Traffic Sign Recognition

Văduva Vlad, Hriţu Toma, Iordache Constantin, Granu Dragoș, Karp Adrian

December 22, 2023

Contents

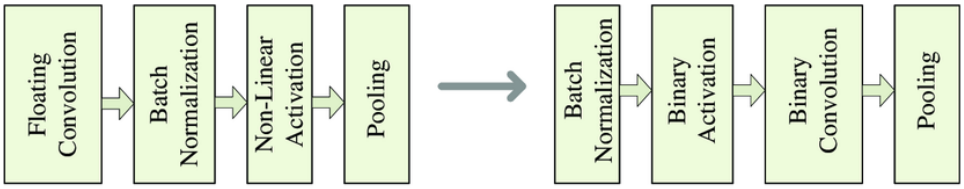
1 Benchmark

1.1 Introduction

Traffic sign recognition is an integral part of any vision system for autonomous driving. As its name entails, Traffic sign recognition is a benchmark for traffic sign classification, which consists of two parts: isolating the bounding box and classifying the sign. This neural network focuses on the latter.

1.2 Structure

This Traffic Sign Recognition uses XNOR-Net architecture. A XNOR-Net network is a binary CNN (BNN) that achieves high accuracy for bigger datasets. The following pictures illustrates the architecture difference between a typical CNN and XNOR-Net. The main difference of the new architecture is that it uses the layers in a different order, and since it is a BNN it has a Binary Activation layer.

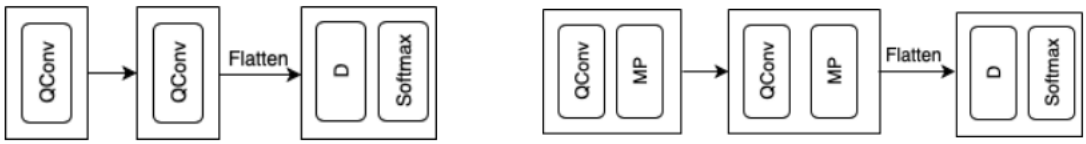


Layer sequence in typical CNN Restructured Layer sequence in XNOR-Net

In XNOR-Net training phase, binary weights are used during forward pass and backward propagation. Full precision weights are used only for the gradient calculation.

XNOR-Net relies on binarizing the input activation and weights by converting them into either $+1$ or -1 using a sign function defined as $x^b = \text{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0 \\ -1, & \text{if } x < 0 \end{cases}$, where x is a real-valued weight or activation and x^b is the binarized output. To directly use logical operations for computing XNOR-Net, all -1s are encoded to 0s.

The considered XNOR architectures for this benchmark are the following:



(a) XNOR(QConv) architecture (b) XNOR(QConv,MP) architecture

Each is composed of two internal blocks and an output dense (fully connected) layer. XNOR architectures assures accuracy of at least 70%. The MP layers have numerous parameters, the model size is big, hence the reduced accuracy when MP is present.

1.3 Dataset

Traffic sign recognition uses the GTSRB dataset for training and testing BNN architectures.
GTSRB is a multi-class, single-image dataset. The dataset consists of images of German road signs in 43 classes. Each class comprises 210 to 2250 images including prohibitory signs, danger signs, and mandatory signs. For training and validation the ratio 80:20 was applied to the images in the train dataset.
The test dataset for Belgium Traffic Signs contains 7095 images of 62 classes and only 23 matches the classes from German dataset.
The Chinese Traffic Signs test dataset contains 5998 traffic sign images for testing of 58 classes, out of wich only 15 match the ones from German Traffic signs
In the end, the Traffic Sign Recognition dataset uses 1818 images from the Belgium dataset and 1590 from the Chinese dataset.
For input, shape is fixed either to 30px × 30px, 48px × 48px, or 64px × 64px.

2 Marabou

Marabou is a neural network verification tool. It is an SMT-based tool that can answer queries regarding the properties of a neural network, transforming these queries into constraint satisfaction problems. Marabou can accommodate networks with different activation functions and topologies, performing high-level reasoning on the network, which can reduce the search space and improve performance.

Features:

- Reachability queries: If inputs are within a certain interval, the output will be guaranteed to be within a typically safe range.
- Robustness queries: Tests whether there are adversarial points for a given input point, that would change the output of the neural network.
- Marabout supports fully connected feed-forward and convolutional neural networks, in the .nnet and TensorFlow formats. Properties can be specified using inequalities over input and output variables or through the Python interface.

2.1 Marabou Linux Installation

Prerequisites: CMake To install Marabou, we first must clone the GitHub repository from the following address: `https://github.com/NeuralNetworkVerification/Marabou`. We create a directory named 'build' by running

```
mkdir build'
```

, and inside this folder we run

```
cmake ..'
```

to compile the tool. Then, to build the static library Marabou, we run

```
cmake .. -DBUILD_STATIC_MARABOU=ON.
```

3 Alpha-Beta-CROWN

Alpha-Beta-CROWN is a neural network verification tool based on an efficient linear bound propagation framework and branch and bound. It can be accelerated efficiently on GPUs and scales well for big convolutional neural networks.
Supported neural network architectures:

- Layers: fully connected, convolutional, pooling, transposed convolution
- Activation functions: ReLU, sigmoid, tanh, arctan, sin, cos, tan
- Residual connections and other irregular graphs

3.1 Alpha-Beta-CROWN Installation

Prerequisites: Conda, miniconda works as well Miniconda Installation: We create a folder named miniconda3:

```
mkdir -p ~/miniconda3
```

We must download miniconda from the main site:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O ~/miniconda3/miniconda.
```

We run the installtaion script:

```
bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
```

After we installed the prerequisites, we can proceed with alpha-beta-crown installation: We clone the GitHub repository:

```
git clone --recursive https://github.com/Verified-Intelligence/alpha-beta-CROWN.git
```

This way, we also clone auto-LiRPA sub-module, which will then be installed with the command

```
 python setup.py install
```

Remove the old environment, if necessary:

```
conda deactivate
```

```
conda env remove --name alpha-beta-crown
```

Install all dependents into the alpha-beta-crown environment:

```
conda env create -f complete_verifier/environment.yaml --name alpha-beta-crown
```

Activate the environment

```
conda activate alpha-beta-crown
```

4 Running a benchmark

We will run Marabou in Linux with a provided test benchmark to see that the tool is running correctly. After installing and building Marabou, we will enter from command line:

```
build\Release\Marabou.exe resources\nnet\acasxu\ACASXU_experimental_v2a_2_7.nnet
```

```
resources\properties\acas_property_3.txt
```

```
(base) [tom@manjarohtv marabou]$ cd Marabou
(base) [tom@manjarohtv Marabou]$ ./build/Marabou resources/nnet/acasxu/ACASXU_ex
perimental_v2a_2_7.nnet resources/properties/acas_property_3.txt
Network: resources/nnet/acasxu/ACASXU_experimental_v2a_2_7.nnet
Number of layers: 8. Input layer size: 5. Output layer size: 5. Number of ReLUs:
 300
Total number of variables: 610
Property: resources/properties/acas_property_3.txt

Engine::processInputQuery: Input query (before preprocessing): 309 equations, 61
0 variables
Engine::processInputQuery: Input query (after preprocessing): 609 equations, 838
 variables

Input bounds:
        x0: [ -0.3035,  -0.2986]
        x1: [ -0.0095,   0.0095]
        x2: [  0.4934,   0.5000]
        x3: [  0.3000,   0.5000]
        x4: [  0.3000,   0.5000]

Branching heuristics set to LargestInterval
unsat
```