Pattern Analysis

Lab Demonstration 2
Pattern Recognition

Shekhar “Shakes” Chandra

In this lab, we will study dimensionality reduction and classification as a way of learning the basics of Tensorflow. The foundation of Tensorflow (TF) is based on linear algebra in a similar manner to Numpy. As a consequence, a number of functions from Numpy are also available in TF.

The lab is partitioned into three main parts. Firstly, we will look to introduce PCA of human faces and classification using the random forest, where you will be asked to create a TF equivalent program. Then use TF to build a traditional convolutional neural network (CNN) approach to do the same. Finally, you will have a chance to implement a CNN solution of your choice from a list of problems and data in order to solve an example recognition problem.

1 Part 1 - Eigenfaces (5 Marks)

We will compute Eigenfaces - the PCA of human faces using Numpy and the funnelled “Labeled Faces in the Wild” (LFW). Scikit learn Eigenfaces example is modified below to show how the PCA model of faces is constructed using Numpy arrays.

First load the relevant data and functions

```
In [59]: from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import numpy as np
```

```
In [60]: # Download the data, if not already on disk and load it as numpy arrays
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
```

Extract the meaningful parameters of the faces dataset

```
In [50]: # introspect the images arrays to find the shapes (for plotting)
n_samples, h, w = lfw_people.images.shape

# for machine learning we use the 2 data directly (as relative pixel
# positions info is ignored by this model)
X = lfw_people.data
n_features = X.shape[1]
```

```
# the label to predict is the id of the person
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]

print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes)
```

It is important in machine learning to split the data accordingly into training and testing sets to avoid contamination of the model. Ideally, you should also have a validation set.

```
In [51]: # Split into a training set and a test set using a stratified k fold
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
# dataset): unsupervised feature extraction / dimensionality reduction
n_components = 150
```

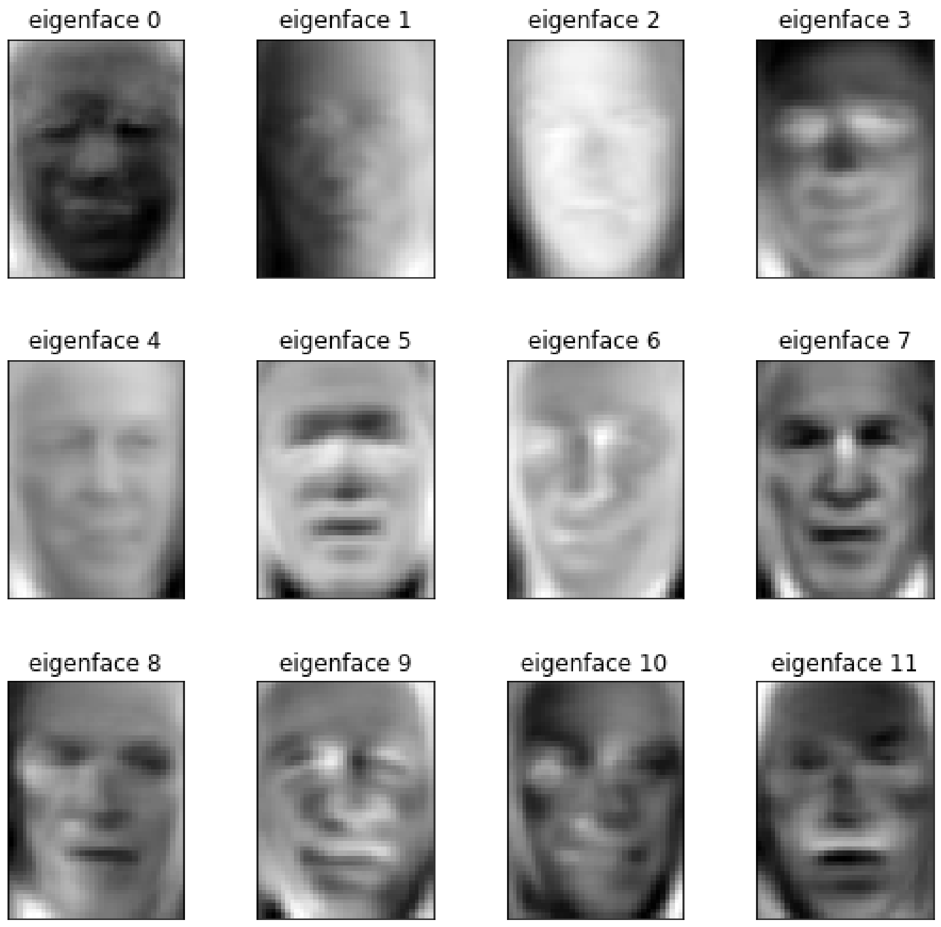
Compute the PCA via eigen-decomposition of the data matrix X after the mean of the training set is removed. This results in a model with variations from the mean. We also transform the training and testing data into 'face space', i.e. the learned sub space of the eigen-faces.

```
In [52]: # Center data
mean = np.mean(X_train, axis=0)
X_train -= mean
X_test -= mean

#Eigen-decomposition
U, S, V = np.linalg.svd(X_train, full_matrices=False)
components = V[:n_components]
eigenfaces = components.reshape((n_components, h, w))

#project into PCA subspace
X_transformed = np.dot(X_train, components.T)
print(X_transformed.shape)
X_test_transformed = np.dot(X_test, components.T)
print(X_test_transformed.shape)
```

