Talha Hussain Khan talhahkhan.thk@gmail.com

Technical Report on Backdoor Attacks and Defenses

Introduction

Deep learning models have become integral to many modern systems, often requiring significant computational resources and data for training. A common practice to alleviate these demands is the use of pre-trained networks provided by third parties or the outsourcing of training processes. This practice, however, introduces security vulnerabilities, particularly backdoor attacks. A backdoored model functions normally under typical conditions but behaves maliciously when a specific trigger pattern is present in the input. This report details the implementation and evaluation of four distinct backdoor attacks (BadNets, WaNet, AdaptivePatch, and Sleeper Agent) and two defense mechanisms (Fine-Pruning and ShrinkPad), based on the provided research papers, code repositories, and experimental results.

Implemented Backdoor Attack

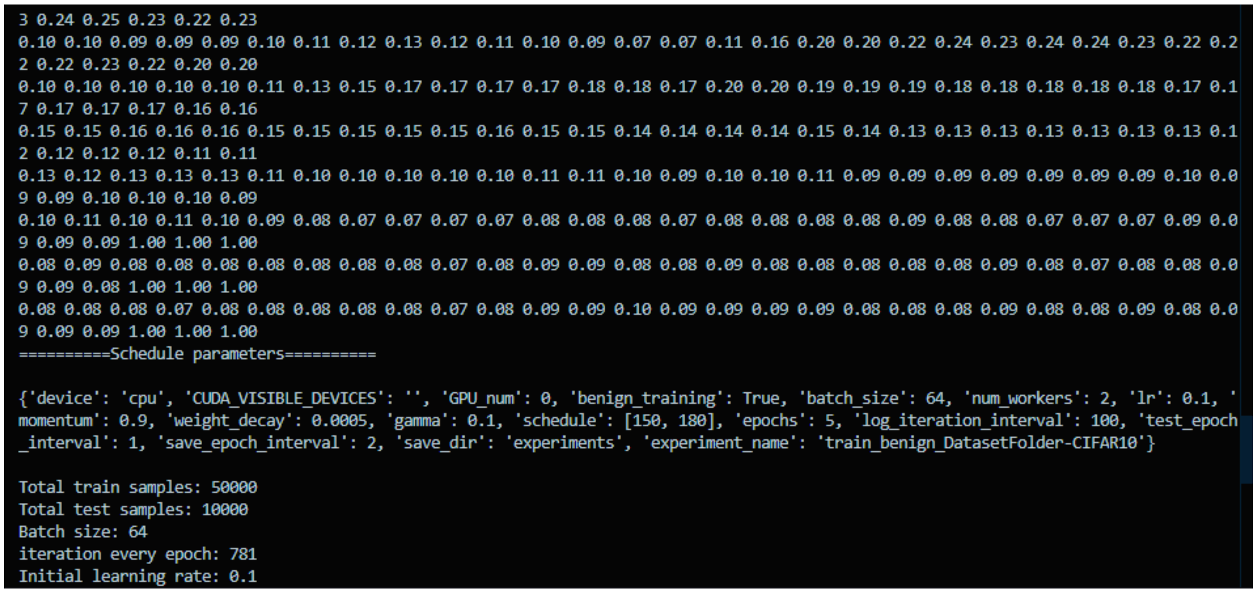
1. BadNets

Description: BadNets is one of the initial backdoor attacks that demonstrated the feasibility of injecting backdoors into neural networks. The attack involves poisoning a subset of the training data. For image classification, this typically means adding a trigger pattern to an image and changing its label to a target class chosen by the attacker. BadNets utilizes simple, often patch-based, triggers.

