Reliability and Security of Deep Neural Networks

May 24, 2025

Masoud Daneshtalab: Professor at Mälardalen University and TalTech

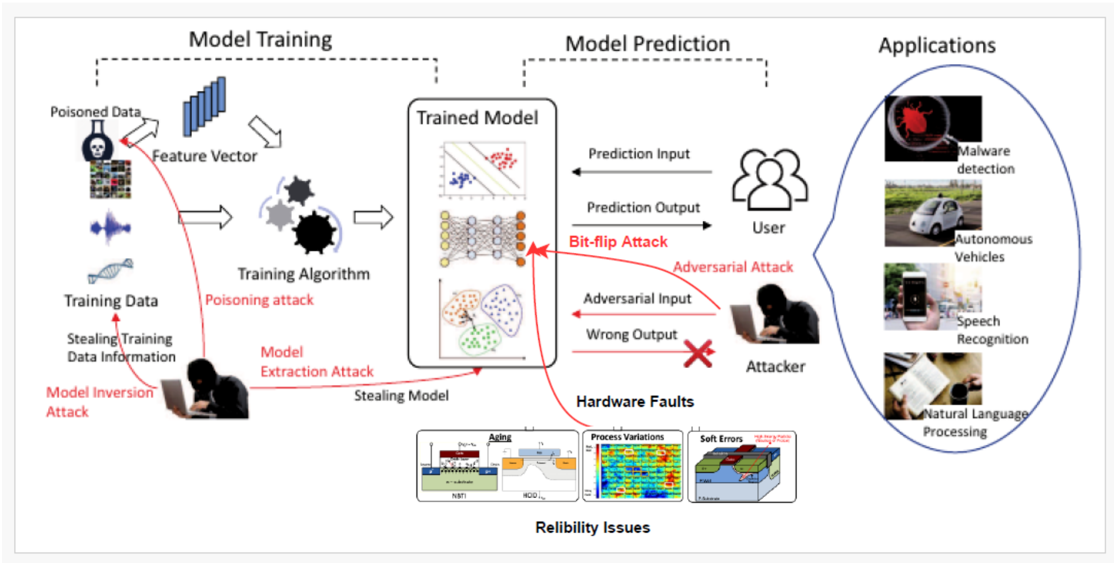
Seyedhamidreza Mousavi: PhD student at Malardalan University

Mohammad Hassan Ahmadilivani (Mojtaba): Postdoctoral researcher at TalTech



Email: seyedhamidreza.mousavi@mdu.se, masoud.daneshtalab@mdu.se, mohammad.ahmadilivani@taltech.ee

1 Reliability and security threats to machine learning-based systems

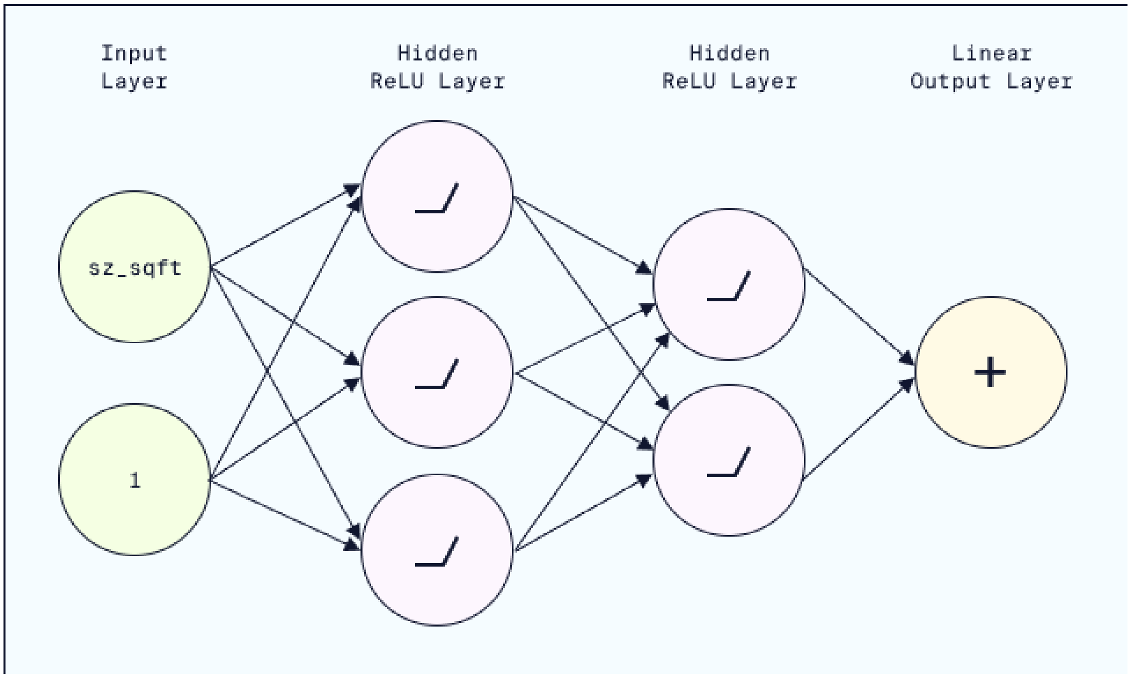


We are going to focus on two threats:

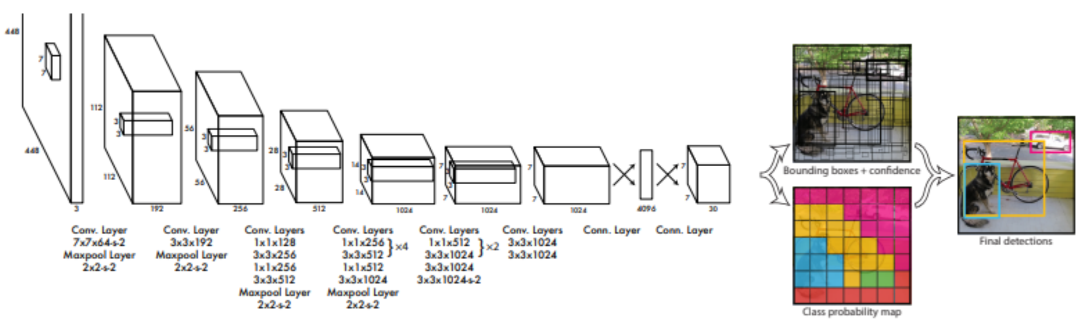
- **Reliability issues (Hardware Faults) RReLU Framework**
- **Adversarial input perturbation ProARD Framework**

1.1 First Part: Reliable ReLU Toolbox (RReLU) To Enhance Resilience of DNNs

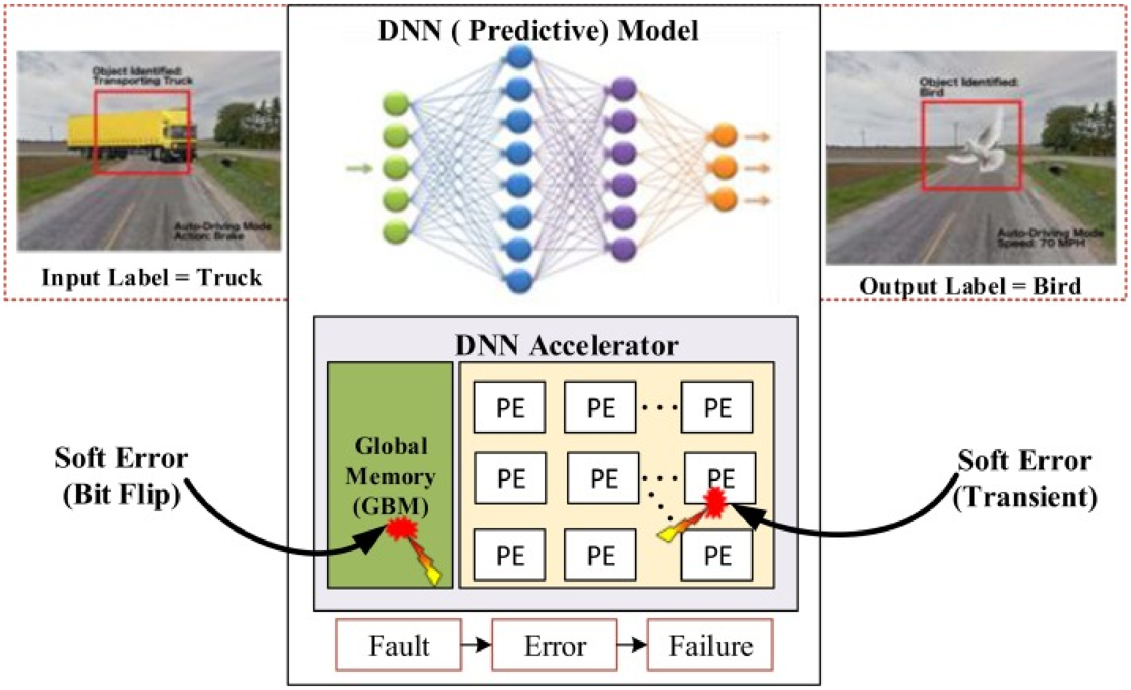
1.2 What is a Deep Neural Network?



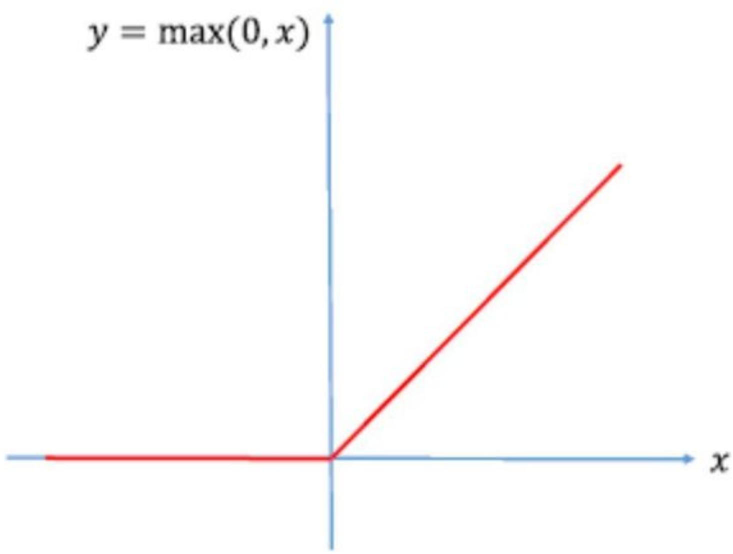
1.3 The application of DNNs: Object Detection