Finally, plot the resulting eigen-vectors of the face PCA model, AKA the eigenfaces

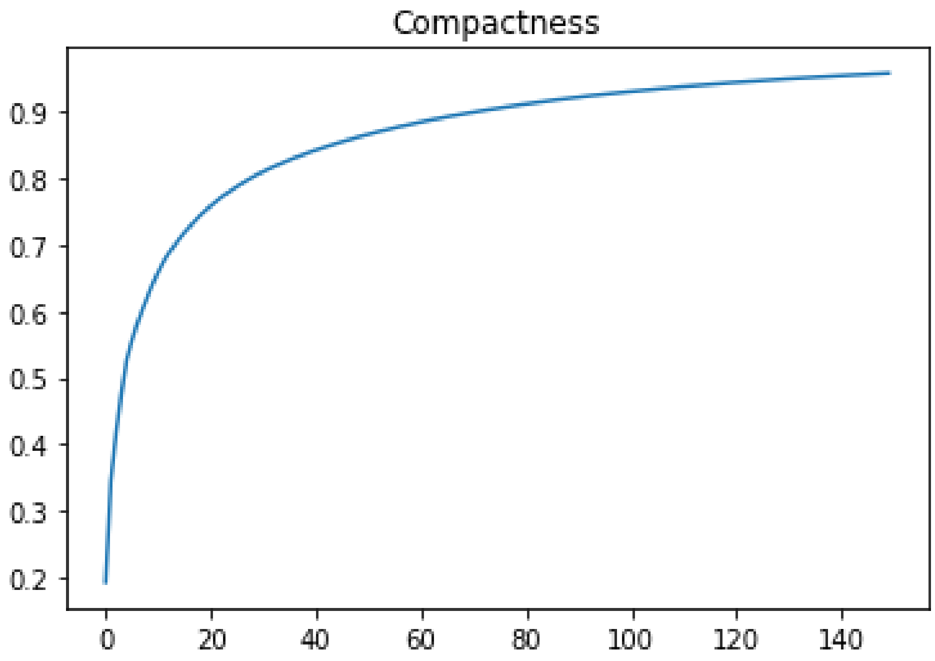
```
In [53]: import matplotlib.pyplot as plt

# Qualitative evaluation of the predictions using matplotlib
```

```
def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
"""Helper function to plot a gallery of portraits"""
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):
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(())

eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
plot_gallery(eigenfaces, eigenface_titles, h, w)

plt.show()
```



