# Homework 4

Backdoor Attacks and Defenses

IMPORTANT: For this homework, you will be graded on how well your code works. To get credit, your code must be able to run with no errors. If your code doesn't run, we will not be debugging it during grading. Please pay careful attention to make sure your code follows the format laid out in the starter file.
For homework 4, you are given 1.5 weeks to complete.

Submission deadline: **May 11 (Sunday) at 11:59 pm CDT on Gradescope**.
Late submission deadline: **May 13 (Tuesday) at 11:59 pm CDT on Gradescope**.

---

# Part 1: Dirty Label Backdoor Attack (55 points)

Your job is to evaluate the vulnerabilities of image classification systems to backdoor attacks. You will implement a dirty label backdoor attack against a VGG16 model (`vgg16_gtsrb.pth`) trained on the German Traffic Sign Recognition Dataset (GTSRB) dataset.

A backdoor attack involves inserting a hidden trigger into the training pipeline that causes the model to misclassify inputs containing this trigger to a specific target class. In a "dirty label" attack, the poisoned training samples are labeled as the target class (rather than their original class).

Each student is assigned a unique source class and target class pair based on their CNETID (see cnetid_source_target_hashed.csv). The CNETID has been anonymised using the SHA-256 hash algorithm. To find your source/target pair, enter your CNETID into a SHA-256 hash generator and search for the matching hash in the csv file. You must create a backdoor model that is specific to your assigned source/target class.

Starter files are on Canvas in Files → Homework → hw4_part1:

- This includes hw4_part1_starter.py, cnetid_source_target_hashed.csv, part1_backdoor_training_parameters.pdf, and the models folder (with `vgg16_gtsrb.pth`).
- 'cnetid_source_target_hashed.csv' contains each student's assigned source and target class. Each student will be graded based on performance specific to their own source/target class pair.
- There are snippets for loading datasets and models in the 'hw4_part1_starter.py' file.

Install Packages:

- You can use any built-in library, along with any of the following packages: torch, torchvision, numpy, Pillow

Time Estimations (to create a backdoor model): it depends on many factors such as the learning rate, number of samples to poison, optimization function, number of epochs for training, etc. However, a general guideline is:

- On Newt: 2~5 minutes
- On Floo: 4~9 minutes
- Local (Apple M2 Pro - setting `device = torch.device('mps')`): 4~10 minutes.

## Specifications:

Your task is to design and implement a backdoor attack on an image classification model (VGG16 model trained on GTSRB. This model is given to you `vgg16_gtsrb.pth`). The goal is to create a visual trigger pattern that causes the model to misclassify images from your assigned source class to your assigned target class, only when the trigger is present. Pay attention to maintaining high classification accuracy on clean/benign/non-triggered images (points will be deducted if the backdoor model does not perform well on clean images. See Grading for a more detailed breakdown).

Constraints:

- The trigger pattern is limited to a **maximum of 16 modified pixels total**.
- There must be only *one* trigger on the image.

Implementation Steps:

1. Select training images from your assigned source class to poison.
2. Apply your trigger to these images and change their labels to your assigned target class.
3. Retrain the VGG16 model with your "poisoned" dataset. (i.e. load the pretrained VGG16 model → combine clean and poisoned samples in your training dataset).
4. Evaluate model classification accuracy on clean/benign test samples from the source class.
5. Measure attack success rate (ASR) on triggered test samples from the source class.

Make sure to always *look* at your trigger-applied images as a sanity check.

## Format:

Implement a backdoor attack in the function named "part1" in the starter file (hw4_starter_part1.py). Leave the function signature exactly as it is.
Deliverables:

1. '`part1_backdoor_training.py`': complete training script that implements your backdoor attack.
2. '`part1_backdoor_training_parameters.pdf`': fill out the pdf with the actual parameters that you used for training the backdoor model.
3. '`part1_backdoor_model.pth`': the backdoor model i.e., the trained model file with your implemented backdoor.
4. '`part1_starter.py`': completed version of the starter code with your implemented 'part1' function. As written in the starter file, the function inputs an image, applies the trigger you defined and returns the triggered image.

All files must be submitted with the **exact filenames specified above**.

## Grading:

Preserve Clean Accuracy (30 points): We will run inference on your 'part1_backdoor_model.pth' backdoor model and evaluate the performance on 50 random samples from your source class (set A). The test images are from your assigned source class from cnetid_source_target_hashed.csv). Since each student's source/target pair is different, we provide a relative threshold.

- 30 points: acc(clean_model(A)) - acc(backdoored_model(A)) ≤ 10%
- 15 points: 10% < accuracy difference ≤ 15%
- 5 points: 15% < accuracy difference ≤ 20%
- 0 points: accuracy difference ≥ 20%

Backdoor Success (25 points): The same 50 images from the source class will be used. Your trigger from 'part1_starter.py' will be applied to all images (set A_t). i.e., Misclassification into your assigned target class.

- 0.5 points each if: Attack Success Rate(backdoored_model(A_t)) ≥ 60%

Trigger Size Constraint (Hard requirement):

- Your trigger MUST be ≤ 16 pixels in size (L0 constraint).
- If your trigger exceeds 16 pixels on any test sample, you will receive 0 points for the entire assignment.
- This is a non-negotiable requirement for adversarial attacks.

# Part 2: Backdoor Defense (45 points)

Your job is to implement the Neural Cleanse [paper] backdoor defense (detection, identification and mitigation technique) to defend against a backdoored model. You are assigned a specific backdoored VGG16 model (trained on GTSRB dataset) (`vgg16_gtsrb_backdoored_0~5.pth`). We provide six different backdoored models. Each student is assigned a different backdoored model for you to defend based on your CNETID (see cnetid_backdoor_hashed.csv). The CNETID has been anonymised using the SHA-256 hash algorithm. To find which backdoor model to use for part2, enter your CNETID into a SHA-256 hash generator and search for the matching hash in the csv file. You must create a defense against your assigned backdoor model.

