Face Recognition with OpenCV

Introduction

OpenCV (Open Source Computer Vision) is a popular computer vision library started by Intel in 1999. The cross-platform library sets its focus on real-time image processing and includes patent-free implementations of the latest computer vision algorithms. In 2008 Willow Garage took over support and OpenCV 2.3.1 now comes with a programming interface to C, C++, Python and Android. OpenCV is released under a BSD license so it is used in academic projects and commercial products alike.

OpenCV 2.4 now comes with the very new FaceRecognizer class for face recognition, so you can start experimenting with face recognition right away. This document is the guide I've wished for, when I was working myself into face recognition. It shows you how to perform face recognition with FaceRecognizer in OpenCV (with full source code listings) and gives you an introduction into the algorithms behind. I'll also show how to create the visualizations you can find in many publications, because a lot of people asked for.

The currently available algorithms are:

- Eigenfaces (see EigenFaceRecognizer::create)
- Fisherfaces (see FisherFaceRecognizer::create)
- Local Binary Patterns Histograms (see LBPHFaceRecognizer::create)

You don't need to copy and paste the source code examples from this page, because they are available in the src folder coming with this documentation. If you have built OpenCV with the samples turned on, chances are good you have them compiled already! Although it might be interesting for very advanced users, I've decided to leave the implementation details out as I am afraid they confuse new users.

All code in this document is released under the BSD license, so feel free to use it for your projects.

Face Recognition

Face recognition is an easy task for humans. Experiments in [252] have shown, that even one to three day old babies are able to distinguish between known faces. So how hard could it be for a computer? It turns out we know little about human recognition to date. Are inner features (eyes, nose, mouth) or outer features (head shape, hairline) used for a successful face recognition? How do we analyze an image and how does the brain encode it? It was shown by David Hubel and Torsten Wiesel, that our brain has specialized nerve cells responding to specific local features of a scene, such as lines, edges, angles or movement. Since we don't see the world as scattered pieces, our visual cortex must somehow combine the different sources of information into useful patterns. Automatic face recognition is all about extracting those meaningful features from an image, putting them into a useful representation and performing some kind of classification on them.

Face recognition based on the geometric features of a face is probably the most intuitive approach to face recognition. One of the first automated face recognition systems was described in [126] : marker points (position of eyes, ears, nose, ...) were used to build a feature vector (distance between the points, angle between them, ...). The recognition was performed by calculating the euclidean distance between feature vectors of a probe and reference image. Such a method is robust against changes in illumination by its nature, but has a huge drawback: the accurate registration of the marker points is complicated, even with state of the art algorithms. Some of the latest work on geometric face recognition was carried out in [38] . A 22-dimensional feature vector was used and experiments on large datasets have shown, that geometrical features alone may not carry enough information for face recognition.

The Eigenfaces method described in [253] took a holistic approach to face recognition: A facial image is a point from a high-dimensional image space and a lower-dimensional representation is found, where classification becomes easy. The lower-dimensional subspace is found with Principal Component Analysis, which identifies the axes with maximum variance. While this kind of transformation is optimal from a reconstruction standpoint, it doesn't take any class labels into account. Imagine a situation where the variance is generated from external sources, let it be light. The axes with maximum variance do not necessarily contain any discriminative information at all, hence a classification becomes impossible. So a class-specific projection with a Linear Discriminant Analysis was applied to face recognition in [18] . The basic idea is to minimize the variance within a class, while maximizing the variance between the classes at the same time.

Recently various methods for a local feature extraction emerged. To avoid the high-dimensionality of the input data only local regions of an image are described, the extracted features are (hopefully) more robust against partial occlusion, illumation and small sample size. Algorithms used for a local feature extraction are Gabor Wavelets ([276]), Discrete Cosinus Transform ([170]) and Local Binary Patterns ([3]). It's still an open research question what's the best way to preserve spatial information when applying a local feature extraction, because spatial information is potentially useful information.

Face Database

Let's get some data to experiment with first. I don't want to do a toy example here. We are doing face recognition, so you'll need some face images! You can either create your own dataset or start with one of the available face databases, http://face-rec.org/databases/ gives you an up-to-date overview. Three interesting databases are (parts of the description are quoted from http://face-rec.org):

- **AT&T Facedatabase** The AT&T Facedatabase, sometimes also referred to as *ORL Database of Faces*, contains ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous background with the subjects in an upright, frontal position (with tolerance for some side movement).
- **Yale Facedatabase A**, also known as Yalefaces. The AT&T Facedatabase is good for initial tests, but it's a fairly easy database. The Eigenfaces method already has a 97% recognition rate on it, so you won't see any great improvements with other algorithms. The Yale Facedatabase A (also known as Yalefaces) is a more appropriate dataset for initial experiments, because the recognition problem is harder. The database consists of 15 people (14 male, 1 female) each with 11 grayscale images sized 320×243 pixel. There are changes in the light conditions (center light, left light, right light), facial expressions (happy, normal, sad, sleepy, surprised, wink) and glasses (glasses, no-glasses).

  The original images are not cropped and aligned. Please look into the **Appendix** for a Python script, that does the job for you.

- **Extended Yale Facedatabase B** The Extended Yale Facedatabase B contains 2414 images of 38 different people in its cropped version. The focus of this database is set on extracting features that are robust to illumination, the images have almost no variation in emotion/occlusion/... . I personally think, that this dataset is too large for the experiments I perform in this document. You better use the AT&T Facedatabase for intial testing. A first version of the Yale Facedatabase B was used in [18] to see how the Eigenfaces and Fisherfaces method perform under heavy illumination changes. [136] used the same setup to take 16128 images of 28 people. The Extended Yale Facedatabase B is the merge of the two databases, which is now known as Extended Yalefacedatabase B.

Preparing the data

Once we have acquired some data, we'll need to read it in our program. In the demo applications I have decided to read the images from a very simple CSV file. Why? Because it's the simplest platform-independent approach I can think of. However, if you know a simpler solution please ping me about it. Basically all the CSV file needs to contain are lines composed of a filename followed by a ; followed by the label (as *integer number*), making up a line like this:

```
/path/to/image.ext;0
```

Let's dissect the line. /path/to/image.ext is the path to an image, probably something like this if you are in Windows: C:/faces/person0/image0.jpg. Then there is the separator ; and finally we assign the label 0 to the image. Think of the label as the subject (the person) this image belongs to, so same subjects (persons) should have the same label.

Download the AT&T Facedatabase from AT&T Facedatabase and the corresponding CSV file from at.txt, which looks like this (file is without ... of course):

```
./at/s1/1.pgm;0
./at/s1/2.pgm;0
...
./at/s2/1.pgm;1
./at/s2/2.pgm;1
...
./at/s40/1.pgm;39
./at/s40/2.pgm;39
```

Imagine I have extracted the files to D:/data/at and have downloaded the CSV file to D:/data/at.txt. Then you would simply need to Search & Replace ./ with D:/data/. You can do that in an editor of your choice, every sufficiently advanced editor can do this. Once you have a CSV file with valid filenames and labels, you can run any of the demos by passing the path to the CSV file as parameter:

```
facerec_demo.exe D:/data/at.txt
```

Please, see **Creating the CSV File** for details on creating CSV file.

Eigenfaces

The problem with the image representation we are given is its high dimensionality. Two-dimensional $p \times q$ grayscale images span a $m = pq$-dimensional vector space, so an image with 100×100 pixels lies in a $10,000$-dimensional image space already. The question is: Are all dimensions equally useful for us? We can only make a decision if there's any variance in data, so what we are looking for are the components that account for most of the information. The Principal Component Analysis (PCA) was independently proposed by Karl Pearson (1901) and Harold Hotelling (1933) to turn a set of possibly correlated variables into a smaller set of uncorrelated variables. The idea is, that a high-dimensional dataset is often described by correlated variables and therefore only a few meaningful dimensions account for most of the information. The PCA method finds the directions with the greatest variance in the data, called principal components.

Algorithmic Description of Eigenfaces method

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a random vector with observations $x_i \in R^d$.

1. Compute the mean μ

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

2. Compute the the Covariance Matrix S

$$S = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^{T};$$

3. Compute the eigenvalues λ_i and eigenvectors v_i of S

$$Sv_i = \lambda_i v_i, i = 1, 2, \ldots, n$$

4. Order the eigenvectors descending by their eigenvalue. The k principal components are the eigenvectors corresponding to the k largest eigenvalues.

The k principal components of the observed vector x are then given by:

$$y = W^T(x - \mu)$$

where $W = (v_1, v_2, \ldots, v_k)$.

The reconstruction from the PCA basis is given by:

$$x = Wy + \mu$$

where $W = (v_1, v_2, \ldots, v_k)$.

The Eigenfaces method then performs face recognition by:

- Projecting all training samples into the PCA subspace.
- Projecting the query image into the PCA subspace.
- Finding the nearest neighbor between the projected training images and the projected query image.

Still there's one problem left to solve. Imagine we are given 400 images sized 100×100 pixel. The Principal Component Analysis solves the covariance matrix $S = XX^T$, where $size(X) = 10000 \times 400$ in our example. You would end up with a 10000×10000 matrix, roughly $0.8GB$. Solving this problem isn't feasible, so we'll need to apply a trick. From your linear algebra lessons you know that a $M \times N$ matrix with $M > N$ can only have $N - 1$ non-zero eigenvalues. So it's possible to take the eigenvalue decomposition $S = X^TX$ of size $N \times N$ instead:

$$X^TXv_i = \lambda_i vi$$

and get the original eigenvectors of $S = XX^T$ with a left multiplication of the data matrix:

$$XX^T(Xv_i) = \lambda_i(Xv_i)$$

The resulting eigenvectors are orthogonal, to get orthonormal eigenvectors they need to be normalized to unit length. I don't want to turn this into a publication, so please look into [63] for the derivation and proof of the equations.

Eigenfaces in OpenCV

For the first source code example, I'll go through it with you. I am first giving you the whole source code listing, and after this we'll look at the most important lines in detail. Please note: every source code listing is commented in detail, so you should have no problems following it.

