APPLIED MACHINE LEARNING
LAB ACTIVITIES (LAB 7)

28/11/19

CONVOLUTIONAL NEURAL NETWORK WITH KERAS

This workbook is designed to guide you through the activities proposed for today's lab. As you will be working independently, feel free to proceed through the text at your own pace, spending more time on the parts that are less familiar to you. The workbook contains both hands-on tasks and links to learning materials such as tutorials, articles and videos. When you are unsure about something, feel free to ask our teaching assistants or use Internet resources to look for a solution. At the end of each section, there will be questions and exercises to verify your understanding of the presented information. You may need to do some research to answer the questions.

1. Handwritten Digits Classification

In this lab we will create a simple CNN for MNIST (Modified National Institute of Standards and Technology database) that demonstrates how to use all of the aspects of a modern CNN implementation, including Convolutional layers, Pooling layers and Dropout layers.

The MNIST database is a large database of handwritten digits that is commonly used for training various image processing systems. It was created by "re-mixing" the samples from NIST's (National Institute of Standards and Technology) original datasets. The MNIST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set were taken from NIST's training dataset, while the other half of the training set and the other half of the test set were taken from NIST's testing dataset. There have been a number of scientific papers on attempts to achieve the lowest error rate; one paper, using a hierarchical system of convolutional neural networks, manages to get an error rate on the MNIST database of 0.23%. The original creators of the database keep a list of some of the methods tested on it. In their original paper, they use a support-vector machine to get an error rate of 0.8%.

The Keras deep learning library provides a convenience method for loading the MNIST dataset. The dataset is downloaded automatically the first time this function is called and is stored in your home directory in *~/.keras/datasets/mnist.pkl.gz* as a 15 megabyte file. This is very handy for developing and testing deep learning models. To demonstrate how easy it is to load the MNIST dataset, we will first write a little script to download and visualise the first 4 images in the training dataset.

```
# plot ad hoc MNIST instances
import keras
from keras.datasets import mnist
import matplotlib.pyplot as plt

# load (downloaded if needed) the MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# plot 4 images as gray scale
plt.subplot(221)
plt.imshow(x_train[0], cmap=plt.get_cmap('gray'))
plt.subplot(222)
plt.imshow(x_train[1], cmap=plt.get_cmap('gray'))
plt.subplot(223)
plt.imshow(x_train[2], cmap=plt.get_cmap('gray'))
plt.subplot(224)
plt.imshow(x_train[3], cmap=plt.get_cmap('gray'))

# show the plot
plt.show()
```

Let us import other classes and functions needed, Again, we always initialise the random number generator to a constant seed value for reproducibility of results.

```
# to see versions
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
import sys
import tensorflow as tf

from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D

# fix random seed for reproducibility
import numpy as np
np.random.seed(7)
```

Let us use *version* functions to see the versions of Python, TensorFlow and Keras.

```
# print
print('Python version : ', sys.version)
print('TensorFlow version : ', tf.__version__)
print('Keras version : ', keras.__version__)
```

As we already loaded the MNIST dataset, we now just reshape it so that it is suitable for use training a CNN. In Keras, the layers used for two-dimensional convolutions (Conv2D) expect pixel values with the dimensions `[height][width][channels]`. In the case of RGB, the first-dimension channels would be 3 for the red, green and blue components and it would be like having 3 image inputs for every colour image. In the case of MNIST where the channels values are grey scale, the pixel dimension is set to 1.

```
img_rows = 28
img_cols = 28
input_shape = (img_rows, img_cols, 1)

# reshape to be [samples][height][width][channels]
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
```

As before, it is a good idea to normalise the pixel values to the range 0 and 1 and one hot encode the output variable.

```
# normalise inputs from 0-255 to 0-1
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
```

Let's print to see the input shape.

```
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

Let's do *hot encode outputs* for labels. A one hot encoding is a representation of categorical variables as binary vectors. This first requires that the categorical values be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1. For example, if we had the sequence: 'red', 'red', 'green'. We could represent it with the integer encoding: 0, 0, 1. And one hot encoding of

