2D Face Recognition Using PCA, ICA and LDA

Shireen Elhabian and Aly Farag

October 2009

Contents

Face Recognition Problem

The general statement of the face recognition problem can be stated as follows: Given a still or video image of a scene, identify or verify one or more persons in the scene using a stored database of faces. The solution to the problem involves face detection (a field of research in itself) from cluttered scenes, feature extraction from the face region, recognition or verification. There is a subtle difference between the concepts of face identification and verification: identification refers to the problem when an unknown face is presented to the system, and it is expected to report back the identity of the individual from a database of faces, whereas in verification, there is a claimed identity submitted to the system, which needs to be confirmed or rejected. Figure 1 illustrates a typical face recognition procedure.

Before the face recognition system can be used, there is an *enrollment* phase, wherein face images are introduced to the system to let it *learn* the distinguishing features of each face. The identifying names, together with the discriminating features, are stored in a database, and the images associated with the names are referred to as the *gallery* [6]. Eventually, the system will have to identify an image, formally known as the *probe* [6], against the database of gallery images using distinguishing features. The best match, usually in terms of distance, is returned as the *identity* of the probe.

The success of face identification depends heavily on the choice of discriminating features (Figure 1), which is basically the focus of face recognition research. Face recognition algorithms using still images that extract distinguishing features can be categorized into three groups: *appearance-based*, *feature-based*, and *hybrid* methods. *Appearance-based* methods are usually associated with holistic techniques that use the whole face region as the input to the recognition system. In *feature-based* methods, local features such as the eyes, nose, and mouth are first extracted and their locations and local statistics (geometric or appearance) are fed into a structural classifier. The earliest approaches to the face recognition dealt with the geometrical features of the face to come up with a unique signature of the face. The geometric feature extraction approach fails when the head is no longer viewed directly from the front and the targeted features are impossible to measure. The last category (*hybrid*) has its origin in the human face perception system that combines both holistic and feature-based techniques to identify the face. Whatever type of computer algorithm is applied to the recognition problem, all face the issue of intra-subject and inter-subject variations. Figure 2 demonstrates the meaning of intra-subject and inter-subject variations.

The main problem in face recognition is that the human face has potentially very large intra-subject variations while the inter-subject variation, which is crucial to the success of face identification, is small, as shown in Figure 2. Intra-subject variation is usually due to 3D head pose, illumination, facial expression, occlusion due to other objects, facial hair and aging.

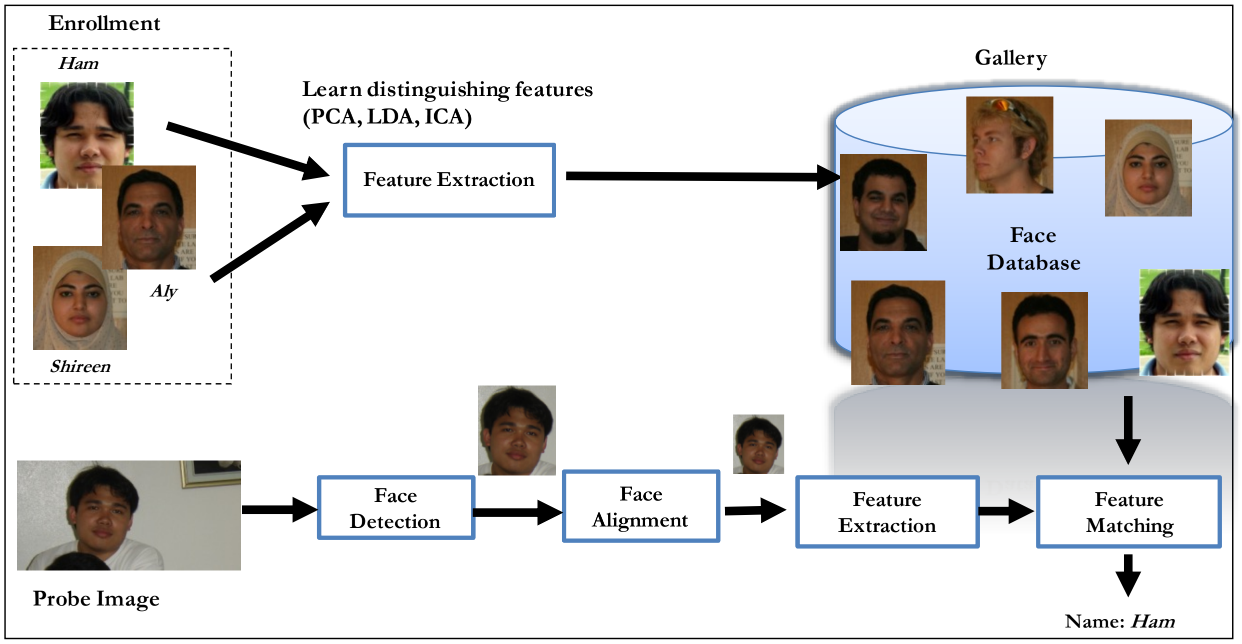

Figure 1: Face Recognition Process, courtesy of [5], the general block diagram of a face recognition system consists of four processes; the face is first detected (extracted) from the given 2D then the extracted face is aligned (by size normalization), discriminant features are then extracted in order to be matched with users enrolled in the system database, the output of the system is the face ID of the given person's image.

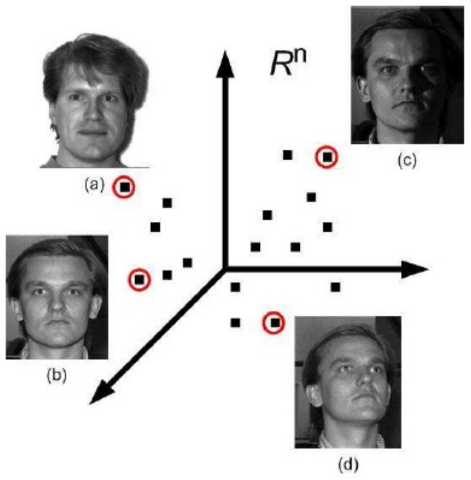

Figure 2: Inter-subject versus intra-subject variations. (a) and (b) are images from different subjects, but their appearance variations represented in the input space can be smaller than images from the same subject b, c and d [6].

Database of Faces

The Yale Face Database [1] consists of 165 grayscale images from 15 individuals. There are 11 images per person, with one image per face expression or configuration: *center-light*, *w/glasses*, *happy*, *left-light*, *w/no glasses*, *normal*, *right-light*, *sad*, *sleepy*, *surprised*, and *wink*.

The Yale database simulates the *inter-subject* vs. *intra-subject* problem in face recognition and will be used in this project. The database can be downloaded from http://cvc.yale.edu/projects/yalefaces/yalefaces.html. (Note: Use the *Mozilla* browser to download. The tar file (*yalefaces.tar*) can be extracted using WinRAR.)

Task 0: Download the face databases.

For the Yale database, the resulting files after extraction have file extensions corresponding to face expressions (e.g. *subject01.centerlight*) but are actually GIF files. Convert the images to JPEG and then arrange them according to the following rules:

- *subject01* images must be under the folder *s1*, *subject02* under *s2*, and so on …
- For each subject, rename **.centerlight* to *1.jpg*, *.glasses to *2.jpg*, and so on …

Task 1: Convert the images to JPEG, rename, and put them under specified folders (see Figure 3).

```
i = 1;
f = filesep; % '\'

dirName = ['s', num2str(i)];
mkdir(dirName) % create directory

% *.centerlight
subjectName = ['subject0',num2str(i),'.centerlight'];
im = imread(subjectName,'gif');
figure, imshow(im)
imwrite(im, [dirName,f,'1.jpg'], 'jpg')

% *.glasses
subjectName = ['subject0',num2str(i),'.glasses'];
im = imread(subjectName,'gif');
figure, imshow(im)
imwrite(im, [dirName,f,'2.jpg'], 'jpg')
```

Figure 3: Code snippet for creating new folders, renaming files, etc.

Face Detection

The images in the face database, unfortunately, contain both the face and a large white background (Figure 4). Only the face region is needed for face recognition and a background can affect the recognition process. Therefore, a *face detection step* is necessary.

Figure 4: Uncropped images of the Yale face database.

A face detection module is provided by Intel OpenCV [2]. Intel OpenCV can be readily downloaded (http://sourceforge.net/project/showfiles.php?group_id=22870). Download OpenCV (exe file) and install it on your PC. In order to use this library within Matlab framework, you will need to download *Open CV Viola-Jones Face Detection in Matlab* from Matlab Central (http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=19912&objectType=file).

This zip file contains source code and windows executables for carrying out face detection on a gray scale image. The code implements Viola-Jones adaboosted algorithm for face detection by providing a mex implementation of OpenCV's face detector to be used in Matlab. Instructions for use and for compiling can be found in the Readme file.

