National University of Singapore

EE5907: Pattern Recognition
Face Recognition

CA2 Report

Li Haipeng (A0260034W)

11/11/2022

CONTENTS

1. Dataset

The dataset concludes 25 different subjects out of CMU PIE dataset and 10 personal face photos. The personal face photos are converted to gray-scale images and resized to (32,32). For each chosen subject, use 70% of the provided images for training and use the remaining 30% for testing.

The basic dataset preprocess code is in the dataset.ipynb. The information of the dataset is shown below.

Table 1.1 Dataset informations

data_idx	[1, 2, 4, 7, 13, 14, 16, 17, 22, 23, 26, 27, 29, 33, 36, 43, 47, 50, 52, 53, 57, 58, 63, 66, 67]
Number of PIE images	4250
Number of PIE train images	2975
Number of PIE test images	1275
Number of self images	10
Number of self train images	7
Number of self test images	3
Number of whole train images	2982

The raw face images will be converted to 1024 dimensional vector by the function get_img_vector.

2. PCA for Feature Extraction, Visualization and Classification

a) PCA based data distribution visualization

Randomly sample 500 images from the CMU PIE training set and your own photos. Apply PCA to reduce the dimensionality of vectorized images to 2 and 3 respectively. Visualize the projected data vector in 2d and 3d plots. Highlight the projected points corresponding to your photo.

From the 2d plot, we can find that the red points are in the other points, which mean the self-photos are difficult to be classified if we only reduce the dimensionality to 2. From the 3d plot we can find that it is easy to draw the boundary between the self-photos and the CMU PIE images. However, it is still difficult to classify the 25 different classes of CMU PIE datasets