The source code for this demo application is also available in the src folder coming with this documentation:

```
/*
 * Copyright (c) 2011. Philipp Wagner <bytefish[at]gmx[dot]de>.
 * Released to public domain under terms of the BSD Simplified license.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *   * Neither the name of the organization nor the names of its contributors
 *     may be used to endorse or promote products derived from this software
 *     without specific prior written permission.
 *
 *   See <http://www.opensource.org/licenses/bsd-license>
 */

#include "opencv2/core.hpp"
```

```cpp
#include "opencv2/face.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"

#include <iostream>
#include <fstream>
#include <sstream>

using namespace cv;
using namespace cv::face;
using namespace std;

static Mat norm_0_255(InputArray _src) {
    Mat src = _src.getMat();
    // Create and return normalized image:
    Mat dst;
    switch(src.channels()) {
    case 1:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC1);
        break;
    case 3:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC3);
        break;
    default:
        src.copyTo(dst);
        break;
    }
    return dst;
}

static void read_csv(const string& filename, vector<Mat>& images, vector<int>& labels, char separator = ';') {
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file) {
        string error_message = "No valid input file was given, please check the given filename.";
        CV_Error(Error::StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line)) {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if(!path.empty() && !classlabel.empty()) {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
        }
    }
}

int main(int argc, const char *argv[]) {
    // Check for valid command line arguments, print usage
    // if no arguments were given.
    if (argc < 2) {
        cout << "usage: " << argv[0] << " <csv.ext> <output_folder> " << endl;
        exit(1);
    }
    string output_folder = ".";
    if (argc == 3) {
        output_folder = string(argv[2]);
    }
    // Get the path to your CSV.
    string fn_csv = string(argv[1]);
    // These vectors hold the images and corresponding labels.
    vector<Mat> images;
    vector<int> labels;
    // Read in the data. This can fail if no valid
    // input filename is given.
    try {
        read_csv(fn_csv, images, labels);
    } catch (const cv::Exception& e) {
        cerr << "Error opening file \"" << fn_csv << "\". Reason: " << e.msg << endl;
        // nothing more we can do
        exit(1);
    }
    // Quit if there are not enough images for this demo.
    if(images.size() <= 1) {
        string error_message = "This demo needs at least 2 images to work. Please add more images to your data set!";
        CV_Error(Error::StsError, error_message);
    }
    // Get the height from the first image. We'll need this
    // later in code to reshape the images to their original
    // size:
    int height = images[0].rows;
    // The following lines simply get the last images from
```

```cpp
// your dataset and remove it from the vector. This is
// done, so that the training data (which we learn the
// cv::BasicFaceRecognizer on) and the test data we test
// the model with, do not overlap.
Mat testSample = images[images.size() - 1];
int testLabel = labels[labels.size() - 1];
images.pop_back();
labels.pop_back();
// The following lines create an Eigenfaces model for
// face recognition and train it with the images and
// labels read from the given CSV file.
// This here is a full PCA, if you just want to keep
// 10 principal components (read Eigenfaces), then call
// the factory method like this:
//
//      EigenFaceRecognizer::create(10);
//
// If you want to create a FaceRecognizer with a
// confidence threshold (e.g. 123.0), call it with:
//
//      EigenFaceRecognizer::create(10, 123.0);
//
// If you want to use _all_ Eigenfaces and have a threshold,
// then call the method like this:
//
//      EigenFaceRecognizer::create(0, 123.0);
//
Ptr<EigenFaceRecognizer> model = EigenFaceRecognizer::create();
model->train(images, labels);
// The following line predicts the label of a given
// test image:
int predictedLabel = model->predict(testSample);
//
// To get the confidence of a prediction call the model with:
//
//      int predictedLabel = -1;
//      double confidence = 0.0;
//      model->predict(testSample, predictedLabel, confidence);
//
string result_message = format("Predicted class = %d / Actual class = %d.", predictedLabel, testLabel);
cout << result_message << endl;
// Here is how to get the eigenvalues of this Eigenfaces model:
Mat eigenvalues = model->getEigenValues();
// And we can do the same to display the Eigenvectors (read Eigenfaces):
Mat W = model->getEigenVectors();
// Get the sample mean from the training data
Mat mean = model->getMean();
// Display or save:
if(argc == 2) {
    imshow("mean", norm_0_255(mean.reshape(1, images[0].rows)));
} else {
    imwrite(format("%s/mean.png", output_folder.c_str()), norm_0_255(mean.reshape(1, images[0].rows)));
}
// Display or save the Eigenfaces:
for (int i = 0; i < min(10, W.cols); i++) {
    string msg = format("Eigenvalue #%d = %.5f", i, eigenvalues.at<double>(i));
    cout << msg << endl;
    // get eigenvector #i
    Mat ev = W.col(i).clone();
    // Reshape to original size & normalize to [0...255] for imshow.
    Mat grayscale = norm_0_255(ev.reshape(1, height));
    // Show the image & apply a Jet colormap for better sensing.
    Mat cgrayscale;
    applyColorMap(grayscale, cgrayscale, COLORMAP_JET);
    // Display or save:
    if(argc == 2) {
        imshow(format("eigenface_%d", i), cgrayscale);
    } else {
        imwrite(format("%s/eigenface_%d.png", output_folder.c_str(), i), norm_0_255(cgrayscale));
    }
}

// Display or save the image reconstruction at some predefined steps:
for(int num_components = min(W.cols, 10); num_components < min(W.cols, 300); num_components+=15) {
    // slice the eigenvectors from the model
    Mat evs = Mat(W, Range::all(), Range(0, num_components));
    Mat projection = LDA::subspaceProject(evs, mean, images[0].reshape(1,1));
    Mat reconstruction = LDA::subspaceReconstruct(evs, mean, projection);
    // Normalize the result:
    reconstruction = norm_0_255(reconstruction.reshape(1, images[0].rows));
    // Display or save:
    if(argc == 2) {
        imshow(format("eigenface_reconstruction_%d", num_components), reconstruction);
```

```
        } else {
            imwrite(format("%s/eigenface_reconstruction_%d.png", output_folder.c_str(), num_components), reconstruction);
        }
    }
    // Display if we are not writing to an output folder:
    if(argc == 2) {
        waitKey(0);
    }
    return 0;
}
```

I've used the jet colormap, so you can see how the grayscale values are distributed within the specific Eigenfaces. You can see, that the Eigenfaces do not only encode facial features, but also the illumination in the images (see the left light in Eigenface #4, right light in Eigenfaces #5):
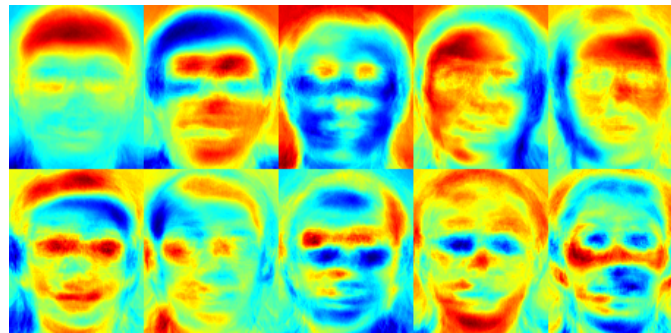
image

We've already seen, that we can reconstruct a face from its lower dimensional approximation. So let's see how many Eigenfaces are needed for a good reconstruction. I'll do a subplot with $10, 30, \ldots, 310$ Eigenfaces:

```
// Display or save the image reconstruction at some predefined steps:
for(int num_components = 10; num_components < 300; num_components+=15) {
    // slice the eigenvectors from the model
    Mat evs = Mat(W, Range::all(), Range(0, num_components));
    Mat projection = LDA::subspaceProject(evs, mean, images[0].reshape(1,1));
    Mat reconstruction = LDA::subspaceReconstruct(evs, mean, projection);
    // Normalize the result:
    reconstruction = norm_0_255(reconstruction.reshape(1, images[0].rows));
    // Display or save:
    if(argc == 2) {
        imshow(format("eigenface_reconstruction_%d", num_components), reconstruction);
    } else {
        imwrite(format("%s/eigenface_reconstruction_%d.png", output_folder.c_str(), num_components), reconstruction);
    }
}
```

10 Eigenvectors are obviously not sufficient for a good image reconstruction, 50 Eigenvectors may already be sufficient to encode important facial features. You'll get a good reconstruction with approximately 300 Eigenvectors for the AT&T Facedatabase. There are rule of thumbs how many Eigenfaces you should choose for a successful face recognition, but it heavily depends on the input data. [296] is the perfect point to start researching for this:

image

Fisherfaces

The Principal Component Analysis (PCA), which is the core of the Eigenfaces method, finds a linear combination of features that maximizes the total variance in data. While this is clearly a powerful way to represent data, it doesn't consider any classes and so a lot of discriminative information *may* be lost when throwing components away. Imagine a situation where the variance in your data is generated by an external source, let it be the light. The components identified by a PCA do not necessarily contain any discriminative information at all, so the projected samples are smeared together and a classification becomes impossible (see http://www.bytefish.de/wiki/pca_lda_with_gnu_octave for an example).

The Linear Discriminant Analysis performs a class-specific dimensionality reduction and was invented by the great statistician Sir R. A. Fisher. He successfully used it for classifying flowers in his 1936 paper *The use of multiple measurements in taxonomic problems* [78] . In order to find the combination of features that separates best between classes the Linear Discriminant Analysis maximizes the ratio of between-classes to within-classes scatter, instead of maximizing the overall scatter. The idea is simple: same classes should cluster tightly together, while different classes are as far away as possible from each other in the lower-dimensional representation. This was also recognized by Belhumeur, Hespanha and Kriegman and so they applied a Discriminant Analysis to face recognition in [18] .

