TML Assignment # 3

TEAM 16

Ahrar Bin Aslam

Muhammad Mubeen Siddiqui

ROBUSTNESS

In this assignment, our goal was to train an image classification model that performs well not just on clean images, but also on images that have been slightly modified in a way that tricks the model. These modified images are known as adversarial examples. To the human eye, they look nearly identical to the original images, but they can cause even high-performing models to make incorrect predictions.

We focused on defending against two common types of adversarial attacks: Fast Gradient Sign Method and Projected Gradient Descent. The challenge was to build a model that can handle such attacks and still make accurate predictions.

Our Approach:

Dataset

The dataset was provided as a PyTorch .pt file containing a custom TaskDataset class. It consisted of RGB images with class labels ranging from 0 to 9. Each image was resized to 32×32 pixels and converted to a tensor using standard PyTorch transforms. The input format matched what our model expected three RGB channels with 32×32 resolution.

We deliberately avoided applying any additional image augmentation techniques like flipping or cropping. This decision was made to keep the training process consistent and ensure that the adversarial examples remained meaningful and interpretable throughout training.

Model

We used ResNet34 architecture from PyTorch's torchvision library. This was one of the allowed model choices for the assignment. We selected ResNet34 from the list of allowed models (resnet18, resnet34, resnet50) because it offers a solid trade-off between model complexity and computational efficiency. While ResNet18 is faster to train, it lacks the power needed for handling subtle adversarial perturbations. On the other hand, ResNet50 is deeper and but is computationally heavier, which made training harder. Moreover, we modified the final fully connected layer to output 10 values, one for each class in the dataset.

Training Process

We trained the model using the Adam optimizer with a learning rate of 0.001 and a batch size of 32 for 20 epochs. To make the model more robust, we used adversarial training.

In each training step, we included:

- The original (clean) image
- An FGSM-generated adversarial image

- A PGD-generated adversarial image

For each of these three versions, we calculated the cross-entropy loss and then averaged the three losses to get the final training loss. This helped the model learn to handle both clean and adversarial inputs during training.

FGSM was implemented as a simple one-step attack with a perturbation strength (epsilon) of 0.03. For PGD, we used 3 iterative steps with a step size of 0.01, while keeping the same epsilon value. After each step, the adversarial examples were clipped to ensure their pixel values stayed within valid image bounds and did not deviate too far from the original input. This configuration gave us a good balance between attack strength and training efficiency. While increasing the number of PGD steps could have improved the model's robustness further, it also significantly slowed down training. Given our computational limits on Google Colab, we chose a setting that was both effective for us.

Evaluation

We used only the training data for evaluation and did not set aside a separate validation set. Instead, we monitored performance after each epoch by checking how the model performed on clean, FGSM, and PGD images. Although a learning rate scheduler wasn't used, the training process remained stable throughout all 20 epochs.

Before submission, we ran the test scripts provided to verify that:

- The model's input shape was correct (3×32×32)
- The output shape matched the number of classes (10)
- A dummy forward pass ran successfully without errors

Final Results

After training, our model achieved the following results:

- Clean accuracy: 52.1%
- FGSM accuracy: 37.1%
- PGD accuracy: 36.8%

Additional Information

We faced a few key challenges while working on this task:

1. Balancing clean and adversarial accuracy
   One of the trickiest parts was finding a good balance. When we focused too much on adversarial training, clean accuracy would drop. When we prioritized clean accuracy, robustness suffered. Tuning this balance without a validation set required a lot of trial and error.
2. Limited PGD steps due to resource constraints
   Ideally, PGD should use more steps to generate stronger adversarial examples. However, doing this made training very slow, especially in Google Colab. We settled

on 3 PGD steps, which was enough to make the model more robust without exhausting Colab's resources.

3. No validation set
   Without a separate validation set, we couldn't objectively measure generalization during training. This made it harder to tune hyperparameters like learning rate or training duration.