APPLIED MATHS

PROJECT REPORT

Face Recognition

January 6, 2017

Yeman Brhane Hagos
Vu Hoang Minh

1 INTRODUCTION

For over decade, Face Recognition (FR) has become a popular area of research in computer vision to identify and verify a face from a picture or a frame from a video compared to a stored database of faces [1]. The applications of FR can be listed as (1) security systems, (2) commercial identification and (3) marketing tool [3].

One approach for FR problem is using Principal ComponentAanalysis (PCA) for computing eigenfaces derived from the covariance matrix of the probability distribution over the high-dimensional of faces proposed by Turk and Pentland in 1991 [4].

In this project, we apply PCA for FR.

2 METHODOLOGY

The FR problem using PCA comprises two major steps:

1. Normalization
2. Face Recognition

2.1 Normalization

Recall that the position of a face can be presented by six parameters: three rotations and three translations. In different scenes, a detected face in a frame may encounter two issues: (1) Great amount of variance in its appearance and (2) Lighting difference caused by different light sources. Thus, in order to solve the problem of Normalization, two problems above need to to be resolved subsequently.

Normalization

The purpose of Normalization is prior to recognition to account for scale, orientation, and location variations. The main idea of Normalization is to transform all faces with given facial features to map to predetermined ones in 64×64 window. In specific, the locations of these features include:

- Left eye center
- Right eye center
- Tip of nose
- Left mouth corner
- Right mouth corner

Overall, the procedure of Normalization is shown in Algorithm 1.

Algorithm 1 Normalization

Input: $images$
Output: $normalized\_images$

procedure normalization($images$)
  $\bar{F} \leftarrow F_1$
  apply SVD
  compute transformation matrix, M, aligning $\bar{F}$ with determined position
  while $error > \epsilon$ **do**
    $\bar{F}_{t-1} = \bar{F}$
    for each image **do**
      compute M_i
      $F'_i \leftarrow M_i F_i$
      $\bar{F} \leftarrow \frac{1}{N} \sum_{i=1}^{N} F'_i$
    $\bar{F}_t = \bar{F}$
    $error \leftarrow \bar{F}_t - \bar{F}_{t-1}$
  for each image **do**
    save F_i as normalized image

Light Correction

In FR, illumination variance is another issue. It is know that image variation due to lighting changes is larger than that due to different personal identity [5]. As a result, in different circumstances with different light conditions, a face would have different intensity.

To work out this issue, some light correction methods to account for non-uniform illumination are also applied. One of them is Singluar Value Decomposition (SVD). The following model can be used:

$$f(x, y) = ax + by + cxy + d \tag{1}$$

However, the dataset we are working on can be considered as perfect illumination because of similar lighting condition and camera. Consequently, we do not have to apply Light Correction in this project.

2.2 Recognition

After normalization and light correction, FR process can be examined with the purpose of finding the best matched face in Train set with a given face in Test set using PCA.

In general, the procedure of FR is shown in Algorithm 2.

FR starts with concatenating p training images with size 64×64 to obtain a matrix D. Next, PCA is performed to achieve converted images called *eigenfaces*.

Algorithm 2 Face Recognition

Input: $image$
Output: $matched\_images$
1: **procedure** recognizeface($image$)
2: concatenate p training 64×64 images to obtain D
3: $\Sigma \leftarrow \frac{1}{p-1} D^T D$
4: extract k principle components to get Φ
5: compute projected train images $\phi_i \leftarrow X_i \cdot \Phi$
6: compute projected test image $\phi_T \leftarrow X_T \cdot \Phi$
7: compute difference between ϕ_T and each ϕ_i
8: find minimum difference to get matched image

3 IMPLEMENTATION

3.1 Normalization

We begin the process by checking the correctness of images and feature text files. A true image and its corresponding text file should satisfy the following conditions:

- Text file has exact five points, which is mentioned in Subsection 2.1, with both x and y coordinates
- Text file has true facial features compared to corresponding image
- Except five features points mentioned above, there would be no useless information. To be clear, the text file should be consistent.

In fact, there is a number of text files having incapable information; thus making debugging process be longer.

After that, we check the accuracy of each image/text file, we face another problem with inconsistency in naming i.e some files begin with capital, some are not, which occurs the problem in finding corresponding text file. Thus, a function to rename all of image and text files to lowercase, for example *aBC1* will become *abc1*, was created.

Then, Algorithm 1 is applied to obtain normalized images.

During the Normalization process, we have faced a difficulty in transforming given facial features to predetermined points. In specific, *imtransform* function apply to an additional, unspecified, translation when computing the new image. When we use this syntax, *imtransform* automatically shifts the origin of our output image to make as much of the transformed image visible as possible. Thus, we have to examine the *XData* and *YData* input parameters explicitly.

Overall, the list of functions listed in Appendice are shown as follows with specific goals.

1. Main
    - ***main_normalize*** main function in Normalization

2. Rename image and text files
    - ***renameimageandfeature***: rename images and their corresponding texts to lower case
3. Transform and normalize all images
    - ***transformImages*** transform all images to map their facial features to a computed mean matrix
    - ***normalizeimages*** iterate and compute mean aligned features
4. Compute error, mean facial features and transformation matrix
    - ***computetransarray***: apply SVD method to estimate six parameters of the affine transformation matrix
    - ***computemeanfacialfeatures***: compute mean facial features with given predefined locations and feature directory
    - ***computeerror*** compute the error between F_t and F_{t-1}
5. Read feature text file
    - ***readfeaturetext***: read one text file of facial features
    - ***readallfeaturetext***: read and save all facial features in M x 2N matrix
6. Light correction
    - ***correctlightoneimage***: correct light of one image

3.2 Face Recognition

3.3 Graphical User Interface

4 GRAPHICAL USER INTERFACE

raphical user interface.