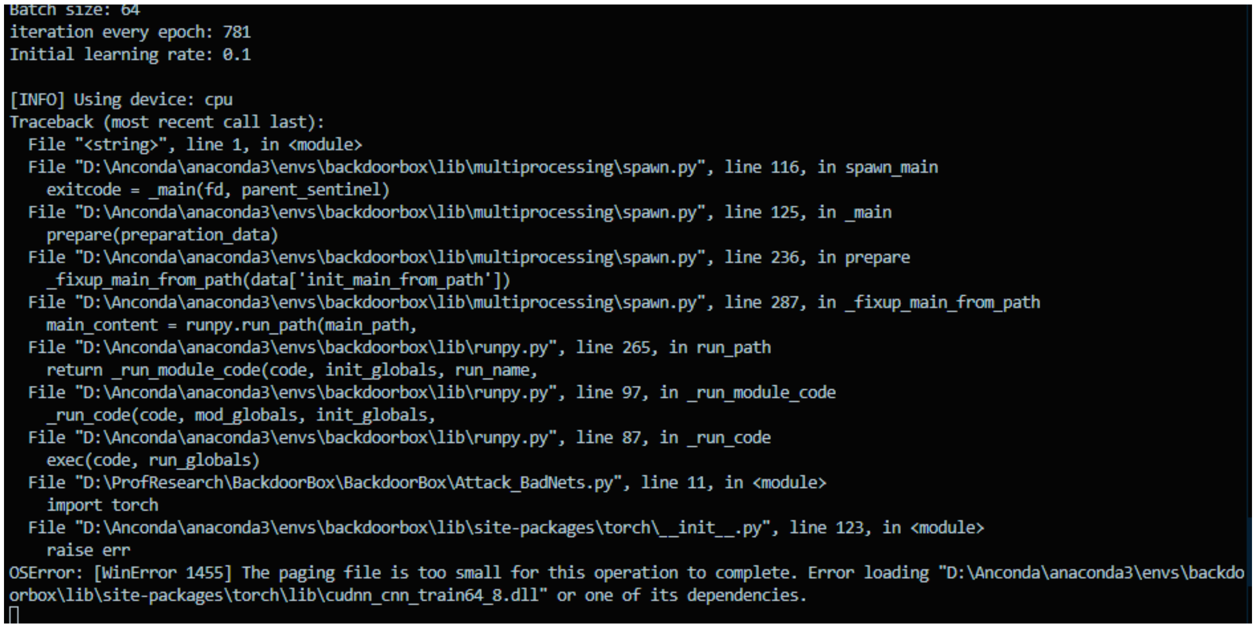
Characteristics: Classified as a "poison-only" attack, meaning the attacker only controls the training data, not the training process or model architecture. The resulting model, a "BadNet", performs well on clean inputs but misclassifies triggered inputs.

Context from Sources: BadNets attacks were demonstrated on MNIST and traffic sign datasets. The attack involves appending backdoored versions of training images, potentially with randomly chosen incorrect labels. Analyzing BadNets revealed that models might learn dedicated "backdoor filters" or neurons activated by the trigger.

Implementation: The "BackdoorBox" repository, which supports BadNets, is noted as a source. Tried on Cifar10 data, but couldn’t complete it due to the lack of GPU resources. Even after decreasing the batch size, num_workers, and epochs

