

## **Todo list**

# Skin detection in images using machine learning techniques for color and texture recognition

Stefan Sebastian

June 10, 2018

# Contents

<b>1</b>	<b>Introduction TODO</b>	<b>3</b>
1.1	Introduction . . . . .	3
1.2	Applications . . . . .	3
<b>2</b>	<b>Theoretical background</b>	<b>5</b>
2.1	Related work . . . . .	5
2.1.1	Models using color analysis . . . . .	5
2.1.2	Models using segmentation . . . . .	7
2.1.3	Models using textural features . . . . .	8
2.2	Image segmentation . . . . .	8
2.2.1	Thresholding . . . . .	9
2.2.2	Edge detection . . . . .	9
2.2.3	Clustering . . . . .	10
2.3	Skin detection by color . . . . .	12
2.3.1	Color spaces . . . . .	12
2.3.2	Explicit thresholding . . . . .	13
2.3.3	Skin Probability Map . . . . .	14
2.3.4	Gaussian classifiers . . . . .	14
2.4	Skin detection by texture . . . . .	15
<b>3</b>	<b>Application development</b>	<b>16</b>
3.1	Proposed model . . . . .	16
3.1.1	Datasets . . . . .	16
3.1.2	Parameters . . . . .	16
3.1.3	Evaluation . . . . .	17
3.2	Implementation . . . . .	18
3.2.1	The image segmentation module . . . . .	18
3.2.2	The color detection module . . . . .	19
3.2.3	The texture analysis module . . . . .	21
3.2.4	The evaluator module . . . . .	21
3.2.5	User interface . . . . .	22

<b>4</b>	<b>User manual</b>	<b>24</b>
4.1	Introduction . . . . .	24
4.2	Detection view . . . . .	24
4.3	Evaluator view . . . . .	25
4.4	Color model training view . . . . .	26
4.5	Texture model training view . . . . .	27
4.6	Common configuration frames . . . . .	28
4.6.1	Segmentation . . . . .	28
4.6.2	Color detection . . . . .	28
4.6.3	Texture detection . . . . .	29
4.6.4	Resize options . . . . .	29
4.7	User Feedback . . . . .	29
<b>5</b>	<b>Conclusions</b>	<b>31</b>

# Chapter 1

## Introduction TODO

### 1.1 Introduction

Skin detection means identifying pixels and regions in images which correspond to human skin. Starting from these regions we can build models that detect people or certain body parts.

There are many challenges in creating a good skin detector. One of the most challenging aspects is handling illumination. While the human eye can easily distinguish the same color in varying degrees of brightness this is not a trivial task for computers. The second aspect we must consider are the variety of skin tones. A good model should be able to handle people of all races. Another thing to note is the fact that there are a lot of objects and materials that resemble skin such as leather, wood, sand etc. Consequently, most models have a rather large false positive rate (around 10%).

The purpose of this paper is to present the best models currently employed in skin detection and to propose a new model that combines some state of the art detectors. To the best of my knowledge no method that combines image segmentation, color and texture analysis has been published in this area of research. For every approach in skin detection I included a theoretical introduction followed by some examples of its practical use in skin detection.

### 1.2 Applications

One of the first applications in this field of study was scanning internet images in order to filter pornographic content [1]. The model proposed in that paper scans the images for large areas of skin and then applies some geometrical analysis looking for elongated shapes.

Another application was detecting anchorpersons in talk shows [2] for automatic annotation and storage. The data can be used for quickly finding videos containing specific persons or calculating how much time each anchorman was on stage. This is an ideal application for skin detection because news videos are usually shot indoors. Consequently, the environment can be set up so it won't contain objects with skin color tones.

Skin detection is usually an intermediary step for detecting more concrete body parts, so I will also present some applications of those methods. Face and hands detection is used to facilitate human computer interaction. By scanning gestures and reactions a software system can respond in a more convenient way to the user. For example the Kinect camera, which connects to the Xbox console, allows the user to navigate menus using hand swipes [3]. Face recognition, the next step after detection, is useful for automatic identification of people [4] which has applications in security, such as unlocking your phone only when it scans your face, and monitoring certain areas, like the systems used by the Chinese government to fine jaywalkers caught on camera without human intervention [5].

# Chapter 2

## Theoretical background

### 2.1 Related work

The subject of skin detection has been of interest for researchers due to its many applications. I could not include all the important papers in this domain due to the sheer number but I tried to cover the most relevant ones, in terms of popularity and results, for each aspect of skin detection.

#### 2.1.1 Models using color analysis

##### Skin Probability Maps

One of the most comprehensive experiments in skin detection was conducted by MJ Jones and JM Rehg[6] who created the Compaq skin dataset, which became a standard for evaluating results of research in this domain. Their main approach was building a Skin Probability Map based on Bayes' Theorem. The resulting classifier, published in 1998, has obtained results comparable to modern approaches.

Skin probability maps are histogram based models, meaning they set a number of bins for each color channel where they keep track of how many times each color pixel appears in the training set. Although  $256^3$  would be an obvious choice for the number of bins (one for each pixel value) they determined that  $32^3$  is the optimal value.

Jones and Rehg[6] made the observation that given even a large training set most pixels are never seen. They explored their dataset of approximatively 2 billion pixels and came to the conclusion that around 77% of the RGB space is empty. This comes in support of their decision to use larger bins, like  $32^3$ .

For this model they used a 6822 photo training set, where 4483 photos contained no skin and 2336 had portions of skin. The testing set had similar dimensions: 6818 total photos (4482 non-skin and 2336 skin). Using these methods they obtained a 80% true positive rate and 8.5% false positive rate or, with a different threshold, a 90% TP rate and 14.2% FP rate.

## Gaussian Models

This approach consists of modeling human skin distribution as a Gaussian probability density function or a combination of Gaussian functions.

The previous paper[6] by Jones and Rehg also proposes a combination of two mixture models for skin and non-skin classes. Each model is composed of 16 gaussians and was trained on approximately 74% of the histogram data, because only that data was available at the time. The results were similar to those of the previous model: 80%/9.5% and 90%/15.5%.

Lee and Yoo[7] presented a Single Gaussian in CbCr and a Gaussian Mixture in IQ, both trained and tested on the Compaq dataset with the results of 90%/33.3% and 90%/30%, respectively.

Yang and Ahuja[8] propose a finite Gaussian mixture model. They theorize that this offers an improvement over a single Gaussian function and justify this hypothesis using statistical tests on normality and number of components, namely the Hawkins' Test on Normality and Homoscedasticity and McLachlan's Bootstrap Test. The selected methods of determining the model's input parameters is the Expectation Maximization (EM) technique. This experiment was performed on the Michigan face database that contains around 9.5 million skin pixels. The pixels were transformed from RGB to CIE LUV color space in order to separate and then remove the illumination component. The next step was applying a segmentation algorithm on the image and the analyzing each pixel's color. Every region that had more than 70% of its pixels classified as skin was labeled as a skin region. However, they do not offer and performance metrics on their experiment.

## Explicit thresholding

Another popular approach is explicit thresholding. This domain consists of finding simple rules that apply to each pixel's color features and determine if it's part of the skin class. Some examples include [9], [10], [11], [12] for face detection systems in different color spaces with varying results.

A set of thresholds for YCrCb space where proposed by Chai and Ngan[10]. They set the ranges for Cb from 77 to 127 and for Cr from 133 to 173 and worked with the ECU database.

Dai and Nakano[11] created a model for YIQ, an orthogonal color space, which only used the I component (which stands for in-phase). The range they provided was [0, 50], however most of the images in their databases where of people with yellow skin.

An interesting approach, proposed by Gomez and Morales[13], is having a learner find these rules. They use RCA, a constructive induction algorithm, to build rules expressed with simple arithmetic operations in the rgb space. Their method achieves better results than the Bayesian SPM on their dataset, however it is computationally slower. RCA stands for Restricted Covering Algorithm which resembles a general covering algorithm with the restriction of trying to build a single rule for each class (in this case, a rule for skin detection). The strategy implemented for



RCA was finding attributes which cover either a large number of true positives or a few false positives. The starting attributes were  $r$ ,  $g$ ,  $b$  and the constant  $1/3$ , which would generate new attributes using the operators :  $+$ ,  $*$ ,  $-$  and squaring. One of their best and simplest generated models looks like this:

$$\begin{aligned} \frac{r}{g} &> 1.185 \quad \text{and} \\ \frac{r * b}{(r + g + b)^2} &> 0.107 \quad \text{and} \\ \frac{r * g}{(r + g + b)^2} &> 0.112 \end{aligned} \tag{2.1}$$

In comparison with the C4.5 decision tree algorithm, the RCA method obtained slightly worse results but with much simpler rules. They used a custom dataset containing images of more than 2000 people and obtained around 90% both in precision and recall.

Brand and Mason [14] applied the technique from [12], which uses the YI'Q' space with the following threshold  $14 < I' < 40$ , on the Compaq dataset and obtained 94.7% TP rate and 30.2% FP rate.