Neural Cleanse works by finding the minimal trigger pattern needed to cause misclassification to each possible class. Then, using anomaly detection, it identifies if any class requires a significantly smaller trigger pattern (indicating a backdoor). Then, reverse-engineering the true trigger pattern for that class. Then mitigating the backdoor through various techniques. For this homework, you are required to implement the unlearning (fine-tuning) method and produce a *'clean' model*. Read the paper for more detail. If you need, there is a TensorFlow version of the implementation by the authors [here].

Starter files are on Canvas in Files → Homework → hw4_part2:

- This includes hw4_part2_starter.py, cnetid_backdoor_hashed.csv, part2_defense_training_parameters.pdf
- The backdoored models are [here] and the models folder (with `vgg16_gtsrb_backdoored_0~5.pth`).
- cnetid_backdoor_hashed.csv' contains each student's assigned backdoored model. Each student will be graded based on performance specific to their own backdoored model.
- There are snippets for loading datasets and models in the 'hw4_part2_starter.py' file.

The six backdoored models are [here] (with `vgg16_gtsrb_backdoored_0~5.pth`). These correspond to your assigned model in cnetid_backdoor_hashed.csv.

Install Packages:

- You can use any built-in library, along with any of the following packages: torch, torchvision, numpy, Pillow

Time Estimations (for the entire defense including detection, identification and mitigation): it depends on many factors such as the learning rate, optimization function, number of epochs for training during mitigation, etc. However, a general guideline is:

- On Newt: 14 ~ 25 minutes
- On Floo: 17 ~ 26 minutes
- Local (Apple M2 Pro - setting `device = torch.device('mps')`): 20+ minutes

## Specifications:

Your task is to implement the Neural Cleanse defense on a backdoored model (VGG16 model trained on GTSRB + trigger pointing to a single target class. This model is given to you `vgg16_gtsrb_backdoored_0~5.pth` check cnetid_backdoor_hashed.csv for the backdoored model assigned to you).

Implementation Guideline:

1. Detection Phase
    a. Implement the trigger optimization algorithm to find the minimal perturbation needed to misclassify inputs to each class.
    b. Apply the optimization for all classes in the GTSRB dataset (43 classes).
    c. Implement the median absolute deviation (MAD) anomaly detection to identify the backdoored target class.
2. Identification Phase
    a. Identify the suspected backdoored target label
    b. Extract the reverse engineered trigger pattern (i.e. approximation of the actual trigger used to poison the backdoor model)
3. Mitigation phase (unlearning method)
    a. Create a training set with both clean inputs and inputs with the reverse engineered trigger applied
    b. Train the model to classify all images (included trigger-applied ones to their correct classes)
    c. Use the clean training set from GTSRB for this re-training (fine-tuning) process

Make sure to always do visual inspection on your predicted reverse-engineered trigger pattern.

## Format:

Implement a defense against a backdoored model in a function called “part2” in the starter file (hw4_starter_part2.py). Deliverables:

1. ‘`part2_defense.py`’: complete training script implementing the entire defense.
2. ‘`part2_reverse_engineered_trigger.pth`’: the reverse engineered trigger pattern as a PyTorch tensor file (.pth) during your defense implementation. You need this to load the saved trigger pattern in the part2 function that applies this trigger on images (used in grading).
3. ‘`part2_defense_training_parameters.pdf`’: fill out the pdf with the actual parameters that you used for training the clean model (mitigating backdoors).
4. ‘`part2_backdoor_defended_model.pth`’: the defended/clean model after backdoor mitigation i.e., the model saved after training the model with trigger-applied inputs.
5. ‘`part2_starter.py`’: completed version of the starter code with implemented ‘part2’ function. Leave the function signature exactly as it is. As written in the starter file, the function inputs an image, applies the reverse engineered trigger and returns the triggered image.
6. ‘`part2_backdoor_defense_trigger_prediction.pdf`’: your predicted backdoored target class and visual representation of your reverse-engineered trigger.

All files must be submitted with the **exact filenames specified above**. For files that are not already given to you, you need to create a new file with the correct content. When you submit, make sure all the files are in the same directory (i.e. do not create a folder and put some files in a folder).

## Grading:

Detection (5 points): We will view `part2_backdoor_defense_trigger_prediction.pdf`

- 5 points: Correctly identify the target class.

Mitigation (20 points): We will run inference on your ‘`part2_backdoor_defended_model.pth`’ (i.e., the clean model that you get after implementing Neural Cleanse) and evaluate the performance on 50 random samples from the GTSRB testset. Since the backdoored models are different (but with similar difficulty level), we provide a relative threshold.

- 20 points: |acc(defended_model(A)) - acc(backdoored_model(A))| ≤ 7%
- 10 points: 7% < |accuracy difference| ≤ 15%
- 5 points: 15% < |accuracy difference| ≤ 20%
- 0 points: |accuracy difference| ≥ 20%

Backdoor Success (20 points): We will select 50 random samples from the GTSRB test set, apply your reverse-engineered trigger to these images using your `part2` function (from `part2_starter.py` (giving test set A_t)). We will then pass these trigger-applied images through your defended model (`part2_backdoor_defended_model.pth`) and calculate the attack success rate (the % of these trigger-applied images that are classified as the target class).

- 20 points: Attack Success Rate(defended_model(A_t)) ≤ 10%
- 10 points: 10% < Attack Success Rate(defended_model(A_t)) ≤ 30%
- 0 points: Attack Success Rate(defended_model(A_t)) ≥ 30%