and in running more evolutions with a larger number of generation or using more training videos, it should be possible to improve the results considerably. The results can also be improved by increasing the number of evaluated images for the fitness function can improve the results, in order to reduce the variance of the fitness for a program.

Generally, all these possibilities show that the work can be useful for the search for the optimal feature for a classification task.

Chapter 6

Conclusion

After a manual search for features suitable for the phase recognition, the idea of feature generation was self-evident. Inspired by several sources, Genetic Programming was the method we chose, due to personal interest into this area. The creation of a framework for Genetic Programming was the first time-consuming task of this work. It included the definition of a programming language fulfilling all requirements, as well as implementing the virtual machine for this programming language. Then the fitness criteria was defined and the evolution started. At last the features were evaluated and compared and finally integrated into a multi-class decision system.

The results show that the identification of a surgical phase from an endoscopic image is possible. The Genetic Programming created several interesting features which are useful for this specific task. These features did not outperform all basic features in any phase pair, because of the limits that are set on execution time and memory usage for a generated feature. These limits were imposed by the major problem within this approach: the high calculation time of the evolution. The calculation time varies due to randomness and the complexity of the problem. Still, the search for a feature solving a special classification problem can be almost completely automated using the approach proposed and tested in this work. With enough computing power, better results could be achieved.

The information that is gained with the proposed method could provide additional input for the recognition of the surgical workflow. This is possible since this method uses only image sequences and no further information to gain abstract workflow information.

Part III

Appendix

Appendix A

Programs

A.1 Water filling feature

Here a sample of a manual implemented program. It is implemented in the Programming language defined in 4. All jump and call targets are labeled with a name. These names are converted in addresses before passing it to the Virtual Machine.

```
#WaterFilling
READINPUT
EDGES 64 64
EXTRACTMAX 1
PUSH 9
MULTIPLY
PUSH 10
DIVIDE
SAVE 3 -1
LOAD 3
THRESHOLD -1 255
PUSH -1
#NEWSTARTING: NOP
EXTRACTMAX 1
PUSH 255
EQUALS
IF #&FINISH
EXTRACTLOCMAX 1
SAVE 2 -1
SAVE 1 -1
PUSH 0
LOAD 1
LOAD 2
SETPIXEL -1 -1 1
LOAD 1
LOAD 2
```

```
#BACK: CALL #WATERFILL
LOAD 13
ADD 11 -1
LOAD 13
SAVE 13 1
PUSH 1
MINUS
ADD 12 -1
DUPLICATE
PUSH -1
EQUALS
NOT
IF #&STARTINGPOINTFINISHED
JUMP #BACK
#&STARTINGPOINTFINISHED: NOP
CALL #WFHISTS
LOAD 9
LOAD 14
GREATER
IF #&MAXFILLLINGTIME
LOAD 9
SAVE 14 -1
LOAD 12
SAVE 15 -1
#&MAXFILLLINGTIME: NOP
LOAD 11
LOAD 16
GREATER
IF #&MAXWATERAMOUNT
LOAD 11
SAVE 16 -1
LOAD 9
SAVE 20 -1
LOAD 12
SAVE 21 -1
#&MAXWATERAMOUNT: NOP
LOAD 12
LOAD 18
GREATER
IF #&MAXFRKCOUNT
LOAD 12
SAVE 18 -1
LOAD 9
SAVE 19 -1
#&MAXFRKCOUNT: NOP
SAVE 9 0
SAVE 10 0
```

```
JUMP #NEWSTARTING
#&FINISH: NOP
POP
LOAD 14
LOAD 15
LOAD 18
LOAD 19
LOAD 16
LOAD 20
LOAD 21
FOR 0 14 #LFCTHS
DUPLICATE
SAVE 1 -1
PUSH 22
PLUS
LOAD -1
DUPLICATE
SAVE 2 -1
LOAD 1
PUSH 36
PLUS
LOAD -1
LOAD 2
DIVIDE
#LFCTHS: NOP
JUMP #LAST
#WATERFILL: NOP
ADD 9 1
SAVE 2 -1
SAVE 1 -1
#ONEUP: LOAD 1
PUSH 1
PLUS
LOAD 2
CALL #SUBSCRIPT1
#ONEDOWN: LOAD 1
PUSH 1
MINUS
LOAD 2
CALL #SUBSCRIPT2
#ONERIGHT: LOAD 1
LOAD 2
PUSH 1
PLUS
CALL #SUBSCRIPT3
#ONEDOWN: LOAD 1
LOAD 2
```

```
PUSH 1
PLUS
CALL #SUBSCRIPT4
RETURN
#SUBSCRIPT1: NOP
#SUBSCRIPT2: NOP
#SUBSCRIPT3: NOP
#SUBSCRIPT4: NOP
SAVE 2 -1
SAVE 1 -1
LOAD 1
PUSH -1
GREATER
IF #&RETURNSUBSCRIPT1
LOAD 1
INPUTDIMENSION 0
SMALLER
IF #&RETURNSUBSCRIPT2
LOAD 2
PUSH -1
GREATER
IF #&RETURNSUBSCRIPT3
LOAD 2
INPUTDIMENSION 1
SMALLER
IF #&RETURNSUBSCRIPT4
LOAD 1
LOAD 2
READPIXEL -1 -1 1
PUSH 255
EQUALS
IF #&RETURNSUBSCRIPT5
PUSH 0
LOAD 1
LOAD 2
SETPIXEL -1 -1 1
LOAD 1
LOAD 2
ADD 13 1
#&RETURNSUBSCRIPT1: NOP
#&RETURNSUBSCRIPT2: NOP
#&RETURNSUBSCRIPT3: NOP
#&RETURNSUBSCRIPT4: NOP
#&RETURNSUBSCRIPT5: NOP
RETURN
#WFHISTS: NOP
LOAD 9
```

```
PUSH 60
GREATER
IF #&FTH1
ADD 28 1
LOAD 12
ADD 41 -1
JUMP #FTH2
#&FTH1: NOP
LOAD 9
PUSH 10
DIVIDE
DUPLICATE
PUSH 22
PLUS
ADD -1 1
PUSH 36
PLUS
LOAD 12
ADD -1 -1
#FTH2: NOP
LOAD 12
PUSH 6000
GREATER
IF #&FCH1
ADD 35 1
LOAD 9
ADD 49 -1
JUMP #FCH2
#&FCH1: NOP
LOAD 9
PUSH 1000
DIVIDE
DUPLICATE
PUSH 29
PLUS
ADD -1 1
PUSH 43
PLUS
LOAD 9
ADD -1 -1
#FCH2: NOP
RETURN
#LAST: NOP
#FINAL: NOP
#END
```

