|  |  |  |
| --- | --- | --- |
| Fall 2024 | CS53331/4331 Adversarial Machine Learning | Assignment 2 |

|  |  |
| --- | --- |
| **Title** | **The Certification and privacy attack stuff** |
| **Due date** | November 4th |
| **First Name** | Ucchwas Talukder |
| **Last Name** | Utsha |
| **Student ID** | R11836597 |
| **Marks** | 100 |

Note: Please answer the following questions and submit them through Blackboard. Be sure to submit it to assignment 1. DO NOT write the report by hand and submit a scanned version. Just write the answers in a Word document and submit it. Both Word and PDF submissions are accepted.

# Submission Instruction (3 documents)

You are required to submit three documents:

1. ***Report.*** Just fill out the above report and submit it as a Word or PDF document.
2. ***Ipynb file.*** The code that you have written. Preferably in an ipynb document. You can submit it as a .py file as well.
3. ***Txt file of the code.*** We need your code in the .txt file as well. Use whatever way you prefer. The fastest would be to download the file as a .py file and change the extension to .txt

# Objectives

This assignment has three main objectives:

### Part 1

1. Continuing on Assignment 1 objectives, the first objective in this assignment will be to certify the build VGG from the last assignment using Randomized smoothing.

### Part 2

1. Implement privacy attack (label one and shadow modeling)

### Part 3

1. Use DP as a defense

# Get started

Download the assignment files from Blackboard. You will need the report (This file), the .ipynb file where you will put your code, the dataset, and the given model (if any). The GTSRB dataset will be used for both parts of this assignment.

# Dataset

The German Traffic Sign Recognition Benchmark (GTSRB) dataset consists of almost 51K images of traffic signs. There are 43 classes, and the size of their images is 32×32 pixels. Some of the images are shown below. Please download the dataset from Blackboard or [here](https://drive.google.com/file/d/1lt6qvYnkEZ-qXREufkzjonocn_1_moPi/view?usp=drive_link). More information about the dataset can be found [here](https://www.kaggle.com/datasets/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign).



As we did with earlier assignments, it is recommended that you upload the dataset into your personal Google Drive to follow the Colab instructions as they are. Of course, if you prefer to use other than Colab, you will need a similar preprocessing.

# Instruction for Colab (repeated from Assignment 0)

To get started with Google Colab, simply go to [Google Colab](https://colab.research.google.com/), sign in with your Google account, and create a new notebook. You can write and execute Python code directly in the notebook. To access your dataset stored in your Google Drive (previous step), first run the following code to mount your Drive:

A black background with white text

Description automatically generated

Follow the authorization steps, and your Drive will be accessible at **/content/drive/My Drive/.** You can then load your dataset into the notebook by providing the correct file path. This part of the code is provided for you in the .ipynb file of Assignment 0. You will need to setup the drive connection and run the code.

To use the free GPU provided by Colab, you can change the runtime to access a GPU by clicking on **"Runtime" > "Change runtime type” and** selecting **"T4 GPU"** from the **Hardware accelerator** dropdown menu. You can always use higher GPU powers at a cost (Colab Pro is $10 per month), but you should be fine with the free version, considering that you start the assignment early enough.

Colab comes with many pre-installed libraries, but if you need to install additional Python packages, you can do so with pip. For example:



Remember to save your work frequently.

After you've completed your work in Google Colab, you can easily download your notebook from Google Colab, go to **"File" > "Download" > "Download.ipynb"**.

# Other than Colab

If you don’t prefer Colab or notebook, you always have the option to run it on your computer (especially if it has a GPU) or access HPCC resources at TTU (needs an account with my permission).

