Data Science Lab

JoseMourinho Group

Robustness of Neural Networks

Marc KASPAR marc.kaspar@dauphine.eu
Othman HICHEUR othman.hicheur@dauphine.eu
Constant BOURDREZ constant.bourdrez@dauphine.eu

Submission Date : December 10, 2024

2024/2025 - Semester I

Contents

1 Introduction **2**

2 Implemented attacks **2**

3 Training a robust NN **2**

3.1 Adversarial Training 3

3.2 Parametric Noise Injection 3

4 Stochastic Local Winner Takes It All **3**

5 Results **4**

6 Future work **6**

1 Introduction

Neural networks have achieved impressive success across various domains but remain vulnerable to adversarial threats. Addressing these vulnerabilities requires a comprehensive understanding of weaknesses in machine learning models, particularly their susceptibility to attacks that exploit fragile decision boundaries. To tackle these weaknesses, it is crucial to study attack mechanisms like adversarial perturbations and poisoning. These strategies expose the flaws in a model's generalization, paving the way for developing robust countermeasures. Building on this knowledge, the implementation of defense mechanisms becomes essential. Techniques such as adversarial training and robust optimization aim to protect models by reinforcing their stability against known threats. Finally, understanding and refining decision boundaries links these efforts together. Decision boundary analysis provides insights into model behavior under adversarial conditions and guides the design of resilient systems capable of maintaining performance even in challenging scenarios. This exploration integrates these key components to advance the robustness of neural networks.

2 Implemented attacks

In this project, we implemented various types of adversarial attacks to evaluate the robustness of our network. Notably, strong performance against one type of attack does not guarantee resilience against others. The input and latent spaces of the classifier can be characterized in multiple ways, meaning a network might be vulnerable to certain representations while remaining robust to others.

FGSM is a single-step attack designed to craft adversarial examples by perturbing the input in the direction of the gradient of the loss function. The perturbed input is computed as:

$$\mathbf{x}_{\text{adv}} = \mathbf{x} + \epsilon \cdot \text{sign}(\nabla_{\mathbf{x}} \mathcal{L}(f_\theta(\mathbf{x}), y)),$$

where $\mathbf{x}$ is the original input, y is the true label, ϵ is the perturbation budget, and $\nabla_{\mathbf{x}} \mathcal{L}(f_\theta(\mathbf{x}), y)$ is the gradient of the loss $\mathcal{L}$ with respect to the input. The sign function ensures that the perturbation is applied in the direction that maximally increases the loss.

To implement FGSM, we compute the gradient using backpropagation, extract its sign, and add the scaled perturbation to $\mathbf{x}$. For example:

PGD extends FGSM by iteratively applying the gradient update while keeping the perturbation within a predefined ℓ_∞-norm ball. Each iteration updates the adversarial example as:

$$\mathbf{x}_{\text{adv}}^{(k+1)} = \text{Proj}_{\mathcal{B}_\epsilon(\mathbf{x})} \left(\mathbf{x}_{\text{adv}}^{(k)} + \alpha \cdot \text{sign}(\nabla_{\mathbf{x}} \mathcal{L}(f_\theta(\mathbf{x}_{\text{adv}}^{(k)}), y)) \right),$$

where α is the step size, $\mathcal{B}_\epsilon(\mathbf{x})$ is the ℓ_∞-norm ball around $\mathbf{x}$, and Proj projects the perturbed input back onto this ball.

The CW (Carlini & Wagner) [1] attack formulates the adversarial example generation as an optimization problem that minimizes this:

$$\min_{\mathbf{z}} \|\tanh^{-1}(\mathbf{z}) - \mathbf{x}\|_2^2 + c \cdot \max(0, f(\mathbf{x}'_y) - \max_{j \neq y} f(\mathbf{x}'_j)) + \lambda \cdot \|\mathbf{z}\|_2^2$$

3 Training a robust NN

Building robust neural networks involves designing strategies to improve their resilience against adversarial attacks and other sources of uncertainty. This section explores two key methods for enhancing robustness: adversarial training, which directly incorporates adversarial examples into the training process, and parametric noise injection, which introduces controlled noise into the model to encourage stability and improve generalization.

