# DeepPAC: PAC based Robustness Analysis for DNNs

Renjue Li[1,2], Pengfei Yang[1], Cheng-Chao Huang[3,4],
Youcheng Sun[5], Bai Xue[1,2] and Lijun Zhang[1,2]
[1]SKLCS, Institute of Software, CAS [2]University of Chinese Academy of Sciences
[3]Nanjing Institute of Software Technology, ISCAS [4]Pazhou Lab, China [5]University of Manchester

## ABSTRACT

In this note, we give a brief introduction to DeepPAC for its artifact evaluation. DeepPAC is a tool to analyse robustness for deep neural networks (DNNs) with the probably approximately correct (PAC) guarantee. The main function of DeepPAC is to verify the so-called "PAC-model robustness property" for a DNN and an input region defined by an $L_\infty$ ball. This note is to help users quickly understand the basic information of DeepPAC (including the operating environment required) and reproduce our previous results. DeepPAC is available at https://github.com/CAS-LRJ/DeepPAC and the exported docker container can be found in https://drive.google.com/file/d/1LiyMdNrwB5AYVSCgnErRralTybEdTNGz/view?usp=sharing.

## 1 INTRODUCTION

DeepPAC is a tool to analyse robustness for deep neural networks (DNNs) with the probably approximately correct (PAC) guarantee, which is an an implementation of the algorithms in our technical article titled "Towards Practical Robustness Analysis for DNNs based on PAC-Model Learning". DeepPAC is now open source under the Apache-2.0 License, and we intend to apply the Reusable & Available Badges in the artifact evaluation.

The main function of DeepPAC is to verify the so-called "PAC-model robustness property" for a DNN and an input region defined by an $L_\infty$ ball. Additionally, the maximum robustness radius can be estimated via a binary search on the radius of the $L_\infty$ ball. If you are interested in how DeepPAC works, please refer to the technical paper [4].

## 2 REQUIREMENTS

DeepPAC is developed based on Python 3.7.8, which is compatible with both Windows and Linux. Users can install all dependencies via Conda. For the artifact evaluation, we provide a docker image for Linux that has already been initialized with all dependencies. However, some extra steps are needed for preparing the docker environment. The overall requirements are listed below:

- Linux Kernel: > 3.10
- Docker: >= 19.03
- RAM: 16GB (or better)
- GPU: Nvidia, Architecture >= Kepler, 6GB memory at least and supports CUDA 11.1
- Nvidia Driver: >= 418.81.07

Hereafter, we only provide the instructions to reproduce in the docker environment. If you want to install DeepPAC on physical machines, please refer to the readme document in our artifact.

*Nvidia Support for Containers.* DeepPAC utilises a GPU to sample on large DNNs, so the Nvidia Container Toolkit is required for Docker. The tutorial can be found here[1]. You can run the docker image without GPU support, but it will be extremely slow on large DNNs like ResNets.

*Gurobi Licenses for Containers.* In our previous experiments, we use Gurobi [1] as linear programming (LP) solver. The Gurobi package is already included in the docker image but it does not contain a valid license. Gurobi is academic free and you can apply an academic container license on the Gurobi Web License Manager[2] after you register. When you have the license file (`gurobi.lic`), you need to mount it in `/opt/gurobi` in the docker container. The tutorial can be found in Section 4.3 in the Gurobi documentation[3]. Note that you have to connect to a recognized academic institution network such as a university network for the license. If you have problems with Gurobi, we also provide an open-source solver CBC [3]. However, CBC could be much slower than Gurobi (up to 52 times slower).

## 3 USAGE

To run DeepPAC in docker, you first need to import the docker image from the tar ball and create a container correspondingly.

```
sudo gunzip -c DeepPAC.tar.gz | sudo docker \
import - deeppac:v1
sudo docker run -it --gpus all \
--security-opt seccomp:unconfined deeppac:v1 bash
cd ~/DeepPAC
conda activate DeepPAC
python . -h
```

Note that the flag secomp:unconfined boosts the performance of python programs in containers. DeepPAC could be 100% slower without this flag.

We provide some examples to facilitate the use of DeepPAC . You can modify the augments of the examples to verify other networks on different inputs with different parameters. You can find the corresponding network files in the `models` folder according to the readme document in our artifact.

*MNIST Case.* Verify the PAC-model robustness of CNN1 on the first image of the MNIST training set with radius 1, error rate 0.01, and confidence 0.999. The number of samples used for the first focused learning phase is 2000, and 7000 for the second phase. We use Gurobi as the LP solver.

```
python . -ncf models.mnist_ERAN_convs \
-nc ConvSmallNetwork \
```

[1]https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#docker
[2]https://license.gurobi.com/manager/licenses
[3]https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/9.5/quickstart_linux.pdf

```
-m ./models/convSmallRELU__Point.pth -d mnist \
-r 1 -eps 0.01 -eta 0.001 -ind 0 -train \
-FT 2000 -ST 7000 -solver gurobi
```