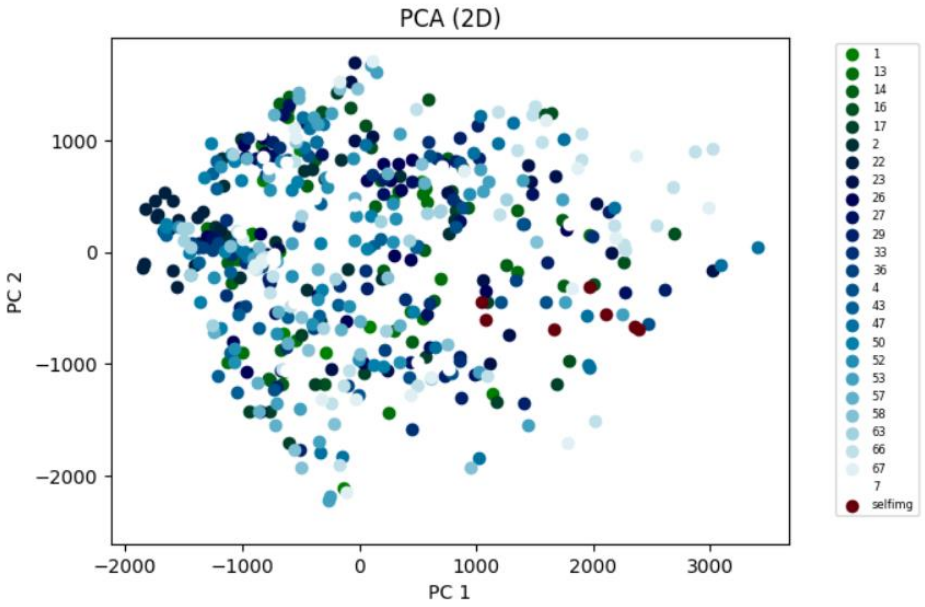

Fig 2. 1 Visualize the projected data vector in 2d plots

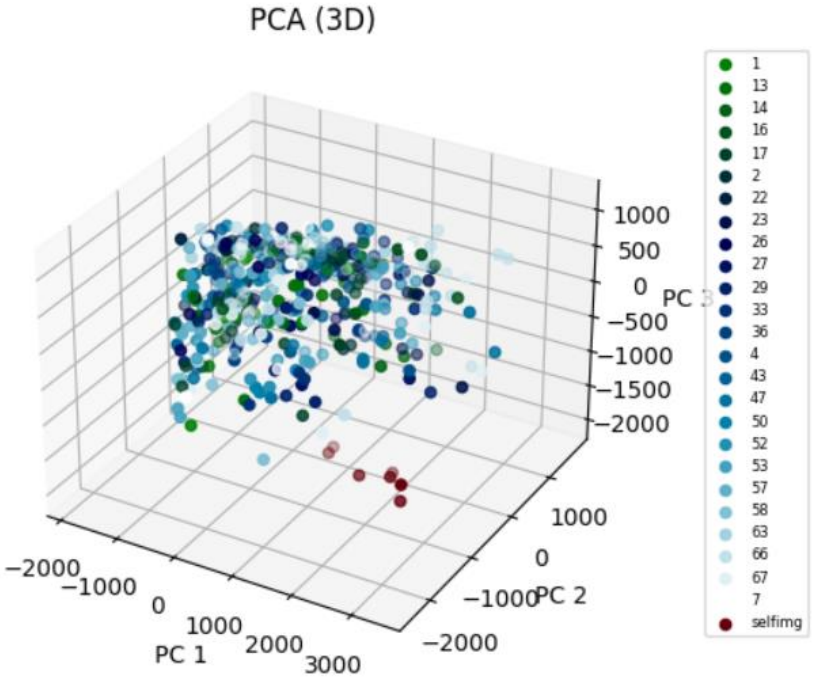

Fig 2. 2 Visualize the projected data vector in 3d plots

Visualize the corresponding 3 eigenfaces used for the dimensionality reduction. This three eigenfaces means the three most significant components extracted from the whole training image features.

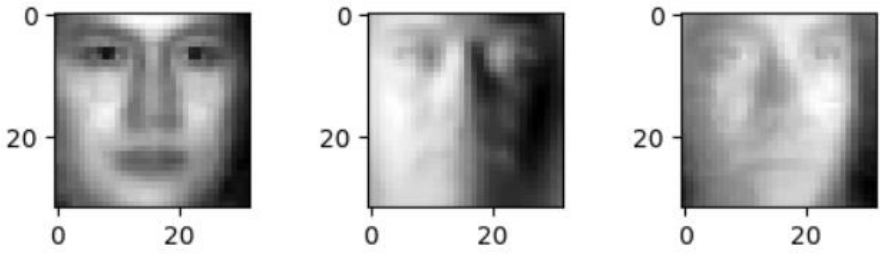

Fig 2. 3 Visualize the corresponding 3 eigenfaces

b) PCA plus nearest neighbor classification results

From the result, we can find the classification accuracy of self-photos is 100%, which is consistent with previous analysis in the 3d plot. With the larger dimensionality, there will be more principal components, and the accuracy will increase. After some discussion with other students, my test result is quite good, and it may be had something to do with the choice of the dataset.

Table 2. 1 Classification accuracy of PCA

Dimensionality	CMU PIE	Self
40	94.98%	100%
80	96.55%	100%
200	96.86%	100%

3. LDA for Feature Extraction and Classification

a) LDA based data distribution visualization

Apply LDA to reduce data dimensionality from to 2, 3 and 9. Visualize distribution of the sampled data (as in the PCA section) with dimensionality of 2 and 3 respectively (similar to PCA). The dataset concludes 25 classes of CMU PIE images and 10 self-photos.

From the 2d and 3d plot, we can find the points of same class will gather together. Especially for the self-photos, the points almost overlap on each other, so it is easy to classify the self-photos from both plot. Besides with the dimensionality increases to 3, we can see from the 3d plot that the clustering effect is quite clear.