# Additional resources

1. TensorFlow resource <https://www.tensorflow.org/>
2. PyTorch resources <https://pytorch.org/get-started/pytorch-2.0/>
3. Deep learning with Python <https://dl-with-python.readthedocs.io/en/latest/>
4. Get started with Colab <https://colab.research.google.com/>

# Part1 Certified Training

# Task 1: Randomized Smoothing (30 pts)

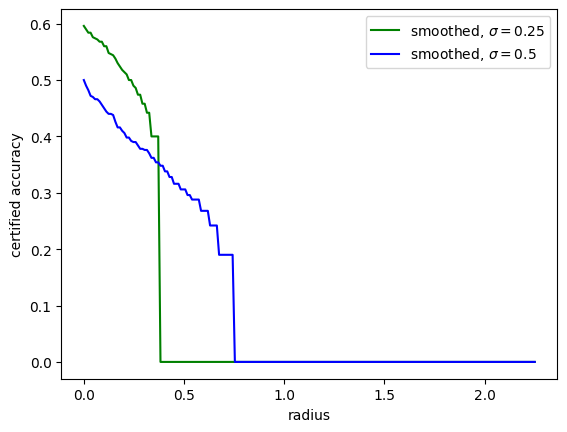
Our first task with implement randomized smoothing for the VGG model from last time. Your task is to download the model and use [Adversarial Robustness Toolbox (ART)](https://adversarial-robustness-toolbox.readthedocs.io/en/latest/) to implement randomized smoothing to certify the model. [This notebook](https://github.com/Trusted-AI/adversarial-robustness-toolbox/blob/main/notebooks/output_randomized_smoothing_mnist.ipynb) should be your guide to implementing the task. You will use as Gaussian noise parameter. Once needed, use 100 as the number of samples to estimate the certifiable radius. You will find the standard and certified accuracy for different noises. You will also plot the L2 radius versus certified accuracy.

***Notes:*** Training will take time to generate the results. Try it first with only 2 training rounds and then increase to 25. Certification for all the testing will take forever. Thus, whenever you need a testing set, use a subset of the testing samples, not all. We already provided that subset for you in a variable that you must look for ☺.

1. [10 pts] Implement what is required.
2. [10 pts] Fill in Table 1 with the required values

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 1: Standard and Certified Accuracies for different models | | | |  |
| **Model** | **L2 Radius** | **Standard accuracy** | **Standard coverage rate** | **Certified accuracy** |
| VGG16 original |  | 96.8% |  |  |
| Smoothed | 0.1 | 10.6% | 10.2% | 7.6% |
| Smoothed | 0.1 | 3.0% | 4.0% | 2.2% |
| Smoothed | 0.1 | 4.0% | 28.6% | 3.4% |

1. [5 pts] Plot L2 radius (from 0 to 3) vs certified accuracy for the three models (original, smoothed 0.25, and smoothed 5) in one figure. An example of some smoothed certified accuracies is below.



**Ans:**

A graph with blue and orange lines

Description automatically generated

1. [5 pts] Briefly provide insights on the results.

**Ans:** The results indicate that as the noise level (σ²) increases, the standard accuracy and certified accuracy generally decrease. This trend suggests a trade-off between model accuracy and robustness: adding more noise to achieve higher certification thresholds (for adversarial robustness) comes at the cost of reduced classification accuracy on the test data. The original VGG16 model achieves the highest standard accuracy without smoothing, while the smoothed models with larger noise levels achieve low certified accuracies, especially for larger σ values. This implies that while randomized smoothing can provide some degree of robustness, it significantly impacts the model's overall accuracy, especially at higher noise levels.

# Part 2 Privacy Attacks

# Task 2-Label-only attack (25 pts)

Now, let’s do some **privacy attacks**. We will focus on Membership inference attacks. This task will implement label-only attacks, while the next will implement shadow modeling. As discussed in class, the label-only attack is developed based on the fact that it costs an attacker so much to generate a successful adversarial sample for a member sample. In contrast, the cost will be much less for a non-member sample. Thus, it uses HopSkipJump to generate a successful adversarial sample and then measure the perturbation using the L2 norm. If the perturbation is above a threshold, it will be classified as a member; if not, it will be classified as a non-member. Hence, we will follow the following steps to perform the attack (Your reference code is [this notebook](https://github.com/Trusted-AI/adversarial-robustness-toolbox/blob/main/notebooks/label_only_membership_inference.ipynb)):

1. Generate a dataset (already given): The training and testing datasets are merged into one, with the training classified as members and the testing as non-members. We picked 1500 samples from this dataset to build our attack model.
2. Calculate the threshold (to be implemented by you): Your task here is to use existing implementations in ART to calculate/calibrate the threshold. All parameters are given to you in the code (in comments).
3. Infer the data: Use the 1500 provided samples to predict whether they are members. Then, compare your performance to the ground truth and present the results.

