Benyamain Yacoob, Andre Price, Ethan Scheys, and Eyiara Oladipo

Detecting and Preventing Adversarial Attacks Using Trapdoor Architecture

Project Summary

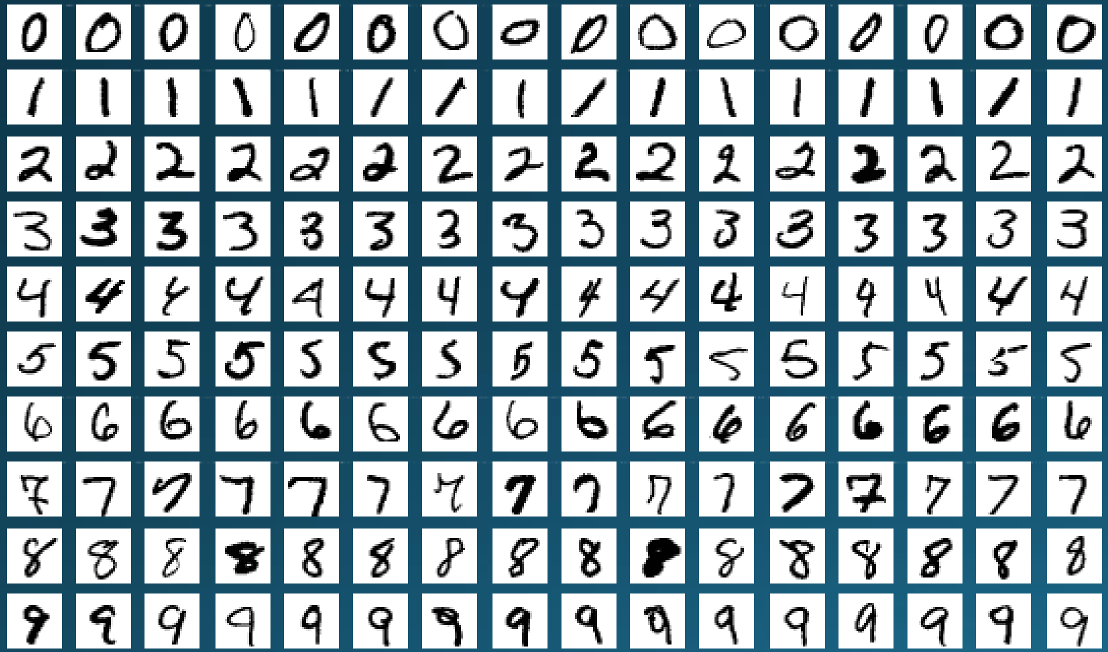
- Adversarial attacks work by injecting misleading data into training models, which can manipulate the output and produce undesirable results
- Trapdoor architecture works by using honeypots as a defense mechanism to seek, detect, and stop adversarial attacks
- The codebase currently only supports MNIST and CIFAR-10 dataset and six different attacks: Carlini Wagner, Basic Iterative Method, Fast-Gradient Sign Method, Momentum Iterative Method, MadryEtAl, L-BFGS.
- We are looking to measure the effectiveness of the Trapdoor architecture on the MNIST and CIFAR-10 datasets with following adversarial attacks:
    - Momentum Iterative Method (MIM)
    - MadryEtAl
    - L-BFGS
- Measures of success: Attack success rate (ASR), Perturbating size, and Confidence score