To use the Face detection program you need to set path in matlab to the bin directory of the downloaded zip file. "FaceDetect.dll" is used by versions earlier than 7.1 while "FaceDetect.mexw32" is used by later versions. The two files "cv100.dll" and "cxcore.dll" should be placed in the same directory as the other files.

Matlab 7.0.0 R14 or Matlab 7.5.0 R2007b and Microsoft visual studio 2003 or 2005 are required for compilation.

Instructions for compiling:

- Setup Mex compiler: Type "mex -setup" in the command window of matlab. Follow the instructions and choose the appropriate compiler. The native C compiler with Matlab did not compile this program. MS visual studio compilers are preferred.

- Change path to the /src/ directory and issue the command

```
mex FaceDetect.cpp -I../Include/ ../lib/*.lib -outdir ../bin/
```

  The compiled files are stored in the bin directory. Place these output files along with "cv100.dll" and "cxcore.dll" and the classifier file "haarcascade_frontalface_alt2.xml" in desired directory for your project and set path appropriately in matlab.

NOTE: compiling with Visual Studio 2005 version 8 compilier requires that a compiler sepcific dll be included along with the zip file. All the compiling on this zip are with visual studio 2003 version 7.1 compiler.

Usage:

```
FaceDetect (<Haar Cascade XML file>, <Gray scale Image>)
```

The function returns Nx4 matrix. In case no faces were detected, N=1 and all four entries are -1. Otherwise, N=number of faces in the image and the vector contains the x, y, width and height information of the face.

Task 2: Face detection using *Open CV Viola-Jones Face Detection in Matlab.* All the Yale database faces must be cropped automatically using face detection, such that only the face region remains. The images must then be resized to 60x50, see figure 5, refer to figure 6 for code sample.

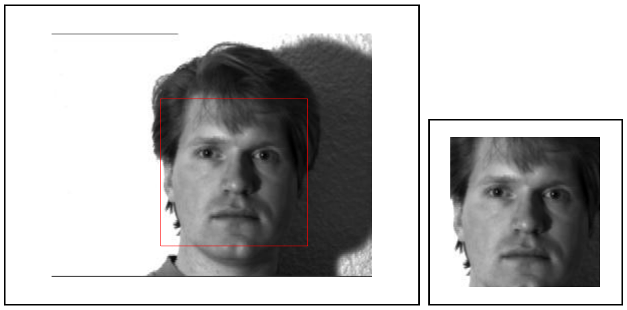

Figure 5: Face detection results using Intel OpenCV.

```
function cropFace = faceDetectCrop(fname, show)

A = imread (fname);
if isrgb(A)
   Img = double (rgb2gray(A));
else
  Img = double(A);
end

Face = FaceDetect('haarcascade_frontalface_alt2.xml',Img);

% face coordinates
[r c] = size(Face);

if (r == 1) % one face detected
    x = Face(1);
    y = Face(2);
    w = Face(3); % width
    h = Face(4) % height
else
    % get the row with the biggest area
    area = zeros(1,r);
    for i = 1:r
        w = Face(r,3); % width
        h = Face(r,4) % height
        area(i) = w*h;
    end

    [y I] = max(area);

    % chosen face region
    x = Face(I,1);
    y = Face(I,2);
    w = Face(I,3); % width
    h = Face(I,4); % height
end

cropFace = imcrop( A , [x y w h] );

if (show == 1)
    figure, imshow(A)
    hold on
    rectangle('Position',[x y w h],'EdgeColor','r');
    hold off
    figure, imshow(cropFace)

end

% Script M-file: mainFaceDetect.m
clear all,clc
close all

fname = 'subject01b.jpg';
show = 1;
cropFace = faceDetectCrop(fname, show);
```

Figure 6: Code snippet for using *Open CV Viola-Jones Face Detection in Matlab*

Training/Test Images

To create training and testing datasets for the experiments, the concept of *K-fold* cross-validation is utilized, as illustrated in Fig. 7. To create a K-fold partition of the dataset, for each of *K* experiments, use *K-1* folds for training and the remaining one for testing. The advantage of *K-fold* cross validation is that all the examples in the dataset are eventually used for both training and testing.

Leave-one-out (see Fig. 8) is the degenerate case of *K-fold* cross validation, where *K* is chosen as the total number of examples. For a dataset with *N* examples per class (person), perform *N* experiments. For each experiment use *N-1* examples for training and the remaining example for testing. The true error is estimated as the average error rate on test examples

In practice, the choice of the number of folds depends on the size of the dataset. For large datasets, even *3-fold* cross validation will be quite accurate. For very sparse datasets, we may have to use *leave-one-out* in order to train on as many examples as possible.

The goal is to arrive at a better estimate of the error rate (or classification rate). There are a specific number of training and test images for each experiment. Using this approach, the true error is estimated as the average error rate of the *K* experiments.

Task 3: Create a function *getTraining.m* and *getTest.m*. The images must first be converted to single-channel images (*pgm* file), with pixels scaled (0, 1) instead of (0, 255). See Fig. 9 for the function arguments and output.

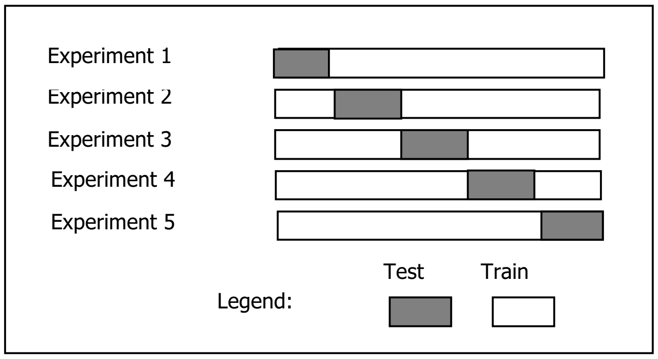

Figure 7. *K-fold* partition of the dataset.

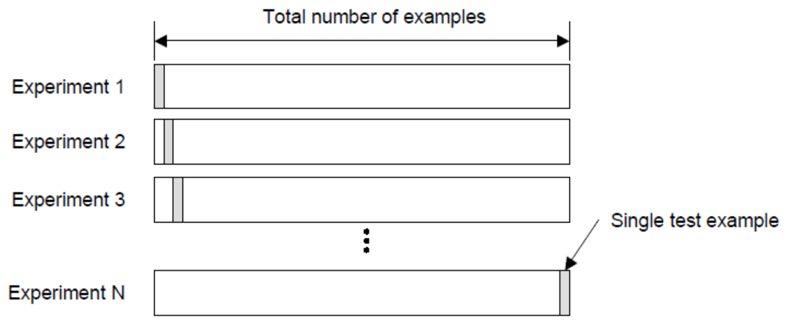

Figure 8. *Leave-one-out* partition of the dataset.

```
trainData = getTrain( [1 2 3 4 5 6 7 8 9 10] );
testData = getTest( [11] );

i = 1; % iterate subject
j = 1; % iterate images/subject
f = filesep;
dirName = ['s',num2str(i)];
im = imread([dirName,f,num2str(j),'.jpg'], 'jpg');
% convert to pgm
imwrite(im, [dirName,f,num2str(j),'.pgm'], 'pgm')

% scale to (0,1)
im = im/255;
```

Figure 9: Code snippet for *getTrain.m*, *getTest.m*, converting to *pgm* and scaling to (0, 1)

Feature Matching - Recognition

It seems that we are one step ahead to talk about feature matching and recognition before feature extraction, however for instructional purposes we postpone discussing feature extraction to the next section. Recognition is a matter of comparing a feature vector of a person in the gallery (database) with the one computed for the probe image (person), giving a similarity score. It can be viewed as if the probe is ranking the gallery with this similarity score, such that the most closest person in the gallery having the maximum similarity score to the probe image will be ranked as one, hence the similarity score to each person in the gallery will be ordered in a decreasing order. A probe image is correctly recognized in Rank-n system if it was found in the first n-gallery images being ordered by the similarity score to the probe image.

Similarity Measures

While more elaborate classifiers exist, most face recognition algorithms use the *nearest-neighbor* (NN) classifier as the final step due to the absence of training. The distance measures of the NN classifier will be in terms of the L1 (1) and L2 (2) norm, and the *cosine* (3) distance measures. For two vectors **x** and **y**, the similarity measures are defined as

$$d_{L1}(\mathbf{x},\mathbf{y}) = \sum_i |x_i - y_i| \tag{1}$$

$$d_{L2}(\mathbf{x},\mathbf{y}) = (\mathbf{x}-\mathbf{y})^T(\mathbf{x}-\mathbf{y}) \tag{2}$$

$$d_{\cos}(\mathbf{x},\mathbf{y}) = -\frac{\mathbf{x}\bullet\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} \tag{3}$$

Task 4: Create a function that computes the similarity between two vectors (i.e. dist = getDist(x, y, 'L1'))

Cumulative Match Score Curves (CMC) [10]

The identification method is a closed-universe test, that is, the sensor takes an observation of an individual that is known to exist in the database. The person's discriminating features are compared to those stored in the database and a similarity score is developed for each comparison. The similarity scores are then sorted in a descending order. In an ideal operation, the highest similarity score is the comparison of that person's recently acquired normalized signature with that of the person's normalized signature in the database. The percentage of times that the highest similarity score is the correct match for all individuals is called as the top match score.

