Contents

1 Set-ups **2**

2 Preprocessing by PCA **3**

2.1 Visualisation 5

3 Training by SVM **5**

3.1 Evaluation 6

4 Training by CNN **8**

4.1 Codes 9

4.2 Visualisation 11

5 Conclusion **11**

5.1 Improvements 11

5.2 Further study 12

5.3 Reflection 13

6 References **14**

1 Set-ups

In this essay we address facial recognition by SVM and CNN. The focus of the essay is the codes to solve the problem but not the theory behind.

The language of codes in this essay is Python 3.8. The following libraries are used:

1. Scikit-learn 0.23.2
2. NumPy 1.18.5
3. Matplotlib 3.3.3
4. TensorFlow 2.3.1

For codes in this essay, methods and functions will be bold, attributes will be italic for easier reading.

The dataset we use in this essay is the **Labeled Faces in the Wild** (LFW) people dataset. It contains 13233 images of 5749 people of dimension 62×47. It also contains labels of each image, i.e. the person who is in the image. For simplicity we focus on classifying people with more than 70 faces in the dataset. We run the code below to extract those data:

Algorithm 1: Data_extraction.py

```
Result: data.txt, label.txt
import numpy as np
from sklearn.datasets import fetch_lfw_people

lfw_people = fetch_lfw_people(min_faces_per_person = 70, resize = 0.4)
X = lfw_people.data
y = lfw_people.target
target_names = lfw_people.target_names

num_person = len(target_names)

data = np.column_stack((X,y))
np.savetxt('data.txt',data)

f = open('label.txt','w')
for i in range(0,num_person) do
 |  f.write('%s\ n' % target_names[i])
end
f.close()
```

By reading these data, we have the following useful basic statistics:

$$\text{n\_samples} = 1288$$
$$\text{n\_features} = 1850$$
$$\text{n\_classes} = 7.$$

When we try to train models below, we can simply retrieve data from the text files:

Algorithm 2: Program fragment — data retrieval

Result: X, y, n_samples, n_features, n_classes

```
import numpy as np
data = np.loadtxt('data.txt')
X = data[:, 0 : −1]
y = data[:, −1]

target_names = []
f = open('label.txt','r',newline="\n")
lines = np.loadtxt(f, dtype=str, delimiter=",")
for line in lines do
 |  target_names.append(line)
end
f.close()

(n_samples, n_features) = X.shape
n_classes = len(target_names)
```

2 Preprocessing by PCA

Here we use PCA to reduce the dimension of images.

PCA is an unsupervised algorithm that can be implemented easily using SVD. Apart from dimension reduction, it also has useful properties such that zero mean, variance maximization and decorrelating features. By further multiplication by a diagonal matrix, it has the effect of whitening. This sometimes helps increase the accuracy of the program

we train. Here we imlpement PCA using sklearn:

Algorithm 3: Program fragment — PCA

Result: X_train_pca
Require: X_train,y_train
Require: n_components

From sklearn.decomposition **import** PCA

pca = **PCA**(*n_components*=n_components,
svd_solver='randomized',*whiten*=True).**fit**(X_train)
X_train_pca = pca.**transform**(X_train)

The same transformation can be done to X_test. Here we have a hyperparameter n_components, which is the number of components to be kept in PCA. We would discuss the choice of hyperparameters later.

Another dimension reduction technique is K-means clustering. In this method, we replace pixels of similar colour by a representative. The number of representative is the number of clusters K. An image of dimension $m \times n$ can be seen as a $mn \times 1$ vector, i.e. mn samples, each with one feature (colour). Note when we apply clustering for image compression, our algorithm depends solely on the image to be compressed, so this may not be appropriate for future supervised learning and we shall not use it here.