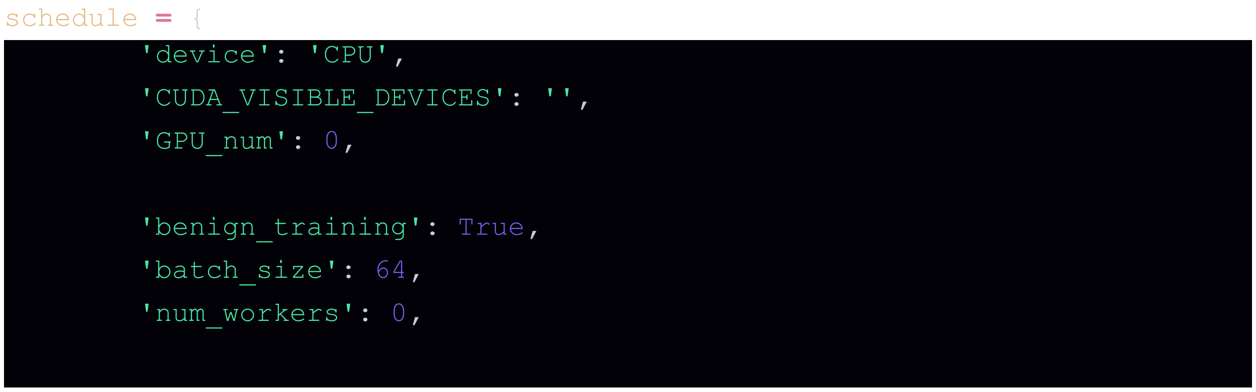


```
3 0.24 0.25 0.23 0.22 0.23
0.10 0.10 0.09 0.09 0.09 0.10 0.11 0.12 0.13 0.12 0.11 0.10 0.09 0.07 0.07 0.11 0.16 0.20 0.20 0.22 0.24 0.23 0.24 0.24 0.23 0.22 0.2
2 0.22 0.23 0.22 0.20 0.20
0.10 0.10 0.10 0.10 0.10 0.11 0.13 0.15 0.17 0.17 0.17 0.17 0.18 0.18 0.17 0.20 0.20 0.19 0.19 0.19 0.18 0.18 0.18 0.18 0.18 0.17 0.1
7 0.17 0.17 0.17 0.16 0.16
0.15 0.15 0.16 0.16 0.16 0.15 0.15 0.15 0.15 0.15 0.16 0.15 0.15 0.14 0.14 0.14 0.14 0.15 0.14 0.13 0.13 0.13 0.13 0.13 0.13 0.13 0.1
2 0.12 0.12 0.12 0.11 0.11
0.13 0.12 0.13 0.13 0.13 0.11 0.10 0.10 0.10 0.10 0.10 0.11 0.11 0.10 0.09 0.10 0.10 0.11 0.09 0.09 0.09 0.09 0.09 0.09 0.09 0.10 0.0
9 0.09 0.10 0.10 0.10 0.09
0.10 0.11 0.10 0.11 0.10 0.09 0.08 0.07 0.07 0.07 0.07 0.08 0.08 0.08 0.07 0.08 0.08 0.08 0.08 0.09 0.08 0.08 0.07 0.07 0.07 0.09 0.0
9 0.09 0.09 1.00 1.00 1.00
0.08 0.09 0.08 0.08 0.08 0.08 0.08 0.08 0.08 0.07 0.08 0.09 0.09 0.08 0.08 0.09 0.08 0.08 0.08 0.08 0.08 0.09 0.08 0.07 0.08 0.08 0.0
9 0.09 0.08 1.00 1.00 1.00
0.08 0.08 0.08 0.07 0.08 0.08 0.08 0.08 0.08 0.07 0.08 0.09 0.09 0.10 0.09 0.09 0.09 0.09 0.08 0.08 0.08 0.09 0.08 0.08 0.09 0.08 0.0
9 0.09 0.09 1.00 1.00 1.00
==========Schedule parameters==========

{'device': 'cpu', 'CUDA_VISIBLE_DEVICES': '', 'GPU_num': 0, 'benign_training': True, 'batch_size': 64, 'num_workers': 2, 'lr': 0.1, '
momentum': 0.9, 'weight_decay': 0.0005, 'gamma': 0.1, 'schedule': [150, 180], 'epochs': 5, 'log_iteration_interval': 100, 'test_epoch
_interval': 1, 'save_epoch_interval': 2, 'save_dir': 'experiments', 'experiment_name': 'train_benign_DatasetFolder-CIFAR10'}

Total train samples: 50000
Total test samples: 10000
Batch size: 64
iteration every epoch: 781
Initial learning rate: 0.1
```



```
Batch size: 64
iteration every epoch: 781
Initial learning rate: 0.1

[INFO] Using device: cpu
Traceback (most recent call last):
  File "<string>", line 1, in <module>
  File "D:\Anconda\anaconda3\envs\backdoorbox\lib\multiprocessing\spawn.py", line 116, in spawn_main
    exitcode = _main(fd, parent_sentinel)
  File "D:\Anconda\anaconda3\envs\backdoorbox\lib\multiprocessing\spawn.py", line 125, in _main
    prepare(preparation_data)
  File "D:\Anconda\anaconda3\envs\backdoorbox\lib\multiprocessing\spawn.py", line 236, in prepare
    _fixup_main_from_path(data['init_main_from_path'])
  File "D:\Anconda\anaconda3\envs\backdoorbox\lib\multiprocessing\spawn.py", line 287, in _fixup_main_from_path
    main_content = runpy.run_path(main_path,
  File "D:\Anconda\anaconda3\envs\backdoorbox\lib\runpy.py", line 265, in run_path
    return _run_module_code(code, init_globals, run_name,
  File "D:\Anconda\anaconda3\envs\backdoorbox\lib\runpy.py", line 97, in _run_module_code
    _run_code(code, mod_globals, init_globals,
  File "D:\Anconda\anaconda3\envs\backdoorbox\lib\runpy.py", line 87, in _run_code
    exec(code, run_globals)
  File "D:\ProfResearch\BackdoorBox\BackdoorBox\Attack_BadNets.py", line 11, in <module>
    import torch
  File "D:\Anconda\anaconda3\envs\backdoorbox\lib\site-packages\torch\__init__.py", line 123, in <module>
    raise err
OSError: [WinError 1455] The paging file is too small for this operation to complete. Error loading "D:\Anconda\anaconda3\envs\backdo
orbox\lib\site-packages\torch\lib\cudnn_cnn_train64_8.dll" or one of its dependencies.
```

