

Classification of Images using Color, CBIR Distance Measures and Genetic Programming

An evolutionary Experiment

Stian Edvardsen

Master of Science in Informatics

Submission date: June 2006

Supervisor: Herindrasana Ramampiaro, IDI

Abstract

In this thesis a novel approach to image classification is presented. The thesis explores the use of color feature vectors and CBIR – retrieval methods in combination with Genetic Programming to achieve a classification system able to build classes based on training sets, and determine if an image is a part of a specific class or not.

A test bench has been built, with methods for extracting color features, both segmented and whole, from images. CBIR distance-algorithms have been implemented, and the algorithms used are *histogram Euclidian distance*, *histogram intersection distance* and *histogram quadratic distance*. The genetic program consists of a function set for adjusting weights which corresponds to the extracted feature vectors. Fitness of the individual genomes is measured by using the CBIR distance algorithms, seeking to minimize the distance between the individual images in the training set. A classification routine is proposed, utilizing the feature vectors from the image in question, and weights generated in the genetic program in order to determine if the image belongs to the trained class.

A test-set of images is used to determine the accuracy of the method. The results shows that it is possible to classify images using this method, but that it requires further exploration to make it capable of good results.

Forord

Denne oppgaven er min avsluttende oppgave på masterstudiet i Informatikk på NTNU.

Veien har vært lang og svært ujevn, men med mange oppturer. Først takker jeg for en forståelsesfull og trivelig veileder. Jeg retter også en takk til alle som i lengre tid har tatt seg tid til å høre meg rable i vei om genetiske fortreffeligheter, alle som har klappet meg på skulderen når jeg satt fast og en unnskyldning til alle eventuelle som ble et utløp for frustrasjoner.

Først og fremst vil jeg takke samboeren min, som har holdt ut med en heller fraværende type nå i sluttspurten. Nå blir det ferie Iris!

Stian Edvardsen

Trondheim, 15. juni, 2006

1 INTRODUCTION	5
1.1 MOTIVATION FOR THIS THESIS	5
2 THEORY	7
2.1 INFORMATION RETRIEVAL	7
2.1.1 <i>Information Retrieval and Data Retrieval</i>	7
2.1.2 <i>Queries</i>	7
2.1.3 <i>Indexing</i>	8
2.1.4 <i>Evaluating</i>	9
2.2 CONTENT BASED IMAGE RETRIEVAL	11
2.2.1 <i>Color</i>	12
2.2.1.1 Color indexing and retrieval	13
2.2.1.2 Color Space.....	15
2.2.1.3 Color Space conversions.....	17
2.2.2 <i>Shape</i>	19
2.2.3 <i>Texture</i>	19
2.2.4 <i>Three common retrieval methods</i>	20
2.2.4.1 Euclidian distance.....	20
2.2.4.2 Histogram intersection distance.....	20
2.2.4.3 Quadratic distance.....	20
2.2.5 <i>Queries</i>	21
2.2.5.1 By example.....	21
2.2.5.2 By low level features	21
2.2.5.3 By textual IR (classification of images).....	22
2.2.6 <i>State of the art: CBIR</i>	22
2.2.6.1 Blobworld.....	22
2.2.6.2 QBIC.....	22
2.2.6.3 VisualSEEK.....	23
2.2.6.4 WebSEEK.....	24
2.3 IMAGE CLASSIFICATION AND IMAGE ANNOTATION	24
2.3.1 <i>State of the art: Supervised image classification</i>	25
2.4 EVOLUTIONARY ALGORITHMS.....	26
2.4.1 <i>Natural Evolution</i>	26
2.4.2 <i>Genes</i>	28
2.4.3 <i>Evolution and variation</i>	28
2.4.4 <i>Genetic Algorithms</i>	30
2.4.4.1 GA operators – from biology to computers.....	31
2.4.5 <i>Genetic Programming</i>	33
2.4.6 <i>Difference between GP and GA operators</i>	35
2.4.6.1 Crossover.....	35
2.4.6.2 Mutation.....	35
2.4.7 <i>Application and limitation of Genetic Programming</i>	36
2.4.8 <i>Genetic Programming and image classification</i>	36
3 EVALUATION METHODS OF THIS THESIS.....	37
4 CLASSIFICATION SYSTEM.....	37
4.1 INTRODUCTION	37
4.2 THE IDEA	37
4.3 THE CLASSES	38
4.4 FEATURE EXTRACTION.....	40
4.5 DISTANCE ALGORITHMS	41
4.6 GENETIC PROGRAMMING.....	43
4.6.1 <i>Terminal set</i>	43
4.6.2 <i>Function set</i>	43
4.6.3 <i>Fitness function</i>	43
4.6.4 <i>Parameters for controlling the run</i>	45
4.6.5 <i>Criterion for terminating a run</i>	45
4.6.6 <i>Termination report</i>	45
4.6.7 <i>Final report</i>	46
4.6.8 <i>Selection</i>	46

4.7 CLASSIFICATION	47
5 EVALUATION	48
5.1 EVALUATION OF RETRIEVAL METHODS	48
5.2 INITIAL EVALUATION OF THE GP.....	55
5.3 MODIFICATION OF THE GP FUNCTION SET.....	58
5.4 CLASSIFICATION	59
5.4.1 Classification tryout and evaluation	59
6 CONCLUSION	63
6.1 EVALUATION AND DISCUSSION OF THE CLASSIFICATION SYSTEM.....	63
6.2 EVALUATION OF THE WORK-PROCESS	64
REFERENCES	65
APPENDIX A – SOURCE CODE.....	68
IMPLEMENTATION NOTES.....	68
OPERATOR CLASS	87
DATABASE CLASS	108
GENERATOR CLASS.....	114
GENETIC PROGRAM	127
EXAMPLES OF EVOLVED GENETIC PROGRAM.....	144
<i>Evolved genome before modification</i>	144
<i>Evolved genome after modification</i>	149

1 Introduction

The introduction of the Internet with its massive amount of information has led to a high focus on the science of information retrieval as a mean to organize, store, representing and accessing these information items. Textual based information retrieval has been thoroughly researched and implemented with great success. As hardware has improved and available bandwidth has grown, the use of multimedia objects such as sound clips, movies and images has become more and more common. A branch of information retrieval, called Multimedia Information Retrieval (Multimedia IR) deals with the oddities of indexing and retrieving multimedia information.

This paper will focus on images, an information object widely used in the internet, although not as easily retrieved as textual information. Retrieval of images is, for the great majority of search engines, done by collecting data around the image, such as the image file-name, html tags and surrounding text. This leaves the actual image more or less ignored. The field which studies the retrieval process based on the content of an image, is called Content Based Image Retrieval (CBIR). CBIR uses methods that analyze the actual bits and pieces, that is, color, texture, shape and various other features of an image. There have been many different approaches on how to deal with the subjects such as feature extraction, indexing and the retrieval process. One approach is to make an attempt to *classify* the image into a more textual described context. With the image classified, it can be retrieved using more traditional and better understood retrieval methods. There have been different suggestions on how to do this, most based on statistical methods. The writer's hypothesis is that there must be some way to achieve classification using CBIR methods otherwise used as retrieval tools. This paper's approach is to try and utilize Genetic Programming for this task. Genetic programming is inspired by Darwin's evolutionary theories, which means it's a self-modifying code that evolves around the "survival of the fittest" principle.

Using CBIR methods, more specifically image retrieval algorithms, in combination with Genetic Programming this paper will try to explore an alternative way of classifying images into semantic, textual meaning without the use of surrounding information.

The goal for this thesis is not a fully functional classification system, but to explore the possibilities of this thesis approach.

1.1 Motivation for this thesis

There were several reasons for me to choose this assignment. Evolutionary methods are

something i never worked with, but I find it a highly interesting subject, and it was something i'd like to know more about. Content-based IR is a fairly new science which means there are more unexplored areas to research, perhaps in contrast to traditional IR, which is well explored and harder to find possible improvements in. Combining genetic algorithms with CBIR distance methods, in order to classify images, three areas which I'm not very well traversed, seemed to me like a big challenge, one I was not certain I could overcome. Therefore it became the natural choice.

2 Theory

This paper deals with content based information retrieval methods and genetic algorithms. A walk-through of the different theories and methods is necessary in order to understand the background for this paper's assumptions.

2.1 Information Retrieval

Information Retrieval (IR) is a term used for almost all aspects of the task of making information more available to the user. Information can loosely be defined as some sort of organized data, which is considered useful or meaningful to someone¹. Information Retrieval deals with the representation, storage, organization of, and access to information items [26, p 1]. Information items can be anything including text files, video clips, audio and images. Traditional IR deals mostly with text. Textual IR is mostly what will be described in this chapter.

2.1.1 Information Retrieval and Data Retrieval

It is important to understand that here is a difference between retrieving information and retrieving data. To retrieve data means to retrieve all object which satisfy a defined set of conditions. Retrieving information is an attempt to gather the data which the information system deems *relevant* for the user *based* on a users query. This means that the data may not satisfy the conditions in the query fully, but is considered "close enough" to be given as a result. Considering what data is "close enough" the query to be presented as a result, is more or less the essence of the problems developing an information retrieval system.

Data retrieval systems, such as a database system, are founded on mathematical relations theories and are operated by query languages such as SQL. Operations in such as systems can be SELECT FROM, JOIN, etc., and will return results with 100% accuracy. Since IR-systems are based on the "close enough" principal, results will inevitably be less accurate. Where data retrieval systems are based on theory, IR-systems are based on *heuristics*. Heuristics can be defined as "*a rule of thumb, based on trial and error*"[unknown], meaning "*in similar circumstances, this method has worked to some success, so that is what we'll use*". A shorter definition of the term may be a *qualified guess*.

2.1.2 Queries

A query is the input the search is based upon. Queries in textual IR often consist of a word(*term*) or a collection of terms (*single word queries*). It is also often possible to create *context*

1 Another, less credited definition, is that information is a measure of how surprising something is

queries, which means searching for words close to other words. A *phrase* is a collection of words which is considered to be in sequence. The documents retrieved are documents which contains this phrase. Unimportant words such as “in”, “the”, “at”, etc are normally not considered in such a search. Another example of a context query is a *proximity*-query. This is a more relaxed version of the phrase query, and its measured value depends on the closeness of each of the words.

Other queries are *boolean queries*, which uses typical boolean operators such as *and*, *or*, and *not*. Ranking is often not necessary with Boolean queries, since the documents are considered relevant or not relevant. A more relaxed version of the Boolean query is the use of *fuzzy Boolean* operators, which is an attempt to make the operators (i.e *and*, *or*, *etc*) not absolute, and therefore possible to rank. *Natural language*-queries treats each query and document as a vector of term weights. The documents retrieved are the ones which are close to the query.

2.1.3 Indexing

Although it is possible, and in fact often used in a combination with indexing, searching all the text of a document collection can be very time consuming. Full text searches are most effective when done on a limited collection, and not on a large one, such as the internet. Most commercial search engines are capable of searching through several billion sites, and are expected to do so in a very short period of time. A full text search through all these sites would take forever, even with the heavy computer equipment such companies have available. The solution is to *index* every site they include in their search. To index is to “boil down” the document to its “essence”. This seems rather vague, but that is because the definition of the essence is different in most search engines. Even though most incorporate classic techniques, which will be described, many of them have their own idea of what the best essence (or index) consist of. Perhaps the most successful company incorporating an original way of index their documents (based on the Ph.D-thesis of the founders) is Google.

Google was founded by Larry Page and Sergey Brin, two Ph.D students at Stanford University. They had the idea that if (web) documents were analyzed based on their relationship to each other, a search engine would be able to produce better results than a search based on comparing the query to each document independently. In a highly simplified sense, the more pages that link to a specific site, the more relevant information it contains. The success of Google speaks of the correctness of this theory[28].

A more classic way of indexing text files is the use of *inverted files*, which consist of two things; vocabulary and occurrences of the words in the vocabulary. For each instance of a word in

the vocabulary, an occurrence is added to the inverted file. This occurrence can point to a specific location in the text, point to a document, or more common a combination of both.

2.1.4 Evaluating

In order to evaluate an IR-system, there are several measures used to determine the quality of the system. There are different aspects of an IR-system, all of which contribute to the cumulated quality of the entire system. The different aspects are typically *processing quality*, that is, the time and space efficiency of the system, *search quality*, which is the effectiveness of results and an overall *system quality*, which tries to measure the satisfaction of the user. In this paper I will in large degree disregard processing quality and system quality and focus on search quality measures of which there are several different measures. All of these involve counting relevant documents in some way or the other, so first we will define a *document* in this paper's sense of the term as "Any source of information, in material form, capable of being used for reference or study or as an authority". [27]. *Relevance* can be defined as a measure of how well the result meets the need of the user that issued the query.

Of these, perhaps the most important two are *Precision* and *Recall*. The figure below shows the relationship between recall, precision, relevant documents (the red line) and the collection (the grey circle).

Precision is the fraction of the relevant documents which has been retrieved:

$$Precision = | \text{Relevant retrieved} | / | \text{All retrieved documents} |$$

Using the figure, this translates to

$$Precision = A / (A+B)$$

Recall is the fraction of the relevant documents which has been retrieved:

$$Recall = | \text{Relevant retrieved} | / | \text{All relevant documents} |$$

$$Recall = A / (A+D)$$

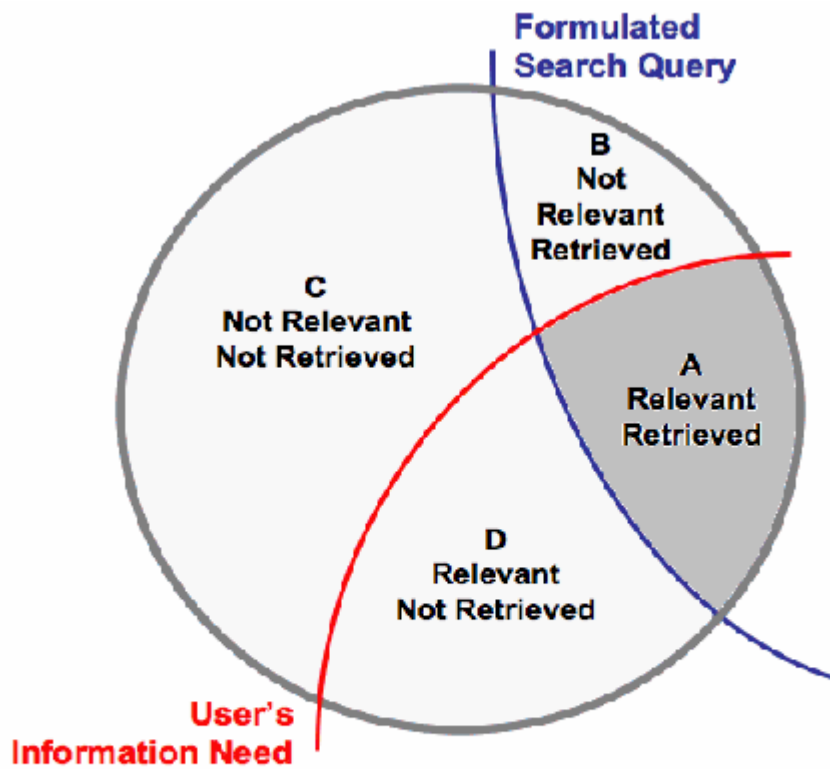


Fig 1: Query / User's need chart

A standard evaluation strategy for IR-systems is the use of *average precision versus recall* figures.

These figures enables us to evaluate quantitatively both the quality of the overall answer set and the breadth of the retrieval algorithm. Further, they are simple, intuitive, and can be combined in a single curve.[26, p 75-82]. Below (fig 2) is an example of a precision vs recall figure. In order to calculate the precision/recall rate as accurate as possible, it is common to run a series of tests with different queries and generate precision/recall figures for all of these. Finally we calculate the

average with the formula $\bar{P}(r) = \sum_{i=1}^{N_q} \frac{P_i(r)}{N_q}$ where $\bar{P}(r)$ is the average precision at the recall level r ,

N_q is the number of queries used and $P_i(r)$ is the precision at recall level r for the i -th query .[26, p 77]

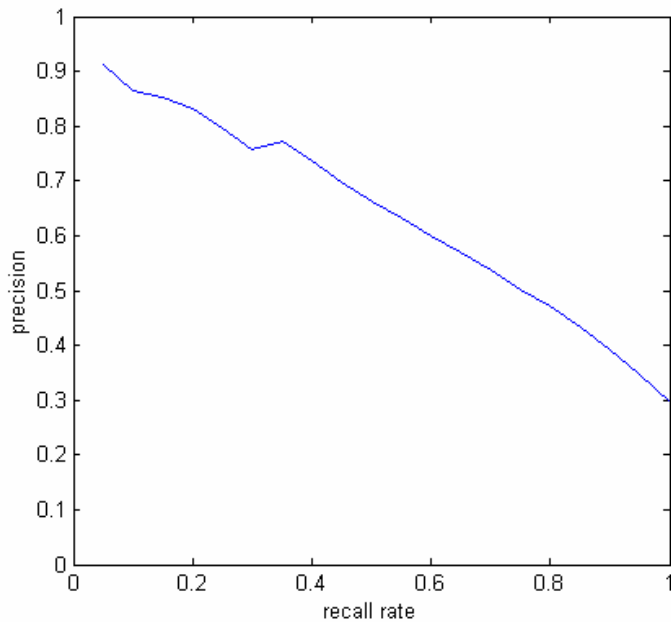


Fig 2: Precision versus recall rate figure

One of the strengths of this kind of diagrams is that they are an easy way of comparing different retrieval algorithms, which will be useful later in this paper. There are alternative ways of measuring retrieval effectiveness, such as finding the *harmonic mean F* of recall and precision, and the *E-measure*, which let you decide what is of most interest; recall or precision. I will not explore these further since they will not be used in this paper.

2.2 Content Based Image Retrieval

A fairly new branch in information retrieval is *content based image retrieval* (CBIR). CBIR is, as the name implies, the science of how we can index and retrieve images based on its contents. The reason for this being a new science is probably due to increased hardware performance, which had made it possible to use images on a much larger scale than before. An image consists of different materials than textual documents. Textual documents consists of terms, phrases and, with some extensions; hyperlinks.

There are mainly four different ways of searching images. The first one is searching for images based on attributes such as file name, date of creation and manually added categories, such as subject and creator. This makes can be made into a relational model and searches are made based on these attributes. The problem with this method is that auto-generated information about the images may not be enough to describe the content, and adding categories manually is a very time

consuming task.

The second approach is to use automated object recognition, which is often used in specific domains such as medical images. Automated object recognition is time-consuming and very difficult to achieve in a general system.

The third approach is to use free text to annotate the images, and use textual IR techniques to retrieve them. This approach can be divided in two; manual annotation and automatic annotation. With manual annotation, specialists annotate appropriate text to each image. The problem here is, of course, in a massive image collection, this will be an enormous task, and is still prone to subjectivity and incompleteness. Automatic image annotation can be divided in two groups; partitioned or hierarchical, where the latter decide if an image is either one of two categories. This base category can have two sub-categories, which the image can be further divided into. The image will finally have a “tree” of categories which can be used in a search. The first approach has flat structure where the image gets annotated with different subjects all in one go. The annotation method basically says “this image contains trees, sky and a giraffe”. Most popular search engines such as Yahoo and Google use an altered mix between method 1 and 3. They use things as file names, size and dates, but also analyses the HTML page the image is in. They take advantage of such things as the anchor-tag, and surrounding text for properly index the images.

The fourth approach is perhaps what most people think of when they talk about CBIR. This approach use the most fundamental elements an image consists of. When we talk about what an image consists of, we refer to this as *features*. When we talk about *low-level features*, we are talking about the lowest, the most basic features of an image. Images can typically be divided in three different low-level features: Color, shape and texture.

2.2.1 Color

Color is perhaps the feature in an image which is the easiest to comprehend for newbies in CBIR. Each pixel in a digital image consist of a color element. In a grey-scale image this color element typically range from 0 to 255, where 0 is black, 255 is white, and the values between is the different shades of grey from black to white. In a color image, let’s say with 24 bits *color resolution*, (which means that 24 bits are used for color information for each pixel), an (often even) piece of the 24 bits is assigned to each of the three color components in the image. The most common *color space* used is RGB. Color Space is defined as a model for representing color in terms of intensity values[4] RGB stands for Red-Green-Blue, and in this example the 24 bit color image, 8 bits is used to represent each of the components. In this example the color components makes it possible to represent $(2^8)^3$ different colors, which is more than the human eye can

differentiate from. Even though RGB is the most common color space used, it is not always the best choice when working with CBIR. Later, when we deal with color space and histograms, other and perhaps more appropriate color spaces will be introduced.

2.2.1.1 Color indexing and retrieval

The typical way of indexing color is the use of *color histograms*. A color histogram is a table with different color values as the index key, and the sum of the occurrences of these colors as the value. Lets say we have an 2-bit grey-scale image with 8 pixels. With two bits representing color, the image has 4 possible colors to choose from. Now let us say the pixel values are as following:

0-1-1-1-0-3-2-2

The histogram of this image will look like this:

Pixel-value	0	1	2	3
Occurrences	2	3	2	1

Table 1: Histogram example

In order to make histograms quicker to process, it is also common to use *bins*, where a certain range of pixel values are stored. For instance, in an 8-bit grey scale image (with 256 different shades of grey), we can divide the histogram in 16 bins. The 256 different pixels values are then spread accordingly over the histogram bin. On grey scale pictures this may not be an important task, but with color images with 256^3 different color values, using bins can make a substantial improvement on performance.

Now, the purpose of any indexing technique is to ease the retrieval process, making it more effective and hopefully more accurate. How can we use this histogram in order to retrieve other, similar images? The most basic technique is to subtract the query histogram color-for-color with each of the histograms in the image collection. The sum of each color in the result of the subtraction is then computed. The histogram which the calculated sum is least is deemed to be most relevant.

This technique is not perfect. Let's say we have two images:



Fig 3, Example of “identical” images

These two images have two more or less identical histograms, and will therefore be considered as identical images, which is obviously wrong. The same goes with images which are the same, but with a color shift in one of the images (i.e one of the images is darker than the other). Even though these two images may be identical to the human eye, they will have completely different histograms.

There are several ways of increase the accuracy of retrieval. For starters we can pre-process the image and histograms so the chance of almost identical images will appear almost identical when indexed. One standard thing to do, is to equalize the histograms. To equalize the histograms means to change the histogram to a histogram that is constant for all brightness values. This is not a perfect technique, but will in most cases lead to a certain improvement. Another way of pre-process histograms is to build a *perceptually weighted histogram* (PWH). According to [1], when building a PWH, *the 10 perceptually most similar colors are found for each pixel. The distance between the pixel and the 10 representative colors are calculated. Then weights inversely proportional to the color distance are assigned to these 10 representative colors.* This method has shown better performance than the basic retrieval method.

Other pre-processing methods can be dividing the image into several pieces and build a histogram of each of the pieces. This part-histograms can be compared to the corresponding part-histogram in the collection-images. This helps with problem of fig. 3, since the spatial relationship between the pixels is considered. Another problem is that histograms tend to favour background color. The objects in the foreground are often the objects the user is interested in, but these objects can often be lost in the background data. One way of dealing with this is separating foreground and background and create histograms for each of them.

There is also possible to improve the basic retrieval method by taking into account the

similarity among colors. This is more or less the same as PWH, only at the other end, but since the histogram-bins already are made, some information is lost. In short, this method returns a smaller distance if similar (closer) colors are different, then the difference between non-similar colors.

2.2.1.2 Color Space

Color space typically defines a one-to four-dimensional space. In the case of the grey scale image, the color space will have one dimension, a dimension that defines the level of “greyness”. In a color image, there is typically three dimensions, one for each of the primary color component. In certain cases a fourth dimension is used, which is referred to as the *alpha-channel*, which is a mask that indicates how the pixel’s colors should be merged with another pixel when two images are overlaid, one on top of the other. In other cases the fourth dimension is a white reference point.

We have already described the RGB color model as the most commonly used. According to [5, p 290-294] the reason for its widespread use is because its ideally suited for hardware implementations, and because the human eye is strongly perceptive to red, green and blue components.

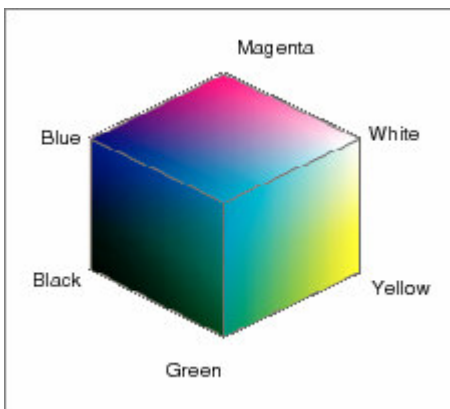


Fig 4: RGB color model

There are several other color models, for instance the CIE XYZ color model which is a model which uses a white reference point and combines it with the *chromaticities* of each primary color. This leads to a color model which is device independent, but hard to implement, and not particularly intuitive.

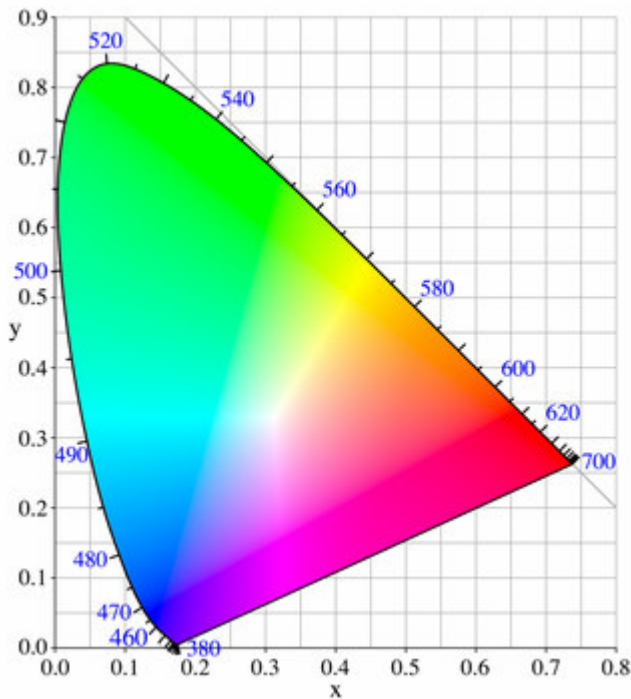


Fig 5: CIE XYZ color model

As we can see in figure 1, the green component is at some distance from the reference white compared to blue and red (reference white is typically at $(x = 0,31, y = 0,32)$). Two other color spaces has derived from the XYZ color space is the CIE L^*u^*v and CIE L^*a^*b , which are so called uniform color spaces. In uniform color spaces, equal distances approximately represent equal perceived color differences.

One more color space model I want to mention, is the *HSV* (also called *HSB* and *HSI*) *color model*. HSV stands for Hue, Saturation and Value. Some call it HSI, which is the same model, only the *Value* is replaced by *Intensity*. In HSB, the B is for Brightness. The HSV color space is typically represented as a cone, shown on figure 5. The Hue indicates what color it is, and is normally in a range of 0-360 (a circle). Some users of the model, normalize this to 0%-100%. The Saturation tells how saturated, that is, how rich the color is. The less saturated a color is the more grey and bland it will look. Saturation range from 0-100%. The Intensity indicates how bright the color is. The biggest strength of the HSV model, is that it is intuitive, it is easy to describe a color using this model, it makes histogram equalization over all components easier to implement (when equalizing RGB histograms, a thing called a *color shift* can occur, which means that for instance the green component is distributed differently than the blue component, thus the combined green and blue vector will represent a different color than the original value).

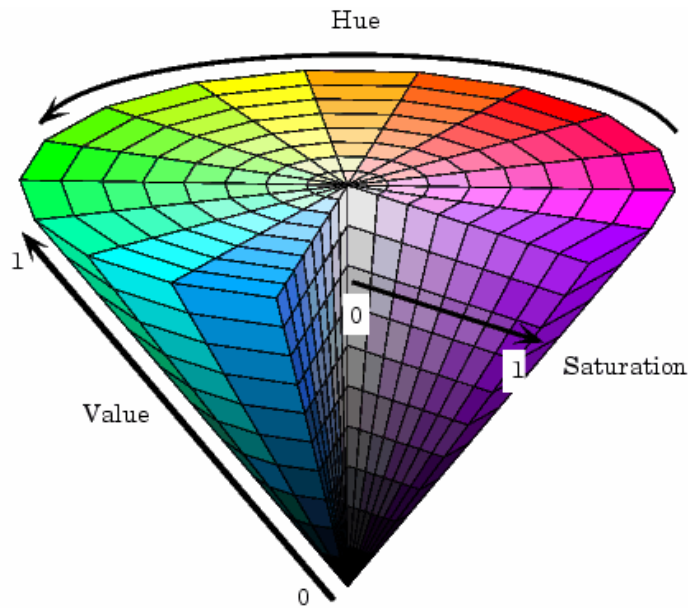


Fig 6: HSV Color Space

2.2.1.3 Color Space conversions

Sometimes it is desirable to be able to convert from one color space to another. In some cases this is an easy process, but in other cases, such as converting from RGB to XYZ, it is a bit more complicated. Since XYZ will not be used later, I will not explore this further, and focus on conversions between RGB and HSV.

Conversion from RGB to HSV is as following:

$$H = \arccos \left\{ \frac{\frac{1}{2}[R - G] + (R - B)}{\sqrt{(R - G)^2 + (R - B)(G - B)}} \right\}$$

$$S = 1 - \frac{3}{R + G + B} [\min(R, G, B)]$$

$$V = (1 / 3) (R + G + B)$$

When we convert from HSV to RGB, it gets a bit more complicated. Since Hue defines color, and Hue is circular, we must use a different formula for each value of Hue.

For H in [1..120] :

$$b = (I/3) (1 - S)$$

$$r = (I/3) \left[1 + \frac{S \cos H}{\cos(60 - H)} \right]$$

$$g = I - (r + b)$$

For H in [121..240]

$$H = H - 120$$

$$r = (I/3) (1 - S)$$

$$g = (I/3) \left[1 + \frac{S \cos H}{\cos(60 - H)} \right]$$

$$b = I - (r + g)$$

For H in [241..360]

$$H = H - 240$$

$$r = (I/3) (1 - S)$$

$$g = (I/3) \left[1 + \frac{S \cos H}{\cos(60 - H)} \right]$$

$$b = I - (g + r)$$

Red = 3Ir, Green = 3Ig, Blue = 3Ib/100

In this thesis solution, the HSV model is used extensively.

2.2.2 Shape

Shape is the contours and shapes of objects represented in the image. The process of extracting shapes often goes like this: First we find the contours in the image, then we segment the image into the different contours and index these. Finding contours can be obtained by using *chain codes* which is an algorithm that “walks” around the edge of *regions*, creating a set of straight lines around the region. A region could for instance be an area of similar color or an area that is in some way different from the rest of the image. These lines can be further simplified with *polygon approximation* which makes the lines less jagged[5]. Further we can capture the essence of the shape using Fourier Descriptors. The shapes can be indexed by simple statistical moments, such as the mean and variance.

2.2.3 Texture

Texture is a way of extracting what kind of “surface” an image or object has. [6] has proposed 6 different features which describes texture:

- Coarseness: Coarseness is related to roughness and perhaps the most fundamental feature of texture. The larger the individual image elements, the coarser the image.
- Contrast: This feature measured by using four parameters: dynamic range of grey levels of the image, polarization of the distribution of black and white on grey-level histograms or ratio of black and white areas, sharpness of edges and period of repeating patterns
- Directionality measures elements shape and placement.
- Line likeness measures is concerned with the shape of a texture element
- Regularity measures variation of an element placement rule.
- Roughness measures whether or not an object is smooth. A polished ball will have little roughness, while a mountain will have a rough surface (at least close up).

These features can be indexed and compared with an appropriate distance measure.

2.2.4 Three common retrieval methods

We have several ways of measuring distance between a query and an image collection. I will describe the three used in this paper:

2.2.4.1 Euclidian distance

Euclidian distance is perhaps the most basic method of measuring distance between histograms. It compares the histogram bins one-for-one, not taking to account similar colors in other bins. The Euclidian distance can be computed as:

$$d^2(h,g) = \sum_A \sum_B \sum_C (q(a,b,c) - t(a,b,c))^2$$

where d is the distance, and q and t is the histograms to be compared.

