# TML25_A3_12 Report

**Release Tag -** v1.0.0-TML25_A3_12

```
https://github.com/highgroundmaster/TML25_A3_12/releases/tag/v1.0.0-TML25_A3_12
```

| Name | ID |
|---|---|
| Rithvika Pervala | 7072443 |
| Bushra Ashfaque | 7070555 |

## Problem Statement

The goal was to train a **robust classifier** that achieves high accuracy on both **clean data** and **adversarial examples** generated via **FGSM** and **PGD** attacks. The challenge included balancing robustness against adversarial perturbations while maintaining >50% clean accuracy, adhering to strict submission requirements (e.g., input/output shapes, allowed model classes).

## Our Approach

Our solution focused on **adversarial training** with hybrid attack generation and parameter tuning. We also spilt the data into Training and Validation to gauge the Accuracy scores after every epoch of training.

### Hyperparamaters

- We use the following parameters after careful trail and error usage for the best accuracy

```
EPOCHS = 30 # number of epochs
LEARNING_RATE = 0.075
WEIGHT_DECAY = 1e-4
FGSM_EPSILON = 0.03 # FGSM Attack strength
PGD_EPSILON = 0.03 # PGD Attack Strength
ALPHA = 0.007   # PGD step size
PGD_ITER = 5   # Number of Interattions in PGD
RESNET_MODEL = "resnet18"  # Resnet Model settings
```

### Hybrid FGSM + PGD Training

`adversarial_train_epoch(model, train_loader, optimizer, device, fgsm, pgd)`

- A function that train the given resnet model `model` for one epoch on the loaded FGSM and PGD Torchattack Objects - `fgsm` and `pgd` .

- In this function we combine the clean images from the `train_loader` with adversarial generated imaged from FGSM **FGSM** (single-step, fast attack) and **PGD** (multi-step, stronger attack) - thus creating a 2:1 ratio of newly generated adversarial data to clean train data.
- We then train on this augmented data and predict the Loss and Training accuracy in the function.

## Evaluation

### `evaluate_robustness(model, val_loader, device, attack=None)`

- Through this function we also validate the trained model per epoch on either Clean Accuracy, FGSM Accuracy or PGD Accuracy on the validation data `val_loader` .
- The argument `attack` takes in which Robustness Accuracy to calculate
    - `None` - Clean Accuracy
    - `fgsm` Torchattack Object = FGSM Accuracy
    - `pgd` Torchattack Object = PGD Accuracy
- The validation helps in accurately generalizing the model to server data - giving a clear idea on the direction the training is going.

## Complete Training and Evaluation

### `train_and_evaluate()`

- In this function we present the proper blueprint for training the resnet model - with important parameters and hyperparameters initialized
- FGSM and PGD Torchattack modules are instantiated based on the hyperparameters `FGSM_EPSILON=0.03` , `PGD_EPSILON=0.03` , `ALPHA=0.07` , `PGD_ITER=5` .
    - These values ensured strong perturbations without destabilizing training.
    - We chose a lower PGD Iteratations `PGD_ITER=5` due to computational reasons - higher number is resulting in higher amount of epoch time leading to Colab time runouts.
- **Loss Function** - we use **Cross Entropy Loss**
- **Optimizer** - **SGD** is used with a weight decay of `WEIGHT_DECAY=1e-4`
- **Scheduler -** A **Step Scheduler** is used to regular the Learning Rate as the epochs progress to stablize the training an find the global optimums.
    - `step_size=6` . `gamma=0.5`
- Once the needed parameters are instantiate, we run the following per epoch
    - `adversarial_train_epoch`
    - `evaluate_robustness(None)`
    - `evaluate_robustness(fgsm)`
    - `evaluate_robustness(pgd)`

- **Checkpoint -** A checkpoint is created when there is a new best clean accuracy is found in any epoch - the model state is save accordingly.

---

# Research Questions Explored

### Scheduler Choice

Why chose a Step Scheduler over other methods like Cosine Annealing?

- We choose a Step Scheduler over Cosine Annealing Scheduler as we've observed, the rapid decrease in the learning rate cause the model to not cover the broad terrain before it could focus on the local regions leading to losses in accuracies.
- A steady slow decrease with a `gamma=0.5` for every 6 epochs helped with generalizing over broad terrains at higher LRs for 6 epochs and then focusing on the local optimums at Lower LRs for 6 epochs - creating a good balance of finding the global optimum.

