Technische Universität München
Fakultät für Informatik
Prof. N. Navab, Ph.D.

WS 16/17
Exercise Sheet 5
December 24, 2016

Exercises in Praktikum Machine Learning - CNNs

Exercise 1 Setting things up

For the set of exercises in Convolutional Neural Networks (CNNs), you will work on TensorFlow™, which is an open source library for numerical computation using data flow graphs. TensorFlow comes with an extensive documentation as well as tutorials and examples that can help you familiarize with the library, understand the mechanics and fulfill the exercises. A Python API is also available, including a Neural Network branch where common CNN layers are provided (a list is available at: `https://www.tensorflow.org/api_docs/python/nn/`).

To complete the following exercises, first download and install the latest version, following these instructions. Ubuntu/Linux, Mac OS X and Windows are all supported. Alternatively, you could get it from the GitHub repository. If you have a CUDA-capable GPU, it is preferable to install TensorFlow with GPU support.

Exercise 2 Classification using a pre-trained model

To make sure that your library is well set up and to get started, in this exercise you will use a pre-trained model for image classification. You can read and follow this tutorial. Here, we are interested in networks trained on the ImageNet classification data including 1000 classes.

a) Use the pre-trained Inception-v3 architecture to classify some images.
Hint: study `classify_image.py`.

b) Show the top-5 classification results for each image.

Exercise 3 Network Definition

In this exercise, you will implement the network depicted in Figure 1. The layers that you should use – convolution, ReLU, pooling – are illustrated with the corresponding symbols (order is top to bottom). FC represents fully-connected layers, which can be implemented by reshaping the input tensor (i.e. a set of feature maps) into a batch of vectors and then multiplying by a weight matrix and adding biases. Inputs to the network are expected to be RGB images of resolution 64×64. The rest of the sizes depicted in the figure are the expected sizes of the feature maps (after each set of operations takes place), where first two dimensions represent the spatial resolution and the third is the number of channels.

After you implement the network structure and randomly initialize its weights, you should use an interface called TensorBoard in order to visualize the graph. More detailed information on this tool's usage is also provided by the TensorBoard README.

Exercise 4 Training

In the following you will you will train and employ a network for image recognition. CIFAR-10 is common a benchmark dataset for this problem.

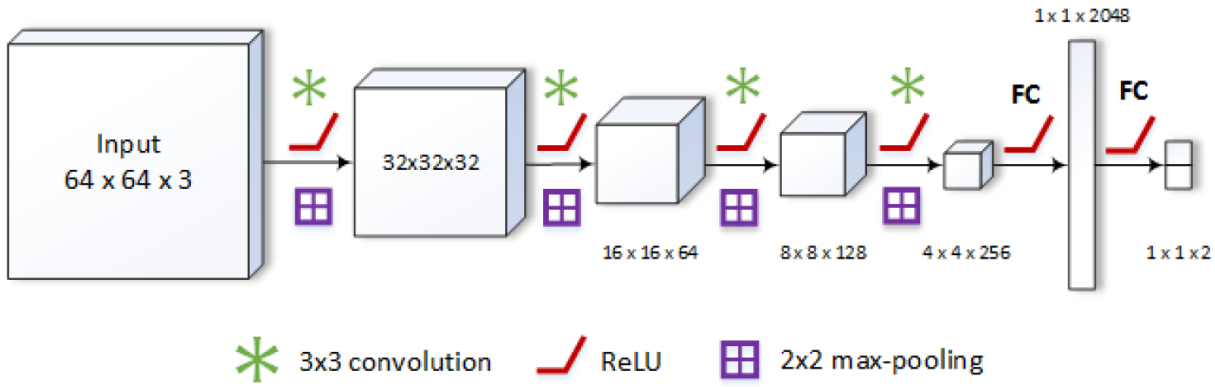

Figure 1: A simple and small CNN architecture.

First, read and understand the CIFAR example provided by TensorFlow (the multi-GPU version is not needed). Modify the example to train on CIFAR-10 but only classifying cat or non-cat; basically implementing a cat vs. background classifier. The network and the dataset should be small enough such that you can train on a simple laptop without GPU support in a couple of hours. 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 positive and negative samples.

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

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

b) Rebalance the dataset to contain as many cat as non-cat samples

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

d) Adapt the hyper-parameters like learning rate and batch size

e) Verify the trained model for classification on images that the network was not trained with (i.e. test set)

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"
European Conference on Computer Vision. Springer International Publishing, 2014.
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 main idea is to systematically cover parts of the input with a gray box and measure how much the output (classification score) changes. The intuition is that the rate of change in the output suggests the importance of the occluded region.

Figure 2 is an example of the expected result. Every square shown in the image was occluded once and the rate of change of the classification (class *sax, saxophone*) was measured. This is shown as a simple heatmap on the right and superimposed on the image on the left. One can see that the actual saxophone is highly important for the image to be classified as *sax, saxophone*. Although the netowkr was never told what to look at, it learned what part of the image makes it classify as saxophone.

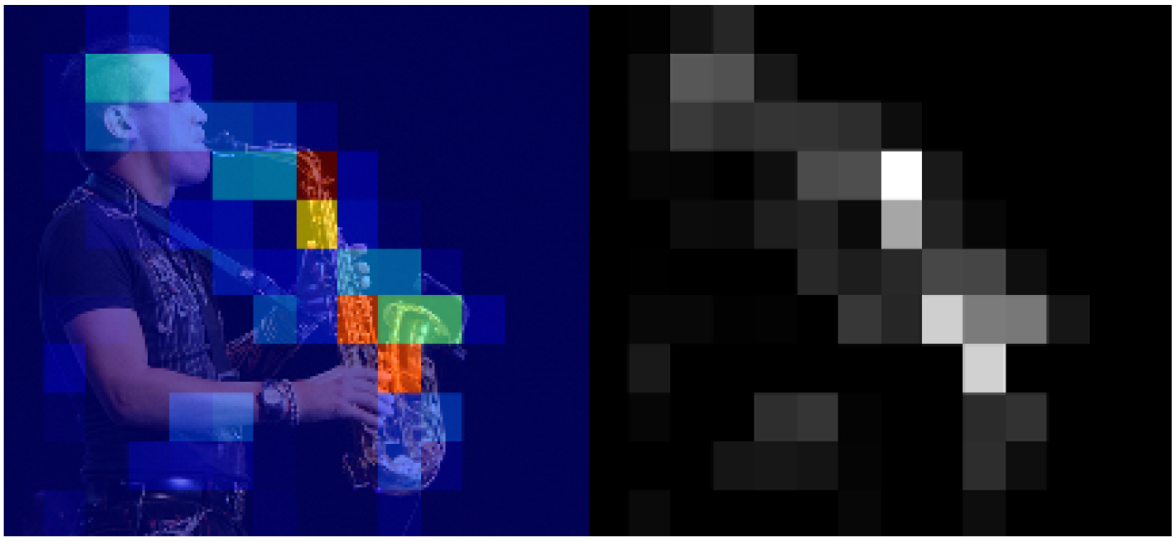

Figure 2: Input occlusion on a saxophone image.

Implement the occlusion technique and visualize the heatmap on some images using the Inception-v3 pre-trained classification model. Optionally: It is also worthwhile to compare different architectures (e.g. AlexNet, VGG, GoogleNet, ResNet) against each other. For this part, you would have to work with TF-Slim, where more classification models are available.