An alternative way to view identification results is to take note if the top five numerically ranked scores contain the comparison of that person's recently acquired normalized signature with that of the person's normalized signature (features) in the database. The percentage of times that one of those five similarity scores is the correct match for all individuals is referred to as the Rank-n-score, where n = 5. The plot of rank-n versus probability of correct identification is called the Cumulative Match Score.

Task 5: Create a function that will generate the CMC curve given the feature vectors of a set of probe images (testing data) and the feature vectors of the gallery (face database used in the training), this function will make use of the function created in task 4, noting that for each similarity measure, there will be a different CMC curve.

Feature Extraction

Despite the high-dimensionality of face images, the appearance of faces is highly constrained (e.g., any frontal view of a face is roughly symmetrical, has eyes on the sides, nose in the middle, etc.) Therefore, the natural constraints dictate that the face images are confined to a subspace (*face space*) of the high-dimensional image space. To recover the face space, this project makes use of PCA, LDA and ICA, each having its own representation (basis images) of the high-dimensional face image space, based on different statistical viewpoints.

The three representations can be considered as a linear transformation from the original image space to the feature vector space, such that $\mathbf{Y} = \mathbf{W}^T\mathbf{X}$, where $\mathbf{Y}$ (d x m) is the feature vector matrix, m is the dimension of the feature vector, $\mathbf{X} = (x_1, x_2, \ldots, x_m)$ represent the (m x n) data matrix, x_i is the (m x 1) face vector and n is the number of face vectors used, and $\mathbf{W}$ is the transformation matrix.

Principal Component Analysis (PCA), Eigenfaces [3]

PCA starts with a random vector **x** with *m* elements, and has *n* samples **x**(1),…, **x**(*n*). For face recognition, the random vector samples are the face images and the elements of **x** are the pixel gray level values. To summarize the PCA method, the algorithm uses the steps below. The first step is to center the vector **x** by subtracting its mean, $\mathbf{x} \leftarrow \mathbf{x} - E\{\mathbf{x}\}$. The mean-centered vector **x** is then linearly transformed to another vector **y** with *d* elements, such that $d << m$, leaving behind a compact representation of the images. The transformation from the *m*- to the *d*-dimensional space starts with the computation of the eigenvectors of the covariance matrix (scatter-matrix) S_X,

$$S_X = \sum_{i=1}^{m} (x_i - \mu_i)(x_i - \mu_i)^T \tag{4}$$

where $\mathbf{x_i}$ and $\boldsymbol{\mu}_\mathbf{i}$ are the original sample vector and overall mean, respectively. The transformation matrix $\mathbf{W_{PCA}}$ is composed of the eigenvectors corresponding to the *d* largest eigenvalues, constructed by stacking the eigenvectors in columns.

The eigenvectors of S_X exhibit interesting visual properties. Consider the first 10 images for each subject as the training images (i.e. *1.jpg*, *2.jpg… 10.jpg*). Perform PCA on the training images. The resulting eigenvectors can be visualized like that of Fig. 10.

Task 6: Consider the first 10 images for each subject as the training images (i.e. *1.jpg*, *2.jpg… 10.jpg*). Perform PCA on the training images. Visualize the first *d* eigenvectors like Fig. 10. See Fig. 11.

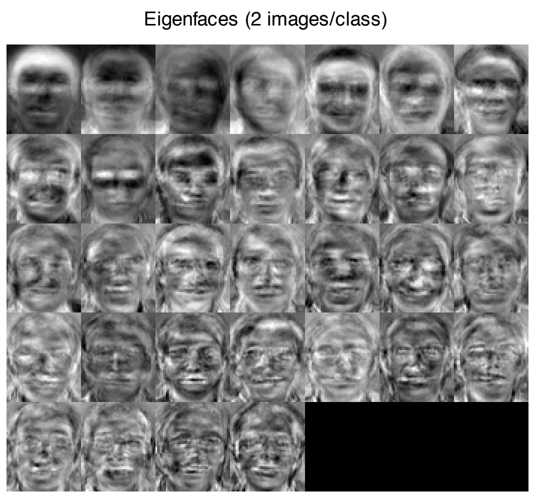

Figure 10: The first 32 eigenvectors, visualized as eigenfaces.

```
[r c] = size(trainPlot);

for i = 1:c
    mx = max(trainPlot(:,i));
    mi = min(trainPlot(:,i));
    trainPlot(:,i) = (trainPlot(:,i)-mi)./(mx-mi);
end

eigFacePlot = reshape(trainPlot,[60 50 1 c]);
```

Figure 11: Code snippet for visualizing eigenfaces.

Task 7: Plot the eigenvalue spectrum (Fig. 12). This provides a visual approximation on how many eigenvectors to choose.

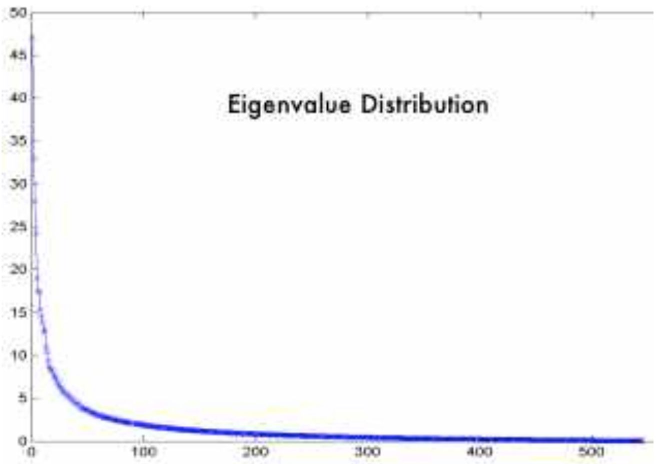

Figure 12: An example of the eigenvalue spectrum plot. In this example, the first 100-200 eigenvectors can be chosen, since the remaining eigenvalues have extremely small magnitudes.

Leaving-one-out cross-validation is a special case of Fig. 7, such that there is only 1 test image and the remaining images of the subject are considered as training. For the Yale face database, *leaving-one-out* cross-validation consists of 11 experiments since there are 11 images each per subject.

Task 8: Perform *leaving-one-out* cross-validation of the PCA algorithm using the Yale database. Use the three similarity measures to classify test images after transforming both test and training images to a lower-dimension vector. Report the error rate for each similarity measure. Generate the CMC curve for each similarity measure, comment on your CMC curves, which measure is better?

	Error Rate (%)		
Method	L1	L2	Cosine
PCA (Eigenface)			

Linear Discriminant Analysis (LDA), Fisherfaces [3]

The goal of LDA is to find basis vectors that exploit class information to improve classification results. LDA is known as the Fisher's Linear Discriminant (FLD) in the face recognition literature.

FLD solves for the transformation matrix $\mathbf{W}_{\mathbf{LDA}}$ by maximizing the ratio of the between-class scatter (S_B) and the within-class scatter (S_W). The two scatter matrices are defined as follows

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T \tag{5}$$

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T \tag{6}$$

where $\boldsymbol{\mu}_i$ is the mean image of class $\boldsymbol{X}_i$, $\boldsymbol{x}_k$ is a sample image, $\boldsymbol{N}_i$ is the number of samples in class $\boldsymbol{X}_i$, c is the number of distinct classes, and $\boldsymbol{\mu}$ is the overall sample mean. The transformation matrix $\mathbf{W}_{\mathbf{LDA}}$ can be computed by solving the generalized eigenvalue problem

$$S_B W = \lambda S_W W \tag{7}$$

where W is the matrix of eigenvectors in its columns and $\boldsymbol{\lambda}$ is a diagonal matrix of eigenvalues. To prevent the singularity of the within-class scatter matrix, PCA is used as a preprocessing step to reduce the dimension of the image vectors to $(m - c)$. LDA can then be used to reduce the vectors to $(c - 1)$.

Task 9: Consider the first 10 images for each subject as the training images (i.e. *1.jpg*, *2.jpg*… *10.jpg*). Perform LDA on the training images without doing the PCA preprocessing step (See Fig. 13). Report your experience.

Task 10: Consider the first 10 images for each subject as the training images (i.e. *1.jpg*, *2.jpg*… *10.jpg*). Perform LDA on the training images with PCA as a preprocessing step, i.e. reduce the dimension of the image vectors to (c – 1), where c is the number of subjects (classes). Visualize the first d fisherfaces like Fig. 14. Compare the generalized eigenvalue analysis to that of *Task 10* (See Fig. 13).

