Transferability Reduced Smooth (TRS) Ensemble Training for Adversarial Transferability Attacks

PanicAttack: Thomas Boudras, Vivien Conti, and Jules Merigot[1]

[1]PSL Research University - Data Science Project

December 2023

1 Introduction

Problem Statement: How to mitigate transferability of adversarial attacks across different models by employing Transferability Reduced Smooth (TRS) ensemble training on the CIFAR-10 dataset.

Our Approach: This project addresses the challenge of adversarial attack transferability in machine learning, where deceptive inputs designed for one model can mislead others. Our aim is to investigate the Transferability Reduced Smooth (TRS) ensemble training method to reduce this transferability, thereby enhancing the robustness of our neural network models against such attacks. We first adversarially trained models on various attacks on the CIFAR-10 dataset to serve as a baseline for our later method. We then delved into the world of ensemble robustness.

2 Adversarial Training

The first step in our adversarial training approach was to select different attacks methods to leverage for our training. For this, we selected a Fast Gradient Sign Method (FGSM) attack, a Projected Gradient Descent (PGD) ℓ_2 norm attack, and finally a PGD l_∞ norm attack. FGSM creates adversarial examples by making small, directed changes to inputs based on loss gradients, offering a fast way to test model robustness. PGD with ℓ_2 norm is an iterative attack that adjusts inputs step-by-step within an ℓ_2 norm constraint (Euclidean distance). PGD with ℓ_∞ norm iteratively modifies inputs while keeping the maximum change in any component under a set limit.

Adversarial training is a method used to enhance the resilience of machine learning models against adversarial attacks. This involves integrating adversarial examples, like those generated by the PGD ℓ_∞ attack, into the training process. By training on a mix of original and adversarially modified images, the model learns to accurately classify both normal and manipulated inputs, which strengthens its defenses against adversarial attacks. The illustration below shows a set of points with a simple decision boundary that does not separate the ℓ_∞-balls (depicted as squares) around them, leading to misclassified adversarial examples (red stars). This makes the model susceptible to adversarial attacks. For robustness against ℓ_∞-norm attacks, a more complex decision boundary is needed for separating the ℓ_∞-balls. Such a robust classifier can be developed through adversarial training.

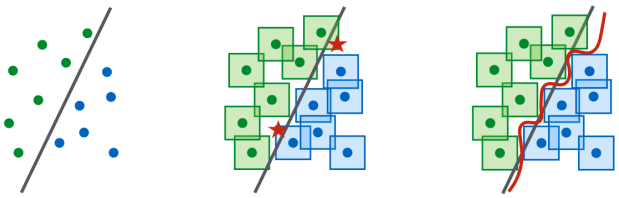

Figure 1: Illustration of standard vs. adversarial decision boundaries. [1]

Using the three methods mentioned previously, we adversarially trained three models with the provided architecture to improve their accuracies when adversarially attacked. We know that the PGD ℓ_∞ norm attack is the most effective against neural network models, so we focused on adversarially training our best model on this specific attack. For this, we implemented a `pgd_attack()` function that allowed us to attack input images within the ℓ_∞ norm constraints. During our training, we then attacked 30% of the input images and mixed them with our natural input images in order to bolster our model. To further improve our robustness against these adversarial attacks, we added `Dropout()` layers to our neural network model to introduce elements of randomness during training.

2.1 Hyper-parameter Tuning

For the PGD ℓ_∞ norm attack, our initial results, while good, were not up to our standards. We therefore decided to implement a dynamic ϵ and α, as well as a learning rate scheduler for our ADAM [2] optimizer in order to achieve better performance. We used a dynamic ϵ value that was set at $\epsilon = 0.03$ at the beginning of training and linearly increased to $\epsilon = 0.08$ as the epochs progressed. We did the same for our α value beginning at $\alpha = 0.01$ and ending at $\alpha = 0.03$. For our learning rate, we used a linear scheduler to decrease it over the epochs, starting at $\gamma = 0.001$ and decreasing by 50% every epoch, which caused a better descent in our model gradient loss.

The improved performance after fine-tuning is reflected in three main accuracy metrics that can be seen in the table below. These accuracies are the natural accuracy of our model, the PGD ℓ_∞ attack accuracy, and the PGD

ℓ_2 attack accuracy. Since the PGD ℓ_∞ attack tends to be the most effective, we chose to only fine-tune our model that was adversarially trained on ℓ_∞ attacks. This is the defensive method described in the previous section. The accuracy is calculated by dividing the number of correctly predicted images by the total amount of images, where an images is correctly predicted if the predicted label is the same as the actual label of the image.