## Hidden Markov Models

Sigal et al.[20] implemented a method for real time skin detection in videos. Their model predicts the evolution of the skin color histogram using a second order Markov model, with an initial prediction from a Bayes classifier over the Compaq dataset. It uses the EM algorithm which consists of two steps: E(frame segmentation based on histogram) and M(histogram adaptation based on feedback from the current frame).

### 2.1.2 Models using segmentation

Frerk and Al-Hamadi[15] proposed a method that applies image segmentation combined with a Bayesian SPM. The first step is calculating the probability for each pixel in an image and creating  $P_I$ , the pixel probability image. The color and position of each pixel is used as input for a SLIC algorithm, an adaptation of k-means, that calculates the superpixels. The parameter  $k$  for segmentation is set to be the number of pixels in the image divided by  $30^2$ , so that all regions have a size of approximatively  $30 \times 30$ . A probability is then computed for each superpixel and compared to a threshold.

In [16] a similar approach to [15] is taken. Firstly, a segmentation is performed (using the Quick shift algorithm) and superpixels are extracted. A probability is calculated for each superpixel as the average of its component pixels probabilities. Then, a CRF(Conditional Random Field) method is used in order to obtain smoother skin regions. A CRF is a probabilistic method that works with graph based structures and models a distribution of type  $P(y|x)$ . The pair considered in this article is color difference and boundary length between neighboring superpixels.

This model was tested on the Compaq database and obtained a 91.17% TP rate and 13.12% FP rate.

### 2.1.3 Models using textural features

[17] presents a model based on Artificial Neural Networks that analyzes both color and textural features. The color features used are the mean color, the standard deviation and the skewness. These are calculated for each channel (R, G, B). The inputs for the ANN also include the following textural information: Entropy, Energy, Contrast and Homogeneity, which are computed from the Gray-Level Co-Occurrence Matrix. The networks has three layers: an input layer, a hidden layer (with 50 neurons) and an output layer. The model was trained on 300 images (80x80 px) of skin and non-skin textures and tested on 100, obtaining a 96% accuracy in classifying whether an image is a skin patch or not.

Medjram et al.[18] proposed another method that combines color and texture information. The first step is converting the initial image to the YCbCr color space. Skin regions are identified using thresholding like the following:  $77 \leq Cb \leq 127$ ,  $133 \leq Cr \leq 173$ . Secondly, the image is sharpened in order to enhance its texture. The features considered from the GLCM are Contrast, Homogeneity and Energy, calculated over a 5x5 matrix. The last step is applying a Support Vector Machine classifier on the initial image to classify texture patches.

In [19] they use Gabor wavelet transforms (filters that analyze orientation and frequency) to compute textural attributes. After identifying the initial skin regions a watershed segmentation algorithm is applied to increase true acceptance rate.

## 2.2 Image segmentation

Image segmentation is a technique for dividing an image into several regions that contain similar pixels. These partitions are often called super-pixels and represent an abstraction layer over the initial image. They can be characterized by color, border or shape(circle, ellipse, polygon, etc.)[21]. The main purpose of segmentation is to represent areas of interest in an image such as faces, fields, buildings, etc, and usually serves as a preparation step for a more complex detection algorithm.

Ideally, the resulting regions should have the following characteristics: uniformity according to the selected feature, such as color or texture, a small number of holes(subregions that differ considerably from the container region), a notable difference from the neighboring areas and smooth borders.[21].

This problem has been researched extensively over the years however there is not a single best solution available. I will present some of the algorithms presented in literature.

### 2.2.1 Thresholding

Thresholding is one of the simplest approaches to image segmentation. It consists of finding a threshold  $T$  for the gray level of pixels in the image. Therefore, we can classify every pixel by comparing its brightness to  $T$ . This technique is a perfect fit for separating objects from a darker background but has severe limitations in other tasks[22].

One of the problems with global thresholding, choosing a single value for  $T$  over the whole image, is dealing with different levels of illumination. For that reason the local thresholding approach was developed. This expands the previous method by using multiple thresholds for different parts of the image. Chow and Kaneko [23] applied a similar method for detecting the left ventricle of the heart in x-ray pictures. They divide the original image in multiple blocks that do not overlap and calculate a threshold for each of them.

Threshold selection can be done manually or automatically and usually involves analysis of a gray level histogram where the size of each bar is proportional to the number of pixels with that brightness[24]. For example, in an image that contains some objects in front of a darker background the histogram is likely to contain two peaks separated by a valley and we can choose  $T$  somewhere in between those peaks.

Automatic threshold detection can be done by modeling object and background populations with a normal distribution like the experiments presented in [25]. The method proposed calculates a least-squares fit of a function  $f(i)$ , where  $i$  is the gray level, to the histogram, using a hill climbing algorithm. The function takes into account the mean and standard deviation of the histogram. Lastly, the best fitting  $f(i)$  is the used for classification.

However simple, all these methods have limitations, such as not considering spatial factors, and having no control over border smoothness and holes inside detected regions.

### 2.2.2 Edge detection

These methods aim to solve image segmentation problems by detecting all edges in images. An edge is characterized by a sudden change in pixel intensity[22]. The result of edge detection is an image that represents to classes of pixels: part of an edge or not. I will present some of the most popular methods.

The Roberts Detection[26] is one of the fastest methods due to its simplicity. It uses the following convolution masks:

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad G_y = \begin{bmatrix} 1 & 1 \\ -1 & 0 \end{bmatrix}$$

Each of theses calculates a gradient value for an orientation and their values can be combined to determine an absolute magnitude of the gradient using:  $|G| = \sqrt{G_x^2 + G_y^2}$ . Lastly, we can apply thresholding for the resulting value to determine if it is part of an edge or not.

Other similar detectors are the Prewitt and the Sobel detectors[26]. The difference is that they use a larger convolutional mask, 3x3, and in case of the former, more orientations.

Some soft computing techniques such as: neural networks, genetic algorithms and fuzzy logic[26], have also been applied in edge detection problems. However they don't offer significant improvements in performance over the simpler and faster methods.

### 2.2.3 Clustering

We can view the image segmentation process as a clustering problem. All pixels are associated with a region, which corresponds to a cluster.

#### K-means

One popular approach to segmentation problems is the K-means algorithm, which can divide  $n$  data points in  $k$  clusters, where  $k$  is known in advance. This can be applied to image segmentation if we consider pixel intensities as the values on which we are doing clustering. The algorithm is simple and efficient on large volumes of data[22], however the number of clusters must be known in advance, which can be problematic as the number of distinct regions can vary from image to image.

#### Mode seeking. The Quick shift algorithm

Mode seeking algorithms are methods for clustering without needing to know the number of clusters in advance. The first step of these algorithms, as presented in [27], is calculating a Parzen density estimate over all given points:  $P(x) = \frac{1}{N} \cdot \sum_{i=1}^N \cdot k(x - x_i)$ . The next step is moving each point towards a mode of  $P(x)$  on a trajectory determined by the gradient of  $P$ . Clusters are formed by the points converging on the same modes.

The Quick shift algorithm[28] is one of the most efficient in this family. It starts by calculating a Parzen density estimate, most often using an isotropic Gaussian window:

$$P(x) = \frac{1}{2\pi\sigma^2N} \cdot \sum_{i=1}^N e^{\frac{-\|x-x_i\|^2}{2\sigma^2}} \quad (2.2)$$

Every point is then linked to the nearest one with a higher density and the resulting structure is a tree. In order to get our regions we can either cut branches with a size larger than  $\tau$  or analyzing data points less than  $\tau$  distance away on the linking step. If we choose the latter the result will be a collection of trees, each representing a cluster.

In the application of Quick shift in image segmentation a relevant feature space must be chosen. An option is using the RGB components and position( $x$ ,  $y$  coordinates) of each pixel. As the position does not always has fixed bounds (image size can vary) we can scale the position components depending on our input data, such that the importance of color and position remains similar[28].

An example pseudocode implementation as presented in [28] looks like 1. When calculating

# density computation;

**for**  $x$  in all pixels **do**

$P[x] = 0$ ;

**for**  $n$  in all pixels less than  $3 * \sigma$  away **do**

$P[x] += \exp(-(f[x] - f[n])^2 / (2 * \sigma * \sigma))$

**end**

**end**

# neighbor linking;

**for**  $x$  in all pixels **do**

**for**  $n$  in all pixels less than  $\tau$  away **do**

**if**  $P[n] > P[x]$  and distance( $x, n$ ) is smaller than to previous parent **then**

$d[x] = \text{distance}(x, n)$ ;

            parent[ $x$ ] =  $n$ ;

**end**

**end**

**end**

**Algorithm 1:** The Quick shift segmentation algorithm

the density we limit our search to a  $3\sigma$  window because the contributions for pixels further away should be small[28].

Some advantages of Quick shift are simplicity of implementation, speed ( $O(N^2)$ ), ability to work on any type of data and the control of fragmentation with the given parameters[27].

## 2.3 Skin detection by color

Skin pixel detection by color means classifying a pixel while considering only its color features. A first step in applying this approach is selecting a color space.