A.2 A generated program, Gen74-04

```
#ProgramCode
#Length:
38
0 0 0 0
1 8454 10747 -1    #READINPUT restore original image
0 0 0 0
0 0 0 0
1 0 0 0             #READINPUT restore original image
11 4 1 0            #DILATE
12 4 1 0            #ERODE
11 4 1 0            #DILATE
12 4 1 0            #ERODE
1 -1 -1 -1          # READINPUT restore original image
18 -1 9882 21463    #FOR 0 -  9882 , blocksize: 21463
2 1 0 0             #GRADIENT
7 0 0 0             #EXTRACTMAX
7 0 0 0             #EXTRACTMAX
8 0 0 0             #EXTRACTMIN
0 0 0 0
8 0 0 0             #EXTRACTMIN
2 0 1 0             #GRADIENT
7 0 0 0             #EXTRACTMAX
7 0 0 0             #EXTRACTMAX
8 0 0 0             #EXTRACTMIN
8 0 0 0             #EXTRACTMIN
2 -1 -1 2609        #GRADIENT reading 2 values from the stack
0 0 0 0
6 3 0 0             #DOWNSCALE
4 100 100 0         #EDGE
5 9 9 0             #GAUSS
11 4 1 0            #DILATE
12 4 1 0            #ERODE
3 64 0 0            #HISTOGRAMM 64
7 0 0 0             #EXTRACTMAX
7 0 0 0             #EXTRACTMAX
8 0 0 0             #EXTRACTMIN
8 0 0 0             #EXTRACTMIN
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
#END
```

Detailed algorithm

Line 1-12: the program starts with reading the input and some NOP operations. The first actions are in line 7 to 10, here two times a closing is performed. The results are deleted by another READINPUT result, and a for loop with a to long inner block is set. This preamble of 12 commands has not much effect on the result, only an -1 will be pushed on the stack by the for command. See A.1(a) for the original sample image.

