Deep Learning with MatConvNet Practical Guide I

Joost van de Weijer, Marc Masana, German Ros

In these practical sessions we apply Deep Convolutional Networks to the problem of scene recognition. We will use MatConvNet (http://www.vlfeat.org/matconvnet/) which is a MATLAB toolbox implementing Convolutional Neural Networks (CNNs) for computer vision applications. This practical session is based on the Hands-on Deep Learning course prepared by German Ros[1].

In this first session you will learn:

I. Basic skills to work with MatConvNet; and understand how deep networks are defined.
II. How to do inference with a pre-trained network and extracting features from the network.
III. Fine-tuning a pre-trained network for the MIT scene recognition task.

CNNs are known to require a lot of data to train all their parameters. In this session we will use the AlexNet which is a famous network with several convolutional layers followed by several fully connected layers (see figure 1).

0. Get Started

We will only use CPU during this practical session and will not use GPU computation.
Unzip the tarball unzip matconvnet-1.0-beta17.zip.
For Windows follow the instructions on this website: http://www.vlfeat.org/matconvnet/install/
For Linux and MAC look at the instructions in the file: InstallHelp.txt .

I. Basic MatConvNet skills.

1. We have tested this code on MATCONVNET version 17 which we provide and recommend you to also use during these sessions. Unpack the DL_session1.zip file on your computer.

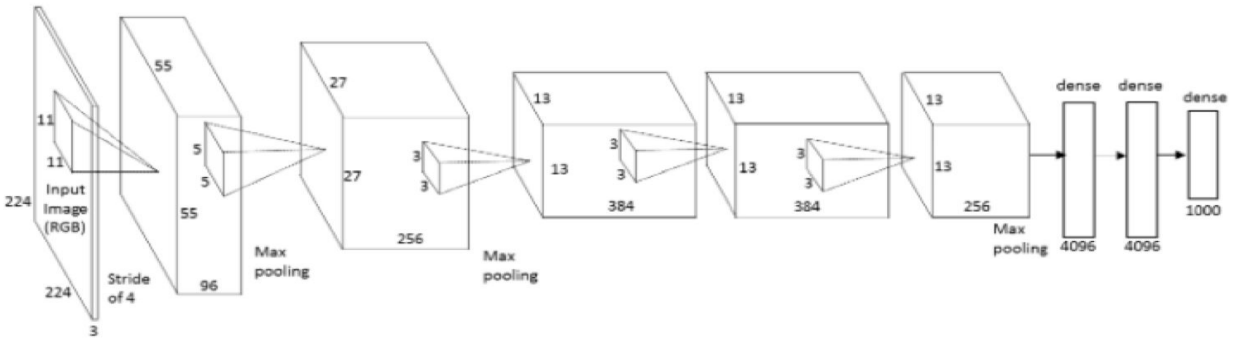

Fig 1: Overview of AlexNet.

[1] http://www.cvc.uab.es/~gros/index.php/hands-on-deep-learning-with-matconvnet

2. Open session01_script.m to start. Execute one by one the commands in the script and follow the comments. The AlexNet has been pre-trained on the ImageNet dataset with over 1 million different object classes.
3. Have a closer look at inference_alexnet.m to see how an image is prepared to be evaluated by AlexNet.

II. Fine tuning with AlexNet

Next we will use the pre-trained AlexNet.

1. Run the code to compute the IMDB (IMage DataBase) which is the MATLAB structure containing the images, their labels, and the division in train, validation and testset.
2. Open the file alexnet_train_small.m and have a closer look at the code. First have a look at the parameters which are set in the beginning. These include batchSize, numEpochs, learningRate, weightDecay and the 'continue' parameter. You should understand all of these and you should play with them during the coming exercises. Next have a look at the definition of the network (all the lines which start with net.addLayer). The layers are defined in the folder $matconvnet-1.0-beta17/matlab/+dagnn .

   We will consider the first layer of the network in detail:

   net.addLayer('conv1', dagnn.Conv('size', [11 11 3 96], 'hasBias', true, 'stride', [4, 4], 'pad', [15 15 15 15]), {'input'}, {'conv1'}, {'conv1f' 'conv1b'});

   - The layer is a convolutional layer and its name is 'conv1'.
   - This convolutional layer has 96 filters of dimensions [11,11,3], and has bias.
   - The stride is [4,4] which means that the convolution is only computed once for every 4 pixels (as consequence the output is around 4x smaller)
   - There is padding applied of [15 15 15 15] which means that before the convolution 15 zeros are added to the left, right, top and bottom of the input.
   - The input to this layer is layer 'input' the output is layer 'conv1'
   - the parameters are named 'conv1f' and 'conv1b'.

   Have a look at the other layers and see if you understand all parameters.

3. Continue with the script and start training the network (we will only run it for 2 epochs now). Then continue until EXERCISE 1.

EXERCISE 1:
Finetune AlexNet on the MIT images of scale 227x227x3. Change the code which currently takes images of 37x37x3 as an input to a code which accepts images of size 227x227x3. The original alexNet was adapted to accept images of 37x37x3 which makes the training faster but less accurate. You have to undo the changes.

- Open the code of AlexNet.m code and adapt it to images 227x227x3. To do so you have to undo several changes which are performed on the layers. To change the network to accept small images we added padding in layer 1 (which was not there), and we changed some parameters in 'fc6'. If you look at the dimensionality of the data when reaching fc6 you should understand how to change this layer.
- We only transferred the parameters from the pre-trained network for the convolutional layers but not for 'fc6' and 'fc7'. Change this in the code of AlexNet.m in the function function initNet_FineTuning(net, netPre).
- Because this takes a lot of memory we will use another function to create the IMDB called createIMDB_DISK.m which does not store the images. We will also use another getBatch function, which is called getBatchDisk which will load the images. This has already been changed in AlexNet.m and you do not have to change anything.

Compare the results of the two network (the small and regular) for a number of epochs. Also play with the parameter settings.

Optional: Consider applying data augmentation in the getBatchDisk function (e.g. flipping) to improve results.

III. Extracting Features From AlexNet

Next we will use the pre-trained AlexNet to extract features from the images in the MIT dataset and use these in the Bag-of-words exercises.

EXERCISE 2:
In the function inference_alexnet.m you can see how the variables (probabilities) of the last layer are extracted. It is important to first set:

```
alexNet.conserveMemory=0;
```

else it will not memorize the intermediate results of the layers. Using similar instructions as:

```
net.eval({'input', im_});
ff = net.vars(net.getVarIndex('prob')).value;
```

You should be able to extract the features from different layers for the images in the MIT data set. For the convolutional layers you might want to construct a visual vocabulary and use the features in the BOW. The FC layers could be directly used as image representations.

Show the accuracy you obtain for the different layers in AlexNet (for the 5 CONV layers and the 2 FC layers) in your BOW framework. To load .mat files in python look at documentation of scipy.io (e.g. 'import scipy.io as sio' and 'mat_contents = sio.loadmat('example.mat')´)