**Machine Learning Tutorial**

**Universidad Nacional de Tucuman, March 2020**

**Jaume Bacardit**

**Practical 2: Deep Learning**

[**https://github.com/jaumebp/ML-tutorial**](https://github.com/jaumebp/ML-tutorial)

In this practical you will play with a small Convolutional Neural Network to understand what are the basic blocs for its functioning. In the latter part of the tutorial you will see how the activations of the convolutional and pooling layers of the network can be visualised. This practical has been created from two main sources:

* <https://github.com/keras-team/keras/blob/master/examples/mnist_cnn.py>
* <https://towardsdatascience.com/visualizing-intermediate-activation-in-convolutional-neural-networks-with-keras-260b36d60d0>

1. **Building Convolutional Neural Networks**

Let’s walk through the code of mnist\_cnn.py.

* First of all, this example uses a deep learning framework called Keras. Keras is designed to make simple the process of building deep learning models. It sits on top of Google’s TensorFlow, but in most cases you don’t need to know anything about TF.
* Lines 18-23 define some basic parameters of the example, batch size of the training process (number of images processed before each step of back-propagation), number of classes of the dataset, number of epochs (passes through the complete training set), and the resolution of the images used in this example
* Line 26 loads the dataset used in this example. This dataset is called MNIST and it is a very well known benchmark in image classification. The images correspond to hand-written digits, and the class labels are the digit 0-9 to which each image corresponds to. This dataset is very good to use for demo purposes, but it is essentially a toy problem, not challenging anymore
* Lines 29-44 process the images so that they become matrices suitable to be fed to the deep learning models
* Lines 47-48 process the class labels of the images, again so they can be processed by the model.
* Lines 50-66 define the CNN architecture of the model. You will see that many lines are commented. I will explain them later. The model is specified by adding one by one the layers of the network. The first layer is a 2D convolutional layer of kernel size (3,3), 32 filters and ReLU activation function. Next a second convolutional layer is added, with 62 filters. The next step is a MaxPooling layer of size (2,2), designed to shrink the size of the layers (Each window of 2x2 pixels is reduced to a single pixel). The next layer is Flatten(). This layer just changes the shape of the data structure containing the activations of the previous layer, from 3D (12 x 12 x 64) to a 1D vector of size 9216. Afterwards come two fully connected layers – Dense(), i.e. classic layers of Perceptron neurons. One of size 128 and one of size 10, which will be the output layer of the network (with as many neurons as classes has this dataset). The output layer has softmax activation so that the network generates pseudo-probabilities for each class in the problem.
* Line 68 specifies the loss function (how the errors of the network are evaluated during training), the optimisation algorithm used to train the network, and the metrics to generate during training
* Line 72 will print a written summary of the network structure
* Line 74 will train the network using the pre-specified MNIST training set
* Line 78 will test the network, again with the pre-specified MNIST test set
* Line 82 will save the weights of the trained network

1. **What can we do with this network?**

* First, run it as is, to realise how fast/slow is the training process. If the machine you are using has a NVIDIA GPU, Keras should be able to automatically use it. If not, Keras will use as many CPU cores as it can detect for the training process
* Start to uncomment some of disabled layers:
  + Dropout. This layer type will randomly and temporarily set to 0 some percentage of the weights of the network, to make it more resilient
  + BatchNormalisation. This layer type will perform a normalisation process of the activations of the network batch by batch, and layer by layer (whenever it is enabled)
  + You can decide to add more Conv2D, MaxPooling or Dense layers, or change their sizes
* Change the dataset. Comment line 26 and uncomment line 27. You will be using a different dataset called Fashion MNIST, designed to have the same size (number of images, resolution) as MNIST but have different types of objects to be recongnised. You will see that this dataset is slightly more challenging.

1. **Visualise the network activations**

* After you have finished tweaking your network we can move to the other script, cnn\_viz.py
* If you look at this script, most of the lines are exactly the same, but it has a major difference, it does not train a network. This script loads the weights that were saved by the first script (line 75).
* The next lines create a “Model” (abstraction for network) that as output generates the activations that are produced by the layers of the network for a given input image.
* The image will be one from the test set. You can choose which one by changing the variable img\_index (number from 0 to 9999)
* The script first generates an image of this test instance. Then it will generate an image for each of the three convolutional and MaxPooling layers. It will not generate visualisations for the Dense layers because they are just 1D vectors, so they are not suitable to be visualised as images.
* Run this script with different test images to see how the network processes them
* You can even train a network on MNIST and then visualise the activations of an image from the Fashion-MNIST dataset, but the results will probably be pretty random.
* Remember, if you have changed the architecture of the network in the first script, you need to modify the second script accordingly.