DEEP LEARNING

Machine Learning 2018/19: Assignment 5

Torielli Davide Fusaro Fabio
EMARO - European Master on Advanced Robotics
DIBRIS - Dip. Informatica, Bioingegneria, Robotica, Ing. Dei Sistemi
Università Degli Studi Di Genova, December 14, 2018

I. Introduction

DEEP LEARNING means using *deep* neural networks. In *deep learning*, there are different building blocks which are part of the structure. They are basic networks stacked, where each one acts independently taking information from the previous block and giving results to the subsequent block.
In this work, we use the *Deep Learning Toolbox* of Matlab to understand how to use a *Convolutional Neural Network*, or *CNN*, on images. In particular we try different pretrained sets to do various tasks. In fact, seen that the training phase is longer and requires huge amount of data, there are different pretrained networks available like *GoogLeNet*, *ResNet*, *AlexNet*, *VGG networks*.

II. Convolutional Neural Network

CNN are used to deal with large-dimensionality data, like images and sounds from the real world.
CNN has a reduced connective pattern and its weights (of the hidden layer) are *shared*, which means that there are the same few values shared in the layer.
CNN has an architecture divided into 3 groups of layers:

- Convolutional Layer
- Pooling Layer
- Output Layer

1) Convolutional Layer: The Convolutional Layer is the first (the one which takes the inputs). Here, each unit has the same *weight* as the others units.
It is called "Convolutional" because it implements a discrete convolution:

$$y_j = \sum_{i=1}^{d} x_{i+j}\, w_{3-i+j}$$

where the weights w are the ones learned from the training data.
The advantage of this is the optimization done while making computations: for example, with a 100x100 image and 3x3 patches, there are only 9 weights to compute instead of 100000. This is possible because, as said, there are *shared* weights and a few number of connections.

2) Pooling Layer: This "intermediate" layer reduces the dimensionality of variables that come from the Convolutional Layer.
It replaces a set of output values with one scalar; so it acts as a subsampler, possibly nonlinear. An easy way to do this is to take the maximum of the set of values (*max pooling layer*), but there are also different methods (e.g. average).

3) Output Layer: This a usual neural network layer, with fully connected units and an activation nonlinear function (*ReLU* is common when used as final stack of deep networks).

Real complex trained networks have these groups repeated a certain number of time, sometimes creating trees and graphs of sublayers (e.g. in AlexNet we have, among the others, *conv1, ..., pool1, ..., conv4, ..., pool4, ...*).
In general, first layers learn basic features (e.g. edges and blobs for images) and last layers learn more specific feature (e.g. this is a face, a dog, ...).

III. Our Work

Our work consists in the use of the Deep Learning Toolbox of Matlab.
The work is divided in 1+3 tasks:

- **Task 0**: Follow the basic Matlab tutorial on the use the Deep Learning Toolbox.
- **Task 1**: Use pretrained network *GoogLeNet* to classify images.
- **Task 2**: Use pretrained network *AlexNet* as *feature extractor*.
- **Task 3**: Use pretrained network as inizialization to learn new (but similar) task. This is called *transfer learning*.

For the tasks, we use two different pretrained networks: GoogLeNet and AlexNet. AlexNet is the old one (2012, the other 2015) and it is less accurate and slower than GoogLeNet.

We can't give anything we want to a pretrained network: it has its own architecture. This is a constraint for the input, so we can give only images of a given size, because the input layer will have one input per pixel. So we have to resize in some way (*reshape*) the images if these have not the correct dimensions (224-by-224 for GoogLeNet, 227-by-227 for AlexNet).

A. Task 1

In this task we simply take the pretrained network GoogLeNet and use it to classify images from *COIL-100* dataset (from http://www1.cs.columbia.edu/CAVE/software/softlib/coil-100.php).
To classify all the images, we make a matlab script where the first image is taken (and classified) and then an user input is waited to classify another image.

