Adversarial Attacks

Matthieu Neau Alexandros Kouvatseas Luka Lafaye de Micheaux

Abstract

This report shows our project on the adversarial robustness of neural networks by evaluating different attack methods such as FGSM, PGD, and MIM. We implemented adversarial training and gradient regularization to mitigate these attacks. Results show that while these methods enhance resistance to specific adversarial inputs, achieving comprehensive robustness continues to be a challenge.

1. Introduction

In this project, we addressed a straightforward classification task using the CIFAR-10 dataset and a basic model architecture. The primary goal was not to explore optimization techniques or model enhancements, but rather to improve performance through the use of adversarial attacks. During testing, we experimented with various attacking and defending strategies to achieve the best possible performance.

1.1. Adversarial Attacks

Adversarial attacks in machine learning are designed to manipulate models by inputting deceptive data that causes incorrect model outputs. Introduced prominently in the context of deep learning, these attacks subtly alter legitimate data samples in a way that remains undetectable to human observers but leads to significant errors in machine learning predictions. This method exposes vulnerabilities in the model's ability to generalize from training data to new, unseen scenarios.

The process of executing adversarial attacks typically involves crafting inputs that maximize the prediction error of the model—a concept known as the adversarial example. These attacks test the robustness of models by using techniques such as gradient-based methods to identify the most effective perturbations to the input data. The ultimate goal is to strengthen model defenses by identifying and mitigating these weaknesses, ensuring the model performs reliably even when faced with manipulated inputs that attempt to exploit its vulnerabilities.

1.2. CIFAR-10 Dataset

The CIFAR-10 dataset is a fundamental benchmark in the fields of machine learning and computer vision, akin to the MNIST database but with greater complexity. It comprises 60,000 color images, each 32x32 pixels, divided into 10 classes representing different objects such as birds, airplanes, and automobiles. Each class contains 6,000 images, offering a balanced dataset for training and testing various machine learning models. The dataset's diversity and its standardized format make it ideal for evaluating image recognition, classification, and generation tasks, providing a reliable baseline for comparing different computational techniques, as utilized in this project.

2. Attacking Techniques

We proceeded with three attacking techniques. The first two we used, as requested by the professors, were Projected Gradient Descent (PGD) and Fast Gradient Signed Method (FGSM) as requested by the professors. After, we implemented them and run some testing, we proceeded with the development of Momentum Iterative Method (MIM).

2.1. PGD

Mathematically, PGD can be seen as solving an optimization problem where the objective is to maximize the loss function. It introduces perturbation δ to the original input x under the constraint that the L_∞ norm of δ is less than a specified ϵ, representing the maximum allowable perturbation magnitude. The goal is to find the worst-case perturbation that maximizes the loss while staying within the ϵ-ball centered around the original input.

The PGD attack is conducted through iterative updates to the adversarial example, starting from the original image x and iteratively applying the update rule $x^{t+1} = \Pi_{x+S}(x^t + \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(\theta, x^t, y)))$, where α is the step size, $\nabla_x \mathcal{L}(\theta, x^t, y)$ is the gradient of the loss with respect to the input at iteration t, and Π_{x+S} denotes the projection onto the feasible set S of allowable perturbations around x. This iterative process enhances the adversarial capabilities of the perturbation by refining it across multiple steps, thereby enabling the exploration of more effective and subtle perturbations than those generated by single-step methods

like FGSM. This process is observable in the pseudo-code *Algorithm 1* based on the algorithm we made **"PGD.py"**.

Algorithm 1 PGD Attack

Input: Model $model$, images X, labels Y
Parameters: Perturbation ϵ, step size α, iterations N, device $device$
Output: Perturbed images X'

$X' \leftarrow X$
Enable gradient computation for X'
for $i \leftarrow 1$ to N **do**
    $outputs \leftarrow model(X')$
    $loss \leftarrow F.nll\_loss(outputs, Y)$
    Reset gradients: $model.zero\_grad()$
    Compute gradients: $loss.backward()$
    $X' \leftarrow X' + \alpha \cdot \text{sign}(X'.grad)$
    Clip X' within $[X - \epsilon, X + \epsilon]$ and $[0, 1]$
    Detach X' from the current graph
    Re-enable gradient computation for X'
end for
return X'

2.2. FGSM

FGSM utilizes the gradient of the loss function with respect to the input data to create adversarial examples. This method involves a simple yet powerful manipulation where the input is perturbed by adding noise proportional to the sign of the gradient of the loss function. Formally, the perturbation δ is defined as:

$$\delta = \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(\theta, x, y)) \tag{1}$$

where ϵ is a small scalar determining the magnitude of the perturbation, $\nabla_x \mathcal{L}(\theta, x, y)$ denotes the gradient of the loss $\mathcal{L}$ with respect to the input x, θ represents the parameters of the model, and y is the true label for x. The adversarially perturbed image x' is then generated by:

$$x' = x + \delta \tag{2}$$

ensuring that x' remains within the valid data range, typically achieved through clipping. FGSM is designed to be a one-step attack that efficiently finds adversarial examples by exploiting the linear behavior of deep neural networks in high-dimensional spaces. This method, while straightforward, has been shown to be effective against various models and is commonly used as a benchmark for evaluating the robustness of neural networks to adversarial attacks.

