6. Workshop 1-1 : MNIST - FCDeepNets

[Reference] :

- FRANÇOIS CHOLLET, **Deep Learning with Python**, Chapter 2, Section 1, Manning, 2018. (https://tanthiamhuat.files.wordpress.com/2018/03/deeplearningwithpython.pd
(https://tanthiamhuat.files.wordpress.com/2018/03/deeplearningwithpython.pd

In [1]:

```
1 import keras
2 keras.__version__
```

```
/Users/macmini1/anaconda3/lib/python3.6/site-packages/h
5py/__init__.py:36: FutureWarning: Conversion of the se
cond argument of issubdtype from `float` to `np.floatin
g` is deprecated. In future, it will be treated as `np.
float64 == np.dtype(float).type`.
  from ._conv import register_converters as _register_c
onverters
Using TensorFlow backend.
```

Out[1]:

```
'2.2.4'
```

[Hand-written-digit Recognition] :

- A Neural Network which makes use of the Python library Keras to learn to classify hand-written digits.
- The problem we are trying to solve here is to classify grayscale images of handwritten digits (28 pixels by 28 pixels), into their 10 categories (0 to 9).
- The dataset is a set of 60,000 training images, plus 10,000 test images, assembled by the National Institute of Standards and Technology (the NIST in MNIST) in the 1980s.

The MNIST dataset comes pre-loaded in Keras, in the form of a set of four Numpy arrays:

In [2]:

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist
```

```
Downloading data from https://s3.amazonaws.com/img-data
sets/mnist.npz (https://s3.amazonaws.com/img-datasets/m
nist.npz)
11493376/11490434 [==============================] - 12
s 1us/step
```

- `train_images` and `train_labels` form the "training set", the data that the model will learn from.
- The model will then be tested on the "test set", `test_images` and `test_labels`.
- Our images are encoded as Numpy arrays, and the labels are simply an array of digits, ranging from 0 to 9. There is a one-to-one correspondence between the images and the labels.

Let's have a look at the training data:

In [3]:

```
train_images.shape
```

Out[3]:

```
(60000, 28, 28)
```

In [4]:

```
type(train_images)
```

Out[4]:

```
numpy.ndarray
```

In [5]:

```
# train_images[:5]
```

In [6]:

```
len(train_labels)
```

Out[6]:

```
60000
```

In [7]:

```
1 train_labels
```

Out[7]:

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

Let's have a look at the test data:

In [8]:

```
1 test_images.shape
```

Out[8]:

```
(10000, 28, 28)
```

In [9]:

```
1 len(test_labels)
```

Out[9]:

```
10000
```

In [10]:

```
1 test_labels
```

Out[10]:

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

(1) Neural Network

Our workflow to build a neural network will be as follow:

- first we will present our neural network with the training data, `train_images` and `train_labels` .
- The network will then learn to associate images and labels. Finally, we will ask the network to produce predictions for `test_images` , and we will verify if these predictions match the labels from `test_labels` .

Let's build our network -- again, remember that you aren't supposed to understand everything about this example just yet.

In [11]:

```
from keras import models
from keras import layers

network = models.Sequential()
network.add(layers.Dense(512, activation='relu', input_shape=(28
network.add(layers.Dense(128, activation='relu'))
network.add(layers.Dense(10, activation='softmax'))
```

- The core building block of neural networks is the "layer", a data-processing module which you can conceive as a "filter" for data.
  - A deep learning model is like a sieve for data processing, made of a succession of increasingly refined data filters -- the "layers".

Here our network consists of a sequence of two `Dense` layers, which are densely-connected (also called "fully-connected") neural layers.

The second (and last) layer is a 10-way "softmax" layer, which means it will return an array of 10 probability scores (summing to 1).

- Each score will be the probability that the current digit image belongs to one of our 10 digit classes.

To make our network ready for training, we need to pick three more things, as part of "compilation" step:

- `A loss function` : the is how the network will be able to measure how good a job it is doing on its training data, and thus how it will be able to steer itself in the right direction.
- `An optimizer` : this is the mechanism through which the network will update itself based on the data it sees and its loss function.
- `Metrics` to monitor during training and testing. Here we will only care about accuracy (the fraction of the images that were correctly classified).

In [12]:

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

Before training, we will preprocess our data by reshaping it into the shape that the network expects, and scaling it so that all values are in the `[0, 1]` interval. Previously, our training images for instance were stored in an array of shape `(60000, 28, 28)` of type `uint8` with values in the `[0, 255]` interval. We transform it into a `float32` array of shape `(60000, 28 * 28)` with values between 0 and 1.

In [13]:

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

We also need to categorically encode the labels :

In [14]:

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

We are now ready to train our network, which in Keras is done via a call to the `fit` method of the network: we `fit` the model to its training data.

In [15]:

```
network.fit(train_images, train_labels, epochs=15, batch_size=12
```

```
Epoch 1/15
60000/60000 [==============================] - 6s 93us/
step - loss: 0.2339 - acc: 0.9294
Epoch 2/15
60000/60000 [==============================] - 5s 79us/
step - loss: 0.0872 - acc: 0.9737
Epoch 3/15
60000/60000 [==============================] - 5s 79us/
step - loss: 0.0556 - acc: 0.9826
Epoch 4/15
60000/60000 [==============================] - 5s 91us/
step - loss: 0.0402 - acc: 0.9876
Epoch 5/15
60000/60000 [==============================] - 5s 81us/
step - loss: 0.0290 - acc: 0.9911
Epoch 6/15
60000/60000 [==============================] - 5s 91us/
step - loss: 0.0235 - acc: 0.9924
Epoch 7/15
60000/60000 [==============================] - 5s 81us/
step - loss: 0.0175 - acc: 0.9944
Epoch 8/15
60000/60000 [==============================] - 5s 81us/
step - loss: 0.0145 - acc: 0.9953
Epoch 9/15
60000/60000 [==============================] - 5s 81us/
step - loss: 0.0109 - acc: 0.9966
Epoch 10/15
60000/60000 [==============================] - 6s 92us/
step - loss: 0.0100 - acc: 0.9967
Epoch 11/15
60000/60000 [==============================] - 5s 81us/
step - loss: 0.0085 - acc: 0.9972
Epoch 12/15
60000/60000 [==============================] - 5s 81us/
step - loss: 0.0076 - acc: 0.9974
Epoch 13/15
60000/60000 [==============================] - 5s 83us/
step - loss: 0.0055 - acc: 0.9983
Epoch 14/15
60000/60000 [==============================] - 6s 94us/
step - loss: 0.0056 - acc: 0.9981
Epoch 15/15
60000/60000 [==============================] - 5s 83us/
step - loss: 0.0054 - acc: 0.9981
```

Out[15]:

```
<keras.callbacks.History at 0x120dfa588>
```

Two quantities are being displayed during training: the "loss" of the network over the training data, and the accuracy of the network over the training data.

We quickly reach an accuracy of 0.989 (i.e. 98.9%) on the training data. Now let's check that our model performs well on the test set too:

In [16]:

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

```
10000/10000 [==============================] - 1s 62us/
step
```

In [17]:

```
print('test_acc:', test_acc)
```

```
test_acc: 0.9817
```

In [18]:

```
print('test_loss:', test_loss)
```

```
test_loss: 0.11624843135551359
```

Prediction

In [19]:

```
test_labels[0]
```

Out[19]:

```
array([0., 0., 0., 0., 0., 0., 0., 1., 0., 0.], dtype=f
loat32)
```

In [20]:

```
import numpy as np
np.argmax(test_labels[0])
```

Out[20]:

```
7
```

In [21]:

```
test_labels_result = []
for i in range(len(test_labels)):
    test_labels_result.append(np.argmax(test_labels[i]))

test_labels_result = np.array(test_labels_result)
test_labels_result
```

Out[21]:

```
array([7, 2, 1, ..., 4, 5, 6])
```

In [22]:

```
test_predict = network.predict(test_images)
test_predict
```

Out[22]:

```
array([[8.8353529e-29, 6.5117192e-23, 1.2437190e-21,
..., 1.0000000e+00,
        2.9669020e-27, 1.4993327e-18],
       [6.8299042e-22, 3.2446444e-15, 1.0000000e+00,
..., 5.6851544e-29,
        2.8711930e-26, 2.3466056e-38],
       [1.6473348e-19, 1.0000000e+00, 1.3341571e-11,
..., 1.0199087e-10,
        3.7541550e-10, 3.3240700e-17],
       ...,
       [9.0719796e-34, 5.5293558e-23, 4.1824918e-32,
..., 2.9199153e-18,
        2.6415056e-20, 5.8992293e-16],
       [1.9390213e-27, 7.5007017e-28, 5.6058738e-32,
..., 7.2193978e-24,
        1.4676804e-13, 3.7338038e-32],
       [2.3808017e-34, 2.6388741e-37, 6.0389028e-36,
..., 0.0000000e+00,
        6.7137918e-29, 4.7761802e-29]], dtype=float32)
```

In [23]:

```
test_predict_result = [np.argmax(test_predict[i]) for i in range

test_predict_result = np.array(test_predict_result)
test_predict_result
```

Out[23]:

```
array([7, 2, 1, ..., 4, 5, 6])
```

Confusion Matrix

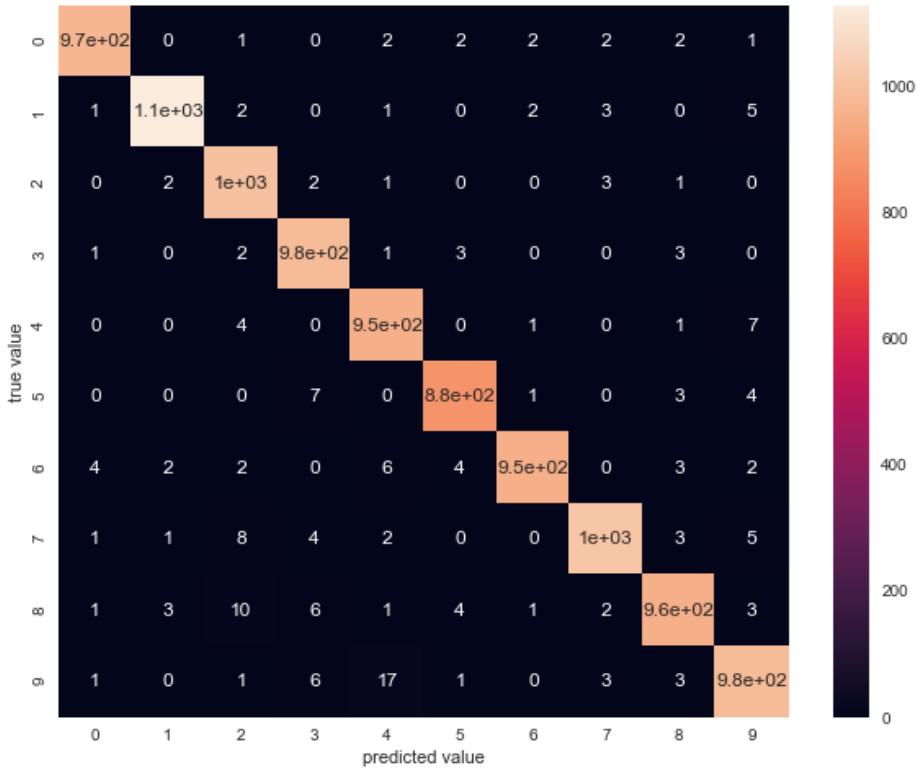
In [24]:

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns; sns.set()

from sklearn.metrics import confusion_matrix

mat = confusion_matrix(test_predict_result, test_labels_result)

plt.figure(figsize=(10,8))
sns.heatmap(mat, square=False, annot=True, cbar=True)
plt.xlabel('predicted value')
plt.ylabel('true value');
```



Our test set accuracy turns out to be 97.8% -- that's quite a bit lower than the training set accuracy.

This gap between training accuracy and test accuracy is an example of "overfitting", the fact that machine learning models tend to perform worse on new data than on their training data.

- This concludes our very first example -- you just saw how we could build and a train a neural network to classify handwritten digits, in less than 20 lines of Python code.

(2) Fully-Connected Deep Network

Building a Fully-Conneted Deep Network with 2 hidden layers :

```
from keras import models

from keras import layers

network = models.Sequential()

network.add(layers.Dense(512, activation='relu',
input_shape=(28 * 28,)))

network.add(layers.Dense(128, activation='relu'))

network.add(layers.Dense(10, activation='softmax'))
```

Q : Is there any improvement on accuracy?

In [ ]:

```
1
```