### 2.3.1 Color spaces

A color space, also called a gamut, represents a set of colors in a way that is independent of the medium in which they are represented (computer screens, cameras, magazines, etc) [29]. The  $L^*a^*b^*$  color space contains all colors that can be seen by the human eye, however most color spaces are smaller due to technical limitations. I will present some of the color spaces which have been used successfully to classify skin pixels.

#### RGB

To start with, RGB is one of the most popular color spaces for working with image data. It matches the color sensitive receptors of the human eye (red, green, blue) and started as a convenient way to represent the colored rays used by CRT screens [30]. While this model is simple to use it has the disadvantage of mixing chrominance and luminance features [30].

#### Normalized RGB

Normalized RGB is a color space with a lighter memory consumption than RGB and its components are calculated as follows [30]:

$$r = \frac{R}{R + G + B}, g = \frac{G}{R + G + B}, b = \frac{B}{R + G + B}. \quad (2.3)$$

The third value can be determined from the other 2 so we can avoid storing it. Other advantages according to [31] include reduced differences caused by illumination and ethnicity, and lower variance of skin color clusters than in the normal RGB space.

#### HSI, HSV, HSL

HSI, HSV, HSL represent perceptual color spaces and they describe the hue, saturation and intensity (or value, lightness). These color spaces are used because they provide invariance to ambient lighting and surface orientation relative to the source of light [31]. We can convert to HSV from RGB using the following formulas [30]:

$$H = \arccos \frac{\frac{1}{2}((R - G) + (R - B))}{\sqrt{((R - G)^2 + (R - B)(G - B))}} \quad (2.4)$$

$$S = 1 - 3 \frac{\min(R, G, B)}{R + G + B} \quad (2.5)$$

$$V = \frac{1}{3}(R + G + B) \quad (2.6)$$

## YCbCr

Orthogonal color spaces, which YCbCr is a member of, provide chrominance and luminance separation as they represent colors with statistically independent components. YCbCr is mostly used by European television studios and in image compression[30]. Y represents luma (or luminance) and Cb, Cr are the blue and red difference chroma components and they can be computed as follows:

$$Y = 0.299R + 0.587G + 0.114Bs \quad (2.7)$$

$$Cb = B - Y \quad (2.8)$$

$$Cr = R - Y \quad (2.9)$$

Having such a simple transformation and a clear separation of the luminance component makes the YCbCr a popular choice for skin detection models.

### 2.3.2 Explicit thresholding

This is one of the simplest skin-color models that can be built. The method aims to define, through the use of simple rules and thresholds, the boundaries of skin clusters in a specific color space. It has been observed in [32] that the colors of human skin tend to cluster in small regions of the color space and human skin pixels differ more in intensity than in color.

An example using the RGB space, from Peer et al.[9] which has been integrated into a face detection system consists of the rules below:

$$\begin{aligned} R > 95 \quad \text{and} \quad G > 40 \quad \text{and} \quad B > 20 \quad \text{and} \\ \max\{R, G, B\} - \min\{R, G, B\} > 15 \quad \text{and} \\ |R - G| > 15 \quad \text{and} \\ R > G \quad \text{and} \quad R > B \end{aligned} \quad (2.10)$$

These thresholds and rules can also be generated using an artificial intelligence algorithm. For example, we could apply genetic programming in a fashion similar to that described in Segaran's book [33]. Rules can be represented as trees whose leafs are color features or constants and inner nodes are simple operators.

The main advantage of using these methods is their speed. For each pixel we apply only a few operations, with a computational time of  $O(1)$ . For this reason, thresholding techniques tend to be used in real time systems, where fast feedback is more important than overall precision.

### 2.3.3 Skin Probability Map

A SPM represents a histogram with multiple bins. Each bin stands for a color or a subset of colors and has a value equal to the probability of holding skin colored pixels. When building a SPM you must choose the color space and the number of bins per color channel.

In order to determine whether a given pixel is a skin colored pixel we apply Bayes' theorem. Here is the form used by Jones and Rehg[6] in one of the most popular papers on statistical skin detection:

$$P(skin|p) = \frac{P(p|skin)P(skin)}{P(p|skin)P(skin) + P(p|\neg skin)P(\neg skin)} \quad (2.11)$$

In this equation "p" is the notation for the occurrence of the given pixel. Therefore  $P(skin|p)$  means the probability of observing skin given our pixel. To determine whether we classify the pixel as skin we compare our probability with the threshold value,  $\beta$ .

$$P(skin|p) > \beta \quad (2.12)$$

The probability of observing skin can be computed as the ratio of the number of skin pixels to the total number of pixels observed in training.

$$P(skin) = \frac{T_S}{T_S + T_N} \quad (2.13)$$

It has been observed that even when using a large data set most pixel values are never seen[6]. This suggests that we might get better results by reducing the number of bins per channel. We can further observe that a small perturbation in the RGB values of a pixel results in a very similar color. Consequently if we classify p as a skin colored pixel then there is a high probability that its neighbors in the color space are also skin colored pixels. This observation is in support of a smaller number of bins for the SPM histogram.

### 2.3.4 Gaussian classifiers

Gaussian classifiers are parametric skin distribution models with the advantages of being compact, therefore using less memory than SPMs, and able to generalize better using less training data[31].

#### Single Gaussian

Considering the observations from the thresholding chapter that skin color pixels tend to cluster in a region of the color space we can model that distribution using an elliptical Gaussian joint probability density function, an example of which was provided by [31]:

$$p(c) = \frac{1}{2\pi^{\frac{1}{2}}|\Sigma|^{\frac{1}{2}}} \cdot e^{-\frac{1}{2}(c-\mu)^T\Sigma^{-1}(c-\mu)} \quad (2.14)$$



In this equation,  $c$  is the color vector,  $\mu$  the mean vector and  $\Sigma$  the covariance matrix. These can be calculated from the training data as follows:

$$\begin{aligned}\mu &= \frac{1}{n} \cdot \sum_{j=1}^n c_j \\ \Sigma &= \frac{1}{n-1} \cdot \sum_{j=1}^n (c_j - \mu)(c_j - \mu)^T\end{aligned}\tag{2.15}$$

Here  $c_j$  are all the color samples used in training the model. In order to establish that the given color describes skin we can compare  $p(c)$  with a threshold that can be determined experimentally for a given training set.

### Gaussian Mixture Models

A Gaussian mixture model is a form of unsupervised learning used to identify subpopulations within an overall population, provided they are normally distributed[34].

It has been observed in [8] that a mixture of Gaussians is better suited for skin detection than a single distribution, especially in datasets with multiple illumination conditions.

They represent a generalization of the single Gaussian. A mixture's density function can be calculated as the sum of individual Gaussians[31]:

$$p(c) = \sum_{i=1}^N w_i \cdot \frac{1}{(2\pi)^{1/2} \cdot |\Sigma_i|^{1/2}} \cdot e^{-\frac{1}{2} \cdot (c - \mu_i)^T \Sigma_i^{-1} (c - \mu_i)}\tag{2.16}$$

In equation 2.16  $c$ ,  $\mu_i$  and  $\Sigma_i$  are the color vector, mean vector and covariance matrix for the  $i$ th Gaussian. Also, each of the  $N$  models has a weight,  $w_i$ , representing its contribution to the mixture.

Determining the unknown parameters can be done with the Expectation Maximization (EM) technique[8]. This is an algorithm of maximum likelihood estimation, or simply speaking finding the parameters that best describe some given data.

The number of Gaussians,  $N$ , is also an important aspect. The most common values used in research fall into the 2 to 16 range[31], the idea behind choosing a larger number is to account for various conditions of illumination.

## 2.4 Skin detection by texture

# Chapter 3

## Application development

### 3.1 Proposed model

The model proposed by this paper is made up of three steps. The first step is image segmentation using the Quick shift algorithm. A Bayesian probability is then calculated for every pixel in the region and the average of these is compared to a threshold to determine if the superpixel's color is likely to be that of skin. Finally, a Support Vector Machine analyzes patches around each pixel and calculates Haralick features in order to classify the texture. The results from color and texture detection are then combined into the final result.

#### 3.1.1 Datasets

For this experiment I have used two databases. First, the Compaq dataset [6], which is one of the most cited in literature, contains 13.000 images out of which 4700 contain skin and was used for the skin color model. The images were downloaded using a web crawler then divided into skin and non-skin images. For every skin image a black and white mask was created by hand to mark the regions of interest.

The second dataset, SFA [35], was created as a combination of FERET(876 images) and AR(242 images) datasets. It contains skin and non-skin patches of dimension from 1x1 to 35x35, which makes it ideal for the texture model. For each dimension, SFA has 3354 skin images and 5590 non-skin images.

#### 3.1.2 Parameters

For image segmentation I tried to find values for the sigma and tau parameters that produce regions as large as possible without distorting the contours. I chose the (3, 5) pair, which can be seen in comparison with other pairs in figure 3.1.