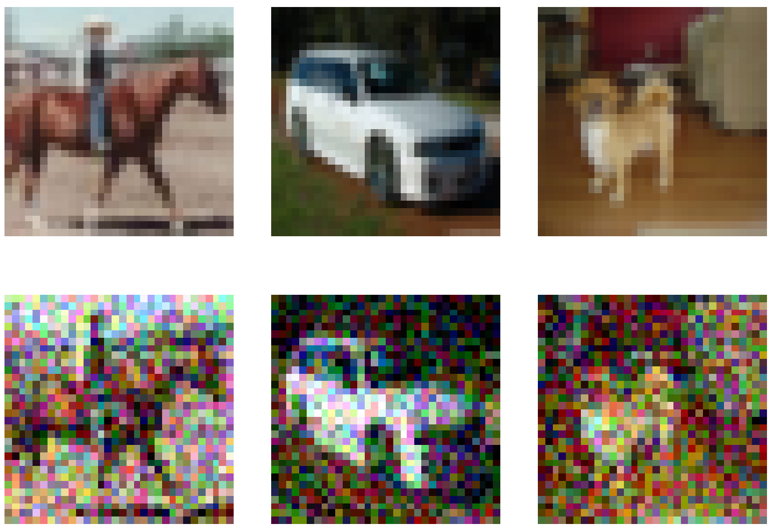

Figure 1: PGD attacks with $\epsilon = 0.3$, $\alpha = 0.01$, steps $= 3$ and ℓ_∞-norm

3.1 Adversarial Training

For our adversarial training, we adopted a straightforward approach. At each iteration, we sampled a number uniformly between 0 and 7. Based on this number, the inputs were either transformed into adversarial examples or left unchanged. The probability of applying an attack, p_{att}, was $\frac{4}{8}$, while the probability of keeping the original input, p_{clean}, was $\frac{4}{8}$. The attacks included PGD, FGSM, CW, or the addition of random noise. Only one attack is performed for each iteration. This strategy was designed to explore the local neighborhood of each input x_i while preserving the decision boundary, thereby improving the model's robustness.

3.2 Parametric Noise Injection

Parametric noise injection is a technique aimed at improving the robustness and generalization of neural networks by introducing controlled noise into the model during training. Unlike adversarial training, which focuses on perturbing inputs to mimic adversarial attacks, parametric noise injection targets the internal parameters or intermediate representations within the model. By exposing the network to noise during training, it learns to stabilize its predictions against small perturbations, resulting in a more robust decision boundary.

As described by [4], this method involves adding noise directly to parameters, such as weights. The noise is drawn from a Gaussian distribution, with the magnitude carefully controlled to avoid disrupting learning. This approach makes the model more resilient to unexpected variations during inference, including those introduced by adversarial attacks or real-world uncertainties.

4 Stochastic Local Winner Takes It All

After implementing adversarial training, we weren't happy with our results so we went for an other method introduced in [3]. The stochastic Winner-Takes-All (Stochastic LWTA) model replaces traditional activation functions like ReLU with stochastic competition blocks, where multiple linear units in each block compete locally, and a winner is selected based on a probabilistic mechanism. This stochastic selection, determined by the units' output magnitudes, introduces a level of randomness that adversarial attacks find difficult to exploit. By enforcing sparsity—where only the winning unit contributes to the output—LWTA blocks streamline the network's responses and enhance generalization. The model incorporates an Indian Buffet Process (IBP) prior, a Bayesian method to dynamically manage connections by enabling or disabling them layer-by-layer. Additionally, the LWTA mechanism extends to convolutional layers, applying the same competitive process among feature maps within kernels. Here is an explanation of how it works for convolutional layers:

- We take $W_b \in \mathbb{R}^{h\times l\times C\times U}$ which is the kernel weight and $X \in \mathbb{R}^{H\times L\times C}$ which is the input and we compute the convolution between the two :

  $$H_{b,u} = W_{b,u} * X \in \mathbb{R}^{H\times L}$$

  where U is the number of feature maps and B, with $b \in 1,...,B$, the number of LWTA blocks.

- Next, for each $h' \in 1,...,H$ and $l' \in 1,...,L$, we determine which $H_{b,u,h',l'}$ will be selected randomly with probability distribution determined by the softmax of all the $H_{b,u,h',l'}$, for all $b \in 1,...,B$

- From the previous step, we obtained a one hot encoded vector which will select the *winner* of the feature maps from one unit.

- Then, we select the best feature maps by doing a dot product between the one hot encoded vector and $H_{b,u}$ to get $Y_{b,u}$. Finally, we concatenate all the $Y_{b,u}$ and we have the output of the convolutional layer $Y \in \mathbb{R}^{H\times L\times BU}$ [3].