Task 11: Perform *leaving-one-out* cross-validation of the LDA algorithm using the Yale database. Use the three similarity measures to classify test images after transforming both test and training images to a lower-dimensional vector. Report the error rate for each similarity measure. Generate the CMC curve for each similarity measure, comment on your CMC curves, which measure is better?

	Error Rate (%)		
Method	L1	L2	Cosine
LDA (Fisherface)			

```
numPCA = numIm - numClass;
plotEig = 0;
PCA = compEigenface(trainData.imTrain, numPCA, plotEig);

trainFisher = trainData.imTrain;
[r c] = size(trainFisher)

% mean-center
me = mean(trainFisher,2);
trainFisher = trainFisher - repmat(me, [1 c]);

% Calculate within-class scatter matrix
Nsize = size(trainFisher,1);
Sw = zeros(Nsize);
meanClass = zeros(Nsize,numClass);
prod = zeros(Nsize);

temp_im = [];
prod = [];
for i = 1:numClass

    % generalize this to any n images/class
    temp_im = trainFisher(:,numImClass*i-(numImClass-1):numImClass*i);
    meanClass(:,i) = mean(temp_im,2);

    % two images/class
    temp_im = temp_im - repmat(meanClass(:,i),[1,numImClass]);

    for j = 1:numImClass
    prod = temp_im(:,j)*temp_im(:,j)';
    Sw = Sw + prod;
    end

end % end for

% Calculate between-class scatter matrix
Sb = zeros(Nsize);
me = mean(trainFisher,2); % overall mean

clear temp_im prod
for i = 1:numClass

    temp_im = meanClass(:,i) - me;
    prod = temp_im*temp_im';
    Sb = Sb + numImClass*prod; % 2 im/class

end
clear prod temp_im

P1 = PCA.eigFace;
Sbb = P1'*Sb*P1; % PCA reduction
Sww = P1'*Sw*P1; % PCA reduction
clear Sb Sw

[V,D] = eig(Sbb, Sww);
```

Task 12: Perform Task 7 and 11 on images that are preprocessed with histogram equalization (*imhist.m*). Compare results.

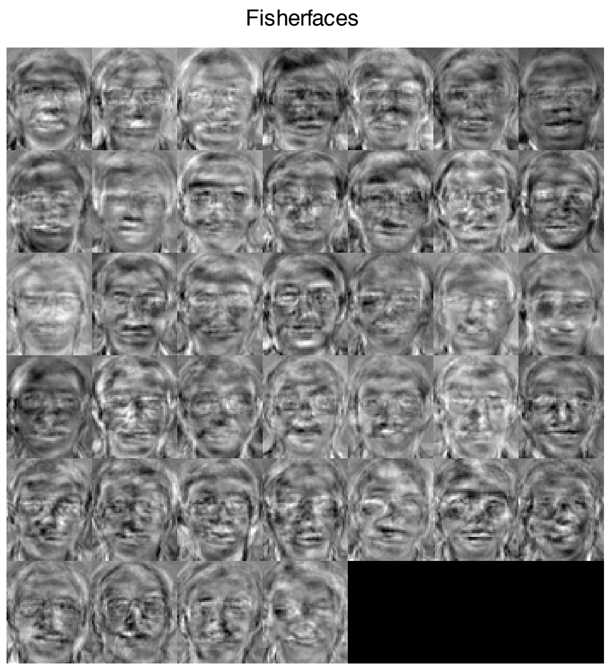

Figure 14: LDA Basis Images (39 Fisherfaces)

Independent Component Analysis (ICA)

While PCA decorrelates the input data using second-order statistics (the covariance/scatter matrix), which results into compressed data with minimum mean-squared re-projection error, independent component analysis (ICA) minimizes both the second-order and higher-order dependencies in the input.

ICA is related to the blind source separation (BSS) [7], where the goal is to decompose an observed signal into a linear combination of unknown independent signals. Consider a number of people (e.g., three) in a room speaking simultaneously, with three microphones placed in different locations to pick up the sound generated by the speakers. The microphones produce three recorded time signals, denoted by $x_1(t)$, $x_2(t)$ and $x_3(t)$. The three signals is a weighted sum of the speech signals emitted by the three speakers, denoted by $s_1(t)$, $s_2(t)$, and $s_3(t)$. The recorded signals $x_i(t)$ can be expressed, in matrix form, as a linear equation:

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} s_1(t) \\ s_2(t) \\ s_3(t) \end{bmatrix} \tag{8}$$

where a_{ij} are parameters that depend on the distances of the microphone from the speakers. It would be useful to estimate the original speaker signals $s_i(t)$, using only the recorded signals $x_i(t)$ and without any knowledge of the mixing parameters a_{ij}. This problem is referred to as the cocktail-party or the blind source separation problem.

This leads to the formal definition of ICA, which is essentially estimating both the matrix **A** (consisting of the mixing parameters a_{ij}) and the speech signals $s_i(t)$, given only the observed signals $x_i(t)$. In compact matrix form, let **s** be the vector of unknown source signals, **x** be the vector of observed mixtures and **A** be the unknown mixing matrix, then the mixing model is written as

$$x = As \tag{9}$$

Important assumptions in the ICA problem include that the source signals must be independent from each other (the speakers in the cocktail-party problem are independent) and the mixing matrix **A** is invertible. The main goal of ICA algorithms [8] is to find the mixing matrix **A** or the separating/unmixing matrix **W** such that

$$u = Wx = W(As) \tag{10}$$

where **u** is an estimation of the independent source signals. Fig. 15 illustrates the blind-source separation problem, using a block diagram.

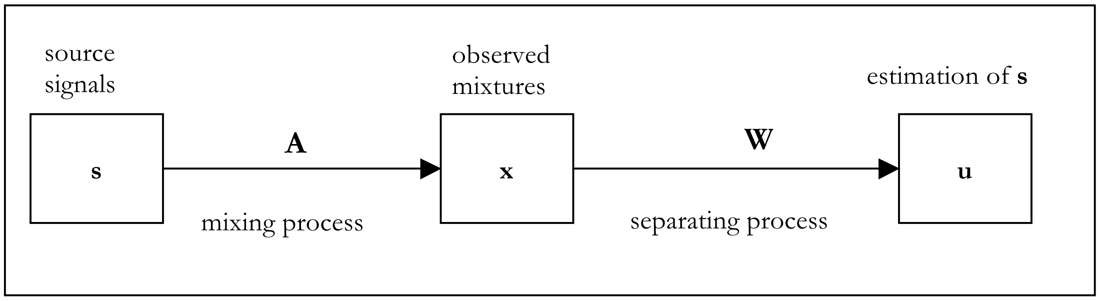

Figure 15: Blind source separation model – courtesy of [5]

Non-Gaussianity Estimation

The fundamental restriction in ICA is that the independent components must be *non-gaussian* for ICA to be possible. To see why gaussian variables make ICA impossible, assume that the mixing matrix is orthogonal and the $\mathbf{s}_i$ are gaussian. Then $\mathbf{x}_1$ and $\mathbf{x}_2$ are gaussian too (by central limit theorem), they are uncorrelated, and of unit variance. The joint density is completely *symmetric*. Therefore, it does not contain any information on the directions of the columns of the mixing matrix **A.** This is why **A** cannot be estimated. Moreover, the distribution of any orthogonal transformation of the gaussian $(\mathbf{x}_1,\mathbf{x}_2)$ has exactly the same distribution as $(\mathbf{x}_1,\mathbf{x}_2)$. Thus, in the case of gaussian variables, we can only estimate the ICA model up to an orthogonal transformation.

Let us now assume that the data vector **x** is distributed according to the ICA data model, i.e. a mixture of independent components. For simplicity, let us assume that all the independent components have identical distributions. To estimate one of the independent components, we consider a linear combination of the $\mathbf{x}_i$, let's denote this by **y** ;

$$y = w^T x \tag{11}$$

where **w** is a vector to be determined, and it's one row of the inverse of **A**, i.e. **W.**

Define $\mathbf{z} = \mathbf{A}^T\mathbf{w}$ and then we have,

$$y = w^T x = w^T A s = z^T s \tag{12}$$

This linear combination would actually equal one of the independent components. The question is now: How could we use the Central Limit Theorem to determine **w** so that it would equal one of the rows of the inverse of **A**? In practice, we cannot determine such **w** exactly, because we have no knowledge of matrix **A**, but we can find an estimator that gives a good approximation. $\mathbf{z}^T\mathbf{s}$ is more Gaussian than any of the $\mathbf{s}_i$, and it is least Gaussian (i.e. non-guassian) if it is equal to one of the $\mathbf{s}_i$ Maximizing the non-Gaussianity of $\mathbf{w}^T\mathbf{x}$ will give us one of the independent components.