2.2.4.2 Histogram intersection distance

[7] has proposed a distance measure that has shown good performance on partial matching of histograms. The method basically calculates the cross-distance between two histograms. This means among other things that colors not presented in the query will not be taken into account when measuring the distance. This reduces the problem with backgrounds explained in chapter 2.2.2.1. The histogram intersection distance can be computed as:

$$d(Q, T) = 1 - \frac{\sum_i \min(q_i, t_i)}{\sum_i t_i}$$

Where d , q and t is the same as in Euclidian distance.

2.2.4.3 Quadratic distance

The quadratic distance, also called *cross distance*, is used in the QBIC-system[8] which will be introduced later on. This method considers the cross-correlation between histogram bins based on the perceptual similarity of the colors represented by the bins. The set of correlation values is represented in a *similarity matrix*. The sum is normalized by the histogram with the fewest samples.

It can be computed like this:

$$d(Q,T) = (Q-T)^t A(Q-T)$$

where d is the distance, Q and T is the histograms, and A is the similarity matrix.

The similarity matrix is computed (for RGB color space) like this:

$$a(i,j) = 1 - d(i,j) / \text{MAX}(d(i,j))$$

And for HSV:

$$a(i,j) = 1 - \frac{1}{\sqrt{5}} \left[\sqrt{(v_i - v_j)^2 + (s_i \cosh i - s_j \cosh j)^2 + (h_i \cosh i - h_j \cosh j)^2} \right]$$

This method is by far the most computer expensive of the three i will present in this paper. It is also the most sophisticated. The effectiveness of this method compared to the computer time spend running it is a matter of discussion. Of these three methods, the intersection distance is the fastest, followed by the Euclidian distance and the quadratic distance last by far.

2.2.5 Queries

So now we have described some basics of CBIR. We have yet to describe how to perform queries in CBIR systems. There are three common way of forming queries in a CBIR system:

2.2.5.1 By example

With this approach we present a sample picture and tell the system to find similar picture to this. Use of *relevance feedback* is common when using this approach. Relevance feedback means that when the system returns a result set, you can choose an image of this result which is closer to what you want, and perform another search. This way you (hopefully) get closer and finally find to the image you want.

2.2.5.2 By low level features

With this approach, you “sketch” the image you are after, using color and often basic geometric tools enabling you to draw shapes. If texture is used in the system, the possibility of indicating what kind of texture you are after is provided, often based on the features described in chapter 2.2.3 . Relevance feedback is often used. The problem with this method is that it requires the user to be an “artist”, and spending time sketching the image. Most users will want a fast way of searching images, and this method is perhaps too time-consuming for most users. Using low-level features directly may often be a problem for people who are not trained in computer image theory,

and they may have problems dealing with the connection between an image and its features. This is due to *the semantic gap* that exists between low level features and high level concepts[24].

2.2.5.3 By textual IR (classification of images)

This approach is used when automatic or manual annotation (or classification) of images is used, as described in the top of this chapter. This approach has a lot going for it, it is a way of submitting queries that most users are used to through other search systems and it is fast. The problem is not with the query, but with the classification methods that exist today. Manual classification is impractical, and there is still a lack of a satisfactory way of classifying images manually. Image classification will be further explored later in this paper.

2.2.6 State of the art: CBIR

There are quite a few running systems demonstrating CBIR, and I will review a few of them:

2.2.6.1 Blobworld

Blobworld is a research project performed by the Computer Science Division, University of California, Berkeley[20]. It segments the image into “blobs” which are corresponding in texture and color. A demo is provided on their homepage, which enables search by example and keywords. The demo-site is unfortunately down, and presumably it will stay like that, since the work on the project was shut down a couple of years ago.

2.2.6.2 QBIC

Qbic is made IBM and stands for Query by Image Content. QBIC is a retrieval engine that retrieves images based on color percentages, color layout and textures occurring in pictures. The The State Hermitage Museum[21] uses the QBIC engine to enable search for paintings using either layout (drawing colored shapes) or color search.

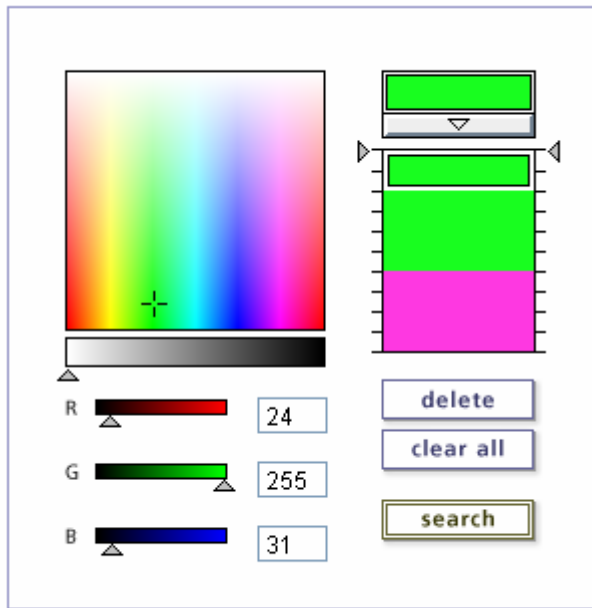
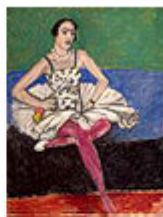


Fig: 7 State Hermitage Museum QBIC color query



1) Zorah Standing

Matisse, Henri
1912



2) Ballerina

Matisse, Henri
Circa 1927



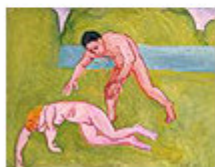
3) Portrait of Empress Alexandra Fyodorovna

Bondarevsky, Nikolai
Kornilyevich
1907



4) Study of an Old Woman's Head

Geoffroy, Henri
Jules Jean 1880s



5) Nymph and Satyr

Matisse, Henri
1908



6) The Mill in Eprave

Fourmois, Theodore
1852

Fig 8: Results from QBIC query

2.2.6.3 VisualSEEK

VisualSEEk [22], made by John R Smith and Shih-Fu Chang is a functional prototype for searching for visual features in an image database. The queries are made by diagramming spatial arrangements of color regions. It uses color and spatial relations as their main features.

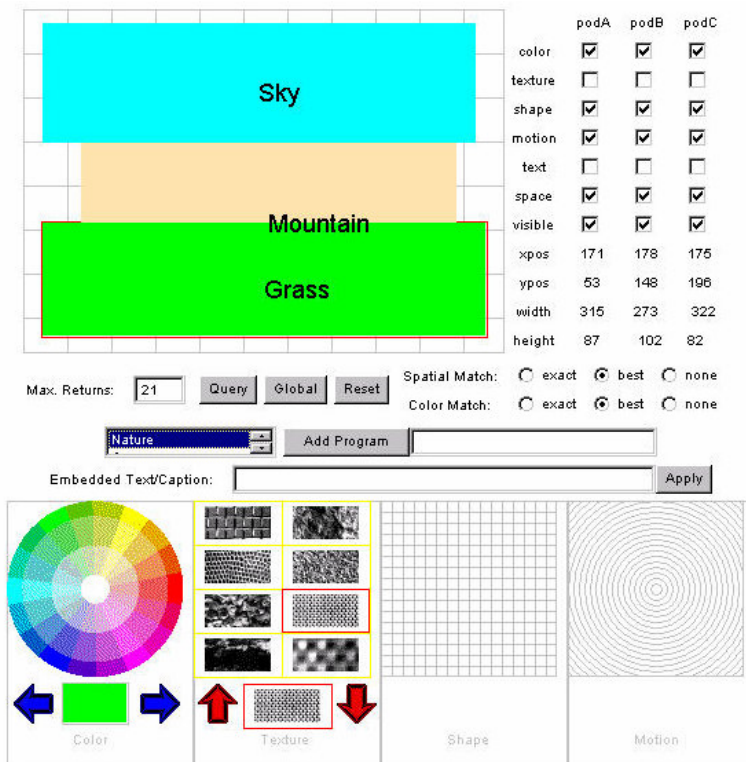


Fig: 9 User interface for a prototype of VisualSEEk

2.2.6.4 WebSEEk

Webseek [23] is made by the same people as VisualSEEk, but it is targeted at pictures and videos on the internet. It consists of a crawler which uses color information and surrounding text to index and catalogue the images and videos. Queries are made by entering keywords or browsing the different categories.

2.3 Image Classification and Image Annotation

Image classification is an attempt to (often textual) label an image with appropriate identifiers. These identifiers is determined by the area of interest, whether it is general classification for arbitrary pictures (for instance from the internet), or a specific domain, for instance medical x-ray images or geographical images of terrain. Image classification is related to *image annotation*, although image annotation generally uses a larger vocabulary (that is, number of classes) than

image classification.

There are two main approaches to image classification; supervised and unsupervised image classification. Supervised classification uses *training sets* of images to create descriptors for each class. The training sets are carefully selected manually to represent a common picture set of that class. The classifier method then analyzes the training set, generating a descriptor for that particular class based on the common features of the training set. This descriptor could then be used on other images, which determines if that image is a part of that class. Supervised image classification is a subset of *supervised* learning. Supervised learning can generate models of two types. Most commonly, supervised learning generates a global model that input objects to desired outputs. In some cases, however, the map is implemented a set of local models. Such algorithms are often implemented using *neural networks*, *decisions trees*, *state vector machines* and *Bayesian statistical methods*. The latter two has shown great promise in this area.

Unsupervised image classification does not rely on a training set, instead it uses *clustering* techniques which measure the distance between images and group the images with common features together. This groups can then be labeled with different class-identifiers. This paper will deal with supervised classification, so unsupervised techniques will not be further explored.

2.3.1 State of the art: Supervised image classification

There have been several approaches to image classification; one of the more promising ones is a study [17] where the researches use image features such as edge direction histograms, edge direction coherence vector, color histograms and Bayesian statistical methods in order to classify indoor from outdoor, and cities from landscapes. Landscape has three subsets; sunset, forest and mountain. The test results show high classification accuracy using this method. A more extensive study have been done in [19] which explore annotating pictures with the use of two-dimensional Hidden-Markov models. The study uses a vocabulary of a few hundred words, and has shown high accuracy in their tests.

An other approach is the use of SVM – state vector machines which has been proposed in[18], by building a confidence based dynamic ensemble of SVM classifiers for the purpose of annotation.

2.4 Evolutionary Algorithms

”Most species do their own evolving, making it up as they go along, which is the way Nature intended. And this is all very natural an organic and in tune with mysterious cycles of the cosmos, which believe that there’s nothing like millions of years of really frustrating trial and error to give a species moral fibre and, in some cases, backbone”

-Terry Pratchett, The Science of Discworld

.

The main source of information for this chapter is [9, kap 1–4]

Evolutionary algorithms stems from Darwin’s long established theories of how organisms on this planet have evolved from single celled amoebas to the advanced organisms we have today. This interpretation is only half true. Human beings may well have evolved from an amoeba, but that is really not the point the evolution theory is trying to make. The point is that evolution makes (or forces) an organism to adapt the best possible way to its environment. As an example of this, we still have amoebas today that have been more or less the same for hundreds of millennia. The reason why these organisms haven’t evolved in the same way as humans have, is that the environment they live in didn’t require them to do so. This may seem to be a digression, but it is actually a vital point when we try to explain the theory behind evolutionary algorithms.

Evolutionary algorithms (EA) fall within the science of Artificial Intelligence (AI), more specifically *machine learning, the study of computer algorithms that improve automatically through experience*[Mitchell 1996, AI]. A related branch is *neural networks*, that tries to simulate the human brain to reach its goal and *Bayesian networks* which try to reason with uncertainty using a statistical approach. The science can be traced back to 1958, when Friedberg proposed an idea of evolutionary algorithm for automatic programming, and initial attempts where made in the 1960s which made the principles behind EA. The research of John Holland at the University of Michigan led to the development of the Genetic Algorithm (GA), which most people associate with when they think of EA today. In 1992, John Koza took the next step forward and introduced Genetic Programming (GP), which is what this paper will take advantage of.

2.4.1 Natural Evolution

When dealing with genetic algorithms it is important to know what theories they are based on. It is also important to know that although GA has adopted some traits found in Darwin’s

theories, they are far from a copy of these theories. GA's are highly simplified versions of what takes place in nature, and only those traits that seem important for the GA scientists are preserved.

Computer based evolution focus on the broad term of evolution, that is, selection of the *fittest*, recombination (breeding) and mutation in order to generate an "organism", or in this case an algorithm which is best suited for its task.

In Charles Darwin's *On the Origin of Species* (1859), he proposed four essential requirements for the process of evolution to occur:

- Reproduction of some individuals within a population
- A degree of variation that affects probability of survival (for example mutation)
- Heritable characteristics, that is, similar individuals arise from similar parents.
- Finite resources, which drive competition and fitness selection (so that amoebas evolve into tax-paying citizens).

With these requirements fulfilled, the individuals in a population will adapt to the specific conditions they live in.

Before we proceed, a brief description of elementary molecular biology is given:

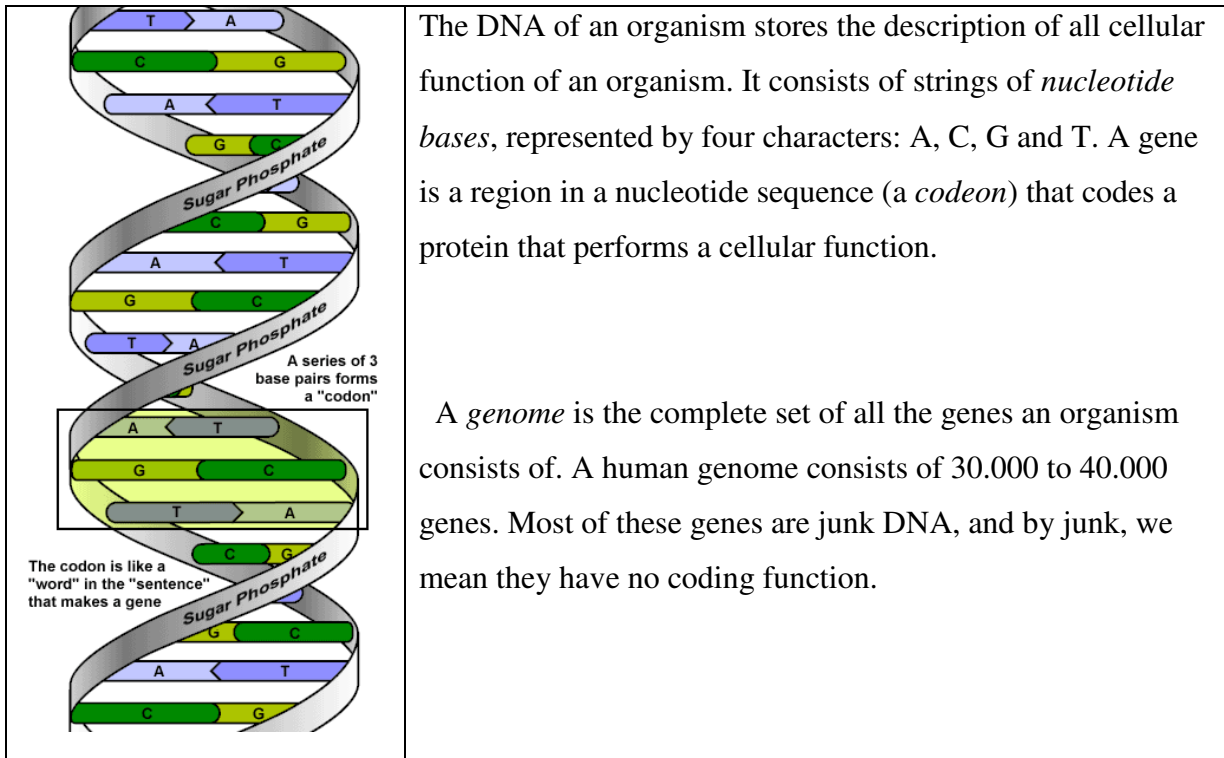


Fig 10: DNA structure

The DNA of an organism stores the description of all cellular function of an organism. It consists of strings of *nucleotide bases*, represented by four characters: A, C, G and T. A gene is a region in a nucleotide sequence (a *codeon*) that codes a protein that performs a cellular function.

A *genome* is the complete set of all the genes an organism consists of. A human genome consists of 30.000 to 40.000 genes. Most of these genes are junk DNA, and by junk, we mean they have no coding function.

2.4.2 Genes

A gene can be described as the primary unit on which natural evolution operates on[9]. Most people have some concept of what a gene is, but it is hard to give an exact definition. [10] defines a gene as a “*unit of hereditary information that occupies a fixed position (locus) on a chromosome. Genes achieve their effects by directing the synthesis of proteins*”. A *genome* is the whole hereditary information of an organism that is encoded in the DNA. This includes all of the genes and the non-coding sequences (the “junk”).

2.4.3 Evolution and variation

The evolutionary traits relevant for this paper are sexual recombination and mutation:

Sexual recombination depends on two organisms (parents) combining their DNA to create a new child organism. Unlike mutation, this is a highly controlled process, and requires that the two parent organisms are virtually identical (with only minor variances to their DNA).

Mutation is an obvious way of insuring variety in a population. Mutation is a process that alters the DNA sequence of base pairs such that different codons are expressed. In real life, this process can be triggered by various causes, such as radiation or chemical mutagens. Copying errors

through recombining can also happen, but not often, since most organisms have the ability to correct these copying errors (*self maintenance*) and a redundancy of codons which can replace “broken” ones. Severe mutation in organisms often leads to disaster. Since the mutation process often works at random with no “plan” of what the outcome will be, severe mutation will in most cases lead to a less viable organism. A small amount of mutation often goes unnoticed, and will in some cases lead to an improvement of that particular organism, strengthening its chances to reproduce. A small example of mutations in humans is the population of most of north-western Europe which have a mutated gene enabling them to digest dairy products, especially milk, much easier than people from other regions. The reason for this mutation can only be speculated upon. Is it because people in north-western Europe drink vast amounts of milk compared to other regions, and therefore “embraced” this mutation in a non-conscious manner, allowing it to spread, or is it just a coincidence? No one knows.

[9] defines three major types of mutation that can occur, each with its individual consequence:

- *Point of base-pair switch mutation*: A single base pair (e.g A-T) is replaced with C-G. This does not happen often, and is often corrected. In some instances, the mutation goes unnoticed, resulting in a minor change of the organism. This is the weakest of mutations, and is probably the most constructive mutation, since it happens very slow and gradually, enabling new opportunities of adapting to the environment.
- *Frame shift mutations*: This is a more severe form of mutation, where a new base-pair gets inserted or deleted from the DNA-sequence, scrambling the DNA from that point in the DNA. A significant length of DNA is modified, and a major change in the organism occur, often not a positive one.
- *Large-scale sequence mutation*: This is the most severe of mutations, rearranging large portions of the genomes DNA. This could happen when exposed to high levels of radiations, and is often fatal for the organism.

2.4.4 Genetic Algorithms

John Holland [11] has developed the dominant outline of genetic algorithms which encapsulates most of the biological evolutionary mechanisms mentioned in the last chapter. There are variants, but all of these spins of the work of Holland. A Holland-type GA algorithm consists of a bitstring, a proportional selection method and crossover variation.

A common starting point for most GA-algorithms is these steps (from [9]):

- Create a randomly generated population of N chromosomes, each of some length m bits.
- Test each chromosome (i.e a possible task solution) within the problem space and assign a measure of fitness $f(x)$.
- Selection phase: Select a pair of chromosomes from the population with probability based on their fitness.
- Apply a set of genetic operators to the two parent chromosomes: With some crossover probability pc , apply crossover at some randomly selected point along each chromosome.
- Apply mutation to each new chromosome with a probability pm .
- Place the new chromosome in the new population
- Replace the old population with the new population, or we may use a steady-state method, in which an overlapping population model is maintained.
- Test if target termination criteria is met, such as a specified best fitness value; else repeat from step 2.

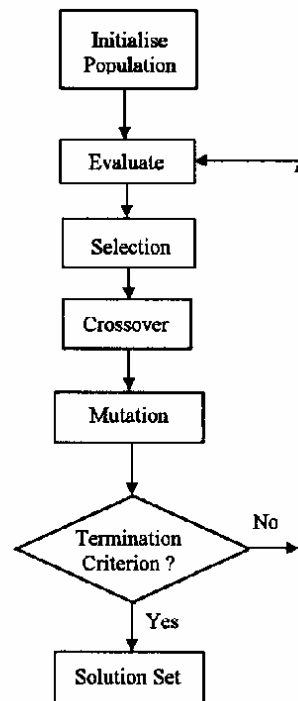


Fig 11: A general Genetic Algorithm

2.4.4.1 GA operators – from biology to computers

The different operators mentioned in the last chapter are “computerized” versions of the processes that happens with living organisms. How do these operators work, and how do they relate to the real processes?

Evaluation and Selection:

In nature, evaluation and selection is more or less one process. If your environment or faults in your genetic make-up kills you before you have been able to pass on your genes, or disables you from passing on your genes, you are, in some sense, been evaluated and found wanting. Since you are dead or infertile, there is no chance for your genes to be selected for the next generation. In a GA, we use a *fitness*-function to evaluate each organism. The fitness function is perhaps the most important operator in GA, since it is this function that defines what the organism optimally should be able to do, and in what direction they will evolve in. If we try to make an overly simplified fitness-function for a living organism it may look something like this:

IF organism NOT(been eaten, stamped or in other ways mauled) AND (is fertile) AND (skilled OR rich OR otherwise attractive) THEN fitness = high

In GA, the next step is to *select* the next organisms which will form the bases for the next generation. Here we have to consider if we want to exploit the best of the population fast (that is, only select a few of the best), or to explore the search space by select more of the last generation, but with more “less fit” ones. If the former way is chosen, we risk that an organism which partially completes the goal, but reaches a dead end, becomes dominant without it ever managing to reach its goal. If we choose the latter way to the extreme, we risk never finding a convergent solution. A wide range of selection routines have been explored, and I will mention a couple of them:

Fitness-Proportionate selection

The roulette wheel method is a very popular implementation, and to sum up this method, the number of individuals allowed to reproduce is equal to the individual’s fitness divided by the average fitness from the population. A criticism of this method is that the initial variance in the population is high, which will lead to a dominant organism and convergence before enough search-space is explored.

Rank Selection

Rank selection is an attempt to overcome the problem with fitness-proportionate method. The method sorts the individuals and rank them from 0 (least fit) to N-1 (fittest), where N is the size of population. The individuals selected for the next generation is then selected by a linear or non-linear function. A linear selection could be something like “select the top 20 individuals for next generation”. A non-linear selection could use the generation number in order to converge faster with later populations.

Criticism of this method is that the convergence takes longer to complete, since the pressure on the population is smaller. Still, it is a simple and efficient method

Crossover (“breeding”)

When we do a crossover in GA, we select two parents, and swap portions of each code to produce children. The size of the portions depends on what the problem area is, and an optimal crossover theorem is still not developed. This results in resorting to experiments and iterations on the specific problem area to find the best crossover –schema.

Parents:

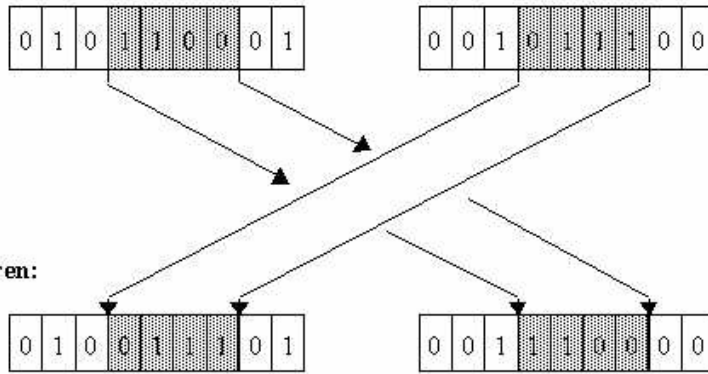


Fig 12: Crossover

Mutation

When we mutate a GA, we replace one of the instructions in the individual with a new (arbitrary) one. There has been some debate on how big a role mutation shall play in GA. In nature, mutation is a small factor compared to crossover, and many GA researches agree with this, but [12] argues that mutation may play an even bigger role than crossover, and in fact be the sole means to evolve individuals. This is of course in the scheme of GA, and would probably not work so well in nature. So the number of individuals mutated in each generation is still somewhat of a debate, and again, the solution may be to experiment with different mutation-rates to find the best results. If for instance you find that your GA converge to fast without solving the problem, a higher rate of mutation may lead to “new ways” to solve the problem.

Termination

If an ideal individual is found, that is, if it is impossible to be more fit, the there is no point in exploring further generations and the GA will terminate. This is not an interpretation of a natural process, where mutation and breeding will continue long after the “ideal” organism is made.

2.4.5 Genetic Programming

Genetic Programming was the next major step in EA-science. It is a sub-class of Genetic Algorithms and differs in the way that its goal is to produce executable code which can be used

directly in the problem area.

[9, p.48] states that Banzhaf gave a useful definition of GP in 1998:

“Genetic programming, shall include systems that constitute or contain explicit reference to program (executable code) or to programming language expressions. So, for example, evolving LISP lists are clearly GP because LISP constitute programming language structures and elements of those lists constitute programming language structures and elements of those lists constitute programming language expressions. Similarly, the common practice among GP researches of evolving C data structures that contain information explicitly referring to programs or program language tokens would also be GP, (Banzhaf et al., 1998, p. 5)”

A limitation of GA is that it uses a fix-length size of the genome. This could possibly hinder the genome of evolving to a sufficiently advanced state. GA also relies on a bit string for representing its genomes. To construct a complicated genome using bit-strings can be a hard task. GP has solved these problems to some degree by using variable-length genomes and the possibility of a tree-based representation. GP also has the possibility of defining a function set, where any kind of abstract and atomic operations could exist.

When we use GP to solve a problem, we have to define 5 different items:

- *The GP terminal set:* “The terminal set is comprised of the inputs to the GP program, the constants supplied to the GP-program, and the zero-argument functions with side effects executed by the GP program” [13]. When dealing with image classification, which we will do later, the terminals consists generally of image features.
- *GP function set:* The function set consists of all the operators, statements and functions that shall be available for the system. From complicated functions to primitive “*”, “+”, etc to logical operators such as AND or NOT. This flexibility enables a genome to be highly advanced if deemed necessary. Even though one may be tempted to throw in as many functions and operators as you can think of, a bit of caution is required. Every element added will increase the search space, increasing the chance of premature convergence. By selecting the most necessary and functional operators, the chance of a successful GP is higher.
- *Fitness measure:* The fitness measure is constructed the same way as in GA.
- *Parameters for controlling the run:* Determines how the run is executed.
- *The method for designating a result and the criterion for terminating a run:* How is the

result going to generated, presented and stored. When will a run terminate? When it has a certain goal, or after a certain generation?

A GP-algorithm can either be tree-based or linear. When a GP is tree-based, it can combine operators and functions as seen in figure 9.

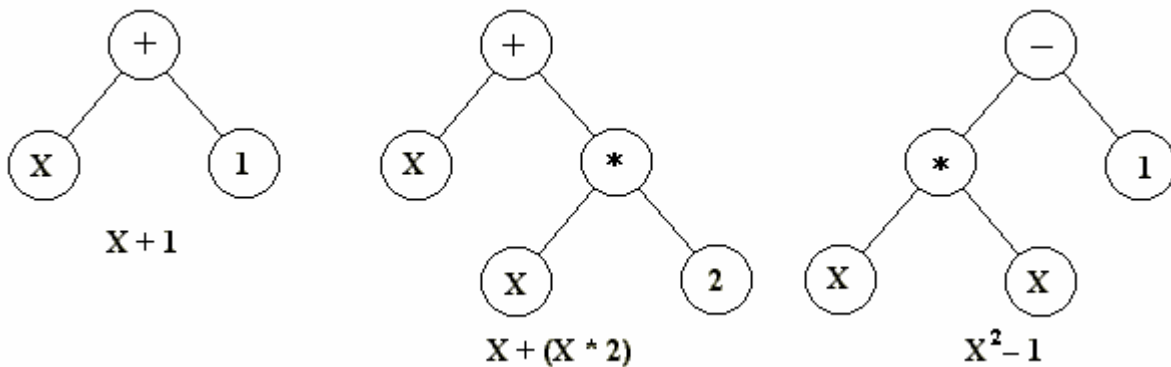


Fig 13: Tree structure in GP, with operators +,-,*, a variable X and two numbers 1 and 2.

A linear GP-algorithm run from top to bottom without using a tree-structure.

2.4.6 Difference between GP and GA operators

2.4.6.1 Crossover

When using a tree-structure in GP, it is important that the crossover result in valid code. The most common way to insure this, is to pick sub-trees from both parents and combine these. In linear GP, the crossover is done more or as in GA.

2.4.6.2 Mutation

When doing linear GP, the mutation happens in a similar way as in GA's. One operator is dropped and replaced by a new random one. Although this can lead to big differences, it is not such a dramatic as mutating a tree-based GP algorithm. The most common way of mutate a tree, is to remove a sub-tree and generate a new random one. This can lead to changes in a very big way, and is a more serious mutation with greater effects.

2.4.7 Application and limitation of Genetic Programming

Genetic Programming is useful and have been applied in complex systems where traditional analysis of the problem domain is difficult and exhaustive search for the optimal solution is almost impossible. Such systems could for instance be the best way to design an integrated circuit (finding the overall shortest route to each component), production systems, robot behaviour, and most NP-complete problems such as the travelling salesman. GP, when properly made, enables fast and accurate ways of finding a solution. It is also used to optimize existing systems by tweaking parameters to gain optimal performance.

The disadvantages with GP and GA is that (as the writer has experienced at length) it is very computationally expensive, it is probabilistic, so it can not guarantee a best solution, and is prone to premature convergence without finding a solution to a problem. Further, it is often problematic to describe a problem as a genome with a fitness function, although less so in GP.

2.4.8 Genetic Programming and image classification

Several papers have been written on GP and CBIR. Most of them dealing with feature extraction, but a few has tried an evolutionary approach to image classification. No commercial system has tried a full scale implementation of any of these methods as far as the writer knows. [14] successfully used tree-based GP to separate text from graphics using several variants of the low level features mean and variance and the arithmetic operators +, -, *, %, where % is a protected division (to avoid division-by-zero errors). Also the function 2^x , the conditional function $\text{iflte}(a,b,x,y)$, which returned x if $a \leq b$, otherwise y . Access function $f[n]$ where the range was between 0 and 11, plus integer constants in the range [0..11]. Experiments showed an accuracy of well above 90% when separating text from graphics.

[15] explores using loops with GP for a two-class binary image classification. The two classes are circles and squares. They discovered that when using loops, the GP required fewer generations to reach an acceptable state. [16] explores using GP in combination with feature vectors, weights and Bayesian statistical methods to be able to identify and filter out offensive (adult) material in certain web pages.

3 Evaluation methods of this thesis

Since this is an experimental study, a lot of the evaluations and analyses will be based on observations from the writer. This includes causes, effects (and side effects) which is considered interesting.

In order to test the hypothesis, some sort of system has to be implemented. It must contain several parts, covering all the different aspects of the problem. That is, feature extraction, distance algorithms, genetic algorithms and classification. The classification system is to be used as a test-bench for exploring the hypothesis, and is not meant to be used as a finished system. The system will then be tested on a training sets and an image collection. The results will be analyzed using precision/recall graphs, and measuring classification accuracy. Iterations and alterations of various parts of the system to bring it closer to the hypothesis will be described and justified as much as possible.

4 Classification system

4.1 Introduction

This paper deals with CBIR, which I had some theoretical knowledge of before I started. It also deals with Genetic Programming and image classification which I had no prior experience in. The classification system that has been built, show traces of that ignorance, since I learned along the way by building it. The theoretical studies I had done prior to building the system helped, but did not adapt perfectly to realization of the system.

I will describe how the system is built, and try to justify my choices along the way. Finally, an evaluation and discussion of the classification scheme, and test result will be done, followed by some thoughts of potential improvements.

4.2 The idea

My hypothesis was that by combining low level image features (color), weights, Genetic Programming and standard CBIR - distance measures, we should be able to extract common features of the different classes, enabling us to classify images based on this. To achieve this we

will define a training set, containing a representative selection of images for each class. The training set will then be measured against each other using standard distance measures. In addition we define a set of weights which will be added to the low-level features used in the distance measuring. The job of the GP-algorithm is to adjust these weights so that the maximum distance between the images in the training set will be as little as possible.