Algorithmic Description of Fisherfaces method

Let X be a random vector with samples drawn from c classes:

$$
\begin{aligned}
X &= \{X_1, X_2, \ldots, X_c\} \\
X_i &= \{x_1, x_2, \ldots, x_n\}
\end{aligned}
$$

The scatter matrices S_B and S_{W} are calculated as:

$$
\begin{aligned}
S_B &= \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T \\
S_W &= \sum_{i=1}^{c} \sum_{x_j \in X_i} (x_j - \mu_i)(x_j - \mu_i)^T
\end{aligned}
$$

, where μ is the total mean:

$$
\mu = \frac{1}{N} \sum_{i=1}^{N} x_i
$$

And μ_i is the mean of class $i \in \{1, \ldots, c\}$:

$$
\mu_i = \frac{1}{|X_i|} \sum_{x_j \in X_i} x_j
$$

Fisher's classic algorithm now looks for a projection W, that maximizes the class separability criterion:

$$
W_{opt} = \arg\max_W \frac{|W^T S_B W|}{|W^T S_W W|}
$$

Following [18], a solution for this optimization problem is given by solving the General Eigenvalue Problem:

$$
\begin{aligned}
S_B v_i &= \lambda_i S_w v_i \\
S_W^{-1} S_B v_i &= \lambda_i v_i
\end{aligned}
$$

There's one problem left to solve: The rank of S_W is at most $(N - c)$, with N samples and c classes. In pattern recognition problems the number of samples N is almost always samller than the dimension of the input data (the number of pixels), so the scatter matrix S_W becomes singular (see [202]). In [18] this was solved by performing a Principal Component Analysis on the data and projecting the samples into the $(N - c)$-dimensional space. A Linear Discriminant Analysis was then performed on the reduced data, because S_W isn't singular anymore.

The optimization problem can then be rewritten as:

$$
\begin{aligned}
W_{pca} &= \arg\max_W |W^T S_T W| \\
W_{fld} &= \arg\max_W \frac{|W^T W_{pca}^T S_B W_{pca} W|}{|W^T W_{pca}^T S_W W_{pca} W|}
\end{aligned}
$$

The transformation matrix W, that projects a sample into the $(c - 1)$-dimensional space is then given by:

$$
W = W_{fld}^T W_{pca}^T
$$

Fisherfaces in OpenCV

The source code for this demo application is also available in the src folder coming with this documentation:

```cpp
/*
 * Copyright (c) 2011. Philipp Wagner <bytefish[at]gmx[dot]de>.
 * Released to public domain under terms of the BSD Simplified license.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *   * Redistributions of source code must retain the above copyright
 *     notice, this list of conditions and the following disclaimer.
 *   * Redistributions in binary form must reproduce the above copyright
 *     notice, this list of conditions and the following disclaimer in the
 *     documentation and/or other materials provided with the distribution.
 *   * Neither the name of the organization nor the names of its contributors
 *     may be used to endorse or promote products derived from this software
 *     without specific prior written permission.
 *
 *   See <http://www.opensource.org/licenses/bsd-license>
 */

#include "opencv2/core.hpp"
#include "opencv2/face.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"

#include <iostream>
#include <fstream>
#include <sstream>

using namespace cv;
using namespace cv::face;
using namespace std;

static Mat norm_0_255(InputArray _src) {
    Mat src = _src.getMat();
    // Create and return normalized image:
    Mat dst;
    switch(src.channels()) {
    case 1:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC1);
        break;
    case 3:
        cv::normalize(_src, dst, 0, 255, NORM_MINMAX, CV_8UC3);
        break;
    default:
        src.copyTo(dst);
        break;
    }
    return dst;
}

static void read_csv(const string& filename, vector<Mat>& images, vector<int>& labels, char separator = ';') {
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file) {
        string error_message = "No valid input file was given, please check the given filename.";
        CV_Error(Error::StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line)) {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if(!path.empty() && !classlabel.empty()) {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
        }
    }
}

int main(int argc, const char *argv[]) {
    // Check for valid command line arguments, print usage
    // if no arguments were given.
    if (argc < 2) {
        cout << "usage: " << argv[0] << " <csv.ext> <output_folder> " << endl;
        exit(1);
    }
    string output_folder = ".";
    if (argc == 3) {
        output_folder = string(argv[2]);
    }
    // Get the path to your CSV.
```

```cpp
    string fn_csv = string(argv[1]);
    // These vectors hold the images and corresponding labels.
    vector<Mat> images;
    vector<int> labels;
    // Read in the data. This can fail if no valid
    // input filename is given.
    try {
        read_csv(fn_csv, images, labels);
    } catch (const cv::Exception& e) {
        cerr << "Error opening file \"" << fn_csv << "\". Reason: " << e.msg << endl;
        // nothing more we can do
        exit(1);
    }
    // Quit if there are not enough images for this demo.
    if(images.size() <= 1) {
        string error_message = "This demo needs at least 2 images to work. Please add more images to your data set!";
        CV_Error(Error::StsError, error_message);
    }
    // Get the height from the first image. We'll need this
    // later in code to reshape the images to their original
    // size:
    int height = images[0].rows;
    // The following lines simply get the last images from
    // your dataset and remove it from the vector. This is
    // done, so that the training data (which we learn the
    // cv::BasicFaceRecognizer on) and the test data we test
    // the model with, do not overlap.
    Mat testSample = images[images.size() - 1];
    int testLabel = labels[labels.size() - 1];
    images.pop_back();
    labels.pop_back();
    // The following lines create an Fisherfaces model for
    // face recognition and train it with the images and
    // labels read from the given CSV file.
    // If you just want to keep 10 Fisherfaces, then call
    // the factory method like this:
    //
    //      FisherFaceRecognizer::create(10);
    //
    // However it is not useful to discard Fisherfaces! Please
    // always try to use _all_ available Fisherfaces for
    // classification.
    //
    // If you want to create a FaceRecognizer with a
    // confidence threshold (e.g. 123.0) and use _all_
    // Fisherfaces, then call it with:
    //
    //      FisherFaceRecognizer::create(0, 123.0);
    //
    Ptr<FisherFaceRecognizer> model = FisherFaceRecognizer::create();
    model->train(images, labels);
    // The following line predicts the label of a given
    // test image:
    int predictedLabel = model->predict(testSample);
    //
    // To get the confidence of a prediction call the model with:
    //
    //      int predictedLabel = -1;
    //      double confidence = 0.0;
    //      model->predict(testSample, predictedLabel, confidence);
    //
    string result_message = format("Predicted class = %d / Actual class = %d.", predictedLabel, testLabel);
    cout << result_message << endl;
    // Here is how to get the eigenvalues of this Eigenfaces model:
    Mat eigenvalues = model->getEigenValues();
    // And we can do the same to display the Eigenvectors (read Eigenfaces):
    Mat W = model->getEigenVectors();
    // Get the sample mean from the training data
    Mat mean = model->getMean();
    // Display or save:
    if(argc == 2) {
        imshow("mean", norm_0_255(mean.reshape(1, images[0].rows)));
    } else {
        imwrite(format("%s/mean.png", output_folder.c_str()), norm_0_255(mean.reshape(1, images[0].rows)));
    }
    // Display or save the first, at most 16 Fisherfaces:
    for (int i = 0; i < min(16, W.cols); i++) {
        string msg = format("Eigenvalue #%d = %.5f", i, eigenvalues.at<double>(i));
        cout << msg << endl;
        // get eigenvector #i
        Mat ev = W.col(i).clone();
        // Reshape to original size & normalize to [0...255] for imshow.
        Mat grayscale = norm_0_255(ev.reshape(1, height));
```

```
          // Show the image & apply a Bone colormap for better sensing.
          Mat cgrayscale;
          applyColorMap(grayscale, cgrayscale, COLORMAP_BONE);
          // Display or save:
          if(argc == 2) {
              imshow(format("fisherface_%d", i), cgrayscale);
          } else {
              imwrite(format("%s/fisherface_%d.png", output_folder.c_str(), i), norm_0_255(cgrayscale));
          }
      }
      // Display or save the image reconstruction at some predefined steps:
      for(int num_component = 0; num_component < min(16, W.cols); num_component++) {
          // Slice the Fisherface from the model:
          Mat ev = W.col(num_component);
          Mat projection = LDA::subspaceProject(ev, mean, images[0].reshape(1,1));
          Mat reconstruction = LDA::subspaceReconstruct(ev, mean, projection);
          // Normalize the result:
          reconstruction = norm_0_255(reconstruction.reshape(1, images[0].rows));
          // Display or save:
          if(argc == 2) {
              imshow(format("fisherface_reconstruction_%d", num_component), reconstruction);
          } else {
              imwrite(format("%s/fisherface_reconstruction_%d.png", output_folder.c_str(), num_component), reconstruction);
          }
      }
      // Display if we are not writing to an output folder:
      if(argc == 2) {
          waitKey(0);
      }
      return 0;
}
```

For this example I am going to use the Yale Facedatabase A, just because the plots are nicer. Each Fisherface has the same length as an original image, thus it can be displayed as an image. The demo shows (or saves) the first, at most 16 Fisherfaces:
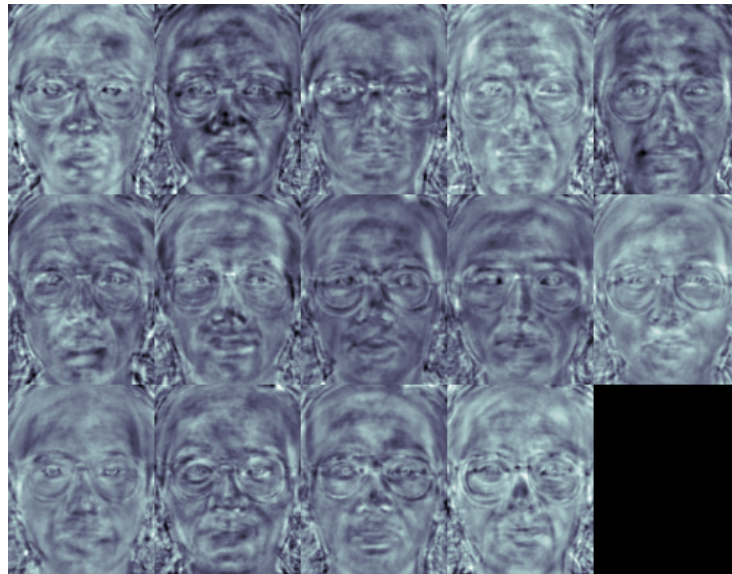
image

The Fisherfaces method learns a class-specific transformation matrix, so the they do not capture illumination as obviously as the Eigenfaces method. The Discriminant Analysis instead finds the facial features to discriminate between the persons. It's important to mention, that the performance of the Fisherfaces heavily depends on the input data as well. Practically said: if you learn the Fisherfaces for well-illuminated pictures only and you try to recognize faces in bad-illuminated scenes, then method is likely to find the wrong components (just because those features may not be predominant on bad illuminated images). This is somewhat logical, since the method had no chance to learn the illumination.