Algorithm 2 FGSM attack

Input: Neural network $model$, images X, labels Y
Parameters: Perturbation ϵ, computation device $device$
Output: Perturbed images X'

Enable gradient computation for X
$outputs \leftarrow model(X)$
$loss \leftarrow F.nll\_loss(outputs, Y)$
$model.zero\_grad()$
Compute gradients: $loss.backward()$
$X' \leftarrow X + \epsilon \cdot \text{sign}(X.grad)$
Clip X' to be within valid pixel range [0, 1]
return X'

2.3. MIM

MIM, on the other hand, enhances the robustness and effectiveness of adversarial attacks by integrating momentum into the iterative process of generating adversarial examples. This method builds upon the basic iterative approach by incorporating a momentum term, which helps in stabilizing update directions over iterations and escaping poor local maxima. The mathematical formulation of the MIM attack is described as follows:

$$g_{t+1} = \mu \cdot g_t + \frac{\nabla_x \mathcal{L}(\theta, x_t, y)}{\|\nabla_x \mathcal{L}(\theta, x_t, y)\|_1 + \epsilon} \tag{3}$$

where g_{t+1} is the accumulated gradient at iteration $t+1$, μ is the decay factor for the momentum term, $\nabla_x \mathcal{L}(\theta, x_t, y)$ represents the gradient of the loss function with respect to the input at iteration t, and ϵ is a small constant to avoid division by zero. The update rule for the adversarial example x_t can be expressed as:

$$x_{t+1} = \text{Clip}_{x,S}(x_t + \alpha \cdot \text{sign}(g_{t+1})) \tag{4}$$

Here, $\text{Clip}_{x,S}$ ensures that the adversarial example remains within the valid pixel range and adheres to the perturbation constraint set by ϵ, the maximum allowable perturbation. The parameter α denotes the step size. MIM's use of momentum allows it to aggregate gradients in a direction that consistently enhances the attack's effectiveness, making it particularly adept at dealing with models defended by gradient masking or other obfuscation techniques.

Algorithm 3 MIM Attack

Input: Model $model$, images X, labels Y
Parameters: Max perturbation ϵ, step size α, iterations N, momentum μ, device $device$
Output: Adversarially perturbed images X'

Initialize $g \leftarrow \mathbf{0}$ (same shape as X)
for $i \leftarrow 1$ to N **do**
  Enable gradient computation for X'
  $outputs \leftarrow model(X')$
  $loss \leftarrow F.nll\_loss(outputs, Y)$
  Compute gradients: $grad \leftarrow$ autograd.grad$(loss, X')[0]$
  Normalize gradients: $grad\_norm \leftarrow$ torch.norm$(grad.view(grad.shape[0], -1), p = 1, dim = 1)$
  $grad\_normalized \leftarrow grad/(grad\_norm.view(-1, 1, 1, 1) + 1e - 8)$
  Update momentum: $g \leftarrow \mu \cdot g + grad\_normalized$
  $X' \leftarrow X' + \alpha \cdot \text{sign}(g)$
  Clip change: $delta \leftarrow$ torch.clamp$(X' - X, min = -\epsilon, max = \epsilon)$
  Clip X': $X' \leftarrow$ torch.clamp$(X + delta, min = 0, max = 1)$
  Detach X' from computation graph
end for
return X'

3. Techniques of Defense and Results

3.1. Adversarial Training

The first attempt that was tried against all attacking methonds, was Adversarial Training. This involves modifying real data points to create adversarial examples, to cause the model to train on them. Mathematically, this means that given legitimate input x and its true label y, an adversarial example x' is created by adding a small perturbation δ such that:

$$x' = x + \delta$$

The perturbation δ is constrained by $\|\delta\|_p < \epsilon$, where ϵ is a small scalar defining the magnitude of the perturbation, and p represents the norm (commonly L_2 or L_∞) used to measure the perturbation size.

The primary objective is to maximize the model's prediction error on the perturbed input. This is achieved by maximizing the loss function $J(\theta, x', y)$ with respect to the model parameters θ, leading to the model misclassifying x':

$$\max_{\delta} J(\theta, x + \delta, y) \quad \text{subject to} \quad \|\delta\|_p < \epsilon$$

To defend against such attacks, adversarial training involves modifying the training process to include adversarial examples:

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\alpha J(\theta, x, y) + (1 - \alpha) J(\theta, x + \delta^*, y)\right]$$

Here, δ^* is the optimal perturbation that maximizes the loss within the ϵ-ball around x, and α is a mixing parameter that balances the importance of natural and adversarial examples in the training process. This approach aims to improve the model's robustness by minimizing the expected loss over both clean and perturbed inputs.

Natural Accuracy	PGD l_∞	PGD l_2
50%	5.99%	25.15%

Table 1. Results of the models trained naturally with no attack or defense mechanism.