Line 13: the next command is a gradient filter in X direction. See A.1(b) for gradient image that results.

Line 14-15: the first interaction with stack is performed via a maximum pixel value extraction. In the sample we got a 166 as result, that is pushed on top of the stack two times.

Line 15-18: the same for the minimum gradient two times. The result for the sample was -162.

Line 19: now a gradient of the gradient image is calculated in y direction. See A.1(c) for this image.

Line 20-23: again the maximum and the minimum value of this image are searched and are pushed two times on the stack. Stack size now is 9, and the minimum was -394 and the maximum 272.

Line 24-25: again a gradient is extracted. This time again in y direction, see A.1(d).

Line 26: a scaling is performed to a third of the size. Here is no difference in the image, A.1(e), but from now on the resolution of the images is only a third of before.

Line 27: the next command is a canny edge detection the parameters 100 and 100 are used as threshold in x and y direction. See A.1(f) for the resulting image.

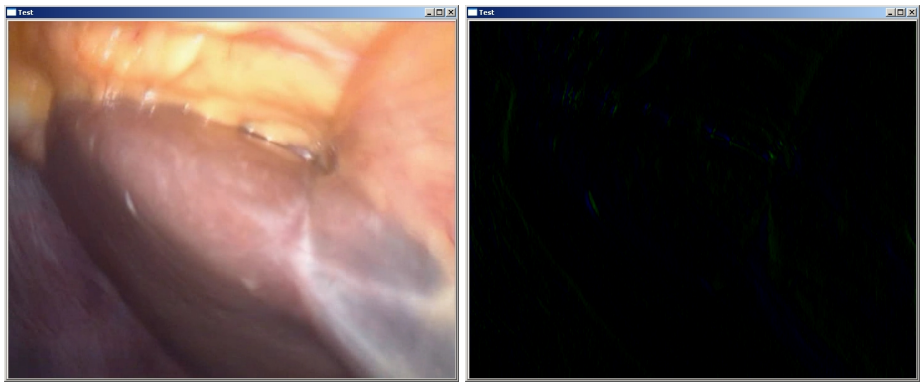
Line 28: the next command is gauss filter with a kernel size of 9 times 9. See A.1(g) for the image.

Line 29-30: a closing is performed. A.1(h)

Line 31: over this gray level image a histogram is calculated with 64 bins.

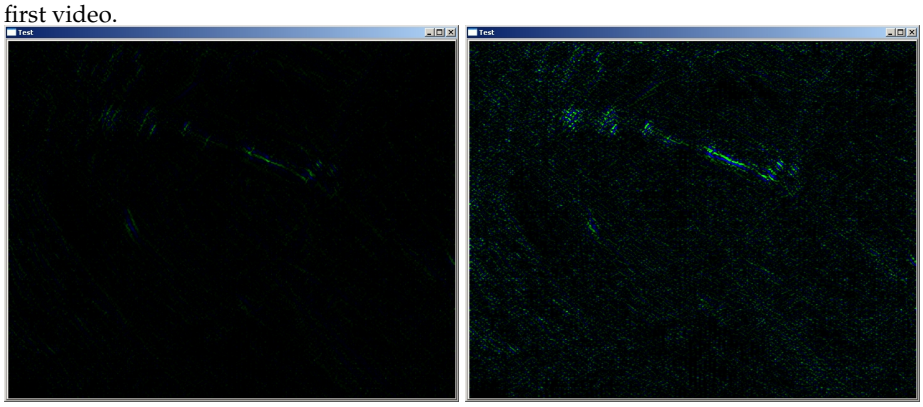
Line 32-36: again two times the minimum and maximum. are extracted two times. These are 0 and 111.

Line 37-38: two NOPs and the end of the program, the resulting output hat the size 77



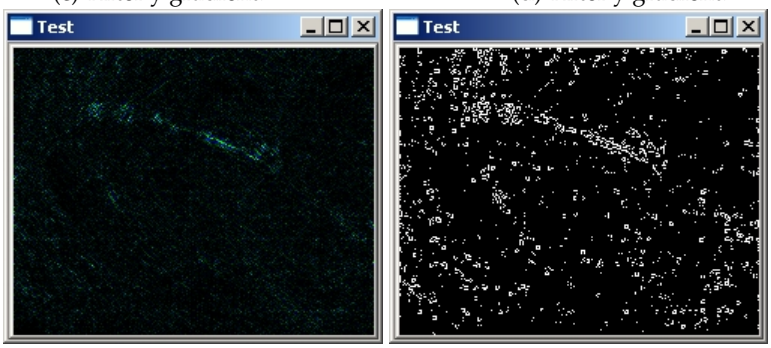
(a) An image from the second phase in the first video.

