



UNIVERSITATEA DE VEST DIN TIMIȘOARA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ  
PROGRAMUL DE STUDII DE MASTERAT: Inginerie  
Software

# PROIECT VERIFICARE FORMALĂ

**COORDONATOR:**Erașcu Mădălina

**STUDENȚI:**Hăloiu Patricia Elena, Gall Robert Ioan, Marko-  
vican Andreia, Enea Andrei Laurentiu, Bota Rafael Dorin

TIMIȘOARA  
2023

UNIVERSITATEA DE VEST DIN TIMIȘOARA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ  
PROGRAMUL DE STUDII DE MASTERAT: Inginerie  
Software

# Traffic Signs Recognition

COORDONATOR: Erașcu Mădălina

STUDENȚI: Hăloiu Patricia Elena, Gall Robert Ioan, Markovican Andreia, Enea Andrei Laurentiu, Bota Rafael Dorin

TIMIȘOARA  
2023

## Abstract

Rețelele neuronale au revoluționat modul în care sunt implementate sistemele complexe, ușurând astfel munca dezvoltatorilor, multi dintre software engineers opteaza acum sa utilizeze rețelele neuronale profunde (DNN-uri) care reprezintă modele de învățare automată create prin antrenarea unor seturi de date.

O interogare de verificare a unei rețele neuronale profunde constă în două părți: (i) o rețea neurală și (ii) o proprietate care trebuie verificată; iar rezultatul său este fie o garanție formală că rețeaua satisface proprietatea, fie o intrare concretă pentru care proprietatea este încălcată (un contraexemplu). Există mai multe tipuri de interogări de verificare pe care Marabou [?] le poate rezolva, precum interogările de accesibilitate, unde dacă inputul se află într-un anumit interval, outputul este garantat a fi într-un anumit interval sau interogările de robustețe, unde verifică dacă există puncte contradictorii în jurul unui anumit input care modifică outputul rețelei.

# Capitolul 1

## Introducere

În acest capitol o să ne familiarizăm cu toolurile și benchmark-ul, vom prezenta metoda de instalare a acestora și dificultățile întâmpinate în acest proces.

### 1.1 traffic\_signs\_recognition

”Benchmark-ul ”Traffic Signs Recognition” este un set de date de clasificare în mai multe categorii desfășurată la IJCNN 2011.

Este folosit pentru recunoașterea automată a semnelor de circulație, el este necesar în sistemele avansate de asistență pentru șoferi și reprezintă o problemă reală complexă în domeniul viziunii artificiale și al recunoașterii modelelor.

Acest set de date este format din peste 50,000 de imagini cu semne de circulație. Acesta reflectă variațiile puternice în aspectul vizual al semnelor datorită distanței, iluminării, condițiilor meteorologice, occluderilor parțiale și rotațiilor. Imaginile sunt completate de mai multe seturi de caracteristici precalculate pentru a permite aplicarea algoritmilor de învățare automată fără cunoștințe de bază în prelucrarea imaginilor.

Setul de date cuprinde 43 de clase cu frecvențe neechilibrate ale claselor. Participanții trebuie să clasifice două seturi de testare, fiecare conținând peste 12,500 de imagini.” [Min11]

### 1.2 Marabou

Marabou este un solver care se bazează pe satisfiabilitatea constrangerilor (SMT), acest solver poate gestiona rețele cu diferite funcții, de ex activare și topologii pentru efectuarea unui rationament asupra rețelei, ceea ce optimizează prin reducerea de spațiu de căutare și îmbunătățirea performanței.

Referitor la verificările unei rețele neuronale, Marabou le poate rezolva prin întrebări de atingere, adică intrările dintre un anumit interval sunt asigurate de ieșirile dintr-un anumit interval, acest lucru de obicei este sigur și de asemenea mai sunt întrebările de robustețe care reprezintă teste ale punctelor adversare de intrare pentru a vedea dacă schimbă ieșirea rețelei.

## 1.3 alpha-beta-CROWN

alpha-beta-CROWN este un solver de rețele neurale bazat pe o abordare matematică care se concentrează pe extinderea limitelor matematice ale funcțiilor liniare pentru a evalua comportamentul rețelelor neurale și pe tehnica de ramificare și limitare, adică se împart problemele complexe în subprobleme mai mici și mai ușor de rezolvat.

