NUS
National University of Singapore

EE 5907 Pattern Recognize

CA2

YU XIAOCHEN

A0132158L

Introduction

For this assignment, we are request to construct a face recognition system. The dataset is provided, and it needs to be separated as training set and test set. We are going to use four algorithms such as PCA, LDA, SVM, GMM and CNN to achieve our purpose.

Data Process

I selected the 1 - 25 subjects in provided PIE folder, and split it into training set and test set. The first 119 images will be group as training set, the rest will be test test. I took 10 selfie-photo and resize it to 32 x 32, and put them into folder 0. The first 7 images will be use as selfie_train, the rest 3 images are selfie_test.

PCA

The main process is to find the eigen vector and the eigen value of the training set, so we can get the eigen face. Use KNN to find the nearest distance, and set the label of test data as the nearest train data.

Step1:

Find the mean of dataset:

$$\underset{D\times 1}{\bar{X}_{mean}} = \frac{1}{N}\sum_{n=1}^{N} \underset{D\times 1}{\bar{X}_n}$$

Step2:

generate the S-matrix and use SVD to get eigenvector and eigenvalue.

$$\underset{D\times D}{S} = \frac{1}{N}\sum_{n} \underbrace{(\bar{X}_n - \bar{X}_{mean})}_{D\times 1}\underbrace{(\bar{X}_n - \bar{X}_{mean})^T}_{1\times D}$$

```
%generate S matrix
s_matrix = (train_diff * train_diff') / size(TrainSet,2);
[U, D, V] = svd(s_matrix);
```

Step3:

Construct the eigenface:

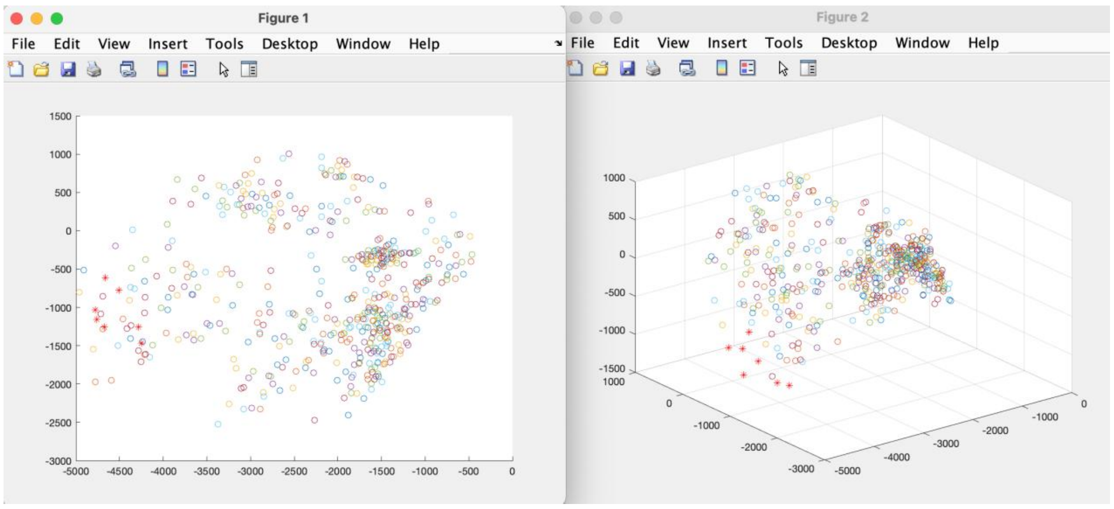
$$\underbrace{\tilde{X}_n}_{D\times 1} = \underbrace{\bar{X}_{mean}}_{D\times 1} + \underbrace{U}_{D\times M}\,\underbrace{U^T}_{M\times D}\left(\underbrace{\bar{X}_n - \bar{X}_{mean}}_{D\times 1}\right)$$

```
figure(5);title('eigenface-200');
hold on;
eigenface_200d = U(:,200)*U(:,200)'*train_diff(:,200)+train_mean;
eigenface_200d = reshape(eigenface_200d,[32,32]);
imshow (mat2gray(eigenface_200d))
```

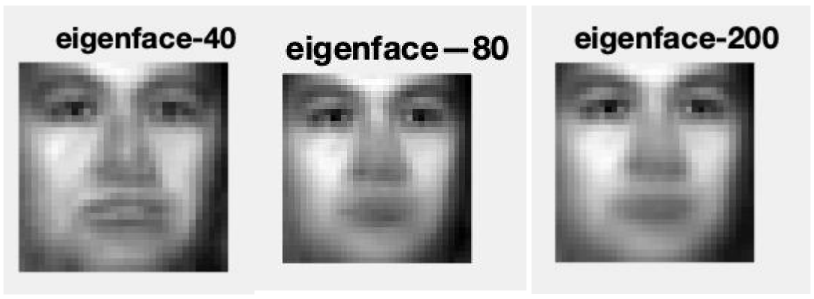
Result:

- Visualization:

The random 500 samples are shown below, and the 7 red * are represent 7 selfie image.



