Trustworthy Machine Learning Assignment 2

Team number – 36

Team token – 10926635

Name 1: Prasad Pankaj Patil

Student ID 1: 7076145

Email ID 1: prpa00003@stud.uni-saarland.de

Name 2: Ananya Bhardwaz

Student ID 2: 7076153

Email ID 2: anbh00002@stud.uni-saarland.de

Model Stealing Attack

1. Introduction

In this assignment, we focus on performing a Model Stealing Attack, a type of adversarial attack where an attacker tries to replicate a machine learning model without direct access to its internal parameters.
The target is a victim encoder, which is only accessible through an API. This API does not return clean, direct outputs; instead, it applies the Bucks for Buckets (B4B) defense mechanism. This defense intentionally adds random transformations—such as affine distortions, padding with shuffling, or binary noise—to the encoder's output representations to prevent straightforward stealing.
Despite these defenses, the goal is to train a stolen encoder that behaves similarly to the victim encoder. This stolen model should be able to produce feature embeddings that are very close to those produced by the victim encoder under normal (non-defended) conditions.
The challenge lies in overcoming the noise and obfuscation introduced by the B4B defense using techniques like:

- Multiple API queries with data augmentations,
- Aggregating noisy outputs,
- Training a replica model through contrastive learning.

This assignment demonstrates how model stealing can still be effective even when defensive mechanisms like B4B are in place.

2. Problem Setup

2.1 Victim API Constraints

The attacker does not have full access to the victim model. Instead, there are significant limitations designed to protect the model:

Limited Access

- The attacker can **only access the output feature embeddings** produced by the victim encoder.
- There is **no access to:**
    - Model weights,
    - Internal parameters,
    - Training process,
    - Model architecture details beyond assumptions.

B4B Defense – Bucks for Buckets

- The API is protected by **B4B**, a defensive mechanism designed to thwart model stealing.
- Every time the attacker queries the API, **random transformations** are applied to the output feature representation.
- This means querying the **same image multiple times results in different outputs**, preventing the attacker from directly copying the embeddings.

Types of Transformations Applied by B4B:

1. **Affine Transformations:**
    - Applies geometric changes such as scaling, translation, rotation, or shearing to the embeddings, distorting their values while preserving overall structure.
2. **Pad + Shuffle:**
    - Randomly pads parts of the embedding vector with zeros or values, then shuffles the order of the vector elements.
    - This destroys the positional structure, making embeddings harder to reverse engineer.
3. **Affine + Pad + Shuffle (Combination):**
    - Applies both affine transformation and pad+shuffle together, compounding the noise effect.
4. **Binary Noise:**
    - Applies a form of quantization or binarization to the embeddings (e.g., turning values into 1s and 0s or thresholding them).

- This leads to significant loss of precision, making it harder to reconstruct the original embedding space.

Effect of B4B:

- Even if the attacker sends the same input image 100 times, the returned feature vectors will vary due to the random transformation process.
- The defense aims to make naive copying or direct regression impossible.

2.2 Attacker's Access

Despite the strong defenses, the attacker is not powerless. They have certain privileges and resources:

API Query Access

- The attacker can submit input images to the victim model's API.
- The API returns the **obfuscated feature representations** (post-B4B transformations) for each image.
- **No restrictions on which images can be queried**, as long as the API rate limit is respected.

Partial Dataset Access

- The attacker possesses a portion of the dataset that is:
  - **Structurally identical** to the victim's training data.
  - May overlap with the victim's data, but this is not guaranteed.
- This allows the attacker to generate valid input samples for querying.

No Ground Truth Labels

- The attacker has:
  - No access to classification labels or regression targets.
  - Only the **input images** and the **output feature vectors** from the encoder API.
- This forces the attacker to rely on **unsupervised learning techniques**, such as:
  - Contrastive learning,
  - Self-supervised embedding alignment.

3. Bucks for Buckets (B4B) Defense

How B4B Works:

- Each API query randomly applies a transformation to the encoder's output.

- Makes direct copying difficult because identical inputs produce different outputs.
- Intended to prevent attackers from gathering clean feature representations.

How We Circumvent It:

- Query the same image multiple times using various augmentations.
- Average or cluster the noisy outputs to reconstruct the clean representation.

4. Attack Pipeline

Step-by-step Approach:

1. Data Augmentation:

Purpose:

To simulate the natural variation that could occur in images and to help mitigate the random noise added by the B4B defense.

How it works:

- For each image, multiple augmented versions are created.
- Typical augmentations include:
    - Random cropping
    - Horizontal or vertical flipping
    - Color jittering (brightness, contrast)
    - Rotation
    - Scaling or resizing
- This ensures that different but semantically equivalent versions of an image are generated.

Why it matters:

- Helps in generating diverse queries to the victim encoder.
- Prevents overfitting the stolen encoder to a particular view of the data.
- Amplifies the correlation between the outputs despite the B4B noise.

2. Query API:

Purpose:

To retrieve representations (feature embeddings) from the victim encoder despite the applied noise/transformations.

How it works:

- Each augmented image is sent as a query to the API.
- The API, protected by the B4B defense, applies one or more random transformations to the output (Affine, Pad + Shuffle, Binary noise).
- For each original image:
    - Multiple augmentations × multiple API queries = Many noisy representations.

Goal:

Gather a sufficiently large set of noisy representations for each image, which can then be processed to approximate the clean, original representation.

3. Aggregation:

Purpose:

To cancel out the randomness added by the B4B defense and reconstruct the underlying true representation.