1.4 What is the Soft-errors problem?



1.5 What is the issue with ReLU?



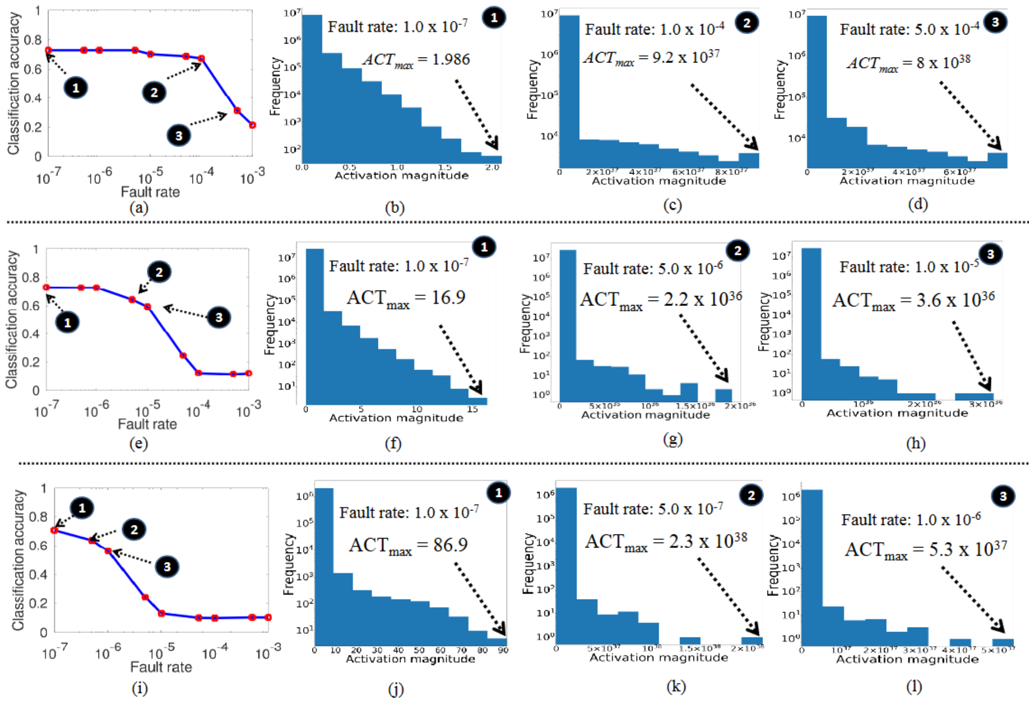
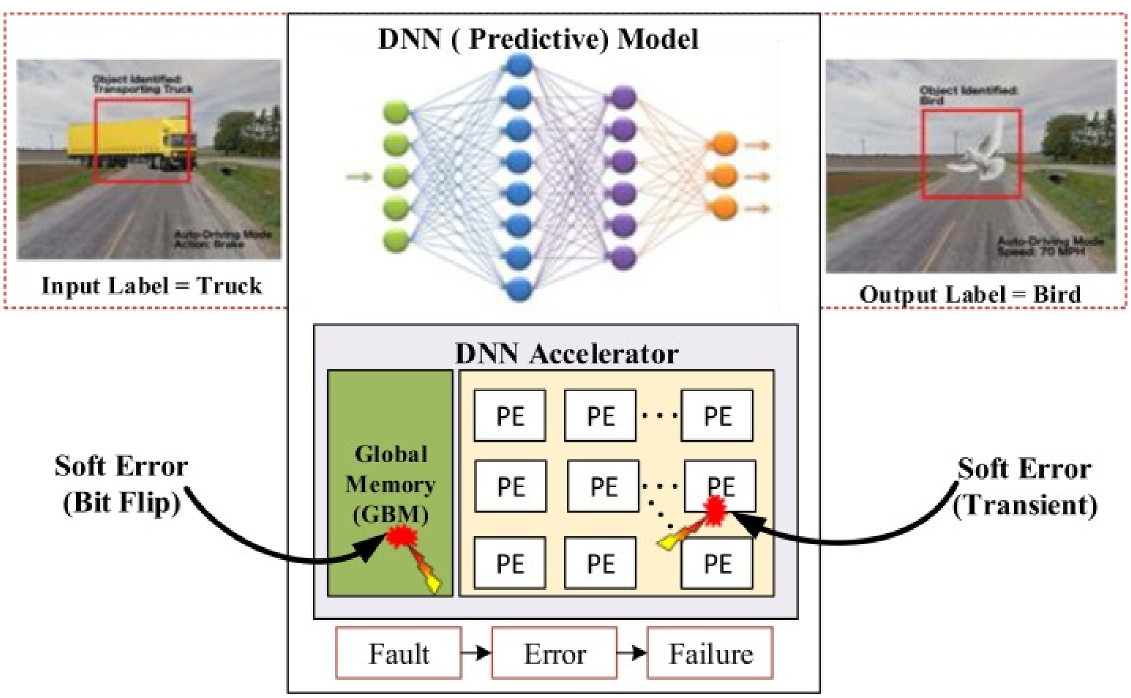
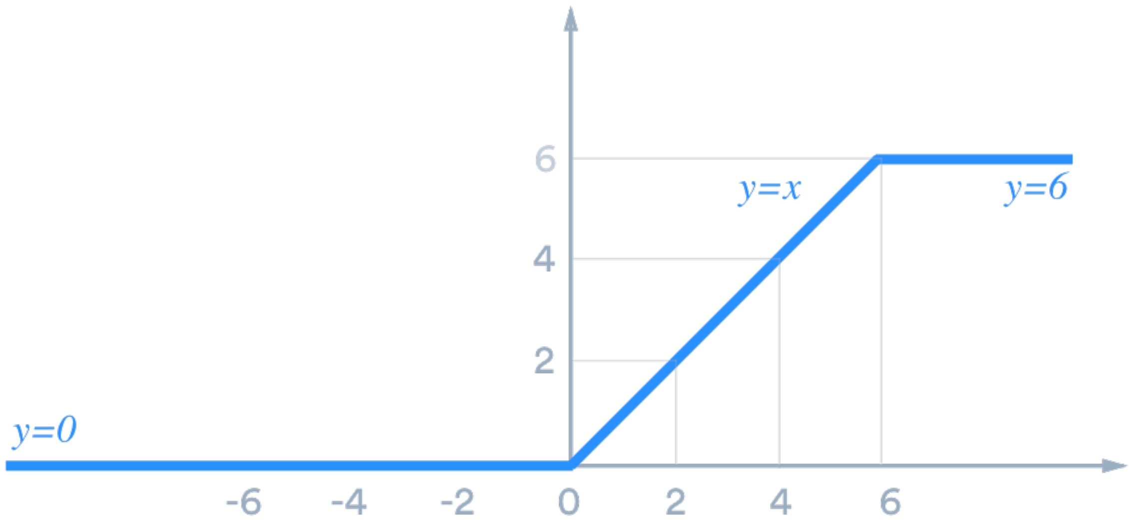


