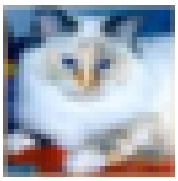
Adversarial Robustness through Non-Adversarial Training by Diversifying Vulnerabilities in a Model Mixture

MarsAttacks: Mathilde Kretz and Alexandre Ngau[1]

[1]PSL Research University - Data Science Project

December 2023



1 Introduction

Problem Statement: Enhancing Adversarial Robustness over the CIFAR10 Dataset through the Non-Adversarial Training of a Model Mixture by Diversifying Vulnerabilities using the DVERGE method [1].

Our Approach. In this project, we aim to enhance the adversarial robustness of a simple classification model over the CIFAR10 dataset by using the training procedure from the DVERGE [1] paper to form a model mixture with diverse and non-overlapping vulnerabilities. Initially, we refined the pretraining of several vanilla classifiers holding the same architecture to establish a solid baseline of independently trained models. Upon achieving the optimal settings, we then applied the DVERGE training procedure using the pretrained vanilla models. This approach ensured we started with optimal-performance vanilla classifiers for the DVERGE model mixture. The project focuses thoroughly on the DVERGE method as a whole, emphasizing improvements in performance and adversarial robustness from the DVERGE model mixture compared to the vanilla classifier.

2 DVERGE [1] : Diversifying Vulnerabilities for Enhanced Robust Generation of Ensembles – Theoretical Overview

In order to detail the method proposed in the paper [1], a few theoretical bases need to be defined.

2.1 Adversarial Attacks and Diversified Vulnerabilities

Adversarial Attacks: Crafting imperceptible input perturbations consistently inducing misclassification in models involves finding a bounded perturbation vector $\delta \in \mathbb{R}^d$ ($\|\delta\| \leq \varepsilon$). Applied strategically to $x + \delta$, the goal is to maximize the loss function $\ell(x + \delta, y)$ under the constraint $\|\delta\|$. Common norms include ℓ_2 and ℓ_∞.

Diversified Vulnerabilities. Unlike classical adversarial training, ensemble methods create submodels with diverse outputs against transfer adversarial examples, strengthening the ensemble's resistance to transfer attacks, even when individual submodels lack robustness. While adversarial training compels models to favor robust features, essentially eliminating vulnerability to specific attacks (see Figure 1(a)), the DVERGE method uniquely isolates adversarial vulnerabilities within each submodel by distilling and diversifying non-robust features. This isolation significantly reduces within-ensemble attack transferability, as illustrated from Figures 1(b) to 1(c). Moreover, the DVERGE method claims to markedly enhance overall ensemble robustness without compromising clean data classification accuracy – a notable achievement attributed to its training procedure based on clean data. This will be detailed in the following subsections.

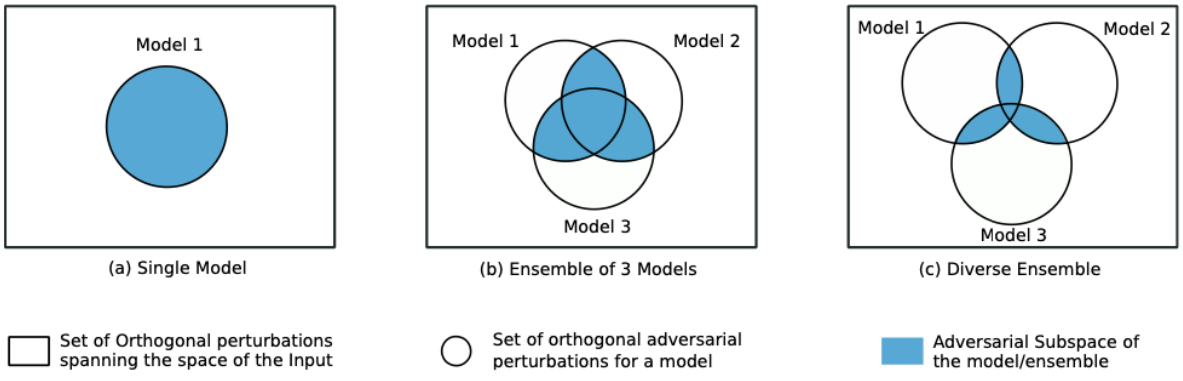


Figure 1:
Illustrating the Difference in Adversarial Robustness Between One Model and an Ensemble (extracted from [2])