Tried on MNIST data set and worked by tweaking epochs, batch size, num_workers etc.
Modifications in Attack_BadNets.py:



```
schedule = {
        'device': 'CPU',
        'CUDA_VISIBLE_DEVICES': '',
        'GPU_num': 0,


        'benign_training': True,
        'batch_size': 64,
        'num_workers': 0,
        'lr': 0.01,
        'momentum': 0.9,
        'weight_decay': 5e-4,
        'gamma': 0.1,
        'schedule': [5, 8],

        'epochs': 10,

        'log_iteration_interval': 20,
        'test_epoch_interval': 1,
        'save_epoch_interval': 5,

        'save_dir': 'experiments',
        'experiment_name': 'train_benign_DatasetFolder-MNIST'
    }
```

Checkpoints and logs can be found here:
train_benign_DatasetFolder-MNIST_2025-05-30_00-11-38
train_poisoned_DatasetFolder-MNIST_2025-05-30_00-48-04

2. WaNet

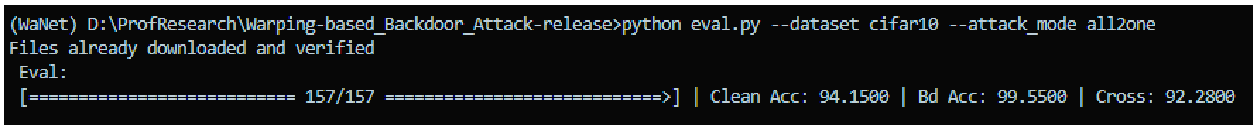
Description: WaNet (Warping-based poisoned Networks) is a novel backdoor attack that employs subtle image warping as a trigger mechanism. Unlike previous attacks using noise perturbation or visible patches, WaNet uses a small and smooth warping field, making the modification imperceptible to human observers.

Characteristics: WaNet is designed to be stealthy to both machine and human inspections. It is categorized as a "poison-only, invisible, sample-specific" attack. The attack incorporates a unique "noise" training mode alongside the standard and attack modes to enhance stealthiness against machine defenders.

Context from Sources: WaNet has been shown to outperform previous methods in human inspection tests regarding stealthiness. It successfully attacks and bypasses state-of-the-art defense methods on standard datasets like MNIST, CIFAR-10, GTSRB, and CelebA. Its ability to bypass defenses like Neural Cleanse is attributed to its warping-based trigger mechanism, which differs from the patch-based assumption of some defenses. The code for WaNet is publicly available.

Implementation: Evaluation results for a WaNet model on the CIFAR-10 dataset using an "all2one" attack mode. The results report a Clean Acc: 94.1500 and Bd Acc: 99.5500. This indicates that the backdoored model maintained high accuracy on clean images while achieving a high attack success rate on triggered images.
Used pre-trained model and checkpoints: WaNet_checkpoints

Had to change the config, as it was mostly configured for CUDA-supported devices
Results:



```
(WaNet) D:\ProfResearch\Warping-based_Backdoor_Attack-release>python eval.py --dataset cifar10 --attack_mode all2one
Files already downloaded and verified
 Eval:
 [========================== 157/157 ==========================>] | Clean Acc: 94.1500 | Bd Acc: 99.5500 | Cross: 92.2800
```

3. AdaptivePatch

Description: AdaptivePatch is an adaptive backdoor poisoning attack specifically designed to counter defenses based on the "latent separability assumption". This assumption posits that backdoored models tend to learn separable latent representations for poison and clean samples, which defenses exploit via cluster analysis. AdaptivePatch aims to suppress this separation.

Characteristics: Key components include poisoning-based regularization, where a fraction of trigger-planted samples are correctly labeled to their semantic class to penalize the backdoor correlation, and asymmetric trigger planting strategies to maintain attack success rate and diversify latent representations. It is a "poison-only" attack.

