In this class you will need to do one project and one article reading that will be presented at the end of the semester. Both project and article cannot be on the same topic. A project is expected to take between 15 and 30 hours, and can be implemented in any imperative programming language you want (however in this project using python is recommended). Readings go much more in-depth than the class lectures, and are thus more complex.

1 Readings

- Tags2Parts: Discovering Semantic Regions from Shape Tags, CVPR 2018 https://arxiv.org/abs/1708.06673

2 Project - Image classification

In this project we study two ways of performing pedestrian/non-pedestrian image classification, first with Support Vector Machine and then with a Deep Learning classifier. You can compare both methods.

The data for this project is available at: https://perso.liris.cnrs.fr/julie.digne/cours/MLdata.zip Images are 128x64 and divided into pedestrians and nonpedestrians, and training data and test data. For both SVM and Neural net, use the training data to optimize the model and test data to check that the classifier works. For python use opencv for HOG descriptor and numpy for matrix computation. Skeleton codes and utilities are available at: at https://perso.liris.cnrs.fr/julie.digne/cours/MLcode.zip

2.1 SVM

Write a SVM based on the HOG descriptors of the images. A code skeleton is provided in python and in matlab. Disclaimer: Matlab is way slower. You are strongly encouraged to work in python with pytorch.
2.2 Deep Learning

Now we’ll do the same with a very simple CNN classifier. A code skeleton is provided in python. You’ll need pytorch, you can run it on the CPU or on the GPU (using google collab or kaggle). A skeleton code is provided.

Dependencies: numpy, torch, torchvision, PIL.

2.2.1 Model

The architecture is as follows:

1. Convolution2d \( (\text{Kernel}_\text{size} = 5 \times 5, \text{output}_\text{channels} = 6) + \text{MaxPool2d} (\text{kernel}_\text{size} = 2) + \text{Relu} \)

2. Convolution2d \( (\text{Kernel}_\text{size} = 5 \times 5, \text{output}_\text{channels} = 16) + \text{MaxPool2d} (\text{kernel}_\text{size} = 2) + \text{Relu} \)

3. Linear layer \( (\text{output}_\text{channels} = 120) + \text{Relu} \)

4. Linear layer \( (\text{output}_\text{channels} = 60) + \text{Relu} \)

5. Linear layer \( (\text{output}_\text{channels} = 1) + \text{Relu} \)

6. sigmoid

There are two important parts in the Classifier class:

- The definition of the class itself with all the layers (that’s where all the parameters to be optimized are defined)
- A forward function that defines how an input is processed throughout the network.

Refer to the pytorch documentation, you’ll need the following functions:

- torch.nn.Conv2d
- torch.nn.Linear
- torch.nn.functional.relu (we’ll use torch.nn.functional for functions that do not need to be trained)
- torch.nn.functional.max_pool2d
- torch.nn.functional.sigmoid

2.2.2 Training

We’ll optimize the Binary Cross Entropy (torch.nn.BCELoss), and use Stochastic Gradient Descent (torch.optim.SGD)