**Name: Abhignya Bhat**
**Netid: Ayb5037**

# LAB 4 : Backdoor Attack

**Github:**
https://github.com/Abhignya-Bhat/ML_Sec_Lab4/tree/main

**Introduction:**
Backdoor attacks on neural networks refer to malicious attempts to compromise the integrity and security of machine learning models. In these attacks, an adversary aims to insert subtle modifications into the training data or manipulate the model itself during the training process, with the goal of introducing a hidden vulnerability. These vulnerabilities, often referred to as "backdoors," can be exploited later to manipulate the model's predictions in a specific way.

**Procedure:**
I was tasked with repairing a BadNet B, accompanied by validation and test datasets containing both clean and poisoned inputs. To address the issue, I had to prune the last pooling layer (pool_3) of BadNet B systematically. This involved the gradual removal of one channel at a time from the pooling layer, with channels prioritized based on their average activation values across the entire validation set. The pruning process occurred in the convolutional layer (conv_3) immediately preceding the pooling layer, as the pooling layer lacked trainable parameters. Essentially, pruning was equivalent to masking the output from the pooling layer for the specified channel. After each channel removal, I assessed the new validation accuracy of the pruned BadNet. The pruning operation continued until the validation accuracy dropped by at least X% below the original accuracy, where X took values of 2%, 4%, and 10%. The resulting network was denoted as B'. Subsequently, I constructed a new network, GoodNet G, with the following functionality: for each test input, the input was processed through both B and B'. If the classification outputs matched (i.e., both networks predicted class i), the final output was class i. In cases where the outputs differed, the final output was designated as N+1.

**Evaluation:**

- To assess the model's performance, I was told to utilize the provided 'eval.py' script by executing the following command in your terminal or command prompt:
  python3 eval.py <clean validation data directory> <poisoned validation data directory> <model directory>
- However, it is worth mentioning that I have already incorporated the evaluation logic from 'eval.py' into my Jupyter notebooks to analyze the model's performance against the specified dataset. Therefore, running the script separately is not necessary, as the evaluation process has been integrated into the notebook for your convenience.

**How to Execute:**

- Download all necessary files as mentioned in the Readme on the repo.
- Ensure you have all the libraries used installed as mentioned in the Readme.
- When running any of the notebook, just put in the correct paths to the data and models wherever required.

**Note:**
I ran the notebooks on Google Colab Pro, as running it on the Free Tier version or locally was not feasible because of memory constraints.

**Results and Observations:**

**Note:**
Following the prescribed task of pruning channels based on decreasing average activations, a notable and rapid decline in accuracy was observed. In fact, the accuracy plummeted by over 4% after pruning just one channel, and this trend continued with a further decrease exceeding 10% after pruning another channel.This did not seem right, so I conducted a second experiment by pruning channels in the opposite order, i.e., in increasing order of average activations. I have provided two Jupyter notebooks, one for each approach (increasing and decreasing order). Additionally, a third notebook is available for evaluating the models using the provided script.

Channels sorted in decreasing order of channel activations:

| | **Accuracy** | **Attack Success Rate** |
|---|---|---|
| **Original** | 98.62% | 100.00% |
| **Repaired (X=2%)** | 94.13% | 100.00% |
| **Repaired (X=4%)** | 94.13% | 100.00% |
| **Repaired (X=10%)** | 74.24% | 100.00% |
| **GoodNet(X=2%)** | 93.95% | 100.00% |
| **GoodNet(X=4%)** | 93.95% | 100.00% |
| **GoodNet(X=10%)** | 74.09% | 100.00% |

Channels sorted in increasing order of channel activations:

| | **Accuracy** | **Attack Success Rate** |
|---|---|---|
| **Original** | 98.62% | 100.00% |
| **Repaired (X=2%)** | 95.90% | 100.00% |
| **Repaired (X=4%)** | 92.29% | 99.98% |
| **Repaired (X=10%)** | 84.54% | 77.20% |
| **GoodNet(X=2%)** | 95.74% | 100.00% |
| **GoodNet(X=4%)** | 92.12% | 99.98% |
| **GoodNet(X=10%)** | 84.33% | 77.20% |

Here, Original =BadNet B, Repaired =pruned BadNet B', GoodNet =Combined model (B & B')

Looking at either table, it's clear that the implementation of the pruning defense provided some level of protection; it was accompanied by a trade-off in terms of accuracy. The attack wasn't significantly avoided by the pruning defense, indicating that the attack was able to resist it. In essence, the pruning defense, while offering some level of protection, did not prove to be fully effective in mitigating the impact of the attack.