- Eigenface for dimension 40, 80 and 200:



- PIE and selfie error rates for dimension = 40, 80 and 200:

dimension	40	80	200
PIE	79.45%	87.76%	92.94%
Selfie	66.67%	66.67%	66.67%

LDA

LDA is similar to pca, try to find the fisher face instead of the eigen face, and use KNN to get the nearest distance.

Step 1:

Find the variance within class scatter sw:

$$S_w = \sum_{i=1}^{C} \frac{n_i}{N} S_i = \sum_{i=1}^{C} \frac{n_i}{N} \left(\frac{1}{n_i} \sum_{x \in C_i} (x - u_i)\ (x - u_i)^T \right)$$

Step 2:

Find the variance between class sb:

$$S_B = \frac{1}{N} \sum_{i=1}^{C} n_i (u_i - u)(u_i - u)^T$$

Step 3:

Compute the fisher face by find the eigenvalues of (SW-1*SB)

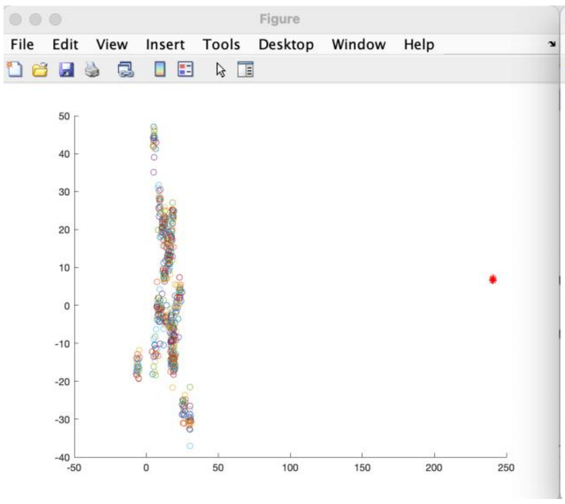
$$\frac{d}{dw} J(w) = (w^T S_w w) S_B w - (w^T S_B w) S_w w = 0$$

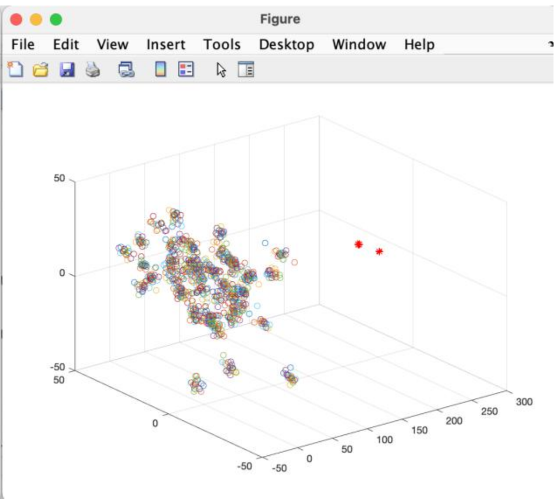
$$S_w{}^{-1} S_B w = J(w) w$$

Result:

- Visualization:

The random 500 samples are shown below, The 7 red * are represent 7 selfie image.





- PIE and selfie error rates for dimension = 2, 3 and 9:

dimension	2	3	9
PIE	14.27%	30.43%	77.1%
Selfie	0%	0%	0%

SVM

According to the libsvm library instruction, the t set to 0 is linear kernel, and we need to change the penalty cost by update the value of c .

```
c = [0.01,0.1,1];

model_linear_80 = svmtrain(TrainLabel,ptrain_80,'-t 0 -c 1');
model_linear_200 = svmtrain(TrainLabel,ptrain_200,'-t 0 -c 1');
[label_80, accuracy_80, values_80] = svmpredict(TestLabel,ptest_80,model_linear_80);
[label_200, accuracy_200, values_200] = svmpredict(TestLabel,ptest_200,model_linear_200);
```

Result:

- Dimension for 80 and 200' error rates for C = 0.01, 0.1 and 1:

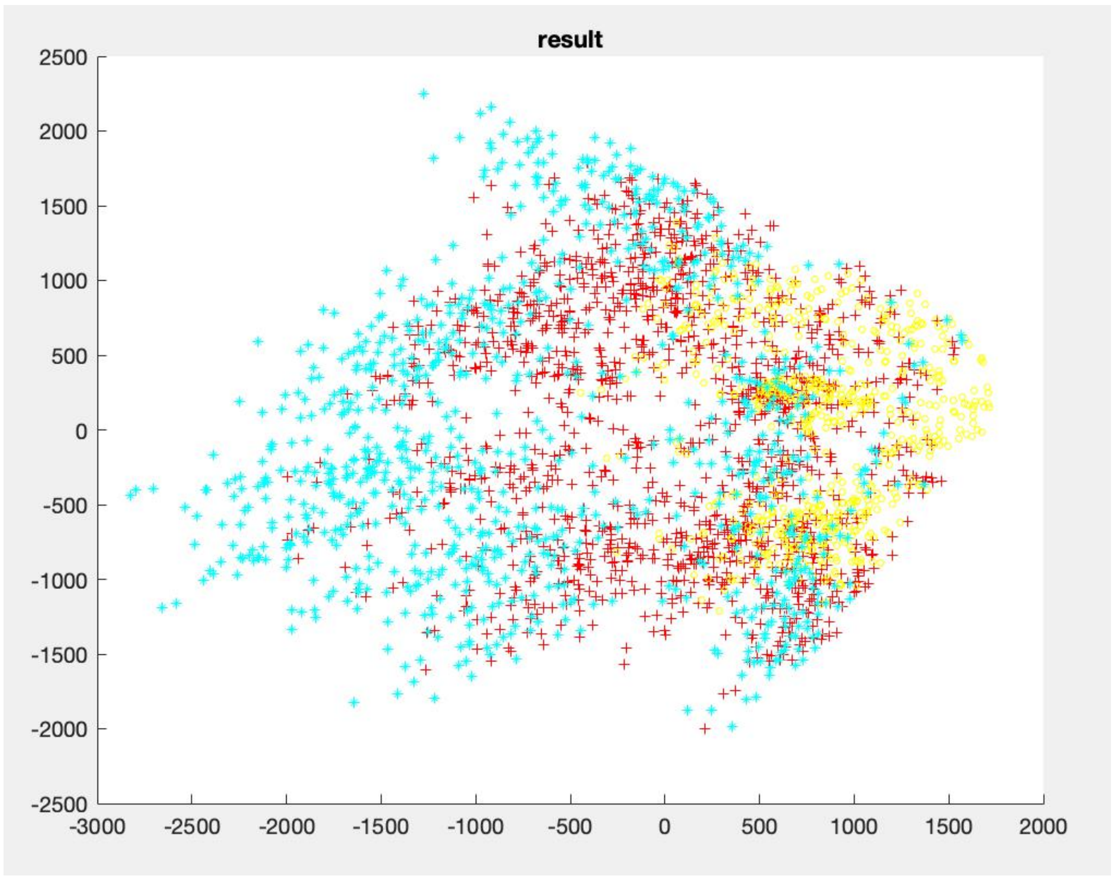
	C = 0.01	C = 0.1	C = 1
Dimension = 80	14.27%	30.43%	77.1%
Dimension = 200	0%	0%	0%

GMM

Step

Result:

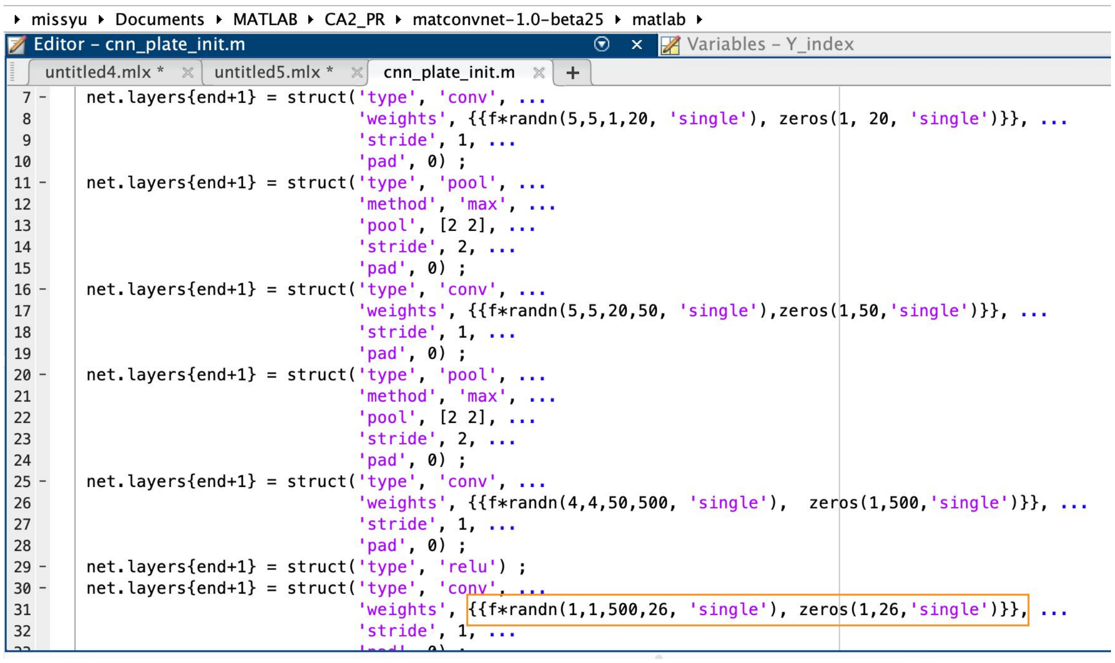
- Visualization:



CNN:

Step1:

Defined the nodes as 20-50-500-26:

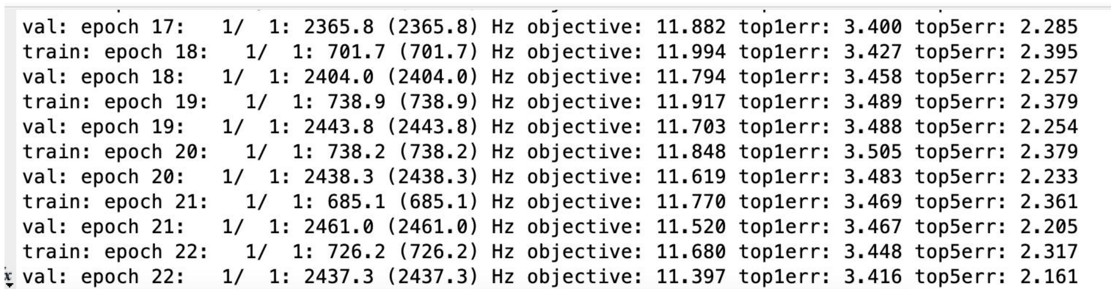


missyu ▸ Documents ▸ MATLAB ▸ CA2_PR ▸ matconvnet-1.0-beta25 ▸ matlab ▸
Editor – cnn_plate_init.m
Variables – Y_index
untitled4.mlx *
untitled5.mlx *
cnn_plate_init.m

```matlab
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(5,5,1,20, 'single'), zeros(1, 20, 'single')}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'pool', ...
                           'method', 'max', ...
                           'pool', [2 2], ...
                           'stride', 2, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(5,5,20,50, 'single'),zeros(1,50,'single')}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'pool', ...
                           'method', 'max', ...
                           'pool', [2 2], ...
                           'stride', 2, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(4,4,50,500, 'single'),  zeros(1,500,'single')}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'relu') ;
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(1,1,500,26, 'single'), zeros(1,26,'single')}}, ...
                           'stride', 1, ...
```

Step 2:

Start to training the data:



```
val: epoch 17:   1/  1: 2365.8 (2365.8) Hz objective: 11.882 top1err: 3.400 top5err: 2.285
train: epoch 18:   1/  1: 701.7 (701.7) Hz objective: 11.994 top1err: 3.427 top5err: 2.395
val: epoch 18:   1/  1: 2404.0 (2404.0) Hz objective: 11.794 top1err: 3.458 top5err: 2.257
train: epoch 19:   1/  1: 738.9 (738.9) Hz objective: 11.917 top1err: 3.489 top5err: 2.379
val: epoch 19:   1/  1: 2443.8 (2443.8) Hz objective: 11.703 top1err: 3.488 top5err: 2.254
train: epoch 20:   1/  1: 738.2 (738.2) Hz objective: 11.848 top1err: 3.505 top5err: 2.379
val: epoch 20:   1/  1: 2438.3 (2438.3) Hz objective: 11.619 top1err: 3.483 top5err: 2.233
train: epoch 21:   1/  1: 685.1 (685.1) Hz objective: 11.770 top1err: 3.469 top5err: 2.361
val: epoch 21:   1/  1: 2461.0 (2461.0) Hz objective: 11.520 top1err: 3.467 top5err: 2.205
train: epoch 22:   1/  1: 726.2 (726.2) Hz objective: 11.680 top1err: 3.448 top5err: 2.317
val: epoch 22:   1/  1: 2437.3 (2437.3) Hz objective: 11.397 top1err: 3.416 top5err: 2.161
```

Trained data are saved in matconvnet-1.0-beta25/data/plate-baseline/

net-epoch-22.mat

Result:

For 20-50-500-26:

The accuracy is 97.839.

For 10-50-500-26:

The accuracy is 97.773.

For 50-50-500-26:

The accuracy is 97.743.