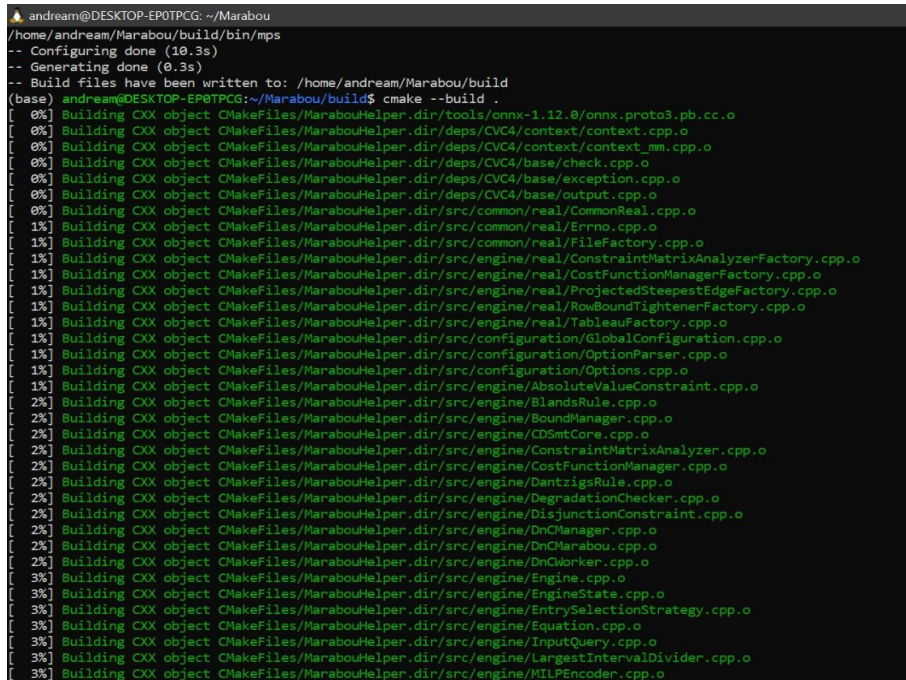
Poate fi eficient pentru viteze de calcul pe GPU și poate scala la rețele relativ mari. De asemenea, susține o gamă largă de arhitecturi de rețele neurale, datorită bibliotecii auto\_LiRPA.

## Modul de instalare și dificultățile întâlnite

În acest capitol prezentăm modul prin care echipa noastră a instalat tool-urile și ce erori am întâmpinat la acest pas. Pentru instalarea Marabou, am descărcat și utilizat WSL (Windows Subsystem for Linux). Am rulat comenzile necesare pentru instalare în cadrul WSL. Pasul inițial a constat în clonarea repository-ului Marabou folosind comanda **git clone https://github.com/NeuralNetworkVerification/Marabou**. Următorul pas a fost executarea comenzilor disponibile pe GitHub, în ordinea următoare:

```
cd path/to/marabou/repo/folder
mkdir build
cd build
cmake ..
```

În urma executării acestor comenzi, rezultatul final ar fi trebuit să fie o instalare completă a Marabou, așa cum este ilustrat în figura de mai jos.



```
andream@DESKTOP-EP0TPCG: ~/Marabou
/home/andream/Marabou/build/bin/mps
-- Configuring done (10.3s)
-- Generating done (0.3s)
-- Build files have been written to: /home/andream/Marabou/build
(base) andream@DESKTOP-EP0TPCG:~/Marabou/build$ cmake --build .
[0%] Building CXX object CMakeFiles/MarabouHelper.dir/tools/onnx-1.12.0/onnx_proto3.pb.cc.o
[0%] Building CXX object CMakeFiles/MarabouHelper.dir/deps/CVC4/context/context.cpp.o
[0%] Building CXX object CMakeFiles/MarabouHelper.dir/deps/CVC4/context/context_mm.cpp.o
[0%] Building CXX object CMakeFiles/MarabouHelper.dir/deps/CVC4/base/check.cpp.o
[0%] Building CXX object CMakeFiles/MarabouHelper.dir/deps/CVC4/base/exception.cpp.o
[0%] Building CXX object CMakeFiles/MarabouHelper.dir/deps/CVC4/base/output.cpp.o
[0%] Building CXX object CMakeFiles/MarabouHelper.dir/src/common/real/CommonReal.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/common/real/Errno.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/common/real/FileFactory.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/real/ConstraintMatrixAnalyzerFactory.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/real/ConstraintMatrixManagerFactory.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/real/ProjectedSteepestEdgeFactory.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/real/RowBoundTightenerFactory.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/real/TableauFactory.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/configuration/GlobalConfiguration.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/configuration/OptionParser.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/configuration/Options.cpp.o
[1%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/AbsoluteValueConstraint.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/BlandsRule.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/BoundManager.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/CDSSmtCore.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/ConstraintMatrixAnalyzer.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/ConstraintMatrixManager.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/DantzigRule.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/DegradationChecker.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/DisjunctionConstraint.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/DnManager.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/DnMarabou.cpp.o
[2%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/DnGlobber.cpp.o
[3%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/Engine.cpp.o
[3%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/EngineState.cpp.o
[3%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/EntrySelectionStrategy.cpp.o
[3%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/Equation.cpp.o
[3%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/EquationQuery.cpp.o
[3%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/LargestIntervalDivider.cpp.o
[3%] Building CXX object CMakeFiles/MarabouHelper.dir/src/engine/MILPDecoder.cpp.o
```