The Fisherfaces allow a reconstruction of the projected image, just like the Eigenfaces did. But since we only identified the features to distinguish between subjects, you can't expect a nice reconstruction of the original image. For the Fisherfaces method we'll project the sample image onto each of the Fisherfaces instead. So you'll have a nice visualization, which feature each of the Fisherfaces describes:

```
// Display or save the image reconstruction at some predefined steps:
for(int num_component = 0; num_component < min(16, W.cols); num_component++) {
    // Slice the Fisherface from the model:
    Mat ev = W.col(num_component);
    Mat projection = LDA::subspaceProject(ev, mean, images[0].reshape(1,1));
    Mat reconstruction = LDA::subspaceReconstruct(ev, mean, projection);
    // Normalize the result:
    reconstruction = norm_0_255(reconstruction.reshape(1, images[0].rows));
```

```
    // Display or save:
    if(argc == 2) {
        imshow(format("fisherface_reconstruction_%d", num_component), reconstruction);
    } else {
        imwrite(format("%s/fisherface_reconstruction_%d.png", output_folder.c_str(), num_component), reconstruction);
    }
}
```

The differences may be subtle for the human eyes, but you should be able to see some differences:

image

Local Binary Patterns Histograms

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:
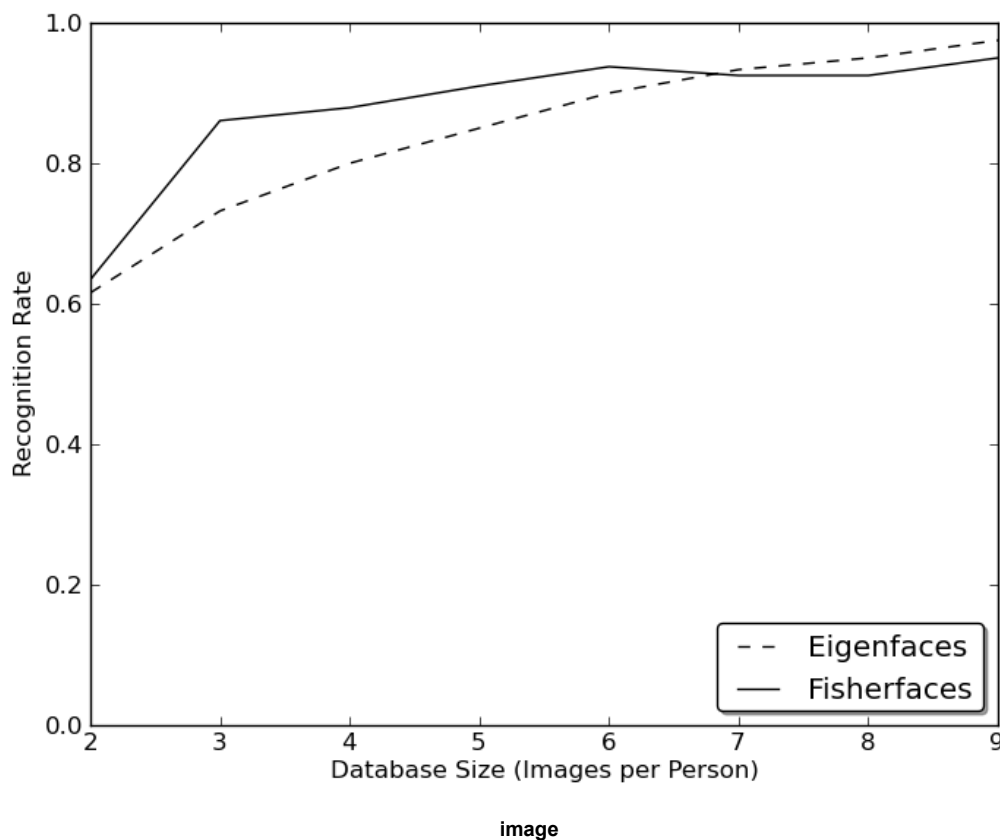
image

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 [165] 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, 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

1. 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:

image
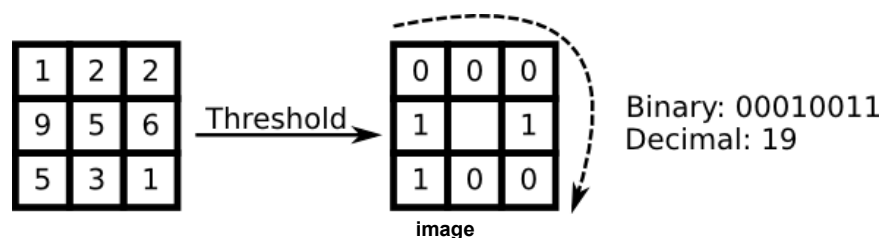
## Algorithmic Description of LBPH method

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:

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

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 [3] . The idea is to align an arbitrary number of neighbors on a circle with a variable radius, which enables to capture the following neighborhoods:

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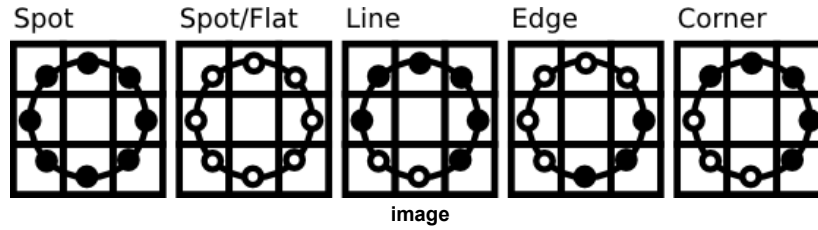
For a given Point (x_c, y_c) the position of the neighbor $(x_p, y_p), p \in P$ can be calculated by:

$$x_p = \quad x_c + R\cos(\frac{2\pi p}{P})$$
$$y_p = \quad y_c - R\sin(\frac{2\pi p}{P})$$

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:

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

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!):
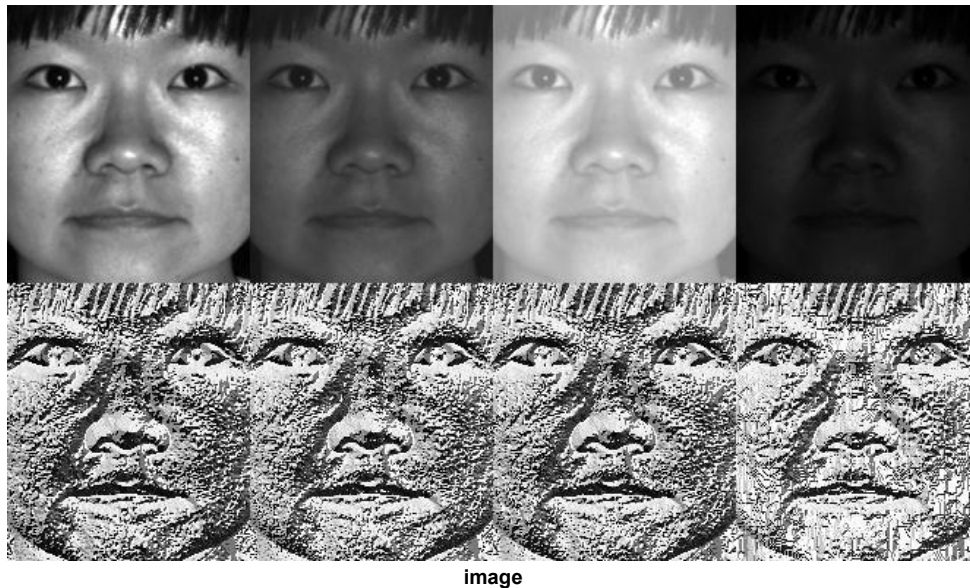
image

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 [3] 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*.

