## Research Engineer Assignment

The goal of this exercise was to classify the test set of a database of sign posts.

## Choice of Algorithm

Considering the type of classification problem using images directly is possible and in this case the problem is very well suited to the use of Convolutional Neural Networks. CNNs usually take fixed size input and the data is composed of data of different size. In order to get a uniformity of picture size I decided to scale it to my decided input size of a 64 pixels wide square image and fill the blanks with noise. This way the pictures are not stretched and the network normally will not learn any pattern from the noisy fill ups.

Another notion that is important to keep in mind is the fact that we have very limited data to begin with; with 10 classes and only 100 images per class. We obviously need to employ regularization methods to make the algorithm work. We could also augment the data we have by scraping for more data or using data augmentation methods such as rotating the images slightly (in the order of 2 to 10 degrees), changing the colour balance slightly or even applying filters to the images. We could then artificially increase the data we have by creating slightly different copies of it.

## Preprocessing

In order to make the data the same size, I first scale the picture without changing its ratio so that its maximum dimension is 64 pixels. I then paste that picture on a salt and pepper noisy picture. This results in pictures that are not stretched in any way and are all the same format.

## First Approach

At first, I separated the data between training and testing myself with a random split and got some acceptable results with a conventional network of 3 convolutional layers and 2 fully connected layers. The results are included in the file log\_random\_initialization.txt. With a validation accuracy of 95%, it seemed like it offered a very good solution to the problem. Sadly, I had not seen that the test/train data split had already been done.

## Second Approach

Using the original split we get much worse results (as seen in log\_original\_initialization.txt) this is due to the fact that the test/train data split is not balanced, with some classes only having 20 images in the training set.

In [8]: np.sum(labels\_test,0)

Out[8]: array([ 20., 20., 80., 20., 80., 20., 20., 20., 20., 80.])

In [9]: np.sum(labels\_train,0)

Out[9]: array([ 80., 80., 20., 80., 20., 80., 80., 80., 80., 20.])

## Final Approach

I just inverted the test and train text files for those classes and got much more coherent results (explicated in log\_corrected\_initialization.txt). We get a final result of 97% accuracy on validation data with 100% on training data. It seems that without data augmentation or additional training data, passing this threshold will be slightly difficult.

## Potential Improvements

With additional time, I would have introduced some form of data augmentation, most probably a rotation of the image of 2.5 and 5 degrees in either direction to multiply the data size by 5. I would potentially also try different structures for the CNN (I have trained all the CNN here using my cpu on a laptop, thus did not try anything too exotic because of the slow convergence rate).

## How to run the code

The actual CNN is self contained in the signs\_classifier.py file. It can be run by using %run signs\_classifier.py in a python interface (I used ipython personally).

The other files are data\_control.py which scans the data and figures out what the images path and the related targets are and preprocess\_data.py which performs the preprocessing task I explain in the preprocessing part of the report. They can both be run directly with no arguments. Running preprocess\_data.py will overwrite the images with the processed ones. I kept a copy of the original data in a zip file in case of corruption.

If you just wish to run the entire code on a new set of images, you should add a folder in the CV\_ML\_Excersise\_Signs folder with the 3 .py files and run in order data\_control.py, preprocess\_data.py and signs\_classifier.py.

You can also had inference testing (sanity testing) at the end of signs\_classifier.py by adding another test\_labels and test\_images sets and using them the way I did to calculate accuracy-test.