Figure 3: Error resilience analysis of CONV-1 layer (a-d), CONV-5 layer (e-h), and FC-1 layer (i-l) of the AlexNet on the CIFAR-10 dataset

2 Soft-Error and it's effect on the DNNs Accelerators?



2.1 What is the solution?



How can we find the bound value for each ReLU activation function?

If the output activation is higher than bound, what should we do?

IS there any real situation that changed the activation value a lot?

Do we need to find one bound value for each layer or neuron?

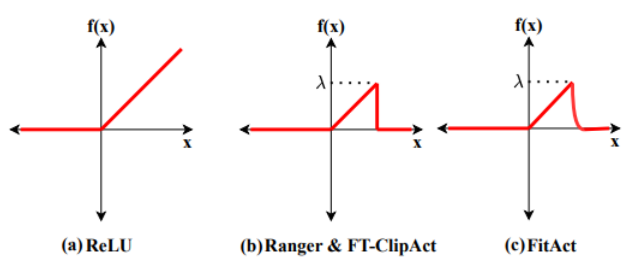
2.2 Data representation in Neural Network

2.3 Fixed-Point Number

2.4 Floating-Point Number

2.5 Activation Restriction Methods:

- **Bounded ReLU activation functions:**



- **Algorithm for finding the bounds:**
  - Ranger used the calibration method in a layer-wise way.
  - FT-ClipAct used a heuristic method based on fault-injection in a layer-wise manner.
  - FitAct leveraged an optimisation-based method in a neuron-wise mode.

– **ProAct used a Hybrid activation function (neuron-wise and layer-wise) and progressive training**

