Pattern Recognition and Machine Learning

James L. Crowley

ENSIMAG 3 - MMIS
Lesson 1

Fall Semester 2019
9 October 2019

Face Detection in Images

Outline

Notation

x_d	A feature. An observed or measured value.
$\vec{X}$	A vector of features. An observation.
D	The number of dimensions for the vector $\vec{X}$
$\{C_k\}$	A set of K classes (or class labels).
K	Number of classes
$\vec{X} \in C_k$	Statement that the observation $\vec{X}$ is a member of class C_k
$\hat{C}_k$	An estimated class label
	For a 2 class detection problem (K=2), $C_k \in \{P,N\}$
$R(\vec{X})$	A recognition function
$\hat{C}_k \leftarrow R(\vec{X})$	A recognition function that predicts $\hat{C}_k$ from $\vec{X}$
$\{\vec{X}_m\}$	Training data for learning.
M	The number of training samples.
$y(\vec{X}_m)$	An annotation (or ground truth) function for $\{\vec{X}_m\}$: $y(\vec{X}_m) \in \{P,N\}$
$g(\vec{X}_m)$	Discriminant function. $0 \le g(\vec{X}_m) \le 1$
$X(i,j)$	An RGB image of size RxC pixels, 8 bits per color
$P(i,j)$	A probability image of size RxC pixels. Each pixel contains the probability (or likelihood) that the pixel (i,j) is a part of a face.
$\{X_n(i,j)\}$	A set of N images for training. $\vec{X}_m = X_n(i,j)$ where $m=n \cdot i \cdot j$
N	The number of training images.
$y(X_n(i,j))$	A ground-truth function that tells if pixel (i,j) of image n is part of a face.
M_T	The number of training pixels in the target class
$h(\vec{X})$	A multidimensional histogram of integer features $\vec{X}$
Q	The number of discrete values for each dimension (quantization) of $h(\vec{X})$
V	The number of cells in the histogram $h(\vec{X})$
$W_n(u,v)$	A rectangular window at (c_i, c_j) and size (width, height)
	spanning from (l, t) to (b, r)

Organization of this course

In this course, we will use face detection as a running example to illustrate different learning techniques. We will implement and experimentally evaluate three different techniques for face detection in images.

Lab 1: Face detection using pixel level skin color detection.
Lab 2: Face detection using the Viola Jones cascade detector
Lab 3: Face detection using multilayer neural networks.

Grades are determined 50% from the labs and 50% from a final exam.

Lab exercises will be implemented, evaluated and reported by teams of 2 students. Team compositions are to be determined for next week and must be finalized by the 3rd lecture. Each team will make an oral presentation on one of the 3 labs, and provide a written report for all three labs. As you are 14 students there will be 7 teams of 2 students. Two teams will make oral reports on labs 1 and 2. Three teams will make an oral presentation on lab 3.

The lab grade is the average from the grades of 3 written reports of the team members plus the oral reports. Written reports will be due 1 week after the oral reports. The primary task for each lab is documentation and performance evaluation. The implementation may use code downloaded from the InterNet PROVIDED that you document the origin of the code. Your task is to document the code and evaluate the effects of changing parameters. You may also program your own implementation and evaluate this. Groups will be encouraged to be creative in implementing, evaluating and reporting the labs.

Labs will be performed using the OpenCV environment running under Python. In lecture 2, Nachwa Aboubakr will provide a TD to assure that each student has Open CV running under Python, and can load and display an image.

Several sets of annotated images of faces are available on the course web site.
For the first exercise we will use the FDDB (Face Detection Data Set and Benchmark) data set maintained at UMASS: http://vis-www.cs.umass.edu/fddb/
All images are RGB with each pixel containing 3 colors: Red, Green and Blue. Face regions have been hand-labeled as both boxes and ellipses.

A typical image in FDDB with its annotated face region looks like this.