2.1 Visualisation

We can visualise the outputs of PCA by matplotlib. This gives us a clear picture of what is going on in the dataset. This can be done with the code below:

Algorithm 4: Program fragment — Visualisation with matplotlib

Result: Figures
Require: pca in program fragment above
Require: h, w, the dimension of image

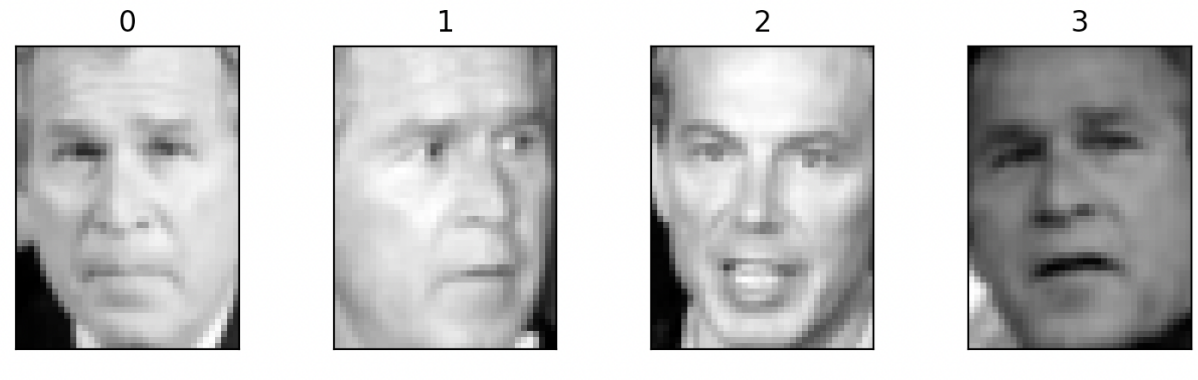
```
import matplotlib.pyplot as plt

eigenfaces = pca.components_.reshape((n_components, h, w))

# A helper function
def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
for i in range(n_row * n_col) do
    plt.subplot(n_row, n_col, i + 1)
    plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
    plt.title(titles[i], size=12)
    plt.xticks(())
    plt.yticks(())
end
```

This shows how PCA reduces the dimension of the images. Below are some examples using n_component = 125:

This is the original picture:

3 Training by SVM

Here we use SVM to train an algorithm to classify different faces. It is clear that SVM is a powerful algorithm in classification. We can also use it for multi-class classification by training N classfiers for N classes to implement one-vs-all approach. In the scope of our dataset, SVM may perform better than algorithms that base on bayesian approach as we



have only a small dataset, so other typical algorithms may not have enough samples to determine all parameters, causing overfitting.

Apart from that, SVM is also easy to implement using sklearn. We use a RBF kernel for SVM. The code is given below:

Algorithm 5: Program fragment — SVM

Result: C, γ, parameters of kernel
Require: X_train_pca, y_train

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42)

