Adversarial Attacks

Team:
Attack_Of_Cifar

Submitted By:

Kanupriya Jain
Mohamed Ali
Anna Krysta

Contents

1 Introduction

In this project, we explore adversarial attacks and defense mechanisms in deep learning models, focusing on improving both clean accuracy and robustness. Initially, we implemented FGSM (Fast Gradient Sign Method) and PGD (Projected Gradient Descent) adversarial training to enhance the model's resilience to attacks. Building on this foundation, we introduced AutoAttack as a key metric to evaluate the model's robustness. To further improve the accuracy-robustness trade-off, we incorporated MixedNUTS, a training-free approach that combines standard and robust classifiers through nonlinearly mixed outputs.

2 AutoAttack accuracy with adversarial training

In this section, we provide details on adversarial training using FGSM and PGD attacks (the code is provided in the `autoattack_models_final.ipynb` file).

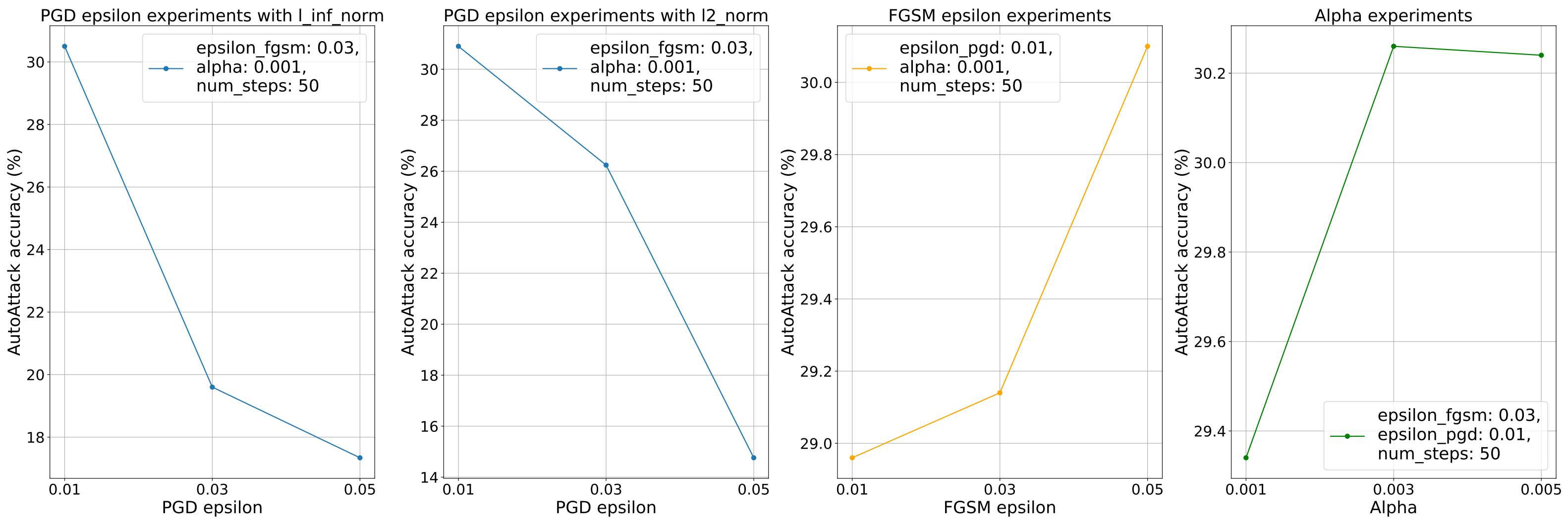


Figure 1: AutoAttack accuracy for different adversarial training configurations after 10 epochs.

We trained and evaluated our neural network on the CIFAR-10 dataset with adversarial training under multiple configurations of Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD). The configurations included variations in the following parameters:

- **Epsilon for FGSM and PGD:** maximum allowable perturbation magnitude for the entire adversarial perturbation.
- **Alpha:** maximum allowable perturbation per update step in PGD attacks, used to clamp updates within the range $[-\alpha, \alpha]$.
- **Number of Steps:** number of iterations for PGD attacks.

Each configuration incorporated adversarial examples generated by clean samples, FGSM, and PGD attacks during training. The AutoAttack accuracy of the trained models was reported after 10 epochs.

