Facial Detection
CAP 6411: Computer Vision Systems

Jaime Soto

December 7, 2010

Contents

1 Assignment Overview

Download one or more of the labeled face detection databases in Table 14.2. Generate your own negative examples by finding photographs that do not contain any people. Implement following face detectors:

1. Boosting (Algorithm 14.1) [5] based on simple area features, with an optional cascade of detectors [7].
2. PCA face subspace [3]

2 AdaBoost

2.1 Questions

- How did you select the threshold for weak classifiers?
- How many weak classifiers did you use?
- Which weak classifiers were finally selected in the strong classifier?
- Did you use integral images?
- Did you use cascade?
- What is the overall performance?

2.2 Comments

3 Principal Component Analysis

Principal component analysis (PCA) is a simple method for facial detection and recognition. PCA takes advantage of the similarities between faces to produce a series of templates called eigenfaces during a training phase. A facial image can then be approximated through a weighted sum of the eigenfaces. Similarity between two facial images can be computed as a distance measure.

3.1 Face Detector Training

The training images (Γ_i) are a set of MIT CBCL frontal face database [2]. Some implementations include a pre-processing histogram equalization of each training image to standarize the brightness and contrast [1]. Each of the N training images is represented as a column in a training set matrix Γ. Each of the training images is normalized (Φ_i) by subtracting the mean image Ψ [4]:

$$\Psi = \frac{1}{M}\sum_{i}^{M}\Gamma_i \tag{1}$$

$$\Phi_i = \Gamma_i - \Psi \tag{2}$$

The next step requires computing the eigenvalues and eigenvectors of the covariance matrix C of Φ. The dimensions of C will be $M^2 \times M^2$ if each column in Φ contains M elements. Therefore, C will have M^2 eigenvectors and eigenvalues. The images in the MIT CBCL database have 19×19 pixels, so there are $19^4 = 130321$ eigenvectors and eigenvalues for C. However, it is possible to obtain N eigenvectors and eigenvalues without computing C explicitly. The covariance matrix is defined as:

$$C = \frac{1}{N} \sum_{i}^{N} \Phi_i \Phi_i^T = \Phi \Phi^T \tag{3}$$

The size of $L = \Phi^T \Phi$ is $N \times N$. Consider the eigenvalue decomposition of $\Phi^T \Phi$ instead:

$$\Phi^T \Phi \mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{4}$$

Where $\mathbf{v}_i$ is an eigenvector of L and λ_i is its corresponding eigenvalue. If we pre-multiply both sides by Φ we obtain:

$$\Phi \Phi^T \Phi \mathbf{v}_i = \lambda_i \Phi \mathbf{v}_i \tag{5}$$

Notice that $\Phi \mathbf{v}_i$ are the eigenvectors of $C = \Phi \Phi^T$. We can now find N eigenfaces $\mathbf{u}_i$ by computing:

$$\mathbf{u}_i = \sum_{k}^{M} \mathbf{v}_{lk} \Phi_k, l = 1, ..., M \tag{6}$$

This reduces calculations significantly since the size of the training set is often significantly smaller than the square of the number of pixels per training image ($N \ll M^2$) [6]. When sorted by descending eigenvalue order, many of the last eigenfaces will not contribute significantly and can therefore be ignored [1].