After this is, we should have a set of weights which we can use to determine whether or not an arbitrary image belongs to that class or not, or in other words, a descriptor of that class. Although I didn't have the time to implement and test it, it should be possible to combine these descriptors, making a multi-class classification scheme.

I will now go into depth and explain each essential parts of the test-system.

4.3 The Classes

I decided on defining five classes and build training sets for each of them. Each training set consists of 10 images, handpicked subjectively. The training set was more critical than I had thought, and have serious consequences for the classification routine later on. This will be discussed later in this paper.

The five classes I defined were all outdoor topics of various nature and city scenes. The pictures were collected from various sources on the internet.

Class 1: Forest

Training set:



Fig 14: The forest class' training set

Class 2: Winter

Training set:



Fig 15: The winter class' training set

Class 3: Mountains

Training set:

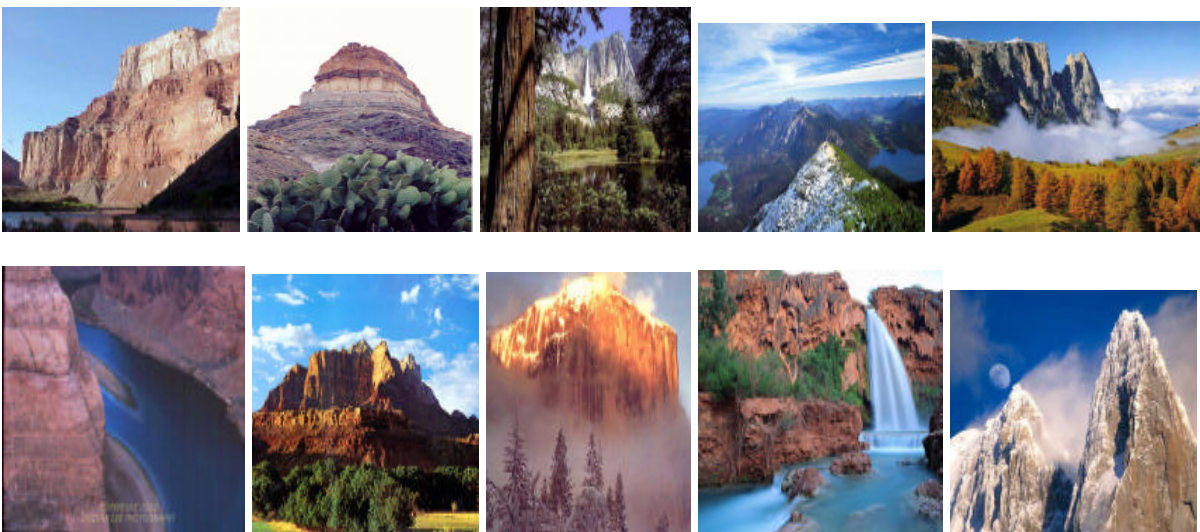


Fig 16: The mountain class' training set

Class 4: Desert

Training set:

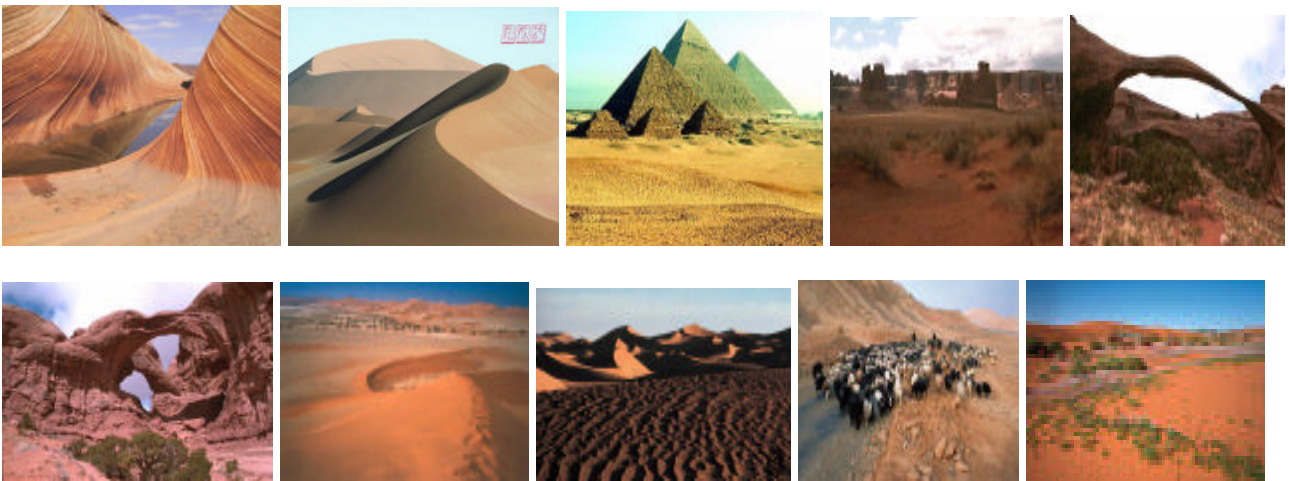


Fig 17: The forest class' training set

Class 5: Buildings or cities

Training set:



Fig 18: The forest class' training set

4.4 Feature extraction

The first important thing to consider is what kind of low-level features is to be used. Ideally, as many features as possible should be used to ensure that enough common factors to the training set will be found. There are two problems with this approach; the search space for the GP to explore

will expand for each feature added, making local convergence more likely. The reason for this is because of problem two, which is limited computer capacity. In order to explore the extra search-space, one could build larger populations which make it more likely that most (or at least a relevant part) of the search space is explored. In addition to adding the extra features for each individual in GP, a larger population must also be used. My one attempt on adding extra features resulted in 48 hours processing time for 10 generation with a population of 200 each. So in order to make a valid result, I decided to make it simple and use color histograms. Color histograms alone are seldom enough to base a real retrieval- or classification system on, but it had to do for this experiment.

In order to improve retrieval precision/recall, and thereby classification I decided to use HSV color space for my color histograms. HSV has shown to give better retrieval performance than RGB [25], and is easier to equalize. In addition, each image is segmented into 64 even parts, and a histogram is made for each of these. This is to enabling discovery of partial matching. The histograms for the HSV colors have bin sizes of 18, 3 and 3 respectively. The reason for this is to reduce computation time, and prioritizing Hue, since this is perhaps the most important part of the histogram, and reducing it further will lead to unacceptable inaccuracy. The hue-bins are filled with a special subjective hue-truncate method, which divide the bins into perceptually similar color (“red-ish”, “purple-ish”, etc. The histograms are then equalized over the brightness and saturation specter (the V and S in HSV). Equalizing the Hue-specter will lead to a color shift, so this is avoided. The histogram bins are then normalized to contain a value between [0..1].

4.5 Distance Algorithms

The distance algorithms used are Euclidian distance, histogram intersection distance and Quadratic distance. These were implemented after the formulas described earlier in this paper. Now, the idea is to use these algorithms to evaluate each genetic program, so it would be interesting to see if their retrieval accuracy had any later effects on the accuracy of the finished system. Remember, I have two kinds of histograms; one for the whole image and 64 “part”-histograms, so I reasoned that it would be likewise interesting to see if any special combination of distance-method/histogram pair performed better than the other. There are 16 different combinations for this:

Histogram for the whole picture	Histograms for segmented picture
Euclidian	Euclidian
Euclidian	Intersection
Euclidian	Quadratic

Euclidian	N/A
Intersection	Euclidian
Intersection	Intersection
Intersection	Quadratic
Intersection	N/A
Quadratic	Euclidian
Quadratic	Intersection
Quadratic	Quadratic
Quadratic	N/A
N/A	Euclidian
N/A	Intersection
N/A	Quadratic
N/A	N/A

Table 2: Distance combinations

We can remove the last instance in the table, where no algorithm is used for either of the histograms. I also had to discard every instance where the quadratic distance algorithm was used on the 64 histograms of the segmented picture. The image collection contained over 900 pictures, and the algorithm was too slow to compute results in a reasonable amount of time. That leaves 11 combinations to be investigated.

In the distance algorithms, the part-histograms are treated such that each part histogram in the query will be compared to the best matching part-histogram of the target. The overall “best” distance will be used as the collective distance between two segmented images.

When two methods are used, the combined distance was computed like this:

$\frac{(\text{dist}_w + \text{dist}_p)}{2}$, where dist_w is the distance for the whole picture, and dist_p is the partial match.

Precision / Recall graphs of some these combination will be discussed later in this paper.

4.6 Genetic Programming

As the theory of GP states earlier in this paper, there are 5 main things to consider of when we build a genetic program; *the GP terminal set, the GP function set, fitness measure, parameters for controlling the run, the method for designating a result and the criterion for terminating a run.*

4.6.1 Terminal set

The GP terminal set used, is the color histograms discussed last earlier, including weights for both kinds of histograms. The segmented histograms got a common set of weights, not separate ones. This was again due to search space and computational cost. In addition to that, variables telling what distance algorithms to use was added.

4.6.2 Function set

The GP function was perhaps hardest to define. Since I didn't have any prior experience to GP, I thought it was best to keep it simple, and therefore make the GP linear, not tree-based. This turned out to be a somewhat limiting choice, but at the time I had figured that out, a point of no return had passed. The initial function set consisted of functions to increment or decrement a "counter" which decided what the weight-index corresponding to the bins in the different histograms that was "active" (remember, we have three kinds of bins in each histogram; Hue, Saturation and Value, so counters were made for each of them). In addition to that, there were functions to increment or decrement the value of each weight, using the counter-variable as an index. When we discuss the results, we will see that there were major issues with this function set, and an improved one will be presented there.

4.6.3 Fitness function

The fitness measure was also a bit of a conundrum. I had to find a way to both utilize the distance algorithms, and at the same time making sure the individual algorithms in my GP was properly rewarded when going in the right direction.

First attempt:t

My first attempt failed miserably. The fitness function was set to add the generated weights to both the query and target histograms, and then measuring the distance between each and every one of the training set. The fitness was based on what algorithm generated the least maximum distance.

Pseudo-code for finding MaxDistance, first attempt

For all images

 Add weights to image

 Temp = Add together the distance to all images*weights except the current image

 If maxDistance < Temp Then maxDistance = Temp

End For

The formula for this was $fitness = 1000000 - maxDistance$. 1000000 was used to insure a positive result (sometimes the maxDistance got ridiculously high) . The optimal goal (which would have caused the GP to terminate) was a maximum distance of 0, that is, a fitness of 1000. When running the GP for the first time, I was amazed to see that it produced a perfect individual after about 40 generations. People experienced with GP (or human nature for that matter), could probably have predicted what had happened. When examining the result-set I discovered that the “perfect” individual had turned all the weights to zero, thus inducing zero maximum distance in the training set, and happily retired. This may be of interest to any employer complaining that the employees will do as little work as they can get away with.

Second attempt:

The second was to only add the weights to the query histogram, and calculated the distance as $1000000 - (max - min)$, where max is the maximum distance mentioned in the first attempt, and min is the minimum distance.

Pseudo-code for finding minDistance and MaxDistance, second attempt

For all images

Add weights to image

Temp = Add together the distance to all images except the current image

If minDistance > Temp Then minDistance = Temp

If maxDistance < Temp Then maxDistance = Temp

End For

This worked considerably better, and creating some impressive algorithms, at least to look at. I will come back to this when we discuss the results.

4.6.4 Parameters for controlling the run

Since my GP was linear, I didn't have to define a maximum tree-depth. I also chose not to restrict the number of functions in a genome, since I didn't experience any worse results without this restriction. I also experimented with loops, as discussed in [15], but reasoned that this may not be the right problem domain for using loops. The structure is 1 iteration, with as many functions as the individual see fit.

4.6.5 Criterion for terminating a run

The criterion for terminating a run is the perfect individual, which produces a maximum distance of zero in the training set. As long as the training set images don't consist of identical images, this individual is difficult to obtain. The other criterion is a maximum number of generations allowed. This criterion is vital, since the final report isn't generated until a run is finished.

4.6.6 Termination report

For each individual, a termination report is made when the algorithm has run and has been evaluated. The termination report includes information about the weights and what the fitness of the algorithm is. Also included is the max-value calculated in the fitness-function as well as the minimum distance achieved. This termination report is used to sort out the individuals whose genes will be used in the next generation.

4.6.7 Final report

When the entire GP is finished (either a perfect individual is found, or the maximum allowed number of generations is reached), a final report is made containing the best individual for each generation, the weights this individual has generated and the distance.

Excerpt from a final report:

```
(generation)1: 99984560,32767,99984536,95,  
Vekter brukt (hele) H : 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
S : 1, 1, 1,  
I : 1, 1, 1,  
Avstanden mellom min og max er 99984560,2430883  
Max = 15,4387569117159 Min = 8,44143859365354  
  
Bilde: C:\BILDER\trainS1.jpg.his, blokker brukt: X:5 Y:6  
Bilde: C:\BILDER\trainS10.jpg.his, blokker brukt: X:3 Y:3  
Bilde: C:\BILDER\trainS4.jpg.his, blokker brukt: X:3 Y:1  
Bilde: C:\BILDER\trainS5.jpg.his, blokker brukt: X:2 Y:4  
Bilde: C:\BILDER\trainS2.jpg.his, blokker brukt: X:0 Y:3  
Bilde: C:\BILDER\trainS3.jpg.his, blokker brukt: X:6 Y:0  
Bilde: C:\BILDER\trainS6.jpg.his, blokker brukt: X:1 Y:1  
Bilde: C:\BILDER\trainS7.jpg.his, blokker brukt: X:0 Y:2  
Bilde: C:\BILDER\trainS8.jpg.his, blokker brukt: X:4 Y:0  
Bilde: C:\BILDER\trainS9.jpg.his, blokker brukt: X:4 Y:2
```

Fig 19: A final report example

4.6.8 Selection

The selection method used is the rank-selection method mentioned earlier, using a linear function to select the best individuals. The termination reports for each individual are sorted from “most fit” to “least fit” and the top 10% of the population is selected for next generation.

After the selection is made, the selected individuals will undertake either a crossover, and in this system, *cloning*. Cloning is not a common thing in nature, but has a place in GP. Cloning is to make an exact copy of the individual and transfer it to next generation. Unfortunately, I didn't have the time to implement a mutation process.

4.7 Classification

After running the GP, we have all we need to attempt to classify an image. The classification is simple; either the image is a part of this class, or it isn't. In order to classify the image we need several items; we need the genetic code generated from the best individual (we can also hardcode the weights, since they're what the genetic code generates and they are provided in the final report). We also need to know what the "best" blocks used are from the segmented image (these could also be computed, but it is unnecessary since they are provided with the final report), if a distance algorithm has been used on the segmented image. By best, it means that that particular block was used as the query when finding the distance between two segmented pictures, according to the description of the.

Then, we need some *boundaries* for the class. If an image is inside this boundary, it is defined as a part of that class. The boundaries will be defined as follows: We multiply the weights of the class-GP to each of the training set-images history bins. We then sum up the histograms using this formula:

$$\text{Class_nr}(\text{image}) = \frac{\sum_{i=0}^{l-1} (wp(i) * \text{best}(p(i))) + \sum_{i=0}^{l-1} (ww(i) * w(i))}{2}$$

Where l is the length of the vector, wp is the GP-generated weights for the segmented image, $\text{best}(p)$ is the histogram which is chosen as the relevant in the segmented image distance algorithm (in the *distance algorithms* chapter). ww is the GP-generated weights for the whole image histogram, and w is the histogram for the whole image.

Then we can define the boundaries like this:

$$\text{Bottom_boundary} = \min(\text{Class_nr}(\text{training set}))$$

$$\text{Top_boundary} = \max(\text{Class_nr}(\text{training set}))$$

For experimental purposes, top and bottom boundaries are absolute, which means that anything outside this boundary is rejected. The boundaries can be precompiled to save computation time.

To find out if an image is a part of that class or not, we extract the two features used, apply the $\text{Class_nr}(\text{image})$ function, and check if the returned number is inside the boundary.

5 Evaluation

In this evaluation, I will first try to evaluate the different combinations of retrieval methods as mentioned in chapter 4.5, then I will do a preliminary evaluation of the GP and classification results. The function set of the GP will then be altered and a new evaluation will take place. The GP results, potential and limitations will be discussed, as well as the problems and limitations with the classification method.

5.1 Evaluation of retrieval methods

To make the evaluation process easier, I removed all but 11 images in the image database from each theme for each average precision/recall figure. This means that when I wanted to retrieve “snow” pictures, only 11 of the pictures in the database of about 900 images contained pictures of snow. When another theme was tested, the snow pictures were put back in the database, and 11 pictures of the new theme were left in the database. To calculate the average, I queried the database using the 11 images. The best result would always be the query picture, since it was in the database, so that was always discarded. When this was done for all combinations of algorithms, calculating the average was a straight-forward process, using the method described in chapter 2.1.4. The average precision/recall and some combinations of distance algorithms were put into a joint average precision/recall figure for each theme. The combinations selected was chosen because I had been able to generate GP-weights for all themes using those three methods, perhaps making it possible to draw a parallel between the precision/recall graphs and classification success. Unfortunately I didn't have the time to test all combinations. The notation in the examples and the figures is algorithm - algorithm, where the first indicate what algorithm is used on the whole image, the second what algorithm is used on the partial image match. One note on the partial image match; since it will locate the smallest possible distance, thus giving back very small distances, it is only meant as an aid to further improve the accuracy.

First I will give two examples of queries, and two result sets based on different algorithms for each.

Query 1 (theme: Buildings and cities) :



fig 20: Sample query 1

Results (ignoring the first result, which is the query image) :

Using Quadratic distance – N/A



fig 21: Result set of Quadratic-N/A to query 1

Using Intersection distance – N/A



fig 22: Result set of Intersection-N/A to query 1

Using Euclidian – N/A



fig 23: Result set of Euclidian-N/A to query 1

Query 2:



fig 24: Example query 2

Using Quadratic – Intersection



fig 25: Result set of Quadratic-Intersection to query 2

Using Intersection – Intersection

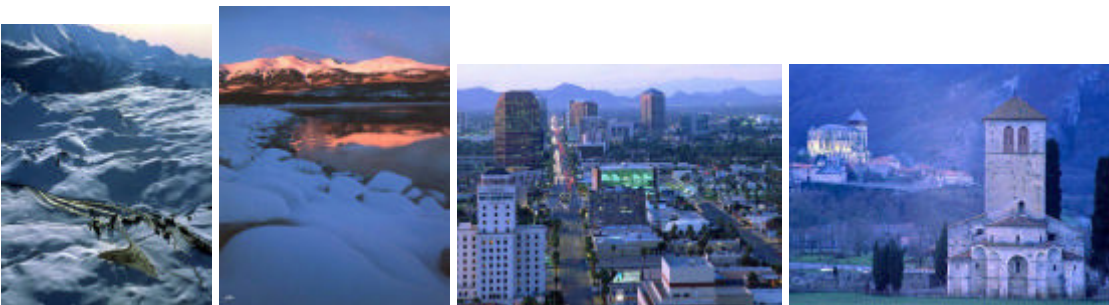


fig 26: Result set of Intersection-Intersection to query 2

Using Euclidian – Intersection

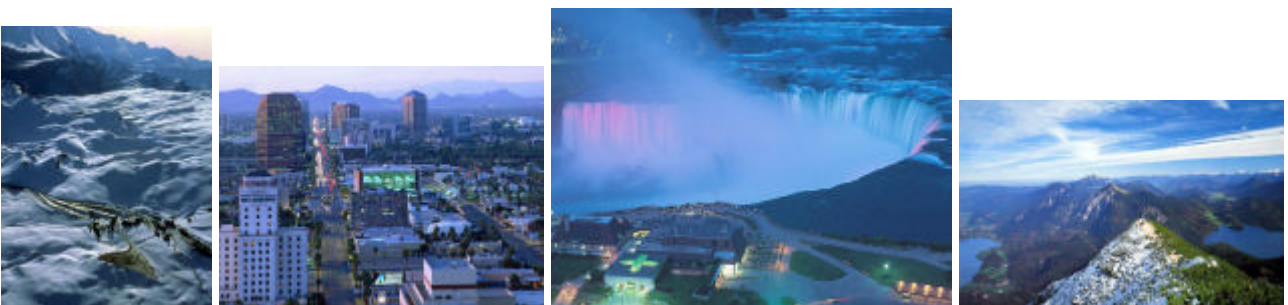


fig 27: Result set of Euclidian-Intersection to query 2

As we can see, especially with the quadratic algorithm in query 2, there can be massive differences in accuracy, depending on what the query is. I really have no explanation of why the quadratic method works so well in query 1, but is way of target in query 2. That's a topic for another paper.

Average Precision/Recall figures

“Forest”:

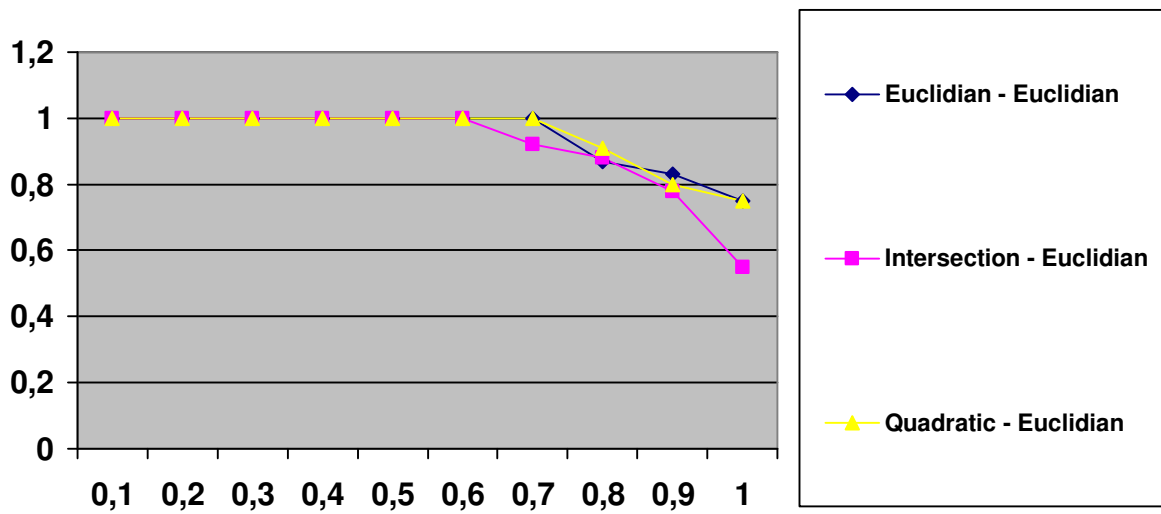


fig 28: Average precision/recall figure for “Forest”

“Winter”

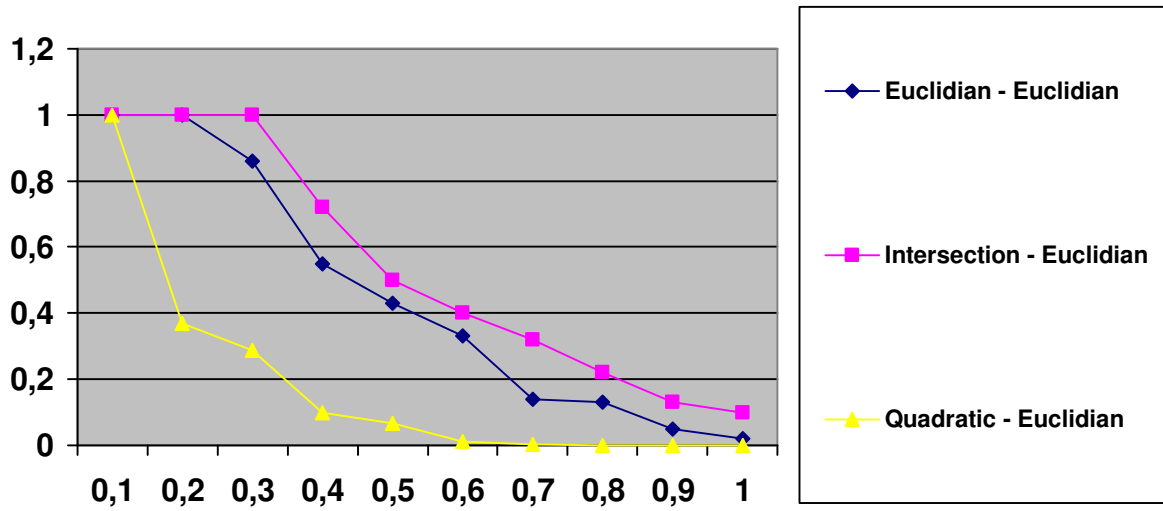


fig 29: Average precision/recall figure for “Winter”

“Mountains”

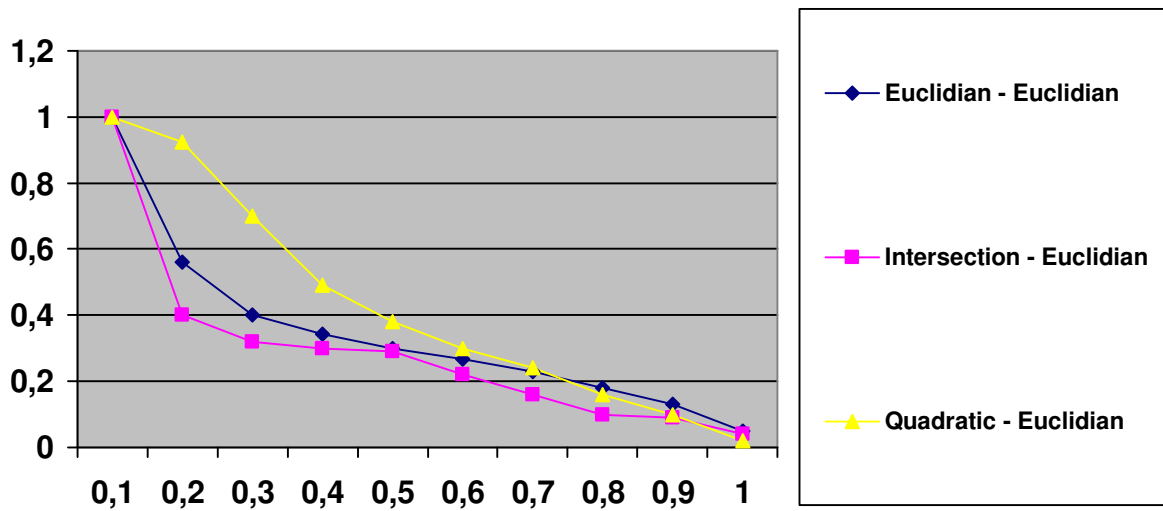


fig 30: Average precision/recall figure for “Forest”

“Desert”

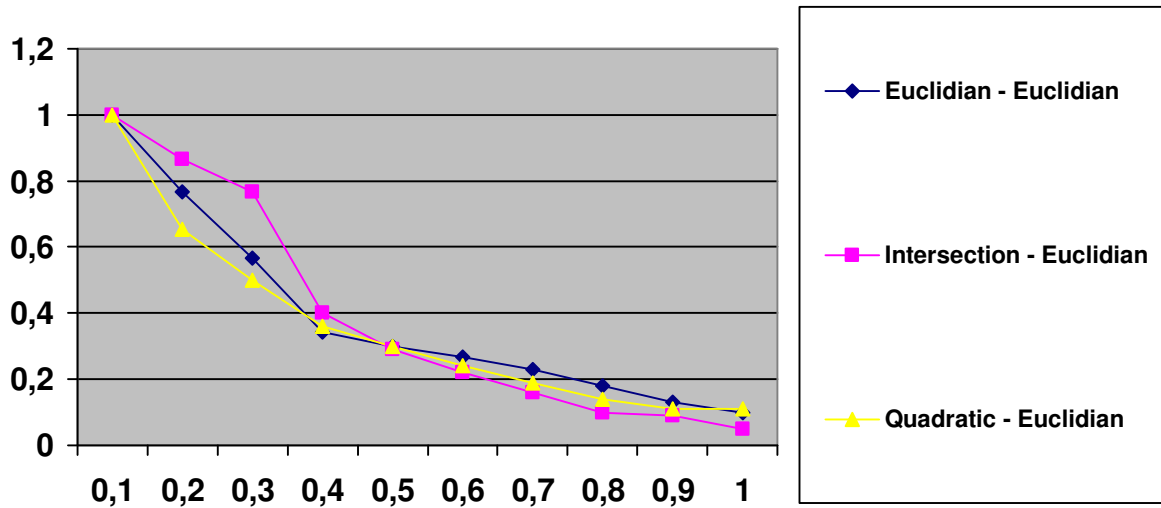


fig 31: Average precision/recall figure for “Desert”

“Buildings”

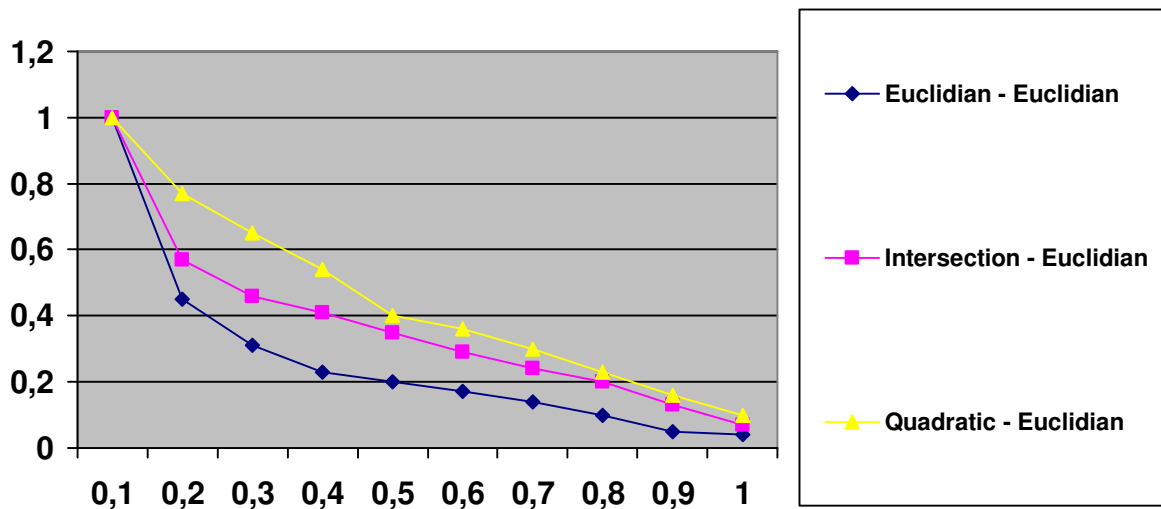


fig 32: Average precision/recall figure for “Buildings”

It may not be wise to draw a conclusion out of these kinds of diagrams, since judging what is relevant or not is a highly subjective task. These graphs may indicate that the intersection algorithm is a small step ahead on some of the retrieval tasks, and also indicate that the quadratic algorithm may not be worth its long computing time. I will discuss if a comparison between these graphs and the eventual classification result can be made.

5.2 Initial evaluation of the GP

The reason for this “initial” evaluation was that the first design, which was described in [kap] was not a great success, although it showed some interesting results.. I therefore chose to evaluate the first attempt, make some changes and make a second evaluation. Based on those two evaluation, we can attempt to describe possible ways of improving the algorithm. I will focus on the function set of the GP. There were other weaknesses as well, involving both architecture and implementation, but since it was the function set that was altered before the second evaluation, I will start there.

The function set consisted of functions to increment or decrement a “counter” which decided what the weight-index corresponding to the bins in the different histograms that was “active”. In addition to that, there were functions to increment or decrement the value of each weight, using the counter-variable as an index. Now, the problem with this method is that the algorithm have no way of analyzing the histogram in any way. This leads the algorithm to randomly push any button (that is, shift weights) blindly, in hope of that will work. That, in theory at least, is not a big problem. Sooner or later the GP will home into a viable solution. The problem with doing things in real life is that all sorts of other problems emerge. In this case it was the lack of enough computer power. After about 90 generations all the physical memory was in use, forcing the system to run on virtual memory.

The longest run was with a population of 100, and a maximum of 150 generations. It lasted 48 hours excluding the occasional random shutdowns. In this case I experimented with using a step-limit of about 40. That means that if the number of instructions inside NotFinished() is below the step-limit, it will reiterate the code. The step-limit gets increased for each executed instruction, and is checked every iterations. This also means that it is no actual step-limit as it is more of a minimum-step-limit. The number of instructions inside the loop can be as high as it wants, but will stop after 1 iteration if the number of instructions is higher than the step-limit. The training set was a “forest”-theme with 8 training images.

