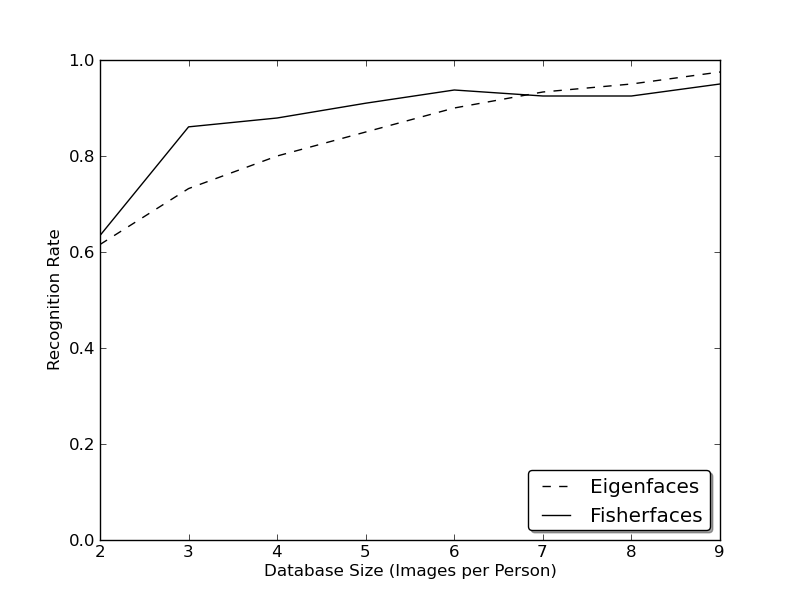
[Local Binary Patterns Histograms](https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html" \l "id38)

Eigenfaces and Fisherfaces take a somewhat holistic approach to face recognition. You treat your data as a vector somewhere in a high-dimensional image space. We all know high-dimensionality is bad, so a lower-dimensional subspace is identified, where (probably) useful information is preserved. The Eigenfaces approach maximizes the total scatter, which can lead to problems if the variance is generated by an external source, because components with a maximum variance over all classes aren’t necessarily useful for classification (see <http://www.bytefish.de/wiki/pca_lda_with_gnu_octave>). So to preserve some discriminative information we applied a Linear Discriminant Analysis and optimized as described in the Fisherfaces method. The Fisherfaces method worked great... at least for the constrained scenario we’ve assumed in our model.

Now real life isn’t perfect. You simply can’t guarantee perfect light settings in your images or 10 different images of a person. So what if there’s only one image for each person? Our covariance estimates for the subspace *may* be horribly wrong, so will the recognition. Remember the Eigenfaces method had a 96% recognition rate on the AT&T Facedatabase? How many images do we actually need to get such useful estimates? Here are the Rank-1 recognition rates of the Eigenfaces and Fisherfaces method on the AT&T Facedatabase, which is a fairly easy image database:

[](https://docs.opencv.org/2.4/_images/at_database_small_sample_size.png)

So in order to get good recognition rates you’ll need at least 8(+-1) images for each person and the Fisherfaces method doesn’t really help here. The above experiment is a 10-fold cross validated result carried out with the facerec framework at: <https://github.com/bytefish/facerec>. This is not a publication, so I won’t back these figures with a deep mathematical analysis. Please have a look into [[KM01]](https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#km01) for a detailed analysis of both methods, when it comes to small training datasets.

So some research concentrated on extracting local features from images. The idea is to not look at the whole image as a high-dimensional vector, but describe only local features of an object. The features you extract this way will have a low-dimensionality implicitly. A fine idea! But you’ll soon observe the image representation we are given doesn’t only suffer from illumination variations. Think of things like scale, translation or rotation in images - your local description has to be at least a bit robust against those things. Just like [**SIFT**](https://docs.opencv.org/2.4/modules/nonfree/doc/feature_detection.html#SIFT%20:%20public%20Feature2D), the Local Binary Patterns methodology has its roots in 2D texture analysis. The basic idea of Local Binary Patterns is to summarize the local structure in an image by comparing each pixel with its neighborhood. Take a pixel as center and threshold its neighbors against. If the intensity of the center pixel is greater-equal its neighbor, then denote it with 1 and 0 if not. You’ll end up with a binary number for each pixel, just like 11001111. So with 8 surrounding pixels you’ll end up with 2^8 possible combinations, called *Local Binary Patterns* or sometimes referred to as *LBP codes*. The first LBP operator described in literature actually used a fixed 3 x 3 neighborhood just like this:

[](https://docs.opencv.org/2.4/_images/lbp.png)

[Algorithmic Description](https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#id39)

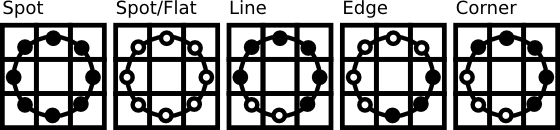
A more formal description of the LBP operator can be given as:

LBP(x_c, y_c) = \sum_{p=0}^{P-1} 2^p s(i_p - i_c)

, with (x_c, y_c) as central pixel with intensity i_c; and i_n being the intensity of the the neighbor pixel. s is the sign function defined as:

\begin{equation}
s(x) =
\begin{cases}
1 & \text{if $x \geq 0$}\\
0 & \text{else}
\end{cases}
\end{equation}

This description enables you to capture very fine grained details in images. In fact the authors were able to compete with state of the art results for texture classification. Soon after the operator was published it was noted, that a fixed neighborhood fails to encode details differing in scale. So the operator was extended to use a variable neighborhood in [[AHP04]](https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#ahp04). The idea is to align an abritrary number of neighbors on a circle with a variable radius, which enables to capture the following neighborhoods:

[](https://docs.opencv.org/2.4/_images/patterns.png)

For a given Point (x_c,y_c) the position of the neighbor (x_p,y_p), p \in P can be calculated by:

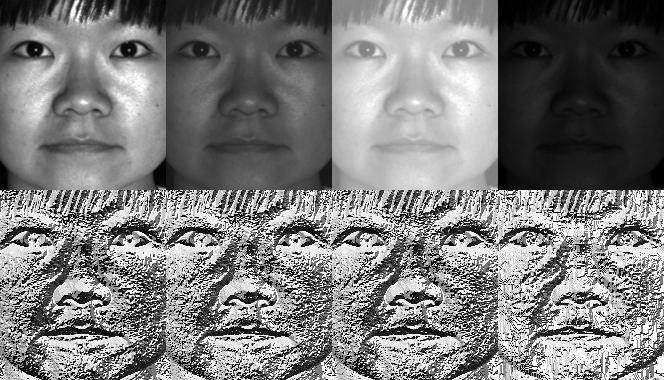
\begin{align*}
x_{p} & = & x_c + R \cos({\frac{2\pi p}{P}})\\
y_{p} & = & y_c - R \sin({\frac{2\pi p}{P}})
\end{align*}

Where R is the radius of the circle and P is the number of sample points.

The operator is an extension to the original LBP codes, so it’s sometimes called *Extended LBP* (also referred to as *Circular LBP*) . If a points coordinate on the circle doesn’t correspond to image coordinates, the point get’s interpolated. Computer science has a bunch of clever interpolation schemes, the OpenCV implementation does a bilinear interpolation:

\begin{align*}
f(x,y) \approx \begin{bmatrix}
    1-x & x \end{bmatrix} \begin{bmatrix}
    f(0,0) & f(0,1) \\
    f(1,0) & f(1,1) \end{bmatrix} \begin{bmatrix}
    1-y \\
    y \end{bmatrix}.
\end{align*}

By definition the LBP operator is robust against monotonic gray scale transformations. We can easily verify this by looking at the LBP image of an artificially modified image (so you see what an LBP image looks like!):

[](https://docs.opencv.org/2.4/_images/lbp_yale.jpg)

So what’s left to do is how to incorporate the spatial information in the face recognition model. The representation proposed by Ahonen et. al [[AHP04]](https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#ahp04) is to divide the LBP image into m local regions and extract a histogram from each. The spatially enhanced feature vector is then obtained by concatenating the local histograms (**not merging them**). These histograms are called *Local Binary Patterns Histograms*.