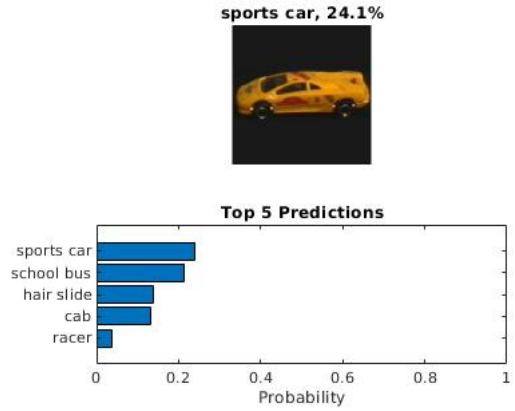

Fig. 1: *Example of classified image with the top five predictions*

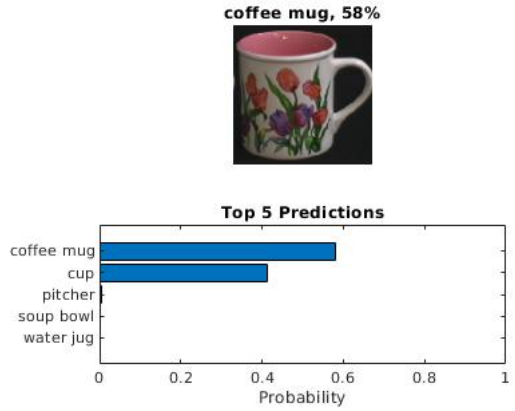

Fig. 2: *Example of classified image with the top five predictions*

In the figure above (Fig. 1 and Fig. 2) we show the results of two examples using GoogLeNet as "black box" classificator.

B. Task 2

In this task we extract learned image features from the pretrained network AlexNet, and use them to train an image classifier. We use the Caltech101 dataset (http://www.vision.caltech.edu/Image_Datasets/Caltech101/) but, to have result in a reasonable time, we reduce the categories of the dataset from the original 101 to 6, so less data to analyze.

We know that deeper layers contain higher-level features, constructed using the lower-level features of earlier layers. So we can choose to take a lower level *fully connected* layer (e.g. fc6) to have lower-level representation of the images, or a higher level *fc* (e.g. *fc8*) to have higher-level representation. However, these layers are all deep ones, we can't the first ones because they contain too few information.

Fig. 3: *Example of classified images with fc6 layer*

Fig. 4: *Example of classified images with fc7 layer*

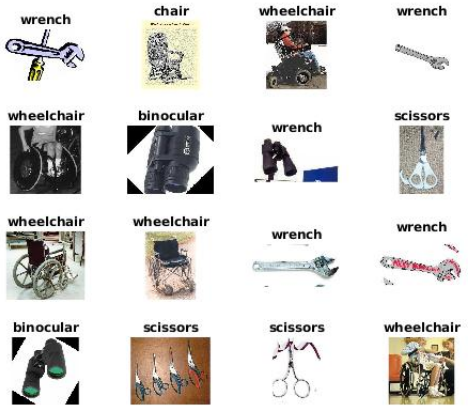

Fig. 5: *Example of classified images with fc8 layer*

	Accuracy
fc6	95%
fc7	93%
fc8	90%

TABLE I: *Accuracy for different layers*

Here are shown results of classification after feature extraction, for different extracted layers (from AlexNet) (Fig. 3, Fig. 4, Fig. 5).
In this case, the more the layer is deeper, the more the results are worst (Table I). This because deeper layers have learned too much specific features, and so we have loosen generality using them.

The code:
After loading the AlexNet pretrained network, we extract image features from different layers to test different choices. From the lower one they are: the 17-th *fc6*, the 20-th *fc7*, and the 23-th *fc8*, out of a total of 25 layers.
Extracted the features, we can train the classifier, and then predict to classify images.