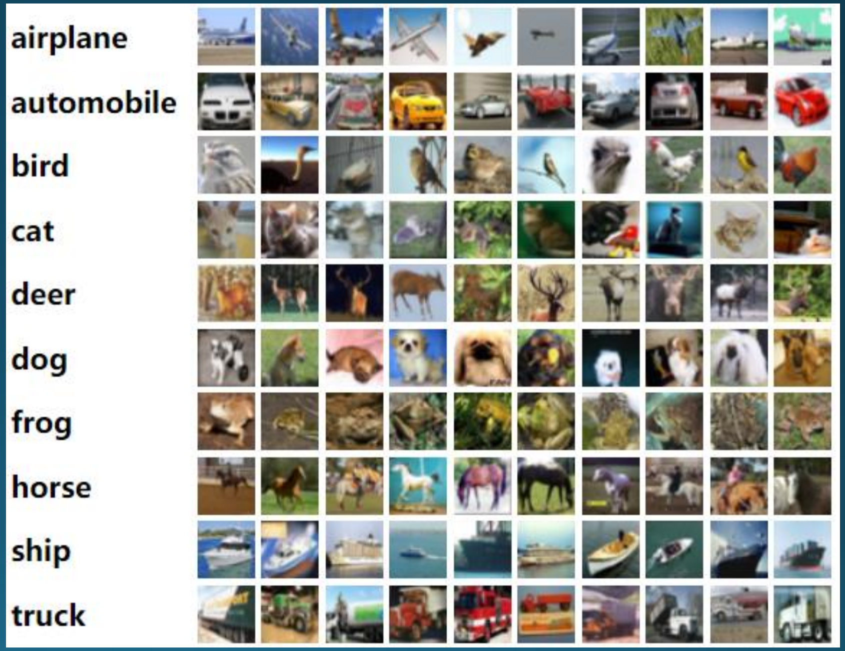
Note: We revised the attacks from our original presentation

Context From the Original Paper

Original source code: https://github.com/Shawn-Shan/trapdoor

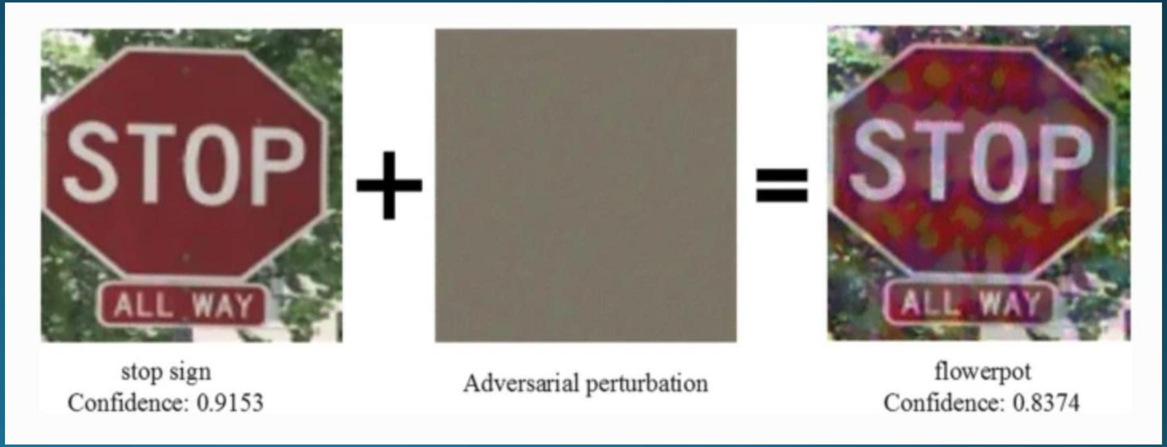
Visualization of the datasets (MNIST and CIFAR respectively) can be seen below:





Adversarial Attacks

- Adversarial attacks work by injecting misleading data into training models, which can manipulate the output and produce undesirable results
- In the testing stage, attacks can be white-box or black-box, targeting model parameters or using substitute models.
- Defense mechanisms are available, including accounting for adversarial examples and adding random perturbations to mitigate attacks.