Model	Natural Accuracy	PGD l_∞	PGD l_2
PGD	56.25%	17.26%	26.40%
FGSM	50%	0.46%	8.48%
MIM	37.5%	17.23%	26.36%

Table 2. Results of the models trained with adversarial training.

On the *table 2* we observe that Adversarial attacks have risen the percentages except for FGSM. Tested, though, under FGSM attacks we managed to gain 84% accuracy. This show's that the model was overfitting and was led to memorization. The purpose of FGSM will be meaningful later where we'll analyze the Multi-Attack Technique, where we managed to obtain the best results.

3.2. Elementary regularization techniques

These techniques are based on the idea that neural networks with a large Lipschitz constant are more sensitive to small perturbations in their inputs, making them more vulnerable to adversarial attacks. One way to try to defend against attacks exploiting the gradient with respect to the input is to regularize the model, in a Lipschitzian sense.

3.2.1. Spectral Normalization

Spectral Normalization consists in performing the following operation to each of the layers, which clearly rescales the Lipschitz constant of the layer, and therefore of the network. This is known to be an efficient operation to train the discriminator is a GAN setting (6)

$$\hat{W} = \frac{W}{\sigma(W)}$$

where $W \in \mathbb{R}^{m\times n}$ is the weight matrix of the layer and $\sigma(W)$: the spectral norm of W, defined as the largest singular value of W.

The operation is straightforward to implement in PyTorch. We simply apply the `torch.nn.utils.spectral_norm` method when creating our layers.

3.2.2. Orthogonal Normalization

Albeit more far fetched, this regularization technique has been experienced with to disentangle the latent space (7)

$$\mathcal{L} = \mathcal{L}_{\text{task}} + \lambda \mathcal{L}_{\text{orth}}$$

where $\mathcal{L}_{\text{orth}} = \|W^\top W - I\|_F^2$ is the orthogonal regularization loss. The hyperparameter λ was approximately set by empirically checking that $\|W^\top W - I\|_\infty < 0.1\ foreachlayer$.

This technique proved to be less efficient than Spectral Normalization, probably because the task we

3.2.3. Gradient Regularization

The training process incorporates both adversarial examples and gradient regularization to enhance model robustness. The main steps include, where we also generate adversarial examples by altering the inputs with the aim to mislead the model during training. The new inputs x' are set to require gradients to compute the input gradients after the backward pass on the task loss. The combined loss function contains the task loss but also a gradient penalty term calculated as the mean of the squares of the L_2 norms of these gradients across the batch:

$$P = \frac{1}{N} \sum_{i=1}^{N} \left(\left\| \nabla_{\mathbf{x}_i'} L \right\|_2 \right)^2$$

This regularization technique helps in controlling the influence of input perturbations on the model's output, making the model more robust to small changes or noise in the input data. It's observable in *table 3*, where we tested it for PGD that the model became more robust:

Natural Accuracy	PGD l_∞	PGD l_2
25%	27.43%	27.01%

Table 3. Results of the model with trained with PGD attacks and Gradient Regularization.

3.3. Multi-Attack Technique

In this strategy, we applied multiple attack methods cyclically during the training of the model. Thus, we enhanced the robustness of the model by exposing it to various types of perturbations throughout the training process. In each training epoch, for every batch of data, one of the attacks is selected cyclically (i.e., the attack method cycles through FGSM, MIM, and PGD). The selection is based on the current batch index modulo the number of attacks, ensuring each attack is applied evenly throughout training.

The adversarial examples of each batch are used all together as inputs for the training steps. This way, the unstable performance of FGSM proved essential for the model, as rather than leading to overfitting, it was used in parallel with the other models, thus having more diverse inputs. By observing *table 4* we can see that the performance of this specific technique proved the superior one:

Natural Accuracy	PGD l_∞	PGD l_2
43.75%	28.39%	34.87%

Table 4. Results of the model trained with the MultiAttack technique.

3.4. Conclusions

After evaluating all results, PGD and MIM individually emerged as the most effective attacking techniques. The most successful defense strategy was found to be the use of multiple attacks in training

References

[1] I. Goodfellow et al. (2015), *Explaining and Harnessing Adversarial Examples*, International Conference on Learning Representations (ICLR).

[2] A. Madry et al. (2017), *Towards Deep Learning Models Resistant to Adversarial Attacks*, arXiv preprint arXiv:1706.06083.

[3] N. Carlini, D. Wagner (2017), *Towards Evaluating the Robustness of Neural Networks*, IEEE Symposium on Security and Privacy (SP).

[4] A. Athalye et al. (2018), *Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples*, International Conference on Machine Learning (ICML).

[5] Y. Dong et al. (2018), *Boosting Adversarial Attacks with Momentum*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[6] T. Miyato, T. Kataoka, M. Koyama, Y. Yoshida (2018), *Spectral Normalization for Generative Adversarial Networks*, arXiv preprint arXiv:1802.05957.

[7] B. Liu, Y. Zhu, Z. Fu, G. de Melo, A. Elgammal (2019), *OOGAN: Disentangling GAN with One-Hot Sampling and Orthogonal Regularization*, arXiv preprint arXiv:1905.10836.