TML Assignment 2

Name: Aadrish Mahmood
Student ID: 7075882

Model Stealing Attack on a Protected AI System

This report explains how I built a copy of a protected AI model by "stealing" its behavior through a fake API. The goal was to create my own version that works similarly to the original, but I only had a limited number of tries to figure out how it works.

Why I Chose This Method

I decided to use a ResNet-18 model because:

- It's good at learning patterns and features from images
- It works well without using too much computer power
- It's designed to understand visual information, which is perfect for 32×32 color images
- Using Mean Squared Error (a way to measure mistakes) makes sense when trying to match another model's outputs

I combined a medium-sized dataset (30,000 test images), a reliable neural network, and proper data preparation to make sure my copy learned useful patterns even with limited time.

Important constraint: Due to server problems and a 4-hour waiting period between attempts, I could only train my model using 30,000 images instead of the full amount allowed. Even so, the training worked well and showed that my method was effective.

What Information I Had Access To

1. **Protected AI System**

2. **Public Training Data:** A file called ModelStealingPub.pt containing sample images (3×32×32 pixels, already processed) that I could use to test the system and train my copy.

3. **Evaluation System:** A hidden test dataset that would judge how good my final model was, only available at the end.

How the Attack Works

I treated this like a learning problem: I would show images to both the original system and my copy, then train my copy to give the same answers as the original. Here were the rules I had to follow:

- **Maximum tries:** 100,000 images total (sent in batches of 1,000)
- **Speed limit:** Wait 60 seconds between requests to avoid being blocked
- **Batch size:** Must send exactly 1,000 images at a time

The Theory Behind It

The idea is that if my model can produce the same outputs as the original for many different inputs, it will learn to work similarly. Even though the original system adds random noise for protection, if I use enough examples and a good enough model, I can figure out the underlying patterns.

The Complete Flow

```JSON
1. Get Access → 2. Load Images → 3. Build Your Model

4. Training Loop:

   - Send 1000 images to protected AI

   - Get 1000 responses (1024 numbers each)

   - Feed same images to your model

   - Compare responses

   - Improve your model

   - Wait 60 seconds

   - Repeat until budget runs out

                     ↓
```

```
5. Export Model → 6. Submit for Evaluation
```

Model Stealing Attack - Code Explanation

What It Does

This code copies a protected AI model by learning from its responses to images. It's like reverse-engineering a secret recipe by tasting the output many times.

The Process

1. Setup

- Sets budget: 30,000 image queries maximum (because of lack of time and connection issues)
- Defines connection details and file paths
- Creates image container (TaskDataset class)

2. Get Access

```python
# Either load saved connection or request new access

resp = requests.get(LAUNCH_URL, headers={"token": TOKEN})
```

Gets permission to query the protected AI system.

3. Query Function

```python
def query_api(pil_images, port):

    # Check budget limit
    # Convert images to base64
    # Send to protected AI
    # Get back 1024 numbers per image
    # Update counter
```

Sends images to protected AI and receives its "understanding" (embeddings).

4. Prepare Data

```python
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(...)
])
```

Loads and preprocesses images for training.

5. Build Copy Model

```python
student = models.resnet18(pretrained=False)
student.fc = nn.Linear(in_feats, 1024)  # Output 1024 numbers like original
```

Creates ResNet-18 model to mimic the protected one.

6. Training Loop

```python
for epoch in range(EPOCHS):
    for batch in loader:
        target = query_api(images, PORT)    # Ask protected AI
        outputs = student(images)           # Ask your model
        loss = criterion(outputs, target)   # Compare answers
        # Update your model to match better
        time.sleep(60)  # Wait to avoid rate limits
```

What happens:

- Send 1000 images to both AIs
- Compare their responses
- Adjust your model to match the protected one better
- Wait 60 seconds between batches
- Repeat until budget runs out

7. Export & Submit

python

```python
torch.onnx.export(student, ...)  # Save model
requests.post(SUBMIT_URL, ...)   # Submit for evaluation
```

How I Built It

1. Getting Access to the System

- **Connection:** Send a request to /stealing_launch to get connection details

- **Saving info:** Store the connection details in a file called api_info.json so I don't lose them
- **Authentication:** Use a special token to prove I'm allowed to access the system

2. Making Requests with Limits

```None
Maximum allowed: 100,000 images
Images used so far: 0

Function to ask the system:
- Check if I have enough requests left
- Send the images and get back the results
- Update my counter of how many I've used
```

3. Building and Training My Copy

- **Architecture:** Used ResNet-18 (a popular image recognition model) with a final layer that outputs 1,024 numbers (same as the original)
- **Learning method:** Minimize the difference between my model's outputs and the original's outputs
- **Optimization:** Adam optimizer with learning rate 0.0001
- **Data handling:** Process 1,000 images at a time, shuffle them randomly
- **Training process:** Up to 5 complete passes through the data, stop early if I run out of requests, wait 60 seconds between batches

4. Saving and Submitting My Model

- Export my trained model to a standard format (ONNX) that can be tested
- Verify the model works correctly
- Submit it through the evaluation system

Results

- **Images used:** 30,000 out of 100,000 allowed
- **Training success:** The error steadily decreased, showing the model was learning
- **Final performance:** around 7 (scorecard server was down when I was about to write)
- **Ranking:** N/A

Things I Tried That Didn't Work

❌ Simple Neural Network Instead of ResNet

I first tried using a basic 3-layer network that just looked at flattened pixels. This didn't work because:

- It couldn't understand the spatial relationships in images
- The error stopped improving very quickly
- The final results were terrible (L2 error around 47)

❌ Using Raw Images

I also tried training without properly preparing the images first. This failed because:

- The original system expected processed images, so there was a mismatch
- My model learned unstable and wrong patterns
- The training was very inconsistent

My Files

- **stealing_encoder.py:** The main program that does everything
- **ModelStealingPub.pt:** The dataset of images I used for training

What Could Be Done Better

1. **Smarter image selection:** Instead of using random images, pick the most informative ones
2. **Use full budget:** Run the complete 100,000 queries for better results
3. **Try different models:** Test other architectures like Vision Transformers or EfficientNet
4. **Data augmentation:** Use techniques like rotating or cropping images to improve learning

---

Report by Aadrish Mahmood