2.2 Vulnerable Features and Vulnerabilities Diversity Metric

Distilled Feature. The vulnerable features are measured through a **distilled feature** defined for a model f_i, one of it's layers l, a source data point (x_s, y_s) and a target data point (x, y), as :

$$x'_{f_i^l}(x, x_s) = \arg\min_z \left\| f_i^l(z) - f_i^l(x) \right\|_2^2 \quad \text{s.t.} \quad \|z - x_s\|_\infty \leq \epsilon \tag{1}$$

This distilled feature x' resembles an image that looks like the source image x_s, and is classified as the target image x. This shows that x' reflects the adversarial vulnerability of f_i^l when classifying x. The higher its value, the more vulnerable the feature is because of it being highly misclassified.

Vulnerability Diversity Metric. Given two models f_i and f_j, their **vulnerability diversity metric** is defined as follows :

$$d(f_i, f_j) := \frac{1}{2}\mathbb{E}_{(x,y),(x_s,y_s),l}\left[\mathcal{L}_{f_i}(x'_{f_j^l}(x, x_s), y) \; + \mathcal{L}_{f_j}(x'_{f_i^l}(x, x_s), y)\right] \tag{2}$$

The intuition is that $x'_{f_i^l}(x, x_s)$ is a weak feature of f_j^l if $f_j^l(x'_{f_i^l})$ returns y (the target) rather than y_s (the source), and conversely, $x'_{f_j^l}(x, x_s)$ is a weak feature of f_i^l if $f_i^l(x'_{f_j^l})$ returns y rather than y_s. As a consequence, the cross-entropy loss $\mathcal{L}_{f_j}(x'_{f_i^l}(x, x_s), y)$ is small if f_j's vulnerability on x's non-robust features overlaps with that of f_i, and vice versa. In other words, the vulnerability diversity metric measures the vulnerability overlap between the two models.

2.3 Learning Objective

In theory. As stated above, the more the models are collectively weak (or vulnerable), the lower is the d metric. Maximizing it then naturally comes to be the objective function of the training model:

$$\min_{f_i} \mathbb{E}_{(x,y)}\left[\mathcal{L}_{f_i}(x, y)\right] - \alpha \sum_{j \neq i} d(f_i, f_j) \tag{3}$$

It can be noted that the algorithm seeks both to minimize the cross-entropy loss and to maximize the vulnerability diversity (by minimizing its opposite). The paper however highlights that as y may not be bounded, the training may possibly diverge, and proposes an alternative formulation.

In practice. The learning objective being to prevent f_i^l's weak feature to be shared with f_j^l, the paper claims that this is equivalent to force the weak feature of f_i^l to be a strong feature of f_j^l. This intuition provides the following reformulation of the learning objective, which will be used in practice in the implementation below:

$$\min_{f_i} \mathbb{E}_{(x,y)}\left[\mathcal{L}_{f_i}(x, y) + \alpha \sum_{j \neq i} \mathbb{E}_{(x_s,y_s),l}\left[\mathcal{L}_{f_i}(x'_{f_j^l}(x, x_s), y_s)\right]\right] \tag{4}$$

Mixture model. Trained this way, the resulting DVERGE submodels are individually non-robust, however their vulnerabilities are diversified and do not overlap accross the feature space, as shown in Figure 1. The subsequent DVERGE mixture model outputs the average probability of the DVERGE submodel outputs, which is adversarially robust while still being naturally accurate.

3 DVERGE : Diversifying Vulnerabilities for Enhanced Robust Generation of Ensembles – Implementation

The DVERGE model's training involves separately pretraining submodels as vanilla classifiers on a clean dataset to collect initial features. These submodels are then jointly trained following the DVERGE procedure to diversify weak features, as explained in the training objective (section 2.3).

3.1 Vanilla Classifier Pretraining

Before training the DVERGE model mixture, we wanted to perfect the vanilla classifier in order to start with the optimal performing baseline models we could get on clean data.

As stated in section 1, the vanilla classifier's architecture, fixed for this project, is instantiated in `model.py` under the "`Conv`" class. It comprises two convolutional layers, separated by max pooling, and followed by three fully connected layers.