Local Binary Patterns Histograms in OpenCV

The source code for this demo application is also available in the src folder coming with this documentation:

```
/*
 * Copyright (c) 2011. Philipp Wagner <bytefish[at]gmx[dot]de>.
 * Released to public domain under terms of the BSD Simplified license.
 *
 * Redistribution and use in source and binary forms, with or without
```

```
 * modification, are permitted provided that the following conditions are met:
 *    * Redistributions of source code must retain the above copyright
 *      notice, this list of conditions and the following disclaimer.
 *    * Redistributions in binary form must reproduce the above copyright
 *      notice, this list of conditions and the following disclaimer in the
 *      documentation and/or other materials provided with the distribution.
 *    * Neither the name of the organization nor the names of its contributors
 *      may be used to endorse or promote products derived from this software
 *      without specific prior written permission.
 *
 *   See <http://www.opensource.org/licenses/bsd-license>
 */

#include "opencv2/core.hpp"
#include "opencv2/face.hpp"
#include "opencv2/highgui.hpp"

#include <iostream>
#include <fstream>
#include <sstream>

using namespace cv;
using namespace cv::face;
using namespace std;

static void read_csv(const string& filename, vector<Mat>& images, vector<int>& labels, char separator = ';') {
    std::ifstream file(filename.c_str(), ifstream::in);
    if (!file) {
        string error_message = "No valid input file was given, please check the given filename.";
        CV_Error(Error::StsBadArg, error_message);
    }
    string line, path, classlabel;
    while (getline(file, line)) {
        stringstream liness(line);
        getline(liness, path, separator);
        getline(liness, classlabel);
        if(!path.empty() && !classlabel.empty()) {
            images.push_back(imread(path, 0));
            labels.push_back(atoi(classlabel.c_str()));
        }
    }
}

int main(int argc, const char *argv[]) {
    // Check for valid command line arguments, print usage
    // if no arguments were given.
    if (argc != 2) {
        cout << "usage: " << argv[0] << " <csv.ext>" << endl;
        exit(1);
    }
    // Get the path to your CSV.
    string fn_csv = string(argv[1]);
    // These vectors hold the images and corresponding labels.
    vector<Mat> images;
    vector<int> labels;
    // Read in the data. This can fail if no valid
    // input filename is given.
    try {
        read_csv(fn_csv, images, labels);
    } catch (const cv::Exception& e) {
        cerr << "Error opening file \"" << fn_csv << "\". Reason: " << e.msg << endl;
        // nothing more we can do
        exit(1);
    }
    // Quit if there are not enough images for this demo.
    if(images.size() <= 1) {
        string error_message = "This demo needs at least 2 images to work. Please add more images to your data set!";
        CV_Error(Error::StsError, error_message);
    }
    // The following lines simply get the last images from
    // your dataset and remove it from the vector. This is
    // done, so that the training data (which we learn the
    // cv::LBPHFaceRecognizer on) and the test data we test
    // the model with, do not overlap.
    Mat testSample = images[images.size() - 1];
    int testLabel = labels[labels.size() - 1];
    images.pop_back();
    labels.pop_back();
    // The following lines create an LBPH model for
    // face recognition and train it with the images and
    // labels read from the given CSV file.
    //
    // The LBPHFaceRecognizer uses Extended Local Binary Patterns
```

```cpp
    // (it's probably configurable with other operators at a later
    // point), and has the following default values
    //
    //      radius = 1
    //      neighbors = 8
    //      grid_x = 8
    //      grid_y = 8
    //
    // So if you want a LBPH FaceRecognizer using a radius of
    // 2 and 16 neighbors, call the factory method with:
    //
    //      cv::face::LBPHFaceRecognizer::create(2, 16);
    //
    // And if you want a threshold (e.g. 123.0) call it with its default values:
    //
    //      cv::face::LBPHFaceRecognizer::create(1,8,8,8,123.0)
    //
    Ptr<LBPHFaceRecognizer> model = LBPHFaceRecognizer::create();
    model->train(images, labels);
    // The following line predicts the label of a given
    // test image:
    int predictedLabel = model->predict(testSample);
    //
    // To get the confidence of a prediction call the model with:
    //
    //      int predictedLabel = -1;
    //      double confidence = 0.0;
    //      model->predict(testSample, predictedLabel, confidence);
    //
    string result_message = format("Predicted class = %d / Actual class = %d.", predictedLabel, testLabel);
    cout << result_message << endl;
    // First we'll use it to set the threshold of the LBPHFaceRecognizer
    // to 0.0 without retraining the model. This can be useful if
    // you are evaluating the model:
    //
    model->setThreshold(0.0);
    // Now the threshold of this model is set to 0.0. A prediction
    // now returns -1, as it's impossible to have a distance below
    // it
    predictedLabel = model->predict(testSample);
    cout << "Predicted class = " << predictedLabel << endl;
    // Show some informations about the model, as there's no cool
    // Model data to display as in Eigenfaces/Fisherfaces.
    // Due to efficiency reasons the LBP images are not stored
    // within the model:
    cout << "Model Information:" << endl;
    string model_info = format("\tLBPH(radius=%i, neighbors=%i, grid_x=%i, grid_y=%i, threshold=%.2f)",
            model->getRadius(),
            model->getNeighbors(),
            model->getGridX(),
            model->getGridY(),
            model->getThreshold());
    cout << model_info << endl;
    // We could get the histograms for example:
    vector<Mat> histograms = model->getHistograms();
    // But should I really visualize it? Probably the length is interesting:
    cout << "Size of the histograms: " << histograms[0].total() << endl;
    return 0;
}
```

Conclusion

You've learned how to use the new FaceRecognizer in real applications. After reading the document you also know how the algorithms work, so now it's time for you to experiment with the available algorithms. Use them, improve them and let the OpenCV community participate!

Credits

This document wouldn't be possible without the kind permission to use the face images of the *AT&T Database of Faces* and the *Yale Facedatabase A/B*.

The Database of Faces

Important: when using these images, please give credit to "AT&T Laboratories, Cambridge."

The Database of Faces, formerly *The ORL Database of Faces*, contains a set of face images taken between April 1992 and April 1994. The database was used in the context of a face recognition project carried out in collaboration with the Speech, Vision and Robotics Group of the Cambridge University Engineering Department.

There are ten different images of each of 40 distinct subjects. For some subjects, the images were taken at different times, varying the lighting, facial expressions (open / closed eyes, smiling / not smiling) and facial details (glasses / no glasses). All the images were taken against a dark homogeneous

background with the subjects in an upright, frontal position (with tolerance for some side movement).

The files are in PGM format. The size of each image is 92x112 pixels, with 256 grey levels per pixel. The images are organised in 40 directories (one for each subject), which have names of the form sX, where X indicates the subject number (between 1 and 40). In each of these directories, there are ten different images of that subject, which have names of the form Y.pgm, where Y is the image number for that subject (between 1 and 10).

A copy of the database can be retrieved from: http://www.cl.cam.ac.uk/research/dtg/attarchive/pub/data/att_faces.zip.

Yale Facedatabase A

With the permission of the authors I am allowed to show a small number of images (say subject 1 and all the variations) and all images such as Fisherfaces and Eigenfaces from either Yale Facedatabase A or the Yale Facedatabase B.

The Yale Face Database A (size 6.4MB) contains 165 grayscale images in GIF format of 15 individuals. There are 11 images per subject, one per different facial expression or configuration: center-light, w/glasses, happy, left-light, w/no glasses, normal, right-light, sad, sleepy, surprised, and wink. (Source: http://cvc.yale.edu/projects/yalefaces/yalefaces.html)

Yale Facedatabase B

With the permission of the authors I am allowed to show a small number of images (say subject 1 and all the variations) and all images such as Fisherfaces and Eigenfaces from either Yale Facedatabase A or the Yale Facedatabase B.

The extended Yale Face Database B contains 16128 images of 28 human subjects under 9 poses and 64 illumination conditions. The data format of this database is the same as the Yale Face Database B. Please refer to the homepage of the Yale Face Database B (or one copy of this page) for more detailed information of the data format.

You are free to use the extended Yale Face Database B for research purposes. All publications which use this database should acknowledge the use of "the Exteded Yale Face Database B" and reference Athinodoros Georghiades, Peter Belhumeur, and David Kriegman's paper, "From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose", PAMI, 2001, [bibtex].

The extended database as opposed to the original Yale Face Database B with 10 subjects was first reported by Kuang-Chih Lee, Jeffrey Ho, and David Kriegman in "Acquiring Linear Subspaces for Face Recognition under Variable Lighting, PAMI, May, 2005 [pdf]." All test image data used in the experiments are manually aligned, cropped, and then re-sized to 168x192 images. If you publish your experimental results with the cropped images, please reference the PAMI2005 paper as well. (Source: http://vision.ucsd.edu/~leekc/ExtYaleDatabase/ExtYaleB.html)

Appendix

Creating the CSV File

You don't really want to create the CSV file by hand. I have prepared you a little Python script `create_csv.py` (you find it at `src/create_csv.py` coming with this tutorial) that automatically creates you a CSV file. If you have your images in hierarchie like this (`/basepath/<subject>/<image.ext>`):

```
philipp@mango:~/facerec/data/at$ tree
.
|-- s1
|   |-- 1.pgm
|   |-- ...
|   |-- 10.pgm
|-- s2
|   |-- 1.pgm
|   |-- ...
|   |-- 10.pgm
...
|-- s40
|   |-- 1.pgm
|   |-- ...
|   |-- 10.pgm
```

Then simply call `create_csv.py at` , here 'at' being the basepath to the folder, just like this and you could save the output:

```
philipp@mango:~/facerec/data$ python create_csv.py at
at/s13/2.pgm;0
at/s13/7.pgm;0
at/s13/6.pgm;0
at/s13/9.pgm;0
at/s13/5.pgm;0
at/s13/3.pgm;0
at/s13/4.pgm;0
at/s13/10.pgm;0
at/s13/8.pgm;0
at/s13/1.pgm;0
at/s17/2.pgm;1
at/s17/7.pgm;1
```

```
at/s17/6.pgm;1
at/s17/9.pgm;1
at/s17/5.pgm;1
at/s17/3.pgm;1
[...]
```

Here is the script, if you can't find it:

```python
#!/usr/bin/env python

import sys
import os.path

# This is a tiny script to help you creating a CSV file from a face
# database with a similar hierarchie:
#
#  philipp@mango:~/facerec/data/at$ tree
#  .
#  |-- README
#  |-- s1
#  |   |-- 1.pgm
#  |   |-- ...
#  |   |-- 10.pgm
#  |-- s2
#  |   |-- 1.pgm
#  |   |-- ...
#  |   |-- 10.pgm
#  ...
#  |-- s40
#  |   |-- 1.pgm
#  |   |-- ...
#  |   |-- 10.pgm
#

if __name__ == "__main__":

    if len(sys.argv) != 2:
        print "usage: create_csv <base_path>"
        sys.exit(1)

    BASE_PATH=sys.argv[1]
    SEPARATOR=";"

    label = 0
    for dirname, dirnames, filenames in os.walk(BASE_PATH):
        for subdirname in dirnames:
            subject_path = os.path.join(dirname, subdirname)
            for filename in os.listdir(subject_path):
                abs_path = "%s/%s" % (subject_path, filename)
                print "%s%s%d" % (abs_path, SEPARATOR, label)
            label = label + 1
```

Aligning Face Images

An accurate alignment of your image data is especially important in tasks like emotion detection, were you need as much detail as possible. Believe me... You don't want to do this by hand. So I've prepared you a tiny Python script. The code is really easy to use. To scale, rotate and crop the face image you just need to call *CropFace(image, eye_left, eye_right, offset_pct, dest_sz)*, where:

- *eye_left* is the position of the left eye
- *eye_right* is the position of the right eye
- *offset_pct* is the percent of the image you want to keep next to the eyes (horizontal, vertical direction)
- *dest_sz* is the size of the output image