Note that faces appear with an elliptical form, with major axis in the vertical direction. Annotations of face regions in FDDB are represented as an elliptical region, denoted by a 6-tuple (r_a, r_b, θ, c_x, c_y, 1) where r_a and r_b refer to the half-length of the major and minor axes, θ is the angle of the major axis with the horizontal axis, and c_x and c_y are the column and row image coordinates of the center of this ellipse.

Ellipse Data:

2002/07/24/big/img_82

1

59.268600 35.142400 1.502079 149.366900 59.365500 1

the standard form of an ellipse with a major axis along the horizontal (x) axis is:

$$\frac{(x-c_x)^2}{r_a^2}+\frac{(y-c_y)^2}{r_b^2}=1$$

for any pixel x,y inside the ellipse, $\frac{(x-c_x)^2}{r_a^2}+\frac{(y-c_y)^2}{r_b^2}<1$

For a hypothesis of a face $\vec{X}=\begin{pmatrix} c_x \\ c_y \\ r_a \\ r_b \\ \theta \end{pmatrix}$ can define a ground truth function as $y(\vec{X}_m)$

$$y(\vec{X}_m)=if\left(\frac{(x-c_x)^2}{r_a^2}+\frac{(y-c_y)^2}{r_b^2}\le 1\right) \text{ then P else N.}$$

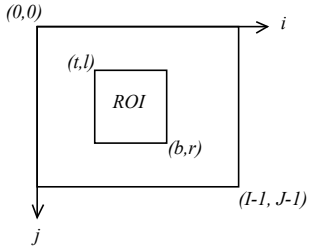
If it is necessary to rotate the face to an angle θ we can use:

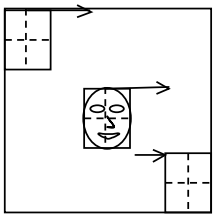
$$\frac{\left((x-c_x)\cos(\theta)+(y-c_y)\sin(\theta)\right)^2}{r_a^2}+\frac{\left((x-c_x)\sin(\theta)+(y-c_y)\cos(\theta)\right)^2}{r_b^2}=1$$

1 Face Detection using a Sliding Window Detector

Faces can occur at many different positions, orientations and sizes (scales) in an image. A common approach is to train a detection function with a standard size and orientation of faces. A sliding window process is then designed to extract (copy) the contents from every possible window for a range of positions, orientations and sizes (scales). Windows are texture mapped to a standard size and orientation used by the recognizer. This transformation is easily performed by the "texture mapping" function found in OpenCV.

Each window is called a Region of Interest (ROI). In many face detectors, the ROI is simply a sliding window that scans the entire image. This is the technique used, for example, with the Viola Jones Face Detector.





Generally both a range of sizes and positions of the ROI are scanned. Size can be on a linear scale or a logarithmic scale. There are good reasons to use a logarithmic scale for size (e.g. fractional powers of 2).

A common representation for the ROI is a rectangle represented by four coordinates: (left, top, right, bottom) or (*l, t, r, b*)

l - "left" - first column of the ROI.
t - "top" - first row of the ROI.
r - "right" - last column of the ROI.
b - "bottom" - last row of the ROI

(*l, t, r, b*) can be seen as a bounding box, expressed by opposite corners (l,t), (r,b). The center position, width and height are simply

$$c_i = \frac{l+r-1}{2}, \qquad c_j = \frac{b+t-1}{2}, \qquad w = l-r+1, \qquad h = b-t+1$$

In some cases it is more convenient to fix c_i, c_j, w, h and calculate (*l, t, r, b)*
In this case the bounding box ROI is:

$$t = c_j - \frac{h-1}{2}, \qquad b = c_j + \frac{h-1}{2}, \qquad l = c_i - \frac{w-1}{2}, \qquad r = c_i + \frac{w-1}{2}$$