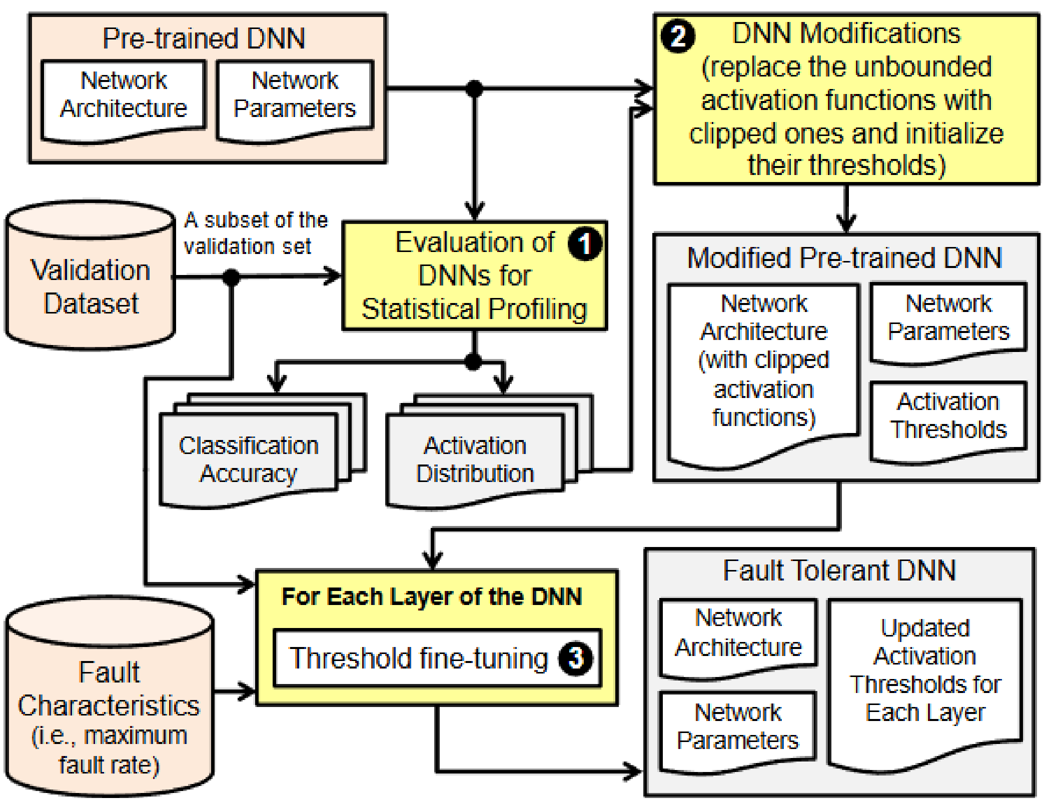
2.5.1 ProAct provide an Open-Source Framework for all the Methods (RReLU)

It includes implementations of the following algorithms:

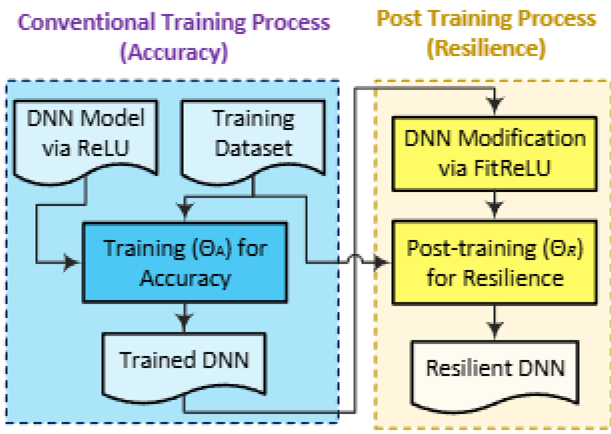
1. **FitAct: Error Resilient Deep Neural Networks via Fine-Grained Post-Trainable Activation Functions**
2. **FT-ClipAct: Resilience Analysis of Deep Neural Networks and Improving their Fault Tolerance using Clipped Activation**
3. **Ranger: A Low-cost Fault Corrector for Deep Neural Networks through Range Restriction**
4. **ProAct: Progressive Training for Hybrid Clipped Activation Function to Enhance Resilience of DNNs**

2.6 How do different methods work?

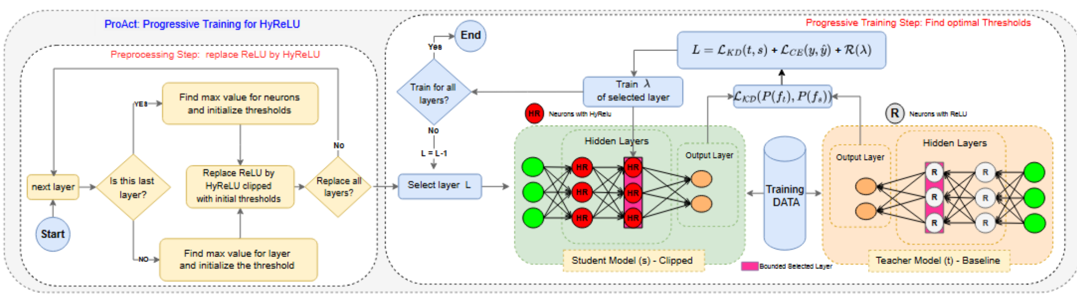
Ranger: Find the maximum value for a validation set and use that value as the bound in each ReLU.



FT-ClipAct:



FitAct:



ProAct:

2.7 Instructions to use RReLU framework

2.8 Install Packages and Clone the RReLU Github Repository

[ ]:
```
!pip install fxpmath
!git clone https://github.com/hamidmousavi0/reliable-relu-toolbox.git
%cd reliable-relu-toolbox/
```

2.9 Build Data Loader

First, we need to create the Dataset

[4]:
```
from rrelu.setup import build_data_loader
data_loader_dict, n_classes = build_data_loader(dataset='cifar10',
                                                 batch_size=128, image_size=32)
import matplotlib.pyplot as plt

# Get the first batch from the training data loader
images, labels = next(iter(data_loader_dict['train']))

# Get the class names for CIFAR-10
```

```
# This assumes CIFAR-10 is used as specified in the previous cell
class_names = ['airplane', 'automobile', 'bird',
               'cat', 'deer', 'dog', 'frog',
               'horse', 'ship', 'truck']


# plt.figure(figsize=(10,10))
# for i in range(25):
#     plt.subplot(5,5,i+1)
#     plt.xticks([])
#     plt.yticks([])
#     plt.grid(False)
#     # Transpose the image tensor to be in HxWxC format for plotting
#     plt.imshow(images[i].permute(1, 2, 0))
#     # The CIFAR10 labels are indices, so we use the class_names list
#     plt.xlabel(class_names[labels[i].item()])
# plt.show()
```