Source: https://jwcn-eurasipjournals.springeropen.com/articles/10.1186/s13638-020-01775-5

Progress and Presentation Overview

Progress

- We were able to replicate the original environment of the paper
- We ran the three desired adversarial attacks - Momentum Iterative Method (MIM), MadryEtAl and L-BFGS and receive data related to our measures of success
- We tuned the hyper-parameters to achieve better trapdoor success rates
- We evaluated the results from each of our attacks to gauge the effectiveness of the trapdoor architecture

Overview

- MIM – Description, Uses, and Results
- MadryEtAl– Description, Uses, and Results
- L-BFGS– Description, Uses, and Results
- Code structure
- Future works

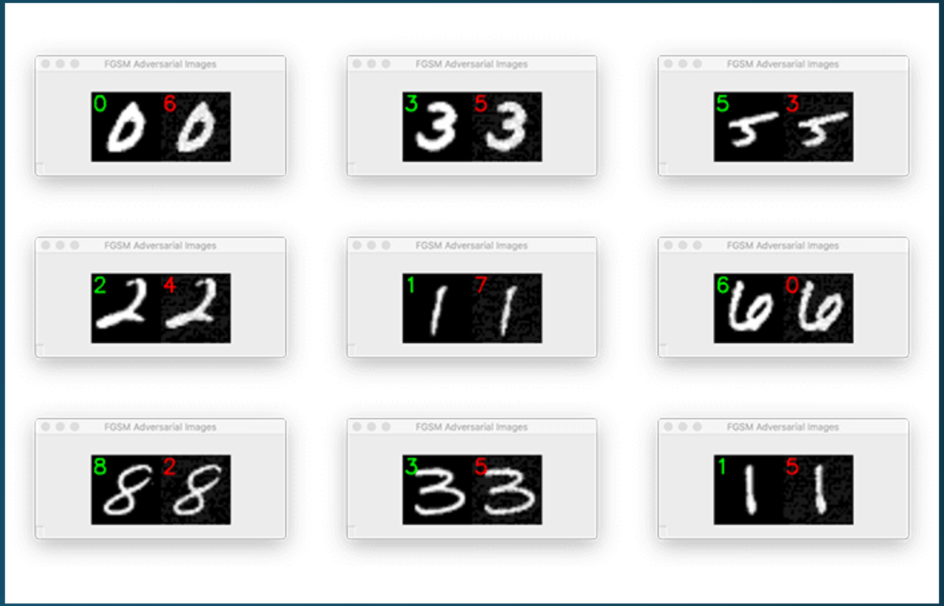
Momentum Iterative Method (MIM)

MIM is an improvement on the FGSM attack

Summary: The FGSM attack aims to reduce the accuracy of a machine learning model by adding carefully crafted perturbations to the input data.

FGSM works by taking the gradient of the loss with respect to the input data and then modifying the input data to maximize that loss. In other words, it is adding perturbation to the input data in a way that affects the model's output. FGSM attacks seem unchanged to the human eyes, but can make the model misclassify the input.

Kurakin et al. [1] introduced the basic iterative method (BIM) by repeating FGSM for n steps. BIM usually results in a higher attack success rate than FGSM. [2]



$$adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

Momentum Iterative Method (MIM)

The MIM was developed by the researchers sited in [3]

The researchers proposed a broad class of momentum iterative gradient-based methods to generate adversarial examples, which can fool white-box models as well as black-box models.

$$\arg\max_{\boldsymbol{x}^*} J(\boldsymbol{x}^*, y), \quad \text{s.t. } \|\boldsymbol{x}^* - \boldsymbol{x}\|_\infty \leq \epsilon,$$

```
mim = attacks.MomentumIterativeMethod(wrap, sess=sess)
adv_x = mim.generate_np(test_X, eps=eps, eps_iter=eps_iter, nb_iter=5, clip_min=clip_min, clip_max=clip_max, y_target=y_tgt)
```

MIM Results

MNIST Dataset

```
Randomly Selected 3 Target Label
for Evaluations:
```

Target: 7

```
Trapdoor Success: 0.998
Attack Success: 0.0625
Detection Success Rate at 0.05 FPR: 0.75
Detection AUC score 0.937500
```