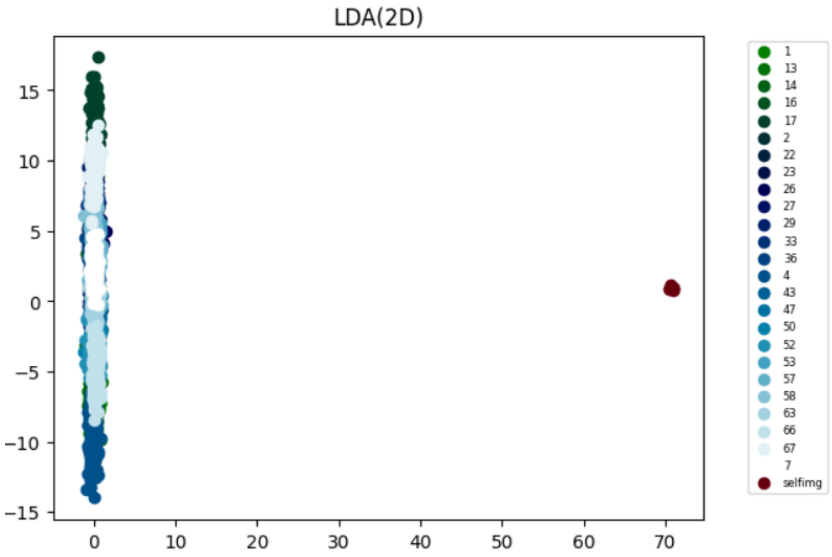

Fig 3. 1 Visualize the projected data vector in 2d plots

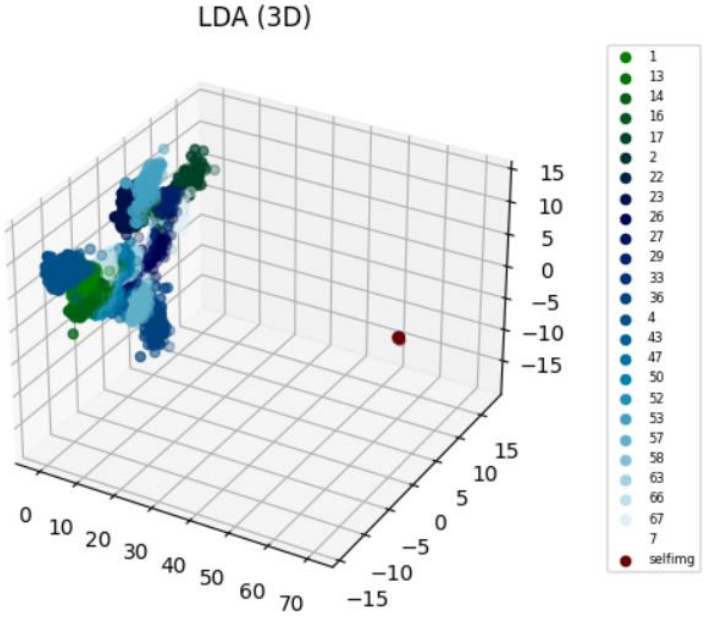

Fig 3. 2 Visualize the projected data vector in 3d plots

b) LDA plus nearest neighbor classification results

From the result we can find it is easy to classify the self-photos, which is the same with what we find from the plots. With the increase of dimensionality, the classification accuracy of CMU PIE test data will increase. Especially when the dimensionality comes to 9, the accuracy is already over 90%, which means that compared to PCA, LDA can use low data dimensionality to get a high accuracy.

Table 3. 1 Classification accuracy of LDA

Dimensionality	CMU PIE	Self
2	44.71%	100.0%
3	60.78%	100.0%
9	92.39%	100.0%

4. SVM for Classification

Use the raw face images (vectorized) and the face vectors after PCA pre-processing (with dimensionality D of 80 and 200) as inputs to linear SVM. Try different penalty parameter C values of 0.01, 0.10, 1.00. The dataset concludes 25 classes of CMU PIE images and 10 self-photos. The result is shown below.

Theoretically, when the penalty parameter C is big, the tolerance for the wrong samples at the boundary will be lower and the fitting degree of the samples will be higher. On the contrary, when C is small, it is more likely to make wrong classification when training.

However, from the result, we find that the accuracy (with dimensionality of 80 and 200) is same and all quite high when we change the value of C.

From the table, we can notice that the more dimensionality, the higher accuracy will be. When dimensionality is 200, the accuracy is almost the same with the raw face images.