Robustness Evaluation with AutoAttack

Robustness evaluation using AutoAttack, as shown in Figure 1, provided further insights. For the PGD experiment with the L_∞ norm and the L_2 norm, the best AutoAttack accuracy was achieved for $\epsilon = 0.01$, suggesting that a low value of ϵ is the optimal choice. The accuracy was slightly better for the L_2 norm compared to the L_∞ norm, indicating that the L_2 norm may offer better resilience to adversarial perturbations. In the second FGSM experiment, the best ϵ value was, in fact, the highest: $\epsilon = 0.05$. This trend suggests that the FGSM attack may perform more effectively when larger perturbations are permitted, improving the training model's ability to generalize to unseen adversarial examples. For the third experiment, the best-performing α value was the moderate value of $\alpha = 0.003$, with AutoAttack accuracy decreasing when α was either higher or lower than 0.003. This observation indicates the importance of balancing the step size in PGD attacks to maximize the model's robustness without overfitting to specific adversarial patterns. For the final best configuration, with PGD epsilon $= 0.01$, FGSM epsilon $= 0.05$, and $\alpha = 0.003$, the model was trained over 30 epochs, achieving an AutoAttack accuracy of **41.02%**. In this configuration, the L_2 norm was utilized.

Discussion

The results indicate that combining FGSM and PGD adversarial examples during training enhances both validation accuracy and robustness. Achieving optimal performance requires a balanced selection of FGSM and PGD parameters, such as setting PGD $\epsilon = 0.01$, FGSM $\epsilon = 0.05$, and using approximately 50 steps. In contrast, excessively large perturbation magnitudes or insufficient training steps diminished both robustness and generalization.

For FGSM, lower ϵ values performed poorly because they did not provide enough perturbation to simulate strong adversarial attacks, leading to less effective adversarial training. On the other hand, excessively large ϵ values for PGD attacks resulted in unrealistic perturbations that may cause the model to overfit to adversarial examples, compromising its performance on clean data.

These findings provide valuable guidance for designing robust and generalizable adversarial training frameworks. By selecting appropriate combinations of attack parameters, adversarial training can significantly enhance model robustness without sacrificing accuracy on clean data.

3 MixedNUTS: Training-Free Accuracy-Robustness Balance via Nonlinearly Mixed Classifiers

We implemented the MixedNUTS method from [1], a training-free approach that balances clean accuracy and robustness. It combines outputs from a standard classifier (optimized for clean data) and a robust classifier (resilient to adversarial attacks).

3.1 Motivation and Core Idea

MixedNUTS leverages ensemble methods without retraining base models. The mixed classifier considers ***heterogeneous mixing***, with one base classifier specializing in the benign attack-free scenario and the other focusing on adversarial robustness. The method leverages the "benign confidence property," where robust classifiers are more confident in correct predictions. Nonlinear transformations amplify these margins while combining outputs from robust and standard classifiers.

Here, the base classifiers specialize in different data and the mixed classifier combines their advantages. Hence, we focus on the two-model setting, where the role of each model is clearly defined.

3.2 Workflow

Consider g_{std} and h_{rob} which are standard classifier trained for high clean accuracy and robust classifier respectively defined from $\mathbb{R}^d \to \mathbb{R}^c$. Here d is the input dimension and c is the number of classes. We can denote the proposed mixed model with $f_{\text{mix}} : \mathbb{R}^d \to \mathbb{R}^c$. The i^{th} output logit of the mixed model follows the formulation,

$$f_{\text{mix,i}}(x) = \log((1-\alpha)\sigma \circ g_{\text{std,i}}(x) + \alpha \ \sigma \circ h_{\text{rob}}(x))$$

where $\alpha \in [1/2, 1]$

While working with MixedNUTS model, we have A standard classifier outputs logits focusing on clean accuracy and a robust classifier outputs logits emphasizing adversarial robustness.Then we apply a transformation to the robust classifier logits to emphasize its confidence in correct predictions while mitigating overconfidence in incorrect ones and we call this mixture $f^M_{\text{mix}}(\cdot)$ where M is the transformation.

