1 Computer Vision: Day 3 - Convolutional Neural Networks

After having used the very powerful HOG descriptors yesterday, today you will let the computer find out on its own which image features are important in order to solve the classification problem. While the definition of the according models in not difficult, training these models certainly is. We will rely on a library that was particularly designed to simplify the training process: tensorflow. You will learn how to define flow graphs in tensorflow, how to train them and use, evaluate, interpret, and visualize the final result.

1.1 Libraries

For today, the following libraries can and should be used to work on the exercises:

- `six.moves.cPickle` — a library for fast serialization
- `numpy` — a library for efficient matrix operations
- `matplotlib.pyplot` — a library for displaying images and plotting graphs
- `cv2` — a library for basic image processing operations
- `scipy` — a library for scientific computations
- `sklearn` — a library for machine learning
- `joblib` — a library for memoisation
- `tensorflow` — a library for deep learning

You can use other libraries that are installed on the lab computers.

1.2 Exercises

You can and are encouraged to reuse code from the days before, e.g., the functions to load images from GTSRB and split the data into training, validation, and test set.

1. Until you are sure that your graph is training correctly, it is advised to only train on a small toy example.
2. Today, you can use all color channels, if you like.
3. Define a multi-layer fully connected neural network to classify the traffic signs and evaluate its performance.
4. Define a multi-layer network with convolutional layers, max-pooling layers and in the end fully connected layers and evaluate its performance.
5. Make sure to train your favorite model on a sufficiently large training set and save the model for tomorrow. This may take half an hour depending on your hardware.

1.3 Hints

1. Until you are sure that your graph is training correctly, it is advised to only train on a small toy example.
2. If you have operations that take a long time, consider memoization in order to not wait for them more than once.
3. `tensorflow` works a lot like `numpy`, just for symbolic variables. Many functions even carry the same name.

1.4 Functions

Some of the following functions may or may not come in handy when working on the exercises. Have a look at the documentation if they seem worth the while.

`numpy.mean` — Computes the average value of an array along a given axis
`numpy.std` — Computes the standard deviation of an array along a given axis
`tensorflow.placeholder,Variable,constant` — Building blocks for all tensorflow models
`tensorflow.reset_default_graph` — Useful for checkpointing (to save the model)
`tensorflow.truncated_normal` — Generates a few random values (useful for initializing variables)
`tensorflow.nn.conv2d` — Performs a convolution like the one you implemented on Day 1
`tensorflow.nn.max_pool` — Reduces the image size and only keeps the maximum values from each shrinked rectangular region
`tensorflow.nn.relu` — Only propagates the values that are greater than 0
`tensorflow.cast` — Changes the type of a `tensorflow`-Variable
`tensorflow.nn.softmax_crossentropy_with_logits` — Softmax-Crossentropy-Loss (do not write that yourself)
`tensorflow.train.AdamOptimizer` — A slightly more sophisticated gradient descent algorithm to optimize a model
`tensorflow.train.Saver` — Helper object for saving checkpoints of the training process to the hard disk
`tensorflow.train.Saver.save` — Stores the current model and its state to the hard disk
`tensorflow.train.Saver.restore` — Restores the current model and its state from the hard disk
`tensorflow.global_variables_initializer` — Creates an operation that if executed initializes the entire model
`tensorflow.Session` — Class representing a running version of the defined model
`tensorflow.Session.run` — Starts the model with a certain configuration of placeholders