The next step is to discuss quantitative measures of nongaussianity for a random variable to be able to use nongaussianity in ICA estimation. The classical measure of nongaussianity is *kurtosis*, otherwise known as the fourth-order cumulant (note that the random variable here is mean-centered with unit variance). The kurtosis of y is defined as:

$$\text{kurt}(y) = E\{y^4\} - 3\left(E\{y^2\}\right)^2 \tag{13}$$

Since **y** is of unit variance, the kurtosis equation simplifies to $E\{y^4\} - 3$. Therefore, the kurtosis can be considered as the normalized version of the fourth moment $E\{y^4\}$. The kurtosis for a Gaussian is zero because the fourth moment is equal to $3(E\{y^2\})^2$. For most nongaussian random variables, the value for kurtosis is nonzero. Kurtosis can be positive or negative. Random variables that have negative kurtosis are called subgaussian (flatter, more uniform, shorter tail than Gaussian), and those with positive values for kurtosis are referred to as supergaussian (more peaked, than Gaussian, heavier tail).

Another measure for nongaussianity is the concept of *negentropy*, which is based on the information-theoretic quantity of entropy. The entropy of a random variable can be interpreted as the degree of information that the observation of the variable gives. The more unpredictable (random) and unstructured the variable is, the larger the entropy value. For a discrete random variable Y, the entropy H is defined as:

$$H(Y) = -\sum_i P(Y = a_i)\log P(Y = a_i) \tag{14}$$

where a_i are the possible values of **Y**. The entropy definition can also be generalized to the continuous case and is often called the differential entropy. The differential entropy **H** of a random variable **y** with density *f(y)* is defined as:

$$H(y) = -\int f(y)\log f(y)\,dy \tag{15}$$

A fundamental result of information theory [7] is that the Gaussian random variable has the largest entropy among all random variables of equal variance, which means that entropy can be used to measure nongaussianity. To obtain a measure of nongaussianity that is zero for Gaussian random variables and always nonnegative, a slightly modified version of differential entropy is employed, which is called negentropy. Negentropy *J* is defined as:

$$J(y) = H(y_{gauss}) - H(y) \qquad (16)$$

The use of negentropy as a measure for nongaussianity is well-justified in information theory but the problem with it lies in it being computationally difficult to compute. There are several approximations for entropy in the literature to alleviate this problem [7]. The classical method of approximating negentropy is using higher-order moments:

$$J(y) \approx \frac{1}{12} E\{y^3\}^2 + \frac{1}{48} kurt(y)^2 \qquad (17)$$

The random variable y is assumed to be of zero mean and unit variance. However, the validity of such approximations may be rather limited. To avoid the problems encountered with the preceding approximation, new approximations were developed based on the maximum-entropy principle:

$$J(y) \approx \sum_{i=1}^{p} k_i \left[E\{G_i(y)\} - E\{G_i(v)\}\right]^2 \qquad (18)$$

where k_i are some positive constants, and v is a Gaussian variable of zero mean and unit variance . The variable y is assumed to be of zero mean and unit variance, and the functions G_i are some nonquadratic functions. In particular, choosing G that does not grow too fast, one obtains more robust estimators. The following choices of G have proved very useful:

$$G_1(u) = \frac{1}{a_1} \log\cosh a_1 u \quad , \quad G_2(u) = -\exp(-u^2/2) \qquad (19)$$

where $1 \le a_1 \le 2$ is constant

ICA-Estimation Approaches

Two popular methods in estimating the ICA model are, Minimization of Mutual Information Maximum Likelihood Estimation.

1. <u>Minimization of Mutual Information</u>

Using the concept of differential entropy, mutual information *I* between *m* random variables can be define as following,

$$I(y_1, y_2, \ldots, y_m) = \sum_{i=1}^{m} H(y_i) - H(y) \tag{20}$$

Mutual information is the natural measure of the dependence between random variables. Its value is always nonnegative, and zero if and only if the variables are statistically dependent. When the original random vector **x** undergoes an invertible linear transformation **y** = **Wx**, the mutual information for **y** in terms of **x** is

$$I(y_1, \ldots, y_m) = \sum_i H(y_i) - H(\mathbf{x}) - \log|\det \mathbf{W}| \tag{21}$$

Consider the scenario when y_i is constrained to be uncorrelated and of unit variance, which implies that $E\{\mathbf{yy}^T\} = \mathbf{W}E\{\mathbf{xx}^T\}\mathbf{W}^T = \mathbf{I}$. Applying the determinant on all sides of the equation leads to:

$$\det I = 1 = \det\left(WE\{xx^T\}W^T\right) = (\det W)\left(\det E\{xx^T\}\right)\left(\det W^T\right) \tag{22}$$

Hence det**W** must be constant since det $E\{\mathbf{xx}^T\}$ does not depend on **W**. For **y** of unit variance, entropy and negentropy differ only by a constant and sign. Therefore, the fundamental relation between entropy and negentropy is:

$$I(y_1, \ldots, y_n) = C - \sum_i J(y_i) \tag{23}$$

where C is a constant not dependent on **W**. Thus finding an invertible transformation **W** that minimizes the mutual information is roughly equivalent to finding directions in which negentropy (a concept related to nongaussianity) is maximized.

2. Maximum Likelihood Estimation

To derive the likelihood of the noise-free ICA model, a well-known result on the density of a linear transform is used. According to the result, the density p_x of the mixture vector (the ICA model), **x** = **As** is

$$f_x(x) = |\det W| f_s(s) = |\det W| \prod_{i=1}^{n} f_i(s_i) \tag{24}$$

where $\mathbf{W} = \mathbf{A}^{-1}$, and f_i denote the densities of the independent components $\mathbf{s}_i$. The density p_x can also be expressed as a function of **x** and $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2 \ldots \mathbf{w}_n)^T$, that is,

$$f_x(x) = |\det W| \prod_{i=1}^{n} f_i\left(w_i^T x\right) \tag{25}$$

Assuming that there are T observations of **x**, denoted by **x**(1), **x**(2), …, **x**(T), and after some manipulations, the final equation for the log-likelihood is:

$$L = \sum_{t=1}^{T} \sum_{i=1}^{n} \log f_i\left(w_i^T x(t)\right) + T \log\left|\det W\right| \qquad (26)$$

The problem with this approach is that density functions f_i must be estimated correctly, otherwise ML estimation will give a wrong result.

ICA Gradient Ascent

This algorithm is based on maximizing the entropy of the estimated components. Assume that we have n mixtures $\mathbf{x_1, \ldots, x_n}$ of n independent components/sources $\mathbf{s_1, \ldots, s_n}$:

$$x_j = a_{j1}s_1 + a_{j2}s_2 + \ldots + a_{jn}s_n \quad \text{for all j} \qquad (27)$$

Assume that the sources has a common cumulative density function (cdf) g and probability density function (pdf) p_s. Then given an unmixing matrix $\mathbf{W}$ which extracts n components $\mathbf{u} = (\mathbf{u_1, \ldots, u_n})^T$ from a set of observed mixtures $\mathbf{x}$, the entropy of the components $\mathbf{U} = g(\mathbf{u})$ will be, by definition:

$$H(U) = H(x) + E\left\{\sum_{i=1}^{n} \ln p_s(u_i)\right\} + \ln|W| \qquad (28)$$

where $\mathbf{u_i} = \mathbf{w_i}^T\mathbf{x}$ is the ith component, which is extracted by the ith row of the unmixing matrix $\mathbf{W}$. This expected value will be computed using m sample values of the mixtures $\mathbf{x}$. By definition, the pdf p_s of a variable is the derivative of that variable's cdf g:

$$p_s(u_i) = \frac{d}{du_i} g(u_i) \qquad (29)$$

Where this derivative is denoted by $g'(\mathbf{u_i}) = p_s(\mathbf{u_i})$, so that we can write:

$$H(U) = H(x) + E\left\{\sum_{i=1}^{n} \ln g'(u_i)\right\} + \ln|W| \qquad (30)$$

We seek an unmixing $\mathbf{W}$ that maximizes the entropy of $\mathbf{U}$. Since the entropy $H(\mathbf{x})$ of the mixtures $\mathbf{x}$ is unaffected by $\mathbf{W}$, its contribution to $H(\mathbf{U})$ is constant, and can therefore be ignored. Thus we can proceed by finding that matrix $\mathbf{W}$ that maximizes the function:

$$h(U) = E\left\{\sum_{i=1}^{n} \ln g'(u_i)\right\} + \ln|W| \qquad (31)$$

Which is the change in entropy associated with the mapping from $\mathbf{x}$ to $\mathbf{U}$. We can find the optimal $\mathbf{W^*}$ using gradient ascent on h by iterartively adjusting $\mathbf{W}$ in order to maximize the function h. In order to perform gradient ascent efficiently, we need an expression for the gradient of h with respect to the matrix $\mathbf{W}$. We proceed by finding the partial derivative of h with respect to one scalar element $\mathbf{W_{ij}}$ of $\mathbf{W}$, where $\mathbf{W_{ij}}$ is the element of the ith row and jth column of $\mathbf{W}$. The weight $\mathbf{W_{ij}}$ determines the proportion of the *j*th mixture $\mathbf{x}_j$ in the *i*th extracted component $\mathbf{u}_i$. Given that $\mathbf{u} = \mathbf{Wx}$, and that every component $\mathbf{u}_i$ has the same pdf *g'*. The partial derivative of *h* with respect to the *ij*th element in **W** is:

$$\frac{\partial}{\partial W_{ij}} h(U) = E\left\{\sum_{i=1}^{n} \psi(u_i) x_j\right\} + \left[W^{-T}\right]_{ij} \tag{32}$$