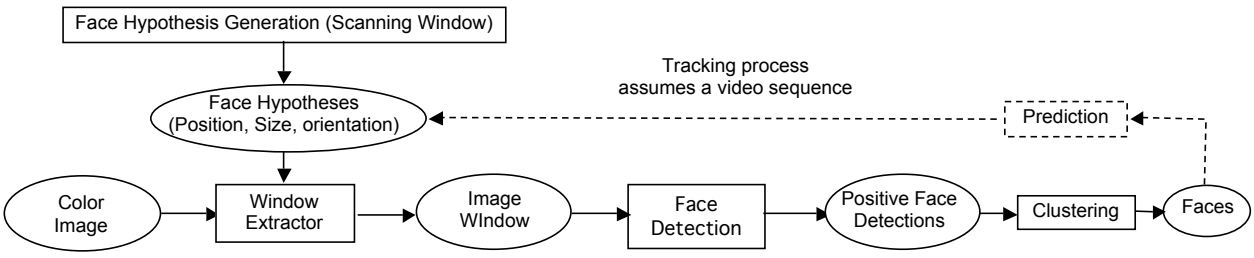
The ratio of w and h can be fixed or variable. It is convenient to assure that w and h are odd numbers so that the boundaries fall on an integer pixel.

For detection in static images, we will need to test a range of ROIs with different positions and sizes. Note that with sufficient computing power, all windows can be processed in parallel.

In a scanning window system, the scanning step size (s) may also be varied. This may be the same size for row and column direction, or different step sizes may be used (say s_i and s_j). When the discriminant function is highly correlated at adjacent pixels (as is the case with color skin detection), then the resulting detection function will be very smooth. In this case a larger step size may be sufficient. The largest reasonable value for s is half the width of a target (and half the height if different step sizes in row and column are used). Beyond this the detection degrades rapidly because of aliasing effects, as can be demonstrated using signal processing techniques that are beyond the scope of this class.

When computing time is an issue, the detection algorithm can be optimized using hierarchical search, starting with a large step size (Typically a power of 2) and recursively searching at higher step sizes near detected points.

Recognition typically returns a "likelihood" estimate for a face at many adjacent positions, orientations and sizes. It is necessary to cluster these adjacent detections to determine the best position for a single face. This can be done using the center of gravity of the likelihoods.

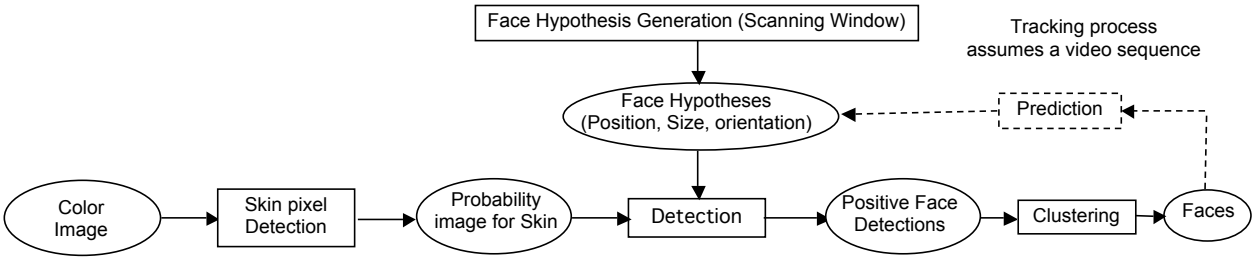


The same process can be used for tracking previous detected faces in a video sequence. In this case, face detection is limited to a small range of positions, sizes and orientations, estimated from the detection from the previous image.

2 Face Detection using Skin Color

Our first lab will use a detector for skin pixels to indicate possible faces. This process can be performed before the sliding window. Skin color can be used to construct a simple detector for skin pixels in images. Color skin pixels can then be used to detect and track "blobs" that represent faces, hands and other skin colored regions in images.

The detector works by first computing the probability that each pixel contains skin. A sliding window (Region of Interest or ROI) is then scanned over the image. At each position, a weighted sum of probabilities is determined. Regions for which the weighted sum is above threshold are detected as faces. Adjacent face detections are grouped to form a single face detection.