Formally, this goal is described as the optimization problem

$$\max_{M \in \mathcal{M}, \alpha \in [1/2,1]} \mathbb{P}_{(X,Y)\sim\mathcal{D}} \left[\arg\max_i f^M_{\text{mix},i}(X) = Y\right] \tag{1}$$

subject to

$$\mathbb{P}_{(X,Y)\sim\mathcal{D}} \left[\arg\max_i f^M_{\text{mix},i}(X + \delta^*_{f^*_{\text{mix}}}(X)) = Y\right] \geq r_{f^M_{\text{mix}}}, \tag{2}$$

where $\mathcal{D}$ is the distribution of data-label pairs, $r_{f^M_{\text{mix}}}$ is the desired robust accuracy of $f^M_{\text{mix}}(\cdot)$, and $\delta^*_{f^*_{\text{mix}}}(x)$ is the minimum-margin perturbation of $f^M_{\text{mix}}(\cdot)$ at x. Note that $f^M_{\text{mix},i}(\cdot)$ implicitly depends on M and α.

M is achieved via:

- **Layer Normalization (LN):** Standardizes logits to have zero mean and unit variance.
- **Clamping and Exponentiation:** Enhances margins between correct and incorrect predictions.

M is parametrized by $s \in (0, +\infty)$, a scaling constant, $p \in (0, +\infty)$, an exponent constant, and $c \in \mathbb{R}$, a bias constant that adjusts the cutoff location of the clamping function. In the next section, we will see if and how we can find s,p,c and alpha.

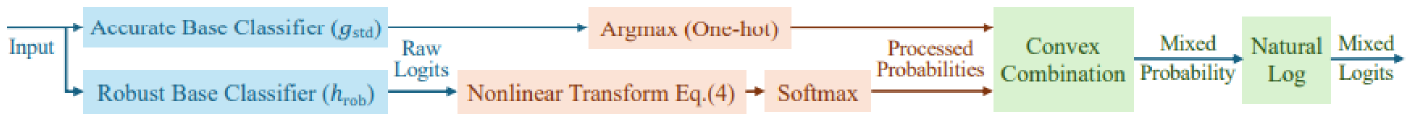


Figure 2: Overview Of MixedNUTS

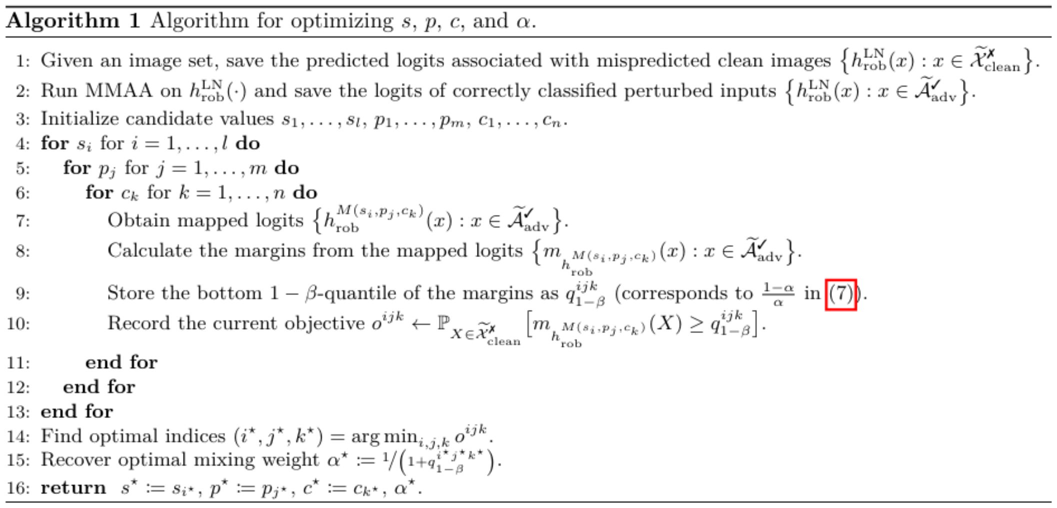
4 Optimization

Given two assumptions—(1) M preserves the order of the classes, and (2) if the robust model correctly classifies clean data, the standard model does so as well—the original objective can be reformulated as minimizing the misclassification probability on clean examples while imposing a robust accuracy constraint on adversarial examples.

$$\min_{M\in\mathcal{M},\,\alpha\in[1/2,1]} \mathbb{P}_{X\sim\mathcal{X}^{\times}_{\text{clean}}}\left[m_{h^M_{\text{rob}}}(X) \geq \frac{1-\alpha}{\alpha}\right] \quad \text{subject to} \quad \mathbb{P}_{Z\sim\mathcal{X}^{\checkmark}_{\text{adv}}}\left[m^{\star}_{h^M_{\text{rob}}}(Z) \geq \frac{1-\alpha}{\alpha}\right] \geq \beta,$$

Here, $\beta \in [0,1]$ controls the desired level of robust accuracy relative to h_{rob}, and the formulation avoids dependency on the standard base classifier g_{std}, enabling the replacement of g_{std} without redesigning transformations.