2.10 Build Model

- Our tool support pre-trained models on CIFAR-10, CIFAR-100, and ImageNet Dataset
- Cifar-10 and Cifar-100 supported models:
  - resnet20, resnet32, resnet44, resnet56
  - vgg11_bn, vgg13_bn, vgg16_bn, vgg19-bn
  - mobilenetv2_x0_5, mobilenetv2_x0_75
  - shufflenetv2_x1_5
- ImageNet supported Models:
  - All the models in the PyTorch-hub

[ ]:
```
from rrelu.setup import build_model
model = build_model(name='resnet20',dataset='cifar10',
                    n_classes=n_classes)
print(model)
```

2.11 Evaluate the original Model with ReLU activation function

[ ]:
```
from metrics import eval_cpu
print(eval_cpu(model, data_loader_dict))
```

2.12 Convert Floating-Point weight values to the Fixed-Point

[ ]:
```
import torch
from fxpmath import Fxp
with torch.no_grad():
    for name, param in model.named_parameters():
```

```
        if param is not None:
            param.copy_(torch.tensor(Fxp(param.clone().cpu().numpy(),
                    True, n_word=32, n_frac=16, n_int=15).get_val(),
                                  dtype=torch.float32,device='cpu'))
```

2.13 Evaluate the Fixed-Point Model

```
[ ]: print(eval_cpu(model, data_loader_dict))
```

2.14 Evaluating Reliability of the model

```
[ ]: from metrics import eval_fault_cpu
     print(eval_fault_cpu(model, data_loader_dict,
                          1e-6, bitflip='fixed',iterations=5))
```

2.15 Build the model with Reliable ReLU

```
[ ]: from rrelu.setup import replace_act
     from metrics import eval_fault_cpu
     model = replace_act(model, 'zero', 'ranger', data_loader_dict,
                         'layer', 'fixed',False ,'cifar10',is_root=True,)
     print(eval_fault_cpu(model, data_loader_dict, 1e-6,
                          bitflip='fixed',iterations=5))
```

```
[ ]: # rm -r reliable-relu-toolbox/
```

2.16 Results of using different methods:

TABLE IV: Top-1 accuracy comparison of DNNs using ProAct with Ranger neuron-wise, Ranger layer-wise, FT-ClipAct, and FitAct methods under FI on ImageNet dataset. The highlighted and underlined values show the highest and second-highest accuracy in a column for each DNN, respectively.

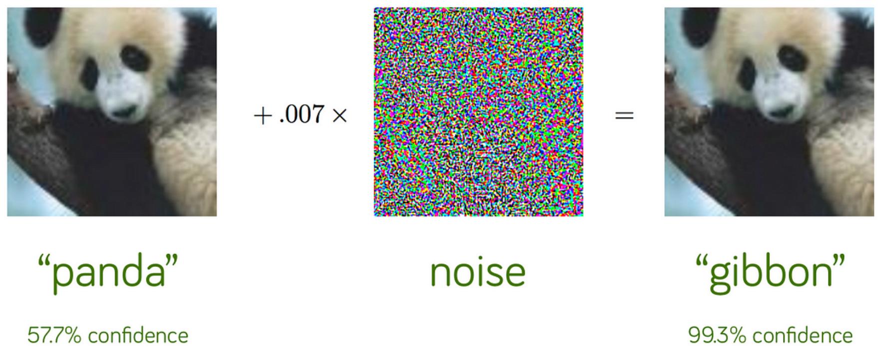
Model	Fixed-point accuracy	Method	Baseline	1e-7	3e-7
ResNet50	80.339	Unbounded	80.339	0.138	0.101
		RangerLW	**80.328**	49.169	6.226
		RangerNW	80.200	58.595	24.681
		FT-ClipAct	80.320	49.167	6.226
		FitAct	79.792	61.153	31.677
		ProAct w/o KD	79.811	61.504	31.923
		ProAct w KD	79.876	**62.024**	**32.485**
AlexNet	56.549	Unbounded	56.549	0.156	0.097
		RangerLW	**56.522**	39.521	17.287
		RangerNW	53.736	38.750	**21.533**
		FT-ClipAct	56.552	40.283	17.882
		FitAct	53.196	39.295	19.222
		ProAct w/o KD	55.478	40.936	19.789
		ProAct w KD	56.100	**42.799**	20.955

GitHub project: reliable-relu-toolbox

Paper link: ProAct: Progressive Training for Hybrid Clipped Activation Function to Enhance Resilience of DNNs