Moreover, we use the ELBO loss during the training:

$$\mathcal{L} = -\sum_{(X_i,Y_i)\in\mathcal{D}} \mathrm{CE}(Y_i, f(X_i,\hat{\xi})) - \sum_b \left[\log q(\hat{\xi}_b) - \log p(\hat{\xi}_b)\right]$$

where CE is the cross-entropy loss, $\{X_i, Y_i\} \in \mathcal{D}$ is a pair of data and label, $f(X_i,\hat{\xi})$ is the class probabilities of X_i generated by the last softmax layer of the last LWTA block and $q(\hat{\xi}_b), p(\hat{\xi}_b)$ are the probability distribution of the posterior and prior [3].
For our experiments with the method of [3], we use the model WideResNet34 [5] and we applied the previous techniques. We train on 30 epochs, 1 widen factor for the model and a learning rate of 0.1 with stochastic gradient descent optimizer. Furthermore, we choose $\epsilon = \frac{8}{255}$, $\alpha = 0.007$ and 20 steps for the adversarial training, where we only use the PGD attacks [2].

5 Results

Model	Natural Accuracy (in %)	PGD Accuracy (in %)	FGSM Accuracy (in %)	CW Accuracy (in %)
Regular training	63.67	1.37	0.98	34.08
Noise Injection	57.42	1.07	1.37	26.46
Smoothing	57.81	1.07	2.34	29.59
Smoothing, PGD	62.89	0.39	5.27	26.86
Smoothing, PGD, FGSM	56.64	0.39	6.54	23.67
Smoothing, PGD, FGSM, CW	58.98	0.59	6.35	34.08
Smoothing, PGD, FGSM, CW + Noise Injection	59.12	14.26	53.28	56.9
Stochastic LWTA	**67.9**	36.23	11.33	**68.75**

Table 1: Accuracy Evaluation on Different Attack Types, $\epsilon = 0.3$, $\alpha = 0.01$, $\kappa = 0.01$, $c = 1e^{-4}$

The table 1 (results from local) and 2 (results from the platform) presents the progression of metrics on the testing data for various methods. The Stochastic Local Winner Takes it All method [3] achieves the best results, with the highest accuracies for both cleaned and attacked data, except for

Model	Natural Accuracy (in %)	PGD Accuracy aggregated with L_2 and L_∞(in %)	FGSM Accuracy (in %)	CW Accuracy (in %)
Stochastic LWTA	**68.8**	**114.6**	**11.5**	**66.0**

Table 2: Accuracy Evaluation on Different Attack Types, $\epsilon = 0.3$, $\alpha = 0.01$, $\kappa = 0.01$, $c = 1e^{-4}$

the FGSM attacks (because we don't train over FGSM attacks). On the platform, our results were the best this year. However, for adversarial training, we appear to have failed in defending against PGD attacks. The decision boundary generated is overly dependent on the l_2 norm; while the performance was strong against l_2 PGD attacks on the platform, it was not as effective for l_∞ PGD attacks. Our final method still lacks computational efficiency, as both the training and inference phases are highly resource-intensive for such a relatively small task.

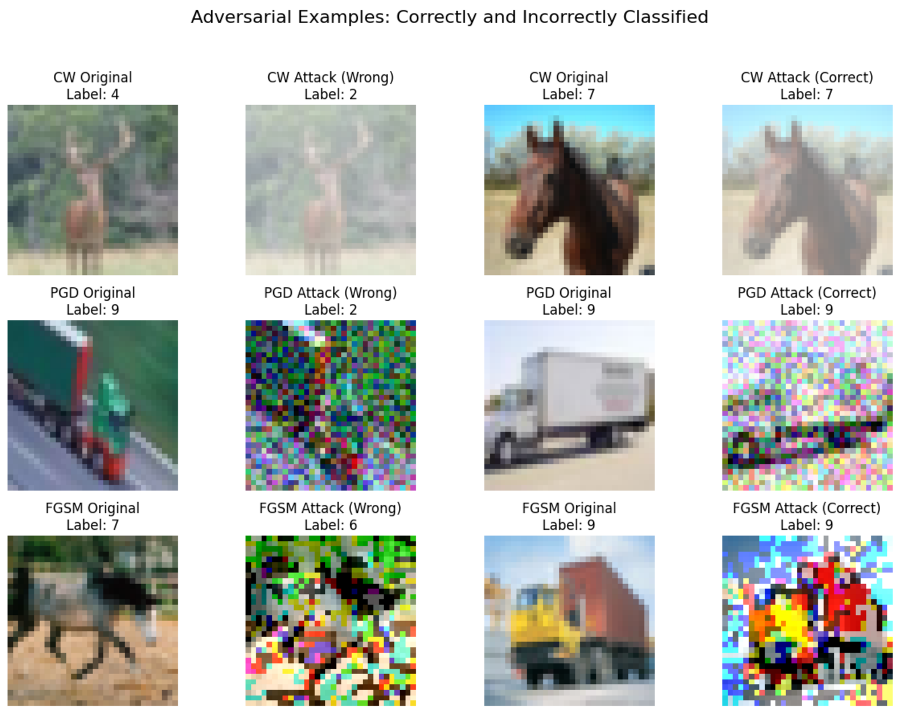
Adversarial Examples: Correctly and Incorrectly Classified
CW Original Label: 4
CW Attack (Wrong) Label: 2
CW Original Label: 7
CW Attack (Correct) Label: 7
PGD Original Label: 9
PGD Attack (Wrong) Label: 2
PGD Original Label: 9
PGD Attack (Correct) Label: 9
FGSM Original Label: 7
FGSM Attack (Wrong) Label: 6
FGSM Original Label: 9
FGSM Attack (Correct) Label: 9