Algorithm:
1) Compute probability of skin at each pixel.
2) Test for faces at possible positions and sizes (scanning Window).
3) Cluster adjacent detections.
4) Estimate precise face position, size and orientation from clusters of detections.

The function typically returns a "likelihood" estimate for a face at many adjacent positions, orientations and sizes.

We can break the problem down into 3 challenges.
Each challenge requires experimentally determining a set of parameters.
You are asked to determine these parameters experimentally.

2.1 Challenge 1: Detecting skin pixels with color

Base Line: Ratio of RGB histograms

The first challenge is to transform a color (RGB) image, $X(i,j)$ into an image where each pixel provides an estimate of the probability of skin, $P(i,j)$.

Assume a color image : $X(i,j) = \begin{pmatrix} R \\ G \\ B \end{pmatrix}(i,j)$

The algorithm will use a lookup table to convert color to probability.

$$P(i,j) \leftarrow L(\vec{X}(i,j))$$

The lookup table is constructed as a ratio of histograms, as explained below.
We can improve the results can be obtained by using a normalized color space as well as by tuning the quantization of the histogram to the data.

In the following, assume that we have a color image, where each pixel *(i,j)* is a color vector, $X(i,j)$, composed of 3 integers between 0 and 255 representing Red, Green and Blue.

$$X(i,j) = \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Suppose that we have N color images of size CxR pixels, $X_n(i,j)$.
This gives a total of $M = C \times R \times N$ pixels.
Call this set of pixels $\vec{X}_m = X_n(i,j)$ where $m = i \cdot j \cdot n$

Suppose that we have a ground truth function $y(X_m)$ that tells us whether each pixel is target (P or Positive) or not target (N or Negative). A subset of M_T pixels that belong to a target class, T.

We allocate two tables $h(\vec{X})$ and $h_T(\vec{X})$ and use these to construct two histograms.
we can also write this as $\forall_m h(\vec{X}_m) = h(\vec{X}_m) + 1$

and $\forall_m y(\vec{X}_m) = P : h_T(\vec{X}_m) = h_T(\vec{X}_m) + 1 \; ; \; M_T \leftarrow M_T + 1$

For a color vector, $\vec{X} = \begin{pmatrix} R \\ G \\ B \end{pmatrix}$ we have two probabilities:

$$P(\vec{X}) = \frac{1}{M} h(\vec{X}) \quad \text{and} \quad P(\vec{X} \mid T) = \frac{1}{M_T} h_T(\vec{X})$$

Bayes rule tells us that we can estimate the probability that a pixel belongs to target class (Skin) given its color, $\vec{X}$ as:

$$P(Skin \mid \vec{X}) = \frac{P(\vec{X} \mid \text{Skin}) P(\text{Skin})}{P(\vec{X})}$$

P(Skin) is the probability that a pixel belongs to the target class.
This can be estimated from the training data by:

$$P(Skin) = \frac{M_T}{M}$$

From this we can show that the probability that a pixel is skin, is simply the ratio of the two tables.

$$P(Skin \mid \vec{X}) = \frac{P(\vec{X} \mid Skin) P(Skin)}{P(\vec{X})} = \frac{\frac{1}{M_T} h_t(\vec{X}) \cdot \frac{M_t}{M}}{\frac{1}{M} h(\vec{X})} = \frac{h_T(\vec{X})}{h(\vec{X})}$$

We can use this to compute a lookup table $L_{Skin}(\vec{X}) = \frac{h_T(\vec{X})}{h(\vec{X})}$

if $h(\vec{X}) = 0$ then $h_{Skin}(\vec{X}) = 0 = 0$ because $h_{Skin}(\vec{X})$ is a subset of $h(\vec{X})$.
we will need to test this case to avoid divide by 0.

If we ASSUME that a new image, *X(i,j)*, has similar illumination and color composition then we can use this technique to assign a probability to each pixel by table lookup. The result is an image in which each pixel is a probability *P(i,j)* that the pixel *(i,j)* belongs to class skin.

$$P(i, j) = L_{Skin}(\vec{X}(i, j))$$

Details:

1) Alternative color codings may provide better recognition.
2) Different color quantizations may provide better recognition.