The most suitable area for texture analysis was also determined experimentally. As can be observed in figure 3.2 the 5x5 window offers a good detection of skin areas while the true

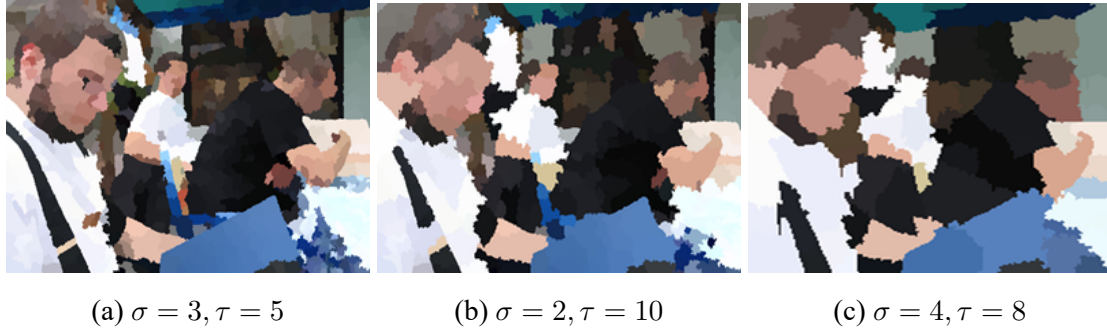


Figure 3.1: Segmenting an image with different parameters

positive rate goes down drastically for larger ones.

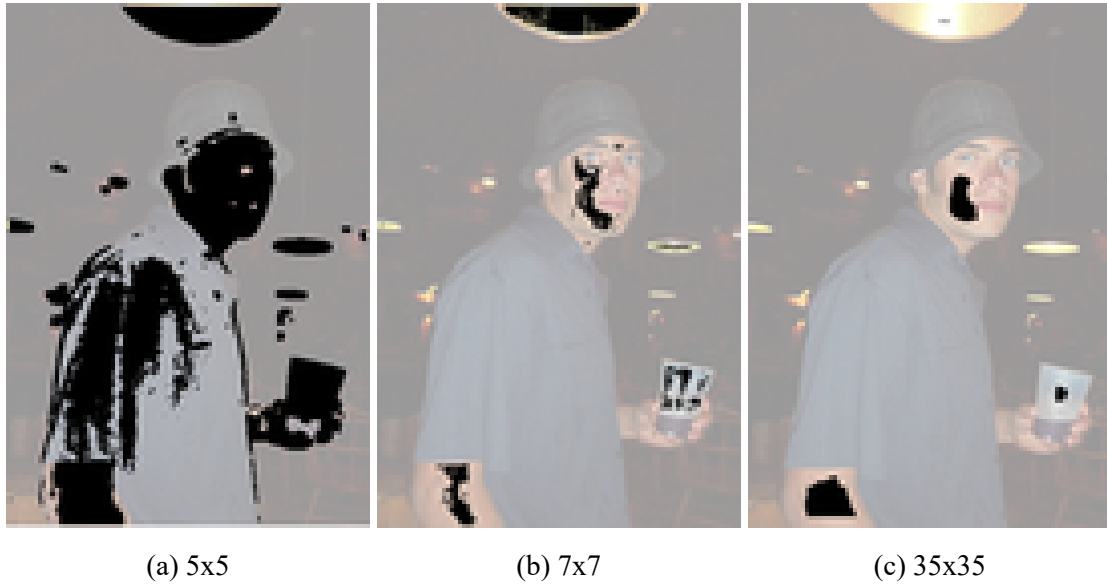


Figure 3.2: Texture detection with different windows

I tried building color models in RGB, HSV and YCrCb and concluded that Bayesian color analysis obtains roughly the same results regardless of the color space chosen. For every pixel, I chose the maximum probability on a distance of 4 (similar to the bin size used in [6]) on every feature. The threshold value can be used to control the TPR/FPR ratio.

### 3.1.3 Evaluation

The model was evaluated on the Compaq skin database in order to compare it with other approaches from literature. There is a difference however, in the number of images chosen for evaluation. While most references use a 6000 image set for testing I used only 50 images due to time constraints.

For each image in the testing set I calculated the True Positive Rate and the False Positive Rate by comparing each pixel from my output to the given mask and then I averaged the results.

$$TPR = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN} \quad (3.1)$$

Equation 3.1 shows how those metrics were calculated. TP, FP, TN and FN stand for the numbers of true positives, false positives, true negatives and false negatives, respectively.

Table 3.1: Evaluation of the proposed model on the Compaq dataset

TPR	FPR	Threshold
86.57%	28.48%	0.1
76.29%	15.54%	0.25
81.59%	21.29%	0.167

## 3.2 Implementation

The application is made up out of five main modules, pictured in figure 3.3: segmentation, *color\_analysis*, *texture\_analysis*, evaluator and *application\_gui*. The first three provide the basic image processing operations, the evaluator provides an interface for detection and performance measurements while the top module offers a visual interface for user convenience. I will present them in detail in separate sections.

Some design decisions that influenced all modules are representing images as numpy arrays (a python library for array processing) and using named tuples as keys for all maps.

Named tuples are a feature implemented in the 'namedtuple' library and are an efficient way of representing keys. The two tuples used in this application are pixel position (x,y components) and color features.

Most of the trivial image operations such as reading, writing and converting color space have been done using the 'opencv' library.

All progress messages generated by the application are handled to a logger. This can be injected at startup to display images on console, file or pass them to another process to be displayed on a GUI.

### 3.2.1 The image segmentation module

An overview of this module can be seen in figure 3.4. The segmentation functionality is provided by the QuickshiftSegmentation class, which exposes two public methods, *apply(image)* and *get\_superpixels(image)*, and takes the sigma and tau parameters in the constructor. The first method applies Quick shift segmentation on the image and returns an image where regions are visible, being colored with the same value as the root pixel. The second method returns a map

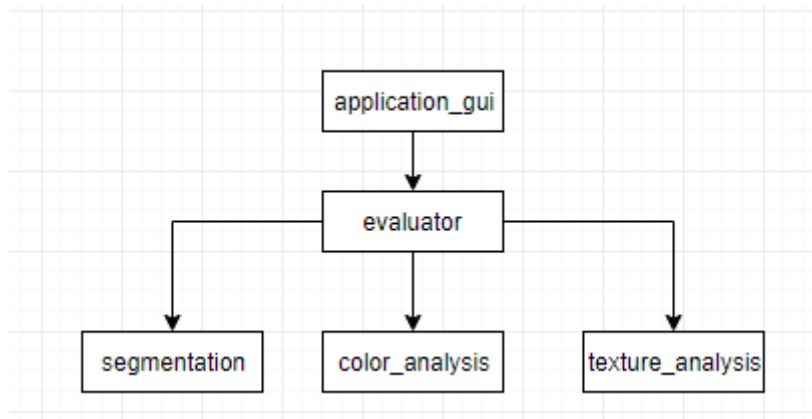


Figure 3.3: Main modules

containing all identified regions. The keys are roots pixels and the values their corresponding superpixels.

Regions are represented by the Superpixel class whose fields are: the initial image, position of the root pixel and a list of positions of all corresponding pixels.

The Distances class encapsulates some methods used by the segmentation algorithm to calculate pixel and feature distances.

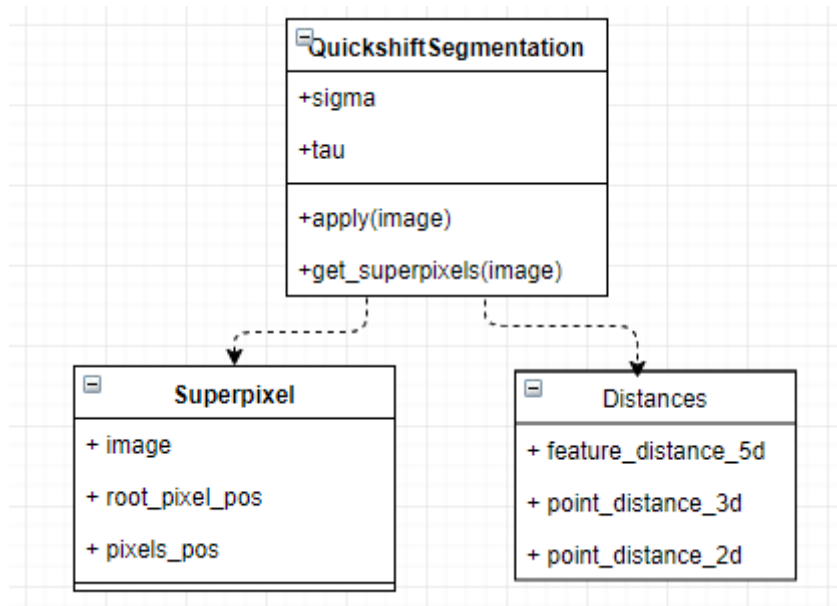


Figure 3.4: Image segmentation module

### 3.2.2 The color detection module

This module can be divided into two submodules: training, figure 3.5, and detection, figure 3.6.

In order to train a color detection model you must call the *train<sub>spm\_model</sub>* static method which takes a SPMTrainConfiguration object. The configurable fields are the database (SFA or

Compaq), the color space (RGB, HSV, YCrCb) and paths for resources.