Defensive Method	Natural Acc	PGD ℓ_∞ Acc	PGD ℓ_2 Acc
PGD ℓ_∞ Adversarial Training	41.25	37.33	41.71

Table 1: Best results of our defensive method on the testing platform.

Our adversarially trained PGD ℓ_∞ model performs relatively well on ℓ_∞ attacks as we can see by our prediction accuracy value of 37.33% on the data. This translates over as well to PGD ℓ_2 attacks with an accuracy of 41.71%. Unfortunately, our natural accuracy, which is the accuracy of our model on the data when not attacked, seems to suffer during our adversarial training. This is most likely due to slight overfitting duirng training. In the next section, we will discuss the new method we employed to boost our model accuracies.

3 Transferability Reduced Smooth (TRS) Ensemble Training

In this section will be explaining the method known as Transferability Reduced Smooth (TRS) proposed by Yang et al.[3], which we implemented in order to achieve better perfomance results. As described in article, TRS is an effective robust ensemble training approach to reduce the transferability among base models. We employed this on the ensemble of our three previous models for robustness.

As a reminder, even though we decided to mainly fine-tune and test our model trained on PGD ℓ_∞-norm attacks, the three models we adversarially trained are the following: **Model 1 :** Trained on FGSM attack,
Model 2 : Trained on PGD ℓ_2-norm attack, **Model 3 :** Trained on PGD ℓ_∞-norm attack

3.1 Ensemble Robustness via Transferability Minimization

Adversarial transferability refers to the phenomenon where adversarial examples used to deceive one neural network model are also effective at deceiving other models, even if those models have different architectures or were trained on different data. This implies that an attacker does not need specific knowledge about the target model to craft effective adversarial examples. Instead, examples can be created using a different model that they fully understand, and then applied to attack the target model.

To mitigate this adversarial transferability, the TRS method seeks enforce the smoothness of models in order to improve robustness, as well as reduce the loss gradient similarity between models in order to introduce global model orthogonality. This will make the ensemble of models within TRS more robust to transferable adversarial attacks. It is important to mention that both model smoothness and loss gradient similarity reduction msut be down in parallel for this method to be sucessful, which will be explained below.

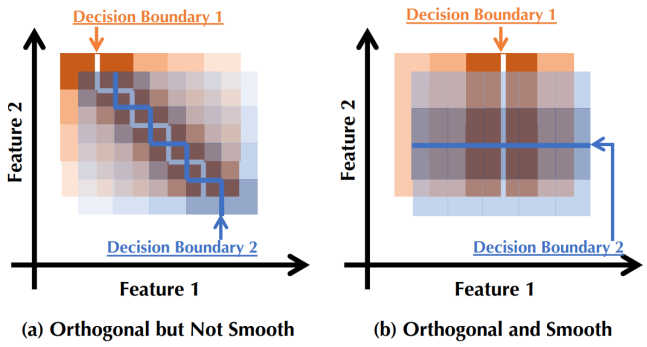

Figure 2: Relationship between adversarial transferability, gradient orthogonality, and model smoothness. [3]

In the illustration above, we can see that gradient orthogonality alone cannot minimize transferability as the decision boundaries between two classifiers can be arbitrarily close yet orthogonal almost everywhere. We must

include model smoothness with gradient orthogonality in order to provide a stronger guarantee on model diversity. Knowing this, we can move on to defining and explaining the loss functions that will allow us to perform the enforcments mentioned above.

4 TRS Implementation

For the following section detailing the various aspects of TRS and our implementations, we will be using two base models $\mathcal{F}$ and $\mathcal{G}$ as examples in the functions as done by Yang et al. [3] However, we will be adapting this method to our three previously defined models. Therefore, we will explain how to reduce adversarial transferability among base models $\mathcal{F}$ and $\mathcal{G}$ by enforcing model smoothness and low loss gradient similarity **at the same time**.