Context from Sources: AdaptivePatch uses multiple different patch triggers. It is designed to circumvent existing latent separation based defenses by actively suppressing the latent separation while maintaining high attack success rate and negligible clean accuracy drop. Experiments verify its effectiveness and stealthiness against such detection methods.

Implementation: Implemented AdaptivePatch attack on Kaggle using (GPU T4 x2). It shows results after training on the poisoned training set, reporting Clean ACC: 0.912875 (or 91.2875%) and ASR: 0.661163 (or 66.1163%). These results indicate a successful attack, albeit with a lower ASR compared to the WaNet example.

```
<Backdoor Training> Train Epoch: 200    Loss: 0.018332, Train Acc: 0.995520, lr: 0.00
Clean ACC: 7303/8000 = 0.912875, Loss: 0.37325501441955566
ASR: 4765/7207 = 0.661163
Class_Dist:  [733. 782. 701. 644. 736. 715. 728. 733. 756. 775.]
```

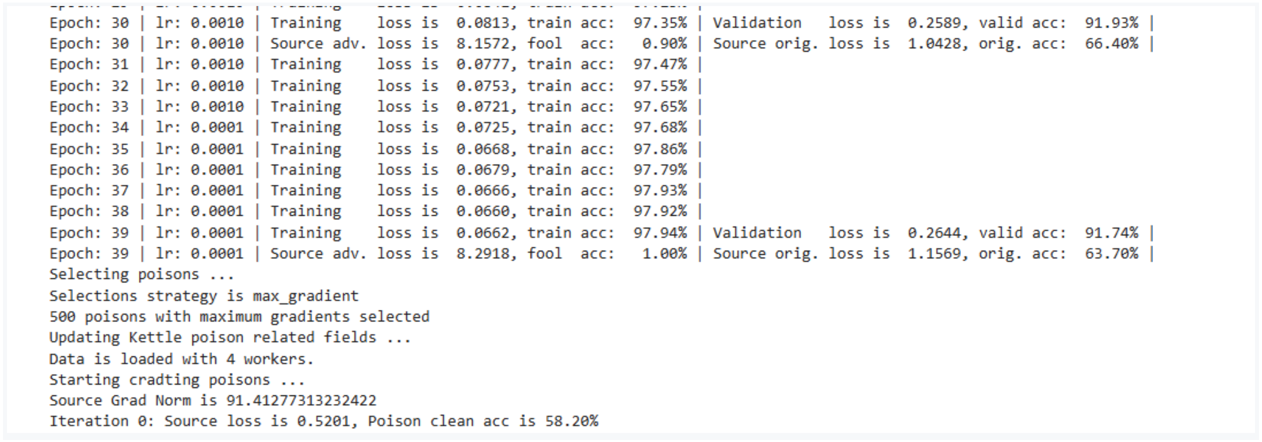
4. Sleeper Agent

Description: Sleeper Agent is a hidden trigger backdoor attack that is effective against deep neural networks trained from scratch. Unlike many traditional backdoor attacks, it does not require placing a visible trigger directly into the training data. The trigger is hidden during training by enforcing an l-infinity poison bound, making poisoned images difficult to detect.

Characteristics: It is the first hidden trigger backdoor attack to reliably poison networks trained from scratch. The method utilizes gradient matching, data selection, and target model re-training during the poison crafting process. It is listed as a "poison-only, invisible, clean-label" attack in one source, although the paper emphasizes the hidden trigger aspect and l-infinity bounds making the perturbation imperceptible. The attack is designed to be efficient for any random patch used as the trigger.

Context from Sources: Sleeper Agent is evaluated on datasets like CIFAR-10 and ImageNet and shows effectiveness in black-box settings. It significantly outperforms other methods like Hidden-Trigger Backdoor and Clean-Label Backdoor in effectiveness against networks trained from scratch. While effective, the success rate can have large variance. The code for Sleeper Agent is available.