Depending on the value of the database parameter the trainer creates either a `SfaComponentExtractor` or a `CompaqComponentExtractor`. Both of these classes provide a method `extract_components()` that returns a `BayesSPMComponents` object, which encapsulates: the number of skin and non-skin pixels, a map with the number of apparitions for each pixel and a map with the number of apparitions as skin.

The resulting model, a `SPMModel` object, is serialized into a file. It contains the selected color space and the components extracted on the previous step.

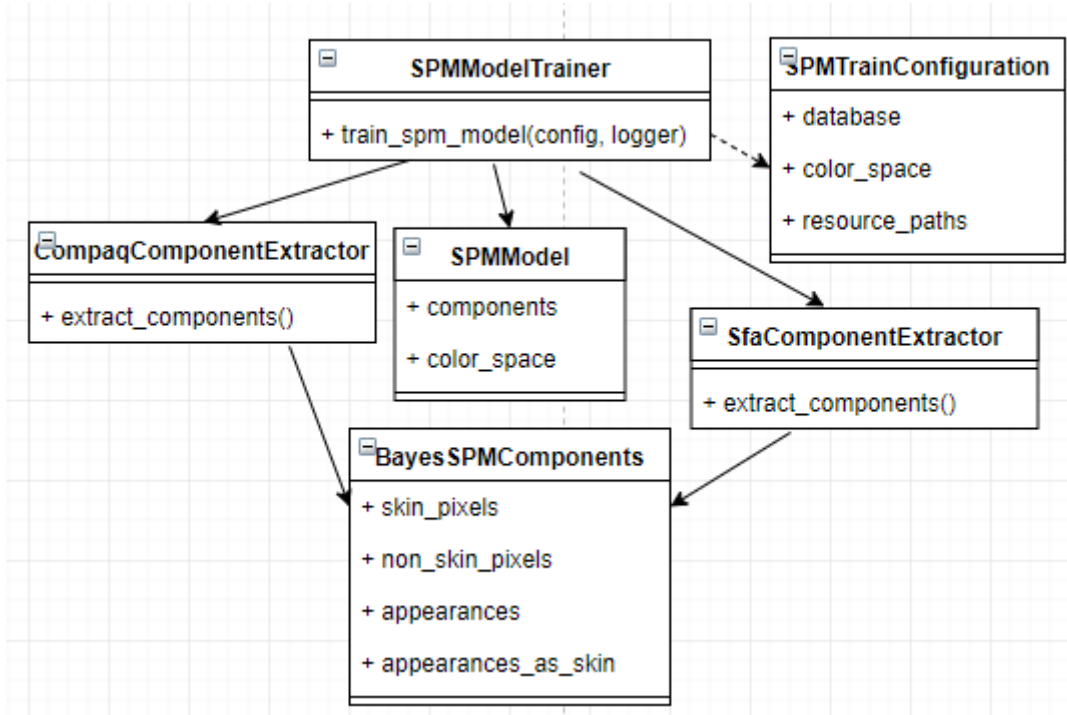


Figure 3.5: Color detector training submodule

Depending on the input parameters the `SPMDetectorFactory` provides one of the following detectors: `SimpleDetector`, `NeighborDetector` or `AverageOnSuperpixelDetector`. The first one is the simplest approach as it calculates a probability for each pixel independently. The second, takes into account an area around each pixel. This simulates the histogram based approach with the difference that the considered pixel is always at the center of the area. The last detector calculates the average probability of a given superpixel.

All detectors use the methods defined in `ProbabilityCalculator` through an intermediary layer. This proxy, `CachedProbabilityCalculator`, was created to speedup operations by caching probabilities. The need for memorizing these values arose from the fact that images usually contain a small subset of pixel values duplicated many times.

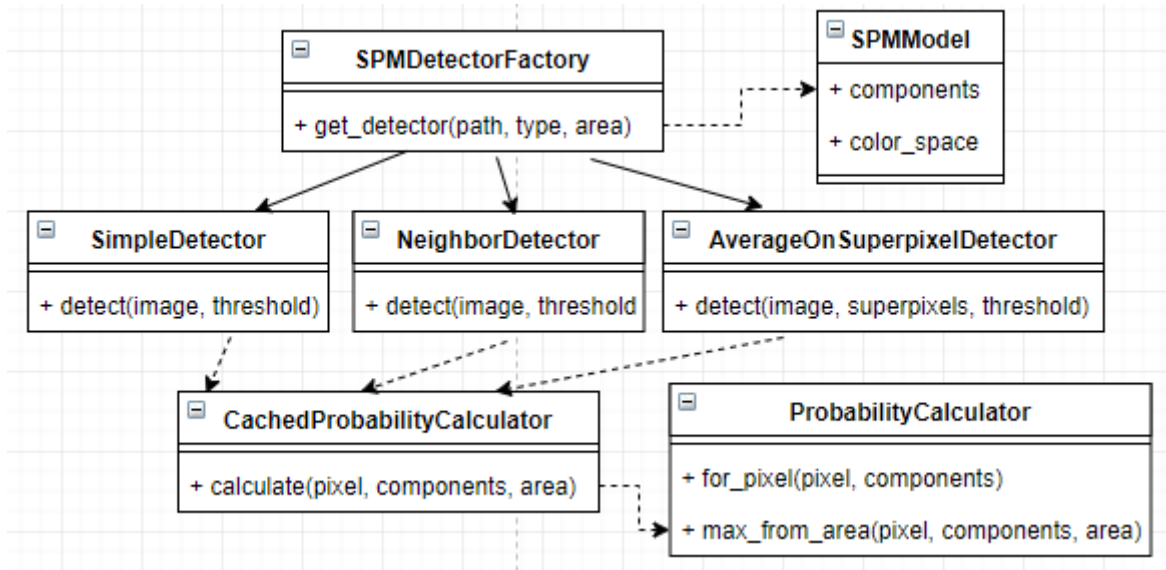


Figure 3.6: Color detection submodule

### 3.2.3 The texture analysis module

Similarly to the previous module, texture analysis can be divided into two submodules, one for training, figure 3.7, and one for detection, figure 3.8.

The model used in skin texture detection, TextureModel, has an attribute for extracting textural features from images (feature\_extractor), one for classifying those values (classifier), and label for interpreting the classifier's results (skin\_label). This model is trained by a Haralick-ModelTrainer which its parameters from a TextureTrainConfig object.

In this project I only implemented a model based on Haralick's features but the design allows incorporating other methods. The HaralickModel extends the basic TextureModel and overrides the feature extraction method. It is trained by a HaralickModelTrainer which injects a HaralickFeatureExtractor, which encapsulates the feature extraction method taken from the mahotas library, and SvmClassifier, which trains a Support Vector Machine classifier using the sk\_learn library. The model trainer is also responsible for selecting a method of preparing the input data, in this case using the TrainDataPreparer class, which computes Haralick's features and appends the corresponding label.

There are two types of detectors available. The GridDetector slides a window over the image and classifies each of those blocks. The PerPixelDetector is a more precise approach because it builds a window around each pixel in the image and only marks the center pixel. Although the second approach gains a noticeable increase in precision, it is much slower than the first.

### 3.2.4 The evaluator module

The evaluator module has a simple structure, as seen in figure 3.9. The Evaluator class contains methods for evaluating a model(run\_simulation) and for applying it to an image(run\_detection).

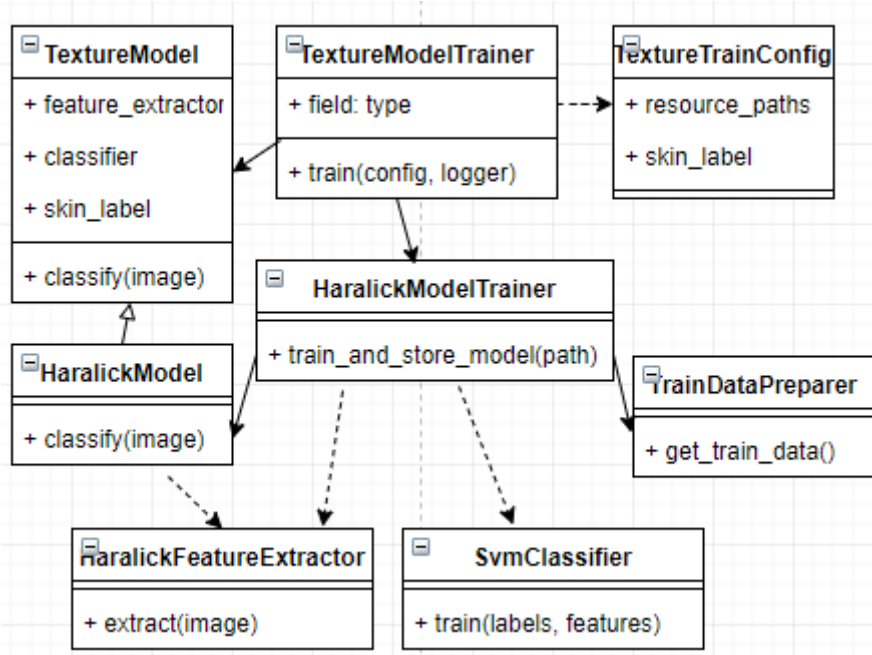


Figure 3.7: Texture detector trainer submodule

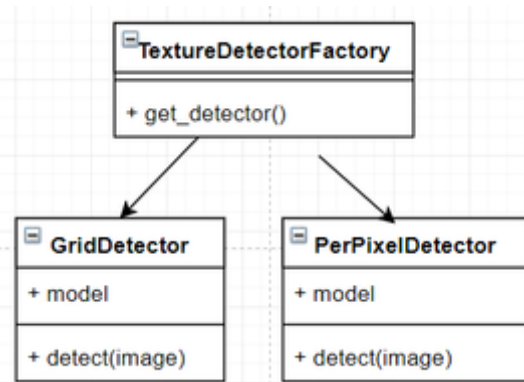


Figure 3.8: Texture analysis submodule

It takes as input a RunConfiguration object which contains informations about the desired skin and texture models and segmentation parameters.