The best individual in the first generation produced a distance of 999987,895 (best distance possible was 1000000). The code for this one was

```
public class Operator_gp_Gen1_Ant14 : Operator_gp {
```

```

public override void Execute() {
    for (
        ; NotFinished();
    ) {
        incHH();
        decHHue();
        incHSat();
        decHSat();
    }
    evaluateDistance();
    setRelevant();
    stopClock();
}
}

```

Function descriptions:

- incHH() increments the *counter* for the Hue histogram in Whole image (*Hele* in Norwegian)
- decHHue() decrements the weight in the whole image's hue-histogram at location *counter*
- incHSat() increments the weight in the whole image's saturation-histogram at location *counter*
- decHSat() decrements the weight in the whole image's saturation-histogram at location *counter*

Already in the first generation we see redundant code. The incHSat() and decHSat() cancel each other out, but since this one produced the least distance, the code will probably be preserved for the next generation.

The weights for the whole image generated by this algorithm was

```

H: 1, 0,9, 0,9, 0,9, 0,9, 0,9, 0,9, 0,9, 0,9, 0,9, 1, 1, 1, 1, 1, 1, 1, 1
S: 1, 1, 1
V: 1, 1, 1

```

It was at a too early stage to try to extract any useful information from this, so let's jump to the best individual of the whole run, which is from the 147'th generation.

This individual produced a fitness-score of 999999,796. A definitive improvement of the 999987,895 score after 1 generation. The code for this individual can be found in the appendix (it is too large to be put here). It is meant as an example of how difficult it is to read genetic code.

The weights this individual produced was interesting, since it didn't produce the results I was expecting. I would guess it would either increment or decrement the weights that corresponded to the common colors in the training set, or increment or decrement the other weights that didn't

correspond. The weights for the whole image generated by this individual was

```
H : 1, 1, 1, 0,9, 0,9, 0,2, -0,7, -1,2, 0,5, 0,5, 1, 1, 1, 1, 1, 1, 1, 1
S : 0,3, 0,9, 1
V : 1,1, 0,9, 1,1
```

Now, the dominant color for a typical forest image is located in bin 5, 6 and sometimes 7. This includes different shades of green and yellow. Bin 5 is more or less untouched, while bin 6, 7, 8, 9 and 10 are decreased. I spend some time wondering about this, and try to utilize the classifier to check if that worked (only using the histogram for the whole image). The result of this was that about 1/3 of the database got classified as “forest”, which was obviously wrong. Finally I discovered that the problem lay with the training set. One of the images contained a forest with a field of blue-ish flowers of in front. The flowers were low in intensity, so they were actually hard to spot, but it resulted in a histogram with green and blue as dominant colors. I replaced the picture with three others, and run the GP again.

The new best individual produced these weights:

```
H : 1, 1, 1, 0,9, 0,6, 0,1, -1,6, 0,2, 0,7, 1, 1, 1, 1, 1, 1, 1, 1, 1
S : 3, 3, 1
V : 1,1, 1,1, 1,1
```

Here we see an improvement from the last run, and the weights are more concentrated on the green area. A peculiar shift in the Saturation weight also happened, and I still have no good explanation for this. It is of some importance to know that the initial state of the counters is the middle bin of each histogram. As we can see from both weight sets, it looks like the GP is incrementally decreasing the values around it, in this case with a small shift to the left. I ran this GP a third time, this time with the counters pointing to the first element.

This time it showed some interesting results.

Half way in (the 80'th generation), the best individual had this weight set:

```
H : 0,6, 0,3, -0,3, -0,2, 0,8, 0,8, 1,2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
S : 3, 3, 1
V : 1,1, 0,9, 1,1
```

We see that the “lowest point” has moved from the far left to right, maybe indicating that it will end up and perhaps stop in the “forest-color” area.

When finished with 150 generations, the best individual had its lowest point in the green-yellow sector, but that doesn't necessary mean anything. To be sure, we would have to let it run for maybe another 100 generations and evaluate the result, but that would probably lead to the ultimate

blow to my computer equipment.

The last run I made in this first test was with the hue-counter placed directly in the green area of the histogram. This time it staid in place when the run was over. Best individual generated weights for the final test-run was

H : 1, 1, 1, 0,3, -0,2, -1,1, 0,5, 0,9, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
S : 3, 3, 1
V : 1,1, 1,1, 1,1

Now I was reasonably convinced that the GP would target the dominant colors in some way, but the classification tests still generated very low accuracy. Something had to be done.

5.3 Modification of the GP function set

Learning from the experience of the first test, I decided to add some extra functionality to the GP. Since I had deduced that the GP seems to be partial to dominant colors, I added 6 functions for this. The first three had the task of increasing the weight of the first-, second-, and third-most dominant hue-bin in the histogram. The last three had the same task, but would decrease the values.

In order to be fair, I also added the same six functions for the lowest, second-lowest and third-lowest value. These extra functions forced me to drop the number of generations, but I reasoned that the GP would come with a solution faster, so it would hopefully add up. I also added the segmented histograms into the equation, but didn't have time to implement extra functions for them. The number of generations was dropped to 90. I will now list the best individual's weights for several different retrieval algorithms to see if they generate different weights. The weights of the segmented picture didn't change significantly either directions, probably because the function set for operating these wasn't sophisticated enough to compete with the functions for the whole image (that is, the twelve added functions), and therefore got squeezed out of the population. Therefore I will not list those.

Weights for Intersection – Euclidian:

H : 1, 1, 1, 1, 1,2, 2,8, 2,5, 1, 1, 2, 1, 1, 1, 1, 4,7, 2, 1, 1
S : 1, 1, 1
V : 1, 1, 1

Weights for Euclidian – Euclidian

H : 1, 1, 1, 1, 1,2, 1,6, 1,6, 1, 1, 1, 1, 1, 1, 1, 0,6, 1, 1, 1
S : 1, 1, 1
V : 1, 1, 1

Weights for Quadratic – Euclidian

H : 1, 1, 1, 1, 1, 2, 1, 6, 1, 6, 1, 1, 1, 1, 1, 1, 1, 1, 0, 5, 1, 1, 1,
S : 1, 1, 1
V : 1, 1, 1

Here we see that the GP is behaving as one might suspect it would. Here we see that the dominant colors are increased, and in the two latter cases, one of the least used colors are decreased. It is a bit peculiar to observe that the first case actually increases the weight of the same color. The color is, by the way, the third lowest color. Why they all decide to change the third-lowest color, I really could not say. A small surprise may be that the Quadratic and Euclidian algorithm produces almost the exact same weights in this example. So now we have a GP who more or less behaves as we expect it to do (I will discuss the connection between expectations and results later in the paper), we can move on to the classification.

5.4 Classification

Weights for all the different themes were made, using the GP discussed in last chapter. They all followed the same characteristics at the “forest” example, namely raising the most common colors, and either rising or lowering one or two of the least used colors. When using the intersection-euclidian algorithm, the tendency to raise the weights of the colors with little representation persisted, although in some cases, it lowered these. Besides that, it produced higher weights than the two other algorithms. The Euclidian – Euclidian and Quadratic – Euclidian lowered one or more of the lowest color values every time.

5.4.1 Classification tryout and evaluation

So, if we remember how the classification routine was constructed, that is, if an image is within the limits of the *minBoundary* and *maxBoundary*, it is said to belong to that class. The way the GP generates the weights, that is, increasing colors that are heavily used in the images, will lead to higher class number to those images who has those dominate colors. There is some variance to what happens to the least-used colors (increasing or decreasing), which could lead to some issues, but we will disregard that for now. There is also a possibility to remove the *maxBoundary*, since any image which get a higher score, is even more dominant in the respective colors, and is therefore probably a member of that class.

Since time was running short, a thorough classification experiment wasn't an option. I decided to define a set of images that was defined as part of that class, and divided these into two parts; one considered hard and one considered easy. I also defined a set of images as not a part of that class, and did the same division. I define an easy image as an image who is very close in color composition compared to the training set. Likewise, a hard image, is an image with unusual colors for that particular class, or it may be a picture where the class is only a part of the image. I don't expect the classifier to perform especially well on that particular task.

Example:

Winter





In class, easy	In class, hard	Not in class, easy	Not in class, hard
			

fig 33: Winter test set

Forest

In class, easy	In class, hard	Not in class, easy	Not in class, hard
			

fig 34: Forest test set

Buildings/Cities

In class, easy	In class, hard	Not in class, easy	Not in class, hard
			

fig 35: Buildings/Cities test set

Deserts

In class, easy	In class, hard	Not in class, easy	Not in class, hard
			

fig 36: Deserts test set

Mountains

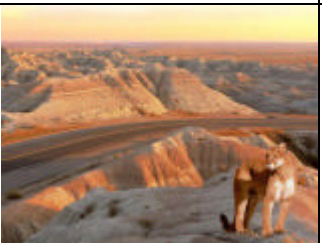



In class, easy	In class, hard	Not in class, easy	Not in class, hard
			

fig 37: Mountains test set

I then ran the classification routine on the images (10 in each subset), and calculated the accuracy (that is, how many were classified correct).

Classified correctly:

<i>Euclidian:</i>	In class, easy	In class, hard	Not in class, easy	Not in class, hard
Winter	80%	10%	100%	50%
Forest	90%	0%	100%	30%
Building/Cities	60%	10%	100%	60%
Deserts	70%	0%	100%	60%
Mountains	30%	0%	100%	80%
<i>Intersection:</i>				
Winter	80%	0%	100%	50%
Forest	90%	0%	100%	30%
Building/Cities	60%	10%	100%	70%
Deserts	60%	0%	100%	70%
Mountains	30%	0%	100%	80%
<i>Quadratic:</i>				
Winter	80%	10%	100%	50%
Forest	90%	0%	100%	30%
Building/Cities	60%	10%	100%	60%
Deserts	70%	0%	100%	60%
Mountains	30%	0%	100%	80%

Table 3: Accuracy of classification

As we can see in this table, the classification method proposed is definitely best suited to tell what class an image doesn't belong in. It looks like it also performs reasonably well on "easy" classifications. Hard images belonging to the class didn't classify at all. We see a correlation between the classes that didn't classify their own (easy) images well, particularly the mountain class, that they do well in defining that an image is not of that class. One may say they are too strict. There is virtually no difference in effect between using Euclidian distance algorithm and quadratic.

There is some small impact of the heavier weighting in the intersection algorithm, but not to an apparent positive effect.

6 Conclusion

This chapter will evaluate the capabilities of the result, whether it has fulfilled the hypothesis, and what improvements that can and should be made. After that, I will discuss how well the work-process has worked for me during this study.

6.1 Evaluation and discussion of the classification system

The system that is proposed, is highly experimental, and was an attempt to utilize known distance algorithms in an area that they, as far as the writer is aware of, have not been utilized before. This project was also to combine Genetic Programming with these algorithms, to enable classification of simple classes. This has been achieved, and a working classifier based on distance measures and GP-generated weights have been developed. There is, however, a difference between working and working *well*, which I will be the first to admit. Since developing a system like this is, for me, breaking new ground all the way, errors and misconceptions have been a trusty companion all the way. I'm pleased to see that the system actually is able to demonstrate what the hypothesis said, but I would give my right leg for another six months of making this a real system and more time to collect and analyze the different aspects of it. But *tempus fugit*, and the task has been delimited as much as I could bear along the way.

The classification relies heavily on the training set to perform properly. The more coherent groups of training images you use, the less is the chance classifying something that isn't of that class. But at the same time you limit the possibility of classifying variations of that particular class. This is the case with my training set. Color alone does not contain enough information to insure proper classification. At the very least, some spatial properties should be added in the equation and preferably texture features as well. The genetic program is crude, not tree-based, although the possibility is present and doesn't have support for mutation. A restructuring of the genetic generator would be an option. The function set is perhaps the most critical point. The GP improved vastly after adding "smarter" functions, and could perhaps be used as a test bed for evaluating competing "smart" solutions to problems.

I had also planned a more extensive analysis of the different retrieval methods, and is a bit disappointed to acknowledge that the segmented image features didn't get the attention they needed to function properly.

In the hypothesis I mentioned the probability that the GP will take advantage of dominant color, and that the weights will be altered to adjust these particular color areas. That the hypothesis came true is not necessary a sign of that being the best way, but more that the implementer of the system saw this as the best opportunity. To get some results, I had to take some short-cuts, like adding the twelve extra functions in the GP function set, which enabled the GP to easily modify the weights for the most and least used colors. I did in no way *force* the GP to take advantage of this option. I gave the GP a choice, which more or less was “*use these methods, or keep on stumbling in the dark. It’s up to you*”. Apparently it wasn’t a hard choice.

The classifier itself is also quite primitive. The idea was to demonstrate the use of histograms and weights, and not “cheat”, by using advanced classification/clustering methods, although it probably would have led to a better result. After all, this was an experiment, not an attempt to build an optimal image classification system.

6.2 Evaluation of the work-process

This master thesis is one of the most difficult projects I have ventured on. The difficulty was not the technical or theoretical aspects. I like to develop, and I like to learn, and when I first had decided on my task, it was at times thrilling to see the first results. My first problem was to choose a problem area, and more precise, a problem or theory to explore. Being more of a technician than a scientist, I was frustrated over the restrictions and demands such a thesis made. It took me a long time to find a suitable and “unexplored” problem area. The area I chose to explore was something I have never done or barely heard of before, and that caused an underestimation of the time I had to spend learning, understanding and creating/testing. That caused the actual writing to be pushed further and further down the road, resulting in a race to the finish. A more delimited task would perhaps been more appropriate, but even so, I don’t really regret opening my mouth too wide. I have learned an incredible amount of stuff in a wide range of areas during my time as a master candidate, and if I had delimited the thesis to a small problem in a familiar area, I don’t think I would have possessed the knowledge I do now.

References

- [1] L. Guojun, “*Multimedia Database Management Systems*”, 1999, chapter 1, 5
- [2] B. Conolly, “*Genetic Algorithms: Survival of the fittest: Natural Selection with Windows Forms*”, 2004,
web: <http://msdn.microsoft.com/msdnmag/issues/04/08/GeneticAlgorithms/>
- [3] G. Hartvigsen, “*Forskerhåndboken*”, 1998
- [4] James Z Wang, “*Integrated Region based Image Retrieval*”, Boston, Kluwer Academic Publishers, 2001
- [5] R.C. Gonzalez & R.E Woods, “*Digital Image Processing*” 2. ed., 2002, chapter
- [6] Tamura, H., S. Mori, T. Yamawaki, “*Texture Features Corresponding to Visual Perception*”, IEEE Transactions on Systems, Man, and Cybernetics, Vol 17, No.1 Jan 1995, page 72-77
- [7] James Z Wang “*Integrated Region based Image Retrieval*”, Boston, Kluwer Academic Publishers, 2001
- [8] QBIC,
web: <http://www.qbic.almaden.ibm.com/>
- [9] R. Ghanea-hercock, “*Applied Evolutionary Algorithms in Java*”, 2003, chapter 1 - 4
- [10] Britannica,
web: <http://www.britannica.com>
- [11] Holland J, “*Adaption in Natural and Artificial Systems: An introductory Analysis with Applications to Biology*”, reprint 1992
- [12] D. Whitley, “*A Genetic Algorithm Tutorial*”, Journal of Statistics and Computing, Vol 4, page 65-85, 1994
- [13] W. Banzhaf, P. Nordin, R.E. Keller & F.D Francone, “*Genetic Programming, an Introduction, On the automatic Evolution of Computer Programs and Its Applications*” ,1998
- [14] Davide Agnelli, Alessandro Bollini and Luca Lombardi “*Image classification: an evolutionary approach*”, 2001
web: <http://www.sciencedirect.com/science/article/B6V15-443K10X-6/1/7af8206767ca79f9898fec720a84c656>

[15] Xiang Li and Vic Ciesielski, “*Using Loops in genetic Programming for a two class binary image classification problem*”, 2004,

web: <http://goanna.cs.rmit.edu.au/~vc/papers/aust-ai04.pdf>

[16] Will Archer Arentz and Bjørn Olstad, “*Classifying offensive sites based on image content*”, Computer Vision and Image Understanding 94 (2004) p. 295-310, 2003

web: http://www.idi.ntnu.no/~willa/papers/site_classifier.pdf

[17] Vailaya, A., Figueiredo, M.A.T., Jain, A.K., Zhang, H.J., “*Image Classification for Content-Based Indexing*”, *IEEE transactions on image processing*, No. 1, January 2001, page. 117-130,

web: <http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=892448>

[18] B. Li, L.-S Goh and E. Y Chang “*Confidence based Dynamic Ensemble for Image Annotation and Semantics discovery*” ACM Multimedia, 2003,

web: <http://www.mmdb.ece.ucsb.edu/~echang/acm-de.pdf>

[19] J.Li, J.Z Wang, “*Automatic Linguistic Indexing of Pictures by a statistical modeling approach*”, *IEEE Transactions on pattern analysis and machine intelligence*, vol 25, no. 9, 2003,

web: <http://www-db.stanford.edu/~wangz/project/imsearch/ALIP/PAMI03/01227984.pdf>

[20] Blobworld,

web: <http://elib.cs.berkeley.edu/blobworld/>

[21] Hermitage Web-site,

web: <http://www.heritagemuseum.org/cgi-bin/db2www/qbicSearch.mac/qbic?selLang=English>

[22] VisualSEEK,

web:

<http://www.ee.columbia.edu/dvmm/researchProjects/MultimediaIndexing/VisualSEEk/VisualSEEk.htm>

[23] WebSEEk,

web:

<http://www.ee.columbia.edu/dvmm/researchProjects/MultimediaIndexing/WebSEEk/WebSEEk.htm>

[24] A. W. Smeulders, M. Worring, S Santini, A. Gupta and R. Jain “Content Based Image Retrieval at the end of the Early Years”, IEEE Trans. Pattern Analysis and Machine Intelligence, 2000,

web: <http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/34/19391/00895972.pdf>

[25] N. Sebe, M.S Lew, “A maximum likelihood Investigation into Color Indexing”, Leiden Institute of Advanced Computer Science, 2003

web: <http://staff.science.uva.nl/~nicu/publications/vi.pdf>

[26] R. Baeza-Yates & B. Ribeiro-Neto, “*Modern Information Retrieval*” (1999), chapter 1-5

[27] : M. Buckland, “ *What is a “digital document”?* ”, 1998,

web: <http://www.sims.berkeley.edu/~buckland/digdoc.html>

[28] Google Corporate Information

web: “<http://www.google.com/corporate/history.htm>”

APPENDIX A – source code

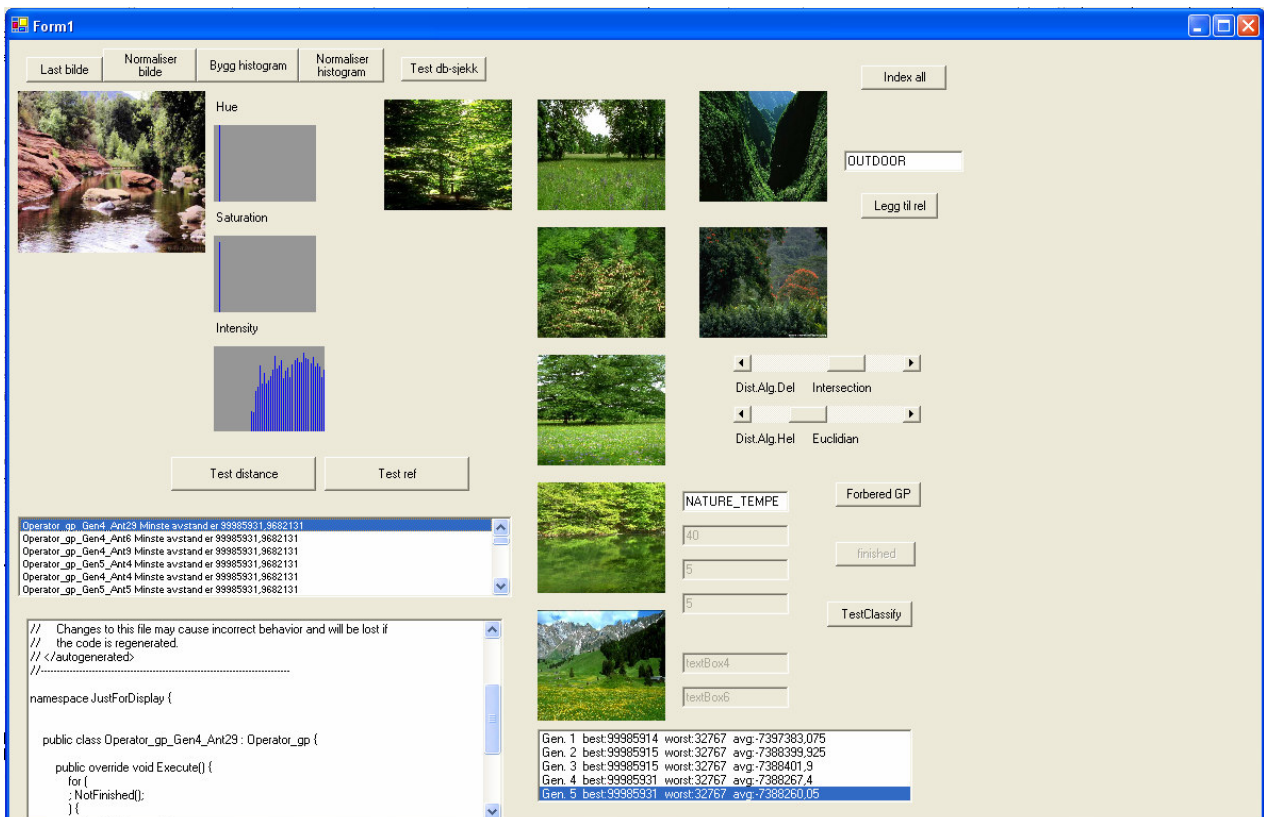
Implementation notes

This system was implemented in C#, due to the strong object-oriented nature of the language, its speed and its garbage collector. The database used was Oracle Express, with simple tables where different training sets (images) could be constructed.

The GP generator is built on a modified version of [2], which was made as a demonstration of Genetic Programming. It utilizes CodeDOM – Code Document Object Model, which have capabilities such as in-system compiling and running. This means that the individuals in the GP is stored as .cs files, and compiled to .dll before run. The advantage of this is that you have the source code as well as the .dll to each genome available.

Due to the excessive experimentation with this code, it has lots of redundant methods, constants and parameters which partly must take the blame for the heaviness of the program. I didn't remove these redundancies, since it's still not a finished system, and may be of interest. The commentary is not the best, and is a mix of Norwegian and English.

GUI:



GUI class (form1.cs):

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Data.OracleClient;
using System.IO;
using System.Reflection;

using System.CodeDom;
using System.CodeDom.Compiler;
using Microsoft.CSharp;

namespace Classify_tryout
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {

        private Operator_gp gp; // The GP

        string klas; // klassifikasjonsnavn

        private Generator mGenerator;
        private int mCurrentGeneration = 0;

        // holds the results from the last generation
        private ArrayList mLastGenerationResults = new ArrayList();

        // used for the step function which walks the selected
        // ant through its history and displays the grid path
        private ArrayList mSelectedGPHistory;
        //private Operator_gp.TrailSpace mSelectedAntLastHistory;

        // grid file loaded at startup
        //private string mGridFileName = "DefaultGrid.xml";

        // provider and generator used to display code for
        // a particular selected ant.
        CodeDomProvider mCodeprovider;
        ICodeGenerator mCodegen;

        // this is the Ant data that is saved in listBox1
        private class GPDisplayItem
        {
            public Generator.TypeExecutionResult mData;
            public string mTypeClassName;
            public GPDisplayItem
                (string GPClassName,
                 Generator.TypeExecutionResult Data)
            {
                mData = Data; mTypeClassName = GPClassName;
            }
            // for listBox1 row content
            public override string ToString()
            {
                return mTypeClassName + " Minste avstand er " +

```

```

        mData.mResults.mRelevant.ToString();// +
        //" relevante bilder på " +
        //mData.mResults.mTime.ToString() +
        //" sekunder";
    }
}

private System.Windows.Forms.Button btnLastopp;
public System.Windows.Forms.PictureBox pictureBox1;
public System.Windows.Forms.PictureBox picHRed;
private System.Windows.Forms.PictureBox picHGreen;
private System.Windows.Forms.PictureBox picHBlue;
private System.Windows.Forms.Label label1;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Button button1;
private System.Windows.Forms.Button button2;
private System.Windows.Forms.Button button3;
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.Container components = null;
private System.Windows.Forms.ListBox listBox1;
private System.Windows.Forms.Button button4;
private System.Windows.Forms.Button button5;
private System.Windows.Forms.Button button6;
//private System.Data.OracleClient.OracleConnection oracleConnection1;
private System.Data.OracleClient.OracleDataAdapter oracleDataAdapter1;
private System.Data.OracleClient.OracleCommand oracleSelectCommand1;
private System.Data.OracleClient.OracleCommand oracleInsertCommand1;
private System.Data.OracleClient.OracleCommand oracleUpdateCommand1;
private System.Data.OracleClient.OracleCommand oracleDeleteCommand1;
private System.Data.OracleClient.OracleConnection connection;
private System.Windows.Forms.Button button7;

private Operator processing;
private System.Windows.Forms.Button button8;
private System.Windows.Forms.PictureBox pBres1;
private System.Windows.Forms.PictureBox pBres2;
private System.Windows.Forms.PictureBox pBres3;
private System.Windows.Forms.PictureBox pBquery;
private System.Windows.Forms.Label lblProcessed;
private System.Windows.Forms.HScrollBar hScroll;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.PictureBox picWorst1;
private System.Windows.Forms.PictureBox picWorst2;
private System.Windows.Forms.Label label5;
private System.Windows.Forms.Label lblAlgD;
private System.Windows.Forms.Label lblAlgH;
private System.Windows.Forms.HScrollBar hScrollH;
private System.Windows.Forms.PictureBox pBres5;
private System.Windows.Forms.PictureBox pBres4;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.TextBox textBox3;
private System.Windows.Forms.TextBox textBox4;
private System.Windows.Forms.TextBox textBox5;
private System.Windows.Forms.TextBox textBox6;
private System.Windows.Forms.Button btnPrepGP;
private System.Windows.Forms.RichTextBox richTextBox1;
private System.Windows.Forms.ListBox listBox2;
private System.Windows.Forms.Button btnGenerate;
private System.Windows.Forms.TextBox txtDBnavn;
private System.Windows.Forms.Button btnAddrel;
private System.Windows.Forms.Button button9;
private System.Windows.Forms.Button button10;
private DbManagement db;

```



```

public Form1()
{
    //
    // Required for Windows Form Designer support
    //
    InitializeComponent();

    //
    // TODO: Add any constructor code after InitializeComponent call
    //
    processing = new Operator( ); //histogram-operations
    db = new DbManagement();// the database

}

/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
        base.Dispose( disposing );
    }
}

#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
    // Fjernet for å ikke bruke så mange sider.
}
#endregion

/// <summary>
/// The main entry point for the application.
/// </summary>
[STAThread]

static void Main()
{
    Application.Run(new Form1());
}

private class TestGP : Operator_gp
{
    public override void Execute()
    {
    }
}

private void Empty_GP(string[] dbHistos,Operator.histogrammer[]
histogrammer, string[] relHistos, int[,,,] pH, int[,,,] pS, int[,,,] pI, int[]
wH, int[] wS, int[] wI, int StepLimit, int RelGoal, Operator op,
DbManagement database, string klassifik)
{
    int mRelevante = 10;
}

```

```

int mTid = 0;
int mMinimumSteps = 3;

textBox4.Text = mRelevante.ToString();
textBox6.Text = mMinimumSteps.ToString();

gp = new TestGP();
gp.Init(hScrollH.Value, hScroll.Value, dbHistos, histogrammer,
relHistos, pH, pS,pI, wH, wS, wI,StepLimit, RelGoal, op, /*database,*/
klassifik);

mGenerator = new Generator(typeof(Operator_gp), 3);

// provider and generator used to display code for
// a particular selected ant.
mCodeprovider = new CSharpCodeProvider();
mCodegen = mCodeprovider.CreateGenerator();
}

private void SetGoalsFromGrid()
{
// set default goal to the number of grid squares with food
// and the step goal to the number of grid squares
// with either food or a gap
int mRelevant = 10;
int mMinimumSteps = 10;

textBox4.Text = mRelevant.ToString();
textBox6.Text = mMinimumSteps.ToString();
}

private void btnLastopp_Click(object sender, System.EventArgs e)
{
OpenFileDialog filechooser = new OpenFileDialog( );
filechooser.RestoreDirectory = true;

if( filechooser.ShowDialog( ) == DialogResult.OK )
{
this.pictureBox1.Image = new Bitmap( new
Bitmap(filechooser.FileName ) );

this.Invalidate( );
}

this.processing.setImage( new Bitmap( this.pictureBox1.Image ) );
}

private void button2_Click(object sender, System.EventArgs e)
{
this.processing.calcHisto( picHRed, picHGreen, picHBlue );
this.pictureBox1.Image = this.processing.getImage( );

//this.menuItem5.Enabled = true;
}

private void button3_Click(object sender, System.EventArgs e)
{
int [] histoH = new int [processing.hueBins];
int [] histoS = new int [processing.satBins];
int [] histoI = new int [processing.intBins];

String str;
str = "";

this.processing.equilizeHist( picHRed, picHGreen, picHBlue, out
histoH, out histoS, out histoI, true );
this.pictureBox1.Image = this.processing.getImage( );
}

```

```

listBox1.Items.Add("HUE");
listBox1.Items.Add("01 02 03 04 05 06 07");
for (int i = 0; i<processing.hueBins; i++) //test
    str += histoH[i].ToString()+ " - ";

listBox1.Items.Add(str);

str="";
listBox1.Items.Add("SAT");
listBox1.Items.Add("01 02 03 04 05");
for (int i = 0; i<processing.satBins; i++)
    str += histoS[i].ToString()+ " - ";

listBox1.Items.Add(str);

str="";
listBox1.Items.Add("INT");
listBox1.Items.Add("01 02 03 04 05");
for (int i = 0; i<processing.intBins; i++)
    str += histoI[i].ToString()+ " - ";

listBox1.Items.Add(str);

    this.Invalidate( );
}

private void button4_Click(object sender, System.EventArgs e)
{
    // Loop through and add 50 items to the ListBox.
    //listBox1.Items.Add("Item ");
    for (int i = 0; i < 50; i++)
    {
        listBox1.Items.Add("Item " + i.ToString());
    }
}

private void button5_Click_1(object sender, System.EventArgs e)
{
    int [] qHistoH = new int [processing.hueBins];
    int [] qHistoS = new int [processing.satBins];
    int [] qHistoI = new int [processing.intBins];

    int [] tHistoH = new int [processing.hueBins];
    int [] tHistoS = new int [processing.satBins];
    int [] tHistoI = new int [processing.intBins];

    Bitmap q = new Bitmap("C:\\Bilder\\q.jpg");
    Bitmap t = new Bitmap("C:\\Bilder\\t.jpg");

    processing.equilizeHist(q, out qHistoH, out qHistoS, out qHistoI);
    processing.equilizeHist(t, out tHistoH, out tHistoS, out tHistoI);
    MessageBox.Show("Distance =
"+processing.distance(qHistoH,qHistoS,qHistoI, tHistoH, tHistoS,tHistoI,
hScroll.Value).ToString());
}

private void button6_Click(object sender, System.EventArgs e)
{
    int divX=8;
    int divY=8;

    Bitmap q = new Bitmap("C:\\Bilder\\q.jpg");
    Bitmap t = new Bitmap("C:\\Bilder\\t.jpg");
    int [,] histoqH = new int [divX, divY, processing.hueBins];

```