(b) After x gradient.



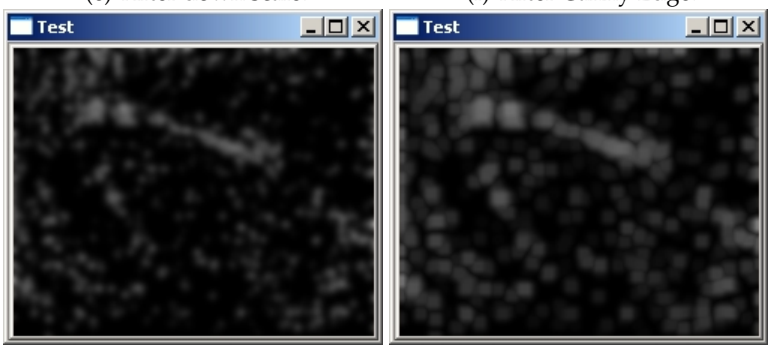
(c) After y gradient.

(d) After y gradient.



(e) After down scale.

(f) After Canny Edge.



(g) After Gauss.

(h) After closing.

Figure A.1: The image sequence represents the intermediate states during an execution of the Program here.

Bibliography

[Ahmadi, 2003] Ahmadi, Seyed-Ahmad. 2003. *Automatic workflow retrieval in surgery*. TU München.

[Ahmadi *et al.* , 2006] Ahmadi, Seyed-Ahmad, Sielhorst, Tobias, Stauder, Ralf, Horn, Martin, Feussner, Hubertus, & Navab, Nassir. 2006. Recovery of surgical workflow without explicit models. *In: MICCAI '06*.

[Allwein *et al.* , 2000] Allwein, Erin L., Shapire, Robert E., & Singer, Yoram. 2000. Reducing Multiclass to Binary: A Unified Approach for Margin Classification. vol. 1.

[Banzhaf *et al.* , 1998] Banzhaf, W., Nordin, P., Keller, R. E., & Francone, F. D. 1998. *Genetic Programming: An Introduction*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Bishop, n.d.] Bishop, Christopher M. *Pattern Recognition and Machine Learning*.

[Burges, 1998] Burges, Christopher J.C. 1998. A Tutorial on Support Vector Machines for Pattern Recognition. *In: Zeitschrift Data Mining and Knowledge Discovery*, vol. 2. Verlag Springer Netherlands.

[Darwin, 1859] Darwin, Charles. 1859. *The Origin of Species*.

[Dietterich & Bakiri, 1996] Dietterich, Thomas G., & Bakiri, Ghulum. 1996. Solving Multiclass Learning Problems via Error-Correcting Output Codes. vol. 2.

[Farmer *et al.* , 2004] Farmer, Michael E., Bapna, Shweta, & Jain, Anil K. 2004. Large Scale Feature Selection Using Modified Random Mutation Hill Climbing. IEEE.

[Forgy, 1965] Forgy, E. 1965. Cluster analysis of multivariate data: Efficiency vs. interpretability of classifications.

[Ghosh & Mitchell, 2006] Ghosh, Payel, & Mitchell, Melanie. 2006. Segmentation of Medical Images Using a Genetic Algorithm. *Pages 1171–1178 of: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM Press New York, NY, USA.

[Horn & Schunck, 1980] Horn, Berthold K.P., & Schunck, Brian G. 1980. Determining Optical Flow. *In: AIM-572*. Massachusetts Institute of Technology Cambridge, MA, USA.

[Hsu & Lin, 2000] Hsu, Chih-Wei, & Lin, Chih-Jen. 2000. A Comparison of Methods for Multiclass Support Vector Machines. vol. 13. IEEE.

[Iakovidis *et al.* , 2005] Iakovidis, Maroulis, Karkanis, & Brokos. 2005. A Comparative Study of Texture Features for the Discrimination of Gastric Polyps in Endoscopic Video. *Pages 575–580 of: Proceedings. 18th IEEE Symposium on Computer-Based Medical Systems, 2005.* IEEE.