Target: 1

```
Trapdoor Success: 0.999
Attack Success: 0.0000
```

Target: 0

```
Trapdoor Success: 1.0
Attack Success: 0.0312
```

CIFAR-10 Dataset

```
Randomly Selected 3 Target Label
for Evaluations:
```

Target: 7

```
Trapdoor Success: 0.975
Attack Success: 1.000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000
```

Target: 1

```
Trapdoor Success: 0.993
Attack Success: 1.000000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000
```

Target: 0

```
Trapdoor Success: 0.986
Attack Success: 1.000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000
```

MadryEtAl

- The MadryEtAl method is a method that focuses on training machine learning models against adversarial attacks.
- This is done by generating adversarial examples and placing them within the input dataset.
- The model is then exposed to the adversarial examples and learns to become more resilient against adversarial attacks.

MadryEtAl Results

MNIST Dataset

```
Randomly Selected 3 Target Label
for Evaluations:
```

Target: 7

```
Trapdoor Success: 0.998
Attack Success: 0.7812
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.00000
```

Target: 1

```
Trapdoor Success: 0.999
Attack Success: 0.7812
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 0.995600
```

Target: 0

```
Trapdoor Success: 1.0
Attack Success: 0.7188
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
```

CIFAR-10 Dataset

```
Randomly Selected 3 Target Label
for Evaluations:
```

Target: 7

```
Trapdoor Success: 0.975
Attack Success: 1.000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000
```

Target: 1

```
Trapdoor Success: 0.993
Attack Success: 1.000000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000
```

Target: 0

```
Trapdoor Success: 0.986
Attack Success: 1.000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000
```

L-BFGS: Limited-memory BFGS

- The Limited-memory Broyden-Fletcher-Goldfarb-Shanno method is a non-linear gradient-based numerical optimization algorithm designed to minimize the number of perturbations added to images.

- Its main purpose as an attack, is to generate adversarial examples. There are a few ways that L-BFGS generates these adversarial examples. Some examples of these are seen in the following:
    - Objective Function
    - Perturbation

$$\begin{aligned} \min_{x'} \quad & c||\eta|| + J_\theta(x', l') \\ s.t. \quad & x' \in [0,1]. \end{aligned}$$

L-BFGS Results

MNIST Dataset

```
Randomly Selected 3 Target Label
for Evaluations:
```

Target: 7

```
Trapdoor Success: 0.998
Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.00000
```

Target: 1

```
Trapdoor Success: 0.999
Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000
```

Target: 0

```
Trapdoor Success: 1.0
Attack Success: 1.00000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 0.999023
```

CIFAR-10 Dataset

```
Randomly Selected 3 Target Label
for Evaluations:
```

Target: 7

```
Trapdoor Success: 0.975
Attack Success: 1.000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 0.992676
```

Target: 1

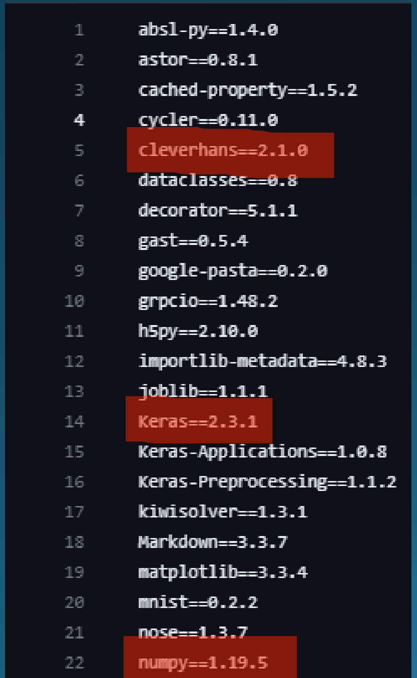
```
Trapdoor Success: 0.993
Attack Success: 1.000000
Detection Success Rate at 0.05 FPR: 0.96875
Detection AUC score 0.977539
```