```

//Delhistogram
int [,] histoqS = new int [divX, divY, processing.satBins];
int [,] histoqI = new int [divX, divY, processing.intBins];

int [,] histotH = new int [divX, divY, processing.hueBins];
int [,] histotS = new int [divX, divY, processing.satBins];
int [,] histotI = new int [divX, divY, processing.intBins];

int [] qHistoWH = new int [processing.hueBins]; //Hist. over hele
bildet
int [] qHistoWS = new int [processing.satBins];
int [] qHistoWI = new int [processing.intBins];

int [] tHistoWH = new int [processing.hueBins];
int [] tHistoWS = new int [processing.satBins];
int [] tHistoWI = new int [processing.intBins];

processing.equilizeHist(q, out qHistoWH, out qHistoWS, out qHistoWI);
processing.equilizeHist(t, out tHistoWH, out tHistoWS, out tHistoWI);
//Distance hele bildet
MessageBox.Show("Distance =
"+processing.distance(qHistoWH,qHistoWS,qHistoWI, tHistoWH, tHistoWS,tHistoWI,
hScroll.Value).ToString());

processing.segmentAndEq(q, divX, divY, ref histoqH, ref histoqS, ref
histoqI);
processing.segmentAndEq(t, divX, divY, ref histotH, ref histotS, ref
histotI);

//Distance "beste-match" med delbilder
MessageBox.Show("Distance =
"+processing.bestDistance(histoqH,histoqS,histoqI, histotH,histotS,histotI,
hScroll.Value).ToString());

db.open();
MessageBox.Show(db.add_histogram("testhisto.jpg", histoqH, histoqS,
histoqI, qHistoWH, qHistoWS,qHistoWI, processing).ToString());
}

public void Spørring(string mySelectQuery)
{
OracleCommand myCommand = new OracleCommand(mySelectQuery,
connection);
connection.Open();
OracleDataReader myReader =
myCommand.ExecuteReader(CommandBehavior.CloseConnection);
while(myReader.Read())
{
MessageBox.Show(myReader.GetString(1));
}
myReader.Close();
}

private int index_all(bool drop_before)
{
int divX = 8;
int divY = 8;
int res =0;

db.open();
if (drop_before)
{
res=db.executeCommand("DELETE FROM INDICES");
}
else res = -1;
}

```

```

        string[] files = Directory.GetFiles(db.bilde_kat, "*.jpg");

        int [,] histobH = new int [divX, divY, processing.hueBins];
//Delhistogram
        int [,] histobS = new int [divX, divY, processing.satBins];
        int [,] histobI = new int [divX, divY, processing.intBins];

        int [] histobWH = new int [processing.hueBins]; //Hist. over hele
bildet
        int [] histobWS = new int [processing.satBins];
        int [] histobWI = new int [processing.intBins];

        for (int i =0; i< files.Length; i++)
        {
            lblProcessed.Text=files[i];
            this.Update();

            Bitmap b = new Bitmap(files[i]);

            processing.equilizeHist(b, out histobWH, out histobWS, out histobWI);

            processing.segmentAndEq(b, divX, divY, ref histobH, ref histobS, ref histobI);

            db.add_histogram(files[i], histobH, histobS, histobI, histobWH, histobWS, histobWI,
            processing);

            b.Dispose();
        }

        db.close();
        return res;
    }

    private void find_n(string bitmapFileName) //Finner nærmeste bilde i DB
    {
        int divX = 8;
        int divY = 8;

        Bitmap b = new Bitmap(bitmapFileName);

        this.pbQuery.Image = new Bitmap(bitmapFileName);

        int [,] histoqH = new int [divX, divY, processing.hueBins];
//Delhistogram til spørring
        int [,] histoqS = new int [divX, divY, processing.satBins];
        int [,] histoqI = new int [divX, divY, processing.intBins];

        int [,] histoH = new int [divX, divY, processing.hueBins];
//Delhistogram
        int [,] histoS = new int [divX, divY, processing.satBins];
        int [,] histoI = new int [divX, divY, processing.intBins];

        int [] histoWH = new int [processing.hueBins]; //Hist. over hele
bildet
        int [] histoWS = new int [processing.satBins];
        int [] histoWI = new int [processing.intBins];

        int [] histoqWH = new int [processing.hueBins]; //Hist. over hele

```

```

bildet for spørring
    int [] histoqWS = new int [processing.satBins];
    int [] histoqWI = new int [processing.intBins];

    processing.equilizeHist(b, out histoqWH, out histoqWS, out histoqWI);

    processing.segmentAndEq(b, 8, 8, ref histoqH, ref histoqS, ref
histoqI);

    float tmpDistW = 2; //setter disse til en uoppnåelig distanse.
    float tmpDistP = 2;
    float distW = 2;
    float distP = 2;

    string [] res;
    string histo_nameP="";
    string histo_nameW="";

    db.get_histograms(out res);

    int debug_size = 150;
    //float [] comb = new float [res.Length];
    float [] comb = new float [debug_size];
    //string [] index = new string [res.Length]; //brukes til Sort
etterpå
    string [] index = new string [debug_size];

    float bestCombDistance = 2;
    string bestCombHist = "";

//
    for (int i = 0; i<res.Length; i++)
    for (int i = 0; i<debug_size; i++)
    {
        processing.fileToHisto(res[i],out histoH,out histoS,out
histoI, out histoWH,out histoWS,out histoWI);

        tmpDistW = processing.distance(histoqWH,histoqWS,histoqWI,
histoWH,histoWS,histoWI, hScroll.Value);

        histo_nameW = ( tmpDistW < distW ) ? res[i] : histo_nameW;
        distW = ( tmpDistW < distW ) ? tmpDistW : distW;

        tmpDistP = processing.bestDistance(histoqH,histoqS,histoqI,
histoH,histoS,histoI, hScroll.Value);

        histo_nameP = ( tmpDistP < distP ) ? res[i] : histo_nameP;

        distP = ( tmpDistP < distP ) ? tmpDistP : distP;

        comb[i] = (tmpDistW + tmpDistP)/2 ;
        index[i] = res[i];

        bestCombHist = ( ((tmpDistW + tmpDistP)/2) < bestCombDistance)
? res[i] : bestCombHist;
        bestCombDistance = ( ((tmpDistW + tmpDistP)/2) <
bestCombDistance) ? ((tmpDistW + tmpDistP)/2) : bestCombDistance;
        listBox1.Items.Add(res[i]+" : "+comb[i].ToString());

    }

    Array.Sort( comb, index);

```

```

        pBres1.Image = new Bitmap(index[0].Replace(".his",""));
        pBres2.Image = new Bitmap(index[1].Replace(".his",""));
        pBres3.Image = new Bitmap(index[2].Replace(".his",""));

        picWorst1.Image = new Bitmap(index[index.Length-
1].Replace(".his",""));
        picWorst2.Image = new Bitmap(index[index.Length-
2].Replace(".his",""));

        string tmp = "";
        for (int i = 0; i<debug_size; i++)
            tmp +=index[i]+ " : "+ comb[i].ToString()+"\n";
    }

private void find_nearest(string bitmapFileName) //Finner nærmeste bilde i
DB
{
    //db.open();
    int divX = 8;
    int divY = 8;

    Bitmap b = new Bitmap(bitmapFileName);

    this.pBquery.Image = new Bitmap(bitmapFileName);

    int [,] histoqH = new int [divX, divY, processing.hueBins];
//Delhistogram til spørring
    int [,] histoqS = new int [divX, divY, processing.satBins];
    int [,] histoqI = new int [divX, divY, processing.intBins];

    int [] histoqWH = new int [processing.hueBins]; //Hist. over hele
bildet for spørring
    int [] histoqWS = new int [processing.satBins];
    int [] histoqWI = new int [processing.intBins];

    processing.equilizeHist(b, out histoqWH, out histoqWS, out histoqWI);

    processing.segmentAndEq(b, 8, 8, ref histoqH, ref histoqS, ref
histoqI);

    string [] histo_files;

    string [] res_navn;
    float [] res_dist;

    db.get_histograms(out histo_files);

    processing.find_nearest(histoqH, histoqS, histoqI,
histoqWH, histoqWS, histoqWI, histo_files, hScroll.Value, hScrollH.Value, out res_navn, out
res_dist);

    pBres1.Image = new Bitmap(res_navn[0].Replace(".his",""));

```

```

        pBres2.Image = new Bitmap(res_navn[1].Replace(".his", ""));
        pBres3.Image = new Bitmap(res_navn[2].Replace(".his", ""));
        pBres4.Image = new Bitmap(res_navn[3].Replace(".his", ""));
        pBres5.Image = new Bitmap(res_navn[4].Replace(".his", ""));

        picWorst1.Image = new Bitmap(res_navn[res_navn.Length-
1].Replace(".his", ""));
        picWorst2.Image = new Bitmap(res_navn[res_navn.Length-
2].Replace(".his", ""));

        string tmp = "";
        //for (int i = 0; i<debug_size; i++)
        for (int i = 0; i<res_navn.Length; i++)
            listBox1.Items.Add(res_navn[i]+" : "+res_dist[i].ToString());
    }

    private void button7_Click(object sender, System.EventArgs e)
    {
        OpenFileDialog filechooser = new OpenFileDialog();
        filechooser.RestoreDirectory = true;

        if( filechooser.ShowDialog() == DialogResult.OK )
        {
            find_nearest(filechooser.FileName);
        }
    }

    private void button8_Click(object sender, System.EventArgs e)
    {
        MessageBox.Show("rader slettet: "+index_all(true).ToString());
    }

    private void button9_Click(object sender, System.EventArgs e)
    {
    }

    private void hScroll_Scroll(object sender,
System.Windows.Forms.ScrollEventArgs e)
    {
        switch(hScroll.Value)
        {
            case -1:
                lblAlgD.Text="N/A";
                break;
            case 0:
                lblAlgD.Text="Euclidian";
                break;
            case 1:
                lblAlgD.Text="Intersection";
                break;
            case 2:
                lblAlgD.Text="Quadratic (cross)";
                break;
        }
    }

    private void hScrollBar1_Scroll(object sender,
System.Windows.Forms.ScrollEventArgs e)
    {

```



```

        switch(hScrollH.Value)
        {
            case -1:
                lblAlgH.Text="N/A";
                break;
            case 0:
                lblAlgH.Text="Euclidian";
                break;
            case 1:
                lblAlgH.Text="Intersection";
                break;
            case 2:
                lblAlgH.Text="Quadratic (cross)";
                break;
        }
    }

private void Form1_Load(object sender, System.EventArgs e)
{
    klas = textBox1.Text;
}

private void btnPrepGP_Click(object sender, System.EventArgs e)
{
    OpenFileDialog filechooser = new OpenFileDialog( );
    filechooser.RestoreDirectory = true;
    int divX = 10; int divY=10;
    int [,] pH = new int [divX, divY, processing.hueBins];
//Delhistogram til spørring
    int [,] pS = new int [divX, divY, processing.satBins];
    int [,] pI = new int [divX, divY, processing.intBins];

    int [] wH = new int [processing.hueBins]; //Hist. over hele bildet
for spørring
    int [] wS = new int [processing.satBins];
    int [] wI = new int [processing.intBins];

    string [] histofiles;
    string [] relHistos;

    db.get_histograms(out histofiles);
    db.getRelHistos(klas,out relHistos);

    Operator.histogrammer[] histogrammer;

    processing.getClassesFromHistos(histofiles, out histogrammer);

    for (int i =0; i<histogrammer.Length; i++)
        histogrammer[i].setNormHistos();

    Bitmap b;

    if( filechooser.ShowDialog( ) == DialogResult.OK )
    {
        b = new Bitmap(filechooser.FileName);

        processing.equilibizeHist(b, out wH, out wS, out wI);

        processing.segmentAndEq(b, 8, 8, ref pH, ref pS, ref pI);
    }

    gp = new TestGP();

    double[] wh = new double[18];
    double[] ws = new double[3];
    double[] wi = new double[3];

```

```

double th =0;
double ts =0;
double ti =0;

for (int i = 0; i<histofiles.Length; i++)
{
    th = processing.returnClassNr(histogrammer[i].nhistoWH,wh);
    ts = processing.returnClassNr(histogrammer[i].nhistoWS,ws);
    ti = processing.returnClassNr(histogrammer[i].nhistoWI,wi);

    MessageBox.Show(histofiles[i]+": "+Math.Sqrt(th*th + ts*ts +
ti * ti).ToString());

}

gp.Init(hScrollH.Value, hScroll.Value, histofiles,histogrammer,
relHistos, pH, pS, pI, wH, wS, wI, 400,10,processing, /*db*/ klas);

// pass the base class we will generate execute
// methods for. In addition to its "FacingFood" function
// generate functions for 2 argumants and 3 arguments
mGenerator = new Generator(typeof(Operator_gp), 3);

// provider and generator used to display code for
// a particular selected ant.
mCodeprovider = new CSharpCodeProvider();
mCodegen = mCodeprovider.CreateGenerator();

}

private Generator.TypeExecutionResult NewGeneration()
{
    int algH = gp.algH;
    int algP = gp.algP;
    int [,] pH = gp.histoPH;
    int [,] pS = gp.histoPI;
    int [,] pI = gp.histoPS;

    string [] histos = gp.histos;
    string [] rHistos = gp.relHistos;
    Operator.histogrammer[] histogrammer = gp.learning_histos;

    int [] wH = gp.histoWH;
    int [] wS = gp.histoWI;
    int [] wI = gp.histoWS;

    // Cell[,] mGrid = mAntTrailGrid.GetGrid();
    Generator.ExecutionResults mExecutionResults =
        mGenerator.NewGeneration
            (algH, algP, histos, histogrammer, rHistos,
pH,pS,pI,wH,wS,wI,400,10,processing, klas, mLastGenerationResults,
mCurrentGeneration,
Convert.ToInt16(textBox2.Text),
Convert.ToInt16(textBox3.Text));
//Convert.ToInt16(textBox4.Text));

    // higher fitness is better
    mExecutionResults.mTypeExecutionResults.Sort();
    // so we sort and reverso so the "best" ant is
    // in slot 0
    mExecutionResults.mTypeExecutionResults.Reverse();

    listBox2.Items.Add(mExecutionResults);
    listBox2.Refresh();
}

```

```

        mLastGenerationResults =
            mExecutionResults.mTypeExecutionResults;

        // the best ant
        return (Generator.TypeExecutionResult)
            mLastGenerationResults[0];
    }

private void btnGenerate_Click_1(object sender, System.EventArgs e)
{
    textBox2.Enabled = false;
    textBox3.Enabled = false;
    textBox4.Enabled = false;
    textBox5.Enabled = false;
    textBox6.Enabled = false;
    listBox1.Items.Clear();
    listBox2.Items.Clear();
    listBox2.Enabled = false;
    richTextBox1.Text="";
    btnGenerate.Enabled = false;

    mLastGenerationResults = new ArrayList();
    mCurrentGeneration = 1;
    // this is the best possible ant - the most food
    // in the minimum possible steps
    // we will stop the run before the specified
    // number of generations if we get an ant that is
    // this good.
    Operator_gp.TerminationReport goalReport =
        Operator_gp.TerminationReport.GoalReport
            (0); //mål
    int mBestAntFitness = Operator_gp.TerminationReport.Fitness
        (new
Operator_gp.TerminationReport(99999999, null, null, null, 99999, -99999, null, null, null));
    int mTargetFitness = Operator_gp.TerminationReport.Fitness
        (goalReport);
    while(
        ( mCurrentGeneration <= Convert.ToInt16(textBox5.Text))
        &
        // mBestAntFitness.CompareTo(mTargetFitness) <= 0
        mBestAntFitness.CompareTo(mTargetFitness) >= 0
        )
    {
        mBestAntFitness = Operator_gp.TerminationReport.Fitness
            (NewGeneration().mResults);
        mCurrentGeneration++;
    }

    btnGenerate.Text = "finished";
    listBox2.Enabled = true;

    // Save results summary in run.log
    using (StreamWriter sw = new StreamWriter("lastrun.csv"))
    {
        IEnumerator genListEnum =
            listBox2.Items.GetEnumerator();
        while(genListEnum.MoveNext())
        {
            (Generator.ExecutionResults) r =
                genListEnum.Current;
            sw.Write(r.ToLogString() + ",");

            Generator.TypeExecutionResult bestAnt =
                (Generator.TypeExecutionResult)
                    r.mTypeExecutionResults[0];

            sw.WriteLine(bestAnt.mResults.ToLogString());
        }
    }
}

```

```

    }

    // To_DO: Allow reinitialization and a new run
    // without restarting
    //btnGenerate.Enabled = true;
}

private void textBox2_TextChanged(object sender, System.EventArgs e)
{
}

private string BuildExecuteMethodForDisplay
(string ClassName, Generator.ExpressionTree Tree)
{
    CodeCompileUnit mDisplayCompileUnit =
        new CodeCompileUnit();
    CodeNamespace mDisplaySpace =
        new CodeNamespace("JustForDisplay");
    mDisplayCompileUnit.Namespaces.Add(mDisplaySpace);
    mDisplaySpace.Types.Add
        (mGenerator.BuildClass(ClassName, Tree));

    StringWriter sw = new StringWriter();
    // Create a TextWriter to a StreamWriter to an output file.
    IndentedTextWriter tw = new IndentedTextWriter
        (sw);
    // Generate source code using the code generator.
    mCodegen.GenerateCodeFromCompileUnit
        (mDisplayCompileUnit, tw, new CodeGeneratorOptions());

    return sw.ToString();
}

```

```

// display the lineage and the generated code
private void listBox1_SelectedIndexChanged
(object sender, System.EventArgs e)
{
    if (listBox1.SelectedItem is GPDisplayItem)
    {
        GPDisplayItem selected =
            (GPDisplayItem)listBox1.SelectedItem;
        //btnReplay.Enabled = true;
        richTextBox1.Text =
            "GP Name:" +
            selected.mData.mTypeClassName + "\n" +
            "Parent1: " +
            selected.mData.mParent1ClassName + "\n" +
            "Parent2: " +
            selected.mData.mParent2ClassName + "\n" +
            "Generated Code:" + "\n" +
            BuildExecuteMethodForDisplay
            (selected.mTypeClassName, selected.mData.mTree);
    }
    else
    {
        //btnReplay.Enabled = false;
        richTextBox1.Text = "";
    }
}

```

```

private void listBox2_SelectedIndexChanged(object sender, System.EventArgs
e)
{
    listBox1.Items.Clear();
    richTextBox1.Clear();
    listBox1.Refresh();

    Generator.ExecutionResults selectedItemResults =
        (Generator.ExecutionResults)listBox2.SelectedItem;
    IEnumerator iEnum =
        selectedItemResults.mTypeExecutionResults.

```

```

        GetEnumerator();

while(iEnum.MoveNext())
{
    Generator.TypeExecutionResult antResult =
        (Generator.TypeExecutionResult)iEnum.Current;

    listBox1.Items.Add(
        new GPDisplayItem
            (antResult.mTypeClassName, antResult));
}
listBox1.Refresh();

}

private void btnAddrel_Click(object sender, System.EventArgs e)
{
    OpenFileDialog filechooser = new OpenFileDialog();
    filechooser.RestoreDirectory = true;
    filechooser.Multiselect=true;

    if( filechooser.ShowDialog() == DialogResult.OK )
    {
        for (int i = 0; i<filechooser.FileNames.Length; i++)

            db.add_element(txtDBnavn.Text, (filechooser.FileNames[i].Replace("bilder", "BILDER")
+"".his"));

    }

}

public class testGP : Operator_gp
{

    public override void Execute()
    {
        for (
            ; NotFinished();
            )
        {
            incHighest();
            incHighest();
            incHH();
            decHI();
            incHighest();
            decHS();
            decSecLowest();
            incSecHighest();
            incHighest();
            decSecLowest();
            incHH();
            incHighest();
            incHH();
            incHH();
            incHighest();
            decSecLowest();
            decSecLowest();
            decHI();
            incThirdHighest();
            incThirdHighest();
            incThirdHighest();
            incThirdHighest();
            incSecHighest();
            incThirdHighest();
            incThirdHighest();
        }
    }
}
}

```

```

private void button9_Click_1(object sender, System.EventArgs e)
{
    OpenFileDialog filechooser = new OpenFileDialog();
    filechooser.RestoreDirectory = true;

    int divX = 10; int divY=10;
    int [,] pH = new int [divX, divY, processing.hueBins];
    int [,] pS = new int [divX, divY, processing.satBins];
    int [,] pI = new int [divX, divY, processing.intBins];

    int [] wH = new int [processing.hueBins]; //Hist. over hele bildet
    int [] wS = new int [processing.satBins];
    int [] wI = new int [processing.intBins];

    string [] histofiles;
    string [] relHistos;

    db.get_histograms(out histofiles);
    db.getRelHistos(klas,out relHistos);

    Operator.histogrammer[] histogrammer;

    processing.getClassesFromHistos(relHistos, out histogrammer);

    for (int i =0; i<histogrammer.Length; i++)
        histogrammer[i].setNormHistos();

    Bitmap b;

    if( filechooser.ShowDialog() == DialogResult.OK )
    {
        b = new Bitmap(filechooser.FileName);

        processing.equilizeHist(b, out wH, out wS, out wI);

        processing.segmentAndEq(b, 8, 8, ref pH, ref pS, ref pI);
    }

    testGP mingp = new testGP();

    mingp.Init(hScrollH.Value, hScroll.Value,histofiles,histogrammer,
relHistos, pH, pS, pI, wH, wS, wI, 400,10,processing, /*db*/ klas);

    mingp.Execute();

    string r = "";
    for (int i = 0; i<mingp.HHWeights.Length; i++)
        r+=i.ToString()+" ": "+mingp.HHWeights[i].ToString()+"\n";

    MessageBox.Show(r);

    double[] nWH = new double[wH.Length];
    double[] nWS = new double[wS.Length];
    double[] nWI = new double[wI.Length];

    double[, ,] nPH = new double[8,8,18];
    double[, ,] nPS = new double[8,8,3];
    double[, ,] nPI = new double[8,8,3];

    nWH = processing.normHisto(wH);
    nWS = processing.normHisto(wS);

```

```

nWI = processing.normHisto(wI);

nPH = processing.normSegHisto(8,8,18,pH);
nPS = processing.normSegHisto(8,8,3,pS);
nPI = processing.normSegHisto(8,8,3,pI);

int[] x = new int[] {5,3,5,5 ,3,5,3,4,4,1,7,0};
int[] y = new int[] {6,4,4,1 ,2,1,1,0,2,6,3,0};

int classifyX = 0; //Relevant blokk i spørring
int classifyY = 0;

double P = 0;//Hva den "bolken" er. Brukes for å finne best bolk
(average).

double max = -99999; //total max
double min = 99999;
double tmp = 0;

double h=0;
double s=0;
double intensity=0;

int error=0;

for (int ii = 0; ii<histogrammer.Length; ii++)
{
    h = processing.returnClassNr(x[ii], y[ii], histogrammer[ii].nhistoPH,
histogrammer[ii].nhistoWH, mingp.HHWeights, mingp.PHWeights);
    s = processing.returnClassNr(x[ii], y[ii],
histogrammer[ii].nhistoPS, histogrammer[ii].nhistoWS, mingp.HSWeights, mingp.PSWeights);
    intensity = processing.returnClassNr(x[ii], y[ii],
histogrammer[ii].nhistoPI, histogrammer[ii].nhistoWI, mingp.HIWeights, mingp.PIWeights);

    tmp = Math.Sqrt(h*h + s*s + intensity*intensity);

    max = (tmp > max) ? tmp:max;
    if (tmp< min)
    {
        min = tmp;
        error = ii;
    }
    //min = (tmp < min) ? tmp:min;
}
MessageBox.Show(histofiles[error]);

for (int ii =0; ii<histogrammer.Length; ii++)
{
    h = processing.returnClassNr(x[ii], y[ii],
histogrammer[ii].nhistoPH, mingp.PHWeights);
    s = processing.returnClassNr(x[ii], y[ii],
histogrammer[ii].nhistoPS, mingp.PSWeights);
    intensity = processing.returnClassNr(x[ii], y[ii],
histogrammer[ii].nhistoPI, mingp.PIWeights);
    P += Math.Sqrt(h*h +s*s + intensity*intensity);
}
P = P / histogrammer.Length; //Average relevant block

tmp = 0;
double tmp2=999999999999;

for (int c1 = 0; c1<8; c1++)
    for (int c2 = 0; c2<8; c2++)
    {
        h = processing.returnClassNr(c2, c2, nPH,
mingp.PHWeights);

```

```

mingp.PSWeights);
mingp.PIWeights);

intensity*intensity) - P);

        s = processing.returnClassNr(c2, c2, nPS,
intensity = processing.returnClassNr(c2, c2, nPI,

        tmp = Math.Abs(Math.Sqrt(h*h + s*s +

        classifyX = (tmp < tmp2) ? c1:classifyX;
        classifyY = (tmp < tmp2) ? c2:classifyY;

        tmp2 = (tmp < tmp2) ? tmp:tmp2;

    }

        h = processing.returnClassNr(classifyX, classifyY, nPH, nWH,
mingp.HHWeights, mingp.PHWeights);
        s = processing.returnClassNr(classifyX, classifyY, nPS, nWS,
mingp.HSWeights, mingp.PSWeights);
        intensity = processing.returnClassNr(classifyX, classifyY, nPI, nWI,
mingp.HIWeights, mingp.PIWeights);

        tmp = Math.Sqrt(h*h + s*s + intensity*intensity);

        if ((max>tmp) && (tmp>min))
            MessageBox.Show("Definert som skau. Verdi:"+tmp.ToString());
        else
            MessageBox.Show("Ikke definert som skau.
Verdi:"+tmp.ToString()+". Max er "+max.ToString()+", Min er "+min.ToString());

    }

private void button10_Click(object sender, System.EventArgs e)
{
    int[] r = new int[4];
    double[] w = new double[4];

    int[] res = new int[4];

    r[0] = 10;
    r[1] = 10;
    r[2] = 10;
    r[3] = 10;

    w[0] = 0.1;
    w[1] = 0.6;
    w[2] = 0.2;
    w[3] = 0.1;

    res = processing.addWeights(r,w);

    MessageBox.Show(res[0].ToString()+", "+ res[1].ToString()+",
"+res[2].ToString()+", "+res[3].ToString());

}

}
}

```


Operator class

This is the class where histogram extraction, truncation, equalization, normalization and anything related takes place

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.Windows.Forms;
using System.IO;

namespace Classify_tryout
{
    /// <summary>
    /// Summary description for LogicalOperator.
    /// </summary>
    public class Operator
    {
        public class histogrammer
        {
            public int[, ] histoPH; //Part histogram
            public int[, ] histoPS;
            public int[, ] histoPI;

            public int[] histoWH; //whole histogram
            public int[] histoWS;
            public int[] histoWI;

            public double[, ] nhistoPH; //Part histogram
            public double[, ] nhistoPS;
            public double[, ] nhistoPI;

            public double[] nhistoWH; //whole histogram
            public double[] nhistoWS;
            public double[] nhistoWI;

            public double[] resHH;
            public double[] resHS;
            public double[] resHI;

            public double resH;
            public double resS;
            public double resI;
            public double resHele;

            public double[, ] resPH;
            public double[, ] resPS;
            public double[, ] resPI;

            public double PresH;
            public double PresS;
            public double PresI;

            public double PresHele;

            public histogrammer(int H, int S, int I)
            {
                resHele = 0;
                PresHele = 0;
            }
        }
    }
}
```

```

        histoPH = new int[8,8,H];
        histoPH = new int[8,8,S];
        histoPH = new int[8,8,I];

        histoWH = new int[H];
        histoWS = new int[S];
        histoWI = new int[I];

        resPH = new double[8,8,H];
        resPS = new double[8,8,S];
        resPI = new double[8,8,I];

        resHH = new double[H];
        resHS = new double[S];
        resHI = new double[I];

        resH = 0;
        resS = 0;
        resI = 0;

        PresH = 0;
        PresS = 0;
        PresI = 0;
    }

    public void setNormHistos()
    {
        Operator op = new Operator();
        op.normHisto(this.histoWH, out nhistoWH);
        op.normHisto(this.histoWS, out this.nhistoWS);
        op.normHisto(this.histoWI, out this.nhistoWI);

        op.normSegHisto(8,8,op.hueBins,this.histoPH,out
this.nhistoPH);
        op.normSegHisto(8,8,op.satBins,this.histoPS,out
this.nhistoPS);
        op.normSegHisto(8,8,op.intBins,this.histoPI,out
this.nhistoPI);

    }

}

private Bitmap bmpimg;
public int hueBins = 18;
public int satBins = 3; //sat og int bør være like store
public int intBins = 3;

public int divX=8;
public int divY=8;

public Operator()
{
    //
    // TODO: Add constructor logic here
    //
}

public int[] addWeights(int[] histoInn, double[] weights)
{
    int[] res = new int[histoInn.Length];

    for (int i = 0; i<histoInn.Length; i++)
        res[i] = (int) Math.Round(histoInn[i]*weights[i]);

    return res;
}

```

```

    }

    public int[, ,] addWeights(int[, ,] histoInn, double[] weights)
    {
        int[, ,] res = new int[8,8,histoInn.Length];

        for (int x = 0; x<8; x++)
            for(int y = 0; y<8; y++)
                for (int i = 0; i<histoInn.Length / (8*8); i++)
                    res[x,y,i] = (int)
Math.Round(histoInn[x,y,i]*weights[i]);

        return res;
    }

    //px og py = relevant blokk i hvert segmenterte histogram

    public double returnClassNr(int pX, int pY, double[, ,] NPhistoInn, double[]
NhistoInn, double[] Wweights, double[] Pweights)
    {
        double w=returnClassNr(NhistoInn,Wweights);
        double p=returnClassNr(pX,pY,NPhistoInn,Pweights);

        return (w + p)/2;
        //Math.Sqrt(w + p);
    }

    public double returnClassNr(int pX, int pY, double[, ,] NhistoInn, double[]
Pweights)
    {

        double[] res = new double[NhistoInn.Length];
        double res2 = 0;

        for (int i = 0; i<NhistoInn.Length / (8*8); i++)
        {
            res[i] = NhistoInn[pX,pY,i]*Pweights[i];
            res2 += /*i**/res[i];
        }

        //res2 /= NhistoInn.Length;

        return res2;
    }

    public double returnClassNr(double[] NhistoInn, double[] weights)
    {
        double[] res = new double[NhistoInn.Length];
        double res2 = 0;

        for (int i = 0; i<NhistoInn.Length; i++){
            res[i] = NhistoInn[i]*weights[i];
            res2 += /*i**/res[i];
        }

        //res2 /= NhistoInn.Length;

        return res2;
    }

    public double[] normHisto(int[] histoInn)
    {
        double[] res;

```

```

        normHisto(histoInn, out res);
        return res;
    }

    public void normHisto(int[] histoInn, out double[] normHisto)
    {
        normHisto = new double[histoInn.Length];
        int samples = 0;

        for (int i = 0; i < histoInn.Length; i++)
            samples += histoInn[i];

        //    MessageBox.Show(samples.ToString());

        for (int i = 0; i < histoInn.Length; i++)
            normHisto[i] = (double) histoInn[i] / (double) samples;
    }

    public double[, ] normSegHisto(int divX, int divY, int lengthHisto, int[, ]
histoInn)
    {
        double[, ] res;
        normSegHisto(divX, divY, lengthHisto, histoInn, out res);
        return res;
    }

    public void normSegHisto(int divX, int divY, int lengthHisto, int[, ]
histoInn, out double[, ] normHisto)
    {
        normHisto = new double[divX, divY, lengthHisto];

        int samples = 0;

        for (int i = 0; i < lengthHisto; i++)
            samples += histoInn[0,0,i];

        //    MessageBox.Show(samples.ToString());

        for (int x = 0; x < divX; x++)
            for (int y=0; y< divY; y++)
                for (int i = 0; i < lengthHisto; i++)
                    normHisto[x,y,i] = (double) histoInn[x,y,i] /
(double) samples;
    }

    public void getClassesFromHistos(string[] histos, out histogrammer[] res)
    {
        res = new histogrammer[histos.Length];

        for (int i = 0; i<histos.Length; i++)
        {
            res[i] = new histogrammer(hueBins, satBins, intBins);
            //    MessageBox.Show(histos[i].Replace("BILDER", "HISTOGRAM"));
            fileToHisto(histos[i].Replace("BILDER", "HISTOGRAM"), out
res[i].histoPH, out res[i].histoPS, out res[i].histoPI, out res[i].histoWH, out
res[i].histoWS, out res[i].histoWI);
        }
    }

    public void setImage( Bitmap bmp )
    {
        bmpimg = ( Bitmap )bmp.Clone( );
    }

    public Bitmap getImage( )