In this example, $h(\vec{X})$ and $h_T(\vec{X})$ are composed of $2^8 \cdot 2^8 \cdot 2^8 = 2^{24}$ cells.
We define this as the Capacity of the histogram, V

$V = 2^8 \cdot 2^8 \cdot 2^8 = 2^{24}$ cells.

In general, $V = Q^D$ where Q is the number of values per feature and D is the number of features.

This can result in a very sparse histogram. The reliability can be improved by using more training images from more cases.

A naive statistics view says to have at least 10 training samples for histogram cell.
That is $M \geq 10\ V$. However, it is often easier to work with powers of 2.
For example, $2^3 = 8 \approx 10$ which is approximately 10.
This suggest that we required $M \geq 8\ V = 2^3\ V$.

Thus we would need $2^3 \cdot 2^{24} = 2^{27}$ training pixels. 2^7 Meg.
(Note that a 1024 x 1024 image contains 2^{20} pixels. This is the definition of 1 Meg)

Obtaining 2^7 Meg pixels is not a problem for $P(\vec{X}) = \frac{1}{M} h(\vec{X})$ but may be a problem for training the histogram of target pixels $P(\vec{X} \mid Skin) = \frac{1}{M_T} h_T(\vec{X})$.

A more realistic view is that the training data must contain a variety of training samples that reflect that variations in the real world.

What can we do? Two approaches:
We can reduce the number of values, Q, for each feature, or
we can reduce the number of features.

For example, for many color images, Q=32 color values are sufficient to detect objects. We simply divide each color R, G, B by 8.

R' = Trunc(R/8), G'=Trunc(G/8), B'=Trunc(B/8).

We can also use physics to look for features that are "invariant".

Variation: Detection with Chrominance

Luminance captures local surface orientation (3D shape) while Chrominance is a signature for object pigment (identity). The color of pigment for any individual is generally constant. Luminance can change with pigment density (eg. Lips), and skin surface orientation, but chrominance will remain invariant.

Several methods exist to transform the (RGB) color pixels into a color space that separates Luminance from Chrominance.

$$\begin{pmatrix} L \\ c_1 \\ c_2 \end{pmatrix} \Leftarrow \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

A popular space for skin detection is computed by dividing R and G by luminance. These are often called "r" and "g" in the literature.

Luminance: L= R+G+B

Chrominance : $r = c_1 = \frac{R}{R+G+B}$ $g = c_2 = \frac{G}{R+G+B}$

The terms r and g have values between 0 and 1. To count statistics we need integer values. For this, it is common to represent r and g as integer numbers coded with Q values between 0 and Q–1 by :

$$r = trunc\left((Q-1)\cdot\frac{R}{R+G+B}\right) \qquad g = trunc\left((Q-1)\cdot\frac{G}{R+G+B}\right)$$

From experience, Q = 32 color values seems to work well for skin with most web cameras, but this may change for certain data sets and should be experimentally verified.

Thus we can use a normalized vector $\vec{X} = \begin{pmatrix} r \\ g \end{pmatrix}$ as an invariant color signature for detecting skin in images.

Suppose we have a set of N training images $\{X_n(i,j)\}$ of size *RxC* where each pixel is an RGB color vector. This gives a total of *M=NxIxJ* color pixels.

Suppose that M_T of these are labeled as skin pixels i.e. $y(X_n(i,j)) = P$

We allocate two tables: $h(r,g)$ and $h_T(r,g)$ of size Q xQ.
As before:

For all i,j,n in the training set $\{X_n(i,j)\}$:
BEGIN

$$r = trunc\left((Q-1)\cdot\frac{R}{R+G+B}\right) \quad g = trunc\left((Q-1)\cdot\frac{G}{R+G+B}\right)$$

$$h(r,g) = h(r,g)+1$$

IF $(y(X_n(i,j) == P)$ *THEN* $h_T(r,g) = h_T(r,g)+1$; $M_T \leftarrow M_T+1$

END

As before, we can obtain a lookup table L_{skin}(r,g) that gives the probability that a pixel is skin. $L_{skin}(r,g) = \frac{h_T(r,g)}{h(r,g)}$