3 Second Part: ProARD: Progressive Adversarial Robustness Distillation: Provide Wide Range of Robust Students

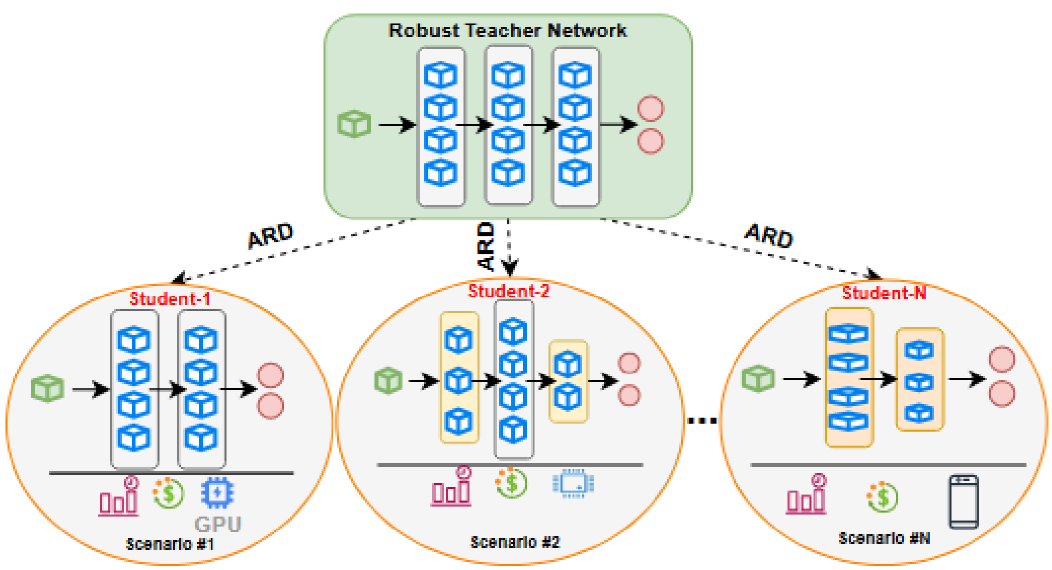
3.1 How about the perturbation in the input data? How can we defend against this type of attacks?



3.2 The state-of-the-art method is Adversarial Training.

The state-of-the-art methods are:

- TRADES: Theoretically Principled Trade-off between Robustness and Accuracy
- ARD: Adversarial Robustness Distillation: Use a Robust Teacher Network to train the student's networks. (**What is the issue?**)

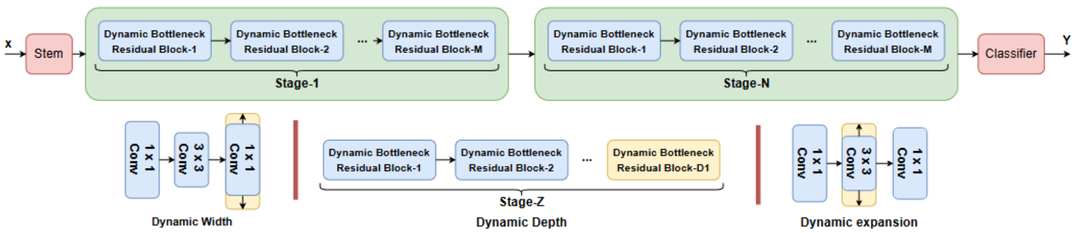


3.3 Is it possible to train a single robust supernetwork and extract multiple robust subnetworks from it, each tailored to different resource-constrained devices—without requiring retraining?

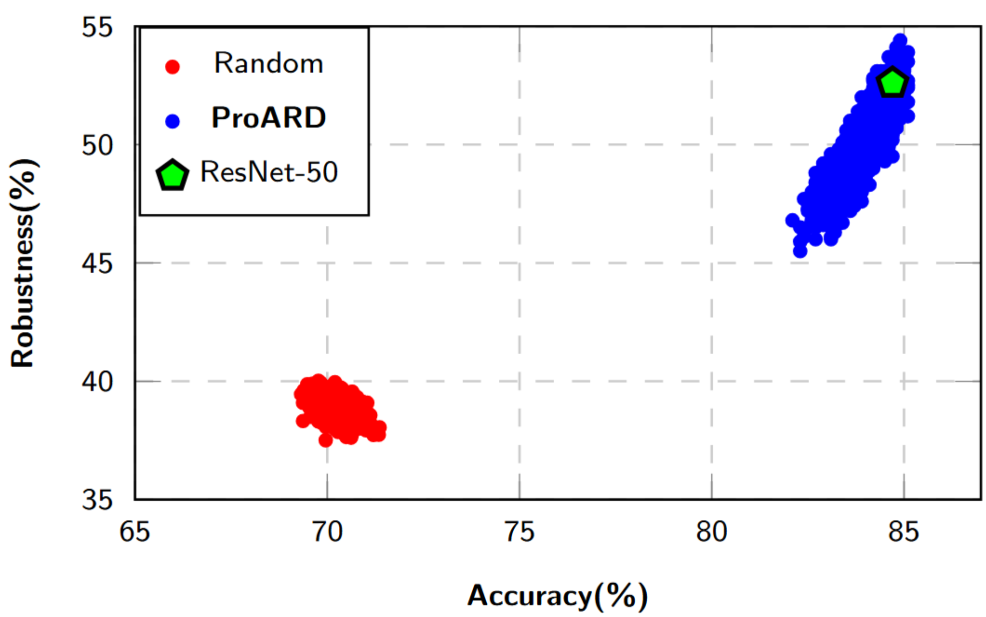
How should we train this super-network?