If we consider all the element of **W**, then we have:

$$\nabla h = W^{-T} + E\{\psi(u) x^T\} \tag{33}$$

Where ∇h is an *n* x *n* Jacobian matrix of derivatives in which the *ij*th element is $\partial h / \partial W_{ij}$. Given a finite sample of *N* observed mixture values of $\mathbf{x}^k$ for *k* = *1,2,…,N* and a putative unmixing matrix **W**, the expectation can be estimated as:

$$E\{\psi(u) x^T\} = \frac{1}{N}\sum_{k=1}^{N} \psi(u^k)[x^k]^T \quad \textit{where} \quad u^k = Wx^k \tag{34}$$

Thus the gradient ascent rule, in its most general form will be:

$$W_{new} = W_{old} + \eta \nabla h \quad \textit{where} \quad \eta \text{ is a small constant} \tag{35}$$

Thus the rule for updating **W** in order to maximize the entropy of $\mathbf{U} = g(\mathbf{u})$ is therefore given by:

$$W_{new} = W_{old} + \eta\left(W^{-T} - \frac{2}{N}\sum_{k=1}^{N} \tanh(u^k)[x^k]^T\right) \tag{36}$$

Preprocessing for ICA

1. <u>Centering</u>

The most basic and necessary preprocessing is to center the data matrix **X**, that is, subtract the mean vector, $\boldsymbol{\mu} = \mathbf{E(X)}$ to make the data a zero-mean variable. With this, **s** can be considered to be zero-mean, as well. After estimating the mixing matrix **A**, the mean vector of **s** can be added back to the centered estimates of **s** to complete the estimation. The mean vector of **s** is given by $\mathbf{A}^{-1}\boldsymbol{\mu}$, where $\boldsymbol{\mu}$ is the mean vector of the data matrix **X**.

2. <u>Whitening</u>

Aside from centering, whitening the observed variables is a useful preprocessing step in ICA. The observed vector **x** is linearly transformed to obtain a vector that is *white*, which means its components are uncorrelated (zero covariance) and the variance is equal to unity. In terms of covariance, the covariance of the new vector $\tilde{x}$ equals the identity matrix,

$$E\{\tilde{x}\tilde{x}^T\} = I \tag{37}$$

There are several ways to whiten the data set, one popular method for whitening is to use the eigen-value decomposition (EVD) of the covariance matrix $E\{xx^T\} = VDV^T$, where **V** is the orthogonal matrix of eigenvectors of $E\{xx^T\}$ and **D** is the diagonal matrix of its eigenvalues, **D** = diag(d_1, ...,d_n).

Whitening can now be done by:

$$\tilde{x} = VD^{-\frac{1}{2}}V^T x \tag{38}$$

where the matrix $\mathbf{D}^{-1/2}$ is computed by a simple component-wise operation as $D^{-1/2} = \mathrm{diag}(d_1^{-1/2}, ..., d_n^{-1/2})$.

Whitening transforms the mixing matrix into a new one,

$$\tilde{x} = VD^{-\frac{1}{2}}V^T x = VD^{-\frac{1}{2}}V^T As = \tilde{A}s \tag{39}$$

Here we see that whitening reduces the number of parameters to be estimated. Instead of having to estimate the n^2 parameters that are the elements of the original matrix **A**, we only need to estimate the new, orthogonal mixing matrix $\tilde{A}$ which contains $n(n-1)/2$ degrees of freedom. Thus one can say that whitening solves half of the problem of ICA. For simplicity of notation, we denote the preprocessed data just by **x**, and the transformed mixing matrix by **A**, omitting the *tildes*.

Because whitening is a very simple and standard procedure, much simpler than any ICA algorithms, it is a good idea to reduce the complexity of the problem this way. It may also be quite useful to reduce the dimension of the data at the same time as we do the whitening. Then we look at the eigen values d_j *of* $E\{\mathbf{xx}^T\}$ and discard those that are too small, as is often done in the statistical technique of principal component analysis (PCA). This has often the effect of reducing noise. Moreover, dimension reduction prevents over-learning, which can sometimes be observed in ICA.

Centering and whitening combined is referred to as *sphering*, and is necessary to speed up the ICA algorithm. Sphering removes the first and second-order statistics of the data; both the mean and covariance are set to zero and the variance are equalized. When the sample data inputs of the ICA problem are sphered, the full transformation matrix $\mathbf{W_I}$ is the product of the sphering matrix $\mathbf{W_Z}$ and the matrix learned by the ICA **W**, that is,

$$W_I = WW_Z \tag{40}$$

ICA for Face Recognition - Architecture I

There are two fundamentally different architectures for applying ICA to face recognition, which will be named Architecture I and II [9]. In Architecture I, the face images in the data matrix X are considered to be a linear mixture of statistically independent basis images S combined by an unknown mixing matrix A. The goal of the ICA algorithm is to solve the weight matrix W, which is used to recover the set of independent basis images. Figure 16 illustrates the Architecture I for face recognition. The face images are considered variables and the pixels the observations for the variables in Architecture I. The source separation, therefore, occurs in the face space.

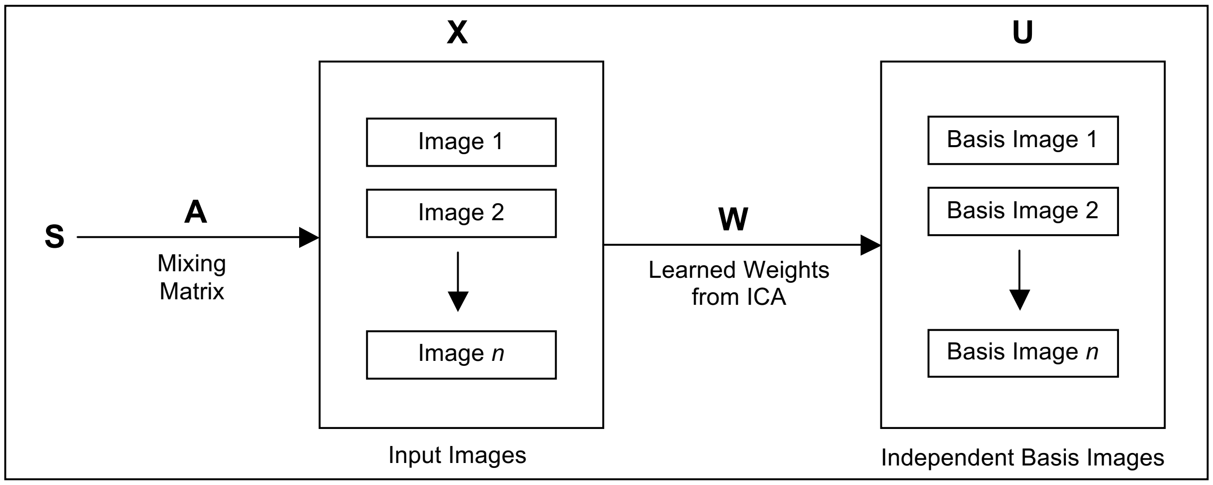

Figure 16: ICA Architecture I [8]. The goal in this architecture is to find statistically independent basis images.

Bartlett [9] uses PCA as a preprocessing step before the main ICA algorithm to project the data into a subspace of a certain dimension, which also happens to be the number of independent components produced by ICA.

Task 13: Construct the data matrix **X** using the face images used in the training stage.

Task 14: Perform data centering and whitening on the constructed data matrix **X,** see Fig. 17.

```
%% Preprocessing - data sphering
Mu = mean(x);
covM = cov(x);

%% 1. Centering
% subtract the mean from the observed mixture
x = x - repmat(Mu,[N,1])

%% 2. Whitening
% get decorrelating matrix (whitening matrix)
whitening_matrix = 2*inv(sqrtm(covM));

% decorrelate mixes so cov(x')=4*eye(n);
x = x * whitening_matrix;
```

Figure 17: Code snippet for ICA pre-processing

Task 16: Apply ICA Gradient Ascent algorithm on the data matrix **X** by using the eigenvectors from the PCA (the first *d* eigenvectors) of **X** to produce the statistically independent basis images. see Fig. 19. Plot the change in entropy versus iterations. Comment on your results. Visualize the ICA basis images, like Fig.18. Generate the CMC curve for each similarity measure, comment on your results.

