Bauhaus-Universität Weimar

Assignment 6

Deep Convolutional Neural Networks for Scene Classification

Convolutional neural networks (CNN) have shown to be valuable for many different tasks like natural language processing as well as image-based object and scene recognition. In order to learn more about CNN's, their properties and possible architectures, you have to train, test and adjust a given CNN. For the task of scene recognition, we provide representative images from 15 different classes: 'bedroom', 'coast', 'forest', 'highway', 'livingroom', 'mountain', 'office', 'opencountry', 'store', 'street', 'suburb' and 'tallbuilding' (Figure 1).

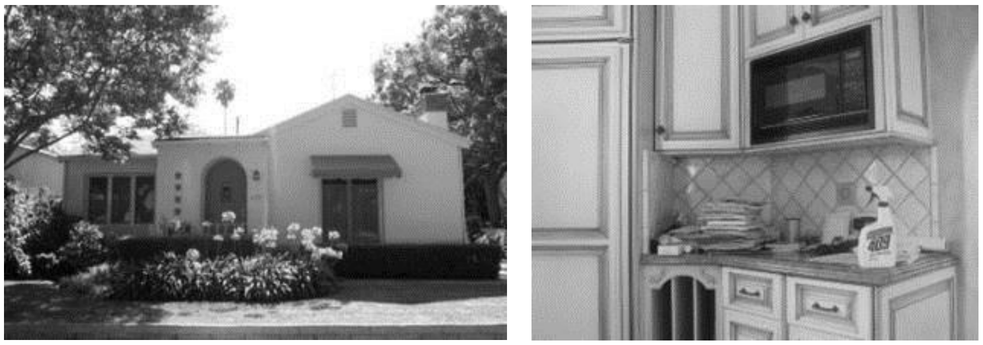

Figure 1: Example images for classes 'suburb' and 'kitchen'

The goal is to maximize the CNN classification accuracy using validation data, i.e., data which was not used for training. Try to improve the CNN so that you reach a classification accuracy around 70 %.

1. **Install MatConvNet library** (only available for MATLAB, version 1.0-beta20) from

http://www.vlfeat.org/matconvnet/

For installing and compiling simply follow the instructions on

http://www.vlfeat.org/matconvnet/install/#installing-and-compiling-the-library

```
> cd <MatConvNet>
> addpath matlab
> savepath
> vl_compilenn
```

2. **Download the starter code** as well as the required dataset from our homepage (password: IAOR.login2017). Copy the folder `15SceneData` to `matconvnet-1.0-beta20\data`. Copy the provided source files to `matconvnet-1.0-beta20\examples\IAOR_A6`. The dataset contains 15 different scene classes. For each class 100 training and even more test images are provided.

3. **Run MATLAB** as administrator (`mkdir` is called) and change to your `matconvnet` folder (`cd`).

Dr. Jens Kersten
jens.kersten@uni-weimar.de

4. **Run the main fuction** (`IAOR_A6.m`) to train (and also test) the provided CNN from scratch. The training will produce three different plots (Figure 2). In each training epoch, the 1500 training images are randomly subdivided into batches of 50 images. A final "loss layer" backpropagates the estimated loss (the true class memberships are known) in each iteration and the weights are updated correspondingly. The top1 error describes, how often the highest scoring guess is wrong. Answer the following questions:
    - **a.** Describe the axes in brief.
    - **b.** What is the reason for the differences of the graphs train and val?
    - **c.** Increase the number of epochs. What can be observed?
    - **d.** After the training in `IAOR_A6()` we are interested, if the CNN really performs as bad as shown in the plots. Test the trained CNN by hand using some randomly chosen images of the validation set. Hints:
        - i. The training was done using down sampled images (64x64 pix), so your test images need the same pre-pocessing (`imresize`).
        - ii. The final layer type is "`softmaxloss`", which is only needed for training. The type has to be changed before testing, for example with

```
net.layers{end}.type = 'softmax';
```

        - iii. Use the function `vl_simplenn` for testing. The 15-element score vector of the final layer can be obtained by `scores = squeeze(gather(res(end).x))`