Figura 1.1: build *Marabou*

De asemenea, a fost testată funcționalitatea tool-ului Marabou, după instalare, conform imaginii de mai jos.

```
(base) andream@DESKTOP-EP0TPCG:~$ cd Marabou
(base) andream@DESKTOP-EP0TPCG:~/Marabou$ cd build
(base) andream@DESKTOP-EP0TPCG:~/Marabou/build$ ./build/Marabou resources/nnet/acasxu/ACASXu_experimental_v2a_2_7.nnet resources/properties/acas_property_3.txt
-bash: ./build/Marabou: No such file or directory
(base) andream@DESKTOP-EP0TPCG:~/Marabou/build$ cd ..
(base) andream@DESKTOP-EP0TPCG:~/Marabou$ ./build/Marabou resources/nnet/acasxu/ACASXu_experimental_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, 610 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
```

Figura 1.2: „Test benchmark” ACASXu” folosind Marabou

Pentru instalarea alpha-beta-CROWN, am folosit tot WSL, ca și în cadrul Marabou. Am descărcat și instalat miniconda, conform documentației alpha-beta-CROWN, după am făcut clone la alpha-beta-crown din repository folosind comanda **git clone https://github.com/Verified-Intelligence/alpha-beta-CROWN**. În următorul pas, am făcut clone la auto lirpa folosind comanda **git clone https://github.com/Verified-Intelligence/auto-LiRPA.git**, în urma clonării, am folosit comenzile pentru setup auto-LiRPA:

```
cd auto_LiRPA
```

**python setup.py install** După încheierea setup-ului, am început să introducem comenzile pentru alpha-beta-Crown, începând cu instalarea tuturor dependențelor în environment-ul alpha-beta-CROWN, folosind comanda **conda env create -f complete-verifier/environment.yaml --name alpha-beta-crown**, urmând să îl și activăm folosind **conda activate alpha-beta-crown** Pentru a vedea dacă am urmat pașii în mod corect, am încercat să testez un exemplu și anume: **cifar\_resnet\_2b.yaml** folosind următoarele comenzi:

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

```
cd complete_verifier
```

```
python abcrown.py --config exp_configs/cifar_resnet_2b.yaml
```

```

BaB round 13
batch: 1280
Average branched neurons at iteration 13: 1.0000
splitting decisions:
split level 0: [/23, 6] [/25, 9] [/25, 9] [/23, 6] [/38, 59] [/38, 59] [/38, 59] [/38, 59] [/38, 59] [/38, 59]
pruning in iteration open status: True
ratio of positive domain = 1332 / 2560 = 0.5203125
pruning-in-iteration extra time: 0.015761613845825195
Time: prepare 1.5648 bound 1.1570 transfer 0.0226 finalize 0.8646 func 3.6092
Accumulated time: func 15.5672 prepare 4.1473 bound 9.1948 transfer 0.0890 finalize 2.1434
Current worst splitting domains lb-rhs (depth):
-1.17321 (21), -1.16208 (21), -1.15939 (21), -1.15913 (21), -1.15691 (21), -1.15581 (21), -1.15061 (21), -1.15044 (21), -1.14785 (21), -1.14428 (21), -1.14
401 (21), -1.14343 (21), -1.14276 (21), -1.13984 (21), -1.13979 (21), -1.13870 (21), -1.13797 (21), -1.13478 (21), -1.13475 (21), -1.13330 (21),
length of domains: 1229
Time: pickout 0.0137 decision 2.3558 set_bounds 1.1001 solve 3.6105 add 0.0569
Accumulated time: pickout 0.0559 decision 8.0479 set_bounds 3.4950 solve 15.5732 add 0.3726
Current (lb-rhs): -1.1732077598571777
5089 domains visited
Cumulative time: 27.61503767967224

```

Figura 1.3: cifar-resnet-2b.yaml

După rularea exemplului, am rulat benchmark-ul traffic signs recognition. Am început prin a instala alfa-beta-CROWN folosind următoarele comenzi:

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

```
cd alpha-beta-CROWN
```