To achieve our goal, we trained three vanilla classifiers for 100 epochs, plotting their training and validation losses. This number of vanilla classifiers comes from the paper [1] as a tarde-off between performance and computational cost of training. The aim was to find the optimal learning duration without overfitting, as Figure 2 demonstrates : after around 30 epochs, the models exhibit overfitting as validation losses increase.

The training procedure also made use of a stochastic gradient descent with a learning rate $\gamma = 1e-3$, and a momentum of 0.9. The former seemed to be the perfect match, while $\gamma = 1e-2$ and $\gamma = 1e-4$ were respectively too large and too small, and did not produce any interpretable results.

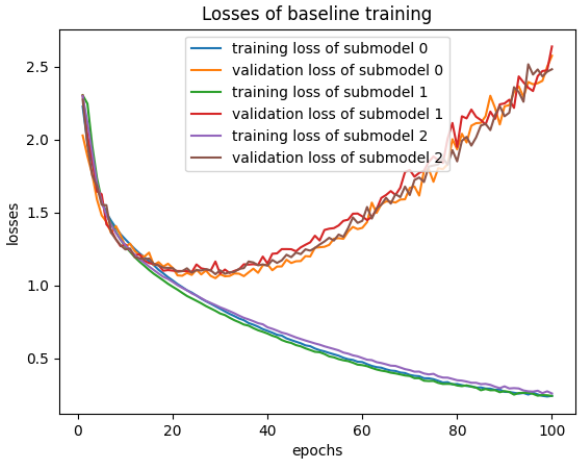


Figure 2:
Training and Validation Loss of Three Baseline Models as a Function of the Epochs

Therefore, we chose a vanilla classifier with the following training configuration : `epoch = 30`, $\gamma = 1e-3$, `momentum = 0.9, batch size = 32`. Table 1 presents results for three classifiers trained with this setup using `test_project.py`. Clean data accuracy is approximately 60%, which is quite good, while PGD L_2 Norm attacked data achieves a relatively good accuracy, around 45%. However, FGSM and PGD L_∞ Norm attacked data exhibit low accuracies, indicating the vanilla classifiers' vulnerability to adversarial attacks.

Note that these are 10-steps PGD attacks with parameters being $\epsilon = 0.03$ and $\alpha = 0.007$.

Vanilla Classifier	Clean Data (%) averaged on 10 tests	FGSM (%)	PGD L_{inf} Norm (%)	PGD L_2 Norm (%)
Vanilla Classifier 1 (30)	61.76	4.10	0.781	43.26
Vanilla Classifier 2 (30)	59.61	4.79	1.270	46.39
Vanilla Classifier 3 (30)	61.64	4.39	1.074	45.12

Table 1: Vanilla Classifiers Accuracy on Clean and Attacked Data
The number (.) represents the number of training epochs

The vanilla classifiers exhibit vulnerabilities to adversarial attacks despite good accuracy on clean data. Starting DVERGE training from epoch 30 of the baseline will reduce computation time while establishing a reference point for clean data classification. This will be further discussed in the next section.

3.2 DVERGE Training

3.2.1 The Training Algorithm

The DVERGE training algorithm (Algorithm 1) serves as the practical implementation of the DVERGE training procedure. It systematically extracts weak features (1) from a layer uniformly selected in a model at every epoch. Then, instead of conventional batches, the batch loader yields two independent pairs, (x, y) and (x_s, y_s), from the `train_loader` to compute weak features for each pretrained submodel at the designated layer. The computation of vulnerability diversity (4) takes place using these weak features within the `distillation.py` file. Following this, a gradient descent operation optimizes the reformulated objective (4) with respect to the features, facilitating the refinement of submodels to enhance mutual strong feature learning. The entire training procedure is encapsulated in the `dverge.py` file, incorporating specified hyperparameters for gradient descent ($\alpha = 18/255$) and distillation ($\epsilon = 8/255$, `steps`$= 10$), chosen for optimality as per [1].

The final DVERGE model is a mixture of models, as stated in section 2.3.