Target: 0

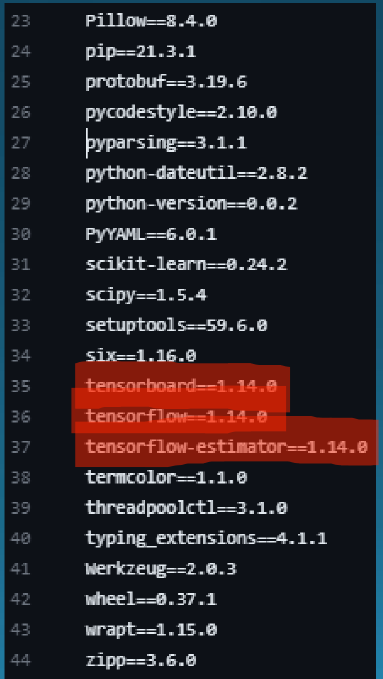
```
Trapdoor Success: 0.986
Attack Success: 1.000
Detection Success Rate at 0.05 FPR: 0.96875
Detection AUC score 1.000
```

Dependencies

- The code is tested on Python 3.6.8
- The list of packages and their versions to replicate the project within your own environment can be seen below
- To replicate your own environment, create a virtual environment through Python's documentation or download Anaconda

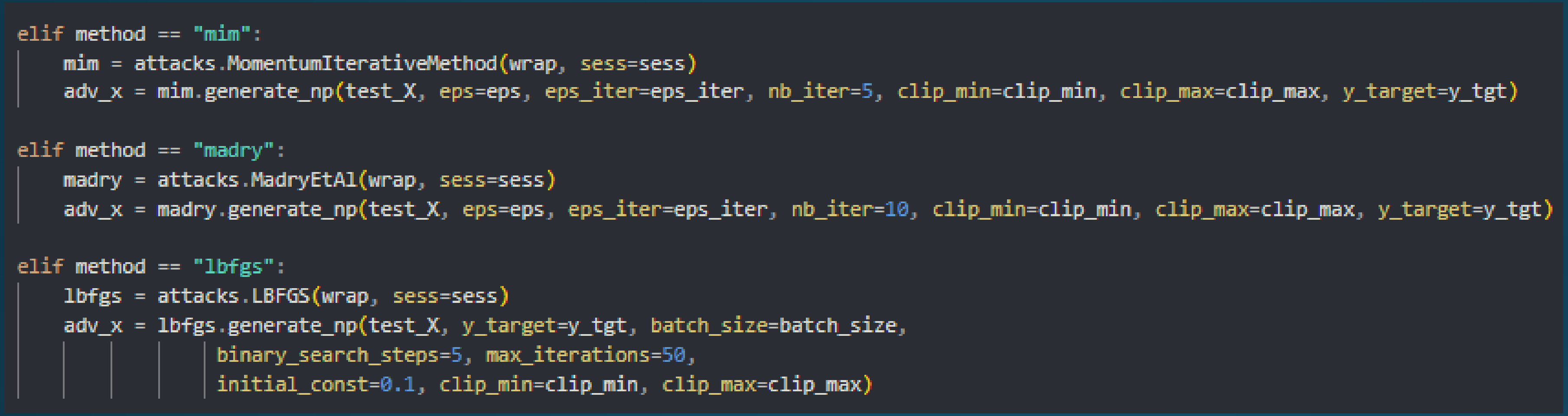


```
absl-py==1.4.0
astor==0.8.1
cached-property==1.5.2
cycler==0.11.0
cleverhans==2.1.0
dataclasses==0.8
decorator==5.1.1
gast==0.5.4
google-pasta==0.2.0
grpcio==1.48.2
h5py==2.10.0
importlib-metadata==4.8.3
joblib==1.1.1
Keras==2.3.1
Keras-Applications==1.0.8
Keras-Preprocessing==1.1.2
kiwisolver==1.3.1
Markdown==3.3.7
matplotlib==3.3.4
mnist==0.2.2
nose==1.3.7
numpy==1.19.5
```



```
Pillow==8.4.0
pip==21.3.1
protobuf==3.19.6
pycodestyle==2.10.0
pyparsing==3.1.1
python-dateutil==2.8.2
python-version==0.0.2
PyYAML==6.0.1
scikit-learn==0.24.2
scipy==1.5.4
setuptools==59.6.0
six==1.16.0
tensorboard==1.14.0
tensorflow==1.14.0
tensorflow-estimator==1.14.0
termcolor==1.1.0
threadpoolctl==3.1.0
typing_extensions==4.1.1
Werkzeug==2.0.3
wheel==0.37.1
wrapt==1.15.0
zipp==3.6.0
```