Given a new RGB image $C(i,j)$: For all i, j :

$$r = trunc\left((Q-1)\cdot\frac{R}{R+G+B}\right) \qquad g = trunc\left((Q-1)\cdot\frac{G}{R+G+B}\right)$$

$$P(i,j) = L_{skin}(r,g)$$

Where $P(i,j)$ is an image in which each pixel is a probability value from 0 to 1.
Probabilities can be expressed, of course be expressed as integers by multiplying by a quantization value (Q).

Free Parameters

A number of parameters remain unspecified. Specification of these parameters generally depends on the application domain and the training data. Often these parameters make it possible to trade off error rates for computing time. To properly evaluate the algorithm, you should measure performance and compare performance over a range of parameters.

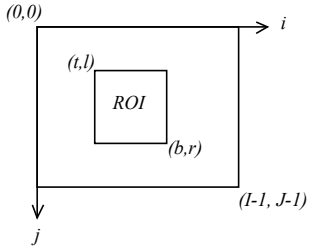
Free parameters for color based skin detection include:

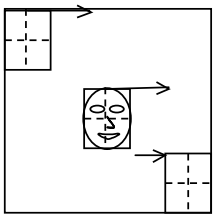
- The use of color coding (eg, RGB, rg, other codings for chrominance)
- Value of Q – the quantification for color color values.
- Training Data – different training and test regimes with variations such as the number of folds, and whether to train with a subset of the folds or all folds.

2.2 Challenge 2: Detecting Faces with Skin Color

We can detect faces from the mass of skin detections within a "Region of Interests" (ROI). The ROI is determined either by a-prior knowledge or by some form of search procedure. In many cases this is based on tracking of targets in the scene. This is not possible for our example, because our training and test data are restricted to static images.

In many vision detectors the ROI is simply a sliding window that scans the entire image as explained above. This is the technique used, for example, with the Viola Jones Face Detector.





A common representation for the ROI is a rectangle represented by four coordinates: (left, top, right, bottom) or (l, t, r, b). Alternatively the center, width and height of the ROI: c_i, c_j, w, h.

For detection in static images, we will need to test a range of ROIs with different positions and sizes. The scanning step size (s) may also be varied. This may be the same size for row and column direction, or different step sizes may be used (say s_i and s_j)

Baseline: Sliding Window Detector

We can compute the ROI parameters for a face as a bounding box for an ellipse.

Let us define a face hypothesis as $\vec{X} = \begin{pmatrix} c_i \\ c_j \\ w \\ h \end{pmatrix}$

Since faces are generally vertical, we may assume that the ellipse is oriented with the major axis aligned with the row direction (j). The bounding box ROI is:

$$t = c_j - \frac{h-1}{2}, \qquad b = c_j + \frac{h-1}{2}, \qquad l = c_i - \frac{w-1}{2}, \qquad r = c_i + \frac{w-1}{2}$$

The likelihood of a face at a position (c_i, c_j) of size (w,h) is:

$$g(\vec{X}) = \frac{1}{w \cdot h} \sum_{i=l}^{r} \sum_{j=t}^{b} P(i,j)$$

We can bias this likelihood to make a decision:

IF $g(\vec{X}_m) + B > 0.5$ THEN $R(\vec{X}_m) = P$ else $R(\vec{X}_m) = N$

And of course

IF $R(\vec{X}_m) = y(\vec{X}_m)$ THEN T else F.

This technique will detect "faces" for a range of positions and sizes. As long at the detections overlap with the face ellipse in the data-base they are TRUE detections.

The problem with this technique is that faces are not square. The ROI gives equal weight to corners as the center. We can do better by weighting the pixels using a Gaussian function. This is called a "robust estimator" because it tends to reject outliers.

