TML Assignment 2

Model Stealing in Self-Supervised Learning

This assignment explores the implementation of a model stealing attack on a self-supervised victim encoder, which is protected by the Bucks-for-Buckets (B4B) defense. The adversary has access to:

- A partial training dataset of the victim encoder
- Knowledge of the dataset structure
- API access to the victim encoder

We implemented two approaches inspired by recent literature to evaluate their performance under the B4B defense. All implementations, representations, and the final stolen model are submitted accordingly.

Approach 1: Query Once and Train with Augmentations (Inspired by StolenEncoder)

This method is based on the *StolenEncoder* paper by Liu et al. [1]. The key observation is that contrastive learning encoders produce similar representations for an image and its augmented versions. Hence, if an attacker queries the API once per image and performs local augmentations, a stolen encoder can be trained to mimic the victim.

Surrogate Dataset

- We collected a small, unlabeled surrogate dataset.
- Each image was queried once through the API.
- The returned 1024-dimensional representation was stored along with the image.

Query Strategy

- A single query per image was used to minimize query cost.
- No augmented images were sent to the victim encoder; augmentations were performed locally during training.

Training the Stolen Encoder

- We used a ResNet-18 architecture as the stolen encoder.

- Multiple augmentations per image were generated using transforms like RandAugment, ColorJitter, RandomGrayscale, and Horizontal Flip.
- The model was trained using InfoNCE loss (a contrastive loss) for 100 epochs.
- Positive pairs consisted of `(augmented_image, victim_representation)`; negative pairs were drawn from different images.

Performance Evaluation

- The fidelity of the representation space was partially assessed using L2 distance.
- Visualization techniques like t-SNE were considered for further qualitative comparison.

Approach 2: Querying with Augmented Images (Inspired by Cont-Steal)

This approach follows the *Cont-Steal* method proposed by Sha et al. [3]. Instead of querying with original images, augmented views are sent directly to the victim encoder during training.

Query Strategy

- For each training batch, we generated augmented versions of images locally.
- These augmented images were sent to the API to get their transformed representations (after B4B defense).
- These were used directly in the contrastive loss along with the other locally generated augmentation to train the stolen encoder.

Contrastive Loss Setup

- Positive pairs were created using:
  ```
  (stolen_encoder(augmented_view_s),
  victim_encoder(augmented_view_t))
  ```

- Negative pairs were formed by mixing representations from different images and encoders.
- Training again used InfoNCE loss and similar augmentations as in Approach 1.

Samples & Querying Challenges

- **Approach 1:** We successfully queried the API for 6000 samples before encountering errors. Despite retrying on different ports, the API became unresponsive. We had to

proceed with the available representations.

- **Approach 2:** We aimed for 15,000 queries using online augmentations but faced server issues after 5000 queries. As a result, training proceeded with a reduced dataset.

Augmentations Used

For both approaches, we used the RandAugment strategy to generate two augmented views per image, as recommended by Sha et al. [3].

Results

- **Approach 1:** Achieved an L2 distance of 26 on 30% of the private data.
- **Approach 2:** Due to API downtime, we could not complete evaluation or submit results at the time of writing this report.
- **MSE Loss:** We also tried using MSE loss but could not submit the results due to API downtime.
- **Local Results:** When running locally, the loss plateaued after a few epochs. We wanted to query for more data including augmented representations and representations of same images to improve on the model but we were running into API issues.

References

[1] Yupei Liu, Jinyuan Jia, Hongbin Liu, and Neil Zhenqiang Gong. *StolenEncoder: Stealing Pre-trained Encoders in Self-Supervised Learning.* Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 2115–2128. https://doi.org/10.1145/3548606.3560586

[2] Adam Dziedzic, Nikita Dhawan, Muhammad Ahmad Kaleem, Jonas Guan, and Nicolas Papernot. *On the Difficulty of Defending Self-Supervised Learning Against Model Extraction.* In ICML 2022.

[3] Zeyang Sha, Xinlei He, Ning Yu, Michael Backes, and Yang Zhang. *Can't Steal? Cont-Steal! Contrastive Stealing Attacks Against Image Encoders.* arXiv preprint, 2022. https://arxiv.org/abs/2201.07513

Repositories and Online Resources Consulted

- Cont-Steal GitHub Repository
- StolenEncoder GitHub Repository

- InfoNCE Loss Explanation: Medium article on NT-Xent Loss

Code Documentation:

convert_to_rgb

Ensures every PIL image is converted to a 3-channel (RGB) tensor before it is passed to ToTensor().

get_train_transform / get_eval_transform

Classic SimCLR-style augmentations:
• Random crop
• Horizontal flip
• Color jitter
• Random grayscale

ContrastiveStealingDataset

Generates two augmented “views” of the *same* image plus the target representation produced by the stolen model.

StolenEncoder

• Backbone: ResNet-18
• After four residual stages + global-average pool → 512-dimensional vector
• Projection head: BatchNorm + ReLU (reshapes the space for contrastive learning)

ContStealNTXent (InfoNCE Loss)

Contrastive loss that
• pulls each student view closer to the other *and* to the teacher embedding,
• treats all other examples in the batch as negatives.

load_and_combine_batches

Reads each representations_XXXX.pkl produced by querying the remote API.
For every 1 000-image slice it concatenates images, labels, and teacher embeddings into an ever-growing surrogate dataset—so the student eventually “sees” the full set without exhausting RAM.

train() – Training Loop

1. Generate z1 and z2.
2. Feed both views + teacher reps into criterion.
3. Back-prop → Adam step
4. Log batch loss with tqdm.

Returns epoch-average losses and saves a checkpoint to models/stolen_encoder.pth.

__main__

pgsql
CopyEdit
• Print startup logs
• Load surrogate dataset ← torch.load() of pickled TaskDataset
• Build cumulative training set via load_and_combine_batches
• Instantiate StolenEncoder, loss function, Adam optimiser
• Train model, plot loss curve, run t-SNE

All artefacts—stolen_encoder.pth, loss_curve.png, Stolen_Model_Embeddings_t-SNE.png—are written to the current working directory for later inspection.