In order to evaluate the model some quality measures are applied for each analyzed picture. These methods include the false positive rate, true positive rate, precision, accuracy and recall and are implemented by the Stats class.

### 3.2.5 User interface

The user interface was build using the tkinter library. This framework was designed to create modular applications. Consequently every window can be composed out of multiple frames, which are reusable components. The main views used in the application are visible in figure 3.10. The various configuration frames were left out of the diagram to avoid clutter.



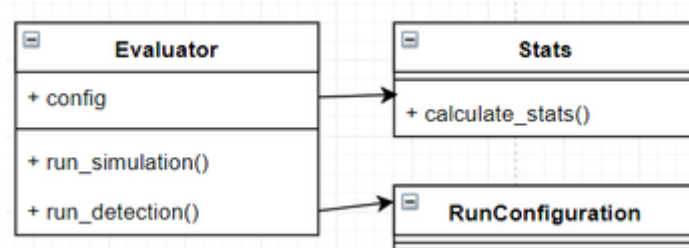


Figure 3.9: The evaluator module

MainView is a tab-view component which provides links to the application's main windows: color model training, texture model training, model evaluation, detection of a picture.

Every component has a ProcessControlFrame which contains a FeedbackFrame and some buttons for starting and stopping operations. When the user presses the 'Start' button a new process is launched which performs the request and logs results into a queue. A monitoring thread reads that from the queue and passes it to the FeedbackFrame for displaying it on the screen. This approach allows the user to follow the application's progress while keeping the user interface decoupled from the algorithms.

The detection view also contains an ImageDisplayFrame, which displays the input and the resulting image.

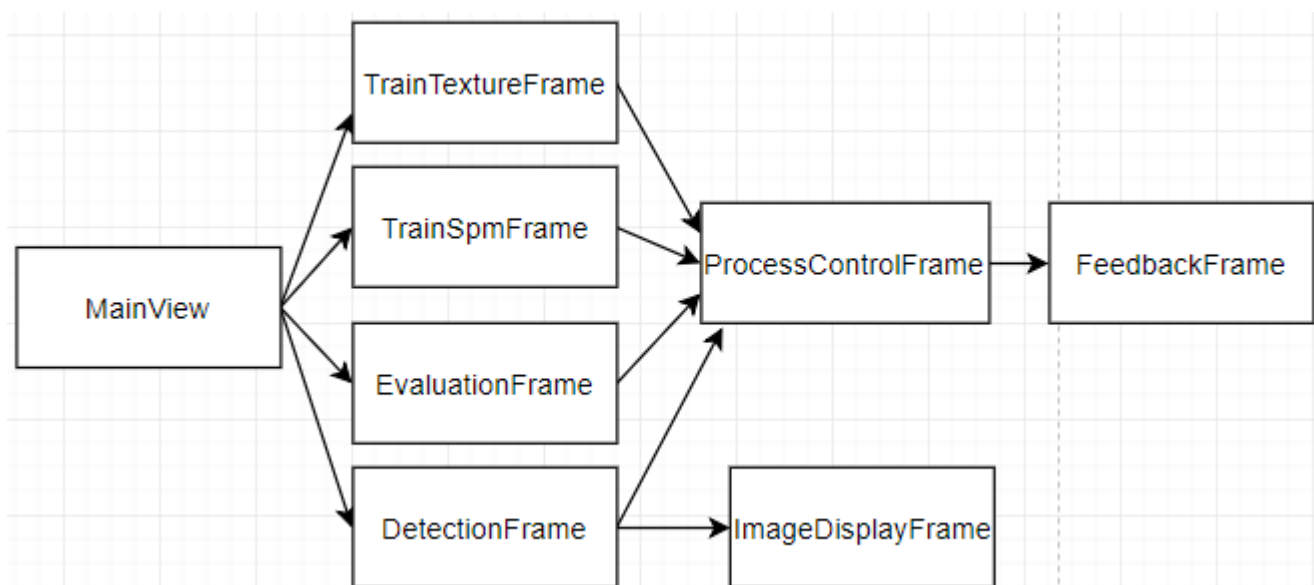


Figure 3.10: The user interface module

# Chapter 4

## User manual

### 4.1 Introduction

The goal of this application is to provide an easy-to-use and highly-configurable interface for training and evaluating skin detection models that share a core structure of image segmentation, color and texture analysis. The user can also test the created models on individual images. As the number of parameters for each task can be overwhelming the user can opt to use the default values that have already been set in each field.

The application has 4 main windows accessible through a tab-view set in the top-left corner. The windows, detailed in the following sections, are the Evaluator, Train SPM, Train texture and Detection windows.

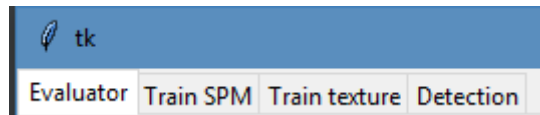


Figure 4.1: The tab-view main menu

### 4.2 Detection view

This view allows the user to test the model on individual images and get visual feedback. The resulting image will be displayed on the screen with a black mask that marks the identified pixels. This window provides a quick way to evaluate a model or to make a demonstration.

If you want to apply the model to a single image you must specify the detection parameters, or go with the default ones, provide a path to the input image (bottom frame on the left side of the screen) and click ‘Start experiment’. The initial image and the result will be displayed on the frames on the right side of the screen. The other configuration frames are explained in detail in section 4.6.

On the middle of the screen, below the buttons for controlling the experiment, you will see updates of the detection process. A progress bar will follow the time consuming operations: segmentation, color and texture analysis.

Figure 4.2: Detection view

### 4.3 Evaluator view

This screen allows the user to evaluate the performance of the final model and follow the progress in real time. The selected model can be configured in various ways which will be presented in detail in the common configurations section 4.6.

The application applies the selected model to images from the provided folders. Input images can have any size and name. Masks should have the same size and name as their corresponding input and should mark skin pixels in black (0, 0, 0). The paths where execution logs and run results are stored are also customizable.

To begin evaluation you must press ‘Start experiment’. The frame below the button displays information about the currently processed picture, the operation in progress and evaluation results.

A progress bar will indicate the status of each action applied to the image. The main actions are, in order: image segmentation (consisting of density computation, neighbor linking and superpixel extraction), color detection (Bayes SPM), texture analysis (Haralick feature, combination of results and metrics calculation). Images obtained for each step (color, texture and their combination) are all saved in the results folder, to facilitate the identification of cases for which the model does not perform as expected.

Evaluator	Train SPM	Train texture	Detection
			Sigma <input type="text" value="3"/>
Segmentation			Tau <input type="text" value="5"/> <input type="checkbox"/> Use position
			Browse model <input type="text" value="spm_compaq_2000_with_ns_4000_rgb.pkl"/>
Color detection			Threshold <input type="text" value="0.1"/> Type <input type="text"/> Neighbour area <input type="text" value="4"/>
			Browse model <input type="text" value="svm_classifier_1000data_5area.pkl"/>
Texture detection			Detection type <input type="text"/> Detection area <input type="text" value="5"/>
			<input type="checkbox"/> Resize image
Image size			Width <input type="text" value="200"/> Height <input type="text" value="200"/>
Resources			<input type="button" value="Browse..."/> input <input type="button" value="Browse..."/> mask <input type="button" value="Browse..."/> run_results <input type="button" value="Browse..."/> logs.txt

Figure 4.3: Evaluator view

After each picture is analyzed by the model, the performance metrics(true positive rate, false positive rate, true negative rate, false negative rate, accuracy, precision and recall) are displayed on the screen and saved to a file. The application builds a csv file containing metrics from each processed picture, that can be used to obtain overall results. A dump of the initial configuration is done in the file *initial\_config.txt* for documentation purposes.

## 4.4 Color model training view

This window is used to train a Bayesian color detection model. You can select a color space, the path to input data (a folder which should contain the following subfolders: train\_images – images containing skin, train\_images\_ns – images not containing skin, train\_masks – masks for images containing skin), the name of the model and the path to the folder where it will be stored.