```
[1, 0]
[1, 0]
[0, 1]
```

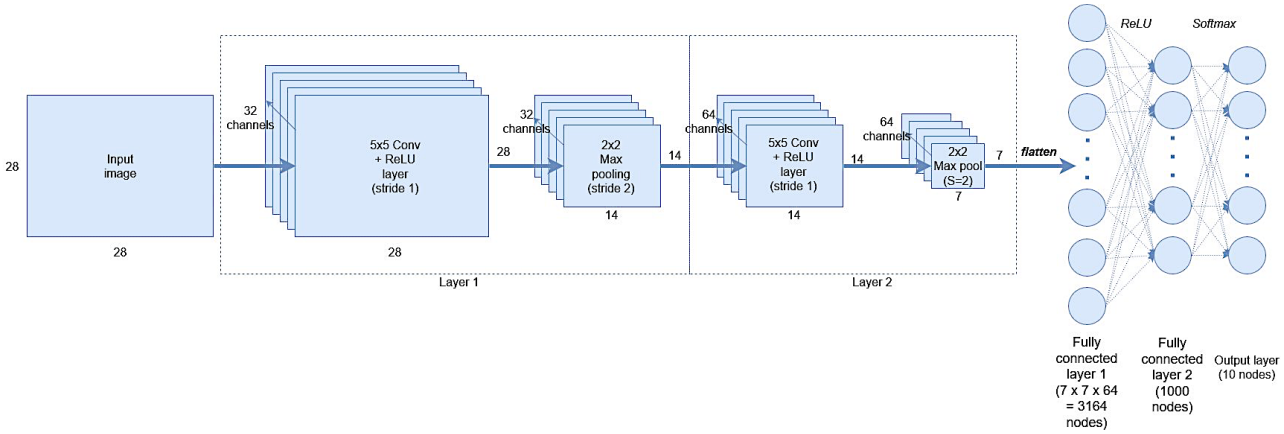
We would like to give the network more expressive power to learn a probability-like number for each possible label value.

```
# one hot encode outputs
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

Next we define our neural network model. Convolutional neural networks are more complex than standard Multilayer Perceptron (MLP), so we will start by using a simple structure to begin with that uses all of the elements for state-of-the-art results. Below summarises the network architecture.

1. The first hidden layer is a convolutional layer called a Conv2D. The layer has 32 feature maps, with the size of 5 × 5 and a rectifier ($relu$) activation function. This is the input layer, expecting images with the structure outline above.
2. Next, we define a pooling layer that takes the maximum value called MaxPooling2D. It is configured with a pool size of 2 × 2.
3. The second hidden layer is also a Conv2D. The layer has 64 features maps, with the size of 5 × 5 and a $relu$ activation function.
4. Next, we define a MaxPooling2D again. It is configured with a pool size of 2 × 2.
5. The next layer is a regularization layer using dropout called Dropout. It is configured to randomly exclude 25% of neurons in the layer in order to reduce overfitting.
6. Next is a layer that converts the 2D matrix data to a vector called $Flatten$. It allows the output to be processed by standard fully connected layers.
7. Next a fully connected layer with 1000 neurons and $relu$ function is used.
8. We again use Dropout. It is configured to randomly exclude 50% of neurons in the layer.
9. Finally, the output layer has 10 neurons for the 10 classes and a softmax activation function to output probability-like predictions for each class.

As before, the model is trained using logarithmic loss and the ADAM gradient descent algorithm. A depiction of the network structure is provided below.