C. Task 3

There are reason to use a pretrained network as initializer and not start from scratch. Fine-tuning a network with transfer learning is usually much faster and easier, and doable with a smaller number of training images.
Firstly, this is done taking the early layers of the pretrained networks and use as early layer of the new network. These first layers have learned primitive features like edges, blobs, colors and so they are still useful for our new network. Then we replace the final layers, that are more specific, with new final layers. The very last layer, the output one, must be chosen accordingly to the number of class of our data. Then learning can be faster and easier (less images are necessary) because we can choose to *freeze* some initial

layers, which means that in the training phase parameters for them are not computed; so the original ones from the pretrained set are not changed. Again, we use GoogLeNet with Caltech101 dataset. But we have to drastically reduce the data (5 categories left) to have results in a reasonable time.

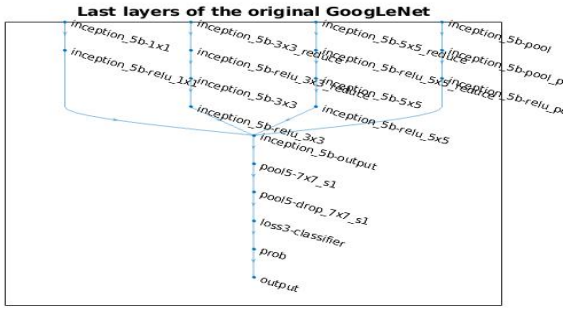

Fig. 6: *Last layers of the original network*

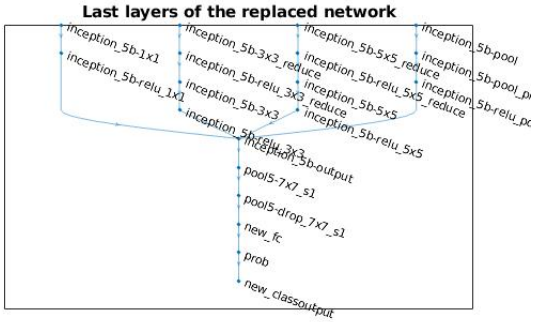

Fig. 7: *Last layers of the replaced network*

The code:

Following the Matlab tutorial:

- We load a pretrained set, the GooLeNet (last layers of the network in Fig. 6).
- We replace the last layers *loss-3-classifier* and *output* with new layers (Fig. 7). A parameter, *WeightLearnRateFactor* is settable to learn faster in the new convolutional layer than in the transferred layer. The other new layer (the output one, which classifies), has to be chosen accordingly to the number of our layers (5).
- We freeze the first 10 layers, so their weights are not modified. We tried with different number of frozen layers. The computation is faster when the number of the frozen layers is bigger (approximately one minute less with 120 frozen layers).
- We train the new network with our images, reducing the default number of epochs to train faster.
- Now the network is ready to classify new images (this is the testing phase) (Fig. 8, Fig. 9).

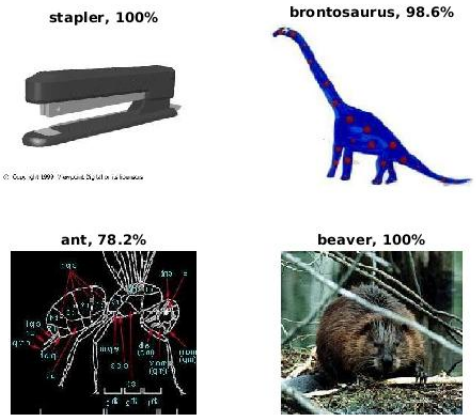

Fig. 8: *Example of classified images*

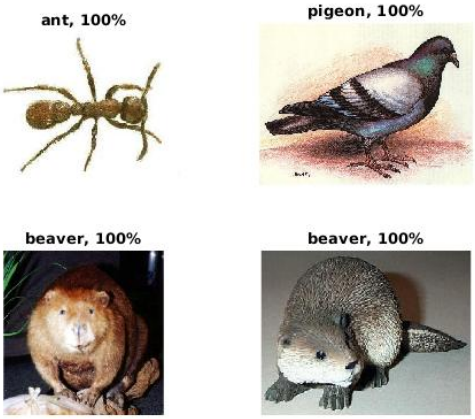

Fig. 9: *Example of classified images*