1 Day 1: Computer Vision Basics

Welcome to the Computer Vision Lab. In the course, we are going to work out some of the principal techniques of modern computer vision using a wide-spread dataset for traffic signs, the German Traffic Sign Recognition Benchmark (GTSRB). In order to quickly gain productive code, we will implement the algorithms purely in Python relying on some widespread libraries for image processing and optimization.

We start the course with some basic but essential image processing operations: input/output operations, type conversions, convolutions, max pooling, and binary classification.

1.1 German Traffic Sign Recognition Benchmark

GTSRB is a benchmark dataset to evaluate the capabilities of video-based driver assistance systems aiming at traffic sign recognition, i.e., the task of providing a traffic sign category to an image containing (basically only) the very same traffic sign. Explicitly, we do not cover the task of detecting, i.e., finding the position of, the traffic sign within a cluttered recording of a street scene. This step is preceding the one we will be covering in this course. The dataset contains the huge number of 39,209 traffic sign images each assigned one of 43 possible classes. The dataset provides variance in multiple ways: the distance to the sign and, hence, the size and resolution of its image, the time of day and weather conditions during recording, occlusions, damages and dirt, to name a few. This makes the problem challenging, even for humans who accomplish the task with an error rate of 1.16%. The images in the original dataset are of different sizes, however, to help you with your first steps in computer vision, the images are all scaled to a resolution of $32 \times 32 \times 3$ pixels where the last dimension denotes the color channels Red, Green, and Blue.

For more information on the benchmark, please refer to: `http://benchmark.ini.rub.de`

1.2 Libraries

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

- `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, **please use only `scipy.misc.toimage` from this library today**

You can use other libraries that are installed on the lab computers, but please **do not use tensorflow today**.

1.3 Exercises

On the first day, we would like to start with smaller steps and tackle only part of the traffic sign recognition problem. In fact, we would like to first distinguish the two sign classes "Speed Limit 30" and "Speed Limit 50". As both sign classes are circular, have a red border, and do in fact only differ in a single symbol, this is already quite tricky. Hence, many approaches tend to confuse these two categories.

1. Before you start: **Please read the hints at the end of this introduction!**
2. Download the dataset and code from `http://benchmark.ini.rub.de/Dataset/GTSRB_Final_Training_Images.zip`. If you want to use the CIP computers, it should be on the desktop. Use the function `load_gtsrb_images` to get images for the classes. The function is not documented, but maybe you can find out how it works. Display either 5 random examples of a "Speed Limit 30" and of a "Speed Limit 50".

3. Today's approach will not rely on color. Convert all images to grayscale.

Our strategy for distinguishing the two sign classes for today is to look for the digits 3 and 5 in the images. For now we are going to try to identify the digits 3 and 5 on the traffic sign images. For this, we are going to deploy a convolution. First we are going to need a template:

4. Find out an index of a nice "Speed Limit 30" and of a "Speed Limit 50", write down the coordinates of a rectangular region (or region of interest — ROI), extract them, and display them.
5. Prepare the ROIs for convolution and compute both convolutions with each of the traffic sign images from both classes. Store the maximum response for both convolutions for all images. You now have a two-dimensional feature representation for your training set.
6. Plot the feature representation of your training set. Mark the two classes with different markers. What do you see?
7. Choose parameters $w \in \mathbb{R}^2, b \in \mathbb{R}$ by hand to define a useful classifier $w^T x > b$ where x is the feature vector of a traffic sign image consisting of the maximum convolution response with the 3 and 5 ROI.
8. Predict the classes of the images with the classifier you choose. Draw another plot with the feature vectors of all your images. Use different colors for correctly and incorrectly predicted classes.
9. Compute and display the error rate of your classifier.
10. Show a random choice of misclassified images (if there are any).

1.4 Hints

1. If you are stuck for a longer time, call for one of the supervisors to help you out.
2. Test your code with a toy problem. Reduce the amount of data that is loaded and processed to as few examples as possible. It is nothing more time-consuming (and frustrating) to wait a long time for your results to only see your program exit in an error. If you have a random operation in your code, you can pseudo-randomize it by using a random seed.
3. You may reuse code that you write today, so keep in mind that you might deal with more than two classes tomorrow.
4. Be aware that the numbers in arrays have a defined data type and the effect of arithmetic operations may not be what you expected. In fact, when read from hard disk, the images will all be given in `uint8`.
5. Image coordinates are often a source of confusion. Pay attention that images as numpy-Arrays will be ordered row-wise meaning that the upper left (not lower left!) pixel of an image has the coordinates $[0, 0]$, the first number denotes the row index (vertical!), the second number the column index (horizontal!).
6. Color spaces are another source of error. OpenCV predominantly uses the BGR color space where Blue and Red are flipped with respect to the usual RGB ordering. The display function `matplotlib.pyplot.imshow` may also misinterpret numpy-Arrays. Use the function `scipy.misc.toimage` if necessary.

1.5 Functions

You should be familiar with the basic concepts of the Python programming language. However, 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.

`six.moves.cPickle.dump` — Write anything to a stream (e.g., a file)
`six.moves.cPickle.load` — Load anything from a stream (e.g., a file)
`numpy.array.astype` — Casts the type of a numpy-Array
`numpy.random.randint` — Generates a random integer
`numpy.linalg.norm` — Computes the norm of a vector
`numpy.mean` — Computes the mean value of a vector
`numpy.squeeze` — Removes unit dimensions from a high-dimensional array
`numpy.ravel` — Reshapes a multidimensional array to a one-dimensional
`matplotlib.pyplot.figure` — Opens a new display
`matplotlib.pyplot.imshow` — Shows an image
`matplotlib.pyplot.plot` — Plots curves and lines
`matplotlib.pyplot.show` — Draws the display
`cv2.cvtColor` — Converts images between color spaces