### Impact of Attack Strength (Epsilon)

How does varying `eps` affect robustness and convergence?

- **Insight**: `eps=0.03` (≈8/255) balanced perturbation strength and training stability. Smaller values slowed robustness gains, while larger values destabilized learning.

1. **Effectiveness of Hybrid FGSM + PGD Training**

   Does training on both FGSM and PGD improve generalization compared to single-attack training?

   - **Insight**: Hybrid training improved robustness against FGSM but failed against PGD, suggesting potential gradient masking or overfitting to FGSM.

### Role of Weight Decay in Robustness

Can regularization mitigate overfitting tendencies in adversarial training?

- **Insight**: Adding `weight_decay=1e-4` stabilized training and slightly improved clean accuracy by penalizing large weights.

---

# Results

- The validation scores are pretty accurate and generalize the server scores. Hence we can submit our model states based on the clean and adversarial accuracies to our likings. Like below

### Epoch 19 - `resnet18_0.09_61.05_19.pt`

- After the function `train_and_evaluate()` saved the Best Clean Accuracy at epoch 19 with the following validation results

  ```
  Epoch 19/30
  New best clean accuracy: 61.05%
  Train Loss: 1.0604 | Train Acc: 60.95%
  ```

```
Clean Acc: 61.05% | FGSM Acc: 85.44% | PGD Acc: 35.39%
Current LR: 0.011250
```

- The associated model state `resnet18_0.09_61.05_19.pt` is saved and submitted to the server giving the below results.

```
{
    'clean_accuracy': 0.619,
    'fgsm_accuracy': 0.863,
    'pgd_accuracy': 0.33833333333333333
}
```

### Epoch 23 - `resnet18_0.09_61.34_23.pt`

- After the `train_and_evaluate()` function ran for 23 epochs, it gave the best clean accuracy overall with the following validation results

```
Epoch 23/30
New best clean accuracy: 61.34%
Train Loss: 1.0013 | Train Acc: 63.74%
Clean Acc: 61.34% | FGSM Acc: 85.93% | PGD Acc: 32.50%
Current LR: 0.011250
```

- The server results for the corresponding model state `resnet18_0.09_61.34_23.pt` is as follows

```
{
    'clean_accuracy': 0.63,
    'fgsm_accuracy': 0.879,
    'pgd_accuracy': 0.30933333333333335
}
```

  - It shows that while the Clean and FGSM Accuracy have improved, the PGD accuracy decreased

### Final Choice - `resnet18_0.09_61.05_19.pt`

- We decided to go with **Epoch 19 Model** `resnet18_0.09_61.05_19.pt` as it had a stronger PGD attack compared to **Epoch 23 Model.** We preferred a more balanced model.

1. **Clean Accuracy (61.9%)**

   Achieved by balancing adversarial examples with clean data in each batch. This met the submission requirement but left room for improvement.

2. **FGSM Accuracy (86.3%)**

   Surprisingly high, likely due to:

   - **Gradient Masking**: The model might have learned to obscure gradients, making FGSM ineffective.

- **Overfitting**: Training on FGSM examples may have overemphasized superficial robustness.

3. **PGD Accuracy (33.83%)**

   Low compared to FGSM, indicating failure to generalize robustness. Possible causes:

   - **Weak PGD Parameters**: The chosen `PGD_ITER=5` may not have generated sufficiently strong perturbations. But we had to use it due to computational limitations.
   - **Gradient Obfuscation**: The model's robustness to FGSM could be an illusion, as PGD bypasses gradient masking.

---

## Conclusion

Our solution prioritized practical adversarial robustness but faced critical limitations:

- **Unexpected FGSM Performance**: High FGSM accuracy likely stems from gradient masking, not genuine robustness.
- **PGD Vulnerability**: Low PGD accuracy as compared to FGSM highlights the need for stronger attack parameters like higher PGD Iterations or architectural changes.

Future work includes exploring self-supervised pretraining and dynamic loss weighting to narrow the robustness-clean accuracy gap. Despite partial success, the results underscore the complexity of adversarial robustness and the importance of avoiding gradient obfuscation.

---