Practical Work 10 – 21/11/2019
Conv Neural Networks with Keras - part 3

Objectives

The objective of this PW is to practice with some applications of Convolutional Neural Networks (CNN) including transfer learning and auto-encoders. Another objective is to experiments with the **functional API** of Keras that allows to build more complex network structures.

As for the last practical work, we ask you to submit the solution for next week.

Submission

— **Deadline** : Wednesday 12th May, 12 :00 (noon)

— **Format** : Zip with report and iPython notebook.

Exercise 1   Functional API of Keras

Make sure first that you understand the principles of Keras functional API (slides 8–21). Using the CIFAR10 dataset :

a) Re-use one of your best CNN architecture from PW08 or PW09 and transform the model definition from Sequential to the functional API. Observe that you get the same number of parameters and similar performances between the sequential and functional models.

b) Experiment with non-sequential strategies such as *multiple features* or *multiple paths* as described in slides 21–22.

c) Optional : install GraphViz to use the Keras function `plot_model()`. If working[1], use it to generate graphs of the architectures.

d) Use *callbacks* to save the best trained models according to a monitoring of the accuracy on the test set.

Report on your experiments and describe your best configuration through experimenting with 3-4 different architectures. Use a table similar to the example given below and provide the hyper-parameters used for your configurations.

---

1. The installation may not be so obvious.

Model	Architecture	Callback	Acc. train %	Acc. test %
1	Description of architecture...	yes	84.2%	78.4%
2	...	...	...	...
3	...	...	...	...
4	...	...	...	...
5	...	...	...	...

TABLE 1 – Performances on CIFAR10 with different sequential and non-sequential configurations. In the Architecture column, D is the number of filters, w h are the width and height in pixels, S is the stride value, P is the padding value, etc.

Exercise 2   Transfer learning

Keras provides many pre-trained architectures on ImageNet. Such architectures are able to extract robust image features and can generally be used for many different image tasks. The objective of the exercise is to check if such features lead to good results on the Food101 dataset. This dataset contains 101'000 color images of meals corresponding to 101 classes. In order not to have too long trainings, only the first 20 classes will be used. The following tutorial can probably help you with the exercise : https://keras.io/guides/transfer_learning/#build-a-model

a) Review slides 33–36 on using Keras for a transfer learning task.

b) Chose one of the architecture presented here : https://keras.io/applications/. Beware that some architectures are using large memory and lots of cpu (so better move to gpu)!

c) Download the provided notebook `ex2_transfer_learning_stud.ipynb`.

d) Implement the preprocessing and the training phases.

e) Experiment with one or several architectures with different classification heads (hidden-layers, ...).

f) Describe the results in your report which architecture and parameters works best.

Exercise 3   Auto-encoders

We will use the notMNIST data set. First get the `.gz` files from this site : https://github.com/davidflanagan/notMNIST-to-MNIST.

Use as a starting point the notebook `notMNIST-auto-encoder-stud.ipynb`.

Create a shallow dense autoencoder

Using the Keras library, build a very simple autoencoder with one input layer, one output layer and one hidden layer.

a) Chose the dimension of your hidden layer. As this is the size of your encoding, you should chose a "reasonable" dimension, in between the number of classes of your problem and the dimension of input and output layers [2].

b) Define your layers, don't forget that your input dimension is the same as your output.

c) Create your models (using the keras `Model`) : encoder, decoder and autoencoder (encoder and decoder).

d) Don't forget to compile your autoencoder. Fit it on the data !

e) Visualize your encoding image with the input images on the top and their corresponding encoded ones on the bottom. Your visualization may differ slightly from the one on Figure 1

f) How could we evaluate the performance of such a "compression" tool (in terms of loss and in terms of gain of bandwidth) ? You just need to comment – no need to compute anything here [3].

g) Are you now able to use the encoded features to train a simple Support Vector Machine – SVM classifier ? Comment on the performance by comparing the extracted features of the auto-encoder with using the pixel values as input of the SVM.
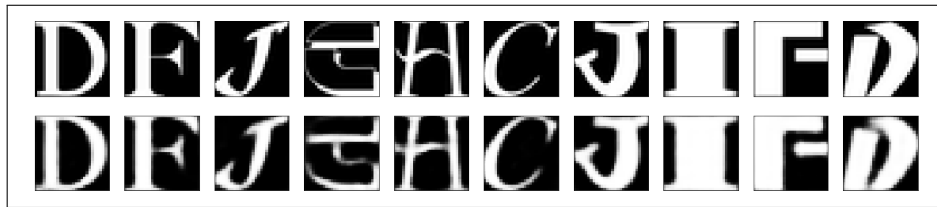

FIGURE 1 – Input and output of the autoencoder

Create a convolutional de-noising autoencoder

Create a model in Keras with the following structure :

— Encoder
    — Convolution layer with 32 filters of size 3 x 3
    — Downsampling (max-pooling) layer of size 2 x 2
    — Convolution layer with 64 filters of size 3 x 3
    — Downsampling (max-pooling) layer of size 2 x 2
    — Convolution layer with 128 filters of size 3 x 3
— Decoder
    — Convolution layer with 128 filters of size 3 x 3
    — Upsampling layer of size 2 x 2 (see `UpSampling2D()`)

---

2. We tried with 32 and it was working well, you may try yourself with different values to observe the impact.
3. If you really want to compute something, you can of course.

— Convolution layer with 64 filters of size 3 x 3

— Upsampling layer of size 2 x 2

— Convolution layer with 1 filter of size 3 x 3

a) Check with the `summary()` method that you have indeed a diabolo network and that your output shape is the same as your input shape.

b) Would it make sense to use the output of the encoder to compress the images? Comment your answer.

c) Generate noisy versions of your train and test data using (see examples in slides).

d) Train your network using a MSE loss and batch sizes of 128.

e) Visualise the denoising capacity on some test data.

Exercise 4   Optional : Review Questions

a) Why (or when) do we need the functional API of Keras?

b) What are the benefits of using transfer learning?

c) What are the potential usage of auto-encoders?

d) In which situations would you use auto-encoders to extract features?