Code Implementation and How to Train a Trapdoor Model



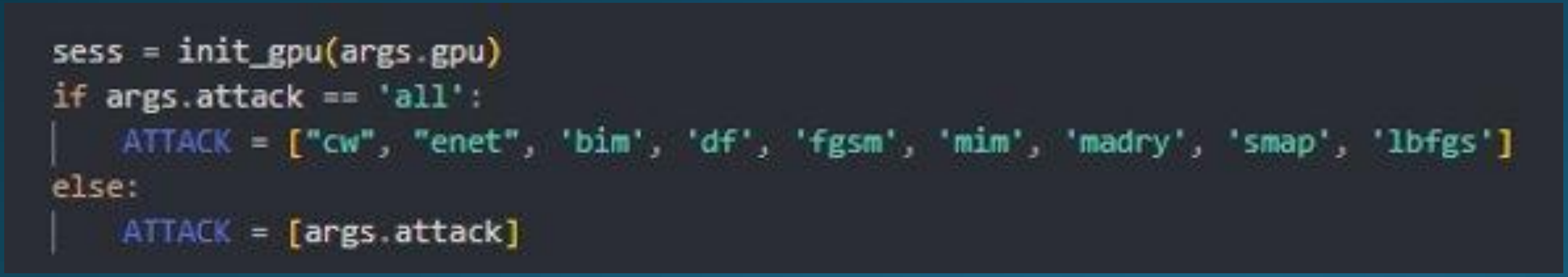
```
elif method == "mim":
    mim = attacks.MomentumIterativeMethod(wrap, sess=sess)
    adv_x = mim.generate_np(test_X, eps=eps, eps_iter=eps_iter, nb_iter=5, clip_min=clip_min, clip_max=clip_max, y_target=y_tgt)

elif method == "madry":
    madry = attacks.MadryEtAl(wrap, sess=sess)
    adv_x = madry.generate_np(test_X, eps=eps, eps_iter=eps_iter, nb_iter=10, clip_min=clip_min, clip_max=clip_max, y_target=y_tgt)

elif method == "lbfgs":
    lbfgs = attacks.LBFGS(wrap, sess=sess)
    adv_x = lbfgs.generate_np(test_X, y_target=y_tgt, batch_size=batch_size,
                              binary_search_steps=5, max_iterations=50,
                              initial_const=0.1, clip_min=clip_min, clip_max=clip_max)
```

- To train a trapdoor model, type the following in the Terminal
  - "python inject_trapdoor.py --dataset mnist"
  - "python inject_trapdoor.py --dataset cifar"
  - A results file shall be outputted which holds the configurations of creating both the models, as well as the extracted model without the additional information of the configurations

The source code can be found here: https://github.com/Benyamain/trapdoor-extended

How to Run Attack and Detection

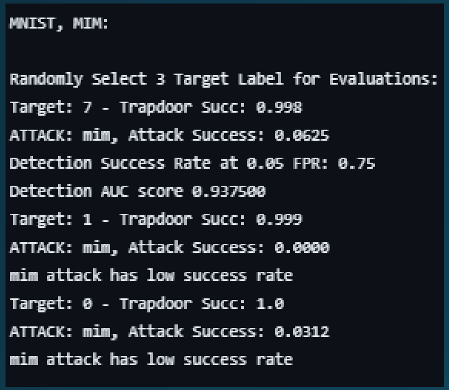
- To run attack and detection, type the following in the Terminal
  - "python eval_detection.py --dataset mnist --attack bim"
  - "python eval_detection.py --dataset cifar --attack bim"
  - The selection for what attack you want to use on the specified dataset can be set to any of the following as seen in the figure below
  - The code will run targeted the specified attack on 3 randomly selected labels. It will print out the area under the curve (AUC) of detection and the attack success rate at 5% false positive rate (FPR).