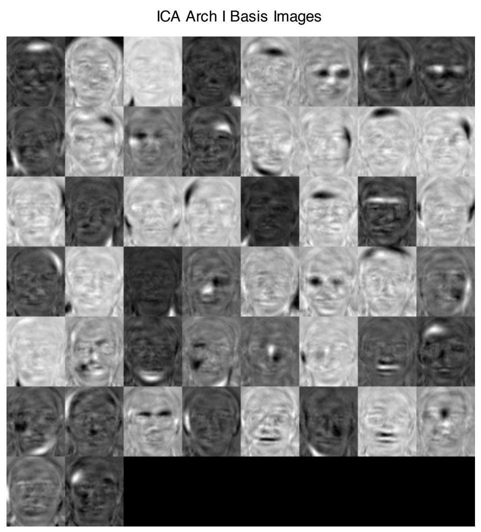

Figure 18: ICA Architecture I Basis Images

```
%% Initializations
% Initialise unmixing matrix W to identity matrix.
W = eye(n,n);

% Initialise u, the estimated source signals.
u = x*W;

maxiter = 10000;  % Maximum number of iterations.
eta = 1;       % Step size for gradient ascent (learning rate).

% Make array hs to store values of function and gradient magnitude.
hs = zeros(maxiter,1);
gs = zeros(maxiter,1);

%% Begin gradient ascent on h ...
for iter = 1:maxiter
    % Get estimated source signals, u.
    u = x*W; % wt vec in col of W.
    % Get estimated maximum entropy signals U = cdf(u).
    U = tanh(u);
    % Find value of function h.
    % h = log(abs(det(W))) + sum( log(eps+1-U(:).^2) )/N;
    detW = abs(det(W));
    h = ( (1/N)*sum(sum(U)) + 0.5*log(detW) );
    % Find matrix of gradients @h/@W_ji ...
    g = inv(W') - (2/N)*x'*U;
    % Update W to increase h ...
    W = W + eta*g;
    % Record h and magnitude of gradient ...
    hs(iter) = h;
    gs(iter) = norm(g(:));
end;

% the estimated independent components
u = x*W;
```

Figure 19: Code snippet for ICA Gradient Ascent

To summarize Architecture I in terms of matrix notation, let **R** be the (*pxm*) matrix containing the first *m* eigenvectors from the PCA preprocessing step (the eigenvectors are stacked column-wise) and *p* is the number of pixels in an image. The convention in ICA is that the rows of the input matrix are the variables and the columns contain the observations, which means that the input to the ICA algorithm is $\mathbf{R}^T$. The *m* independent basis images in the rows of **U** are computed as $\mathbf{U} = \mathbf{W}\mathbf{R}^T$, where **W** is weight matrix estimated from ICA. The (*nxm*) ICA coefficients matrix **B** for the linear combination of independent basis images in **U** is computed as follows [8]. Let **C** be the (*nxm*) matrix of PCA coefficients. **C** can be solved as

$$C = X\,R \Rightarrow X = C\,R^T \tag{41}$$

From $\mathbf{U} = \mathbf{W}\mathbf{R}^T$ and the assumption that **W** is invertible, $\mathbf{R}^T$ in terms of **U** and **W** is: $\mathbf{R}^T = \mathbf{W}^{-1}\mathbf{U}$. Therefore,

$$\mathrm{X} = (\mathrm{C\,W}^{-1})\mathrm{U} = \mathrm{B\,U} \tag{42}$$

The rows of **B** contain the coefficients for linearly combining the basis images to comprise the face images in the corresponding rows of **X**. **X** is the reconstruction of the original data with minimum squared error as in PCA.

ICA for Face Recognition - Architecture II

Architecture II is based on the idea of finding image filters that produce statistically independent outputs from natural scenes. The basis images in Architecture I are statistically independent, but the coefficients that represent the input images in the new space defined by the basis images are not. The role of pixels and images are changed from Architecture I, that is, the pixels are variables and the images are observations. The source separation is performed on the pixels (instead of the face space in Architecture I), and each row of the solved weight matrix **W** is an image. In this architecture (Fig. 20), the ICA algorithm is done on the PCA coefficients rather than the input images to reduce the dimensionality of the image vectors. In matrix notation, the statistically independent coefficients are computed as $\mathbf{U} = \mathbf{W}\,\mathbf{C}^{\mathbf{T}}$ and the actual basis images are obtained from the columns of **RA**.

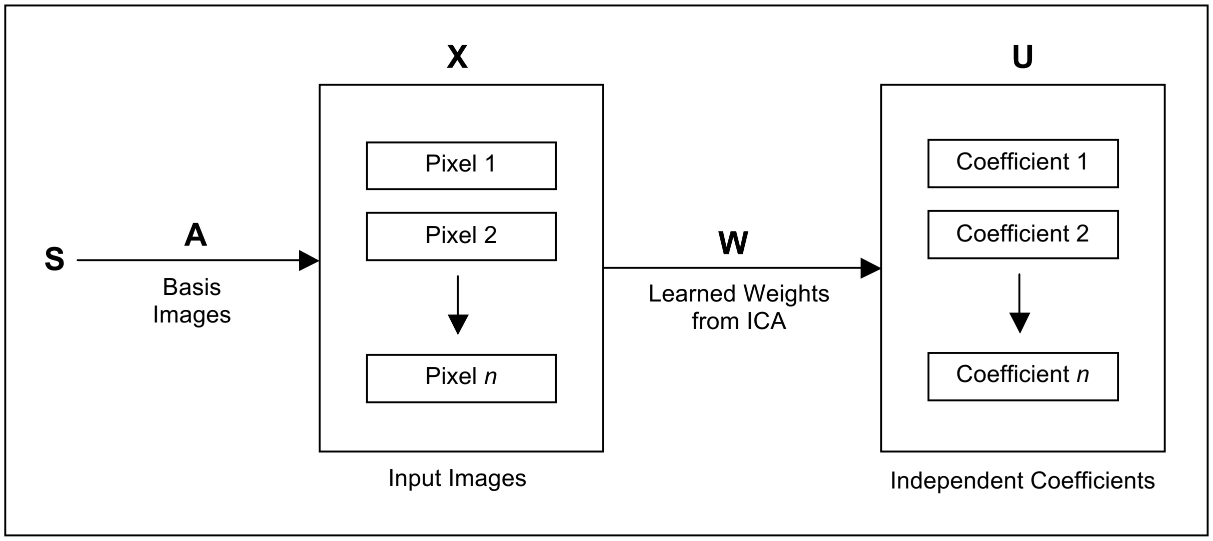

Figure 20: ICA Architecture II [8]. The goal of this architecture is find statistically independent coefficients for face representation.

Task 17: Repeat tasks 13-16 but with different data matrix **X** which is constructed to follow Architecture II.

Correlation-based Pattern Recognition [4]

Correlation is a natural metric for characterizing the similarity between a reference pattern $r(x, y)$ and a test pattern $f(x, y)$, and not surprisingly, it has been used often in pattern recognition applications. Often, the two patterns being compared exhibit relative shifts and it makes sense to compute the cross-correlation $c(\tau_x, \tau_y)$ between the two patterns for various possible shifts τ_x and τ_y as in (43); then, it makes sense to select its maximum as a metric of the similarity between the two patterns and the location of the correlation peak as the estimated shift of one pattern with respect to the other

$$c(\tau_x, \tau_y) = \iint f(x, y) r(x - \tau_x, y - \tau_y) dx dy \qquad (43)$$

where the limits of integration are based on the support of $I(x, y)$. The correlation operation in (43) can be equivalently expressed as

$$\begin{aligned} c(\tau_x, \tau_y) &= \iint F(w_1, w_2) R^*(w_1, w_2) e^{j(w_1\tau_x + w_2\tau_y)} dxdy \\ &= \Im^{-1}\left(F(w_1, w_2) R^*(w_1, w_2)\right) \end{aligned} \tag{44}$$

where $F(w_1,w_2)$ and $R(w_1,w_2)$ are the 2D FTs of $f(x, y)$ and $r(x, y)$. Equation (43) can be interpreted as the test pattern $F(x, y)$ being filtered by a filter with frequency response $H(w_1,w_2) = R^*(w_1,w_2)$ to produce the output $c(\tau_x, \tau_y)$. The goal is to design a suitable filter $H(w_1,w_2)$ that will determine which class the test image belongs to (Fig. 21).