The following implementation can be found in the `TRS_training()` function in our `model.py` file. A crucial part of the TRS method lies in the calculation of the various loss functions. For this we will be defining loss functions during the explanation of our implementation. The main one being $\mathcal{L}_{\text{train}}$.

$$\mathcal{L}_{\text{train}} = \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_{\text{CE}}(\mathcal{F}_i(x), y) + \frac{2}{N(N-1)}\sum_{i=1}^{N}\sum_{j=i+1}^{N}\mathcal{L}_{\text{TRS}}(\mathcal{F}_i, \mathcal{F}_j, x, \delta) \tag{1}$$

This is a calculated loss that combines the Ensemble Cross-Entropy (ECE) loss of a model $\mathcal{F}_i$ and the TRS regularizer loss of any model pair, which will be explained after. This means that this could be adapted to an ensemble of as many models as desired since the TRS loss $\mathcal{L}_{\text{TRS}}$ will be calculated on a model $\mathcal{F}_i$ and the next model $\mathcal{F}_j$ at a time. In our case, we adapted this to the three models mentioned in the *TRS Ensemble Training* section.

The training process focuses on making each model accurate, via the ECE loss, and also on ensuring that the ensemble as a whole is robust and diverse in its learning approach, via the TRS regularizer. Implementing this loss calculation in our ensemble code makes our three model ensemble more robust by counteracting adversarial transferability. First, we computed the cross entropy loss $\mathcal{L}_{\text{CE}}$ in the classical way using `nn.CrossEntropyLoss()` from the PyTorch `nn` module. We used this to calculate the loss for our three models by iterating through them and saving their respective losses in a combined list.

4.1 TRS Regularizer

The proposed TRS Regularizer for the model pair $(\mathcal{F}, \mathcal{G})$ is defined as the loss function $\mathcal{L}_{\text{TRS}}$, which combines two loss functions that will be described in the next subsections along with their implementations in our code.

$$\mathcal{L}_{\text{TRS}}(\mathcal{F}, \mathcal{G}, x, \delta) = \lambda_a \cdot \mathcal{L}_{\text{sim}} + \lambda_b \cdot \mathcal{L}_{\text{smooth}} \tag{2}$$

The parameters λ_a and λ_b are weight balancing parameters that allow for varying the weight of the smoothness of the models and the orthogonality of loss gradient vectors. Overall, the $\mathcal{L}_{\text{TRS}}$ loss function allows for the increase of the robustness of models via **smoothness** from $\mathcal{L}_{\text{smooth}}$, and diversity in learning the patterns of the input data via **orthogonality** of the loss gradient vectors of the models from $\mathcal{L}_{\text{sim}}$.

4.2 Smoothing

We first employed regularization for model smoothness using the loss function $\mathcal{L}_{\text{smooth}}$. This will lead to model robustness because the models will be less susceptible to big changes in the input data, such as adversarial attacks.

$$\mathcal{L}_{\text{smooth}}(\mathcal{F}, \mathcal{G}, x, \delta) = \max_{\|\hat{x}-x\|_\infty \leq \delta} (\|\nabla_{\hat{x}}\ell_{\mathcal{F}}\|_2 + \|\nabla_{\hat{x}}\ell_{\mathcal{G}}\|_2) \tag{3}$$

In essence, $\mathcal{L}_{\text{smooth}}$ evaluates the combined sensitivity of two models to small changes in their inputs. By minimizing this function during training, the models are encouraged to have a more stable and predictable output, enhancing their robustness, especially against adversarial attacks that exploit model sensitivity to input perturbations.

To calculate the smoothed loss $\mathcal{L}_{\text{smooth}}$, we took the CIFAR-10 training dataset and divided it into several parts. One half of the images were left unattacked for the natural accuracy test, while the other half was divided evenly into three parts. Each part corresponds to a PGD l_∞-norm attack on those images for each of the three models. While the attack may not resemble the original purpose of the model, this is done purely in order to create the adversarial examples for each model in order to better train the ensemble. Finally, we recovered the loss gradients associated

with these new input images. All that remains is to take the sum of the norms of these gradients squared, which we put in magnitude form in order to create a more visual value. This smooth loss is defined under the `cos_loss` variable in our `TRS_trainer()` function.