param_grid = 'C': [1e3, 5e3, 1e4, 5e4, 1e5],
          'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1],
clf = GridSearchCV(
  SVC(kernel='rbf', class_weight='balanced'), param_grid
)
clf = clf.fit(X_train_pca, y_train)
```

Running the code shows that the best pair in the two sets of values is

$$C = 1000, \gamma = 0.5.$$

Note in the code, we use train_test_split to split our dataset into two parts as we only get those images in the dataset, so we partition the set to obtain a test set for validation.

3.1 Evaluation

There are many performance metrics to evaluate the performance of an algorithm, so we just use classification report and confusion matrix in sklearn libraries to display those statistics.

The classification report consists of two parts:

1. Precision, recall and F1-score for each class.

2. Accuracy, unweighted average and weighted average of the algorithm.

The first part of the report describes performance of the algorithm in each class and the second part describes the overall performance.

The i, j-th entry of the confusion matrix C is defined as the number of samples classified as class j but their true label is class i. That is, the diagonal entries of C are the number of correctly classified images.

The code is given here:

Algorithm 6: Program fragment — Evaluation

Result: Classification report and confusion matrix
Require: X_test_pca, y_test, target_names, n_classes

from sklearn.metrics **import** classification_report
from sklearn.metrics **import** confusion_matrix

y_pred = clf.**predict**(X_test_pca)

print(**classification_report**(y_test, y_pred, *target_names*=target_names))
print(**confusion_matrix**(y_test, y_pred, *labels*=**range**(n_classes)))

Below are statistics for hyperparameter n_component = 150 and 200 respectively.

1.

	precision	recall	f1-score	support
Ariel Sharon	0.78	0.54	0.64	13
Colin Powell	0.83	0.87	0.85	60
Donald Rumsfeld	0.85	0.63	0.72	27
George W Bush	0.83	0.98	0.90	146
Gerhard Schroeder	0.95	0.80	0.87	25
Hugo Chavez	1.00	0.47	0.64	15
Tony Blair	0.97	0.81	0.88	36
accuracy			0.85	322
macro avg	0.89	0.73	0.78	322
weighted avg	0.86	0.85	0.85	322

$$\begin{pmatrix} 7 & 1 & 0 & 5 & 0 & 0 & 0 \\ 1 & 52 & 1 & 6 & 0 & 0 & 0 \\ 1 & 2 & 17 & 7 & 0 & 0 & 0 \\ 0 & 3 & 0 & 143 & 0 & 0 & 0 \\ 0 & 1 & 0 & 3 & 20 & 0 & 1 \\ 0 & 3 & 0 & 4 & 1 & 7 & 0 \\ 0 & 1 & 2 & 4 & 0 & 0 & 29 \end{pmatrix}$$

2.

	precision	recall	f1-score	support
Ariel Sharon	0.59	0.77	0.67	13
Colin Powell	0.74	0.87	0.80	60
Donald Rumsfeld	0.95	0.70	0.81	27
George W Bush	0.88	0.90	0.89	146
Gerhard Schroeder	0.81	0.68	0.74	25
Hugo Chavez	0.73	0.53	0.62	15
Tony Blair	0.85	0.81	0.83	36
accuracy			0.83	322
macro avg	0.79	0.75	0.76	322
weighted avg	0.83	0.83	0.83	322

$$\begin{pmatrix} 10 & 1 & 0 & 2 & 0 & 0 & 0 \\ 2 & 52 & 1 & 4 & 0 & 0 & 0 \\ 3 & 3 & 19 & 2 & 0 & 0 & 0 \\ 2 & 9 & 0 & 131 & 2 & 1 & 1 \\ 0 & 2 & 0 & 3 & 17 & 1 & 2 \\ 0 & 3 & 0 & 2 & 0 & 8 & 2 \\ 0 & 0 & 0 & 5 & 2 & 0 & 29 \end{pmatrix}$$

We see that in this case, n_component = 150 is a better choice than 200.

4 Training by CNN

Apart from SVM, another popular classifier is convolutional neural network (CNN). We first apply convolution layer to extract important features of faces of different people, say the shape of face, the position of eyes and nose. We then apply some dense layers to classify those features. In this section, we will use the unprocessed data. Otherwise, the purpose of PCA and convolution may overlap, causing loss of important information. Nonetheless, CNN is kind of "black-box", i.e. we know it works but we do not know why it works. Then what we can do is trial and error. We try different combinations of

layers and hidden units to determine the best network. Here we will use TensorFlow for training, and the code will be given after the explanation in this subsection.

We first need to have convolution layer to capture important features, or discarding unnecessary and repeating features. This allows us to train less parameters in later sequential network to reduce the chance of overfitting as well as the training time and memory consumed. However, there is an underlying assumption in the use of convolution — we assume that those important features of an image will be preserved under slight translation. In other words, we assume the features are almost the same in a very small disk on the image. In the convolution layer, we further apply max pooling, batch normalisation and dropout to make our model more robust and having less features to be fit. After convolution layer, we pass the processed data to dense layers for classification. Note we only have a small amount of data, so we may easily overfit the model if we have too many parameters.

After determining the structures of the network, we need to choose the loss function, the optimisation method and the evaluation metric. Here we will use the one suggested by TensorFlow tutorial:

1. Loss function: Sparse Categorical Cross entropy,
2. Optimiser: Adam,
3. Metric: Accuracy.

4.1 Codes

By trial and errors, we can have many different models that lead to similar results. One of the model is given below as a sample code. It consists of two convolution layers and two dense layers. Note each sample in X_train and X_test here have a different shape — (h,w,1) instead of (1,1850), where h and w are the height and width of the original picture.

In the sample, we have two convolution layers and four dense layers. After running the code, we obtain the following result:

$$\text{loss: } 0.642988383769989, \ \text{Accuracy: } 0.8385093212127686,$$

which is similar to the one using PCA and SVM.

Metrics such as F1-score and balanced accuracy are not provided by TensorFlow, so

we can compute them ourselves.

Algorithm 7: CNN sample

Result: θ, set of parameters of the network
Require: X_train, y_train, X_test, y_test
Require: h,w