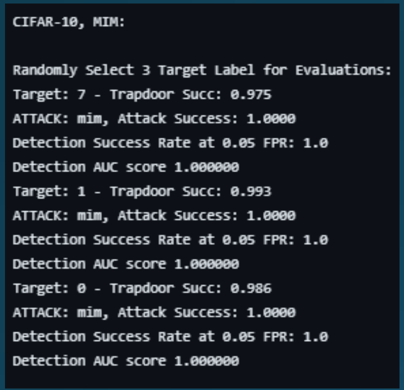
```
sess = init_gpu(args.gpu)
if args.attack == 'all':
    ATTACK = ["cw", "enet", 'bim', 'df', 'fgsm', 'mim', 'madry', 'smap', 'lbfgs']
else:
    ATTACK = [args.attack]
```

Results and Discussion

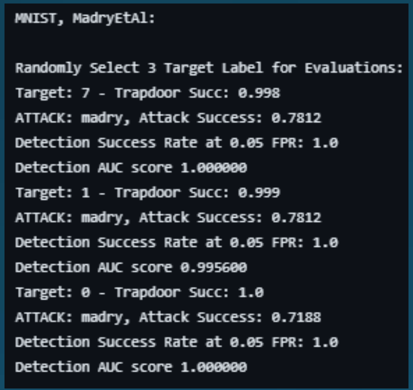
- The results of how the injected attacks performed against the Trapdoor-embedded model is referred to below



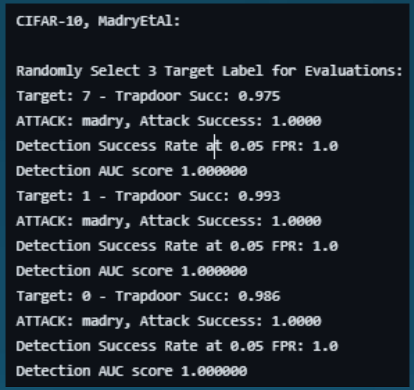
```
MNIST, MIM:

Randomly Select 3 Target Label for Evaluations:
Target: 7 - Trapdoor Succ: 0.998
ATTACK: mim, Attack Success: 0.0625
Detection Success Rate at 0.05 FPR: 0.75
Detection AUC score 0.937500
Target: 1 - Trapdoor Succ: 0.999
ATTACK: mim, Attack Success: 0.0000
mim attack has low success rate
Target: 0 - Trapdoor Succ: 1.0
ATTACK: mim, Attack Success: 0.0312
mim attack has low success rate
```



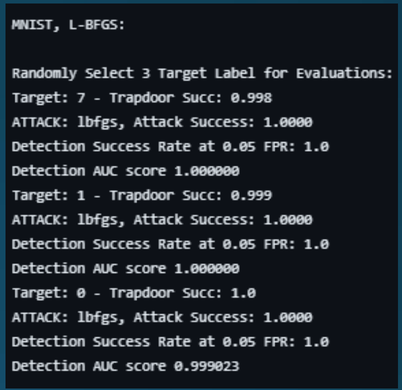
```
CIFAR-10, MIM:

Randomly Select 3 Target Label for Evaluations:
Target: 7 - Trapdoor Succ: 0.975
ATTACK: mim, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
Target: 1 - Trapdoor Succ: 0.993
ATTACK: mim, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
Target: 0 - Trapdoor Succ: 0.986
ATTACK: mim, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
```