The training algorithm was implemented in MATLAB as the `pca_face_train` function:

```
function pca_face_train(directory, pattern, tsize, pca_face_db, nimages = -1, ncomponents = 30)

% Creates a face recognition/detection database from a set of the training images.
% All the training images must have the same size.
%
% Arguments:
%   directory:    a string with the path to the directory that contains the
%                 training images.
%   pattern:      a string with the file pattern that matches the training image
%                 file names, e.g. '*.pgm'.
%   tsize:        a two-element vector with the size of the training images,
%                 e.g. [19 19] for 19 x 19 px.
%   pca_face_db:  a string with the path of the file where the database will
%                 be stored.
%   nimages:      a scalar with the number of training images to consider. Reads
```

```
%                all files that match @pattern in @directory by default.
%   ncomponents:  the number of components (eigenfaces) to include in the database.
%
% (c) Jaime Soto
% CAP 6411 - Computer Vision Systems
% University of Central Florida
% 7 December 2010
%

    workingdir = pwd();
    cd(directory);
    filelist = dir(pattern);
    numrows = tsize(1);
    numcols = tsize(2);

    if (nimages == -1)
    nimages = length(filelist)
    else
    nimages = min(nimages, length(filelist));
    end

    images = zeros(numrows*numcols, nimages);

    % For each image in directory
    for f = 1:nimages
    image = imread(filelist(f).name);

        if (rows(image) ~= numrows || columns(image) ~=numcols)
            fprintf(1, 'Invalid image dimensions: %s (%d, %d)\n', ...
             filelist(f).name, rows(image), columns(image));
            continue;
        end

    if (isrgb(image))
    image = rgb2gray(image);
    end

    %images(:,f) = double(image(:));

        % Optional histogram equalization - commented out for speedup
    % Unfold image into a single column
    % Scale values to (0,1)
    % Perform histogram equalization
    % Rescale to (0, 255)
    images(:,f) = histeq(double(image(:))./255.0).*255.0;
    end

    cd(workingdir);

    % Compute mean image
    mean_img = mean(images')';

    % Subtract mean from each image
    images = images - repmat(mean_img, [1 nimages]);

    % Obtain eigenvectors and eigenvalues
    % Use method proposed by Turk and Pentland (1991)
```

```
    [eigenvectors eigenvalues] = eig(images' * images);

    % Pick the eigenvectors that correspond to the largest eigenvalues
    eigenvectors = eigenvectors(:, 1:ncomponents);

    % Calculate image components
    eigenfaces = images * eigenvectors;
    components = zeros(ncomponents,nimages);

    for i = 1:nimages
    components(:,i) = eigenfaces' * images(:,i);
    end

    save(pca_face_db, 'mean_img', 'eigenfaces', 'components');
end
```

3.2 Training Image Reconstruction

A training image can be easily reconstructed from its components and the eigenfaces:

```
function img = pca_face_reconstruct(index, pca_face_db)

% Reconstructs a training image from a face recognition database.
%
% Arguments:
%   index:        a scalar with the index of the requested training image.
%   pca_face_db:  a string with the path of the file where the database will
%                 be stored.
%
% Returns:
%   A column vector with the pixels of the reconstructed training image.
%
% (c) Jaime Soto
% CAP 6411 - Computer Vision Systems
% University of Central Florida
% 7 December 2010
%

    load(pca_face_db);
    c = components(:,index);
    img = eigenfaces*c + mean_img;
end
```

3.3 Face Detector Testing

A new image Γ is transformed into eigenspace components ω_k by multiplying the transpose of the eigenfaces $\mathbf{u}_k$ to the difference between it and the mean face image:

$$\omega_k = \mathbf{u}_k^T(\Gamma - \Psi) \tag{7}$$

A distance measure between sets of components denotes the similarity between them [6]. This distance can be considered to have to components: a

distance in face space (DIFS) and a distance from face space (DFFS). In this implementation, the DIFS is the Mahalanobis distance from the components of the test image to all the components of the training images. The DFFS is an Euclidean distance from the mean-normalized test image to its corresponding reconstructed image. Figure 1 shows the distribution of DIFS vs. DFFS for 1000 faces and 1000 non-faces. Based on these results, a DIFS threshold of 7.75 (on log scale) was selected for facial detection. However, it is recommended to use probabilistic models to better predict the boundary between the classifications [3].