[Jain & Zongker, 1997] Jain, Anil, & Zongker, Douglas. 1997. Feature Selection: Evaluation, Application, and Small Sample Performance. vol. 19. IEEE.

[Johnson, 2002] Johnson, Colin G. 2002. Deriving Genetic Programming Fitness Properties by Static Analyse. *Page 298 of: Genetic Programming: 5th European Conference, EuroGP 2002, Kinsale, Ireland, April 3-5, 2002. Proceedings*. Springer Berlin / Heidelberg.

[Josh C. Bongard, 2003] Josh C. Bongard, Rolf Pfeifer. 2003. Evolving Complete Agents using Artificial Ontogeny.

[Karkanis *et al.* , 1991] Karkanis, S., Galousi, K., & Maroulis, D. 1991. Classification of Endoscopic Images Based on Texture Spectrum.

[Karkanis *et al.* , 2003] Karkanis, Stavros A., Iakovidis, Dimitris K., Maroulis, Dimitris E., Karras, Dimitris A., & Tzivras, M. 2003. Computer-Aided Tumor Detection in Endoscopic Video Using Color Wavelet Features. *In: IEEE TRANSACTIONS ON INFORMATION TECHNOLOGY IN BIOMEDICINE, VOL. 7, NO. 3, SEPTEMBER 2003 141*. IEEE.

[Keil *et al.* , 2006] Keil, Andreas, Wachinger, Christian, Brinker, Gerhard, Thesen, Stefan, & Navab, Nassir. 2006. Patient Position Detection for SAR Optimization in Magnetic Resonance Imaging. *Pages 49–57 of:* Larsen, Rasmus, Nielsen, Mads, & Sporring, Jon (eds), *Proc. Int'l Conf. Medical Image Computing and Computer Assisted Intervention (MICCAI)*. Lecture Notes in Computer Science, vol. 4191. Springer.

[Kleene, 1952] Kleene, Stephen. 1952. *Introduction to Metamathematics*. Walters-Noordhoff & North-Holland, with corrections (6th imprint 1971); Tenth impression 1991.

[Koza, 1990] Koza, John R. 1990. Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.

[Kudo & Matsumoto, 2001] Kudo, Taku, & Matsumoto, Yuji. 2001. Chunking with Support Vector Machines.

[Lenze, 1997] Lenze, Burkhard. 1997. *Einführung in die Mathematik neuronaler Netze*.

[Li *et al.* , 2003] Li, Shutao, Kwok, James T., Zhua, Hailong, & Wang, Yaonan. 2003. Texture classification using the support vector machines. *Pages 2883–2893 of: Pattern Recognition, Volume 36, Issue 12*. Copyright l' 2003 Pattern Recognition Society. Published by Elsevier Science B.V.

[Lin *et al.* , 2005] Lin, Henry C., Shafran, Izhak, Murphy, Todd E., Okamura, Allison M., Yuh4, David D., & Hager, Gregory D. 2005. Automatic Detection and Segmentation of Robot-Assisted Surgical Motions. *In: Proceedings of MICCAI 2005*. Springer.

[Lo *et al.* , 2003] Lo, Benny P.L., Darzi, Ara, & Yang, Guang-Zhong. 2003. Episode Classification for the Analysis of Tissue/Instrument Interaction with Multiple Visual Cues. *In: Medical Image Computing and Computer-Assisted Intervention - MICCAI*. Springer.

[Ma & Zhang, 1998] Ma, Wei-Ying, & Zhang, Hong-Jiang. 1998. Benchmarking of image features for content-based retrieval. *Pages 253–257 of: Signals, Systems and Computers, 1998. Conference Record of the Thirty-Second Asilomar Conference*. IEEE.

[Majewski & Jedruch, 2005] Majewski, Pawel, & Jedruch, Wojciech. 2005. Endoscopy Images Classification with Kernel Based Learning Algorithms. *Pages 400–405 of: Innovations in Applied Artificial Intelligence*. Springer.

[McConaghy & Gielen, 2006] McConaghy, Trent, & Gielen, Georges. 2006. Canonical form functions as a simple means for genetic programming to evolve human-interpretable functions. *Pages 855–862 of: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM Press.