Algorithm 1 DVERGE training algorithm

Input: `vanilla pretrained submodels`
Output: `DVERGE trained submodels`

for $e \leftarrow 1$ to E **do** ▷ epochs iteration
  $l \sim U(1, L)$ ▷ uniformly choose one of the existing layers
  for $b \leftarrow 1$ to B **do** ▷ batch iteration
    $(x, y) \leftarrow$ batched input-label pairs
    $(x_s, y_s) \leftarrow$ batched source input-label pairs
    for $m \in$ `submodels` **do**
      $x\prime_m = x\prime_{f_m^l}(x, x_s)$ ▷ non robust feature 1
    end for
    for $n \in$ `submodels` **do**
      $\nabla_{f_m} \leftarrow \nabla \left[\sum_{n \neq m} \mathcal{L}_{f_m}(f_m(x'_n), y_s)\right]$ ▷ diversity loss 2 to compute the gradient with respect to features
      $f_m \leftarrow f_m - \alpha \cdot \nabla_{f_m}$
    end for
  end for
end for

3.2.2 The Results

Classifier	Clean Data (%) (averaged on 10 tests)	FGSM (%)	PGD L_{inf} Norm (%)	PGD L_2 Norm (%)
DVERGE Submodel 1 (200)	56.91	7.13	1.563	24.02
DVERGE Submodel 2 (200)	56.91	6.35	0.684	24.80
DVERGE Submodel 3 (200)	62.10	8.40	1.465	30.18
DVERGE Mixture Model (200)	67.38	34.67	10.55	44.82

Table 2: DVERGE Submodels and Mixture Model Accuracy on Clean and Attacked Data
The number (.) represents the number of training epochs

The DVERGE training followed the pretraining of the three vanilla classifiers for 30 epochs (section 3.1) and continued with the DVERGE method for an additional 200 epochs. The training losses, along with the validation loss of the model mixture, are depicted in Figure 3.

In Figure 3, the training loss curves exhibit noise attributed to SGD steps in the optimization process. Despite this, they asymptotically decrease, seemingly reaching a stalling minimum around the 200th epoch. The model mixture's validation loss shows a gradual increase, corresponding to the ensemble's slightly decreasing accuracy.

Therefore, we decided to select the DVERGE mixture model trained after 200 epochs as the final ensemble model. The table 2 above displays the results of three DVERGE trained submodels, and the final DVERGE Mixture Model. Despite the individual submodels experiencing a decrease in PGD L_2 attack accuracy and a stall in FGSM and PGD L_∞ attack accuracy, the DVERGE Model Mixture outperforms in all aspects, including clean data. Notably, it exhibits a 10-point accuracy increase in PGD L_∞ attacks, highlighting the effectiveness of the DVERGE training method in enhancing

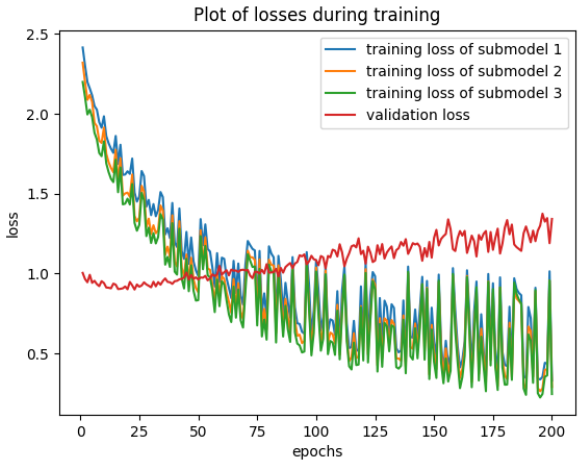


Figure 3:
Training Loss of the Three Submodels of the DVERGE Model Mixture, and Validation Loss of the DVERGE Model Mixture, as a Function of the Epochs

adversarial robustness.

It can be noted that as a downside, the DVERGE training method is very time consuming and computation costly : the 200 epochs took around 8 hours to be completed on a single NVIDIA RTX A6000 GPU.

4 DVERGE+AdvT : DVERGE Enhanced with Adversarial Training

Although the DVERGE training procedure produces a robust model mixture against transfer black-box attacks, the paper [1] claims that enhancing the DVERGE training procedure with adversarial training leads to an increase in the overall robustness while still maintaining a decent accuracy of classification on clean data. This is due to both the diversification of the weak features across the model mixture, and the learning of more robust features, model by model.

Therefore, the DVERGE+AdvT method incorporates adversarial training by including attacked images in the feature distillation process. To activate this training method, set the `adv` variable to `True` in the `main` function of the `dverge.py` file.