If you are using the same *offset_pct* and *dest_sz* for your images, they are all aligned at the eyes.

```python
#!/usr/bin/env python
# Software License Agreement (BSD License)
#
# Copyright (c) 2012, Philipp Wagner
# All rights reserved.
#
# Redistribution and use in source and binary forms, with or without
```

```
# modification, are permitted provided that the following conditions
# are met:
#
#  * Redistributions of source code must retain the above copyright
#    notice, this list of conditions and the following disclaimer.
#  * Redistributions in binary form must reproduce the above
#    copyright notice, this list of conditions and the following
#    disclaimer in the documentation and/or other materials provided
#    with the distribution.
#  * Neither the name of the author nor the names of its
#    contributors may be used to endorse or promote products derived
#    from this software without specific prior written permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
# "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
# LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
# FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
# COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
# INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
# BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER
# CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
# LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN
# ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
# POSSIBILITY OF SUCH DAMAGE.

import sys, math, Image

def Distance(p1,p2):
  dx = p2[0] - p1[0]
  dy = p2[1] - p1[1]
  return math.sqrt(dx*dx+dy*dy)

def ScaleRotateTranslate(image, angle, center = None, new_center = None, scale = None, resample=Image.BICUBIC):
  if (scale is None) and (center is None):
    return image.rotate(angle=angle, resample=resample)
  nx,ny = x,y = center
  sx=sy=1.0
  if new_center:
    (nx,ny) = new_center
  if scale:
    (sx,sy) = (scale, scale)
  cosine = math.cos(angle)
  sine = math.sin(angle)
  a = cosine/sx
  b = sine/sx
  c = x-nx*a-ny*b
  d = -sine/sy
  e = cosine/sy
  f = y-nx*d-ny*e
  return image.transform(image.size, Image.AFFINE, (a,b,c,d,e,f), resample=resample)

def CropFace(image, eye_left=(0,0), eye_right=(0,0), offset_pct=(0.2,0.2), dest_sz = (70,70)):
  # calculate offsets in original image
  offset_h = math.floor(float(offset_pct[0])*dest_sz[0])
  offset_v = math.floor(float(offset_pct[1])*dest_sz[1])
  # get the direction
  eye_direction = (eye_right[0] - eye_left[0], eye_right[1] - eye_left[1])
  # calc rotation angle in radians
  rotation = -math.atan2(float(eye_direction[1]),float(eye_direction[0]))
  # distance between them
  dist = Distance(eye_left, eye_right)
  # calculate the reference eye-width
  reference = dest_sz[0] - 2.0*offset_h
  # scale factor
  scale = float(dist)/float(reference)
  # rotate original around the left eye
  image = ScaleRotateTranslate(image, center=eye_left, angle=rotation)
  # crop the rotated image
  crop_xy = (eye_left[0] - scale*offset_h, eye_left[1] - scale*offset_v)
  crop_size = (dest_sz[0]*scale, dest_sz[1]*scale)
  image = image.crop((int(crop_xy[0]), int(crop_xy[1]), int(crop_xy[0]+crop_size[0]), int(crop_xy[1]+crop_size[1])))
```

```
   # resize it
   image = image.resize(dest_sz, Image.ANTIALIAS)
   return image

def readFileNames():
    try:
        inFile = open('path_to_created_csv_file.csv')
    except:
        raise IOError('There is no file named path_to_created_csv_file.csv in current directory.')
        return False

    picPath = []
    picIndex = []

    for line in inFile.readlines():
        if line != '':
            fields = line.rstrip().split(';')
            picPath.append(fields[0])
            picIndex.append(int(fields[1]))

    return (picPath, picIndex)


if __name__ == "__main__":
  [images, indexes]=readFileNames()
if not os.path.exists("modified"):
    os.makedirs("modified")
for img in images:
    image =  Image.open(img)
    CropFace(image, eye_left=(252,364), eye_right=(420,366), offset_pct=(0.1,0.1), dest_sz=(200,200)).save("modified/"+
    CropFace(image, eye_left=(252,364), eye_right=(420,366), offset_pct=(0.2,0.2), dest_sz=(200,200)).save("modified/"+
    CropFace(image, eye_left=(252,364), eye_right=(420,366), offset_pct=(0.3,0.3), dest_sz=(200,200)).save("modified/"+
    CropFace(image, eye_left=(252,364), eye_right=(420,366), offset_pct=(0.2,0.2)).save("modified/"+img.rstrip().split(
```

Imagine we are given this photo of Arnold Schwarzenegger, which is under a Public Domain license. The (x,y)-position of the eyes is approximately * (252,364)* for the left and *(420,366)* for the right eye. Now you only need to define the horizontal offset, vertical offset and the size your scaled, rotated & cropped face should have.

Here are some examples:

Configuration	Cropped, Scaled, Rotated Face
0.1 (10%), 0.1 (10%), (200,200)	
0.2 (20%), 0.2 (20%), (200,200)	
0.3 (30%), 0.3 (30%), (200,200)	
0.2 (20%), 0.2 (20%), (70,70)	