One parameter you will need to set is the maximum number of queries (used in step 2). Set this parameter to 5 and 10. All that is needed is to answer the questions below.

**Note on resources:** It takes time to generate the adversarial examples. A good GPU computer will take you around 1-3 hours per round. Thus, plan your time accordingly.

1. [10 pts] Implement what is required.
2. [10 pts] Fill out Table 2.

|  |  |  |  |
| --- | --- | --- | --- |
| Table 2: Attack accuracy with different number of queries. | | | |
| **Max number of queries** | **Members accuracy** | **Non-member accuracy** | **Overall accuracy** |
| **5** | 2.76% | 96.13% | 28.47% |
| **10** | .28% | 100% | 27.73% |

1. [5 pts] Briefly provide insights on the results.

**Ans:** Here, we’re trying to guess whether a sample (image) was in the training dataset (member) or the testing dataset (non-member) of a machine learning model. The results indicate a clear disparity in attack effectiveness: high accuracy for non-members (96.13% to 100%) suggests that non-members require minimal perturbation for successful classification, as they haven't been learned as extensively by the model. In contrast, very low accuracy for members (2.76% to 0.28%) reveals that members need significantly more perturbation for successful classification. This happens because training samples are closer to the model’s decision boundary which makes them inherently more challenging to perturb into incorrect classifications. This imbalance results in a low overall accuracy (around 28%). It demonstrates that while the attack effectively distinguishes non-members, it struggles to classify members under limited query conditions. In essence, the model’s familiarity with training samples (members) provides them with a form of resilience against small adversarial changes, while non-members lack this protection.

# Task 3- Shadow Models attacks (25 pts)

Let’s do another privacy attack. This time, it is a “shadow models” attack, discussed in class[. This notebook](https://github.com/Trusted-AI/adversarial-robustness-toolbox/blob/main/notebooks/attack_membership_inference_shadow_models.ipynb) will be your guide. You are going to use ART again, and here are a few steps to follow:

1. Build the shadow models based on the numbers provided below. Set the training ratio for them to 0.5.
2. Measure the accuracies for them (use validation accuracy as a measure)
3. Build the attack model. You can build any binary classifier. You are suggested to use a random forest classifier for simplicity.
4. Evaluate your attack models for members and non-members (the evaluation data is already given to you).
5. Calculate the precision and recall of your attack model.

Your changing parameter in this task is the number of shadow models. You will set it to 1,2 and 5. Please note the needed metrics so you can plan your code accordingly.

**Note on the implementation:** Our implementation will look strange. Shadow models will require lots and lots of data. Thus, in a general setup, researchers use 80% of the data for shadow models, and only 20% is for training/testing/validation. However, given that we are using a model already built from Assignment 1, we cannot use this general setup. Thus, our shadow models’ data will concatenate training and testing while the validation will be left out to test performance accuracy. To evaluate the attack, we generated a separate dataset that is taken from training and testing. You will need to locate and work with that data. In general, your attack results won’t be good due to the above design, but the idea is that you should analyze it.

1. [10 pts] Implement what is required.
2. [10 pts] Fill out Table 3.

|  |  |  |  |
| --- | --- | --- | --- |
| Table 3: Shadow model performance and attack performance | | | |
| **Metric** | **1 shadow model** | **2 shadow models** | **5 shadow models** |
| **Average Accuracy (using validation)** | 94.22% | 93.65% | 93.73% |
| **Members accuracy** | 94.28% | 94.70% | 95.13% |
| **Non-members accuracy** | 5.57% | 5.24% | 5.00% |
| **Overall accuracy** | 49.92% | 49.97% | 50.06% |
| **Attack precision** | 49.96% | 49.98% | 50.03% |
| **Attack recall** | 94.28% | 94.70% | 95.13% |

1. [5 pts] Briefly provide insights on the results.

