# ReliableAI 2022 Course Project

The goal of the project is to design a precise and scalable automated verifier for proving the robustness of fully-connected, convolutional and residual networks with ReLU activation against adversarial attacks. We will consider adversarial attacks based on the $L_\infty$-norm perturbations. A network is vulnerable to this type of attack if there exists a misclassified image inside an $L_\infty$-norm based $\epsilon$-ball around the original image.

We will leverage the DeepPoly relaxation [1] for building our verifier. The ReLU transformer introduced in [1] is a DeepPoly transformer with a minimum area, if one considers a single neuron at a time. However, as explained in the lecture, the ReLU transformer is parametrized by the slope $\alpha$ of the linear lower bound and it is sound for any value $\alpha$ in [0, 1]. In this project, your task is to improve the ReLU transformer in DeepPoly. Your goal is to learn a value $\alpha$ for each neuron in the network which maximizes the precision of the verifier. Given the network and a test case (an image and a radius), your algorithm should first produce a new ReLU transformer (by learning $\alpha$) for each neuron in the network and then run the verification procedure using these transformers.

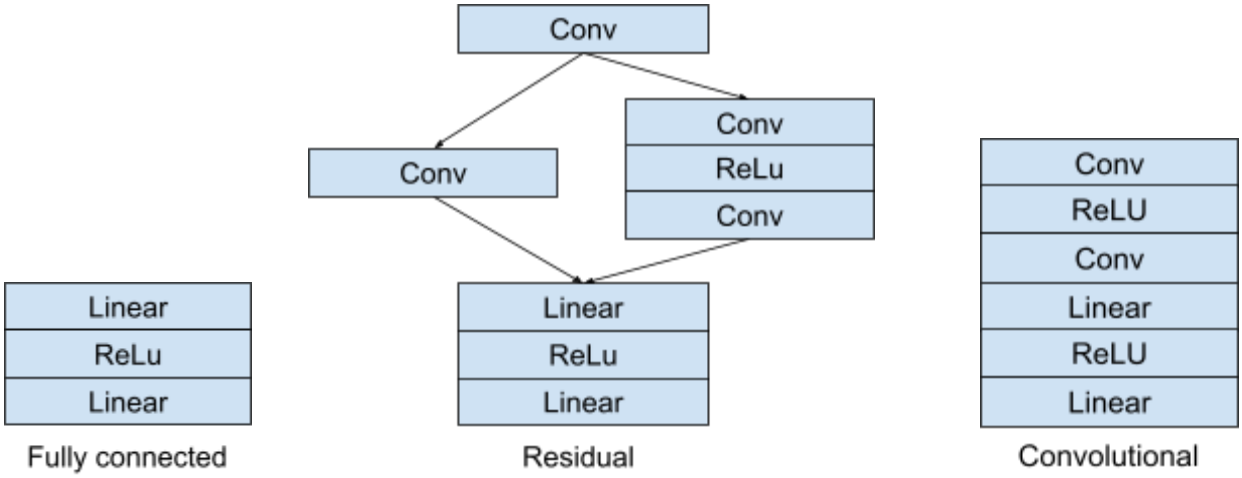
You are allowed to learn an ensemble or a set of different $\alpha$, run DeepPoly verification using each $\alpha$ separately, and then use the information from all runs to verify the robustness. Formally, you are allowed to verify an image if you can prove that all outputs in the intersection of final convex shapes computed using different sets of $\alpha$ are correctly classified.

## Project Task

The input to your verifier is a neural network and a test case. The output is the result of your verification procedure (verified/not verified). Below we provide more details.

**Datasets:** We consider 2 datasets: MNIST and CIFAR-10. MNIST consists of grayscale images of dimensions 28 x 28. CIFAR-10 consists of RGB images of dimensions 32 x 32.

**Networks:** We will run your verifier on 3 fully connected, 4 convolutional and 3 residual neural networks. The networks are trained using different training methods (e.g. standard training, PGD, DiffAI). The architectures and weights of these networks will be provided together with the code release. The architectures are encoded as PyTorch classes in `networks.py`. All trained networks are provided in the folder `nets`. Figure provides an overview of these architectures.




**Example Test Cases:** The example test cases consist of 20 specification files from the MNIST or CIFAR-10 test set, 2 tests for each network, formatted to be used by the verifier. Each specification file contains a single row that consists of comma-separated image label and pixel values. Each pixel value is an integer between 0 and 255. For MNIST image, there are 784 pixel values that represent 28 x 28 grayscale image, and for CIFAR-10 image, there are 3072 pixel values that represent 32 x 32 RGB image. The provided networks expect images with pixel values normalized between 0 and 1. In the file `verifier.py` we provide a function `get_spec` that reads the input specification and returns a normalized image in the format expected by the networks. All example test cases are stored in the folder `examples`.

The epsilon value of a test case can be deduced from its name, e.g., the file img1_0.1.txt defines the 0.1-ball around an image img1. We provide those example test cases for you to develop your verifier and they are **not the same** as the ones we will use for the final grading.
The epsilon value refers to the normalized pixel values (between 0 and 1). Note that you only have to verify images with pixel intensities between 0 and 1: e.g. if perturbation is 0.2 and pixel has value 0.9 then you only have to verify range [0.7, 1.0] for intensities of this pixel, instead of [0.7, 1.1]. File `gt.txt` contains ground truth for each example (output of the master solution).