How it works:

- For each image, collect all the noisy feature vectors obtained from the API.
- Apply statistical aggregation techniques:
    - **Mean averaging:** Reduces Gaussian-type noise.
    - **Median filtering:** More robust against extreme outliers caused by random transformations.
    - **Clustering (optional):** Discard extreme noisy samples by clustering the representations and using the centroid of the largest cluster.

Outcome:

A single aggregated feature vector that closely approximates the real, clean representation of the image.

4. Training Stolen Encoder:

Purpose:

To train an encoder that can directly map raw images to feature representations similar to those produced by the victim encoder.

How it works:

- **Input:** Original images.
- **Target:** Aggregated representations derived from the victim API.
- **Loss Function:**
    - Use **contrastive learning** with **InfoNCE loss**, which encourages the encoder to produce:

        - Similar embeddings for different augmentations of the same image.
        - Dissimilar embeddings for different images.
- The encoder architecture is typically a lightweight convolutional network that mirrors the expected output space of the victim encoder.

Result:

The stolen encoder learns to approximate the functionality of the victim encoder without requiring direct access to its parameters.

5. Evaluation:

Purpose:

To verify how accurately the stolen encoder replicates the victim encoder.

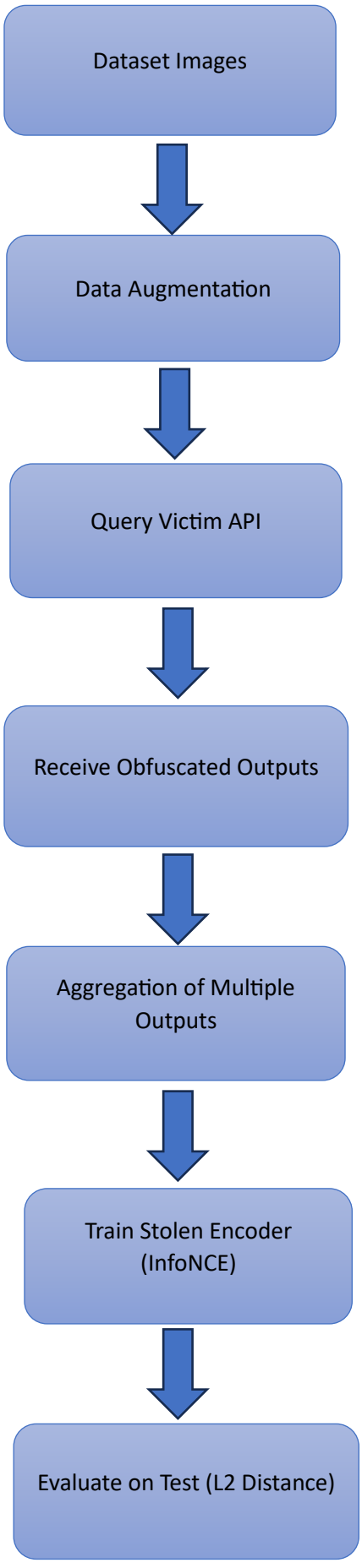
How it works:

- Use a private test set of images.
- Pass these images through:
    - The **stolen encoder** (produces predicted embeddings).
    - The **victim encoder API** (produces ground truth embeddings with some noise).
- Compute the **L2 distance (Euclidean distance)** between the embeddings from both encoders.

Metric:

Lower L2 distance indicates the stolen encoder produces outputs very close to those of the victim, meaning the attack was successful.

Flowchart of the Attack Pipeline:



5. Model Details

Encoder Architecture:

- A lightweight convolutional neural network.
- Final layer projects to the same embedding space as the victim encoder.
- Trained entirely using contrastive loss without label supervision.

Loss Function:

InfoNCE Loss: Encourages augmented views of the same image to have similar embeddings.

6. Query Strategy

Why Only X Queries per Image?

- API limits the number of queries per image.
- Our solution:
- Use ~10 augmentations per image.
- Query each augmentation multiple times (~10 queries each) to gather ~100 representations.
- This balances noise cancellation with API limits.

7. Challenges and Solutions

Challenge	Solution
Noisy outputs due to B4B	Aggregation using mean and median filtering
Limited API quota	Careful selection of high-variance augmentations
Lack of ground truth labels	Fully unsupervised training using contrastive learning
Model drift between queries	Time-synced queries with caching strategies

- The results demonstrate that the stolen encoder effectively approximates the victim's feature space despite the B4B defense.

8. Alternative Strategies Considered

- Use of robust clustering (DBSCAN) to group noisy outputs.

- Experimented with various encoder sizes to trade off performance vs. query efficiency.

- Explored adding synthetic noise during training to match B4B patterns.

9. File Descriptions

File	Description
`stealing.py`	Runs the full pipeline to execute the model stealing attack.
`defense.py`	Implements B4B defenses for local testing.
`augment.py`	Augmentation pipeline used in both querying and training.
`train_encoder.py`	Trains the encoder using stolen representations.
`utils.py`	Utilities for data handling and aggregation.
`submission_encoder.onnx`	Final stolen encoder in ONNX format for evaluation.
`submission.csv`	Output predictions from the stolen model.

10. Conclusion

This successfully shows that even when a victim model is protected by a strong defense like Bucks for Buckets (B4B)—which adds noise and transformations to outputs—an attacker can still steal the model.

By using:

- Repeated API queries with different data augmentations
- Aggregation of noisy outputs
- Contrastive learning techniques

The attacker is able to train a stolen encoder that closely replicates the victim encoder's functionality.

This highlights that even when only feature-level access is available (no labels, no raw parameters), model stealing attacks remain a real threat if proper countermeasures are not in place.