CSV for the AT&T Facedatabase

```
/home/philipp/facerec/data/at/s13/2.pgm;12
/home/philipp/facerec/data/at/s13/7.pgm;12
/home/philipp/facerec/data/at/s13/6.pgm;12
/home/philipp/facerec/data/at/s13/9.pgm;12
/home/philipp/facerec/data/at/s13/5.pgm;12
/home/philipp/facerec/data/at/s13/3.pgm;12
/home/philipp/facerec/data/at/s13/4.pgm;12
/home/philipp/facerec/data/at/s13/10.pgm;12
/home/philipp/facerec/data/at/s13/8.pgm;12
/home/philipp/facerec/data/at/s13/1.pgm;12
/home/philipp/facerec/data/at/s17/2.pgm;16
/home/philipp/facerec/data/at/s17/7.pgm;16
/home/philipp/facerec/data/at/s17/6.pgm;16
/home/philipp/facerec/data/at/s17/9.pgm;16
/home/philipp/facerec/data/at/s17/5.pgm;16
/home/philipp/facerec/data/at/s17/3.pgm;16
/home/philipp/facerec/data/at/s17/4.pgm;16
/home/philipp/facerec/data/at/s17/10.pgm;16
/home/philipp/facerec/data/at/s17/8.pgm;16
/home/philipp/facerec/data/at/s17/1.pgm;16
```

```
/home/philipp/facerec/data/at/s32/2.pgm;31
/home/philipp/facerec/data/at/s32/7.pgm;31
/home/philipp/facerec/data/at/s32/6.pgm;31
/home/philipp/facerec/data/at/s32/9.pgm;31
/home/philipp/facerec/data/at/s32/5.pgm;31
/home/philipp/facerec/data/at/s32/3.pgm;31
/home/philipp/facerec/data/at/s32/4.pgm;31
/home/philipp/facerec/data/at/s32/10.pgm;31
/home/philipp/facerec/data/at/s32/8.pgm;31
/home/philipp/facerec/data/at/s32/1.pgm;31
/home/philipp/facerec/data/at/s10/2.pgm;9
/home/philipp/facerec/data/at/s10/7.pgm;9
/home/philipp/facerec/data/at/s10/6.pgm;9
/home/philipp/facerec/data/at/s10/9.pgm;9
/home/philipp/facerec/data/at/s10/5.pgm;9
/home/philipp/facerec/data/at/s10/3.pgm;9
/home/philipp/facerec/data/at/s10/4.pgm;9
/home/philipp/facerec/data/at/s10/10.pgm;9
/home/philipp/facerec/data/at/s10/8.pgm;9
/home/philipp/facerec/data/at/s10/1.pgm;9
/home/philipp/facerec/data/at/s27/2.pgm;26
/home/philipp/facerec/data/at/s27/7.pgm;26
/home/philipp/facerec/data/at/s27/6.pgm;26
/home/philipp/facerec/data/at/s27/9.pgm;26
/home/philipp/facerec/data/at/s27/5.pgm;26
/home/philipp/facerec/data/at/s27/3.pgm;26
/home/philipp/facerec/data/at/s27/4.pgm;26
/home/philipp/facerec/data/at/s27/10.pgm;26
/home/philipp/facerec/data/at/s27/8.pgm;26
/home/philipp/facerec/data/at/s27/1.pgm;26
/home/philipp/facerec/data/at/s5/2.pgm;4
/home/philipp/facerec/data/at/s5/7.pgm;4
/home/philipp/facerec/data/at/s5/6.pgm;4
/home/philipp/facerec/data/at/s5/9.pgm;4
/home/philipp/facerec/data/at/s5/5.pgm;4
/home/philipp/facerec/data/at/s5/3.pgm;4
/home/philipp/facerec/data/at/s5/4.pgm;4
/home/philipp/facerec/data/at/s5/10.pgm;4
/home/philipp/facerec/data/at/s5/8.pgm;4
/home/philipp/facerec/data/at/s5/1.pgm;4
/home/philipp/facerec/data/at/s20/2.pgm;19
/home/philipp/facerec/data/at/s20/7.pgm;19
/home/philipp/facerec/data/at/s20/6.pgm;19
/home/philipp/facerec/data/at/s20/9.pgm;19
/home/philipp/facerec/data/at/s20/5.pgm;19
/home/philipp/facerec/data/at/s20/3.pgm;19
/home/philipp/facerec/data/at/s20/4.pgm;19
/home/philipp/facerec/data/at/s20/10.pgm;19
/home/philipp/facerec/data/at/s20/8.pgm;19
/home/philipp/facerec/data/at/s20/1.pgm;19
/home/philipp/facerec/data/at/s30/2.pgm;29
/home/philipp/facerec/data/at/s30/7.pgm;29
/home/philipp/facerec/data/at/s30/6.pgm;29
/home/philipp/facerec/data/at/s30/9.pgm;29
/home/philipp/facerec/data/at/s30/5.pgm;29
/home/philipp/facerec/data/at/s30/3.pgm;29
/home/philipp/facerec/data/at/s30/4.pgm;29
/home/philipp/facerec/data/at/s30/10.pgm;29
/home/philipp/facerec/data/at/s30/8.pgm;29
/home/philipp/facerec/data/at/s30/1.pgm;29
/home/philipp/facerec/data/at/s39/2.pgm;38
/home/philipp/facerec/data/at/s39/7.pgm;38
/home/philipp/facerec/data/at/s39/6.pgm;38
/home/philipp/facerec/data/at/s39/9.pgm;38
/home/philipp/facerec/data/at/s39/5.pgm;38
/home/philipp/facerec/data/at/s39/3.pgm;38
/home/philipp/facerec/data/at/s39/4.pgm;38
/home/philipp/facerec/data/at/s39/10.pgm;38
/home/philipp/facerec/data/at/s39/8.pgm;38
/home/philipp/facerec/data/at/s39/1.pgm;38
/home/philipp/facerec/data/at/s35/2.pgm;34
/home/philipp/facerec/data/at/s35/7.pgm;34
```

```
/home/philipp/facerec/data/at/s35/6.pgm;34
/home/philipp/facerec/data/at/s35/9.pgm;34
/home/philipp/facerec/data/at/s35/5.pgm;34
/home/philipp/facerec/data/at/s35/3.pgm;34
/home/philipp/facerec/data/at/s35/4.pgm;34
/home/philipp/facerec/data/at/s35/10.pgm;34
/home/philipp/facerec/data/at/s35/8.pgm;34
/home/philipp/facerec/data/at/s35/1.pgm;34
/home/philipp/facerec/data/at/s23/2.pgm;22
/home/philipp/facerec/data/at/s23/7.pgm;22
/home/philipp/facerec/data/at/s23/6.pgm;22
/home/philipp/facerec/data/at/s23/9.pgm;22
/home/philipp/facerec/data/at/s23/5.pgm;22
/home/philipp/facerec/data/at/s23/3.pgm;22
/home/philipp/facerec/data/at/s23/4.pgm;22
/home/philipp/facerec/data/at/s23/10.pgm;22
/home/philipp/facerec/data/at/s23/8.pgm;22
/home/philipp/facerec/data/at/s23/1.pgm;22
/home/philipp/facerec/data/at/s4/2.pgm;3
/home/philipp/facerec/data/at/s4/7.pgm;3
/home/philipp/facerec/data/at/s4/6.pgm;3
/home/philipp/facerec/data/at/s4/9.pgm;3
/home/philipp/facerec/data/at/s4/5.pgm;3
/home/philipp/facerec/data/at/s4/3.pgm;3
/home/philipp/facerec/data/at/s4/4.pgm;3
/home/philipp/facerec/data/at/s4/10.pgm;3
/home/philipp/facerec/data/at/s4/8.pgm;3
/home/philipp/facerec/data/at/s4/1.pgm;3
/home/philipp/facerec/data/at/s9/2.pgm;8
/home/philipp/facerec/data/at/s9/7.pgm;8
/home/philipp/facerec/data/at/s9/6.pgm;8
/home/philipp/facerec/data/at/s9/9.pgm;8
/home/philipp/facerec/data/at/s9/5.pgm;8
/home/philipp/facerec/data/at/s9/3.pgm;8
/home/philipp/facerec/data/at/s9/4.pgm;8
/home/philipp/facerec/data/at/s9/10.pgm;8
/home/philipp/facerec/data/at/s9/8.pgm;8
/home/philipp/facerec/data/at/s9/1.pgm;8
/home/philipp/facerec/data/at/s37/2.pgm;36
/home/philipp/facerec/data/at/s37/7.pgm;36
/home/philipp/facerec/data/at/s37/6.pgm;36
/home/philipp/facerec/data/at/s37/9.pgm;36
/home/philipp/facerec/data/at/s37/5.pgm;36
/home/philipp/facerec/data/at/s37/3.pgm;36
/home/philipp/facerec/data/at/s37/4.pgm;36
/home/philipp/facerec/data/at/s37/10.pgm;36
/home/philipp/facerec/data/at/s37/8.pgm;36
/home/philipp/facerec/data/at/s37/1.pgm;36
/home/philipp/facerec/data/at/s24/2.pgm;23
/home/philipp/facerec/data/at/s24/7.pgm;23
/home/philipp/facerec/data/at/s24/6.pgm;23
/home/philipp/facerec/data/at/s24/9.pgm;23
/home/philipp/facerec/data/at/s24/5.pgm;23
/home/philipp/facerec/data/at/s24/3.pgm;23
/home/philipp/facerec/data/at/s24/4.pgm;23
/home/philipp/facerec/data/at/s24/10.pgm;23
/home/philipp/facerec/data/at/s24/8.pgm;23
/home/philipp/facerec/data/at/s24/1.pgm;23
/home/philipp/facerec/data/at/s19/2.pgm;18
/home/philipp/facerec/data/at/s19/7.pgm;18
/home/philipp/facerec/data/at/s19/6.pgm;18
/home/philipp/facerec/data/at/s19/9.pgm;18
/home/philipp/facerec/data/at/s19/5.pgm;18
/home/philipp/facerec/data/at/s19/3.pgm;18
/home/philipp/facerec/data/at/s19/4.pgm;18
/home/philipp/facerec/data/at/s19/10.pgm;18
/home/philipp/facerec/data/at/s19/8.pgm;18
/home/philipp/facerec/data/at/s19/1.pgm;18
/home/philipp/facerec/data/at/s8/2.pgm;7
/home/philipp/facerec/data/at/s8/7.pgm;7
/home/philipp/facerec/data/at/s8/6.pgm;7
/home/philipp/facerec/data/at/s8/9.pgm;7
```

```
/home/philipp/facerec/data/at/s8/5.pgm;7
/home/philipp/facerec/data/at/s8/3.pgm;7
/home/philipp/facerec/data/at/s8/4.pgm;7
/home/philipp/facerec/data/at/s8/10.pgm;7
/home/philipp/facerec/data/at/s8/8.pgm;7
/home/philipp/facerec/data/at/s8/1.pgm;7
/home/philipp/facerec/data/at/s21/2.pgm;20
/home/philipp/facerec/data/at/s21/7.pgm;20
/home/philipp/facerec/data/at/s21/6.pgm;20
/home/philipp/facerec/data/at/s21/9.pgm;20
/home/philipp/facerec/data/at/s21/5.pgm;20
/home/philipp/facerec/data/at/s21/3.pgm;20
/home/philipp/facerec/data/at/s21/4.pgm;20
/home/philipp/facerec/data/at/s21/10.pgm;20
/home/philipp/facerec/data/at/s21/8.pgm;20
/home/philipp/facerec/data/at/s21/1.pgm;20
/home/philipp/facerec/data/at/s1/2.pgm;0
/home/philipp/facerec/data/at/s1/7.pgm;0
/home/philipp/facerec/data/at/s1/6.pgm;0
/home/philipp/facerec/data/at/s1/9.pgm;0
/home/philipp/facerec/data/at/s1/5.pgm;0
/home/philipp/facerec/data/at/s1/3.pgm;0
/home/philipp/facerec/data/at/s1/4.pgm;0
/home/philipp/facerec/data/at/s1/10.pgm;0
/home/philipp/facerec/data/at/s1/8.pgm;0
/home/philipp/facerec/data/at/s1/1.pgm;0
/home/philipp/facerec/data/at/s7/2.pgm;6
/home/philipp/facerec/data/at/s7/7.pgm;6
/home/philipp/facerec/data/at/s7/6.pgm;6
/home/philipp/facerec/data/at/s7/9.pgm;6
/home/philipp/facerec/data/at/s7/5.pgm;6
/home/philipp/facerec/data/at/s7/3.pgm;6
/home/philipp/facerec/data/at/s7/4.pgm;6
/home/philipp/facerec/data/at/s7/10.pgm;6
/home/philipp/facerec/data/at/s7/8.pgm;6
/home/philipp/facerec/data/at/s7/1.pgm;6