4.3 Orthogonality

Next, we want to decrease the model loss gradient similarity by minimizing the cosine similarity between loss gradient vectors $\nabla_x \ell_{\mathcal{F}}$ and $\nabla_x \ell_{\mathcal{G}}$. In an optimal case, we want these loss gradient vectors to be orthogonal, which is done by leveraging the absolute value of their cosine similarity. We define loss function $\mathcal{L}_{\text{sim}}$ as the similarity loss between $\nabla_x \ell_{\mathcal{F}}$ and $\nabla_x \ell_{\mathcal{G}}$.

$$\mathcal{L}_{\text{sim}} = \left| \frac{(\nabla_x \ell_{\mathcal{F}})^\top (\nabla_x \ell_{\mathcal{G}})}{\|\nabla_x \ell_{\mathcal{F}}\|_2 \cdot \|\nabla_x \ell_{\mathcal{G}}\|_2} \right| \tag{4}$$

The optimal case of $\mathcal{L}_{\text{sim}}$ implies orthogonal loss gradient vectors, which ensures that the models learn different patterns from the data. This makes adversarial transferability more difficult since promoting diversity in learning allows each model to be less vulnerable to attacks used on other models. This makes the ensemble more robust.

For the loss of similarity, we applied the method described above in (4). We calculated the absolute value of the cosine of the gradients between each pair of models we have using our `Cosine()` function in the `model.py` file. Since we are using the three models described previously, we have a cosine similarity calculation for model pairs 1 and 2, one for pairs 2 and 3, and one for pairs 1 and 3. The similarity loss $\mathcal{L}_{\text{sim}}$ then corresponds to the average of these three cosines similarities. This is defined by the `cos_loss` variable in our `TRS_trainer()` function. This approach enables us to seek an organization that favors the most orthogonal loss gradients possible in the input data space.

4.4 TRS Training

All that's left is to add up the two weighted losses $\mathcal{L}_{\text{sim}}$ and $\mathcal{L}_{\text{smooth}}$, to get the loss $\mathcal{L}_{\text{TRS}}$ described in (2). For the choice of λ_a and λ_b, we tried several values and the one obtained by Yang et al. [3], i.e. $\lambda_a = 100$ and $\lambda_b = 2.5$, also gave us the best results for our implementation. The intuition behind this is to allow for the best balance between model smoothness and orthogonality in our loss $\mathcal{L}_{\text{TRS}}$ since we know that both are necessary for robustness against adversarial transferability. We then add the calculated TRS loss $\mathcal{L}_{\text{TRS}}$ to the aforementioned ensemble cross-entropy loss $\mathcal{L}_{\text{CE}}$ in order to get our final train loss $\mathcal{L}_{\text{train}}$ from (1), which is defined as `epoch_loss` in the `TRS_trainer()` function for our training loss, and then as `valid_loss` for our validation loss. This allows us to compute and print the loss after each epoch to view the progression of the convergence of our ensemble method.

At the very end of the training of our models, the architecture of the TRS ensemble method requires that we take a log softmax of each model, but we decided instead to take the average of the softmax of all three models, and then take the log of that average in order to get better results. This is done in the `forward()` function of our `Net` class. In the `Neural_Network` class we define the architecture of the three models that is then loaded into the `Net` class to begin the ensemble training.

5 TRS Experimentation & Performance

We trained the assembly model with different initial models. First, a training round was performed with untrained models, then a second with our models previously trained on the adversarial attacks mentioned in the *Adversarial Training* section. We noticed that the difference between the two training rounds was not massive, so we decided to train the ensemble starting from untrained models, and training the three models within our `TRS_training()` function in order to stay true to the original implementation from Yang et al. [3].

We deviated from the original plan because we noticed that when training our models with the TRS method, the whole model became very robust against PGD attacks with the l_2 norm, but ineffective against those with the l_∞ norm. We then decided to modify the code slightly and implement some adversarial training. Thus, the first loss, the cross-entropy loss, is calculated from inputs that are partly modified by a PGD attack with the l_∞ norm. We then observe slightly better results, especially on the first epochs. The best results are obtained with 50% data clean and the other attacked by PGD l_∞-norm.

First, we tried to train the TRS ensemble using pre-trained models. We used models that had received adversarial training with a PGD l_2 attack, another that has received adversarial training with PGD l_∞, and a final one that has been trained with FGSM attacks. However, we observed results that were not very encouraging due to our still relatively low natural accuracy, so we tried the TRS ensemble training method with untrained models.