[Source] Adventures in Machine Learning

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1), padding='same',
activation='relu', input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Conv2D(64, (2, 2), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(1000, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
model.summary()
```

Compile and fit the model. The CNN is fit over 12 epochs with a batch size of 128 as defined above. The $verbose$ setting changes the way you see the training progress for each epoch. For example, $verbose = 0$ will show you nothing (silent), $verbose = 1$ will show you an animated progress bar like this: `[==============================]` and $verbose = 2$ will just mention the number of epoch like this: `Epoch 1/10`

```
opt = 'adam'
loss = 'categorical_crossentropy'
metrics = ['accuracy']

# these values are chosen via trial and error
batch_size = 128
epochs = 12

model.compile(optimizer=opt, loss=loss, metrics=metrics)

# fit the model
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1,
validation_data=(x_test, y_test))
```

Verify your understanding:

(a) Use $evaluate$ function in $Model$ class to print the test loss and accuracy values. Refer to https://keras.io/models/model/

(b) How many images are misclassified?

(c) Run the below code blocks and analyse the plots. What do these plots tell?

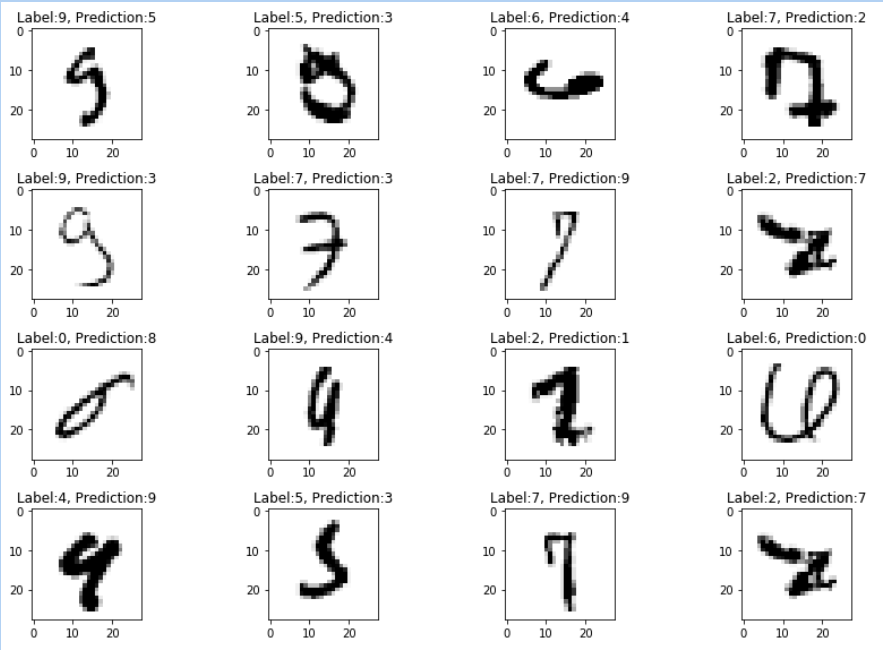
```
print(history)
fig1, ax_acc = plt.subplots()
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Model - Accuracy')
plt.legend(['Training', 'Validation'], loc='lower right')
plt.show()

fig2, ax_loss = plt.subplots()
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Model - Loss')
plt.legend(['Training', 'Validation'], loc='upper right')
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.show()
```

(d) Run the below code block and interpret the results.

```
n = 0
plt.imshow(x_test[n].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
print('The Answer is ', model.predict_classes(x_test[n].reshape((1, 28, 28, 1))))
```

(e) Write a code block that shows the misclassified images and how they are predicted as in the image below.



2. References

[1]. Géron, A., 2017. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.".

[2]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.

[3]. Grus, J., 2019. Data science from scratch: first principles with python. O'Reilly Media.

[4]. Müller, A.C. and Guido, S., 2016. Introduction to machine learning with Python: a guide for data scientists. " O'Reilly Media, Inc.".

[5]. Brownlee, J., 2014. Machine learning mastery.

[6]. Raschka, S., 2015. Python machine learning. Packt Publishing Ltd.

[7]. SAS, 2018, Advanced Predictive Modelling using SAS

[8]. Géron, A., 2019. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media.