With this characterization, for a fixed β, we can find s,p,c and alpha with grid search on a subset of the training data set. Like presented in Figure 3. An implementation of this algorithm is showcased in section 5.



Algorithm 1 Algorithm for optimizing s, p, c, and α.

```
1: Given an image set, save the predicted logits associated with mispredicted clean images $\{h^{\text{LN}}_{\text{rob}}(x) : x \in \widetilde{\mathcal{X}}^{\times}_{\text{clean}}\}$.
2: Run MMAA on $h^{\text{LN}}_{\text{rob}}(\cdot)$ and save the logits of correctly classified perturbed inputs $\{h^{\text{LN}}_{\text{rob}}(x) : x \in \widetilde{\mathcal{A}}^{\checkmark}_{\text{adv}}\}$.
3: Initialize candidate values $s_1, \ldots, s_l$, $p_1, \ldots, p_m$, $c_1, \ldots, c_n$.
4: for $s_i$ for $i = 1, \ldots, l$ do
5:     for $p_j$ for $j = 1, \ldots, m$ do
6:         for $c_k$ for $k = 1, \ldots, n$ do
7:             Obtain mapped logits $\{h^{M(s_i,p_j,c_k)}_{\text{rob}}(x) : x \in \widetilde{\mathcal{A}}^{\checkmark}_{\text{adv}}\}$.
8:             Calculate the margins from the mapped logits $\{m_{h^{M(s_i,p_j,c_k)}_{\text{rob}}}(x) : x \in \widetilde{\mathcal{A}}^{\checkmark}_{\text{adv}}\}$.
9:             Store the bottom $1-\beta$-quantile of the margins as $q^{ijk}_{1-\beta}$ (corresponds to $\frac{1-\alpha}{\alpha}$ in (7)).
10:            Record the current objective $o^{ijk} \leftarrow \mathbb{P}_{X\in\widetilde{\mathcal{X}}^{\times}_{\text{clean}}}\left[m_{h^{M(s_i,p_j,c_k)}_{\text{rob}}}(X) \geq q^{ijk}_{1-\beta}\right]$.
11:        end for
12:    end for
13: end for
14: Find optimal indices $(i^\star, j^\star, k^\star) = \arg\min_{i,j,k} o^{ijk}$.
15: Recover optimal mixing weight $\alpha^\star := 1/\left(1+q^{i^\star j^\star k^\star}_{1-\beta}\right)$.
16: return $s^\star := s_{i^\star}$, $p^\star := p_{j^\star}$, $c^\star := c_{k^\star}$, $\alpha^\star$.
```

Figure 3: Paramters Grid Search

5 Experiments:

The authors observe that h_{rob} exhibits higher confidence margins for correct predictions, even under attack. When mixing the outputs $\sigma \circ h_{\text{rob}}(\cdot)$ and $\sigma \circ g_{\text{std}}(\cdot)$ on clean data, g_{std} can correct h_{rob}'s mistakes due to its higher accuracy. Under attack, h_{rob}'s higher confidence in correct predictions compensates for g_{std}'s mistakes, ensuring the mixed classifier retains robustness. This "benign confidence property" allows the mixed classifier to leverage both clean accuracy from g_{std} and adversarial robustness from h_{rob}.

By introducing nonlinearities, we can treat low-confidence and high-confidence examples differently, significantly amplifying h_{rob}'s benign property and thereby considerably enhancing the mixed classifier's accuracy-robustness balance. To verify this, we experimented with two pretrained ResNet-20 models for clean [2] trained on CIFAR-10 and Resnet-18 robust classifiers [3] trained on PGD attacked CIFAR-10 images (code available in pretrained_resnet.py). Due to limited resources, we couldn't apply the transformation to our own trained models, and results were unsatisfactory due to insufficient training.