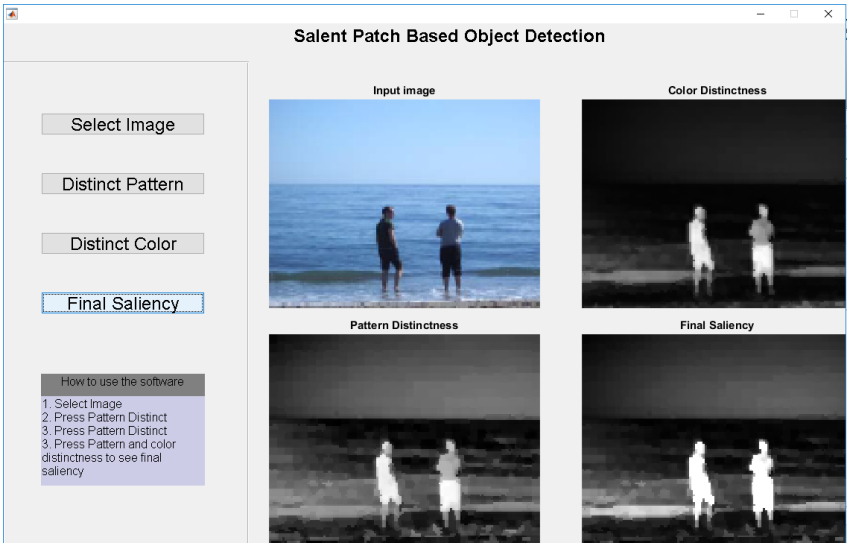

Figure 1: Graphical User Interface

REFERENCES

[1] Roberto Brunelli and Tomaso Poggio. Face recognition: Features versus templates. *IEEE transactions on pattern analysis and machine intelligence*, 15(10):1042–1052, 1993.

[2] Ran Margolin, Ayellet Tal, and Lihi Zelnik-Manor. What makes a patch distinct? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1139–1146, 2013.

[3] Liton Chandra Paul and Abdulla Al Sumam. Face recognition using principal component analysis method. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, 1(1):135–139.

[4] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

[5] Haitao Wang, Stan Z Li, Yangsheng Wang, and Weiwei Zhang. Illumination modeling and normalization for face recognition. In *Analysis and Modeling of Faces and Gestures, 2003. AMFG 2003. IEEE International Workshop on*, pages 104–111. IEEE, 2003.

APPENDICES

Sample code: applying known Priors

```
function [ finalSliancy ] = combine_prior( PatternSalience , colorSalience)
%============================================================
% developed by:
%                 Yeman Brhane Hagos
%                 Minh Vu
%============================================================
%function [ finalSliancy ] = combine_prior( PatternSalience , colorSalience)

%Input parameters are:
%[1] PatternSalience from pattern distinctness
%[2] colorSalience from color distinctness
%
%
%Ouputs are:
%[1] finalSliancy
%
% the final saliency is comuted as follows
%1. we combine pattern and color saliency
%2. Incorpoprating known priors
%1. we note that the salient pixels tend to be grouped together into clusters,
    as they typically
%correspond to real objects in the scene. this is done computing
%centermass of the salient object and puting gaussian at that point
% 2. Photographers put salient object at center : this is done by
% puting Gaussian at centerof the combined saliency image

%% Combining pattern and color salience

combinedSaliancy= PatternSalience .* colorSalience;

% Normalization
combinedSaliancyNor = combinedSaliancy / max (max (combinedSaliancy));
% figure
% imshow (combinedSaliancy , [])
% title ('Comined Salience ');

%% Incorpoprating known priors
%% 1. we note that the salient pixels tend to be grouped together into
    clusters, as they typically
```

```
%correspond to real objects in the scene
[ N, M]= size (combinedSaliancy);
level = graythresh(combinedSaliancyNor);
BW = im2bw(combinedSaliancyNor,level);
%          figure
%          imshow(BW)
% Find Center mass of the salient object and place Gaussian
[row , col]=find (BW==1);
% Center Mass
Center=round ([ sum(row), sum(col)]/ length(row));
% Determine size of Gaussian
x = N-Center(1);
y= M-Center(2);
r= max(Center(1), x);
c= max(Center(2), y);
% Generate Gaussian with center at center mass
gaussian= fspecial('gaussian', [ 2*r 2*c], 1000);
gaussian=gaussian(r+1-Center(1):r+x,c+1-Center(2):c+y);
% size(h)
%  figure
%  imshow(h, [])
SaliencyImage=combinedSaliancy.*( gaussian);
%% 2. Photographers put salient object at center : Gaussian at center
%gausswidth= max (size (combinedSaliancy));
% determine sigma
%sigma= (gausswidth - 1)/ 6; % this one is bluring the pixels at edge

% generate guassian at center
gaussianCenter = fspecial('gaussian', size (SaliencyImage), 150);

finalSliancy = SaliencyImage .* (5 * gaussianCenter);

%          figure
%         imshow (finalSliancy , [])
%          title ('final Salience ');
end
```