```

```

    {
        return ( Bitmap ) bmpimg.Clone( );
    }

    public void fileToHisto(string fileName, out int [,] histoH, out int [,]
histoS, out int [,] histoI,
    out int [] histoWH, out int [] histoWS, out int [] histoWI)
    {

        histoH = new int[8,8,hueBins];
        histoS = new int[8,8,satBins];
        histoI = new int[8,8,intBins];

        histoWH = new int[hueBins];
        histoWS = new int[satBins];
        histoWI = new int[intBins];

        System.IO.FileStream fs = new
System.IO.FileStream(fileName.Replace("\\BILDER\\", "\\HISTOGRAM\\"), FileMode.Open,
FileAccess.Read);
        BinaryReader r = new BinaryReader(fs);

        for (int x=0; x<8; x++)
            for (int y=0; y<8; y++)
            {
                for (int i = 0; i < hueBins; i++)
                    histoH[x,y,i]=r.ReadInt32();
                for (int i = 0; i < satBins; i++)
                {
                    histoS[x,y,i]=r.ReadInt32();
                    histoI[x,y,i]=r.ReadInt32();
                }
            }

        for (int i = 0; i < hueBins; i++)
            histoWH[i]=r.ReadInt32();
        for (int i = 0; i < satBins; i++)
        {
            histoWS[i]=r.ReadInt32();
            histoWI[i]=r.ReadInt32();
        }

        r.Close();
        fs.Close();
    }

    public void histoToFile(string fileName, int [,] histoH, int [,] histoS,
int [,] histoI,
    int [] histoWH, int [] histoWS, int [] histoWI)
    {
        //
        System.Windows.Forms.MessageBox.Show(fileName.Replace("\\bilder\\", "\\histogram\\"
));
        //
        fileName.Replace
        System.IO.FileStream fs = new
System.IO.FileStream(fileName.Replace("\\BILDER\\", "\\HISTOGRAM\\"),
FileMode.OpenOrCreate, FileAccess.Write);
        BinaryWriter w = new BinaryWriter(fs);

        // Add some text to the file.
        for (int x=0; x<8; x++)
            for (int y=0; y<8; y++)
            {
                for (int i = 0; i < hueBins; i++)
                    w.Write(histoH[x,y,i]);
                for (int i = 0; i < satBins; i++)
                {

```

```

        w.Write(histoS[x,y,i]);
        w.Write(histoI[x,y,i]);
    }

    }
for (int i = 0; i < hueBins; i++)
    w.Write(histoWH[i]);
for (int i = 0; i < satBins; i++)
{
    w.Write(histoWS[i]);
    w.Write(histoWI[i]);
}

w.Close();
fs.Close();

}

public void subjectiveHueTrunc(int [] histoH, out int[] histoHut)
{

    histoHut = new int [hueBins];
    int j = -10;
    int teller1 = 0; int teller2 = 0;
    int steps = histoH.Length / hueBins;

    for (int i=0; i<histoH.Length; i++)
    {

        if (j<0) j=360 + j;

        if (j>359) j=j-360;

        if (teller1 == steps)
        {
            teller1 = 0;
            teller2++;
        }

        histoHut[teller2] += histoH[j];

        teller1++;
        j++;

    }

}

public void truncHisto (int [] histoInn, int truncSize, out int[] histoUt)
{
    histoUt = new int [truncSize];
    int teller1 = 0; int teller2 = 0;
    int steps = (histoInn.Length / truncSize)+1;

    for (int i=0; i<histoInn.Length; i++)
    {

        if (teller1 == steps)
        {
            teller1 = 0;
            teller2++;
        }
        //    MessageBox.Show(teller2.ToString()+" og i =
"+i.ToString());
        histoUt[teller2] += histoInn[i];

        teller1++;
    }
}

```

```

    }
}

public void calcHisto( PictureBox bildeR, PictureBox bildeG, PictureBox
bildeB )
{
    ColorSpace color_conv;
    color_conv = new ColorSpace( );

    int cH=0;
    int cS=0;
    int cI=0;

    /*int r = 0;
    int g = 0;
    int b = 0;*/

    BitmapData data = bmpimg.LockBits( new System.Drawing.Rectangle( 0 ,
0 , bmpimg.Width , bmpimg.Height ), ImageLockMode.ReadWrite , PixelFormat.Format24bppRgb
);
    unsafe
    {
        byte* ptr = ( byte* )data.Scan0;

        int remain = data.Stride - data.Width * 3;

        int[ ] histogramR = new int[ 256 ];
        int[ ] histogramG = new int[ 256 ];
        int[ ] histogramB = new int[ 256 ];

        int[ ] histogramH = new int[ 360 ];
        int[ ] histogramS = new int[ 256 ];
        int[ ] histogramI = new int[ 256 ];

        for( int i = 0 ; i < histogramR.Length ; i ++ )
        {
            histogramR[ i ] = 0;
            histogramG[ i ] = 0;
            histogramB[ i ] = 0;

            histogramS[ i ] = 0;
            histogramI[ i ] = 0;
        }

        for( int i = 0 ; i < histogramH.Length ; i ++ )
        {
            histogramH[ i ] = 0;
        }

        for( int i = 0 ; i < data.Height ; i ++ )
        {
            for( int j = 0 ; j < data.Width ; j ++ )
            {
                ColorSpace.RGBToHSI(ptr[2],ptr[1],ptr[0],out cH,
out cS,out cI);

                histogramH[ cH ] ++;
                histogramS[ cS ] ++;
                histogramI[ cI ] ++;

                histogramR[ ptr[2] ] ++;
                histogramG[ ptr[1] ] ++;
                histogramB[ ptr[0] ] ++;
                ptr += 3;
            }
        }
    }
}

```

```

        ptr += remain;
    }
    drawHistogram( histogramH , bildeR );
    drawHistogram( histogramS , bildeG );
    drawHistogram( histogramI , bildeB );
}

bmpimg.UnlockBits( data );
}

//public float distancelv_euclidian
//utdatert
public float distancelv( int [] histo1, int [] histo2, int algorithm)
//Intersection distance for en dim.
{
    int magHisto1 = 0;
    int magHisto2 = 0;
    int mag = 0;
    float tmpVal = 0;

    for (int i = 0; i < histo1.Length; i++)
    {

        magHisto1 += histo1[i];
        magHisto2 += histo2[i];

    }

    if (algorithm == 0)
    {
        mag = ( magHisto1 < magHisto2 ) ? magHisto1 : magHisto2;

        for (int i = 0; i < histo1.Length; i++)
        {

            tmpVal += (float)(Math.Abs(
((float)histo1[i]/(float)magHisto1) - ((float)histo2[i])/ (float) magHisto2));
        }
    }
    else if (algorithm == 1) //intersection
    {
        mag = ( magHisto1 < magHisto2 ) ? magHisto1 : magHisto2;
    }

    else if (algorithm == 2) //quadratic cross distance
    {
        int[,] A = new int[histo1.Length, histo2.Length];

        {

        }

    }

    }
    return tmpVal;
}

public void find_nearest(int [,,] q_hPH,int [,,] q_hPS,int [,,] q_hPI,
    int [] q_hWH, int [] q_hWS,int [] q_hWI, string[] histo_files,

```



```

        int algP, int algH, out string[] res_name, out float[] res_dist)
    {
        find_nearest(q_hPH, q_hPS, q_hPI, q_hWH, q_hWS, q_hWI, histo_files,
            (float)1/ (float)3, (float) 1/ (float)3, (float) 1/ (float) 3,
            (float)1/ (float)3, (float) 1/ (float)3, (float) 1/ (float) 3,
            algP, algH, out res_name, out res_dist);
    }

    public void find_nearest(int [,] q_hPH, int [,] q_hPS, int [,] q_hPI,
        int [] q_hWH, int [] q_hWS, int [] q_hWI, string[] histo_files,
        float wHH, float wHS, float wHI, float wPH, float wPS, float wPI,
        int algP, int algH, out string[] res_name, out float[] res_dist)
    {

        int [,] histoH = new int [8, 8, hueBins]; //Delhistogram
        int [,] histoS = new int [divX, divY, satBins];
        int [,] histoI = new int [divX, divY, intBins];

        int [] histoWH = new int [hueBins]; //Hist. over hele bildet
        int [] histoWS = new int [satBins];
        int [] histoWI = new int [intBins];

        res_name = new string[histo_files.Length];
        res_dist = new float[histo_files.Length];

        float tmpDistW = 0;
        float tmpDistP = 0;

        for (int i = 0; i<histo_files.Length; i++)
        {
            fileToHisto(histo_files[i], out histoH, out histoS, out histoI,
            out histoWH, out histoWS, out histoWI);

            if (algH>-1)
                tmpDistW = distance(q_hWH, q_hWS, q_hWI, histoWH,
            histoWS, histoWI, algH);

            if (algP>-1)
                tmpDistP = bestDistance(q_hPH, q_hPS, q_hPI,
            histoH, histoS, histoI, algP);

            res_name[i] = histo_files[i];

            if ((algP == -1) && (algH != -1))
                res_dist[i] = tmpDistW;
            else if ((algP != -1) && (algH == -1))
                res_dist[i] = tmpDistP;
            else if ((algP != -1) && (algH != -1))
                res_dist[i] = (tmpDistW + tmpDistP) /2;
            else
                res_dist[i] = 9999999;

        }
        Array.Sort( res_dist, res_name);

        //    MessageBox.Show("Bilde "+res[i]+ "ferdig. Avstand =
        "+comb[i].ToString());
    }

    public float distance( int [] q_histoH, int [] q_histoS, int [] q_histoI, int

```

```

[] t_histoH,int [] t_histoS,int [] t_histoI, int algorithm)
{
    float tmp=(float) distance(q_histoH, q_histoS, q_histoI, t_histoH,
t_histoS, t_histoI,(float)1/ (float)3 ,(float) 1/ (float)3, (float) 1/ (float) 3,
algorithm);
    //MessageBox.Show(tmp.ToString());
    return tmp;
}
public float distance( int [] q_histoH,int [] q_histoS,int [] q_histoI, int
[] t_histoH,int [] t_histoS,int [] t_histoI,
float wH, float wS, float wI,
int algorithm)

{

float dist = 2;

if (algorithm == 0) //euclidian
{
float dH = distance1v(q_histoH, t_histoH, algorithm)*(wH*3);
//med vektning
float dS = distance1v(q_histoS, t_histoS, algorithm)*(wS*3);
float dI = distance1v(q_histoI, t_histoI, algorithm)*(wI*3);

dist = (float) Math.Sqrt(dH*dH + dS*dS + dI*dI);

}
else if (algorithm == 1)//intersection
{
int[] magHisto1 = new int[3];
int[] magHisto2 = new int[3];
int mag = 0;

float tmpVal = 0;

for (int i = 0; i < q_histoH.Length; i++)
{
magHisto1[0] += q_histoH[i];
magHisto2[0] += t_histoH[i];
}
for (int i = 0; i < q_histoS.Length; i++)
{
magHisto1[1] += q_histoS[i];
magHisto2[1] += t_histoS[i];
}

for (int i = 0; i < q_histoI.Length; i++)
{
magHisto1[2] += q_histoI[i];
magHisto2[2] += t_histoI[i];
}

mag = ( magHisto1[0]+magHisto1[1]+magHisto1[2] < magHisto2[0]+
magHisto2[1]+ magHisto2[2] ) ?
magHisto1[0]+magHisto1[1]+magHisto1[2] : magHisto2[0]+
magHisto2[1]+ magHisto2[2];

for (int i = 0; i < q_histoH.Length; i++)
{
tmpVal += ( q_histoH[i] < t_histoH[i] ) ?
q_histoH[i]*(wH*3) : t_histoH[i]*(wH*3);
}
}
}

```

```

        for (int i = 0; i < q_histoS.Length; i++)
        {
            tmpVal += ( q_histoS[i] < t_histoS[i]) ?
q_histoS[i]*(wS*3) : t_histoS[i]*(wS*3);
        }

        for (int i = 0; i < q_histoI.Length; i++)
        {
            tmpVal += ( q_histoI[i] < t_histoI[i]) ?
q_histoI[i]*(wI*3) : t_histoI[i]*(wI*3);
        }

        dist = 1 - (tmpVal / mag);
    }
    else if (algorithm == 2) //Cross distance
    {

        Matrix A = new Matrix(q_histoH.Length, t_histoH.Length);

        Matrix[] p = new Matrix[3];
        Matrix[] q = new Matrix[3];

        p[0] = new Matrix(t_histoH.Length,1); //alle ha samme dim.
norm S og I senere
        p[1] = new Matrix(t_histoH.Length,1);
        p[2] = new Matrix(t_histoH.Length,1);

        q[0] = new Matrix(q_histoH.Length,1);
        q[1] = new Matrix(q_histoH.Length,1);
        q[2] = new Matrix(q_histoH.Length,1);

        int stepsS = q_histoH.Length / q_histoS.Length ;//+1;
        int stepsI = q_histoH.Length / q_histoI.Length ;//+1;
        int tellerS = 0; int tS = 0;
        int tellerI = 0; int tI = 0;

        int hi = 0; int hj = 0;
        int si = 0; int sj = 0;
        int vi = 0; int vj = 0;

        //MessageBox.Show(p[0].data[0].Length.ToString());
        for (int i = 0; i<q_histoH.Length; i++)
        {
            //MessageBox.Show(i.ToString()+ " max = " +
p[0].Array[0].Length.ToString());
            p[0].data[i][0] = t_histoH[i];
            q[0].data[i][0] = q_histoH[i];
        }

        for (int i = 0; i<q_histoS.Length; i++)
        {
            for (int j = i*stepsS; j<(i*stepsS) + stepsS; j++)
            {
                p[1].data[j][0] = t_histoS[i];
                q[1].data[j][0] = q_histoS[i];
            }
        }

        for (int i = 0; i<q_histoI.Length; i++)
        {
            for (int j = i*stepsI; j<(i*stepsI) + stepsI; j++)
            {

```

```

        p[2].data[j][0] = t_histoI[i];
        q[2].data[j][0] = q_histoI[i];
    }
}

for (int i = 0; i < q_histoH.Length; i++) //Generere A
{
    tS=0;
    tellerS = 0;
    tI=0;
    tellerI = 0;

    for (int j=0; j < t_histoH.Length; j++)
    {
        if (tellerS == stepsS)
        {
            tS++;
            tellerS = 0;
        }

        if (tellerI == stepsI)
        {
            tI++;
            tellerI = 0;
        }
        hi = q_histoH[i];
        hj = t_histoH[i];
        si = q_histoS[tS];
        sj = t_histoS[tS];
        vi = q_histoI[tI];
        vj = t_histoI[tI];

        A.data[i][j] = (float) 1 - (float) 1/
            Math.Pow((vi - vj),2) + Math.Pow(si *
            Math.Pow(si * Math.Cos(hi) - sj*
            Math.Cos(hj),2))
            );

        tellerS++;
        tellerI++;
    }
}

Matrix tmp = new Matrix(1,1);

//tmp = p[0] - q[0];
tmp.data[0][0] = 0;
for (int i = 0; i < 3; i++)
    for (int j=0; j < 3; j++)
    {
        Matrix tmpj = new Matrix(p[j].Rows,
        tmpj = p[j] - q[j];

        Matrix tmpi = new Matrix(p[i].Rows,
        tmpi = p[i] - q[i];

        Matrix pi_qi = tmpi.Transpose();
    }
}

```

```

        tmp += pi_qi * (A * tmpj);

    }

    dist = Math.Abs((float) tmp.data[0][0]);

    }

    return dist;

}

    public float bestDistance(int [,] q_histoH,int [,] q_histoS,int [,]
q_histoI, int [,] t_histoH,int [,] t_histoS,int [,] t_histoI, int algorithm)
    {
        return (float) bestDistance(q_histoH, q_histoS, q_histoI, t_histoH,
t_histoS,t_histoI,(float)1/ (float)3 ,(float) 1/ (float)3, (float) 1/ (float) 3,
algorithm);
    }

    public float bestDistance(int [,] q_histoH,int [,] q_histoS,int [,]
q_histoI, int [,] t_histoH,int [,] t_histoS,int [,] t_histoI, int algorithm, out int
x, out int y)
    {
        return (float) bestDistance(q_histoH, q_histoS, q_histoI, t_histoH,
t_histoS,t_histoI,(float)1/ (float)3 ,(float) 1/ (float)3, (float) 1/ (float) 3,
algorithm, out x, out y);
    }

    public float bestDistance(int [,] q_histoH,int [,] q_histoS,int [,]
q_histoI, int [,] t_histoH,int [,] t_histoS,int [,] t_histoI, float wHH, float wHS,
float wHI, int algorithm)
    {
        int dummy1;
        int dummy2;
        return bestDistance(q_histoH, q_histoS, q_histoI, t_histoH,
t_histoS, t_histoI, wHH, wHS, wHI, algorithm, out dummy1, out dummy2);
    }

    public float bestDistance(int [,] q_histoH,int [,] q_histoS,int [,]
q_histoI, int [,] t_histoH,int [,] t_histoS,int [,] t_histoI, float wHH, float wHS,
float wHI, int algorithm, out int x, out int y)
    {
        y = 0;
        x = 0;

        int [] tempqH = new int [hueBins];
        int [] tempqS = new int [satBins];
        int [] tempqI = new int [intBins];

        int [] temptH = new int [hueBins];
        int [] temptS = new int [satBins];
        int [] temptI = new int [intBins];

        float tmp_distance=999999999;
        float tmp_ny = 0;
        float res = 0;

        for (int qX=0; qX<8; qX++)
            for(int qY=0; qY<8; qY++)
            {
                for (int counter1=0; counter1<8; counter1++)

                    for (int counter2=0; counter2<8; counter2++)
                    {
                        for (int i = 0; i < hueBins; i++)
                        {
                            tempqH[i]=q_histoH[qX,qY,i];

temptH[i]=t_histoH[counter1,counter2,i];
                        }
                    }
            }
    }

```

```

        for (int i = 0; i < satBins; i++)
        {
            tempqS[i]=q_histoS[qX,qY,i];

temptS[i]=t_histoS[counter1,counter2,i];

            tempqI[i]=q_histoI[qX,qY,i];

temptI[i]=t_histoI[counter1,counter2,i];
        }

    tmp_ny = distance(tempqH,tempqS,tempqI, temptH,temptS,temptI,WHH,WHS,WHI,
algorithm);

    x = ( tmp_ny < tmp_distance ) ? qX : x;
    y = ( tmp_ny < tmp_distance ) ? qY : y;

    tmp_distance = ( tmp_ny < tmp_distance ) ?

tmp_ny : tmp_distance;

    //MessageBox.Show("I
bestdistance"+tmp_ny.ToString());
    }

    res += tmp_distance;
    //MessageBox.Show("i bestDistance: "+res.ToString());
    }

    //MessageBox.Show("i bestDistance: "+res.ToString());
    return (res / (8*8));
}

//Må defineres fra Caller

public void segmentAndEq(Bitmap b, int divX, int divY, ref int[,],
histoH,ref int[,], histoS,ref int[,], histoI)
{
    equilibizeHist(b,divX, divY, ref histoH, ref histoS, ref histoI);
}

public void equilibizeHist( Bitmap bitm,int divX, int divY, ref int[,],
histoH,ref int[,], histoS,ref int[,], histoI)
{

    int cH=0;
    int cS=0;
    int cI=0;

    int r = 0;
    int g = 0;
    int b = 0;

    int stepsX = bitm.Width / divX;
    int stepsY = bitm.Height / divY;

    for (int counterX = 0; counterX<divX; counterX++)
        for (int counterY = 0; counterY<divY; counterY++)
            //todo

            //MessageBox.Show("X= "+counterX.ToString()+" Y=
"+counterY.ToString()+ " stepsX = "+stepsX.ToString()+" stepsY = "+stepsY.ToString());
            BitmapData data = bitm.LockBits( new
System.Drawing.Rectangle( stepsX * counterX , stepsY* counterY ,stepsX , stepsY),
ImageLockMode.ReadWrite , PixelFormat.Format24bppRgb );

```

```

unsafe
{
    byte* ptr = ( byte* )data.Scan0;

    int remain = data.Stride - data.Width * 3;

    int[ ] histogramH = new int[ 360 ];
    int[ ] histogramS = new int[ 256 ];
    int[ ] histogramI = new int[ 256 ];

    for( int i = 0 ; i < histogramS.Length ; i ++ )

        histogramS[ i ] = 0;
        histogramI[ i ] = 0;

    }

    for( int i = 0 ; i < histogramH.Length ; i ++ )
    {
        histogramH[ i ] = 0;
    }

    for( int i = 0 ; i < data.Height ; i ++ )
    {
        for( int j = 0 ; j < data.Width ; j ++ )
        {

ColorSpace.RGBToHSI(ptr[2],ptr[1],ptr[0],out cH, out cS,out cI);

            histogramH[ cH ] ++;
            histogramS[ cS ] ++;
            histogramI[ cI ] ++;
            ptr += 3;

        }

        ptr += remain;
    }

    float[ ] LUTS = equilize( histogramS , data.Width
* data.Height );
    float[ ] LUTI = equilize( histogramI , data.Width
* data.Height );
    ptr = ( byte* )data.Scan0;

    for( int i = 0 ; i < data.Height ; i ++ )
    {
        for( int j = 0 ; j < data.Width ; j ++ )
        {

ColorSpace.RGBToHSI(ptr[2],ptr[1],ptr[0],out cH,out cS,out cI);

            int indexH = cH;
            int indexS = cS;
            int indexI = cI;

            int nValueH = cH;

            int nValueS = (int) LUTS[ indexS ];
            int nValueI = (int) LUTI[ indexI ];

//Kun normaliser intensitet

```

```

        if( LUTS[ indexS ] > 255 )
            nValueS = 255;

        if( LUTI[ indexI ] > 255 )
            nValueI = 255;

    ColorSpace.HSIToRGB(nValueH,nValueS,nValueI, out r, out g, out b);

        ptr[2] = (byte) r;
        ptr[1] = (byte) g;
        ptr[0] = (byte) b;

        ptr += 3;
    }
    //Ferdig^
    ptr += remain;
}

ptr = ( byte* )data.Scan0;

histogramH = new int[ 360 ];
histogramS = new int[ 256 ];
histogramI = new int[ 256 ];

for( int i = 0 ; i < histogramS.Length ; i ++ )
{

    histogramS[ i ] = 0;
    histogramI[ i ] = 0;

}

for( int i = 0 ; i < histogramH.Length ; i ++ )
{
    histogramH[ i ] = 0;
}

for( int i = 0 ; i < data.Height ; i ++ )
{
    for( int j = 0 ; j < data.Width ; j ++ )
    {

        //int mean = ptr[ 0 ];

    ColorSpace.RGBToHSI(ptr[2],ptr[1],ptr[0],out cH, out cS,out cI);

        histogramH[ cH ] ++;
        histogramS[ cS ] ++;
        histogramI[ cI ] ++;

        ptr += 3;

    }

    ptr += remain;
}
int test=data.Height*data.Width;

//MessageBox.Show(histogramI[255].ToString()+ "

test = "+ test.ToString());

int [] tHistoH = new int [hueBins]; //Truncated

H-histogram

```



```

S-histogram
I-histogram

int [] tHistoS = new int [satBins]; //Truncated
int [] tHistoI = new int [intBins]; //Truncated

//truncHisto(histogramH, 32, out tHistoH);
subjectiveHueTrunc(histogramH, out tHistoH);
truncHisto(histogramS, satBins, out tHistoS);
truncHisto(histogramI, intBins, out tHistoI);
//MessageBox.Show("Histo i eqhist:
"+tHistoH[0].ToString()+
counter3++){
tHistoH[counter3];}

counter3++)
{
tHistoS[counter3];
tHistoI[counter3];

histoS[counterX,counterY, counter3] =
histoI[counterX,counterY, counter3] =

}

bitm.UnlockBits( data );

}

}

public void equilizeHist(Bitmap b, out int[] histoH,out int[] histoS,out
int[] histoI)
{
setImage(b);
PictureBox dummy1 = new PictureBox();
PictureBox dummy2 = new PictureBox();
PictureBox dummy3= new PictureBox();

equilizeHist(dummy1, dummy2, dummy3, out histoH, out histoS, out
histoI, false);
}

public void equilizeHist( PictureBox bildeR, PictureBox bildeG, PictureBox
bildeB, out int[] histoH,out int[] histoS,out int[] histoI, bool draw)
{
int cH=0;
int cS=0;
int cI=0;

int r = 0;
int g = 0;
int b = 0;

BitmapData data = bmpimg.LockBits( new System.Drawing.Rectangle( 0 ,
0 , bmpimg.Width , bmpimg.Height ), ImageLockMode.ReadWrite , PixelFormat.Format24bppRgb
);
unsafe
{
byte* ptr = ( byte* )data.Scan0;

```

```

int remain = data.Stride - data.Width * 3;

int[ ] histogramH = new int[ 360 ];
int[ ] histogramS = new int[ 256 ];
int[ ] histogramI = new int[ 256 ];

for( int i = 0 ; i < histogramS.Length ; i ++ )
{
    histogramS[ i ] = 0;
    histogramI[ i ] = 0;
}

for( int i = 0 ; i < histogramH.Length ; i ++ )
{
    histogramH[ i ] = 0;
}

for( int i = 0 ; i < data.Height ; i ++ )
{
    for( int j = 0 ; j < data.Width ; j ++ )
    {
        ColorSpace.RGBToHSI(ptr[2],ptr[1],ptr[0],out cH,
out cS,out cI);

        histogramH[ cH ] ++;
        histogramS[ cS ] ++;
        histogramI[ cI ] ++;
        ptr += 3;
    }
    ptr += remain;
}

float[ ] LUTS = equilize( histogramS , data.Width *
data.Height );
float[ ] LUTI = equilize( histogramI , data.Width *
data.Height );
ptr = ( byte* )data.Scan0;

for( int i = 0 ; i < data.Height ; i ++ )
{
    for( int j = 0 ; j < data.Width ; j ++ )
    {
        ColorSpace.RGBToHSI(ptr[2],ptr[1],ptr[0],out
cH,out cS,out cI);

        int indexH = cH;
        int indexS = cS;
        int indexI = cI;

        int nValueH = cH;
        int nValueS = cS;
        int nValueI = (int) LUTI[ indexI ]; //Kun
normaliser intensitet

        if( LUTI[ indexI ] > 255 )
            nValueI = 255;
    }
}

```

```

r, out g, out b);

        ColorSpace.HSIToRGB(nValueH,nValueS,nValueI, out

        ptr[2] = (byte) r;
        ptr[1] = (byte) g;
        ptr[0] = (byte) b;

        ptr += 3;
    }
    ptr += remain;
}

ptr = ( byte* )data.Scan0;

*/

    histogramH = new int[ 360 ];
    histogramS = new int[ 256 ];
    histogramI = new int[ 256 ];

    for( int i = 0 ; i < histogramS.Length ; i ++ )
    {

        histogramS[ i ] = 0;
        histogramI[ i ] = 0;

    }
    for( int i = 0 ; i < histogramH.Length ; i ++ )
    {
        histogramH[ i ] = 0;
    }

    for( int i = 0 ; i < data.Height ; i ++ )
    {
        for( int j = 0 ; j < data.Width ; j ++ )
        {

            ColorSpace.RGBToHSI(ptr[2],ptr[1],ptr[0],out cH,
out cS,out cI);

            histogramH[ cH ] ++;
            histogramS[ cS ] ++;
            histogramI[ cI ] ++;

            /*
            histogramR[ ptr[2] ] ++;
            histogramG[ ptr[1] ] ++;
            histogramB[ ptr[0] ] ++;*/

            ptr += 3;
        }

        ptr += remain;
    }
    int test=data.Height*data.Width;

    int [] tHistoH = new int [hueBins]; //Truncated H-histogram
    int [] tHistoS = new int [satBins]; //Truncated S-histogram
    int [] tHistoI = new int [intBins]; //Truncated I-histogram

    //truncHisto(histogramH, 32, out tHistoH);
    subjectiveHueTrunc(histogramH, out tHistoH);
    truncHisto(histogramS, satBins, out tHistoS);
    truncHisto(histogramI, intBins, out tHistoI);

```

```

        histoH = tHistoH;
        histoS = tHistoS;
        histoI = tHistoI;

        if (draw)
        {
            drawHistogram( histogramH , bildeR );
            drawHistogram( histogramS , bildeG );
            drawHistogram( histogramI , bildeB );
        }
    }

    bmpimg.UnlockBits( data );
}

public float[ ] equilize( int[ ] histogram , long numPixel )
{
    float[ ] hist = new float[ 256 ];

    hist[ 0 ] = histogram[ 0 ] * histogram.Length / numPixel;
    long prev = histogram[ 0 ];
    string str = "";
    str += ( int )hist[ 0 ] + "\n";

    for( int i = 1 ; i < hist.Length ; i ++ )
    {
        prev += histogram[ i ];
        hist[ i ] = prev * histogram.Length / numPixel;
        str += ( int )hist[ i ] + "  _" + i + "\t";
    }

    //   MessageBox.Show( str );
    return hist;
}

public void drawHistogram( int[ ] histogram , PictureBox bilde )
{
    Bitmap bmp = new Bitmap( histogram.Length + 10 , 310 );
    bilde.Image = bmp;
    int keep = 0;

    BitmapData data = bmp.LockBits( new System.Drawing.Rectangle( 0 , 0 ,
bmp.Width , bmp.Height ) , ImageLockMode.ReadWrite , PixelFormat.Format24bppRgb );

    unsafe
    {
        int remain = data.Stride - data.Width * 3;
        byte* ptr = ( byte* )data.Scan0;

        for( int i = 0 ; i < data.Height ; i ++ )
        {
            for( int j = 0 ; j < data.Width ; j ++ )
            {
                ptr[ 0 ] = ptr[ 1 ] = ptr[ 2 ] = 150;
                ptr += 3;
            }
            ptr += remain;
        }

        int max = 0;
        for( int i = 0 ; i < histogram.Length ; i ++ )
        {
            if( max < histogram[ i ] )

```

```

        max = histogram[ i ];
    }
    for( int i = 0 ; i < histogram.Length ; i ++ )
    {
        ptr = ( byte* )data.Scan0;
        ptr += data.Stride * ( 305 ) + ( i + 5 ) * 3;

        int length = ( int )( 1.0 * histogram[ i ] * 300 / max

);

        for( int j = 0 ; j < length ; j ++ )
        {
            ptr[ 0 ] = 255;
            ptr[ 1 ] = ptr[ 2 ] = 0;
            ptr -= data.Stride;
        }
    }

    bmp.UnlockBits( data );
}
}
}

```

Database class

This is the SQL-part of the program. Some other methods were thrown in here as well, which is bad programming practice, but I don't dare to change anything at this point

```
using System.Windows.Forms;
using System;
using System.Data;
using System.Data.OracleClient;
using System.IO;

namespace Classify_tryout
{
    /// <summary>
    /// Summary description for db_management.
    /// </summary>
    public class DbManagement
    {
        private System.Data.OracleClient.OracleConnection connection;
        public string bilde_kat = "C:\\\\BILDER\\";
        public string histo_kat = "C:\\\\HISTOGRAM\\";

        public DbManagement()
        {
            connection = new System.Data.OracleClient.OracleConnection("user
id=Stian;data source=;password=S9m63gar");
        }

        ~DbManagement()
        {
        }

        public void open()
        {
            this.connection.Open();
        }

        public void close()
        {
            this.connection.Close();
        }

        public int Query_int(string mySelectQuery, int return_nr) //returns 1st
element
        {
            int [] tmp = new int[1];

            OracleCommand myCommand = new OracleCommand(mySelectQuery,
connection);
            //connection.Open();
            OracleDataReader myReader =
myCommand.ExecuteReader(CommandBehavior.CloseConnection);

            myReader.Read();

            return myReader.GetInt32(return_nr);
        }

        public string Query_str(string mySelectQuery, int return_nr)
        {
        }
    }
}
```

```

        int [] tmp = new int[1];

        OracleCommand myCommand = new OracleCommand(mySelectQuery,
connection);
        //connection.Open();
        OracleDataReader myReader =
myCommand.ExecuteReader(CommandBehavior.CloseConnection);

        myReader.Read();
        return myReader.GetString(return_nr);
    }

    public void get_pictures(string[] histograms, out string[] res)
    {
        int antall_pictures = 0;
        this.open();
        OracleCommand myCommand = new OracleCommand("SELECT count(FILENAME)
from indices", connection);
        OracleDataReader antall = myCommand.ExecuteReader();
        antall.Read();
        antall_pictures = antall.GetInt32(0);

        res = new string [antall_pictures];

        myCommand.CommandText="select FILENAME from INDICES";

        OracleDataReader resReader = myCommand.ExecuteReader();

        for (int i = 0; i < antall_pictures; i++)
        {
            resReader.Read();
            res[i]=resReader.GetString(0);
        }

        resReader.Close();
        antall.Close();

        this.connection.Close();
    }

    public void get_histograms(out string[] res)
    {
        int antall_histogram = 0;
        this.open();
        OracleCommand myCommand = new OracleCommand("SELECT
count(histogram_file) from indices", connection);
        //connection.Open();
        OracleDataReader antall = myCommand.ExecuteReader();
        antall.Read();
        antall_histogram = antall.GetInt32(0);

        res = new string [antall_histogram];

        myCommand.CommandText="select HISTOGRAM_FILE from INDICES";

        //myCommand = new OracleCommand("select HISTOGRAM_FILE from INDICES",
connection);
        OracleDataReader resReader = myCommand.ExecuteReader();

        for (int i = 0; i < antall_histogram; i++)
        {
            resReader.Read();
            res[i]=resReader.GetString(0);
        }

        resReader.Close();
        antall.Close();
    }