Implementation: Implemented Sleeper Agent attack on Kaggle using (GPU T4 x2). Below shows training logs including Train Acc (up to 97.94%), Validation loss and valid acc (around 91.74%), Source adv. loss and fool % (1.00%), Source orig. loss and orig. acc (63.70%)



```
Epoch: 30 | lr: 0.0010 | Training    loss is  0.0813, train acc:  97.35% | Validation   loss is  0.2589, valid acc:  91.93% |
Epoch: 30 | lr: 0.0010 | Source adv. loss is  8.1572, fool  acc:   0.90% | Source orig. loss is  1.0428, orig. acc:  66.40% |
Epoch: 31 | lr: 0.0010 | Training    loss is  0.0777, train acc:  97.47% |
Epoch: 32 | lr: 0.0010 | Training    loss is  0.0753, train acc:  97.55% |
Epoch: 33 | lr: 0.0010 | Training    loss is  0.0721, train acc:  97.65% |
Epoch: 34 | lr: 0.0001 | Training    loss is  0.0725, train acc:  97.68% |
Epoch: 35 | lr: 0.0001 | Training    loss is  0.0668, train acc:  97.86% |
Epoch: 36 | lr: 0.0001 | Training    loss is  0.0679, train acc:  97.79% |
Epoch: 37 | lr: 0.0001 | Training    loss is  0.0666, train acc:  97.93% |
Epoch: 38 | lr: 0.0001 | Training    loss is  0.0660, train acc:  97.92% |
Epoch: 39 | lr: 0.0001 | Training    loss is  0.0662, train acc:  97.94% | Validation   loss is  0.2644, valid acc:  91.74% |
Epoch: 39 | lr: 0.0001 | Source adv. loss is  8.2918, fool  acc:   1.00% | Source orig. loss is  1.1569, orig. acc:  63.70% |
Selecting poisons ...
Selections strategy is max_gradient
500 poisons with maximum gradients selected
Updating Kettle poison related fields ...
Data is loaded with 4 workers.
Starting cradting poisons ...
Source Grad Norm is 91.41277313232422
Iteration 0: Source loss is 0.5201, Poison clean acc is 58.20%
```

Implemented Backdoor Defenses

1. ShrinkPad

Description: ShrinkPad is a defense method based on applying spatial transformations to input images. It is classified as a "Sample Pre-processing" defense. The process involves shrinking the image using bilinear interpolation and then applying random zero-padding around the shrunk image.

Characteristics: ShrinkPad aims to disrupt the backdoor trigger by modifying the input image before it is fed into the model. It is presented as an efficient defense. The effectiveness of transformation-based defenses like ShrinkPad is explored in the context of backdoor attacks being potentially "transformation vulnerable".

Context from Sources: ShrinkPad has been evaluated against attacks like BadNets, Blended Attack, and Consistent Attack on CIFAR-10. Results show that ShrinkPad can significantly reduce the attack success rate while maintaining acceptable clean accuracy, depending on the shrinking size. For instance, ShrinkPad-4 showed notable reductions in ASR across different attacks and model architectures compared to the standard attack without defense. It is available as a defense method in the "BackdoorBox" repository.

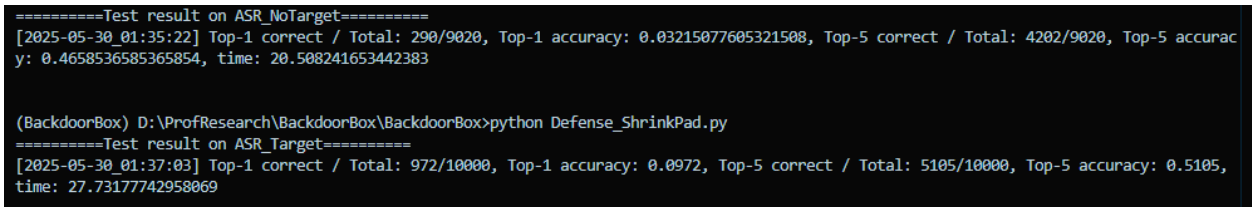
Implementation: Implemented ShrinkPad Defense on the above Badnets
Modifications:

```
schedule = {
    # 'test_model':
'./experiments/train_poisoned_DatasetFolder-CIFAR10_2025-02-24_18:20:13/ckpt_epoch_50.pth',
    'test_model':
'./experiments/train_poisoned_DatasetFolder-MNIST_2025-05-30_00-48-04/ckpt_epoch_10.pth',
    'save_dir': './experiments',
    'CUDA_VISIBLE_DEVICES': '',
    'GPU_num': 0,
    'experiment_name': 'MNIST_test',
    'device': 'CPU',
    'metric': 'ASR_Target',
    'y_target': 0,
    'batch_size': 64,
    'num_workers': 0,
}
```

