Technische Universität München **SS 16**
Fakultät für Informatik **Exercise Sheet 7**
Prof. N. Navab, Ph.D. **May 29, 2016**

Exercises in Praktikum Machine Learning - CNNs

Exercise 1 Setting things up

For this set of exercises you will use MatConvnet, a MATLAB toolbox for Convolutional Neural Networks (CNNs), which can be found here: `http://www.vlfeat.org/matconvnet/`. The most common layers encountered in CNNs are implemented within the toolbox (for an overview check `http://www.vlfeat.org/matconvnet/functions/`) and several pre-trained models are also supported. An extended documentation manual and provided tutorials/examples can be useful to help you familiarize with the toolbox and fulfill the exercises.

In order to complete the exercises, first download the latest version and follow the installation instructions (`http://www.vlfeat.org/matconvnet/install/`) in order to successfully compile the library using `vl_compilenn.m`. For this you need a C++ compiler on your system that is supported by your Matlab version. If you have a CUDA-capable GPU, it is preferable to compile the library with GPU support.

Exercise 2 Classification using a pre-trained model

To make sure that your library is well set up and get started, in this exercise you will use a pre-trained model for image classification.

a) Download the pre-trained ResNet-50 model with 50 weight layers, which can be found here: `http://www.vlfeat.org/matconvnet/pretrained/`.

b) The network follows the DagNN format. Write a proper function that loads the network and uses `eval.m` (in *test* mode) to classify the provided images. Be careful to resize the sample images to the network's input size. *Hint: The MatConvNet Quick Start guide should be helpful.*

c) Show the top 5 classification results for each image.

Exercise 3 Network Definition

In this exercise, you will implement the network depicted Figure 1. Please follow the dagNN structure. The layers that you should use (conv, ReLU, pooling) are illustrated with the corresponding symbols. FC represents fully-connected layers, that is a convolution with the same kernel size as the previous layer's activations (feature maps). Inputs to the network are expected to be RGB images of resolution 64x64. The rest of the sizes depicted in the figure are the expected sizes of the feature maps, with the third dimension being the number of channels.

After you implement and initialize the network, you should visualize the network for the expected input size and calculate the receptive fields for each layer with respect to the input, using the appropriate functions.

Exercise 4 Training

Read and understand the example in `examples/cifar/cnn_cifar.m` from the MatConvNet folder. Modify the example to train on CIFAR-10 but only classifying cat or non-cat.

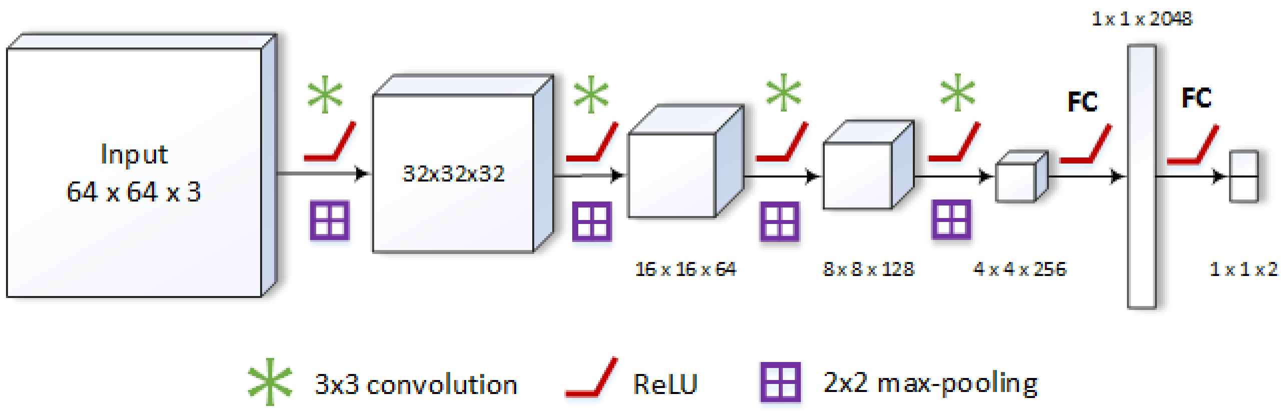

Figure 1: A simple and small CNN architecture.

Basically training a cat vs. background classifier. The network and dataset should be small enough that you can train it on a simple laptop without GPU support in a couple of hours. Play with the learning rate and batch size to improve convergence during training. Test your cat classifier on the images of the test set and report accuracy.

For better performance (training time and accuracy) you might want to exclude some non-cat images so that you have approximately the same amount of cat and non-cat images.

Here are a couple of things that you will need to change to make this work:

a) Use the dagNN structure.

b) Modify the dataset to only two classes: cat and non-cat instead of 10.

c) Change the network architecture so that the output is only a 2 element vector instead of 10.

d) Remove the top-5 error.

e) Adapt the hyper parameters like learning rate and batch size.

f) Rebalance the dataset to contain as many cat as non-cat images.

Exercise 5 Feature Visualization

In this exercise we will implement a very simple visualization technique based on:

Zeiler, Matthew D. and Fergus, Rob.
"*Visualizing and understanding convolutional networks*"
arXiv abs/1311.2901, 2013.
`http://arxiv.org/abs/1311.2901`.

Read the paper and try to understand the different concepts. Here we are interested in the input occlusion technique. The idea is to systematically cover parts of the input image with a gray box and measure how much the output (classification) changes. The intuition is that the rate of change in the output tells us the importance of the occluded region.

Figure 2 is an example of the expected output. Every square shown in the image was occluded once and the rate of change of the classification (class *sax, saxophone*) was measured. It is shown on right as a simple heatmap and on the left superimposed on the image. One can see that the actual saxophone is highly important for the image to be classified as *sax,*

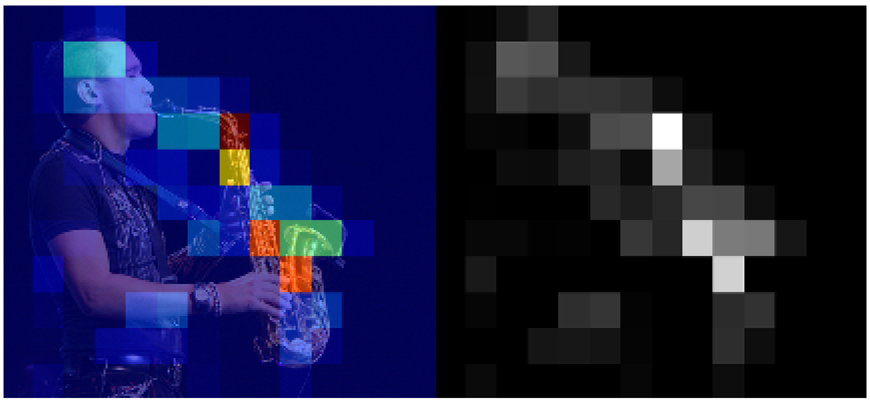
Figure 2: Input occlusion on a saxophone image.

saxophone although the network was never told what to look at, it learned what part of the image makes it classify as saxophone.

Implement the occlusion technique and visualize the heatmap on some images using pre-trained classification models. It is also worthwhile to comapre different architectures (AlexNet, VGG, GoogLeNet, ResNet) against each other.