# TML Assignment 3

## Robustness

For this assignment, we were tasked with protecting a ResNet-X model against adversarial attacks,
while still keeping a relatively high clean accuracy.

## Data Accessible to the Defender

- Labeled training dataset
- Model architecture which can be used

## Approaches Used

We used standard adversarial training to defend our model against the adversary.
Our design choices will be explained here:

### Why Use Model X?

We picked ResNet-18 out of the three available models, as we saw in different papers that the depth of the model only plays a minor role,
and ResNet-18 was the one we could experiment the most with due to the smaller size and hence faster training.

### Why Only Train the Model for X Epochs?

Our best-performing model across all three tests was trained for 5–7 epochs.
Even though the loss decreased further, the test accuracy did not change much, so we stopped the training early.
We also observed that the clean loss dropped too fast beyond a certain point, making lots of models unusable.

### Why Choose X to Generate Adversarial Samples?

At the start, we used "weaker" methods like FGSM and R+FGSM, but we quickly noticed the low performance in the PGD setting,
so we switched to experimenting with PGD. Due to time constraints, we did not experiment with random start and least likely class yet for PGD.

### Which Loss Was Used?

We tried using only the adversarial samples, but this reduced the clean accuracy too much.
As a result, we used both the clean and adversarial loss to balance out the accuracies.

We also tried scaling the adversarial loss by a factor $< 1$ to reduce the effect and increase the clean accuracy, but the effect was negligible.
Our base loss was CE as the task was classification-based.

## Which Epsilon, ... Was Used?

We tried out different values for epsilon until we realized that since we scaled the image by /255,
we also need to scale the epsilon as well.
Alpha was set to a small value to not overshoot each step; at eps/4, which was larger than eps/steps with steps = 10 we had at first.
Next, we tried—similarly to how the timestep in DDPMs is sampled—to randomly select our epsilon based on a normal distribution centered around 8/255. However, experiments have shown that a fixed epsilon performs better, which does not make sense for us in that context.

## Results

We achieved a clean accuracy of 55%, FGSM of 35.6%, and PGD of 17%.
At the end, we sacrificed ~15% PGD accuracy to increase our clean accuracy by ~5% to safely be above 50%.
We also trained another model using R+FGSM which had a clean score of ~55% and FGSM score of ~65%, but the PGD was ~0%, so we scrapped it.

## Observations

When performing adversarial training, we could not get the clean model accuracy to go beyond ~55% while keeping the rest high as well.
It was really hard for us to even get a small value for PGD at the start. FGSM, however, was easily pushed to be larger than the clean accuracy when ignoring PGD.
Training using other setups (i.e., changing the epsilon randomly, etc.) led to a much better local accuracy but to a very bad one on the server. This confused us as we here saw results comaprable to the papaers, i.e. ~80% clean accuracy and 60-70% PGD accuracy but when uploaded the PGD accuracy was $< 1$ % on the server. We assume that it could be the different epsilon used or some other mistake but are not sure how to interpret this. We also observed that training shorter led to a higher clean accuracy, and longer to a much higher adversarial accuracy but lower clean accuracy.
(5 epochs -> 55% clean and 17% adv, and 10 epochs -> 42% clean and 30% adv).
We really struggled with increasing all three at the same time. This stresses the difficulty of the task in practice,
as here the attack and hyperparameters of the attack are unknown.

## Other Ideas and Implementation Details

We tried generating the adversarial samples using a different clean model, but it did not work as well.
Tried out one epsilon per epoch, batch, and image.
Also tested out different loss functions for the PGD, but ended up using CE loss.

**Alternate Approach:**

In this approach, we divided our task into phases: i) create a clean model from clean images ii) fine-tune the clean model on perturbed images. The provided 100k dataset was divided into two parts: 95k for training, 5k for validation.

i) Create a clean model from clean images:
We first train our model on 95K clean images without perturbation and save the clean model based on accuracy on the validation set, i.e, 5k data. We train for 100 epochs. This is implemented in '**other_approach/model_for_clean_image.py'** script.

ii) fine-tuning the clean model on perturbed images:
Then all 100k images are perturbed using the *torchattack* library's *fgsm* and *pgd* classes. Therefore, from 100k clean images, we now have 100k FGSM adversarial images, 100k PGD adversarial images, along with 100k clean images total of 300k images. Now, using a batch size of 128, we perform batch mixing. We prepare every batch with clean images, fgsm, and pgd perturbed images in the following way:

- 50% clean images
- 25% FGSM perturbed images
- 25% PGD perturbed images

Now the clean model is again loaded and fine-tuned using this batch-mixing strategy. In order to avoid bias, within each batch, we perform shuffling so that the model does not see the same type of image for a long time. In this way, we train for 45 epochs and save the model on accuracy for the validation set, which is 15k, i.e, 5k clean, 5k FGSM perturbed, 5k PGD perturbed. This is implemented in '**other_approach/model_for_perturbed_image.py**' script.
**Models Used:** For the clean model, we used ResNet50, pre-trained on ImageNet. We replace the last FC layer with a 10-class fully connected layer.

**Results:** We have fine-tuned the clena model in different combination of FGSM **ε** (epsilon), PGD **ε**, ***α***(alpha), and number of iteration, i.e, fgsm **ε** = 0.01 to 0.05, PGD **ε** = 0.03 to 0.1. Only significant results that we obtained from server are provided in the table below.

| | Clean Accuracy | FGSM | PGD |
|---|---|---|---|
| FGSM **ε** = 0.03 | **0.6276** | **0.3926** | **0.0006** |
| PGD **ε** = 0.03, ***α*** = 0.00214, # of iteration = 14 | | | |
| FGSM **ε** = 0.05 | **0.6326** | **0.4046** | **0.001** |
| PGD **ε** = 0.01, ***α*** = 0.007, # of iteration = 14 | | | |

**Observations:**
We have above 60% clean accuracy, 39%-40% FGSM accuracy for both settings mentioned in the table. However,  the PGD accuracy is quite low. One reason can be the PGD attack with the value we chose for $\boldsymbol{\varepsilon}$, $\boldsymbol{\alpha}$ might be quite far from the PGD attack that the submision system is using. On the contrary our chosen parameter for FGSM might be closer to the submission system.