/home/philipp/facerec/data/at/s16/2.pgm;15
/home/philipp/facerec/data/at/s16/7.pgm;15
/home/philipp/facerec/data/at/s16/6.pgm;15
/home/philipp/facerec/data/at/s16/9.pgm;15
/home/philipp/facerec/data/at/s16/5.pgm;15
/home/philipp/facerec/data/at/s16/3.pgm;15
/home/philipp/facerec/data/at/s16/4.pgm;15
/home/philipp/facerec/data/at/s16/10.pgm;15
/home/philipp/facerec/data/at/s16/8.pgm;15
/home/philipp/facerec/data/at/s16/1.pgm;15
/home/philipp/facerec/data/at/s36/2.pgm;35
/home/philipp/facerec/data/at/s36/7.pgm;35
/home/philipp/facerec/data/at/s36/6.pgm;35
/home/philipp/facerec/data/at/s36/9.pgm;35
/home/philipp/facerec/data/at/s36/5.pgm;35
/home/philipp/facerec/data/at/s36/3.pgm;35
/home/philipp/facerec/data/at/s36/4.pgm;35
/home/philipp/facerec/data/at/s36/10.pgm;35
/home/philipp/facerec/data/at/s36/8.pgm;35
/home/philipp/facerec/data/at/s36/1.pgm;35
/home/philipp/facerec/data/at/s25/2.pgm;24
/home/philipp/facerec/data/at/s25/7.pgm;24
/home/philipp/facerec/data/at/s25/6.pgm;24
/home/philipp/facerec/data/at/s25/9.pgm;24
/home/philipp/facerec/data/at/s25/5.pgm;24
/home/philipp/facerec/data/at/s25/3.pgm;24
/home/philipp/facerec/data/at/s25/4.pgm;24
/home/philipp/facerec/data/at/s25/10.pgm;24
/home/philipp/facerec/data/at/s25/8.pgm;24
/home/philipp/facerec/data/at/s25/1.pgm;24
/home/philipp/facerec/data/at/s14/2.pgm;13
/home/philipp/facerec/data/at/s14/7.pgm;13
/home/philipp/facerec/data/at/s14/6.pgm;13
/home/philipp/facerec/data/at/s14/9.pgm;13
/home/philipp/facerec/data/at/s14/5.pgm;13
/home/philipp/facerec/data/at/s14/3.pgm;13
```

```
/home/philipp/facerec/data/at/s14/4.pgm;13
/home/philipp/facerec/data/at/s14/10.pgm;13
/home/philipp/facerec/data/at/s14/8.pgm;13
/home/philipp/facerec/data/at/s14/1.pgm;13
/home/philipp/facerec/data/at/s34/2.pgm;33
/home/philipp/facerec/data/at/s34/7.pgm;33
/home/philipp/facerec/data/at/s34/6.pgm;33
/home/philipp/facerec/data/at/s34/9.pgm;33
/home/philipp/facerec/data/at/s34/5.pgm;33
/home/philipp/facerec/data/at/s34/3.pgm;33
/home/philipp/facerec/data/at/s34/4.pgm;33
/home/philipp/facerec/data/at/s34/10.pgm;33
/home/philipp/facerec/data/at/s34/8.pgm;33
/home/philipp/facerec/data/at/s34/1.pgm;33
/home/philipp/facerec/data/at/s11/2.pgm;10
/home/philipp/facerec/data/at/s11/7.pgm;10
/home/philipp/facerec/data/at/s11/6.pgm;10
/home/philipp/facerec/data/at/s11/9.pgm;10
/home/philipp/facerec/data/at/s11/5.pgm;10
/home/philipp/facerec/data/at/s11/3.pgm;10
/home/philipp/facerec/data/at/s11/4.pgm;10
/home/philipp/facerec/data/at/s11/10.pgm;10
/home/philipp/facerec/data/at/s11/8.pgm;10
/home/philipp/facerec/data/at/s11/1.pgm;10
/home/philipp/facerec/data/at/s26/2.pgm;25
/home/philipp/facerec/data/at/s26/7.pgm;25
/home/philipp/facerec/data/at/s26/6.pgm;25
/home/philipp/facerec/data/at/s26/9.pgm;25
/home/philipp/facerec/data/at/s26/5.pgm;25
/home/philipp/facerec/data/at/s26/3.pgm;25
/home/philipp/facerec/data/at/s26/4.pgm;25
/home/philipp/facerec/data/at/s26/10.pgm;25
/home/philipp/facerec/data/at/s26/8.pgm;25
/home/philipp/facerec/data/at/s26/1.pgm;25
/home/philipp/facerec/data/at/s18/2.pgm;17
/home/philipp/facerec/data/at/s18/7.pgm;17
/home/philipp/facerec/data/at/s18/6.pgm;17
/home/philipp/facerec/data/at/s18/9.pgm;17
/home/philipp/facerec/data/at/s18/5.pgm;17
/home/philipp/facerec/data/at/s18/3.pgm;17
/home/philipp/facerec/data/at/s18/4.pgm;17
/home/philipp/facerec/data/at/s18/10.pgm;17
/home/philipp/facerec/data/at/s18/8.pgm;17
/home/philipp/facerec/data/at/s18/1.pgm;17
/home/philipp/facerec/data/at/s29/2.pgm;28
/home/philipp/facerec/data/at/s29/7.pgm;28
/home/philipp/facerec/data/at/s29/6.pgm;28
/home/philipp/facerec/data/at/s29/9.pgm;28
/home/philipp/facerec/data/at/s29/5.pgm;28
/home/philipp/facerec/data/at/s29/3.pgm;28
/home/philipp/facerec/data/at/s29/4.pgm;28
/home/philipp/facerec/data/at/s29/10.pgm;28
/home/philipp/facerec/data/at/s29/8.pgm;28
/home/philipp/facerec/data/at/s29/1.pgm;28
/home/philipp/facerec/data/at/s33/2.pgm;32
/home/philipp/facerec/data/at/s33/7.pgm;32
/home/philipp/facerec/data/at/s33/6.pgm;32
/home/philipp/facerec/data/at/s33/9.pgm;32
/home/philipp/facerec/data/at/s33/5.pgm;32
/home/philipp/facerec/data/at/s33/3.pgm;32
/home/philipp/facerec/data/at/s33/4.pgm;32
/home/philipp/facerec/data/at/s33/10.pgm;32
/home/philipp/facerec/data/at/s33/8.pgm;32
/home/philipp/facerec/data/at/s33/1.pgm;32
/home/philipp/facerec/data/at/s12/2.pgm;11
/home/philipp/facerec/data/at/s12/7.pgm;11
/home/philipp/facerec/data/at/s12/6.pgm;11
/home/philipp/facerec/data/at/s12/9.pgm;11
/home/philipp/facerec/data/at/s12/5.pgm;11
/home/philipp/facerec/data/at/s12/3.pgm;11
/home/philipp/facerec/data/at/s12/4.pgm;11
/home/philipp/facerec/data/at/s12/10.pgm;11
```

```
/home/philipp/facerec/data/at/s12/8.pgm;11
/home/philipp/facerec/data/at/s12/1.pgm;11
/home/philipp/facerec/data/at/s6/2.pgm;5
/home/philipp/facerec/data/at/s6/7.pgm;5
/home/philipp/facerec/data/at/s6/6.pgm;5
/home/philipp/facerec/data/at/s6/9.pgm;5
/home/philipp/facerec/data/at/s6/5.pgm;5
/home/philipp/facerec/data/at/s6/3.pgm;5
/home/philipp/facerec/data/at/s6/4.pgm;5
/home/philipp/facerec/data/at/s6/10.pgm;5
/home/philipp/facerec/data/at/s6/8.pgm;5
/home/philipp/facerec/data/at/s6/1.pgm;5
/home/philipp/facerec/data/at/s22/2.pgm;21
/home/philipp/facerec/data/at/s22/7.pgm;21
/home/philipp/facerec/data/at/s22/6.pgm;21
/home/philipp/facerec/data/at/s22/9.pgm;21
/home/philipp/facerec/data/at/s22/5.pgm;21
/home/philipp/facerec/data/at/s22/3.pgm;21
/home/philipp/facerec/data/at/s22/4.pgm;21
/home/philipp/facerec/data/at/s22/10.pgm;21
/home/philipp/facerec/data/at/s22/8.pgm;21
/home/philipp/facerec/data/at/s22/1.pgm;21
/home/philipp/facerec/data/at/s15/2.pgm;14
/home/philipp/facerec/data/at/s15/7.pgm;14
/home/philipp/facerec/data/at/s15/6.pgm;14
/home/philipp/facerec/data/at/s15/9.pgm;14
/home/philipp/facerec/data/at/s15/5.pgm;14
/home/philipp/facerec/data/at/s15/3.pgm;14
/home/philipp/facerec/data/at/s15/4.pgm;14
/home/philipp/facerec/data/at/s15/10.pgm;14
/home/philipp/facerec/data/at/s15/8.pgm;14
/home/philipp/facerec/data/at/s15/1.pgm;14
/home/philipp/facerec/data/at/s2/2.pgm;1
/home/philipp/facerec/data/at/s2/7.pgm;1
/home/philipp/facerec/data/at/s2/6.pgm;1
/home/philipp/facerec/data/at/s2/9.pgm;1
/home/philipp/facerec/data/at/s2/5.pgm;1
/home/philipp/facerec/data/at/s2/3.pgm;1
/home/philipp/facerec/data/at/s2/4.pgm;1
/home/philipp/facerec/data/at/s2/10.pgm;1
/home/philipp/facerec/data/at/s2/8.pgm;1
/home/philipp/facerec/data/at/s2/1.pgm;1
/home/philipp/facerec/data/at/s31/2.pgm;30
/home/philipp/facerec/data/at/s31/7.pgm;30
/home/philipp/facerec/data/at/s31/6.pgm;30
/home/philipp/facerec/data/at/s31/9.pgm;30
/home/philipp/facerec/data/at/s31/5.pgm;30
/home/philipp/facerec/data/at/s31/3.pgm;30
/home/philipp/facerec/data/at/s31/4.pgm;30
/home/philipp/facerec/data/at/s31/10.pgm;30
/home/philipp/facerec/data/at/s31/8.pgm;30
/home/philipp/facerec/data/at/s31/1.pgm;30
/home/philipp/facerec/data/at/s28/2.pgm;27
/home/philipp/facerec/data/at/s28/7.pgm;27
/home/philipp/facerec/data/at/s28/6.pgm;27
/home/philipp/facerec/data/at/s28/9.pgm;27
/home/philipp/facerec/data/at/s28/5.pgm;27
/home/philipp/facerec/data/at/s28/3.pgm;27
/home/philipp/facerec/data/at/s28/4.pgm;27
/home/philipp/facerec/data/at/s28/10.pgm;27
/home/philipp/facerec/data/at/s28/8.pgm;27
/home/philipp/facerec/data/at/s28/1.pgm;27
/home/philipp/facerec/data/at/s40/2.pgm;39
/home/philipp/facerec/data/at/s40/7.pgm;39
/home/philipp/facerec/data/at/s40/6.pgm;39
/home/philipp/facerec/data/at/s40/9.pgm;39
/home/philipp/facerec/data/at/s40/5.pgm;39
/home/philipp/facerec/data/at/s40/3.pgm;39
/home/philipp/facerec/data/at/s40/4.pgm;39
/home/philipp/facerec/data/at/s40/10.pgm;39
/home/philipp/facerec/data/at/s40/8.pgm;39
/home/philipp/facerec/data/at/s40/1.pgm;39
```

```
/home/philipp/facerec/data/at/s3/2.pgm;2
/home/philipp/facerec/data/at/s3/7.pgm;2
/home/philipp/facerec/data/at/s3/6.pgm;2
/home/philipp/facerec/data/at/s3/9.pgm;2
/home/philipp/facerec/data/at/s3/5.pgm;2
/home/philipp/facerec/data/at/s3/3.pgm;2
/home/philipp/facerec/data/at/s3/4.pgm;2
/home/philipp/facerec/data/at/s3/10.pgm;2
/home/philipp/facerec/data/at/s3/8.pgm;2
/home/philipp/facerec/data/at/s3/1.pgm;2
/home/philipp/facerec/data/at/s38/2.pgm;37
/home/philipp/facerec/data/at/s38/7.pgm;37
/home/philipp/facerec/data/at/s38/6.pgm;37
/home/philipp/facerec/data/at/s38/9.pgm;37
/home/philipp/facerec/data/at/s38/5.pgm;37
/home/philipp/facerec/data/at/s38/3.pgm;37
/home/philipp/facerec/data/at/s38/4.pgm;37
/home/philipp/facerec/data/at/s38/10.pgm;37
/home/philipp/facerec/data/at/s38/8.pgm;37
/home/philipp/facerec/data/at/s38/1.pgm;37
```