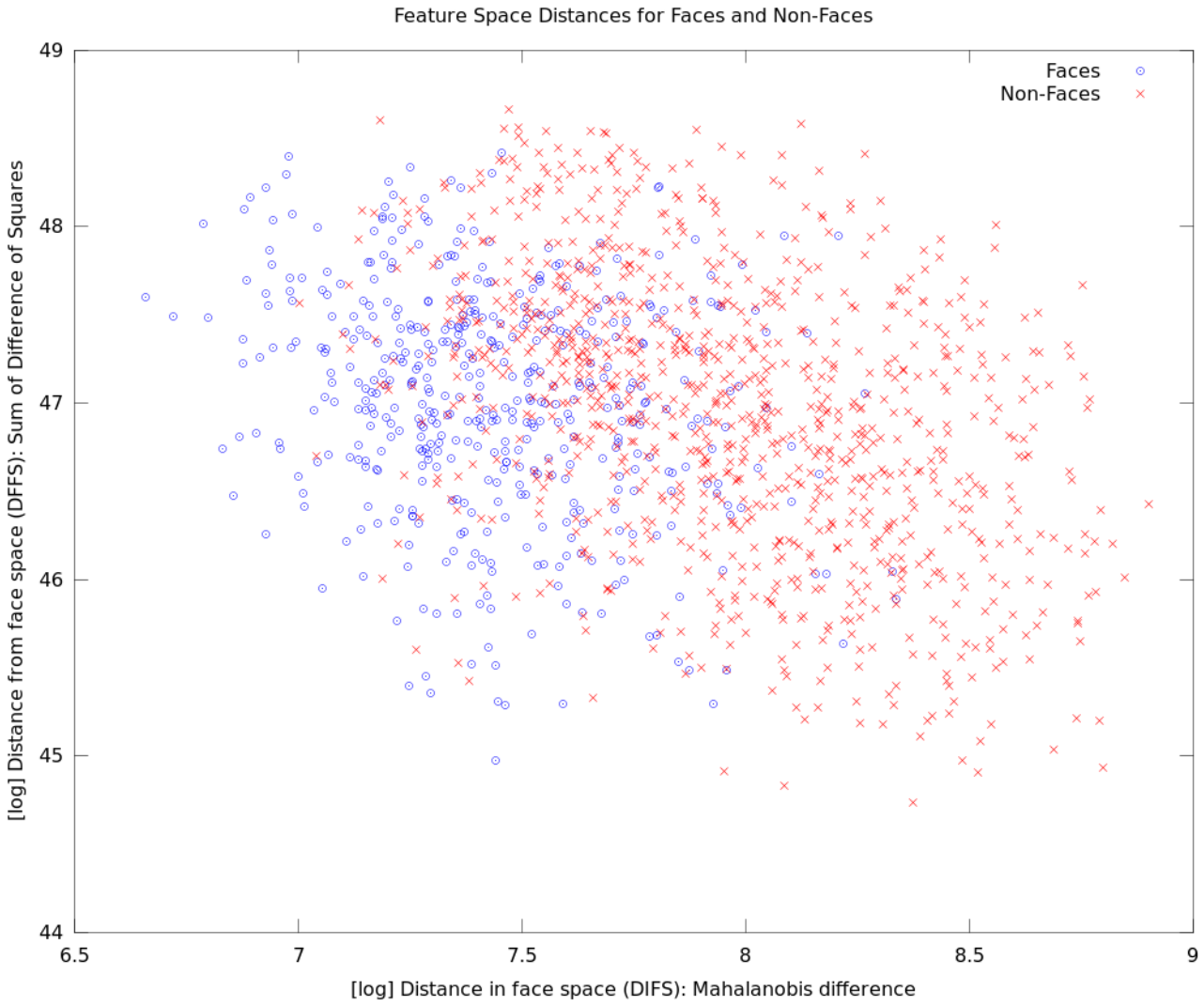

Figure 1: DIFS and DFFS measures for 1000 faces and 1000 non-faces

The face detection algorithm was implemented in MATLAB as the `pca_face_detect` function:

```
function [difs, dffs] = pca_face_detect(directory, pattern, pca_face_db, nimages = -1)

% Performs face detection for a set of test images using a face recognition database.
% All the training images must have the same size as those used in the training
% stage of the face recognition database.
%
% Arguments:
%   directory:    a string with the path to the directory that contains the
%                 test images.
%   pattern:      a string with the file pattern that matches the test image
%                 file names, e.g. '*.pgm'.
%   pca_face_db:  a string with the path of the PCA face database.
%   nimages:      a scalar with the number of test images to consider. Reads
%                 all files that match @pattern in @directory by default.
% Returns:
%   difs:         a column vector with the distances in face space in log scale.
%   dffs:         a column vector with the distances from face space in log scale.
%
% (c) Jaime Soto
% CAP 6411 - Computer Vision Systems
% University of Central Florida
% 7 December 2010
%
    load(pca_face_db);
    workingdir = pwd();
    cd(directory);
    filelist = dir(pattern);

    if (nimages == -1)
    nimages = length(filelist)
    else
    nimages = min(nimages, length(filelist));
    end

    difs = zeros(nimages, 1);
    dffs = zeros(nimages, 1);
    face = zeros(nimages, 1);

    for f = 1:nimages
    image = imread(filelist(f).name);

        if (length(image(:)) ~= length(mean_img))
            continue;
        end

    if (isrgb(image))
    image = rgb2gray(image);
    end

    image = double(image);
    image = histeq(image./255.0).*255.0;

        % Project image to face space
    c = eigenfaces' * (image(:) - mean_img);
```

```
    % Reconstruct the image
    r = eigenfaces*c + mean_img;

    % Compute the distance from face space and distance in face space
    % (Moghaddam and Pentland 1997)
    dffs(f) = log(sum(double(image(:).^2) - r.^2));
    difs(f) = log(mahalanobis(c', components'));
    end

    cd(workingdir);
end
```

3.4 Demonstration

PCA face detection is demonstrated in MATLAB with the `pca_face_demo` function:

```
function pca_face_demo(traindir, facedir, nonfacedir)

% Demonstrates the PCA face recognition/detection algorithm.
%
% Arguments:
%   traindir:     a string with the path to the directory that contains the
%                 training images.
%
%   facedir:      a string with the path to the directory that contains the
%                 face test images.
%
%   nonfacedir:   a string with the path to the directory that contains the
%                 non-face test images.
%
% (c) Jaime Soto
% CAP 6411 - Computer Vision Systems
% University of Central Florida
% 7 December 2010
%
    pca_face_db = 'pca_face_db.mat';
tic();
pca_face_train(traindir, 'face0*.pgm', [19 19], pca_face_db, 200, 200);
fprintf(1, 'training took %f sec\n', toc());

tic();
[fdifs fdffs] = pca_face_detect(facedir, 'cmu_0*.pgm', pca_face_db, 1000);
fprintf(1, 'detection on %d faces took %f sec\n', length(fdifs), toc());

tic();
[ndifs ndffs] = pca_face_detect(nonfacedir, 'cmu_0*.pgm', pca_face_db, 1000);
fprintf(1, 'detection on %d non-faces took %f sec\n', length(ndifs), toc());

threshold = 7.75;
face = fdifs < threshold;
nface = ndifs < threshold;

fprintf(1, 'True positives: %d\n', nnz(face));
fprintf(1, 'True negatives: %d\n', nnz(nface));
fprintf(1, 'False positives: %d\n', nnz(~nface));
```

```
fprintf(1, 'False negatives: %d\n', nnz(~face));
clf
hold on
title('Feature Space Distances for Faces and Non-Faces');
plot(fdifs, fdffs, 'ob');
plot(ndifs, ndffs, 'xr');
legend('Faces', 'Non-Faces');
xlabel('[log] Distance in face space (DIFS): Mahalanobis difference');
ylabel('[log] Distance from face space (DFFS): Sum of Difference of Squares');
print('facespace_distances.png', '-dpng');
hold off
end
```

References

[1] S. Emami. Face Detection and Face Recognition with Real-time Training from a Camera. `http://www.shervinemami.co.cc/faceRecognition.html`, 2010.

[2] MIT Center for Biological and Computational Learning. CBCL Software: Face Data. `http://cbcl.mit.edu/software-datasets/FaceData2.html`, 2000.

[3] B. Moghaddam and A. Pentland. Probabilistic visual recognition for object recognition. *Trans. IEEE Pattern Analysis and Machine Intelligence*, 19(7):696–710, 1997.

[4] D. Pissarenko. Eigenface-based facial recognition. `http://openbio.sourceforge.net/resources/eigenfaces/eigenfaces.pdf`, 2003.

[5] R. Szeliski. Computer Vision: Algorithms and Applications (August 5, 2010 draft). `http://szeliski.org/Book/`, 2010.

[6] M. Turk and A. Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71 – 86, 1991.

[7] P. Viola and M.J. Jones. Robust Real-Time Face Detection. *International Journal of Computer Vision*, 57(2):137 – 154, 2004.