*CIFAR-10 Case.* Verify the PAC-model robustness of Resnet18 on the 10th image of the CIFAR-10 testing set with radius 4, error rate 0.01, and confidence 0.99. The number of samples used for the first focused learning phase is 20000, and 8000 for the second phase. Here we use GPU to sample on Resnet18 and set the batchsize to be 200. We use CBC as the LP solver this time.

```
python . -ncf models.resnet -nc ResNet18 \
-m ./models/Cifar_ResNet18_adv_trim.pth -d cifar10 \
-r 4 -eps 0.01 -eta 0.01 -ind 9 -gpu \
-FT 20000 -ST 8000 -bsize 200 -solver cbc
```

*Imagenet Case.* Verify the PAC-model robustness of Resnet50a on a given church image with radius 3, error rate 0.005, and confidence 0.999. The number of samples used for stepwise splitting is 18000. Here we use GPU to sample on Resnet50 and set the batchsize to be 200.

```
python . -ncf torchvision.models -nc resnet50 \
-m ./models/imagenet_linf_4.pth -d imagenet \
-img ./ImageNet_Samples/church_0.jpeg \
-r 3 -eps 0.005 -eta 0.001 -gpu \
-b 18000 -bsize 200
```

## 4 REPRODUCTION

All the experiments are conducted on a physical machine with Intel I5-11500, Nvidia RTX 3060 and 16GB RAM, and the results are presented in the `logs` folder. The running time is at the bottom of the log file. Note that a different running environment may bring a little change to the results. We provide 5 python files to reproduce all the experiments.

*MNIST Maximum Robustness Radius.* The first experiment calculates the maximum robustness radius $r_{\max}$ for different networks on the MNIST dataset (See Fig 6). We use `mnist_experiments.py` to reproduce this experiment. It has four augments

- -n: the network to be verified (CNN1-6, FNN1-6),
- -gpu: set to use GPU,
- -inc: set to verify the properties with the radius $r_{\max} + 1$,
- -solver: choose the LP solver (gurobi, cbc).

For instance, you can examine the results of the maximum robustness radius of FNN1 as below:

```
python mnist_experiments.py -n FNN1 -solver gurobi
### 25 PAC-Model Robust out of 25 cases.
python mnist_experiments.py -n FNN1 -inc -solver gurobi
### 0 PAC-Model Robust out of 25 cases.
```

The result shows that the maximum robustness radius of FNN1 in our previous experiment is correct. It takes nearly 2400 seconds per run in our settings.

*CIFAR Robustness Rate.* The second experiment estimates the robustness rate of CIFAR-10 CNNs on 100 images with radius 2, 4, 6, 8 (See Fig.7). We use `cifar_batch.py` to reproduce this experiment. It has four augments

- -n: the network to be verified (CNN1-3),
- -gpu: set to use GPU.
- -r: the radius of the $L_\infty$ ball (2, 4, 6, 8),
- -solver: choose the LP solver (gurobi, cbc).

Simply run the python file with augments to estimate the robustness rate. It takes nearly 20000 seconds per run in our settings.

*CIFAR Verification with Different Parameters.* The third experiment tries to calculate the maximum robustness radius of the same image with different parameters (See Tab.2). This experiment is reproduced by `cifar_different_parameters.py`. It has four augments

- -n: The network to be verified (ResNet18, ResNet50, ResNet152),
- -gpu: set to use GPU,
- -ind: the image index (0-4),
- -solver: choose the LP solver (gurobi, cbc).

Simply run the python file with augments to check the maximum robustness radius with different parameters. It ts nearly 2300 seconds per run in our settings.

*Imagenette Targeted Verification.* The fourth experiment tries to verify 20 images with radius 4 on a subset of imagenet dataset (imagenette [2]) using the targeted score difference function (See Targeted Method of Tab.3). We use `imagenette_experiments.py` to reproduce this experiment. It has two augments

- -n: the network to be verified (ResNet50a, ResNet50b),
- -gpu: set to use GPU.

Simply run the python file with augments to check the verification results. It takes nearly 28000 seconds per run in our settings.

*Imagenet Untargeted Verification.* The fifth experiment tries to verify 50 images with radius 4 on the Imagenet dataset using the untargeted score difference function (See Untargeted Method of Tab.3). We use `imagenet_experiments.py` to reproduce this experiment. It has two augments

- -n: The network to be verified (ResNet50a, ResNet50b),
- -gpu: set to use GPU.

Simply run the python file with augments to check the verification results. It takes nearly 31600 seconds per run in our settings.

## 5 EXTRAS

We put extra guidelines in the readme document in our artifact including FAQs and trouble shooting. Please refer to the document if you have any problem. We highly recommend to install the Nvidia Container Toolkit and Gurobi to release the full performance of DeepPAC . Otherwise, the running time of the experiments could be intolerable.

## REFERENCES

[1] L. Gurobi Optimization. Gurobi optimizer reference manual, 2021.
[2] J. Howard. The imagenette dataset, 2019.
[3] johnjforrest, S. Vigerske, H. G. Santos, T. Ralphs, L. Hafer, B. Kristjansson, jpfasano, EdwinStraver, M. Lubin, rlougee, jpgoncal1, h-i gassmann, and M. Saltzman. coin-or/cbc: Version 2.10.5, Mar. 2020.
[4] R. Li. Towards practical robustness analysis for dnns based on pac-model learning. *in ICSE 2022.*