Looking at the accuracy during training, we see two things. For l_∞-norm attacks, the model improves over the first 2 epochs and eventually loses its effectiveness over the course of training. For l_2-norm attacks, on the other hand, accuracy only increases over the epochs. We can also that the loss is decreasing as expected, which tells us that our training is working properly. We therefore had to make a compromise and chose a stopping epoch that gave us the best performing model for both types of attacks. After several tests on the testing plateform, our best results were obtained by choosing to shutdown our training after the tenth epoch (e = 10). This is the model which is pushed onto the "main" branch of our GitHub repository, and is represented as "TRS Ensemble Training 2" in the results table below.

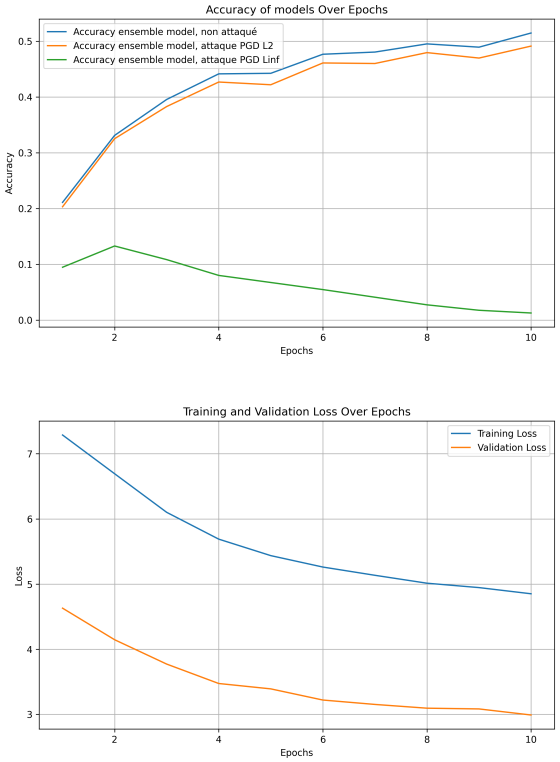

Figure 3: Accuracy plot and Losses plot for TRS ensemble of un-trained models

The final results pulled from the testing plateform can be seen in the table below, which includes one of the original models trained on PGD ℓ_∞-norm attacks with dynamic parameters, our first TRS ensemble with pre-trained models, and our better performing second TRS ensemble without pre-trained models. As we can see, our second TRS ensemble without pre-trained models has significantly better performance metrics, specifically a higher natural accuracy while still maintaining reasonable ℓ_∞-norm accuracy and ℓ_2-norm accuracy. The significant difference in the accuracy results between our generated plots and in the table is most likely due to how the attacks are performed on our local applications versus on the plateform. The platform is most likely calculating with a lower ϵ value, making the attack less potent and therefore being better defended by our TRS ensemble model.

Defensive Method	Natural Acc	PGD ℓ_∞ Acc	PGD ℓ_2 Acc
PGD ℓ_∞ Adversarial Training	41.25	37.33	41.71
TRS Ensemble Training 1	28.75	30.76	35.82
TRS Ensemble Training 2	62.50	22.05	39.27

Table 2: Final results of our defensive methods.

6 Conclusion

In this project, we tackled the issue of adversarial attack transferability in machine learning models, focusing specifically on the CIFAR-10 dataset. Our approach, employing the Transferability Reduced Smooth (TRS) ensemble training method, proved to be a significant stride towards mitigating the transferability of adversarial attacks. By establishing a baseline by adversarially training three models on various attacks, we understood the extent of vulnerability in regular training settings. Subsequently, by implementing the TRS ensemble training, we learned how we can enhance model robustness and effectively reduce the susceptibility to adversarial attacks originally designed for other models. Our results highlight the potential of ensemble training methods in building more secure and reliable neural network models, even though our final results were not as initially expected. This project opens avenues for future research on how to perfect our TRS implementation and achieve efficient model robustness.

References

[1] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," *MIT, arXiv:1706.06083 [stat.ML]*, 2019.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. arXiv:1412.6980 [cs.LG]*, 2015.

[3] Z. Yang1, L. Li1, X. Xu, S. Zuo, Q. Chen, B. Rubinstein, P. Zhou, C. Zhang, and B. Li, "Trs: Transferability reduced ensemble via promoting gradient diversity and model smoothness," *Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021). arXiv:2104.00671 [cs.LG]*, 2021.