Variation: Detection using a Gaussian mask.

In machine vision, fixation serves to reduce computational load and reduce errors by focusing processing on parts of the image that are most likely to contain information. A commonly practice is to use a Gaussian Window to suppress signals outside of the fixated region. A similar technique is used to suppress outliers for robust estimation.

A Gaussian window has the form:

$$G(\vec{P}; \vec{\mu}, \Sigma) = e^{-\frac{1}{2}\left(\vec{P}-\vec{\mu}\right)^T \Sigma^{-1} \left(\vec{P}-\vec{\mu}\right)}$$

Where $\vec{P} = \begin{pmatrix} i \\ j \end{pmatrix}$ represents image positions, and $\vec{\mu} = \begin{pmatrix} \mu_i \\ \mu_j \end{pmatrix}$ is the center position of the window.

The covariance matrix of the window is: $\Sigma = \begin{pmatrix} \sigma_i^2 & \sigma_{ij} \\ \sigma_{ij} & \sigma_j^2 \end{pmatrix}$

The coefficients have a value of 1 at the center of the mask, and taper to 0.1 at a distance of 2σ and to 0.01 at a distance of 3σ.

We can use the parameters of the window to define a face hypothesis. $\vec{X} = \begin{pmatrix} \mu_i \\ \mu_j \\ \sigma_i^2 \\ \sigma_j^2 \\ \sigma_{ij} \end{pmatrix}$

Normally the Gaussian mask should be computed within a ROI determined by a bounding box. Typically the bounding box should be at least 2σ (standard deviations), but if speed is more important the false negatives, then computation time can be reduced by using a smaller ROI, down to as small as 1 standard deviation.

We can define the ROI as a 2σ bounding box:

$$t = c_j - 2\sigma_j, \qquad b = c_j + 2\sigma_j, \qquad l = c_i - 2\sigma_i, \qquad r = c_i + 2\sigma_i$$

To evaluate a face hypothesis, we compute the face likelihood as a weighed sum of the skin probabilities by the Gaussian mask.

$$g(\vec{X}) = \sum_{i=l}^{r} \sum_{j=t}^{b} P(i,j) G(i,j;\vec{X})$$

As before, the discriminant, $g(\vec{X})$, has a value between 0 and 1. This is the likelihood that a face may be found at $\vec{X}$.

As before, faces are detected as:

IF $g(\vec{X}_m) + B > 0.5$ THEN $R(\vec{X}_m) = P$ else $R(\vec{X}_m) = N$

Free parameters to test

As with color skin detection a number of parameters remain unspecified. To properly evaluate the algorithm, you should measure performance and compare performance over a range of parameters.

Free parameters for color face detection include:
For a simple scanning window, these include:

- Width and height of ROI
- Range of positions
- Step size for scanning windows
- The percentage of overlap with the ground truth that is considered a TRUE detection.

For a Gaussian detection window, parameters also include

- The width and height of the ROI compared to the standard deviations of the principal axis.
- The range and step sizes for orientation of the Gaussian window.

2.3 Challenge 3: Face Localization

Detection vs Localization

Detection: A face is considered to be "detected" if a positive detection is found at a position that overlaps the face in the ground truth. The simplest form of test is to verify that the location of the face is within the ellipse of the ground truth label for a face in the image.

For a face hypothesis at $\vec{X} = \begin{pmatrix} c_i \\ c_j \\ w \\ h \end{pmatrix}$ a face is detected if $R(\vec{X})$=P

The detection is TRUE if $R(\vec{X})=y(\vec{X})$

Using the ground truth $y(\vec{X}) = \begin{cases} P & if(\frac{(x-c_x)^2}{r_a^2} + \frac{(y-c_y)^2}{r_b^2} \le 1 \\ N & otherwise \end{cases}$

The Hypothesis is a TRUE POSITIVE detection if $y(\vec{X})$=P and $R(\vec{X})=y(\vec{X})$.

A large number of TP faces will be detected for each ground truth face. If the search space includes multiple scales and orientations, than the number of detected faces will be even larger. All of these correspond to the same face!

