TML Assignment 2

Model Stealing

For this assignment, we were tasked with stealing a model of unknown architecture hidden behind an API,
which was also protected using B4B.

Data Accessible to the Adversary

- Trained encoder of unknown architecture hidden behind an API and protected using B4B
- Subset of encoder's training data

Approach Used

We used Data Augmentations and a Siamese Framework as mentioned in the lecture and combined it with an active learning approach to minimize the number of queries to the api and hence the impact of B4B. In the first round we sample 1000 random images from the dataset and send them to the api. Our encoder is now trained to minimize the distance between its embeddings and the output of the api. The trained model is then used to estimate the uncertainty for each datapoint using MC-Dropout. The next 1000 datapoints are then sampled based on the model uncertainty and the class distribution. This process is repeated until the max queries are reached or the complete dataset is labeled.

Other Ideas and Implementation Details

For 13000 images we took original image representation twice considering there might be some changes because of B4B defense. Also three representations of the same image using three types of augmentation such as horizontal flip, 15 degree rotation, color jitter. As a result for each image we had 5 representations and then we trained our encoder in the following way:

X -> Encoder(X) ->2048->Projector-> output of 1024 dimension

Loss = mse(output, rep1)+cosine(output, rep1) +
mse(output, rep2)+cosine(output, rep2) +
mse(output, rep3)+cosine(output, rep3) +

mse(output, rep4)+cosine(output, rep4) +
mse(output, rep5)+cosine(output, rep5)

As encoder we used Resnet50. At the end of training, we compute L2 distances of some images between representations generated by this encoder and the API. The distance is approximately 0.001. However, after submitting the model to the server, the generated L2 distance is catastrophic approximately 36.

Which basemodel?

We tried different model types and found that Resnet18 resulted in the smallest average loss.
Resnet50 for example had a L2 distance of ~36.

How was the model trained ?

We train one model on the entire dataset.
The model is trained on the new 1000 embeddings and every 5 rounds on all labeled data to prevent "forget".

Which loss functions were used?

We combined two different loss functions.
At start of each training round we only consider MSE but then slowly introduce cosine embedding loss as well.
Using a contrastive loss function as well led to a worse performance and sometimes an increasing loss over the epochs.

Which Augmentations were used?

We experimented with multiple augmentations of different strengths, i. e., cropping, removing parts of the image, etc...
In end we only used small augmentations like flip and rotate as stronger ones activated B4B too quickly and resulted in a larger L2 distance.

Why only train the models for X epochs?

We observed that the model performance hardly changed after ~15-20 epochs during training so we stopped here to prevent overfitting.
When training on the full dataset we use 50 epochs.

Results

The above approach results in a L2 distance of 4.81.
Our solution ranked 4th on the leaderboard.

Observations

The cosine similarity of two unrelated sets on a fresh encoder was 0.975 which seems very high.
Often a model, which was not trained on the full data, outperformed the ones trained longer, which highlights the effect of B4B.

Files and Their Descriptions

attack.py - Main code for the model stealing
attack: runs the attack for n rounds, each round sending 1000 new images to the api. Returns the stolen model at the end

api.py - Handles all the code which interacts with the api.
new_api: Requests a new api/model. Returns TOKEN and SEED.
model_stealing: Sends 1000 images to the api. Returns the embeddings.
upload: Uploads the current stolen model for evaluation. Returns L2 distance to original model.
embeddings_sim: Measures the cosine similarity between 2 different subsets. Returns the average cosine similarity.

utils.py - This file includes different functions required to steal the model.
augment_image and augment_image_single: Augment batch of images and a single image according to AUGMENT and AUGMENT_SINGLE. Return the augmented version.
bayesian_predictions: Uses MC-Dropout to compute the uncertainty for each datapoint. Returns the mean std over the embedding.
predict_next: Uses bayesian_predictions and the class distribution to generate new candidates. Returns the next 1000 images to be sent to the api.
train: trains the model's output to be close to the real encoder's outputs. Returns the trained model.

model.py - The predicted model architecture is stored here.

data_sets.py - In here, the Dataset classes are stored, one for the original data and one for the stolen embeddings together with the predicted embeddings.

Dependencies

The following Python packages and modules are required to run the scripts:

- torch
- torchvision
- requests
- onnxruntime
- tqdm
- kornia
- timm
- scikit-learn

Custom modules

- src.utils
- src.dataset
- src.model
- src.api