Lab Work - SVM vs DeepLearning for pedestrian detection

Julie Digne

September 2020

Data for this lab work 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`.

1 SVM

Write a SVM based on the HOG descriptors of the images. A code skeleton is provided in python. You can also choose to do it in matlab, a hog_feature_vector function and a skeleton code is provided.

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.

2.1 Model

The architecture is as follows:

1. Convolution2d ($Kernel\_size = 5x5, output\_channels = 6$) + MaxPool2d ($kernel_size = 2$) + Relu
2. Convolution2d ($Kernel\_size = 5x5, output\_channels = 16$) + MaxPool2d ($kernel_size = 2$) + Relu
3. Linear layer ($output\_channels = 120$) + Relu
4. Linear layer ($output\_channels = 60$) + Relu
5. Linear layer ($output\_channels = 1$) + 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 Training

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