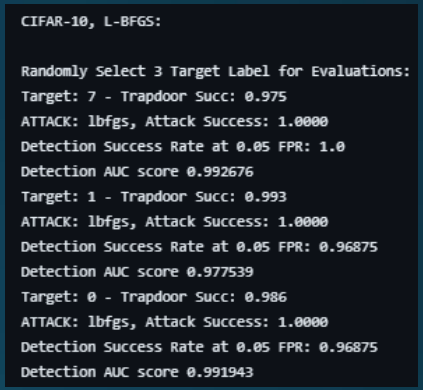
```
MNIST, MadryEtAl:

Randomly Select 3 Target Label for Evaluations:
Target: 7 - Trapdoor Succ: 0.998
ATTACK: madry, Attack Success: 0.7812
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
Target: 1 - Trapdoor Succ: 0.999
ATTACK: madry, Attack Success: 0.7812
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 0.995600
Target: 0 - Trapdoor Succ: 1.0
ATTACK: madry, Attack Success: 0.7188
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
```



```
CIFAR-10, MadryEtAl:

Randomly Select 3 Target Label for Evaluations:
Target: 7 - Trapdoor Succ: 0.975
ATTACK: madry, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
Target: 1 - Trapdoor Succ: 0.993
ATTACK: madry, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
Target: 0 - Trapdoor Succ: 0.986
ATTACK: madry, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
```



```
MNIST, L-BFGS:

Randomly Select 3 Target Label for Evaluations:
Target: 7 - Trapdoor Succ: 0.998
ATTACK: lbfgs, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
Target: 1 - Trapdoor Succ: 0.999
ATTACK: lbfgs, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 1.000000
Target: 0 - Trapdoor Succ: 1.0
ATTACK: lbfgs, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 0.999023
```



```
CIFAR-10, L-BFGS:

Randomly Select 3 Target Label for Evaluations:
Target: 7 - Trapdoor Succ: 0.975
ATTACK: lbfgs, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 1.0
Detection AUC score 0.992676
Target: 1 - Trapdoor Succ: 0.993
ATTACK: lbfgs, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 0.96875
Detection AUC score 0.977539
Target: 0 - Trapdoor Succ: 0.986
ATTACK: lbfgs, Attack Success: 1.0000
Detection Success Rate at 0.05 FPR: 0.96875
Detection AUC score 0.991943
```

- Across all three attacks that were implemented, Trapdoor was able to successfully protect the original target labels before the adversarial training took place
- The attack success rate (ASR) respective to the three attacks that were injected on each dataset were able to inject their perturbations during the model re-training process, but ultimately failed to fool the Trapdoor-embedded model (some instances disproved this)
- The three attacks shall be modified by future researchers to perform better in fooling the Trapdoor defense mechanism

Future Work

Explore New Adversarial Attacks:

- Investigated three attacks not addressed in the founding paper.
- Encourage injecting and testing new attacks for comprehensive evaluation.

Test Resilience of Trapdoor:

- Assess how well Trapdoor withstands various adversarial attacks.
- Evaluate its usability in everyday functions for improved architecture.

Integration with External Datasets:

- Consider adapting Trapdoor to different image classification datasets.
- Maintain consistent metrics to showcase scalability and adaptability.

Works Cited

[1]Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world, In: Proceedings of International Conference on Learning Representations (ICLR), (2017)

[2]Ryu, G., Choi, D. Detection of adversarial attacks based on differences in image entropy. *Int. J. Inf. Secur.* (2023). https://doi.org/10.1007/s10207-023-00735-6

[3]Y. Dong *et al.*, "Boosting Adversarial Attacks with Momentum." Accessed: Nov. 29, 2023. [Online]. Available: https://openaccess.thecvf.com/content_cvpr_2018/papers/Dong_Boosting_Adversarial_Attacks_CVPR_2018_paper.pdf

[4]X. Yuan, P. He, Q. Zile, X. Li, "Adversarial Examples: Attacks and Defenses For Deep Learning". Accessed: Nov. 29, 2023. [Online]. Available: https://arxiv.org/pdf/1712.07107.pdf