The model, consisting of a map of pixel values and their count, is stored alongside color space meta data in a .pkl file, which can be later imported by the application for evaluation.

Similarly to the other views, to start training the model you must press the ‘Start experiment’ button. In the frame below you will see status updates for loading images and extracting features from them as well as a progress bar for each operation.

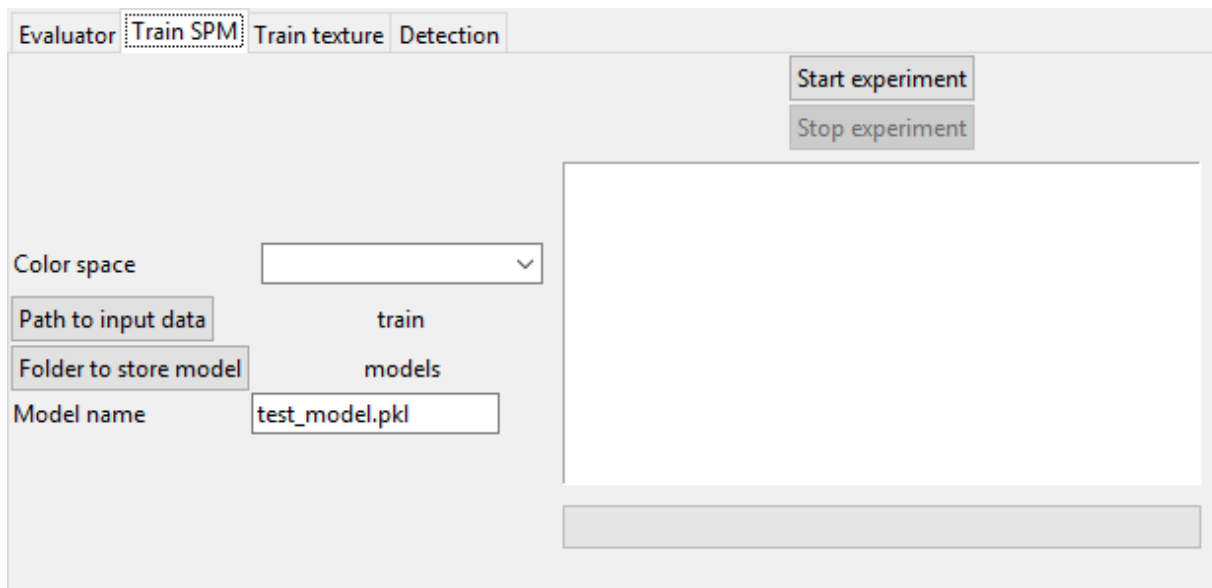


Figure 4.4: Color model training view

## 4.5 Texture model training view

The process of training a texture detector is simple. You must select folders with images of skin, images not containing skin and a path to store the resulting model.

After pressing ‘Start experiment’ the training begins and you will see updates for feature extraction and machine learning analysis. First, the process builds a list of features and tags (skin/non-skin) from positive and negative images. Then, those features are analyzed by a SVM algorithm and the resulting model is stored in a .pkl file. That file contains the SVM’s result, the serialized function for feature extraction and the tag used to identify skin texture.

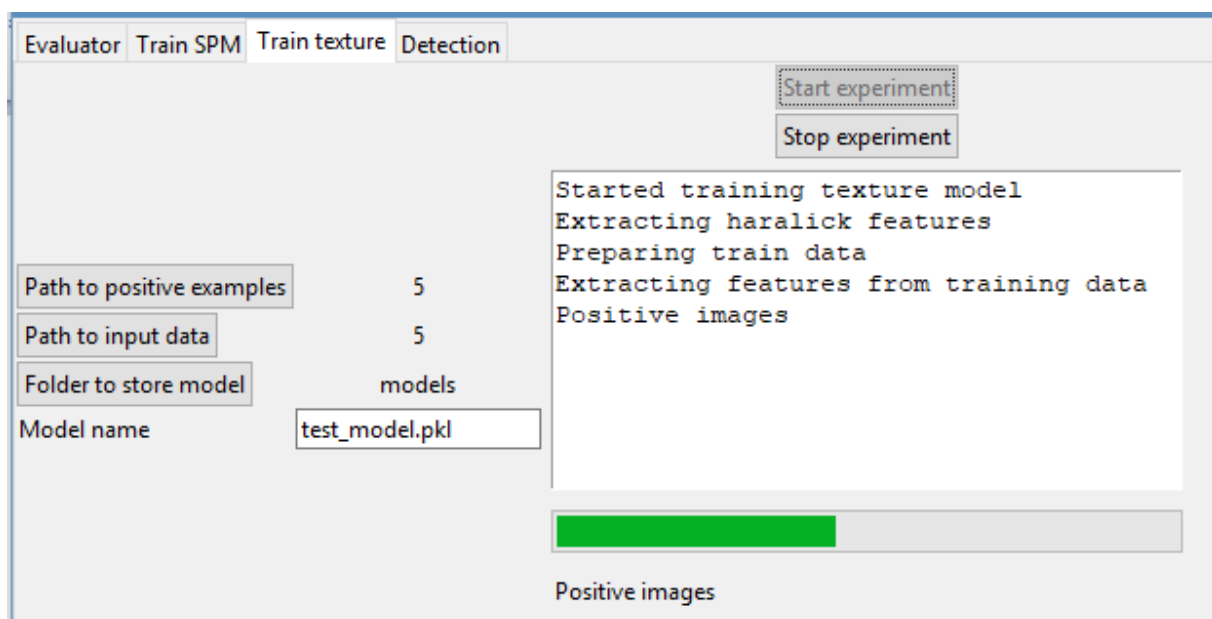


Figure 4.5: Texture model training view

## 4.6 Common configuration frames

Some or all of these model configuration frames are present on multiple views. For this reason they are presented in a single section.

### 4.6.1 Segmentation

The segmentation frame is used for tuning the parameters of the quickshift image segmentation algorithm.

Sigma controls the shape of the superpixels and Tau the size, and you can set them by typing in their adjacent textboxes.

The 'Use position' checkbox indicates whether the algorithm should consider pixel position in image as a feature for density calculation. If this box remains unchecked the density function will consider only the color space information.

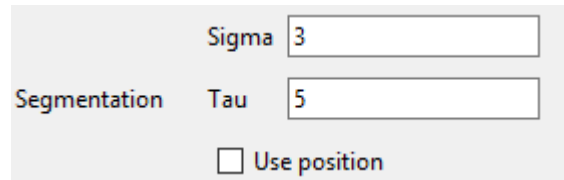


Figure 4.6: Segmentation

### 4.6.2 Color detection

This frame provides configurations for the color detection module.

The SPM model is a pickle (.pkl) file generated from the Train SPM view. This indicates which color model should be selected for Bayesian probability calculations.

The detection threshold is a value between 0 and 1 which indicates at what probability to start considering pixels as skin. Consequently all pixels or clusters for which the model calculates a probability higher than the threshold are classified as skin pixels.

The type of the SPM classifier can be selected from the 'Type' dropdown widget. There are two types: average probability over superpixel and probability for the root of the superpixel. This affects how the probabilities are calculated for the regions identified during segmentation.

The neighbor area represents the considered neighbors during the computation of a pixel's skin probability. The final probability is taken as the maximum value obtained by exploring the color space in every direction on a distance limited by this parameter.

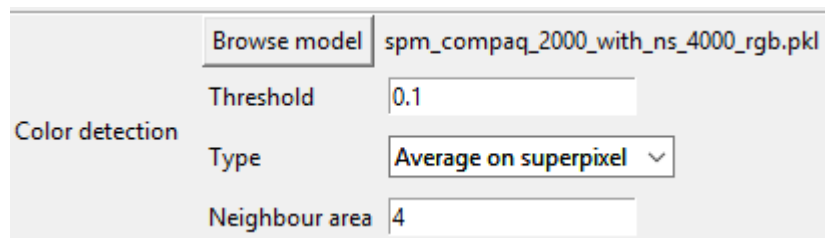


Figure 4.7: Segmentation

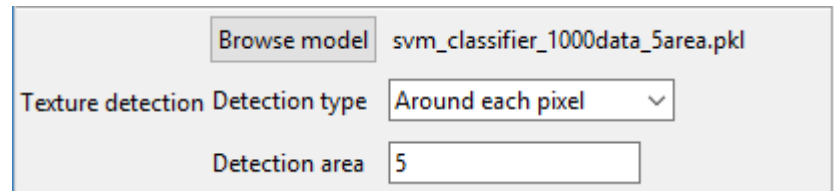
### 4.6.3 Texture detection

By clicking on the 'Browse model' button you can select the texture analysis model, which is a SVM classifier stored as a pickle file. This model is generated from the 'Train texture' window.

From the 'Detection type' dropdown you can select the texture detection strategy:

- Window scan, which slides a window over the image, is fast but produces jagged edges
- Around each pixel, which builds a window around each pixel, is slow but obtains smoother results

The size of these windows is set from the 'Detection area' text field.