**Verifier:** The directory verifier contains the skeleton code of the verifier (file `verifier.py`). The verifier addresses the following problem:

Inputs:

- A fully connected, convolutional or residual neural network
- A test case

Output:

- verified, if the verifier *successfully* proves that network is robust for the test case
- not verified, if the verifier *fails* to prove that network is robust for the test case

Your verifier will be executed using the following command, where <network> is replaced by the network identifier (`net1, net2, ..., net10`) and <test case file> is replaced by a file containing the test case (e.g. `../examples/net1/img1_0.0500.txt`).

```
$ python3 verifier.py --net <network> --spec <test case file>
```

**Your task is to implement a sound and precise robustness verifier by modifying the "analyze" function**. To this end, you must build upon the DeepPoly relaxation [1] as explained before. Your verifier must be **sound**: it must never output verified when the network is not robust to a test case. It should be **precise:** it should try to verify as many test cases as possible while keeping soundness and scalability (we will use a time limit of **1 minute** per test case).

## Testing Conditions and Grading

Your verifier will be tested on the following conditions:

- Inputs to the networks are images from the MNIST or CIFAR-10 dataset.
- $\epsilon$ values range between 0.0001 and 0.9999.
- Configuration of the machine used for testing: Intel Xeon E5-2690 v4 CPUs and 512GB RAM. We will use 14 threads and memory limit 64GB to verify each test case.
- Time limit for verification is 1 minute for each test case
- Your verifier will be executed using Python 3.7

The verifier will be graded based on the precision and soundness of your verifier:

- You start with 0 points.
- **You receive 1 point** for any verification task for which your verifier correctly outputs verified within the time limit.
- **You will be deducted 2 points** if your verifier outputs verified when the network is not robust for the provided test case.

- If your verifier outputs `not verified` you receive 0 points. This means that the maximum number of points that can be achieved by any solution may be less than 100.
- If there is a timeout or memory limit exceeded on a verification task, then the result will be considered as `not verified`.

We do not require your verifier to be floating-point sound and we guarantee that full points can be obtained using torch.float32. If you have questions about the requirements or grading, please ask on Moodle or send an email to mislav.balunovic@inf.ethz.ch.

## Requirements

You should obey the following rules otherwise you may get **0 points** for the project:

1. The implementation must be in Python 3.7.
2. You must use the DeepPoly relaxation. No other relaxations are allowed.
3. You are not allowed to check for counter-examples using any kind of adversarial attack.
4. The only allowed libraries are PyTorch 1.10.0, Torchvision 0.11.1, Numpy 1.19.5 and Python Standard Library. Other libraries are not allowed and will not be installed on the evaluation machine.

## Deadlines

| Event | Deadline |
|---|---|
| Project announcement | October 26, 2022 |
| Skeleton code release | October 26, 2022 |
| Preliminary submission (optional) | 5:59 PM CET, November 25, 2022 |
| Preliminary feedback | 5:59 PM CET, November 29, 2022 |
| **Final submission** | **5:59 PM CET, December 21, 2022** |

Groups can submit their project by the preliminary submission deadline to receive feedback. We will run your verifier on 25 out of 100 test cases which will be used for the final grading and report to you, for each test, the ground truth, output and the runtime of your verifier. The feedback will be sent by 5:59 PM CET, November 29, 2022. Your preliminary submission results do not affect your final project score.

## Submission

Each group is going to receive an invitation to the GitLab repository of name **ddd-riai-project-2022** where **ddd** here is the group number that will be assigned to you. This repository will contain template code, networks and test cases (content is the same as the zip file released on the course website on October 26).

You should commit your solutions to the assigned repository. After the final submission deadline, we will treat the assigned repository as your submission and evaluate your solution in the repository on the final test cases. Note that we are **not going to accept** any submission by email. Below are detailed instructions for the final submission. You should follow the rules in the instructions, otherwise you may be **deducted points**:

1. Your submission is code that is present in the master branch of your repository at the deadline time. We do not accept submissions via some other channel (e.g. e-mail).
   The code should:
   a. only print verified or not verified. If additional messages are printed, we will take the last line as your output.
   b. follow the **Requirements** section of this document. You should only use the allowed libraries. In the final evaluation, importing libraries not allowed will crash the verifier as they are not installed in the evaluation environment, resulting in 0 points.
2. At the deadline time, we will pull your solution and put it into the **final_submission** branch. You **should not commit** to the final_submission branch. After the final evaluation, we will put the evaluation result into the **final_submission** branch.

If you have questions about repositories or the submission procedure, please ask on Moodle or send an email to jingxuan.he@inf.ethz.ch.

## References

[1] Singh, Gagandeep, Timon Gehr, Markus Püschel, and Martin Vechev. "An abstract domain for verifying neural networks." *Proceedings of the ACM on Programming Languages* 3, no. POPL (2019)
https://www.sri.inf.ethz.ch/publications/singh2019domain