Traffic Sign Recognition with Multi-Scale Convolutional Networks

Chris Andrew

Introduction

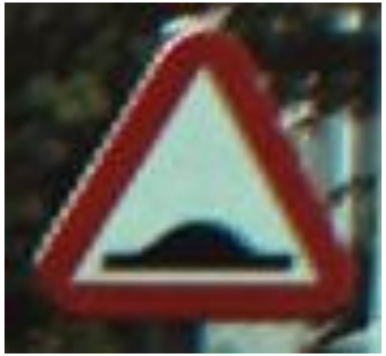
Traffic sign recognition has direct real-world applications such as driver assistance and safety, urban scene understanding, automated driving, or even sign monitoring for maintenance.

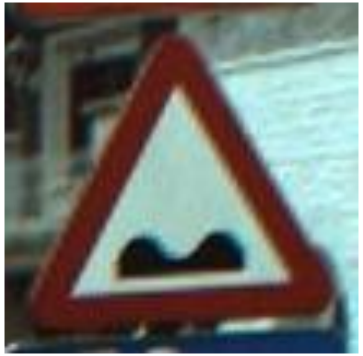
The problem can be broken into two subproblems: **sign detection** and **recognition**

Detection is analogous to localization or image segmentation where we must identify a Region of Interest.

Recognition is difficult

Due to a similar format and layout of traffic signs, there are only small differences between different traffic signs that the neural network must learn to capture.





Detection is easy

It is a constrained image recognition problem, as there is less variability between images of the same class. This is because signs are unique, rigid and intended to be clearly visible for drivers, and have little variability in appearance.

Previous approaches

Computationally-expensive sliding window approaches that solve the detection and classification problems simultaneously. Not very famous, no real time application of this approach. Easy to implement.

Detection is first handled with computationally-inexpensive, hand-crafted algorithms, such as color thresholding. Classification is subsequently performed on detected candidates with more expensive, but more accurate, algorithms.

End to End recognition

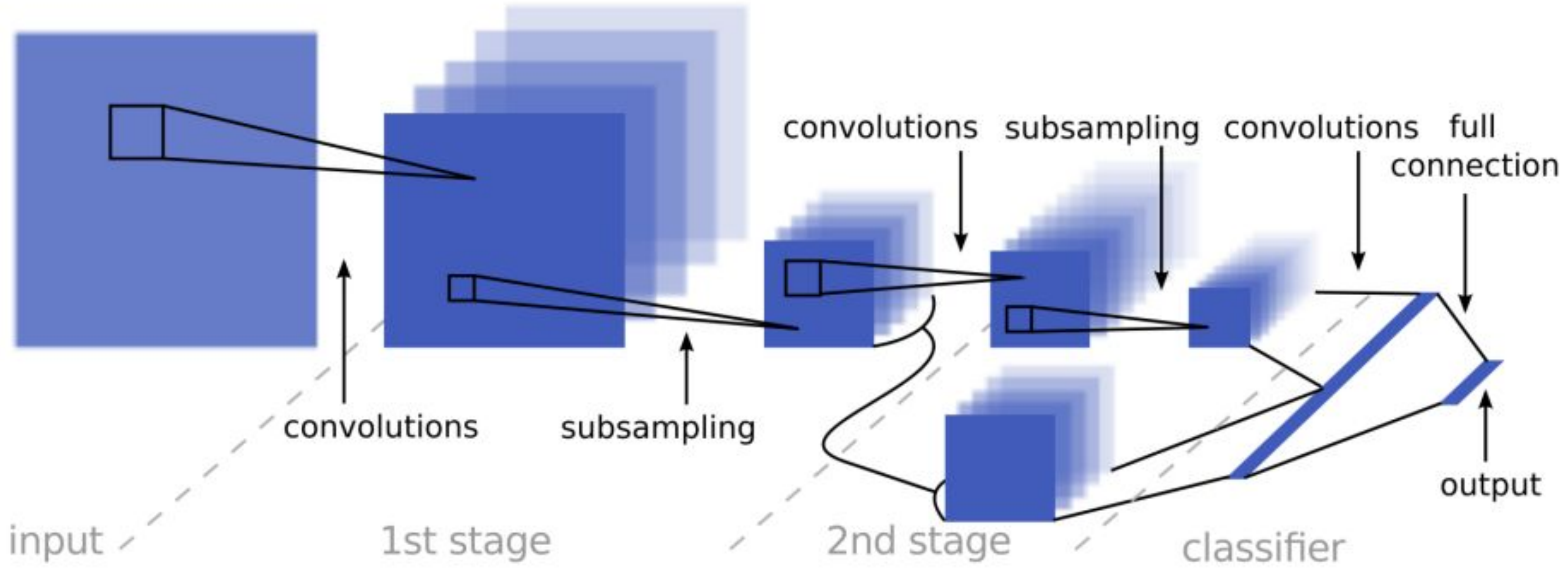
Solves both detection and classification problems together.

No need to separately train networks for detection and then for classification.

More difficult to do using normal algorithms.

Deep learning and convolutional networks allow us to train an end to end pipeline.

Multi-Scale ConvNets



Dataset

The dataset used in the paper was the German Traffic Sign Benchmarks dataset

- 43 classes
- 50,000+ images (32x32x3)

The dataset I used is the BelgiumTS dataset

- 61 classes
- 5000+ images (32x32x3)

70
STOP

Libraries I used

- Tiny-dnn library from OpenCV was used for this project.
  - Highly Optimised
  - C++ (very fast)
  - Does not need GPU for training
- OpenCV for processing the images
- Matplotlib for visualisation

Results

Training was done on **4 cores (i7-6700) clocked at 3.4GHz each.**

Total training time: **50 hours approx.**

Number of epochs: **8571**

Average time per epoch: **21.0010s**

Training accuracy: **97.52%**

Test accuracy: **94.59%**