The screenshot shows a user interface for texture detection. It includes a 'Browse model' button, a text field displaying the file path 'svm\_classifier\_1000data\_5area.pkl', a 'Detection type' dropdown menu currently set to 'Around each pixel', and a 'Detection area' text field with the value '5'.

Figure 4.8: Segmentation

### 4.6.4 Resize options

If you wish to resize the image you must check the 'Resize image' box and then specify the given parameters. This is usually done for increasing computation speed.

If you do not check the box then the image will be processed in its original size.



The screenshot shows a user interface for image resizing. It features a 'Resize image' checkbox, which is currently unchecked. Below it are two text fields: 'Width' and 'Height', both of which are set to the value '200'.

Figure 4.9: Segmentation

## 4.7 User Feedback

The feedback view can be seen in figure 4.10 and is present on each window. It is a text box that displays feedback from the application. It also contains the 'Start experiment' and 'Stop experiment' button which control the process associated with the view: detection, evaluation or training.

On the bottom of the frame there is a progress bar that keeps track of the current operation, for example: density calculation, superpixel extraction, SPM detection etc. That operation is also displayed in the label underneath the progress bar.

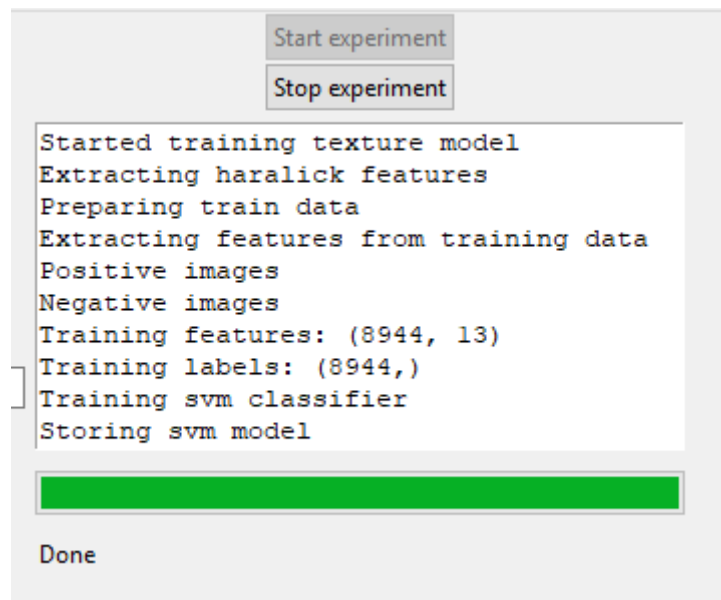


Figure 4.10: Feedback view



## **Chapter 5**

### **Conclusions**

# Bibliography

- [1] M. M. Fleck, D. A. Forsyth, and C. Bregler, “Finding naked people,” in *Computer Vision — ECCV ’96* (B. Buxton and R. Cipolla, eds.), (Berlin, Heidelberg), pp. 593–602, Springer Berlin Heidelberg, 1996.
- [2] B. M. and R. Amsaveni, “Anchor person detection using haar-like feature extraction from news videos,” *International Journal of Computer Applications*, vol. 153, no. 9, pp. 23–27, 2016.
- [3] “The complete list of kinect gesture and voice commands for your referencing pleasure.” <https://news.xbox.com/en-us/2013/11/26/xbox-one-kinect-gesture-and-voice-guide/>. Accessed: 2018-05-03.
- [4] M.-H. Yang, D. J. Kriegman, and N. Ahuja, “Detecting faces in images: A survey,” *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 24, no. 1, pp. 34–58, 2002.
- [5] C. Geib, “If you jaywalk in china, facial recognition means you’ll walk away with a fine.” <https://futurism.com/facial-recognition-china-social-credit/>. Accessed: 2018-05-03.
- [6] M. J. Jones and J. M. Rehg, “Statistical color models with applications to skin detection,” *International Journal of Computer Vision*, vol. 46, pp. 81—96, 2002.
- [7] J. Y. Lee and S. I. Yoo, “An elliptical boundary model for skin color detection,” in *In Proc. of the 2002 International Conference on Imaging Science, Systems, and Technology*, 2002.
- [8] M. hsuan Yang and N. Ahuja, “Gaussian mixture model for human skin color and its applications in image and video databases,” in *Proceedings of SPIE ’99 (San Jose CA)*, pp. 458–466, 1999.
- [9] J. Kovac, P. Peer, and F. Solina, “Human skin colour clustering for face detection,” *EU-ROCON*, 2003.
- [10] D. Chai and K. N. Ngan, “Face segmentation using skin-color map in videophone applications,” *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, vol. 9, no. 4, pp. 551—564, 1999.

- [11] Y. Dai and Y. Nakano, "Face-texture model based on sgld and its application in face detection in a color scene," *Pattern Recognition*, vol. 29, no. 6, pp. 1007—1017, 1996.
- [12] C.Wang and M.Brandstein, "Multi-source face tracking with audio and visual data," *IEEE MMSP*, p. 168, 1999.
- [13] G. Gomez and E. F. Morales, "Automatic feature construction and a simple rule induction algorithm for skin detection," *Proceedings of Workshop on Machine Learning in Computer Vision*, pp. 31—38, 2002.
- [14] Brand, J., and Mason, "A comparative assessment of three approaches to pixellevel human skin-detection," *In Proc. of the International Conference on Pattern Recognition*, vol. 1, pp. 1056–1059, 2000.
- [15] F. Saxen and A. Al-Hamadi, "Superpixels for skin segmentation," *Workshop Farbbildverarbeitung, At Wuppertal*, vol. 20, 2014.
- [16] R. P. Poudel, J. J. Zhang, D. Liu, and H. Nait-Charif, "Skin color detection using region-based approach," *International Journal of Image Processing (IJIP)*, vol. 7, no. 4, 2013.
- [17] N. K. E. Abbadi, N. Dahir, and Z. A. Alkareem, "Skin texture recognition using neural networks," *CoRR*, vol. abs/1311.6049, 2013.
- [18] M. Sofiane, B. M. Chaouki, and M. B. Yamina, "Improved skin detection using colour space and texture," *International Journal of Computer and Information Engineering*, vol. 8, no. 12, 2014.
- [19] Z. Jiang, M. Yao, and W. Jiang, "Skin detection using color, texture and space information," *Fuzzy Systems and Knowledge Discovery*, 2007.
- [20] L. Sigal, S. Sclaroff, and V. Athitsos, "Skin color-based video segmentation under time-varying illumination," *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, vol. 26, no. 7, 2004.
- [21] L. G. Shapiro and G. C. Stockman, *Computer Vision*. New Jersey, Prentice-Hall, 2001.
- [22] R. Dass, Priyanka, and S. Devi, "Image segmentation techniques," *IJECT*, vol. 3, no. 1, 2012.
- [23] C. K. Chow and T. Kaneko, "Automatic boundary detection of the left ventricle from cineangiograms," *Computers and Biomedical Research*, vol. 5, no. 4, pp. 388–410, 1972.
- [24] N. R. Pal and S. K. Pal, "A review on image segmentation techniques," *Pattern Recognition*, vol. 26, no. 9, 1993.

- [25] Y. Nakagawa and A. Rosenfeld, “Some experiments on variable thresholding,” *Pattern Recognition*, vol. 11, pp. 191–204, 1979.
- [26] N. Senthilkumaran and R. Rajesh, “Edge detection techniques for image segmentation – a survey of soft computing approaches,” *International Journal of Recent Trends in Engineering*, vol. 1, no. 2, 2009.
- [27] A. Vedaldi and S. Soatto, “Quick shift and kernel methods for mode seeking,” *Computer Vision – ECCV 2008*, vol. 5305, 2008.
- [28] B. Fulkerson and S. Soatto, “Really quick shift: Image segmentation on a gpu,” in *In Proceedings of the Workshop on Computer Vision using GPUs, held with the European Conference on Computer Vision*, 2010.
- [29] A. Frich, “Color management guide.” <https://www.color-management-guide.com/color-spaces.html/>. Accessed: 2018-04-07.
- [30] V. Vezhnevets, V. Sazonov, and A. Andreeva, “A survey on pixel-based skin color detection techniques,” *GRAPHICON03*, pp. 85–92, 2003.
- [31] P. Kakumanu, S. Makrogiannis, and N. Bourbakis, “A survey of skin-color modeling and detection methods,” *Pattern Recognition*, vol. 40, pp. 1106—1122, 2007.
- [32] J. Yang, W. Lu, and A. Waibel, “Skin-color modeling and adaptation,” *ACCV98*.
- [33] T. Segaran, *Programming Collective Intelligence*. O’Reilly, first ed., 2007.
- [34] J. McGonagle, V. Tembo, and A. Chumbley, “Gaussian mixture model.” <https://brilliant.org/wiki/gaussian-mixture-model/>. Accessed: 2018-04-09.
- [35] J. P. B. Casati, D. R. Moraes, and E. L. L. Rodrigues, “Sfa: A human skin image database based on feret and ar facial images,” *IX Workshop de Visão Computacional, Rio de Janeiro*, 2013.