```
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers,datasets,models
from tensorflow.keras.models import Sequential

model = models.Sequential()

model.add(layers.Conv2D(4, (3, 3), activation='relu', input_shape=(h,w,1)
        ,kernel_regularizer=keras.regularizers.l2(0.03)))
model.add(keras.layers.BatchNormalization())

model.add(layers.Conv2D(5, (2, 2), activation='relu'
        ,kernel_regularizer=keras.regularizers.l2(0.03)))
model.add(keras.layers.BatchNormalization())
model.add(layers.MaxPooling2D((2, 2)))

model.add(keras.layers.Dropout(0.5))
model.add(layers.Flatten())

model.add(layers.Dense(14))
model.add(layers.Dense(16))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(n_classes))

model.compile(optimizer='adam'
  ,loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
        metrics=['accuracy'])

history = model.fit(X_train, y_train, epochs=15, validation_data=(X_test, y_test))
```

In the algorithm, we use two consecutive linear layers with small number of hidden units. This allows us to train less parameters that may achieve a similar result using one linear layer with large amount of hidden units. Consider a linear layer with n inputs and p outputs. We need np parameters to train a single $n \times p$ weight matrix W. When we write $W = UV$ for some $n \times q$ matrix U and $q \times p$ matrix V, we need $(n+p)q$ parameters to train two weight matrices. That is, when q is small, this can be a considerable saving of parameters.

To further improve the accuracy of the algorithm, we can try more structures of the network. For example, we can change the number of hidden units in each layer, the size of filters in convolution as well as the depth of the network.

4.2 Visualisation

We will construct an error plot to visualise the change in errors during training. The code is given below.

Algorithm 8: Program fragment — error plot

Require: history

```
plt.figure()

plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

plt.show()
```

The figure corresponding to the sample code is given here:
We see that the sample network is not that good because overfitting seems to occur.

5 Conclusion

As seen from the discussion before, we need to do more to improve the two algorithms. After studying this algorithm, we may have some further development in this topic.

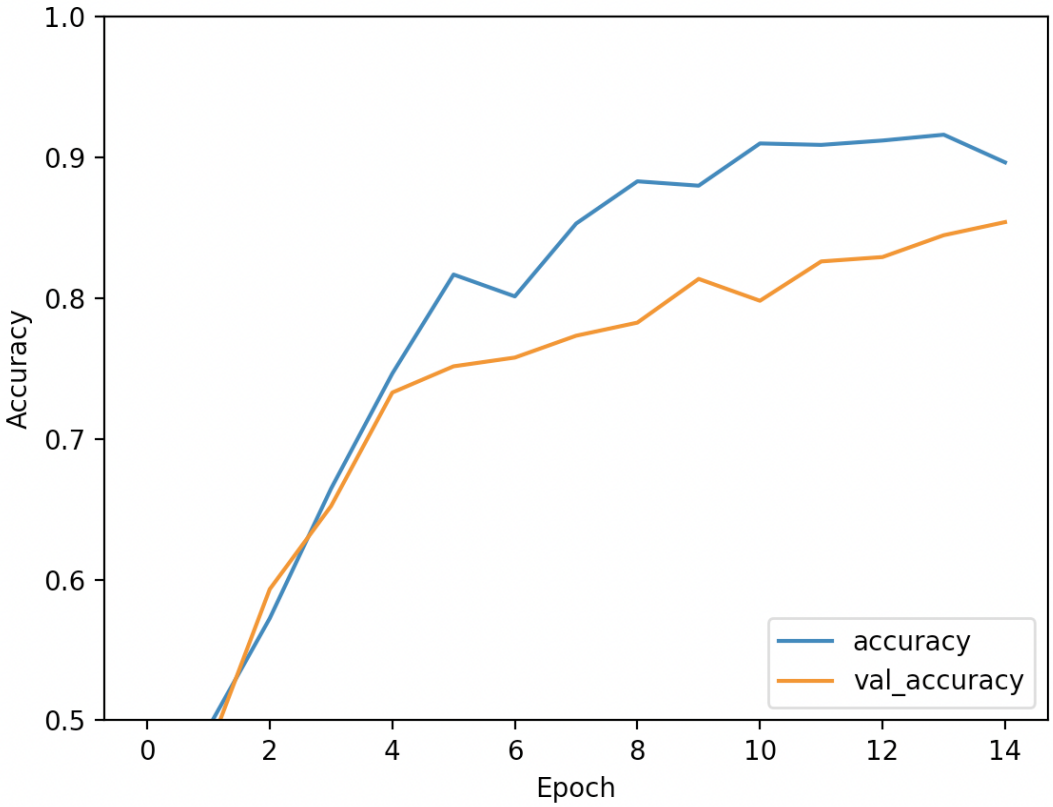
5.1 Improvements

For SVM, we can do the followings:

1. Tune the hyperparameters, say n_components, choice of kernel and class weights.
2. Try different combinations of algorithm. In this essay we try PCA+SVM, but we can also have PCA+logistic or other matrix decompositions+SVM.

For CNN, we can do the followings:

1. Change the structure of the network, say number of hidden units in each layer, depth of network and activation functions.



2. Tune the hyperparameters, say regularization constant, epoches and batches.

Last but not least, we can also improve by observing the dataset. The training set used in this essay is rather imbalanced. We have 13 images of Ariel Sharon but 146 images for George W Bush. We can make the dataset more balanced by resampling techniques, say having copies of images of people who have only a few images, applying class weights to each class and using other performance metric say balanced accuracy and F1-score.

5.2 Further study

The algorithm derived in this essay can only be used to identify the identity of the person shown in the photo. What can be done next is to make this more practical by developing an algorithm that identifies the presence of a person. The combination of the two algorithms allows us to carry out facial recognition in a more general scope. However, this is another lengthy and complicated task.

Another way for further study is to apply facial recognition to more people or smaller dataset. This is yet another challenging task as we may have relatively less samples for each class to train our model.

5.3 Reflection

Although this essay does not address the theory part of machine learning, it gives an illustration of how machine learning can be applied to data science task. Compared to the theories taught in class, application of machine learning feels far more easier as those libraries are well-developed and optimized. One just need to understand those commands and implement the model via API. What's important is that the theories taught in the class allow us to efficiently design and select appropriate models and parameters.

6 References

Here lists the reference of this essay. Codes in this essay are not attached, but similar codes can be found on references.

1. Convolutional Neural Network (CNN) : TensorFlow Core. (n.d.). Retrieved December 02, 2020, from

   https://www.tensorflow.org/tutorials/images/cnn

2. Faces recognition example using eigenfaces and SVMs. (n.d.). Retrieved December 02, 2020, from

   https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html