We should always evaluate the performance of the dimensionality reduction via a compactness plot

```
In [54]: explained_variance = (S ** 2) / (n_samples - 1)
total_var = explained_variance.sum()
explained_variance_ratio = explained_variance / total_var
ratio_cumsum = np.cumsum(explained_variance_ratio)
print(ratio_cumsum.shape)
eigenvalueCount = np.arange(n_components)

plt.plot(eigenvalueCount, ratio_cumsum[:n_components])
plt.title('Compactness')
plt.show()
```



Use the PCA 'face space' as features and build a random forest classifier to classify the faces according to the labels. We then view its classification performance.

```
In [57]: from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

#build random forest
estimator = RandomForestClassifier(n_estimators=150, max_depth=15, max_features=150)
estimator.fit(X_transformed, y_train) #expects X as [n_samples, n_features]

predictions = estimator.predict(X_test_transformed)
correct = predictions==y_test
total_test = len(X_test_transformed)
#print("Gnd Truth:", y_test)
```

```
print("Total Testing", total_test)
print("Predictions", predictions)
print("Which Correct:",correct)
print("Total Correct:",np.sum(correct))
print("Accuracy:",np.sum(correct)/total_test)
```

```
In [58]: print(classification_report(y_test, predictions, target_names=target_names))
```

This should show you the performance of the RF/PCA based face recognition. (1 Marks)

Your task for this part is to re-implement the above PCA algorithm of the Eigenfaces problem in TF using TF functions. (4 Marks)

2 Part 2 - CNNs (5 Marks)

In this part, you will create a CNN based classifier that should hopefully out perform the above Eigenfaces algorithm from the previous part. Multiple layers will be strung together and tied to a fully connected (or dense) layers to result in a classified output.

Using either TF or Keras, implement the following two CNN based classifiers for the same LFW dataset from Part 1:

1. Dense only layers for classification. (1 Mark)
2. Two convolution layers of 3x3 with 32 filters each and dense layers for classification. (1 Marks)

You can reuse the initial elements of your code in Part 1 that loads and creates the training and the testing sets. You may use the Adam optimiser and (sparse) categorical cross entropy loss. Note that convolution layers expect 4D tensors so that you will need normalise and resize your arrays, see for example

```
#normalise
X_train = X_train / 255.0
X_test = X_test / 255.0
X_train = X_train[:, :, :, np.newaxis]
X_test = X_test[:, :, :, np.newaxis]
print("X_train shape:", X_train.shape)
```

Finally, implement the classification of the CIFAR10 dataset using either TF or Keras. You should use at least 3 convolutional layers in your CNN based classifier. You should show loss and accuracy curves of the training of your classification method. You should use proper training, testing and validation datasets and your validational accuracy should reach around 0.8. Usage of pre-built models will generally not be allowed unless approved by the demonstrator. (3 Marks)

3 Part 3 - Recognition (10 Marks)

For this part, you will need to develop your own solution to one of the recognition problems provided. There are two problems provided, each having a different level of difficulty and will ultimately affect the total maximum mark attainable for this section.

Solve **one** of the following problems using TF or Keras, noting that the marks are allocated according to the level of difficulty:

1. UNet based magnetic resonance (MR) image segmentation of the brain via the Preprocessed OASIS dataset - Hard Difficulty (Maximum 8 Marks out of 10)

2. Face generation using generative adversarial networks (GANs) of the CelebA dataset - Very Hard Difficulty (Maximum 10 Marks out of 10)

You must obtain reasonable results and be able to explain all results, network layers and code to the demonstrator in order to obtain the marks available. Usage of pre-built models will generally not be allowed unless approved by the demonstrator. A note of warning, GANs have very chaotic convergence and therefore difficult to train, only attempt it if you have had some previous experience in deep learning and if you do, you do so at your own risk.