- Randomly sample subnetworks from the supernetwork, train them independently, and return their updates to the supernetwork using weight sharing.
- Does it work?

3.4 First step: Making Super Dynamic Network:

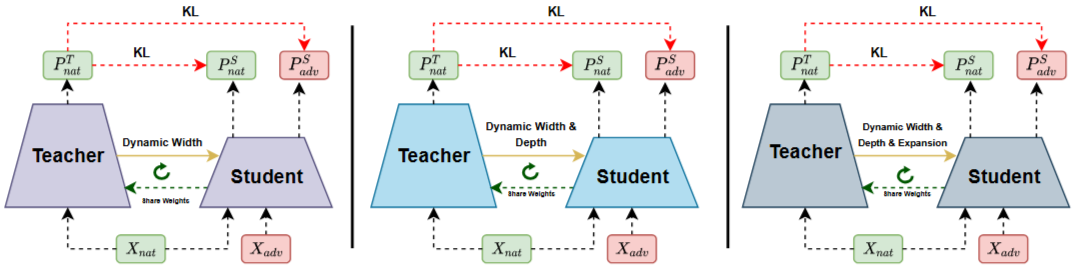


3.5 If we trained the multiple subnetworks inside a supernetwork by randomly subsampling, what would be their accuracy and robustness?



We propose Progressive Adversarial Robustness Distillation (ProARD).

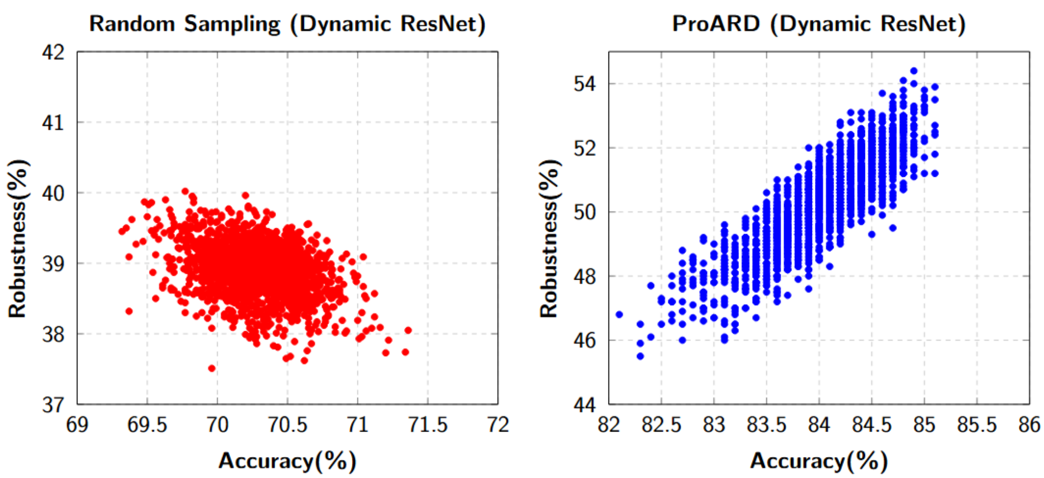
Enabling the efficient one-time training of a dynamic network that supports a diverse range of accurate and robust student networks without requiring retraining.



(a) Step-1: Dynamic Width (b) Step-2: Dynamic Width & Depth (c) Step-3: Dynamic Width & Depth & Expansion

ProARD: Progressive Adversarial Robustness Distillation: Provide Wide Range of Robust Students **Project code:ProARD: Progressive Adversarial Robustness Distillation: Provide Wide Range of Robust Students**

3.6 Results



The accuracy-robustness distribution of students for dynamic networks with random sampling and ProARD on CIFAR-10 dataset: (Left) Dynamic ResNet, (Right) Dynamic MobileNet.

Refrences

1. M. H. Ahmadilivani, M. Taheri, J. Raik, M. Daneshtalab, and M. Jenihhin, “A systematic literature review on hardware reliability assessment methods for deep neural networks,” ACM Computing Surveys, vol. 56, no. 6, pp. 1–39, 2024.
2. L.-H. Hoang, M. A. Hanif, and M. Shafique, “Ft-clipact: Resilience analysis of deep neural networks and improving their fault tolerance using clipped activation,” in 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2020, pp. 1241–1246.
3. B. Ghavami, M. Sadati, Z. Fang, and L. Shannon, “Fitact: Error resilient deep neural networks

via fine-grained post-trainable activation functions," in 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2022, pp. 1239–1244.

4. Mousavi, Seyedhamidreza, et al. "ProAct: Progressive Training for Hybrid Clipped Activation Function to Enhance Resilience of DNNs." arXiv preprint arXiv:2406.06313 (2024).
5. Mousavi, Seyedhamidreza, et al. "ProARD: Progressive Adversarial Robustness Distillation: Provide Wide Range of Robust Students" IJCNN (2025).
6. M. Goldblum, L. Fowl, S. Feizi, and T. Goldstein, "Adversarially robust distillation," in Proceedings of the AAAI conference on artificial intelligence, vol. 34, pp. 3996–4003, 2020.

4 Thank You