**Ans:** Here are some key insights based on the results:

**Increasing Shadow Models:** As the number of shadow models increases from 1 to 5, the average validation accuracy remains stable around 93-94%, which indicates consistent generalization across models.

**Member vs. Non-member Accuracy:**

**High Members Accuracy:** The attack model effectively identifies members, with accuracy improving slightly as more shadow models are used (from 94.28% to 95.13%).

**Low Non-members Accuracy:** The accuracy for non-members is very low (~5%), which suggests that the attack model struggles to distinguish non-members from members.

**Balanced Overall Accuracy and Precision:** Both overall accuracy and attack precision hover around 50%, which implies that the attack model has an average success rate in differentiating members from non-members.

**Recall Improvement:** Attack recall mirrors members' accuracy, indicating that the attack model becomes slightly better at identifying members as more shadow models are used.

The results show that the attack model is biased toward predicting membership. The models achieve high recall for members but poor accuracy for non-members. Shadow models capture general data patterns but don’t provide enough distinction between members and non-members. Also, our setup limits the usual data allocation for shadow models. This approach may reduce the attack's effectiveness, as shadow models typically benefit from larger datasets.

# Part 3

# Task 4- Deferentially private model performance (20 pts)

Now, let’s defend test defense with DP. We have built a DP-SGD model using the TensorFlow privacy library. The model is given to you, and you just need to analyze it. First, check the overall DP code and answer the below questions. Then, analyze the given DP model in terms of accuracy and resiliency to membership inference attacks. Use a label-only membership inference attack with 5 max-queries and the same setup as in Task 2.

1. [5 pts] Answer the following questions about the provided model or code.
   1. What is the value of the L2 norm used to build the model?

**Ans:** 2

* 1. What is the value of the noise added?

**Ans:** 1.6

* 1. What is the batch size and the micro-batch size to build the model?

**Ans:** 128, 128

* 1. Modify the commented code to enhance your accuracy performance (you can accept less privacy in this case). You don’t need to run the code; just tell us about the changes and maybe paste your code here.

**Ans:** Reducing noise\_multiplier from 1.6 to 1.0 can improve accuracy while slightly reducing privacy.

**Code:**

optimizer = tensorflow\_privacy.DPKerasSGDOptimizer(

l2\_norm\_clip=2,

num\_microbatches=128,

noise\_multiplier=1.2,

learning\_rate=LR

)

1. [5 pts] Measure the differential privacy guarantee of the given model.

**Ans:** Differential privacy guarantee (ε) for δ=1e-05: **0.63**

1. [5 pts] Calculate the model’s performance under attacks and any other observations regarding the results.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 4: DP vs original performance | | | | |
| **Model** | **Accuracy** | **Members accuracy** | **Non-member accuracy** | **Overall accuracy** |
| **VGG16 (Original)** | 93.97% | 1.63% | 95.23% | 26.47% |
| **DP** | 50.01% | 2.09% | 95.48% | 26.87% |

1. [5 pts] Briefly explain why DP accuracy is lower than VGG original. Provide insights on the results.

**Ans:** The DP model's accuracy is lower than the original VGG16 due to the noise added to protect individual data privacy. This noise disrupts the model's ability to learn specific data patterns, which results in reduced accuracy.

**Privacy vs. Accuracy:** The DP model sacrifices accuracy for privacy, demonstrating the trade-off inherent in differential privacy.

**Improved Privacy:** The DP model shows slightly better resilience to membership inference attacks, benefiting from the added noise that obscures individual data points.

**Use Cases:** DP models are suitable for privacy-sensitive applications, where protecting data privacy outweighs the need for high accuracy.

# Submission Instruction (3 documents)

You are required to submit three documents:

1. ***Report.*** Just fill out the above report and submit it as a Word or PDF document.
2. ***Ipynb file.*** The code that you have written. Preferably in an ipynb document. You can submit it as a .py file as well.
3. ***Txt file of the code.*** We need your code in the .txt file as well. Use whatever way you prefer. The fastest would be to download the file as a .py file and change the extension to .txt