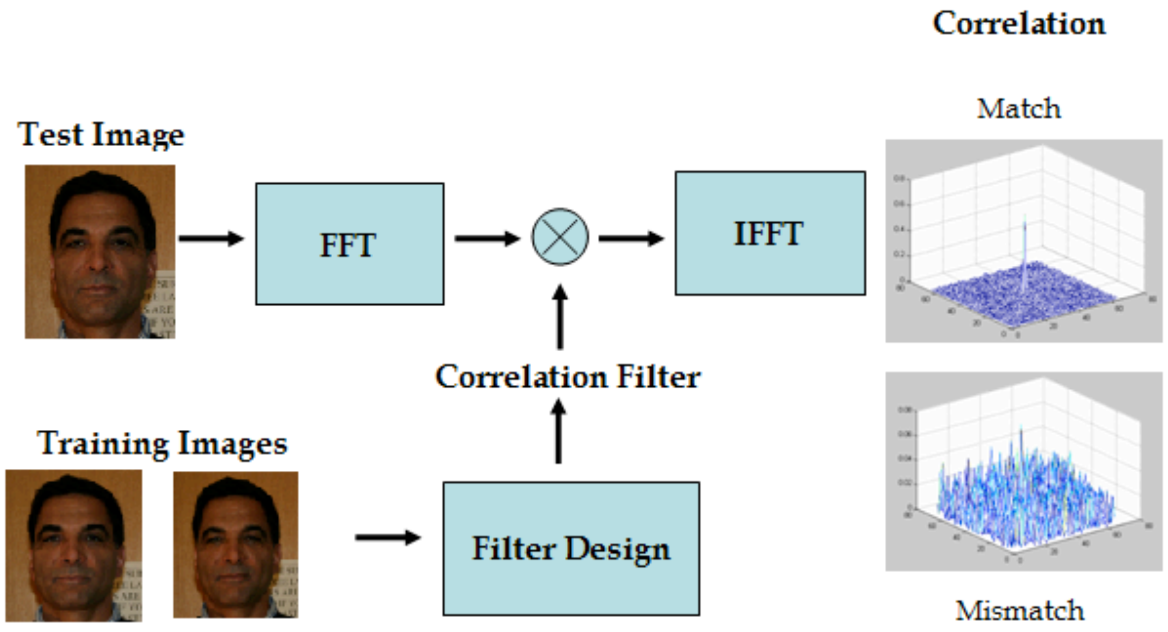

Figure 21: Block diagram of the correlation process.

The filter that will be considered in this project is the minimum average correlation (MACE) filter. The MACE filter design can be summarized as follows. We will now briefly explain the MACE filter design.

Suppose we have n training images, each of size $(d \times d)$. First, the 2-D FTs of these training images are computed and resulting complex arrays are vectorized into columns of a $(d^2 \text{ x } n)$ complex valued matrix **X**. We also use a $(d^2 \text{ x } d^2)$ diagonal matrix **D** whose diagonal entries are the average power spectrum of the n training images. Since **D** is diagonal, we need to store only its diagonal entries and not the complete matrix. The filter is represented by a column vector **h** with d^2 elements. Finally, the filter **h** is required to produce prespecified values u_i at the correlation origin in response to the training images $i = 1, 2, \ldots, n$ and these constraints can be expressed as follows:

$$X^{+}h = u \tag{45}$$

where **u** = [u$_1$ u$_2$ … u$_N$]^T and superscripts T and + denote the transpose and conjugate transpose, respectively. The closed form equation for the MACE filter **h** is

$$h = D^{-1}X\left(X^{+}D^{-1}X\right)^{-1}u \tag{46}$$

```
% Read training data
for i = 1:nsamples
    imstr = [path,f,num2str(i),'.pgm'];
    im = double(imread(imstr, 'pgm'));

    im = imresize(im, [64 64],'bilinear');

    [r,c] = size(im);
    % stack into col matrix
    Xi = fft2(im);
    X(:,i) = Xi(:);

    % perform FFT
    Di = Di + X(:,i).*conj(X(:,i));

end

fprintf('Starting to analyze data\n');

Dave = abs(Di/nsamples);
u = ones(nsamples,1);
Dinv = diag(1./Dave);
h = Dinv*X*inv(X'*Dinv*X)*u;
% h = X*inv(X'*X)*u;

h = reshape(h, [d d]);
im = double(imread('.\mike\5.pgm', 'pgm'));
im = imresize(im, [64 64], 'bilinear');

fprintf('Performing correlation\n');
imf = fft2(im);
corr = abs(ifftshift(ifft2((imf.*conj(h))./abs(imf.*conj(h)))));
figure, mesh(corr)
```

Figure 22: Code snippet for MACE filter design.

The correlation outputs exhibit sharp, high peaks for the authentic and no such peaks for the impostors (Fig. 23). The peak sharpness can be quantified by the peak-to-sidelobe ratio (PSR) defined in Fig. 23, where peak is the largest value in the correlation output and *mean* and *std* are the average value and the standard deviation of the correlation outputs in an annular region (size *20 x 20*) centered on the peak but excluding the peak region (a *5 x 5* region).

Task 18: Duplicate the filter responses in Fig. 21 for authentic and and impostor images, with the PSR values.

Task 19: Perform *leaving-one-out* cross-validation of the correlation pattern recognition approach (MACE filters) using the Yale database.

Task 20: Compare the results of PCA, LDA, ICA and correlation pattern recognition (MACE filters) using their CMC curves with different distance measures.

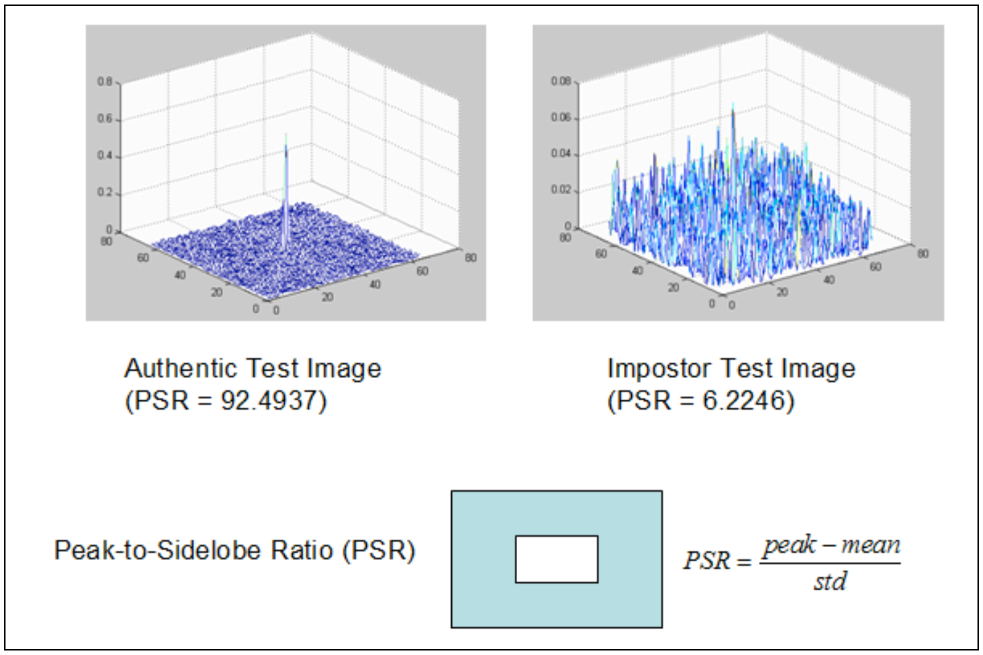

Figure 23: Correlation outputs for authentic and impostor inputs.

References

[1] Yale Face Database, < http://cvc.yale.edu/projects/yalefaces/yalefaces.html >

[2] Intel OpenCV, < http://www.intel.com/technology/computing/opencv/ >

[3] P. Belhumuer, J. Hespanha, and D. Kriegman, "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection," IEEE Trans. Pattern Analysis and Machine Intelligence, 19(7): 711-720, 1997

[4] B. V. Kumar, M. Savvides, and C. Xie, "Correlation Pattern Recognition for Face Recognition," Proc. of the IEEE, Nov. 2006

[5] Ham Rara, DIMENSIONALITY REDUCTION TECHNIQUES IN FACE RECOGNITION, Master thesis, CVIP Lab, University of Louisville, March 24, 2006

[6] X. Lu, "Image Analysis for Face Recognition," Personal Notes, May 2003, http://www.face-rec.org/interesting-papers/General/ImAna4FacRcg_lu.pdf

[7] A. Hyvarinen, J. Karhunen, and E. Oja, "Independent Component Analysis," Wiley, 2001

[8] B. Draper, et. al., "Recognizing faces with PCA and ICA," Computer Vision and Image Understanding 91 (1-2), 115-137

[9] M.S. Bartlett, J.R. Movellan, and T.J. Sejnowski, "Face recognition by independent component analysis," IEEE Transactions on Neural Networks 13 (2002) 1450-1464

[10] D. M. Blackburn, "Face Recognition 101: A Brief Primer," < http://www.frvt.org/DLs/FR101.pdf >