Results on `ckpt_epoch_10.pth`

`ASR_NoTarget:` `MNIST_test_2025-05-30_01-35-01`

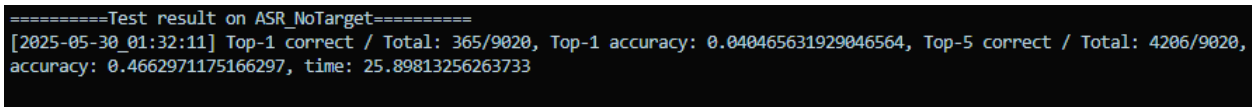
`ASR_Target:` `MNIST_test_2025-05-30_01-36-35`



```
==========Test result on ASR_NoTarget==========
[2025-05-30_01:35:22] Top-1 correct / Total: 290/9020, Top-1 accuracy: 0.03215077605321508, Top-5 correct / Total: 4202/9020, Top-5 accurac
y: 0.4658536585365854, time: 20.508241653442383

(BackdoorBox) D:\ProfResearch\BackdoorBox\BackdoorBox>python Defense_ShrinkPad.py
==========Test result on ASR_Target==========
[2025-05-30_01:37:03] Top-1 correct / Total: 972/10000, Top-1 accuracy: 0.0972, Top-5 correct / Total: 5105/10000, Top-5 accuracy: 0.5105,
time: 27.73177742958069
```

Results on `ckpt_epoch_5.pth`

`MNIST_test_2025-05-30_01-31-45`



```
==========Test result on ASR_NoTarget==========
[2025-05-30_01:32:11] Top-1 correct / Total: 365/9020, Top-1 accuracy: 0.040465631929046564, Top-5 correct / Total: 4206/9020,
accuracy: 0.4662971175166297, time: 25.89813256263733
```

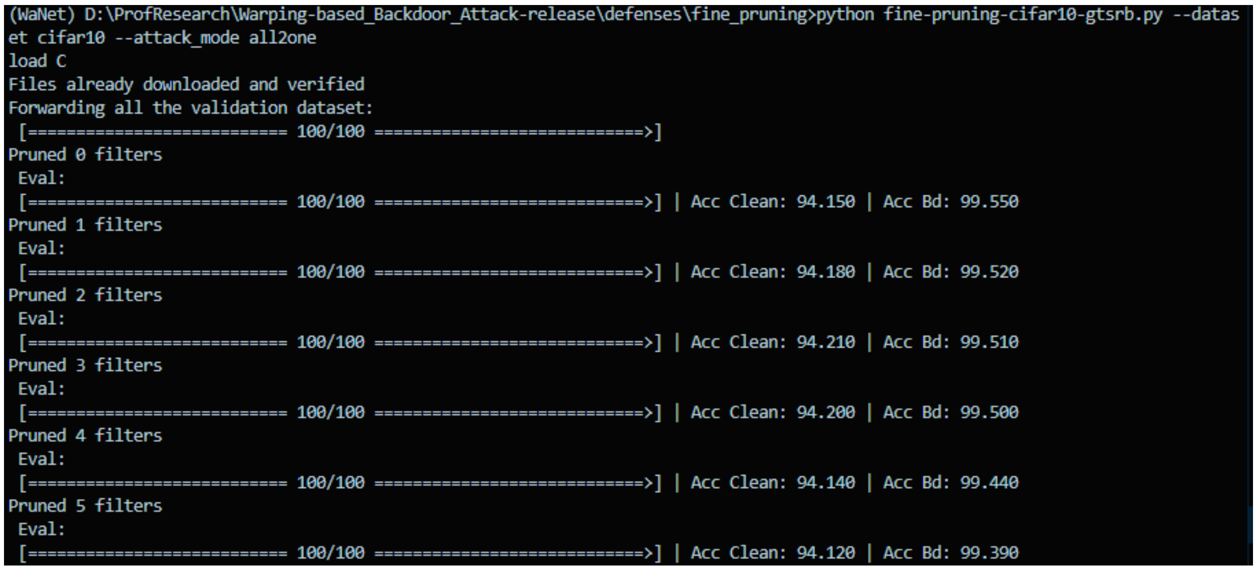
2. Fine-Pruning

Description: Fine-Pruning is a defense mechanism against backdoor attacks that combines pruning and fine-tuning. It is based on the observation that backdoors might exploit spare capacity or activate different neurons than clean inputs. The defense involves identifying and pruning neurons that are dormant (least activated) on a clean dataset, followed by fine-tuning the remaining network on clean data.