Localization (more precisely parameter estimation) specifies precisely where, and with what parameters, the face may be found. There should be only ONE face located for each true face. The face should be located at a position, size and orientation as close to the true face as possible. When searching at multiple scales and orientations, each detection has the form of an error vector.

A distance metric, relating degrees and scale change to position is required to reduce this to a single number. For discrete samples the distance metric can provided implicitly by the sample step size in scale, orientation and position.

We can obtain a location (or parameter estimation) from multiple positive detections by suppressing detections for which the discriminant, $g(\vec{X})$ is not a local maximum. This requires specifying a measure for locality.

We can also estimate the parameters of the face from the moments of a "cloud" of detections at multiple positions, scales and orientations. This is the same robust estimation technique that we used above, extended to robustly estimate position, size and orientation.

Baseline: Localization by non-maximum suppression

In order to suppress non-maximal detections, the easiest method is to build a list of detections hypotheses, $\{\vec{X}\}$ over the desired range of positions, scales, orientations and any other parameters, and then filter this list to remove any hypothesis for which the discriminant not a local maximum. (A maximum within a some distance R.)

$\forall \vec{X}$
$\forall \vec{X}_i, \vec{X}_j \in \{\vec{X}\}$: IF $Dist(\vec{X}_i, \vec{X}_j) < R$ AND $g(\vec{X}_i) < g(\vec{X}_j)$ THEN $\{\vec{X}\} \leftarrow \{\vec{X}\} - \vec{X}_i$

This requires defining some notion of distance that includes position, scale and orientation. This is generally done with regard to an expected range of positions, orientations and scales over which a single face would be detected. For example, using a Mahalanobis distance (Distance normalized by covariance).

Variation: Localization by Robust Estimation

We can use robust estimation to estimate the most likely parameters from a set of detections. To do this, we will calculate the weighted moments from the set of detections.

Suppose that we have a set of N detections: $\{\vec{X}_n\}$ and that for each detection we have a discriminant $g(\vec{X}_n)$.

The mass of the detections is $M = \sum_{n=1}^{N} g(\vec{X}_n)$. This is the zeroth moment of the set of detections.

The "expected value" is $E\{\vec{X}\} = \frac{1}{M}\sum_{n=1}^{N} g(\vec{X}_n)\cdot\vec{X}_n$

This is the first moment (or center of gravity) of the values of $\{\vec{X}_n\}$.

The expected value is a vector of first moments for each parameter:

For D parameters, the center of gravity is a vector

$$\vec{\mu} = E\{\vec{X}\} = \frac{1}{M}\sum_{n=1}^{N} g(\vec{X}_n)\vec{X}_n = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \dots \\ \mu_D \end{pmatrix} = \begin{pmatrix} \frac{1}{M}\sum_{n=1}^{N} g(\vec{X}_n)X_{1n} \\ \frac{1}{M}\sum_{n=1}^{N} g(\vec{X}_n)X_{2n} \\ \dots \\ \frac{1}{M}\sum_{n=1}^{N} g(\vec{X}_n)X_{Dn} \end{pmatrix}$$

The vector $\vec{\mu}$ the vector of weighted averages for the components of $\vec{X}_n$.

If there is only one face inside the set $\{\vec{X}_n\}$ of positive detections then $\vec{\mu}$ provides the most likely estimate for set of parameters the detection, (e.g. position, size and orientation).

In the case of multiple faces, the best estimate for each of the faces can be determined using the Expectation Maximization (EM) algorithm. Alternatively, the estimation may be limited to faces detections within a small region of the image.

Free parameters to test

As with color face detection a number of parameters remain unspecified. To properly evaluate the algorithm, you should measure performance and compare performance over a range of parameters.

Free parameters for color face detection include:
For a simple scanning window, these include:

- Number of Faces in the image
- Range of positions, size and orientations to test
- Distance to use for non-maximal suppression.
- Size of the region used to cluster faces detections.