[Motsinger *et al.* , 2006] Motsinger, Alison A., Hahn, Lance W., Ryckman, Kelli K., & Ritchie, Marylyn D. 2006. Alternative cross-over strategies and selection techniques for grammatical evolution optimized neural networks. *Pages 947–948 of: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM Press.

[Nandi *et al.* , 2006] Nandi, R. J., Nandi, A. K., Rangayyan, R. M., & Scutt, D. 2006. Classification of breast masses in mammograms using genetic programming and feature selection. *Pages 683–694 of: Journal Medical and Biological Engineering and Computing*. Springer Berlin / Heidelberg.

[Padoy *et al.* , 2007] Padoy, N., Horn, M., Feussner, H., Berger, M.O., & Navab, N. 2007. Recovery of Surgical Workflow: a Model-based Approach. *In: Proceedings of Computer Assisted Radiology and Surgery (CARS 2007) 21st International Congress and Exhibition, Berlin, German*.

[Perkins *et al.* , 2000] Perkins, S., Theiler, J., Brumby, S. P., Harvey, N. R., Porter, R. B., Szymanski, J. J., & Bloch, J. J. 2000. GENIE - A Hybrid Genetic Algorithm for Feature Classification in Multi-Spectral Images. *In: SPIE4120*.

[Platt, 1999] Platt, John C. 1999. *Fast training of support vector machines using sequential minimal optimization*. MIT Press Cambridge, MA, USA.

[Rifkin & Klautau, 2004] Rifkin, Ryan, & Klautau, Aldebaro. 2004. In Defense of One-Vs-All Classification. vol. 5. IEEE.

[Schölkopf *et al.* , 2000] Schölkopf, Bernhard, Smola, Alex J., Williamson, Robert C., & Bartlett, Peter L. 2000. A New Support Vector Algorithms. *In: Neural Computation 2000 12*. The MIT Press.

[Sielhorst *et al.* , 2006] Sielhorst, Tobias, Bichlmeier, Christoph, Heining, Sandro, & Navab, Nassir. 2006. Depth perception a major issue in medical AR: Evaluation study by twenty surgeons. *In: Proceedings of MICCAI 2006*. LNCS. Copenhagen, Denmark: Springer, for MICCAI Society.

[Szymanski *et al.* , 2002] Szymanski, John J., Brumby, Steven P., Pope, Paul, Eads, Damian, Esch-Mosher, Diana, Galassi, Mark, Harvey, Neal R., McCulloch, Hersey D.W., Perkins, Simon J., Porter, Reid, Theiler, James, Young, A. Cody, Bloch, Jeffrey J., & David, Nancy. 2002. Feature Extraction from Multiple Data Sources Using Genetic Programming. *In: SPIE4725*.

[Thomason & Soule, 2006] Thomason, Russell, & Soule, Terence. 2006. Redundant genes and the evolution of robustness. *Pages 959–960 of: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. ACM Press.

[Tjoa & Krishnan, 2003] Tjoa, Marta P, & Krishnan, Shankar M. 2003. Feature extraction for the analysis of colon status from the endoscopic images. *In: BioMedical Engineering OnLine*. BioMedical Engineering OnLine.

[Turk & Pentland, 1991] Turk, M.A., & Pentland, A.P. 1991. Face recognition using eigenfaces. *Pages 586–591 of: Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*.

[Vailaya *et al.* , 2001] Vailaya, A., Figueiredo, M.A.T., Jain, A.K., & Zhang, Hong-Jiang. 2001. Content-based Hierarchical Classification of Vacation Images. *Pages 117–130 of: Image Processing, IEEE Transactions*. IEEE.

[Wang & Soule, 2004] Wang, Gang, & Soule, Terence. 2004. How to Choose Appropriate Function Sets for Gentic Programming.

[Wang *et al.* , 2001] Wang, P, Krishnan, S M, Kugean, C, & Tjoa, M P. 2001. Classification of endoscopic images based on texture and neural network. *In: EBMS, 2001*. IEEE.

[Zhang *et al.* , 2003] Zhang, Mengjie, Ciesielski, Victor B., & Andreae, Peter. 2003. A Domain-IndependentWindow Approach to Multiclass Object Detection Using Genetic Programming. *Pages 841–859 of: EURASIP Journal on Applied Signal Processing 2003:8*. 2003 Hindawi Publishing Corporation.

List of Figures