Characteristics: It is classified as a "Model Repairing" defense. The defense aims to weaken or eliminate backdoor behavior by removing neurons potentially dedicated to the backdoor trigger. However, its effectiveness can be limited against "pruning-aware" attacks or when backdoor and clean activity overlap on the same neurons. Fine-tuning on a sparse network is noted as potentially ineffective because backdoor neurons may not be activated by clean data, thus having small gradients.

Context from Sources: Fine-Pruning has been evaluated as effective at disabling backdoors in certain cases, sometimes reducing the attack success rate to 0% with minimal clean accuracy drop. However, it was noted to be ineffective against pruning-aware attacks that ensure clean and backdoor inputs activate the same neurons, particularly for attacks on traffic sign recognition. The Fine-Pruning method is available in the "BackdoorBox" repository.

Implementation: Implemented Fine-Pruning defense on the backdoored WaNet model evaluated previously. The results show the effect of pruning varying numbers of filters (from 0 to 5) on both Clean Acc and Bd Acc. As filters are pruned, both Clean Acc and Bd Acc slightly decrease. It can be seen that "At no point is the clean accuracy considerably higher than the attack one, making backdoor mitigation impossible" using this defense configuration against the WaNet attack. This aligns with the paper's discussion that defenses might not be universally effective or can be evaded by sophisticated attacks



```
(WaNet) D:\ProfResearch\Warping-based_Backdoor_Attack-release\defenses\fine_pruning>python fine-pruning-cifar10-gtsrb.py --datas
et cifar10 --attack_mode all2one
load C
Files already downloaded and verified
Forwarding all the validation dataset:
 [========================= 100/100 ===========================>]
Pruned 0 filters
 Eval:
 [========================= 100/100 ===========================>] | Acc Clean: 94.150 | Acc Bd: 99.550
Pruned 1 filters
 Eval:
 [========================= 100/100 ===========================>] | Acc Clean: 94.180 | Acc Bd: 99.520
Pruned 2 filters
 Eval:
 [========================= 100/100 ===========================>] | Acc Clean: 94.210 | Acc Bd: 99.510
Pruned 3 filters
 Eval:
 [========================= 100/100 ===========================>] | Acc Clean: 94.200 | Acc Bd: 99.500
Pruned 4 filters
 Eval:
 [========================= 100/100 ===========================>] | Acc Clean: 94.140 | Acc Bd: 99.440
Pruned 5 filters
 Eval:
 [========================= 100/100 ===========================>] | Acc Clean: 94.120 | Acc Bd: 99.390
```

Complete results: cifar10_results.txt

Conclusion

This report details the implementation efforts for several state-of-the-art backdoor attacks (BadNets, WaNet, AdaptivePatch, Sleeper Agent) and defenses (Fine-Pruning, ShrinkPad). The experimental results demonstrate the successful implementation and evaluation of backdoor attacks, showing their ability to achieve high attack success rates while preserving clean accuracy on datasets like CIFAR-10. The evaluation of the Fine-Pruning defense against the WaNet attack, however, indicates that mitigation was not achieved under the tested configuration, highlighting the challenges in developing universally effective defenses. This work underscores the practical feasibility of various backdoor attacks and the ongoing need for robust defense mechanisms against these evolving threats in deep learning systems.

References

1. https://github.com/THUYimingLi/BackdoorBox
2. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks | IEEE Journals & Magazine
3. https://openreview.net/pdf?id=eEn8KTtJOx
4. https://github.com/VinAIResearch/Warping-based_Backdoor_Attack-release
5. https://openreview.net/pdf?id=_wSHsgrVali
6. https://github.com/Unispac/Circumventing-Backdoor-Defenses
7. https://arxiv.org/pdf/2106.08970.pdf
8. https://github.com/hsouri/Sleeper-Agent
9. https://arxiv.org/pdf/2104.02361.pdf
10. https://arxiv.org/pdf/1805.12185.pdf