```

```

        this.connection.Close();
    }

    public OracleDataReader select_Query(string mySelectQuery)
    {
        int [] tmp = new int[1];
        //this.open();
        OracleCommand myCommand = new OracleCommand(mySelectQuery,
connection);
        //connection.Open();
        MessageBox.Show(this.connection.ConnectionString);

        return myCommand.ExecuteReader();
        //this.close();
    }

    public void getRelHistos(string DBName, out string[] relHistos)
    {
        int antall_histogram = 0;
        this.open();
        OracleCommand myCommand = new OracleCommand("SELECT
count(INDICE_HIST) from "+DBName, connection);
        //connection.Open();
        OracleDataReader antall = myCommand.ExecuteReader();
        antall.Read();
        antall_histogram = antall.GetInt32(0);

        relHistos = new string [antall_histogram];

        myCommand.CommandText="select INDICE_HIST from "+DBName;
        //myCommand = new OracleCommand("select HISTOGRAM_FILE from INDICES",
connection);
        OracleDataReader resReader = myCommand.ExecuteReader();

        for (int i = 0; i < antall_histogram; i++)
        {
            resReader.Read();
            relHistos[i]=resReader.GetString(0);
        }

        resReader.Close();
        antall.Close();

        this.connection.Close();
    }

    public int ant_relevant(string[] histos, string databaseNavn, int count)
    {
        this.open();
        string q = "SELECT COUNT(INDICE_HIST) FROM "+databaseNavn+" ";
        int res = 0;
        int t = 0;

        if (histos.Length>0)
            q += "WHERE ";

        if (count>histos.Length)
            t = histos.Length;
        else
            t = count;
    }

```



```

        for (int i=0; i<t; i++)
        {
            q+=" INDICE_HIST = "+histos[i];
            if (i != histos.Length -1)
                q += " OR ";
        }

        OracleCommand myCommand = new OracleCommand(q);

        OracleDataReader resReader = myCommand.ExecuteReader();

        resReader.Read();
        res=resReader.GetInt32(0);

        resReader.Close();

        this.close();

        return res;
    }

    public void fileToHisto(string fileName, out int [,] histoH,out int [,]
histoS,out int [,] histoI,
        out int [] histoWH, out int [] histoWS,out int [] histoWI, Operator
process)
    {

        histoH = new int[8,8,process.hueBins];
        histoS = new int[8,8,process.satBins];
        histoI = new int[8,8,process.intBins];

        histoWH = new int[process.hueBins];
        histoWS = new int[process.satBins];
        histoWI = new int[process.intBins];

        System.IO.FileStream fs = new
System.IO.FileStream(fileName.Replace("\\BILDER\\", "\\HISTOGRAM\\"), FileMode.Open,
FileAccess.Read);
        BinaryReader r = new BinaryReader(fs);

        for (int x=0; x<8; x++)
            for (int y=0; y<8; y++)
            {
                for (int i = 0; i < process.hueBins; i++)
                    histoH[x,y,i]=r.ReadInt32();
                for (int i = 0; i < process.satBins; i++)
                {
                    histoS[x,y,i]=r.ReadInt32();
                    histoI[x,y,i]=r.ReadInt32();
                }
            }

        for (int i = 0; i < process.hueBins; i++)
            histoWH[i]=r.ReadInt32();
        for (int i = 0; i < process.satBins; i++)
        {
            histoWS[i]=r.ReadInt32();
            histoWI[i]=r.ReadInt32();
        }

        r.Close();
    }

```

```

        fs.Close();
    }

    public void histoToFile(string fileName, int [,] histoH, int [,] histoS,
int [,] histoI,
        int [] histoWH, int [] histoWS, int [] histoWI, Operator process)
    {
        //
        System.Windows.Forms.MessageBox.Show(fileName.Replace("\\bilder\\", "\\histogram\\"
));
        //          fileName.Replace
        System.IO.FileStream fs = new
System.IO.FileStream(fileName.Replace("\\BILDER\\", "\\HISTOGRAM\\"),
        FileMode.OpenOrCreate, FileAccess.Write);
        BinaryWriter w = new BinaryWriter(fs);

        // Add some text to the file.
        for (int x=0; x<8; x++)
            for (int y=0; y<8; y++)
            {
                for (int i = 0; i < process.hueBins; i++)
                    w.Write(histoH[x,y,i]);
                for (int i = 0; i < process.satBins; i++)
                {
                    w.Write(histoS[x,y,i]);
                    w.Write(histoI[x,y,i]);
                }
            }
        for (int i = 0; i < process.hueBins; i++)
            w.Write(histoWH[i]);
        for (int i = 0; i < process.satBins; i++)
        {
            w.Write(histoWS[i]);
            w.Write(histoWI[i]);
        }

        w.Close();
        fs.Close();
    }

    public int executeCommand(string Query)
    {
        OracleCommand myCommand = new OracleCommand(Query, this.connection);
        //myCommand.Connection.Open();
        return myCommand.ExecuteNonQuery();
        //myConnection.Close();
    }

    public int add_element(string DBnavn, string val)
    {
        this.open();

        int r = executeCommand("INSERT INTO "+ DBnavn +"(INDICE_HIST)
VALUES('"+val+"')");
        this.close();
        return r;
    }

    public int add_histogram(string fileName, int [,] histoH, int [,] histoS,
int [,] histoI,
        int [] histoWH, int [] histoWS, int [] histoWI, Operator p)
    {
        histoToFile(fileName+".his", histoH, histoS, histoI,
        histoWH, histoWS, histoWI, p );
        return executeCommand("INSERT INTO INDICES(FILENAME, HISTOGRAM_FILE)

```

```
VALUES('+fileName+', '+ fileName+".his'");
```

```
    //return 0;
```

```
    }
```

```
    }
```

```
}
```

Generator class

This is the modified generator class from [2].

```
using System;
using System.IO;
using System.Runtime.Remoting;
using System.Threading;
using System.Reflection;
using System.Collections;
using System.CodeDom;
using System.CodeDom.Compiler;
using Microsoft.CSharp;

namespace Classify_tryout
{
    /// <summary>
    /// Summary description for Generator.
    /// </summary>
    public class Generator
    {
        // GenerationN will create the GeneratedCodeN namespace.
        // it will have classes named GeneratedClass1 ..
        // GeneratedClassN where N is number of different types of
        // program to be generates. GeneratedClass<N> will
        // inherit from
        // mBaseType, the class passed to the Generator constructor.
        private static string mNamespacePrefix = "GeneratedCode";

        // the target type
        private Type mBaseType;

        // use a fixed seet so we can replicate runs
        private Random mRandom = new Random(5000);

        private ArrayList mTerminals; // will hold MethodInfo
        public ArrayList Terminals
        {
            get{return mTerminals;}
        }

        public enum FunctionType{IfStatement, ChildCount};
        // a function is either an if statement, with a boolean
        // valued MethodInfo and two slots. Or else its a series of
        // mChildCount slots;
        public class Function
        {
            private FunctionType mFunctionType;
            public FunctionType FunctionType
            {
                get{return mFunctionType;}
            }
            private MethodInfo mMethodInfo;
            public MethodInfo MethodInfo {get { return mMethodInfo;}}

            private int mChildCount;
            public int ChildCount { get { return mChildCount;}}
            public Function(MethodInfo MethodInfo)
            {
                mFunctionType = FunctionType.IfStatement;
                mMethodInfo = MethodInfo;
                mChildCount = 2;
            }
            public Function(int ChildCount)
            {
                mFunctionType = FunctionType.ChildCount;
                mChildCount = ChildCount;
            }
        }
    }
}
```

```

    }
}
// will hold array of functions
private ArrayList mFunctions = new ArrayList();
public ArrayList Functions { get{return mFunctions;}}

// random functions and terminals are used to construct
// the initial population
private Function GetRandomFunction()
{
    return (Function)mFunctions[mRandom.Next(0,mFunctions.Count-1)];
}

private object GetRandomTerminal()
{
    return mTerminals[mRandom.Next(0,mTerminals.Count-1)];
}

private object GetRandomTerminalOrFunction()
{
    int index =
        mRandom.Next(0, mFunctions.Count + mTerminals.Count);
    if (index >= mFunctions.Count)
    {
        return mTerminals[index-mFunctions.Count];
    }
    else
    {
        return mFunctions[index];
    }
}

private void AddTreeNodeToList
(ExpressionTree Tree, ArrayList List)
{
    List.Add(Tree);
    if (Tree.Node.GetType() == typeof(Function))
    {
        foreach(ExpressionTree t in Tree.Children)
        {
            AddTreeNodeToList(t, List);
        }
    }
}

// treelist is used at crossover. Copies the
// tree to an ArrayList, so we can select
// a random crossover point
private ArrayList TreeList(ExpressionTree Tree)
{
    ArrayList mList = new ArrayList();
    AddTreeNodeToList(Tree, mList);
    return mList;
}

// copy SourceTree, except when the subtree node equals
// SourceRemoveSubTree replace it with TargetReplaceSubTree
private ExpressionTree TreeCombine
(ExpressionTree SourceTree,
ExpressionTree SourceRemoveSubTree,
ExpressionTree TargetReplaceSubTree)
{
    ExpressionTree mTreeCopy = SourceTree;

    if (SourceTree == SourceRemoveSubTree )
    {
        mTreeCopy = ExpressionTree.ReplaceNodeWithCopy
            (SourceTree, TargetReplaceSubTree);
    }
}

```

```

else
    if (mTreeCopy.Node.GetType() == typeof(Function))
    {
        for(int i = mTreeCopy.Children.GetLowerBound(0);
            i <= mTreeCopy.Children.GetUpperBound(0); i++)
        {
            ExpressionTree child =
                TreeCombine
                    (mTreeCopy.Children[i],
                     SourceRemoveSubTree, TargetReplaceSubTree);
            mTreeCopy.Children[i] = child;
            // apply mutation
            if (mTreeCopy.Children[i].GetType() !=
                typeof(Function))
            {
                if (mRandom.Next(0,100) < 2)
                {
                    mTreeCopy.Children[i].MutateTerminal
                        ((MethodInfo)GetRandomTerminal());
                }
            }
        }
    }
    return mTreeCopy;
}

//select a pair of parents for breeding
private void SelectParentsAndCloneThem
    (ArrayList Pool,
     // this is the number of potential parents
     // we reproduced from the last generation
     int PoolParentLimit,
     out int Parent1Index,
     out ExpressionTree Parent1,
     out int Parent2Index,
     out ExpressionTree Parent2)
{
    int TotalFitness = 0;
    for (int i = 0; i < PoolParentLimit; i++)
    {
        TotalFitness = TotalFitness +
            Operator_gp.TerminationReport.Fitness
                (((TypeExecutionResult)Pool[i]).mResults);
    }
    Parent1Index = -1;
    Parent2Index = -1;
    int ThisFitness = 0;

    // use fitness-based selection. A parent will be
    // selected at random, where more fit parents
    // have a greater chance of being selected
    while (Parent1Index == -1 | Parent2Index == -1)
    {
        for (int i = 0; i < PoolParentLimit; i++)
        {
            ThisFitness =
                Operator_gp.TerminationReport.Fitness
                    (((TypeExecutionResult)Pool[i]).mResults);
            double range = Convert.ToDouble(ThisFitness)/
                Convert.ToDouble(TotalFitness);
            double random = mRandom.NextDouble();
            // more fit parents will have a greater range
            if (random < range)
            {
                if (Parent1Index == -1) {Parent1Index = i;}
                else {Parent2Index = i;}
            }
        }
    }
}

```

```

    }

    // clone the parents, because the TreeCombine operation
    // will overwrite them.
    Parent1 = ExpressionTree.TreeDeepClone
        (((TypeExecutionResult)Pool[Parent1Index]).mTree);
    Parent2 = ExpressionTree.TreeDeepClone
        (((TypeExecutionResult)Pool[Parent2Index]).mTree);
}

// perform crossover operation, generating new children
// that are mixtures of parent1 and parent2. Select a
// random node from each parent and switch random subress
// within them to form child1 and child2
private void Crossover
    (ExpressionTree Parent1, ExpressionTree Parent2,
    out ExpressionTree Child1, out ExpressionTree Child2 )
{
    ArrayList parent1Nodes = TreeList(Parent1);
    ArrayList parent2Nodes = TreeList(Parent2);
    // select a random subtree from each
    int crossoverpoint1 = mRandom.Next(0,parent1Nodes.Count-1);
    int crossoverpoint2 = mRandom.Next(0,parent2Nodes.Count-1);

    Child1 = TreeCombine
        (Parent1,
        (ExpressionTree)parent1Nodes[crossoverpoint1],
        (ExpressionTree)parent2Nodes[crossoverpoint2]);

    Child2 = TreeCombine
        (Parent2,
        (ExpressionTree)parent2Nodes[crossoverpoint2],
        (ExpressionTree)parent1Nodes[crossoverpoint1]);
}

// ExpressionTree is the "genotype" of each ant.
// it defines the generated expression in its
// Execute() method.
public class ExpressionTree
{
    private Object mNode;
    private Object mParent;
    public Object Node{ get { return mNode;}}
    private ExpressionTree[] mChildren;
    public ExpressionTree[] Children { get { return mChildren;}}

    private ExpressionTree(){}

    public void MutateTerminal(MethodInfo newTerminal)
    {
        if (this.GetType() != typeof(Function))
        {
            this.mNode = newTerminal;
        }
    }

    // clone the tree, but keep its parents
    public static ExpressionTree TreeDeepClone
        (ExpressionTree Source)
    {
        ExpressionTree newTree = new ExpressionTree();
        newTree.mNode = Source.mNode;
        newTree.mParent = Source.mParent;
        if ( Source.mChildren != null)
        {
            newTree.mChildren = new
                ExpressionTree[Source.mChildren.Length];
            for(int i = Source.mChildren.GetLowerBound(0);
                i <= Source.mChildren.GetUpperBound(0); i++)
            {

```

```

        newTree.mChildren[i] =
            TreeDeepClone(Source.mChildren[i]);
    }
}
return newTree;
}

// replace RemovedNode with a copy of InsertedNode
// but keep RemovedNode.mParent
public static ExpressionTree ReplaceNodeWithCopy
(ExpressionTree RemovedNode,
ExpressionTree InsertedNode)
{
    ExpressionTree newTree = new ExpressionTree();
    newTree.mNode = InsertedNode.mNode;
    newTree.mParent = RemovedNode.mParent;
    newTree.mChildren = InsertedNode.mChildren;
    return newTree;
}

// generate a random ExpressionTree from the set of
// terminals and nodes we found in mBaseType. The
// tree will be at most MaxDepth levels deep
public ExpressionTree
(int MaxDepth,
Generator Generator, Object Node, Object Parent)
{
    mNode = Node;
    mParent = Parent;
    if ( mNode.GetType() == typeof(Function) )
    {
        Function mFunction = (Function)mNode;
        mChildren = new
            ExpressionTree[mFunction.ChildCount];
        for( int i = 0; i < mFunction.ChildCount; i++)
        {
            if (MaxDepth > 0)
            {
                mChildren[i] = new ExpressionTree
                    (MaxDepth-1, Generator,
Generator.GetRandomTerminalOrFunction(),
                    this);
            }
            // if we've reached MaxDepth, choose only
            // terminals
            else
            {
                mChildren[i] = new ExpressionTree
                    (MaxDepth-1, Generator,
                    Generator.GetRandomTerminal(),
                    this);
            }
        }
    }
}

public override string ToString()
{
    if ( (System.Type)mNode.GetType() == typeof(Function) )
    {
        Function mFunction = (Function)mNode;
        if ( mFunction.FunctionType ==
            FunctionType.IfStatement )
        {
            return " if ( " +
                mFunction.MethodInfo.ToString() + " ) " +
                " then " + mChildren[0].ToString() +
                " else" + mChildren[1].ToString();
        }
        else
        {

```



```

        string mListOfFunctionCalls = "";
        for( int i = mChildren.GetLowerBound(0);
            i <= mChildren.GetUpperBound(0); i++)
        {
            mListOfFunctionCalls +=
                mChildren[i].ToString();
            if (i != mChildren.GetUpperBound(0))
            {
                mListOfFunctionCalls += " ; ";
            }
        }
        return mListOfFunctionCalls;
    }
}
else
{
    return ((MethodInfo)mNode).ToString();
}
}

// create a generator for the baseclass. It will create
// terminals for each public void parameterless method
// in the BaseClass. It will create an if function for
// each boolean parameterless method. It will create
// functions Call(T,T) . . . Call(T1, T2, . . . ,TN) where
// N is MaxSteps
public Generator(Type BaseClass, int MaxSteps)
{
    mBaseType = BaseClass;
    mTerminals = new ArrayList();
    mFunctions = new ArrayList();
    // first create the terminals
    BindingFlags flags = BindingFlags.Public |
        BindingFlags.Instance;
    MethodInfo[] mBaseClassMethods =
        BaseClass.GetMethods(flags);
    foreach(MethodInfo mi in mBaseClassMethods)
    {
        // Terminals are parameterless void methods, except
        // for "Execute"
        if(mi.GetParameters().Length == 0 &&
            mi.ReturnType == typeof(void) &&
            mi.Name != "Execute" &&
            mi.Name != "evaluateDistance" &&
            mi.Name != "setRelevant" &&
            mi.Name != "stopClock")

        {
            mTerminals.Add(mi);
        }
        if(mi.GetParameters().Length == 0 &&
            mi.ReturnType == typeof(bool) &&
            mi.Name != "NotFinished")
        {
            mFunctions.Add(new Function(mi));
        }
        //if(mi.Name == "TurnLeft")

    }
    for(int i = 2; i<= MaxSteps; i++)
    {
        mFunctions.Add(new Function(i));
    }
    // mFunctions.Add(new Function(2));
}

// generate the statements that express the terminals and

```

```

// functions of the Tree
private CodeStatement[] GenerateStatements
(ExpressionTree Tree)
{
    CodeStatementCollection mCodeStatements =
        new CodeStatementCollection();
    Object mNode = Tree.Node;
    if ( (System.Type)mNode.GetType() == typeof(Function))
    {
        Function mFunction = (Function)mNode;

        if ( mFunction.FunctionType == FunctionType.IfStatement)
        {
            CodeSnippetExpression ifcondition =
                new CodeSnippetExpression
                    (mFunction.MethodInfo.Name.ToString() + "()");
            CodeStatement[] cstrue =
                GenerateStatements(Tree.Children[0]);
            CodeStatement[] csfalse =
                GenerateStatements(Tree.Children[1]);
            CodeConditionStatement ifStatement =
                new CodeConditionStatement
                    (ifcondition, cstrue, csfalse);
            mCodeStatements.Add(ifStatement);
        }
        else
        {
            for( int c = Tree.Children.GetLowerBound(0);
                c <= Tree.Children.GetUpperBound(0); c++)
            {
                CodeStatement[] cs =
                    GenerateStatements(Tree.Children[c]);
                for( int i = cs.GetLowerBound(0);
                    i <= cs.GetUpperBound(0); i++)
                {
                    mCodeStatements.Add(cs[i]);
                }
            }
        }
    }
    else
    {
        mCodeStatements.Add
            (new CodeSnippetExpression
                (((MethodInfo)Tree.Node).
                Name.ToString() + "()"));
    }

    CodeStatement[] cstatements =
        new CodeStatement[mCodeStatements.Count];
    IEnumerator enumStatements =
        mCodeStatements.GetEnumerator();
    int statement = 0;
    while(enumStatements.MoveNext())
    {
        cstatements[statement] =
            (CodeStatement)enumStatements.Current;
        statement++;
    }

    return cstatements;
}

private CodeMemberMethod BuildExecute
(ExpressionTree Tree)
{
    // this is the first part of each execute method:
    // a try catch block where the Termination Report is
    // caught and the exception is saved in the public
    // member exDone.

```

```

CodeMemberMethod m = new CodeMemberMethod();
m.ReturnType = new CodeTypeReference(typeof(void));
m.Attributes = MemberAttributes.Public |
    MemberAttributes.Override;
m.Name = "Execute";

CodeMethodReferenceExpression loopCheckMethod = new
    CodeMethodReferenceExpression();
loopCheckMethod.MethodName = "NotFinished";

CodeMethodInvokeExpression loopCheck = new
    CodeMethodInvokeExpression(loopCheckMethod,
    new CodeExpression[0]);
CodeIterationStatement whileLoop =
    new CodeIterationStatement
    (new CodeSnippetStatement(""),
    loopCheck,
    new CodeSnippetStatement(""),
    GenerateStatements(Tree));
m.Statements.Add(whileLoop);

//Her legger vi distanse i classify
m.Statements.Add(new CodeSnippetExpression("evaluateDistance()"));
m.Statements.Add(new CodeSnippetExpression("setRelevant()"));
m.Statements.Add(new CodeSnippetExpression("stopClock()"));

    return m;
}

// build a subclass of mBaseType. It will have

// Builds the ant subclassIt that will have an Execute()
// method that
// has an infinite loop, repeatedly calling
// base class functions until the base class throws the
// terminating exception. The Execute() method catches the
// terminating exception and returns.
// It's declared public so the class can be rebuild for the
// display form
public CodeTypeDeclaration BuildClass
    (string ClassName, ExpressionTree Tree)
{
    CodeTypeDeclaration mType = new
        CodeTypeDeclaration(ClassName);

    // it inherits from mBaseType;
    mType.BaseTypes.Add(mBaseType.Name);
    mType.Members.Add(BuildExecute(Tree));
    return mType;
}

// a summary of eac ant. Its lineage, its name, its
// ExpressionTree and its terminating exception. The
// terminating exception also has the trail it left.
public class TypeExecutionResult : IComparable
{
    public string mTypeClassName;
    public string mParent1ClassName;
    public string mParent2ClassName;
    public ExpressionTree mTree;
    public Operator_gp.TerminationReport mResults;

    public TypeExecutionResult(string ClassName)
    {
        mTypeClassName = ClassName;
        mParent1ClassName = "";
        mParent2ClassName = "";
    }

    public TypeExecutionResult(

```

```

        string ClassName,
        string Parent1ClassName,
        string Parent2ClassName,
        ExpressionTree Tree)
    {
        mTypeClassName = ClassName;
        mParent1ClassName = Parent1ClassName;
        mParent2ClassName = Parent2ClassName;
        mTree = Tree;
    }

    // IComparable implementation for sorting
    public int CompareTo(object obj)
    {
        if(obj is TypeExecutionResult)
        {
            TypeExecutionResult temp = (TypeExecutionResult) obj;

            return mResults.CompareTo(temp.mResults);
        }

        throw new ArgumentException
            ("object is not a TypeExecutionResult");
    }
}

// a result for each generation.
public class ExecutionResults
{
    public int mGenerationNumber;
    public ArrayList mBuildResults = new ArrayList();
    public ArrayList mTypeExecutionResults = new ArrayList();

    public void TypeExecutionStats
        (out double AverageFitness, out int BestFitness,
         out int WorstFitness)
    {
        int mTotalFitness = 0;
        BestFitness = 0;
        WorstFitness = Int16.MaxValue;
        int mThisFitness;

        foreach(TypeExecutionResult t in mTypeExecutionResults)
        {
            mThisFitness = Operator_gp.TerminationReport.Fitness
                (t.mResults);
            mTotalFitness = mTotalFitness + mThisFitness;
            if (mThisFitness > BestFitness)
            {BestFitness = mThisFitness;}
            if (mThisFitness < WorstFitness)
            {WorstFitness = mThisFitness;}
        }
        AverageFitness = Convert.ToDouble(mTotalFitness)/
            mTypeExecutionResults.Count;
    }

    public override string ToString()
    {
        double mAverageFitness;
        int mBestFitness, mWorstFitness;
        TypeExecutionStats
            (out mAverageFitness, out mBestFitness,
             out mWorstFitness);
        return "Gen. " + mGenerationNumber.ToString() +
            " best:" + mBestFitness.ToString() +
            " worst:" + mWorstFitness.ToString() +
            " avg:" + mAverageFitness.ToString();
    }

    public string ToLogString()
    {

```

```

        // for output to log file
        double mAverageFitness;
        int mBestFitness, mWorstFitness;
        TypeExecutionStats
            (out mAverageFitness, out mBestFitness,
             out mWorstFitness);
        return mGenerationNumber.ToString() +
            "," + mBestFitness.ToString() +
            "," + mWorstFitness.ToString() +
            "," + mAverageFitness.ToString();
    }
}

// each ant class name has the generation and the ant number
private string NewAntName
    ( int GenerationNumber, int ProgramNumber)
{
    return mBaseType.ToString().Replace
        (mBaseType.Namespace+".", "") +
        "_Gen" + GenerationNumber.ToString() +
        "_Ant" + ProgramNumber.ToString();
}

// do a new generation. 10% of the last generation will be
// replicated to this one. The best 10% will be selected.
// Remaining slots will be filled by breded the replicated
// ants
private void ReplicateAndBreedChildren(int GenerationNumber,
    int NumberOfPrograms,
    int MaxTreeDepth,
    ArrayList LastGenerationResults,
    out ArrayList ThisGenerationResults)
{
    ThisGenerationResults = new ArrayList();

    if (LastGenerationResults.Count == 0)
    {
        // first generation - just generate random trees
        for(int i = 0; i<= NumberOfPrograms-1;i++)
        {
            ExpressionTree newTree = new
                ExpressionTree
                (MaxTreeDepth,this,
                 GetRandomFunction(),null);
            string temp = NewAntName
                ( GenerationNumber, i);
            ThisGenerationResults.Add
                (new TypeExecutionResult(
                 NewAntName(GenerationNumber, i)
                 , "", "", newTree));
        }
    }
    else
    {
        // subsequent generation - keep
        // LastGenerationResults and breed them
        // to fill available slots
        int numToKeep = NumberOfPrograms/10;
        int numKept = 0;
        LastGenerationResults.Sort();
        LastGenerationResults.Reverse();
        IEnumerator enumLastGenerationResults =
            LastGenerationResults.GetEnumerator();
        while(enumLastGenerationResults.MoveNext() &
            numKept++ < numToKeep)
        {
            TypeExecutionResult r = (TypeExecutionResult)
                enumLastGenerationResults.Current;
            ThisGenerationResults.Add(r);
        }
    }
}

```

```

    }

    // breed two children for each pair of parents
    // until we've filled all the slots
    for(int i = ThisGenerationResults.Count;
        i<= NumberOfPrograms-2; i=i+2)
    {
        ExpressionTree child1, child2;
        ExpressionTree Parent1, Parent2;
        int Parent1Index, Parent2Index;

        SelectParentsAndCloneThem
            (ThisGenerationResults, numKept - 1,
             out Parent1Index, out Parent1,
             out Parent2Index, out Parent2);

        Crossover
            (Parent1, Parent2, out child1, out child2);

        TypeExecutionResult childResult1, childResult2;

        childResult1 = new TypeExecutionResult
            (
                NewAntName(GenerationNumber, i),
                ((TypeExecutionResult)
                 ThisGenerationResults[Parent1Index])
                .mTypeClassName,
                ((TypeExecutionResult)
                 ThisGenerationResults[Parent2Index])
                .mTypeClassName,
                child1
            );

        childResult2 = new TypeExecutionResult
            (
                NewAntName(GenerationNumber, i+1),
                ((TypeExecutionResult)
                 ThisGenerationResults[Parent1Index])
                .mTypeClassName,
                ((TypeExecutionResult)
                 ThisGenerationResults[Parent2Index])
                .mTypeClassName,
                child2
            );

        ThisGenerationResults.Add(childResult1);
        ThisGenerationResults.Add(childResult2);
    }
}

// create a new generation, generate code, execute it
// and return the results
public ExecutionResults NewGeneration
    (//TODO::

     int algH,
     int algP,
     string[] dbHistos,
     Operator.histogrammer[] histogrammer,
     string[] rHistos,
     int[, ,] pH,
     int[, ,] pS,
     int[, ,] pI,
     int[] wH,
     int[] wS,
     int[] wI,

     int StepLimit,
     int RelGoal,
     Operator op,

```

```

//DbManagement database,
string klassifik,

    //Cell[,] grid,
    ArrayList LastGenerationResults,
    int GenerationNumber,
    int NumberOfPrograms,
    int MaxTreeDepth
)
{
    ExecutionResults mExecutionResults = new ExecutionResults();
    mExecutionResults.mGenerationNumber = GenerationNumber;
    ArrayList thisGenerationResults;

    ReplicateAndBreedChildren(
        GenerationNumber,
        NumberOfPrograms,
        MaxTreeDepth,
        LastGenerationResults,
        out thisGenerationResults);

    CodeCompileUnit mCompileUnit = new CodeCompileUnit();
    mCompileUnit.ReferencedAssemblies.Add
        (mBaseType.Assembly.FullName);
    string mGeneratedNamespaceName =
        mNamespacePrefix + GenerationNumber.ToString();
    CodeNamespace mNamespace = new CodeNamespace
        (mGeneratedNamespaceName);
    // TO_DO: Cleaner to get import name from type.
    //mNamespace.Imports.Add(new CodeNamespaceImport("CodeDom"));
    mNamespace.Imports.Add(new CodeNamespaceImport("System"));
    mNamespace.Imports.Add(new CodeNamespaceImport("Classify_tryout"));

    mCompileUnit.Namespaces.Add(mNamespace);

    IEnumerator resultEnum =
        thisGenerationResults.GetEnumerator();
    while(resultEnum.MoveNext())
    {
        TypeExecutionResult r = (TypeExecutionResult)
            resultEnum.Current;
        mNamespace.Types.Add
            (BuildClass(r.mTypeClassName, r.mTree));
    }

    // Create the generated .cs file
    CodeDomProvider provider = new CSharpCodeProvider();
    // Obtain an ICodeGenerator from a CodeDomProvider class.
    ICodeGenerator gen = provider.CreateGenerator();
    // Create a TextWriter to a StreamWriter to an output file.
    IndentedTextWriter tw = new
        IndentedTextWriter(new StreamWriter
            (mGeneratedNamespaceName + ".cs", false), " ");
    // Generate source code using the code generator.
    gen.GenerateCodeFromCompileUnit
        (mCompileUnit, tw, new CodeGeneratorOptions());
    // Close the output file.
    tw.Close();

    // now compile it
    ICodeCompiler compiler = provider.CreateCompiler();
    CompilerParameters cp = new CompilerParameters();
    cp.GenerateInMemory = true;
    cp.GenerateExecutable = false;
    cp.ReferencedAssemblies.Add("Classify_tryout.exe");
    cp.OutputAssembly = mGeneratedNamespaceName + ".dll";

    CompilerResults cr = compiler.CompileAssemblyFromFile
        (cp, mGeneratedNamespaceName + ".cs");
    for( int i=0; i<cr.Output.Count; i++ )
        mExecutionResults.mBuildResults.Add( cr.Output[i] );
}