5.1 Effect of Non-linear Transformation

We used the clean model from [2] available on torch.hub and we took pretrained robust model from [3]. We used robust logits and separated into two cases. One with margin < 0.2 and one with margin > 0.8.For this purpose the values of parameters for the transformation are $s = 0.1$, $p = 5.5$, $c = 0.1$ The code for this experiment can be found in the jupyter notebook `Margin_dist.ipynb`.The results are as follows-

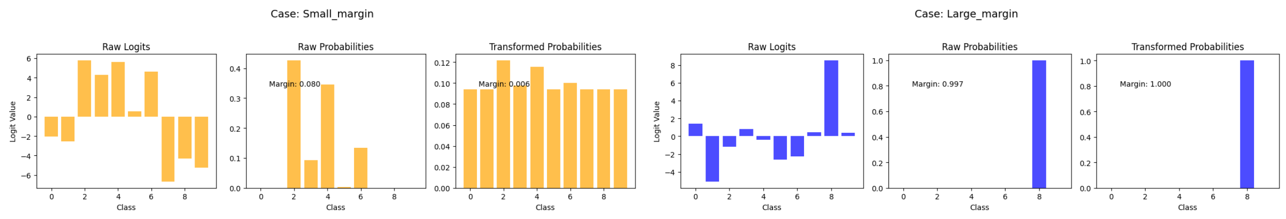


Figure 4: Effect of Non-Linear Transformation on Logits

Observations: Before the transformation, the margin (difference between the top two class probabilities) was low, indicating lower confidence in the prediction. After applying the transformation, the margin decreased slightly further, as the transformation de-emphasizes overconfidence in ambiguous predictions. Before the transformation, the margin was already high, indicating a confident prediction. After the transformation, the margin increased further, reinforcing the robust model's confidence in its correct predictions.

5.2 Margin Distribution

To further confirm the motivation, we experimented with margin distribution for clean and adversarial data using both clean and robust model. We used the same clean and robust models. We generated `PGDL2` adversarial images using `torchattacks` library. For the Non-linear Transformation, we used $s = 0.02$, $p = 8$, $s = 0.1$. The code for this experiment can be found in the jupyter notebook `Margin_dist.ipynb`. Here are the results -

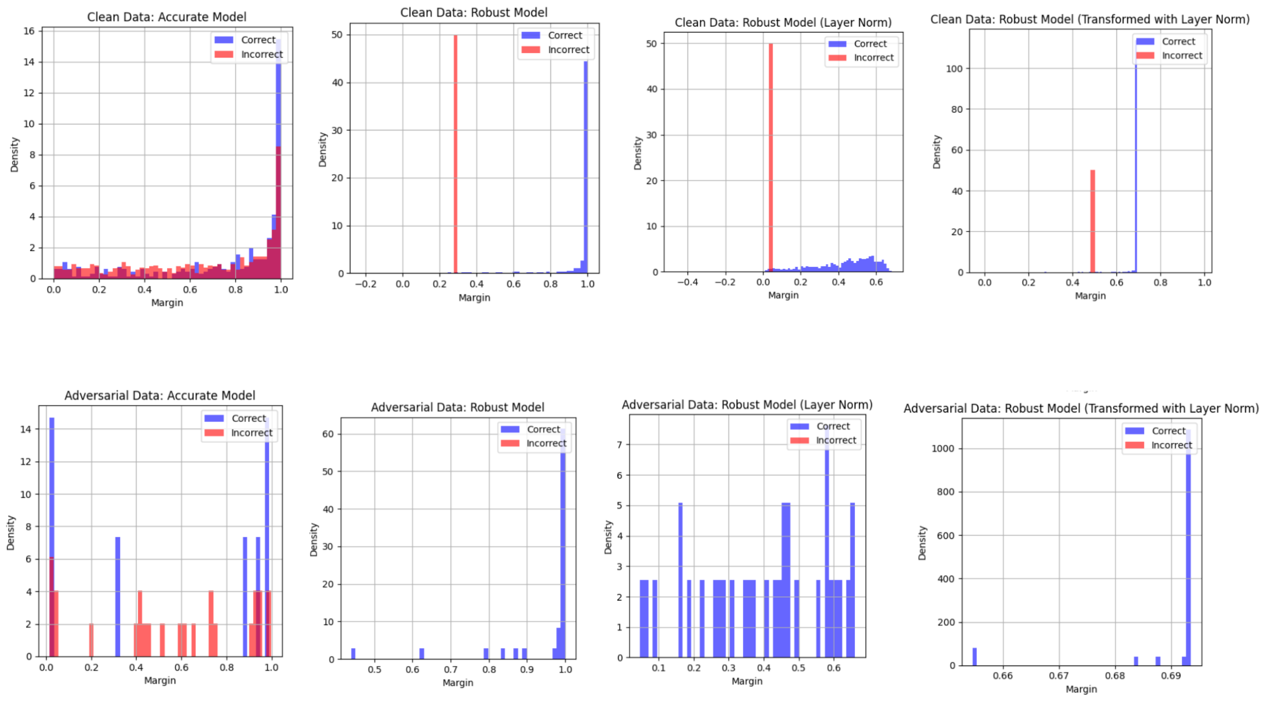


Figure 5: Margin Distribution before and after transformation

Observations:

- For the clean model, the margin distribution of correct predictions (blue) shows a concentration near 1 for clean data, indicating high confidence in correct classifications. However, for adversarial data, the margins for both correct and incorrect predictions are broadly distributed, with significant overlap. This lack of separation reflects the clean model's overconfidence and its vulnerability to adversarial attacks.

- The robust model shows strong performance on clean data, with high margins for correct predictions and low confidence for incorrect ones, though the density of correct predictions is lower than the clean model. After applying layer normalization and transformation, confidence in incorrect predictions increases slightly (0.3 to 0.45), while the density of correct predictions rises significantly (50 to 120). A similar pattern occurs for adversarial data, where normalization reduces confidence in incorrect predictions and smooths the distribution of correct ones.
- The robust model performs well with high confidence in correct predictions. After applying the nonlinear transformation, confidence slightly decreases, but the density of correct predictions increases significantly (from 60 to around 1000), indicating improved robustness and reliability.
- Margin distributions can be optimized by fine-tuning parameters s,p and c, and using advanced clean and robust models to improve performance in both clean and adversarial settings.

5.3 Grid Search Optimization

We can see in the figure below that for two values of β (in blue), we have a slightly better robustness without trading off accuracy. However we can see also the limits of optimizing using Grid search, because we can end-up missing an optimal value if we are not in the good range, or we are using a wide step, ending up performing worse in both accuracy and robustness. The code use to generate this plot and optimize the parameter is available in mixed_nuts_fixed_loop.py.

	Clean Acc	**Robust L2 Acc**
Std	0.81	0.38
Robust	0.50	0.50
Mixed Beta = 0.8	0.81	0.41
Mixed Beta = 0.7	0.81	0.38
Mixed Beta = 0	0.20	0.50

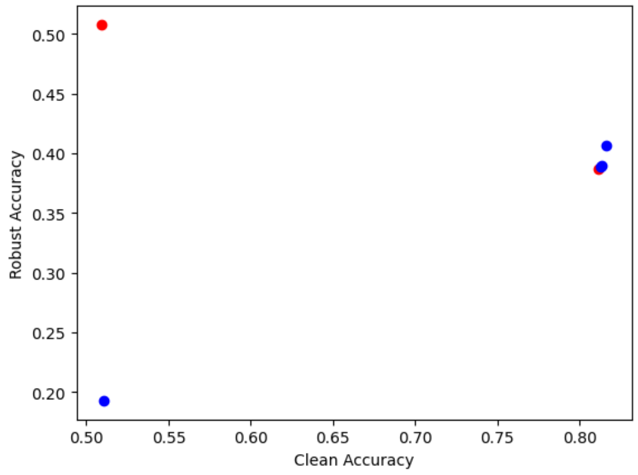


Figure 6: Mixed nuts result for different values of β(in blue), in comparison with the two original models (in red)

6 Conclusion

In conclusion, the MixedNuts model demonstrates significant advantages in terms of robustness, interpretability, and adaptability. One of the standout features of the MixedNuts framework is its training-free nature, allowing it to be easily integrated with any pre-trained clean or robust model without requiring additional training. Moreover, it has fewer parameters to tune, which simplifies deployment and reduces computational overhead compared to retraining robust models from scratch

7 Bibliography

[1] Yatong Bai et al. *MixedNUTS: Training-Free Accuracy-Robustness Balance via Nonlinearly Mixed Classifiers*. 2024. arXiv: 2402.02263 [cs.LG]. URL: https://arxiv.org/abs/2402.02263.

[2] *Pytorch CIFAR models*. URL: https://github.com/chenyaofo/pytorch-cifar-models/tree/master. (Date de consultation : 10/12/2024).

[3] *Pytorch-Adversarial-Training-CIFAR*. URL: https://github.com/ndb796/Pytorch-Adversarial-Training-CIFAR/tree/master. (Date de consultation : 10/12/2024).