Table 4. 1 Classification accuracy of SVM

C	D = 10	D = 80	D = 200	Raw face images
0.01	77.543%	98.9828%	99.2958%	99.2958%
0.10	77.3865%	98.9828%	99.2958%	99.2958%
1.00	74.3349%	98.9828%	99.2958%	99.2958%

5. Neural Networks for Classification (CNN)

Train a CNN with two convolutional layers and one fully connected layer, with the architecture specified as follows: number of nodes: 20-50-500-21. The number of the nodes in the last layer is fixed as 26 as we are performing 21-category (25 CMU PIE faces plus 1 for yourself) classification. Convolutional kernel sizes are set as 5. Each convolutional layer is followed by a max pooling layer with a kernel size of 2 and stride of 2. The fully connected layer is followed by ReLU. Train the network and report the final classification performance.

Use tensorflow to create the CNN model. Reshape the data to (, 32, 32, 3) by the function get_img().The result is shown below.

We can find that after two epochs, the training accuracy is over 90%. The accuracy of the testing set is 98.45%, which is the best among these four methods.

```
cnn_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=20,kernel_size=5,padding='same',input_shape=(32, 32, 3),activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    tf.keras.layers.Conv2D(filters=50,kernel_size=5,padding='same',activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(500, activation='relu'),
    tf.keras.layers.Dense(26,activation='softmax')])
```

Fig 5. 1 CNN model deployment

```
Epoch 1/10
90/90 [==============================] - 2s 23ms/step - loss: 7.0477 - accuracy: 0.4893
Epoch 2/10
90/90 [==============================] - 2s 22ms/step - loss: 0.2870 - accuracy: 0.9260
Epoch 3/10
90/90 [==============================] - 2s 22ms/step - loss: 0.1441 - accuracy: 0.9630
Epoch 4/10
90/90 [==============================] - 2s 23ms/step - loss: 0.0748 - accuracy: 0.9836
Epoch 5/10
90/90 [==============================] - 2s 23ms/step - loss: 0.0304 - accuracy: 0.9913
Epoch 6/10
90/90 [==============================] - 2s 22ms/step - loss: 0.0291 - accuracy: 0.9923
Epoch 7/10
90/90 [==============================] - 2s 22ms/step - loss: 0.0311 - accuracy: 0.9920
Epoch 8/10
90/90 [==============================] - 2s 22ms/step - loss: 0.0803 - accuracy: 0.9783
Epoch 9/10
90/90 [==============================] - 2s 23ms/step - loss: 0.0671 - accuracy: 0.9829
Epoch 10/10
90/90 [==============================] - 2s 22ms/step - loss: 0.0346 - accuracy: 0.9909
39/39 - 0s - loss: 0.0550 - accuracy: 0.9845 - 311ms/epoch - 8ms/step
The loss is:  0.05500802770256996
The accuracy is: 98.45%
```