```

```

        //Feil av en eller annen årsak
        for( int i=0; i<cr.Errors.Count; i++ )

mExecutionResults.mBuildResults.Add(cr.CompiledAssembly.FullName);

mExecutionResults.mBuildResults.Add(cr.CompiledAssembly.FullName.ToString());
    Assembly a = cr.CompiledAssembly;

    // take each type in the assembly, and invoke its Execute() method
    Type[] mytypes = a.GetTypes();
    foreach(Type t in mytypes)
    {
        Operator_gp a2 = (Operator_gp)Activator.CreateInstance(t);

        // the ant will stop after 400 steps, or when
        // its gathered its FoodGoal
        a2.Init(algH, algP, dbHistos,
histogrammer,rHistos,pH,pS,pI,wH,wS,wI,120, RelGoal, op,/* database,*/ klassifik);
        a2.Execute();

        // when the Execute() method finishes, get
        // its terminating exception to save its
        // results
        Operator_gp.TerminationReport doneReport =

a2.mConcludingReport;

        // look up the class names, and copy the
        // terminating exception to the results
        // for the class
        bool found = false;
        resultEnum = thisGenerationResults.GetEnumerator();
        while(resultEnum.MoveNext() & !found)
        {
            TypeExecutionResult loopItem =
((TypeExecutionResult)resultEnum.Current);
            string typeName = t.ToString();
            if (t.ToString().EndsWith(loopItem.mTypeClassName))
            {
                loopItem.mResults = doneReport;
                found = true;
            }
        }
    }

    // return results in order, from best fitness to least fitness
    thisGenerationResults.Sort();
    thisGenerationResults.Reverse();
    mExecutionResults.mTypeExecutionResults = thisGenerationResults;
    return mExecutionResults;
}
}
}

```


Genetic Program

Here are all the function, terminals, fitness and report methods for the genomes.

```
using System;
using System.Timers;
using System.Collections;
using System.CodeDom;
using System.Windows.Forms;

namespace Classify_tryout
{
    /// <summary>
    /// Summary description for Operator_gp.
    /// </summary>
    public abstract class Operator_gp
    {
        //private
        public string[] histos; //histogrammer i DB

        private string[] best10_name; //De 10 nærmeste bildene
        private float[] best10_dist;
        public double[] HHWeights;
        public double[] HSWeights;
        public double[] HIWeights;

        public double[] PHWeights;
        public double[] PSWeights;
        public double[] PIWeights;

        public Operator.histogrammer[] learning_histos;

        private double[, ,] NhistoPH;
        private int indexPH=0;
        private double[, ,] NhistoPS;
        private int indexPS=0;
        private double[, ,] NhistoPI;
        private int indexPI=0;

        private double[] NhistoWH;
        private int indexWH=0;
        private double[] NhistoWS;
        private int indexWS=0;
        private double[] NhistoWI;
        private int indexWI=0;

        public double[] resNhistoPH; //TODO: Ordne res'er til NhistoWH..WI, fullføre
classify.

        public double[] resNhistoPS;
        public double[] resNhistoPI;

        public double[] resNhistoWH; //TODO: Ordne res'er til NhistoWH..WI, fullføre
classify.

        public double[] resNhistoWS;
        public double[] resNhistoWI;
        public double resNhistoH=0;
        public double resNhistoS=0;
        public double resNhistoI=0;
        public double resNhele=0;

        public string[] relHistos;

        public double Relevant = 0; //number of relevant pictures among the 10
```

```

private int Time = 0; //tiden algoritmen bruker

public int[, ,] histoPH; //Part histogram
public int[, ,] histoPS;
public int[, ,] histoPI;

public int[] histoWH; //whole histogram
public int[] histoWS;
public int[] histoWI;

public int algH; //valgt algoritme for hele bildet (-1..2)
public int algP; //valgt algoritme for segmentert bilde (-1..2)

private float wPH; //Vektor for segmentert bilde
private float wPS;
private float wPI;

private float wWH; //Vektor for hele bilde
private float wWS;
private float wWI;

public int[] pX;
public int[] pY;

public Operator processing;
public DbManagement db;

private int mTotalSteps;

private int mStepLimit;

private string klassifikasjonsNavn;

private int goal; // = 10 relevante bilder
private bool mDone = false;

public Classify_tryout.Operator_gp.TerminationReport mConcludingReport;
abstract public void Execute();

private System.Timers.Timer Clock;

//abstract public void Execute();

public void Timer_Tick(object sender, ElapsedEventArgs eArgs)
{
    if(sender==Clock)
    {
        Time++;
    }
}

public Operator_gp()
{
    //
    // TODO: Add constructor logic here
    //
}

public class TerminationReport : IComparable
{

```

```

public double mRelevant;
public double[] wH;
public double[] wS;
public double[] wI;
public double mMax;
public double mMin;

public int[] X;
public int[] Y;

public string[] histogrammer;

private TerminationReport()
{
}

public static TerminationReport GoalReport(int RelevantGoal)
{
    TerminationReport goalReport =
        new TerminationReport();
    goalReport.mRelevant = RelevantGoal;
    return goalReport;
}

public static int Fitness(TerminationReport RunResult)
{
    // Relevante bilder * 1000 slik at relevans alltid har
    // Legger på 1000 for å alltid få positivt resultat.
    // RunResult.mTime +1000;
    return (int) RunResult.mRelevant ;//(RunResult.mRelevant*1000)
}

public TerminationReport(double Relevant,double[] vH, double[] vS,
double[] vI, double max, double min, int[] x, int[] y, string[] histos)
{
    mRelevant = Relevant;
    wH = vH;
    wS = vS;
    wI = vI;
    mMax = max;
    mMin = min;
    X = x;
    Y = y;
    histogrammer = histos;

    //mFoodGathered = FoodGathered;
    //mTrail = Trail;
}

// IComparable implementation for sorting
// Generations conclude with ArrayLists of terminating
// exceptions. We need this to use the ArrayList
// sort method.
public int CompareTo(object obj)
{
    if(obj is TerminationReport)
    {
        TerminationReport temp = (TerminationReport) obj;
        return Fitness(this).CompareTo(Fitness(temp));
    }
    throw new ArgumentException("object is not a
TerminationReport");
}

// String for lastRun.csv run summary file

```

```

public String ToLogString()
{
    string rH = "";
    string rS = "";
    string rI = "";

    string pBlocks = "";

    for (int i = 0; i<wH.Length; i++)
        rH += wH[i].ToString()+" ";
    for (int i = 0; i<wS.Length; i++)
        rS += wS[i].ToString()+" ";
    for (int i = 0; i<wI.Length; i++)
        rI += wI[i].ToString()+" ";

    for (int i = 0; i<X.Length; i++)
        pBlocks += "\nBilde: "+histogrammer[i]+" , blokker brukt:
X:"+X[i].ToString()+" Y:"+Y[i].ToString();

    string res="Vekter brukt (hele) H : "+rH+" \nS : "+rS+" \nI
: "+rI+
"\nAvstanden mellom min og max er
"+mRelevant.ToString()+"\n"+
"\nMax = "+mMax.ToString()+" Min =
"+mMin.ToString()+"\n"+
pBlocks;

    return res;
}

public bool NotFinished()
{
    if (!mDone)
    {
        if (mTotalSteps >= mStepLimit | Relevant == goal)
        {
            mDone = true;
        }
    }
    return ! mDone;
}

public void Init(int aH, int aP, string[] dbHistos, Operator.histogrammer[]
dbHistogrammer, string[] rHistos, int[, ,] pH, int[, ,] pS, int[, ,] pI, int[] wH, int[] wS,
int[] wI, int StepLimit, int RelGoal, Operator op/*, DbManagement database*/, string
klassifik)
{
    algH = aH;
    algP = aP;
    histos = new string[dbHistos.Length];
    klassifikasjonsNavn = klassifik;
    relHistos = new string[rHistos.Length];
    relHistos = rHistos;

    //for (int i = 0; i<dbHistogrammer.Length; i++)
    learning_histos = new Operator.histogrammer[dbHistos.Length];

    learning_histos = dbHistogrammer;

    histos = dbHistos;
}

```

```

best10_dist = new float[histos.Length];

pX = new int[learning_histos.Length];
pY = new int[learning_histos.Length];

Clock = new System.Timers.Timer(1000); //setter tid til sekund
Clock.Elapsed += new ElapsedEventHandler(Timer_Tick);
Clock.Start();

processing = new Operator();

histoPH = pH;
histoPS = pS;
histoPI = pI;

histoWH = wH;
histoWS = wS;
histoWI = wI;

wPH = (float) 1 / (float) 3; //Vekter for segmentert bilde
wPS = (float) 1 / (float) 3;
wPI = (float) 1 / (float) 3;

wWH = (float) 1 / (float) 3; //Vekter for hele bilde
wWS = (float) 1 / (float) 3;
wWI = (float) 1 / (float) 3;

HHWeights = new double[op.hueBins];
HSWeights = new double[op.satBins];
HIWeights = new double[op.intBins];

PHWeights = new double[op.hueBins];
PSWeights = new double[op.satBins];
PIWeights = new double[op.intBins];

for (int i = 0; i<op.hueBins; i++)
{
    HHWeights[i]=1.0;
    PHWeights[i]=1.0;
}

for (int i = 0; i<op.satBins; i++)
{
    HSWeights[i]=1.0;
    PSWeights[i]=1.0;
}
for (int i = 0; i<op.intBins; i++)
{
    HIWeights[i]=1.0;
    PIWeights[i]=1.0;
}

NhistoPH = new double[8,8,op.hueBins];
NhistoPS = new double[8,8,op.satBins];
NhistoPI = new double[8,8,op.intBins];

NhistoWH = new double[op.hueBins];
NhistoWS = new double[op.satBins];
NhistoWI = new double[op.intBins];

op.normSegHisto(8,8,op.hueBins,pH, out NhistoPH);
op.normSegHisto(8,8,op.satBins,pS, out NhistoPS);

```

```

        op.normSegHisto(8,8,op.intBins,pI, out NhistoPI);

        op.normHisto(wH, out NhistoWH);
        op.normHisto(wS, out NhistoWS);
        op.normHisto(wI, out NhistoWI);

        mStepLimit = 1;//StepLimit - Testet values showed that approx 400
returned best results.
        goal = RelGoal;

        mTotalSteps = 0;
        indexWH = 9;
        indexWS = 1;
        indexWI = 1;

        indexPH = 9;
        indexPS = 1;
        indexPI = 1;

    }

    public override string ToString()
    {
        return "Funnet " + Relevant.ToString() + " relevante bilder " + " På
" + Time.ToString() + " Sekunder" ;
    }

    //endret til hele tall fra 0.1

    public void decHighest()
    {
        double[] max=new double[HHWeights.Length];

        int[] x = new int[HHWeights.Length];

        for (int a = 0; a < learning_histos.Length; a++)
            for (int i = 0; i<HHWeights.Length; i++)
                {
                    x[i] =i;

                    max[i] += learning_histos[a].histoWH[i];

                }

        Array.Sort(max, x);
        indexWH = x[HHWeights.Length-1];//[HHWeights.Length-2];
        HHWeights[indexWH]-= 0.1;
        mTotalSteps++;

    }

    public void decSecHighest()
    {
        double[] max=new double[HHWeights.Length];

        int[] x = new int[HHWeights.Length];

        for (int a = 0; a < learning_histos.Length; a++)
            for (int i = 0; i<HHWeights.Length; i++)

```

```

        {
            x[i] =i;

            max[i] += learning_histos[a].histoWH[i];

        }

Array.Sort(max, x);
indexWH = x[HHWeights.Length-2];//[HHWeights.Length-2];
HHWeights[indexWH]-= 0.1;
mTotalSteps++;

}
public void decThirdHighest()
{
    double[] max=new double[HHWeights.Length];

    int[] x = new int[HHWeights.Length];

    for (int a = 0; a < learning_histos.Length; a++)
        for (int i = 0; i<HHWeights.Length; i++)
            {
                x[i] =i;

                max[i] += learning_histos[a].histoWH[i];

            }

    Array.Sort(max, x);
    indexWH = x[HHWeights.Length-3];//[HHWeights.Length-2];
    HHWeights[indexWH]-= 0.1;
    mTotalSteps++;

}

public void incHighest()
{
    double[] max=new double[HHWeights.Length];

    int[] x = new int[HHWeights.Length];

    for (int a = 0; a < learning_histos.Length; a++)
        for (int i = 0; i<HHWeights.Length; i++)
            {
                x[i] =i;

                max[i] += learning_histos[a].histoWH[i];

            }

    Array.Sort(max, x);
    indexWH = x[HHWeights.Length-1];//[HHWeights.Length-2];
    HHWeights[indexWH]+= 0.1;
    mTotalSteps++;

}

public void incSecHighest()
{
    double[] max=new double[HHWeights.Length];

    int[] x = new int[HHWeights.Length];

    for (int a = 0; a < learning_histos.Length; a++)

```

```

        for (int i = 0; i<HHWeights.Length; i++)
        {
            x[i] =i;

            max[i] += learning_histos[a].histoWH[i];

        }

        Array.Sort(max, x);
        indexWH = x[HHWeights.Length-2];//[HHWeights.Length-2];
        HHWeights[indexWH]+= 0.1;
        mTotalSteps++;
    }
}

public void incThirdHighest()
{
    double[] max=new double[HHWeights.Length];

    int[] x = new int[HHWeights.Length];

    for (int a = 0; a < learning_histos.Length; a++)
        for (int i = 0; i<HHWeights.Length; i++)
        {
            x[i] =i;

            max[i] += learning_histos[a].histoWH[i];

        }

        Array.Sort(max, x);
        indexWH = x[HHWeights.Length-3];//[HHWeights.Length-2];
        HHWeights[indexWH]+= 0.1;
        mTotalSteps++;
    }

}

public void incLowest()
{
    double[] max=new double[HHWeights.Length];

    int[] x = new int[HHWeights.Length];

    for (int a = 0; a < learning_histos.Length; a++)
        for (int i = 0; i<HHWeights.Length; i++)
        {
            x[i] =i;

            max[i] += learning_histos[a].histoWH[i];

        }

        Array.Sort(max, x);
        indexWH = x[0];//[HHWeights.Length-2];
        HHWeights[indexWH]+= 0.1;
        mTotalSteps++;
    }

}

public void incSecLowest()
{
    double[] max=new double[HHWeights.Length];

    int[] x = new int[HHWeights.Length];

```



```

        for (int a = 0; a < learning_histos.Length; a++)
            for (int i = 0; i < HHWeights.Length; i++)
                {
                    x[i] = i;

                    max[i] += learning_histos[a].histoWH[i];
                }

        Array.Sort(max, x);
        indexWH = x[1]; // [HHWeights.Length-2];
        HHWeights[indexWH] += 0.1;
        mTotalSteps++;
    }
    public void incThirdLowest()
    {
        double[] max = new double[HHWeights.Length];

        int[] x = new int[HHWeights.Length];

        for (int a = 0; a < learning_histos.Length; a++)
            for (int i = 0; i < HHWeights.Length; i++)
                {
                    x[i] = i;

                    max[i] -= learning_histos[a].histoWH[i];
                }

        Array.Sort(max, x);
        indexWH = x[2]; // [HHWeights.Length-2];
        HHWeights[indexWH] += 0.1;
        mTotalSteps++;
    }

    public void decLowest()
    {
        double[] max = new double[HHWeights.Length];

        int[] x = new int[HHWeights.Length];

        for (int a = 0; a < learning_histos.Length; a++)
            for (int i = 0; i < HHWeights.Length; i++)
                {
                    x[i] = i;

                    max[i] += learning_histos[a].histoWH[i];
                }

        Array.Sort(max, x);
        indexWH = x[0]; // [HHWeights.Length-2];
        HHWeights[indexWH] -= 0.1;
        mTotalSteps++;
    }

    public void decSecLowest()
    {
        double[] max = new double[HHWeights.Length];

        int[] x = new int[HHWeights.Length];

```

```

        for (int a = 0; a < learning_histos.Length; a++)
            for (int i = 0; i < HHWeights.Length; i++)
                {
                    x[i] = i;

                    max[i] += learning_histos[a].histoWH[i];

                }

        Array.Sort(max, x);
        indexWH = x[1]; // [HHWeights.Length-2];
        HHWeights[indexWH] -= 0.1;
        mTotalSteps++;
    }
    public void decThirdLowest()
    {
        double[] max = new double[HHWeights.Length];

        int[] x = new int[HHWeights.Length];

        for (int a = 0; a < learning_histos.Length; a++)
            for (int i = 0; i < HHWeights.Length; i++)
                {
                    x[i] = i;

                    max[i] -= learning_histos[a].histoWH[i];

                }

        Array.Sort(max, x);
        indexWH = x[2]; // [HHWeights.Length-2];
        HHWeights[indexWH] += 0.1;
        mTotalSteps++;
    }

    public void incPHue()
    {
        PHWeights[indexPH] += 1;
        if (PHWeights[indexPH] > 18)
            PHWeights[indexPH] = 18;
        mTotalSteps++;
    }

    public void incPSat()
    {
        PSWeights[indexPS] += 1;
        if (PSWeights[indexPS] > 3)
            PSWeights[indexPS] = 3;
        mTotalSteps++;
    }

    public void incPInt()
    {
        PIWeights[indexPI] += 1;
        if (PIWeights[indexPI] > 3)
            PIWeights[indexPI] = 3;
        mTotalSteps++;
    }

    public void incHHue()
    {
        HHWeights[indexWH] += 1;
        /*if (HHWeights[indexWH] > 18)

```

```

        HHWeights[indexWH] = 18;*/
        mTotalSteps++;
    }
    public void incHSat()
    {
        HSWeights[indexWS] += 1;
        /*
        if (HSWeights[indexWS] > 3)
            HSWeights[indexWS] = 3;*/
        mTotalSteps++;
    }
    public void incHInt()
    {
        HIWeights[indexWI] += 1;
        /*if (HIWeights[indexWI] > 3)
            HIWeights[indexWI] = 3;*/
        mTotalSteps++;
    }

    public void decHInt()
    {
        HIWeights[indexWI] -= 1;
        if (HIWeights[indexWI] <-3)
            HIWeights[indexWI] = -3;
        mTotalSteps++;
    }

    public void decHSat()
    {
        HSWeights[indexWS] -= 1;
        if (HSWeights[indexWS] <-3)
            HSWeights[indexWS] = -3;
        mTotalSteps++;
    }
    public void decHHue()
    {
        HHWeights[indexWH] -= 1;
        if (HHWeights[indexWH] <-18)
            HHWeights[indexWH] = -18;
        mTotalSteps++;
    }
    public void decPSat()
    {
        PSWeights[indexPS] -= 1;
        if (PSWeights[indexPS] <-3)
            PSWeights[indexPS] = -3;
        mTotalSteps++;
    }
    public void decPInt()
    {
        PIWeights[indexPI] -= 1;
        if (PIWeights[indexPI] <-3)
            PIWeights[indexPI] = -3;
        mTotalSteps++;
    }

}

public void incPH()
{
    indexPH++;

    if (indexPH >= processing.hueBins)
        indexPH=processing.hueBins-1;
    mTotalSteps++;
}

public void incPS()
{
    indexPS++;
}

```

```

        if (indexPS >= processing.satBins)
            indexPS=processing.satBins-1;
        mTotalSteps++;
    }
    public void incPI()
    {
        indexPI++;

        if (indexPI >= processing.intBins)
            indexPI=processing.intBins-1;
        mTotalSteps++;
    }

    public void incHH()
    {
        indexWH++;

        if (indexWH >= processing.hueBins)
            indexWH=processing.hueBins-1;
        mTotalSteps++;
    }

    public void incHS()
    {
        indexWS++;

        if (indexWS >= processing.satBins)
            indexWS=processing.satBins-1;
        mTotalSteps++;
    }

    public void incHI()
    {
        indexWI++;

        if (indexWI >= processing.intBins)
            indexWI=processing.intBins-1;
        mTotalSteps++;
    }

    public void decHS()
    {
        indexWS--;

        if (indexWS < 0)
            indexWS=0;
        mTotalSteps++;
    }

    public void decHI()
    {
        indexWI--;

        if (indexWI < 0)
            indexWI=0;
        mTotalSteps++;
    }

```

```

    }

    public void decPH()
    {
        indexPH--;

        if (indexPH < 0)
            indexPH=0;
        mTotalSteps++;

    }

    public void decPS()
    {
        indexPS--;

        if (indexPS < 0)
            indexPS=0;
        mTotalSteps++;

    }

    public void decPI()
    {
        indexPI--;

        if (indexPI < 0)
            indexPI=0;
        mTotalSteps++;

    }

    public double find_Class_nr()
    {
        resNhistoWH = new double[processing.hueBins]; //TODO: Ordne res'er til
        NhistoWH..WI, fullføre classify.
        resNhistoWS = new double[processing.satBins];
        resNhistoWI = new double[processing.intBins];

        resNhistoPH = new double[processing.hueBins]; //TODO: Ordne res'er
        til NhistoWH..WI, fullføre classify.
        resNhistoPS = new double[processing.satBins];
        resNhistoPI = new double[processing.intBins];

        resNhistoH=0;
        resNhistoS=0;
        resNhistoI=0;

        for (int i=0; i<HHWeights.Length; i++)
        {
            resNhistoWH[i] = 0;
            resNhistoPH[i] = 0;
            resNhistoH = 0;
        }
    }

```

```

for (int i=0; i<HSWeights.Length; i++)
{
    resNhistoWS[i] = 0;
    resNhistoPS[i] = 0;
    resNhistoS = 0;
}
for (int i=0; i<HIWeights.Length; i++)
{
    resNhistoWI[i] = 0;
    resNhistoPI[i] = 0;
    resNhistoI = 0;
}
resNhele = 0;

for (int i=0; i<HHWeights.Length; i++)
    for (int b = 0; b<learning_histos.Length; b++)
    {
        resNhistoWH[i] += Math.Abs( NhistoWH[i] -
            learning_histos[b].nhistoWH[i] )/
            ((NhistoWH[i]*NhistoWH[i])+0.000001);
        //      MessageBox.Show("bin "+i.ToString()+" =
"+learning_histos[a].resHH[i].ToString());
    }

for (int i=0; i<HSWeights.Length; i++)
    for (int b = 0; b<learning_histos.Length; b++)
    {
        resNhistoWS[i] += Math.Abs( NhistoWS[i] -
            learning_histos[b].nhistoWS[i] )/
            ((NhistoWS[i]*NhistoWS[i])+0.000001);
    }

for (int i=0; i<HIWeights.Length; i++)
    for (int b = 0; b<learning_histos.Length; b++)
    {
        resNhistoWI[i] += Math.Abs( NhistoWI[i] -
            learning_histos[b].nhistoWI[i] )/
            ((NhistoWI[i]*NhistoWI[i])+0.000001);
    }

for (int i=0; i<HHWeights.Length; i++)
    resNhistoH += resNhistoWH[i] * HHWeights[i];

for (int i=0; i<HSWeights.Length; i++)
    resNhistoS += resNhistoWS[i] * HSWeights[i];

for (int i=0; i<HIWeights.Length; i++)
    resNhistoI += resNhistoWI[i] * HIWeights[i];

resNhele= Math.Sqrt(resNhistoH * resNhistoH+
    resNhistoS * resNhistoS+
    resNhistoI * resNhistoI);

```

```

        //
        return resNhele;
    }

    public void evaluateDistance()
    {
        //    MessageBox.Show(algH.ToString());

        for (int a = 0; a<learning_histos.Length; a++)
        {

            for (int i=0; i<HHWeights.Length; i++)
            {
                learning_histos[a].resHH[i] = 0;
                learning_histos[a].resH = 0;
            }
            for (int i=0; i<HSWeights.Length; i++)
            {
                learning_histos[a].resHS[i] = 0;
                learning_histos[a].resS = 0;
            }

            for (int i=0; i<HIWeights.Length; i++)
            {
                learning_histos[a].resHI[i] = 0;
                learning_histos[a].resI = 0;
            }

            for (int x = 0; x<8; x++)
                for(int y = 0; y<8; y++)
                    for (int i=0; i<PHWeights.Length; i++)
                    {
                        learning_histos[a].resPH[x,y,i] = 0;
                        learning_histos[a].PresH = 0;
                    }
            for (int x = 0; x<8; x++)
                for(int y = 0; y<8; y++)

                    for (int i=0; i<PSWeights.Length; i++)
                    {
                        learning_histos[a].resPS[x,y,i] = 0;
                        learning_histos[a].PresS = 0;
                    }

            for (int x = 0; x<8; x++)
                for(int y = 0; y<8; y++)

                    for (int i=0; i<PIWeights.Length; i++)
                    {
                        learning_histos[a].resPI[x,y,i] = 0;
                        learning_histos[a].PresI = 0;
                    }

            learning_histos[a].resHele = 0;
            learning_histos[a].PresHele = 0;

            for (int b = 0; b<learning_histos.Length; b++)
            {

                learning_histos[a].resHele+=processing.distance(processing.addWeights(learning_his
                tos[a].histoWH,HHWeights),

                processing.addWeights(learning_histos[a].histoWS,HSWeights),

```

```

processing.addWeights(learning_histos[a].histoWI,HIWeights),
                    learning_histos[b].histoWH,
                    learning_histos[b].histoWS,
                    learning_histos[b].histoWI
                    ,algH);

        }

        for (int b = 0; b<learning_histos.Length; b++)
        {

                learning_histos[a].PresHele+=processing.bestDistance(processing.addWeights(learnin
g_histos[a].histoPH,PHWeights),

                processing.addWeights(learning_histos[a].histoPS,PSWeights),

                processing.addWeights(learning_histos[a].histoPI,PIWeights),
                    learning_histos[b].histoPH,
                    learning_histos[b].histoPS,
                    learning_histos[b].histoPI
                    ,algH, out pX[b], out pY[b]);

        }

    }
}

public void setRelevant()
{

    bool funnetH = false;
    bool funnetI = false;
    bool funnetS = false;

    for (int i = 0; i < HHWeights.Length; i++)
        if (HHWeights[i] > 0 ) funnetH = true;

    for (int i = 0; i < HSWeights.Length; i++)
        if (HSWeights[i] > 0 ) funnetS = true;
    for (int i = 0; i < HIWeights.Length; i++)
        if (HIWeights[i] > 0 ) funnetI = true;

    double max = -999999999999999;
    double min = 999999999999999;

    double avg=0;

    double tmp = 0;
    for (int i = 0; i < learning_histos.Length; i++)
    {
        avg+= learning_histos[i].resHele;

        tmp = Math.Sqrt(learning_histos[i].resHele *
learning_histos[i].resHele +
                    learning_histos[i].PresHele *
learning_histos[i].PresHele );
        if (tmp < min){

            min = tmp;}

        if (tmp > max)
            max = tmp;

    }
}

```



```

        avg /= learning_histos.Length;

        if ((max > 0) & funnetH & funnetS )
            Relevant = 99999999 - (max*1000); // avg; //(Math.Abs(max -
min)/100000);
        else
            Relevant = 1; //0 ikke tilatt

            mConcludingReport = new TerminationReport(Relevant,
HHWeights/*NhistoWH*/, HSWeights, HIWeights, max, min, pX, pY, histos);
            //MessageBox.Show("etter set relevant");
        }

        public void stopClock()
        {
            Clock.Stop();
        }

    }
}

```

Examples of evolved genetic program

Here is two genetic programs. The first is before the modification of the GP, the second one is after, and proved to be much more efficient.

Evolved genome before modification

```
public class Operator_gp_Gen116_Ant53 : Operator_gp {  
  
    public override void Execute() {  
        for (  
            ; NotFinished();  
        ) {  
            incHH();  
            decHHue();  
            incHI();  
            incHHue();  
            incHSat();  
            incHHue();  
            incHSat();  
            decHHue();  
            incHH();  
            decHHue();  
            incHH();  
            incHH();  
            incHH();  
            decHHue();  
            decHHue();  
            decHHue();  
            decHHue();  
            decHHue();  
            decHHue();  
            incHS();  
            decHHue();  
            decHInt();  
            decHSat();  
            decHHue();  
            decHHue();  
            decHI();  
            incHSat();  
            incHH();  
            incHS();  
            decHS();  
            decHHue();  
            decHHue();  
            decHHue();  
            decHInt();  
            decHI();  
            incHI();  
            decHSat();  
            decPS();  
            incHH();  
            incHS();  
            decHHue();  
            decHHue();  
            decHHue();  
            decHSat();  
            incHSat();  
        }  
    }  
}
```

decPS ();
decHHue ();
decPS ();
decHHue ();
dechS ();
decHHue ();
dechS ();
decHHue ();
decPS ();
inchS ();
decHHue ();
decPS ();
decHHue ();
deCHI ();
inchH ();
inchInt ();
dechSat ();
deCHI ();
decHH ();
decHH ();
decHHue ();
decHHue ();
inchS ();
decHH ();
decHHue ();
decHHue ();
inchH ();
deCHIInt ();
decHH ();
inchS ();
decHH ();
inchInt ();
decHHue ();
deCHI ();
decHHue ();
deCHIInt ();
deCHI ();
inchS ();
deCHI ();
decHHue ();
deCHI ();
dechS ();
decPS ();
decHHue ();
dechS ();
inchSat ();
decHHue ();
deCHIInt ();
decPS ();
deCHIInt ();
inchH ();
inchH ();
decHHue ();
deCHI ();
decHHue ();
inchInt ();
decHHue ();
inchH ();
decHHue ();
inchH ();
decHHue ();
inchInt ();
decHHue ();
inchH ();

decHHue();
incHHue();
incHH();
decHI();
decHHue();
decHI();
incHI();
decHH();
decHH();
decHHue();
decHHue();
incHSat();
incHI();
incHHue();
incHSat();
incHHue();
incHInt();
decHHue();
decPS();
decHHue();
incHH();
decHHue();
incHInt();
decHHue();
incHH();
decHHue();
incHHue();
decPS();
decHHue();
incHSat();
incHH();
decHHue();
incHSat();
incHSat();
decHInt();
decHHue();
incHHue();
decHHue();
decHHue();
decHHue();
decHHue();
decHInt();
incHHue();
decHH();
decHHue();
decHH();
decHH();
decHHue();
decHHue();
incHSat();
incHSat();
decHHue();
decHHue();
decHHue();
decHInt();
decHS();
decHInt();
incHS();
decHS();
incHSat();
decHHue();
decHInt();
decHS();
decHSat();
decHHue();

decHHue();
decHI();
incHSat();
incHH();
incHS();
dechS();
decHHue();
decHHue();
incHSat();
incHSat();
incHS();
decHH();
incHH();
decHHue();
decHHue();
decHHue();
decPS();
decHHue();
incHH();
decHHue();
decHHue();
incHSat();
decHHue();
decHHue();
incHSat();
decHHue();
decHI();
incHH();
decHHue();
decHHue();
dechSat();
incHS();
decHHue();
incHSat();
decHHue();
incHInt();
decHHue();
decHHue();
dechSat();
dechSat();
incHSat();
incHS();
incHH();
decHH();
decPS();
decHHue();
decHI();
incHH();
incHS();
incHInt();
decPS();
decHHue();
dechS();
incHSat();
incHS();
decHH();
decHHue();
decHInt();
decHHue();
decHInt();
incHSat();
incHS();
decHH();
decHH();

```

        decHH();
        incHS();
        decHH();
        decPS();
        decHInt();
        incHH();
        incHI();
        decHI();
        decHSat();
    }
    evaluateDistance();
    setRelevant();
    stopClock();
}
}

```

Resulting in weights:

```

H : 1, 1, 0,9, 1, 0,3, 0,1, 0, 0, 0, 1,38777878078145E-16, 0,3
,1,38777878078145E-16, 0,4, 0, 0,8, 0,8, 0,3, 0,1
S : 0,9, 1,4, 1,2
V : 1,2, 1, 1

```

Evolved genome after modification

```
public class Operator_gp_Gen99_Ant25 : Operator_gp {  
  
    public override void Execute() {  
        for (  
            ; NotFinished();  
        ) {  
            incHighest();  
            incHH();  
            incHH();  
            incHighest();  
            decSecLowest();  
            incHH();  
            decPS();  
            incHighest();  
            decHI();  
            incHighest();  
            decHS();  
            decHI();  
            incHH();  
            incHH();  
            incHI();  
            decSecLowest();  
            decHI();  
            decSecLowest();  
            decSecLowest();  
            incSecHighest();  
            incHighest();  
            incHH();  
            incHighest();  
            decHI();  
            incThirdHighest();  
            incThirdHighest();  
            incThirdHighest();  
            incThirdHighest();  
            incSecHighest();  
            incThirdHighest();  
            incThirdHighest();  
        }  
        evaluateDistance();  
        setRelevant();  
        stopClock();  
    }  
}
```

Resulting in the weights

```
H : 1, 1, 1, 1, 0,3, 1, 1, 1, 1, 1, 1, 1, 1, 1,4, 1,8, 1,6, 1, 1,  
S : 1, 1, 1  
V : 1, 1, 1
```