Figure 2: Adversarial classification for SLWTA

Figure 2 showcases examples of adversarial attacks, highlighting cases where images are correctly classified and others where they are misclassified. Notably, some highly recognizable images, such as the CW attack on the deer, are misclassified, whereas severely distorted images, such as the truck, remain correctly classified. This discrepancy raises an important question: how can we align algorithmic perception with human perception? For instance, the deer in the top-left image should logically be classified as a deer, while the heavily damaged truck should not be easily recognizable. In certain domains, such as cybersecurity or safety-critical applications in visual recognition, ensuring robustness in both scenarios is mandatory to meet stringent safety and reliability requirements.

Figure 3 demonstrates that our network is often not very confident in its predictions, with the mean probability frequently close to 0.1, which is expected since there are 10 classes. When examining the distribution for correctly classified images, the probability tends to be higher, but it still remains relatively low. This suggests that there is room for improvement in terms of increasing the network's confidence, which could be a focus for future work.

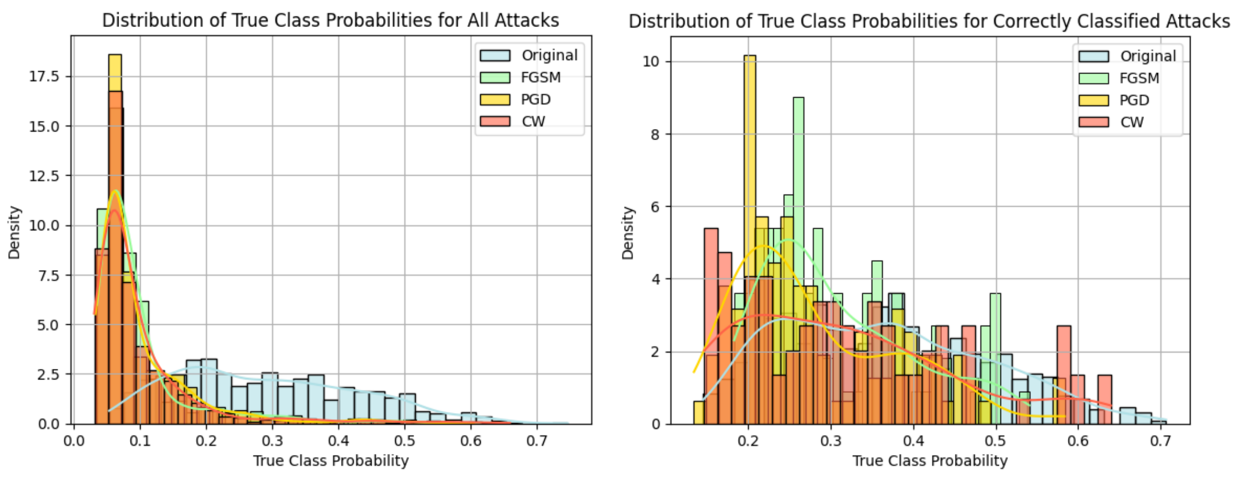

Figure 3: Left: Distribution of the predicted class probabilities for all examples in the test dataset. Right: Distribution of the predicted class probabilities for the correctly classified examples in the test dataset.

6 Future work

Our current method has a processing time of 166ms per image, which is quite excessive. One potential improvement is to optimize our program by reducing the number of layers or adjusting the number of blocks, tailored to the specific requirements of this problem. Another way to enhance our results is by modifying how we introduce stochasticity. In SLWTA, stochasticity is currently introduced by randomly selecting winners during the decision-making process. We could explore alternative approaches, such as selecting the winner probabilistically and incorporating controlled noise to introduce stochasticity while reducing randomness.

References

[1] Nicholas Carlini and David A. Wagner. Towards evaluating the robustness of neural networks. *CoRR*, abs/1608.04644, 2016.

[2] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2019.

[3] Konstantinos P. Panousis, Sotirios Chatzis, and Sergios Theodoridis. Stochastic local winner-takes-all networks enable profound adversarial robustness, 2021.

[4] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Parametric noise injection: Trainable randomness to improve deep neural network robustness against adversarial attack. *CoRR*, abs/1811.09310, 2018.

[5] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks, 2017.