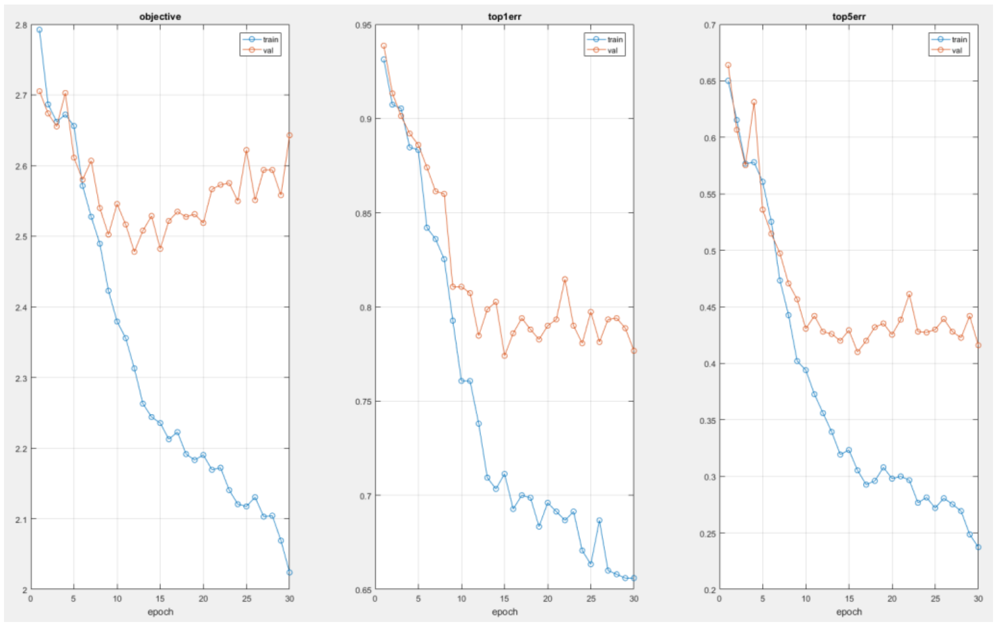

Figure 2: Result Plot provided by IAOR_A6.m

5. **Understanding the CNN**: The following layer types are used here: *convolution, max pooling* and *non-linearity* (RELU: rectified linear unit). Furthermore, after the initialization of the net in `IAOR_A6_cnn_init()`, the architecture is listed in the MATLAB Command Window using the function `vl_simplenn_display`. Use this information to describe how an image is processed by the CNN.

What is the input and output of each layer? You may also try to visualize the process, like done here (http://vision03.csail.mit.edu/cnn_art/index.html).

6. **Improving the classification accuracy**: Unfortunately, the results are really as bad as shown in Figure 2. We now try to improve the performance step by step. Choose **at least three** of the five strategies proposed below. Document all steps and the corresponding achieved improvements!

   a. **Number of training images**: We don't have more training images available. A common strategy to synthetically increase the number of images is to transform them. Test the impact of using randomly flipped training image sets in `getBatch()` using the functions `rand` and `fliplr`. Because the training and test errors decrease more slowly now, you may need more training epochs and/or you may try modifying the learning rate.

   b. **Zero-centering of images**: Simply subtract the mean gray-value from each image in order to obtain a zero-centered image. This is a very useful step in pre-processing. Explain why!

   c. **Regularization of training**: Especially with a high number of epochs, the train error tends to converge to zero while the val top1 error hovers around 40-50 %. This is caused by overfitting, i.e., our CNN works quite good for the training data but cannot be used for unseen data of the same class (bad generalization). Since the number of available training data is limited here, so-called dropout regularization can be used instead. Add the following additional layer directly before the last convolutional layer:

```
net.layers{end+1} = struct('type', 'dropout', 'rate', 0.5);
```

   This randomly turns off network connections at training time to reduce overfitting and therefore prevents a unit in one layer from relying too strongly on a single unit in the previous layer. The dropout rate describes the proportion of connections that are randomly deleted.

   d. **Increasing the deepness of the initial CNN**: Only two layers of our net actually contain learned weights. Other proposed nets contain up to 19 of these layers. However, arranging and parameterize them is not an easy task. Add more layers to the net. A common approach is to use a combination of a "convolutional", "max-pool" and "relu" layer, like already done in the starter code. Caution: The weights, pool and stride variables have to be adjusted, so that the data size in final layer (the one before the softmax layer) is 1 and data depth is 15 (see table output of `vl_simplenn_display`). In general you have to make sure that the data depth output by any channel matches the data depth input to the following channel. Hint: the max-pooling operation covers a window of 7x7 and then is subsampled with a stride of 7. Networks usually do not subsample by more than a factor of 2 or 3 each layer. This might be one parameter to adjust.

   e. **Image size**: All images are resized to 64x64 pixels for training and testing which is good in terms of computational time but actually reduces the level of details drastically. Adjust the whole network in order to process full-size or half-size images.