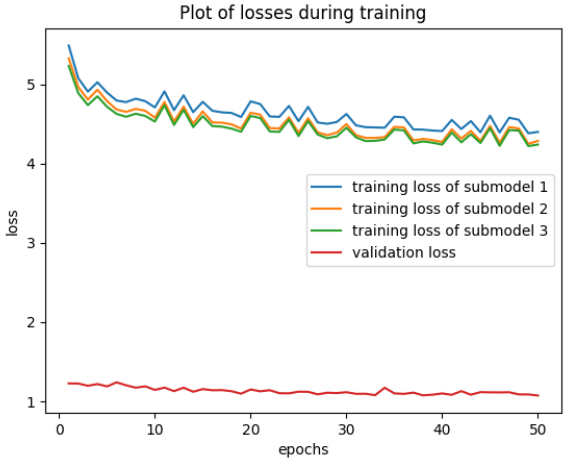


Figure 4:
Training Loss of the Three Submodels of the DVERGE+AdvT Model Mixture, and Validation Loss of the DVERGE+AdvT Model Mixture, as a Function of the Epochs

The table 3 below displays the test results. Clean data accuracy remains high, aligning with the claims in [1], with no decrease. Notably, accuracies on PGD L_∞ and PGD L_2 attacked data show over a 10-point increase compared to Table 2, while FGSM attacked data accuracy remains stable. The DVERGE+AdvT Submodels exhibit greater independent robustness compared to the DVERGE Submodels, emphasizing the efficacy of adversarial training in enhancing model and ensemble robustness. Furthermore, in Figure 4, the loss decrease is consistent, yielding better results from 25 to 50 epochs in Table 3 before hitting a stalling minimum around 40 epochs.

However, it can be noted that as a downside, the DVERGE+AdvT training method is very time consuming and computation costly : the 25 epochs took around 2 hours to be completed, and the 50 epochs around 4 hours, on a single NVIDIA RTX A6000 GPU.

Classifier	Clean Data (%) (averaged on 10 tests)	FGSM (%)	PGD L_{inf} Norm (%)	PGD L_2 Norm (%)
DVERGE+AdvT Submodel 1 (25)	58.36	24.02	19.92	52.05
DVERGE+AdvT Submodel 2 (25)	61.09	26.76	19.82	54.69
DVERGE+AdvT Submodel 3 (25)	64.06	26.76	19.73	55.57
DVERGE+AdvT Mixture Model (25)	63.87	31.54	25.98	57.03
DVERGE+AdvT Submodel 1 (50)	61.17	27.25	21.58	53.91
DVERGE+AdvT Submodel 2 (50)	62.03	27.54	20.51	56.35
DVERGE+AdvT Submodel 3 (50)	65.23	27.73	20.02	57.62
DVERGE+AdvT Mixture Model (50)	63.43	33.40	27.15	59.57

Table 3: DVERGE Submodels and Mixture Model Accuracy on Clean and Attacked Data
The number (.) represents the number of training epochs

5 Conclusion

Our project successfully enhanced the adversarial robustness of a simple classification model over the CIFAR10 dataset using the DVERGE method, a non-adversarial training procedure derived from vulnerability diversification. Applying the DVERGE training procedure efficiently isolated adversarial vulnerabilities and enhanced the overall ensemble robustness without compromising clean data accuracy. We then extended our investigation by incorporating adversarial training, showcasing further improvements in robustness. The trade-off between computational cost and increased robustness should however be considered when performing the latter method.

For future work, transferring the DVERGE and DVERGE+AdvT method to an adversarially trained classifier mixture, or a randomized model mixture could further enhance or validate some of our findings.

Acknowledgments

We would like to acknowledge the contribution of the codebase used in this project, which was inspired by the implementation available at `https://github.com/zjysteven/DVERGE`. The original code is associated with the work presented in the paper titled *DVERGE: Diversifying Vulnerabilities for Enhanced Robust Generation of Ensembles* by Yang et al. [1].

References

[1] H. Yang, J. Zhang, H. Dong, N. Inkawhich, A. Gardner, A. Touchet, W. Wilkes, H. Berry, and H. Li, "Dverge: Diversifying vulnerabilities for enhanced robust generation of ensembles," 2020.

[2] S. Kariyappa and M. K. Qureshi, "Improving adversarial robustness of ensembles with diversity training," 2019.