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

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

După instalarea verifier-ului, am instalat pachetul DNNV, de care vom avea nevoie pentru a converti modelele VGG în VNN-COMP 2023 folosindu-ne de comanda

```
pip install -U --no-deps git+https://github.com/dlshriver/DNNV.git@4d4b124bd739b4ddc8c68fed1af3f85b90386155#egg=dnnv
```

Urmând să clonăm benchmark-urile și script-urile VNN-COMP, folosindu-ne de comenzile:

```
git clone https://github.com/ChristopherBrix/vnncomp2023_benchmarks.git
```

```
./setup.sh
```

După rularea acestor comenzi, am reușit să rulăm tool-ul alpha-beta-crown folosindu-ne de comanda

```
./run_all_categories.sh v1 /home/laur/miniconda3/envs/
alpha-beta-CROWN/vnncomp_scripts $(pwd) results_vit.csv
./counter-examples "traffic_signs_recognition" first
```

```

(alpha-beta-crown) laur@DESKTOP-2UGHQW9:~/miniconda3/envs/alpha-beta-CROWN/complete_verifier/vnncomp2023_benchmarks$ ./run_all_categories.sh v1 /home/laur/miniconda3/envs/alpha-beta-CROWN/vnncomp
scripts $(pwd) results vit.csv ./counter-examples "traffic signs recognition" first
Running measurements with vnncomp folder '/home/laur/miniconda3/envs/alpha-beta-CROWN/complete_verifier/vnncomp2023_benchmarks' for tool scripts in '/home/laur/miniconda3/envs/alpha-beta-CROWN/vnncomp
scripts' and saving results to 'results vit.csv'.
Running traffic signs recognition category from /home/laur/miniconda3/envs/alpha-beta-CROWN/complete_verifier/vnncomp2023_benchmarks/benchmarks/traffic_signs_recognition/instances.csv
Category 'traffic signs recognition' timeout sum: 21600 seconds
./run_all_categories.sh: line 91: bc: command not found
Doing run_single_instance with category 'traffic signs recognition' on onnx network '/home/laur/miniconda3/envs/alpha-beta-CROWN/complete_verifier/vnncomp2023_benchmarks/benchmarks/traffic_signs_
recognition/onnx/3_30_30_QConv_16_3_QConv_32_2_Dense_43_ep_30.onnx' with vnnlib file '/home/laur/miniconda3/envs/alpha-beta-CROWN/complete_verifier/vnncomp2023_benchmarks/benchmarks/traffic_signs_
recognition/vnnlib/model_30_idx_7040_eps_1.00000.vnnlib' and timeout '480.0'
/home/laur/miniconda3/envs/alpha-beta-CROWN/complete_verifier/vnncomp2023_benchmarks/run_single_instance.sh: line 47: bc: command not found
/home/laur/miniconda3/envs/alpha-beta-crown/bin
Preparing alpha-beta-CROWN for benchmark instance in category 'traffic signs recognition' with onnx file '/home/laur/miniconda3/envs/alpha-beta-CROWN/complete_verifier/vnncomp2023_benchmarks/benc
hmarks/traffic_signs_recognition/onnx/3_30_30_QConv_16_3_QConv_32_2_Dense_43_ep_30.onnx' and vnnlib file '/home/laur/miniconda3/envs/alpha-beta-CROWN/complete_verifier/vnncomp2023_benchmarks/benc
hmarks/traffic_signs_recognition/vnnlib/model_30_idx_7040_eps_1.00000.vnnlib'
TOOL_DIR is /home/laur/miniconda3/envs/alpha-beta-crown
python3(418): Operation not permitted
python3(418): Operation not permitted
sudo: a password is required
sudo: a password is required
sudo: a password is required
sudo: a password is required
sudo: a password is required
sudo: a password is required
sudo: a password is required
sudo: a password is required
Thu Dec 21 21:40:47 2023

```

NVIDIA-SMI 545.29.04		Driver Version: 546.17		CUDA Version: 12.3		
GPU	Name	Persistence-M	Bus-Id	Disp.A	Volatile Uncorr. ECC	
Fan	Temp	Perf	Pwr:Usage/Cap	Memory-Usage	GPU-Util Compute M.	
					MIG M.	
0	NVIDIA GeForce RTX 3050	On	00000000:01:00:00	On	N/A	
N/A	54C	P8	7W / 65W	676MiB / 4096MiB	2%	Default
					N/A	

Processes:							GPU Memory
GPU	GI	CI	PID	Type	Process name		Usage
ID	ID						

Figure 1.4: traffic signs recognition



# Bibliografie

- [Min11] Ali A. A. A. Minai. The 2011 international joint conference on neural networks, 2011. Benchmark.