Fig 5. 2 Train and test result of CNN model

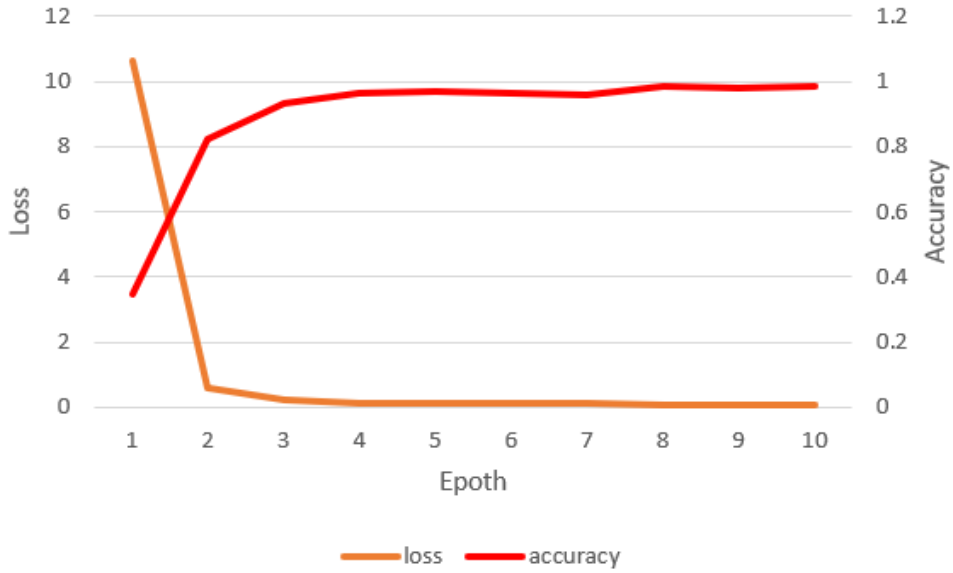

Fig 5. 3 Training loss curves and accuracy curve of the CNN model

Change the network architectures, and use VGG network to train our dataset. The VGG network was proposed by Visual Geometry Group from University of Oxford in 2015[1]. The ConvNet configurations of VGG network is shown in the table below. Choose A configurations, which contains 11 weight layers. Considering that our input is only 32×32 RGB image, so remove 2 max pooling layers. The model deployment is shown in Fig 5.4.

Table 5.1 ConvNet configurations [1]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

```
# change network architectures
VGG11_model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters=64,kernel_size=3,padding='same',input_shape=(32, 32, 3),activation='relu'),
    #tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    tf.keras.layers.Conv2D(filters=128,kernel_size=3,padding='same',activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='same',activation='relu'),
    tf.keras.layers.Conv2D(filters=256,kernel_size=3,padding='same',activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',activation='relu'),
    tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',activation='relu'),
    tf.keras.layers.MaxPool2D(pool_size=2, strides=2),
    tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',activation='relu'),
    tf.keras.layers.Conv2D(filters=512,kernel_size=3,padding='same',activation='relu'),
    #tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(2048, activation='relu'),
    tf.keras.layers.Dropout(rate=0.5),
    tf.keras.layers.Dense(2048, activation='relu'),
    tf.keras.layers.Dense(26,activation='softmax')])
```

Fig 5.4 VGG network model deployment

```
Epoch 1/15
90/90 [==============================] - 55s 601ms/step - loss: 3.9546 - accuracy: 0.0451
Epoch 2/15
90/90 [==============================] - 54s 603ms/step - loss: 3.0515 - accuracy: 0.0877
Epoch 3/15
90/90 [==============================] - 56s 626ms/step - loss: 2.1060 - accuracy: 0.3238
Epoch 4/15
90/90 [==============================] - 61s 682ms/step - loss: 1.2594 - accuracy: 0.5917
Epoch 5/15
90/90 [==============================] - 58s 645ms/step - loss: 0.9157 - accuracy: 0.6944
Epoch 6/15
90/90 [==============================] - 57s 638ms/step - loss: 0.6760 - accuracy: 0.7831
Epoch 7/15
90/90 [==============================] - 58s 640ms/step - loss: 0.4809 - accuracy: 0.8428
Epoch 8/15
90/90 [==============================] - 58s 649ms/step - loss: 0.2988 - accuracy: 0.9008
Epoch 9/15
90/90 [==============================] - 59s 650ms/step - loss: 0.3099 - accuracy: 0.9064
Epoch 10/15
90/90 [==============================] - 60s 670ms/step - loss: 0.2962 - accuracy: 0.9043
Epoch 11/15
90/90 [==============================] - 60s 664ms/step - loss: 0.2080 - accuracy: 0.9410
Epoch 12/15
90/90 [==============================] - 59s 655ms/step - loss: 0.1936 - accuracy: 0.9448
Epoch 13/15
90/90 [==============================] - 58s 645ms/step - loss: 0.2090 - accuracy: 0.9420
Epoch 14/15
90/90 [==============================] - 57s 636ms/step - loss: 0.2327 - accuracy: 0.9322
Epoch 15/15
90/90 [==============================] - 57s 635ms/step - loss: 0.1611 - accuracy: 0.9528
39/39 - 4s - loss: 0.0635 - accuracy: 0.9813 - 4s/epoch - 95ms/step
The loss is:  0.06349285691976547
The accuracy is: 98.13%
```

Fig 5.5 Train and test result of VGG network model

From the result we can see that VGG network is more time-consuming because of its bigger and more convolution layer. The accuracy is high after 10 epochs and accuracy on the test set is 98.13%, which is the same with the simple CNN model. However, our datasets are small and simple, which only contains 2863 training data and the size of input data only $32 \times 32 \times 3$. Therefore, it is unsuitable to use VGG network model for our dataset.

References

[1] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).