Training robust neural networks

Stochastic Local Winner Takes It All

Othman Hicheur, Constant Bourdrez, Marc Kaspar

Université Paris-Dauphine - PSL

December 2024

Attacks implemented: FGSM

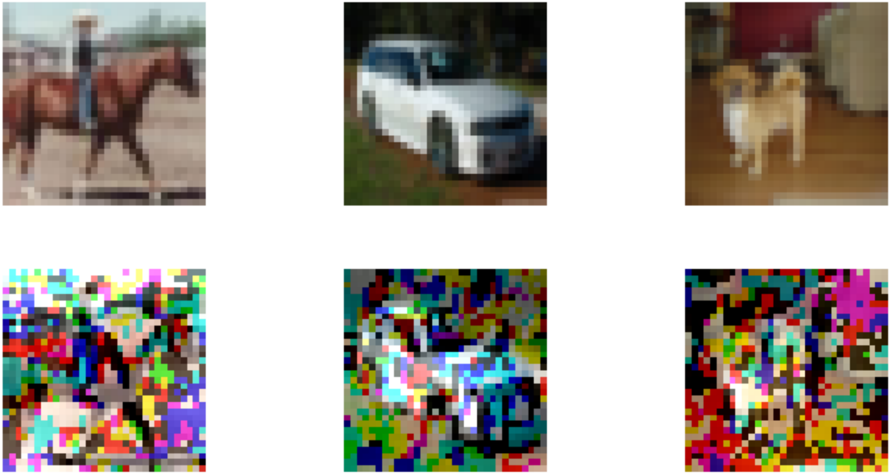


Figure: FGSM attacks with $\epsilon = 0.3$

Attacks implemented: Carlini & Wagner

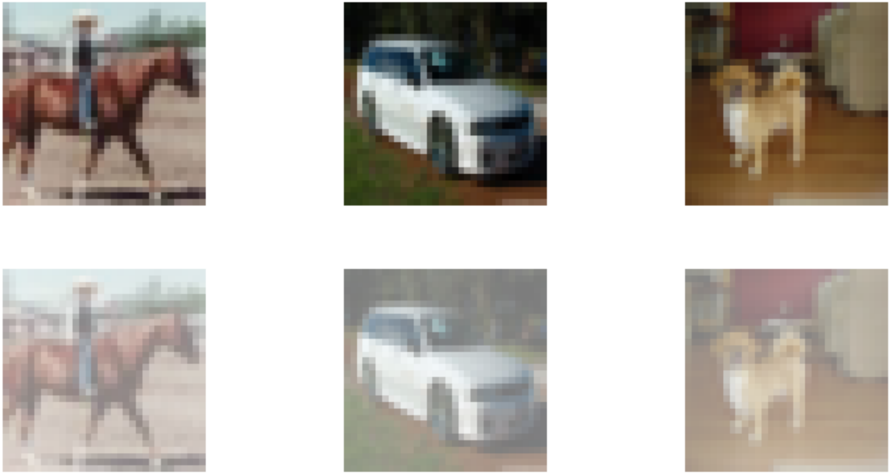


Figure: Carlini & Wagner attacks with $c = 0.0001$, $\kappa = 0.01$ and steps $= 3$

Attacks implemented: PGD

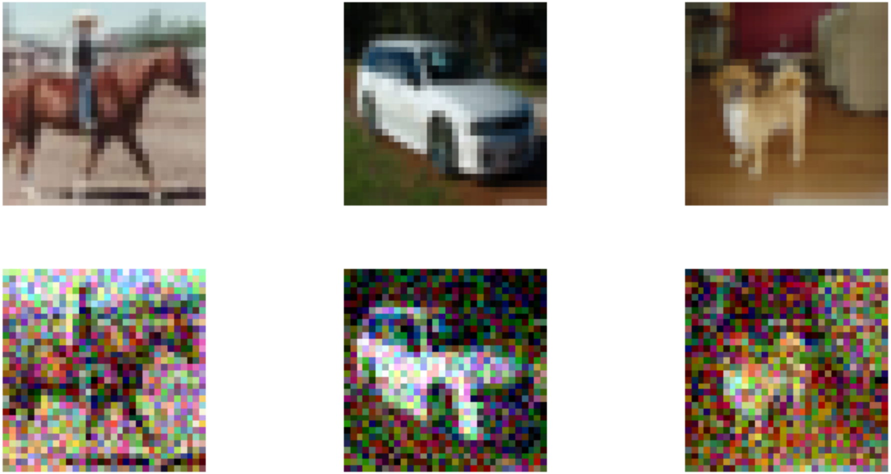


Figure: PGD attacks with $\epsilon = 0.3$, $\alpha = 0.01$, steps $= 3$ and ℓ_∞-norm

All attacks combined

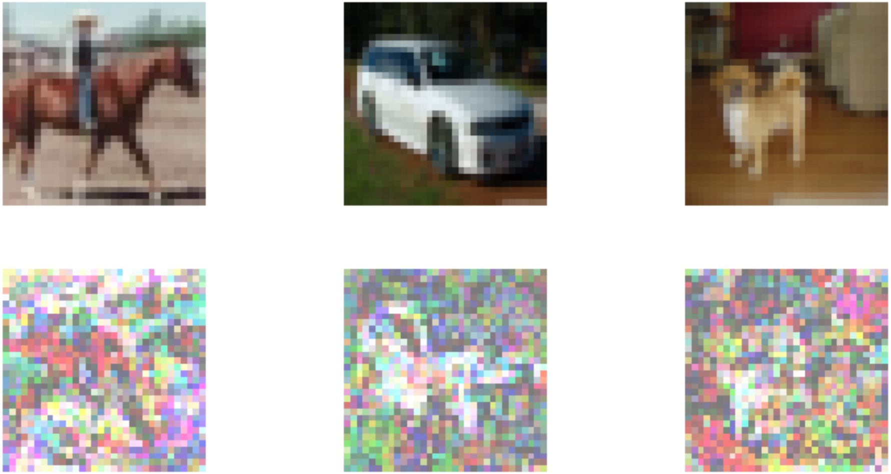


Figure: All 3 previous attacks combined with the same parameters

Adversarial Training

At each iteration, a number is sampled uniformly between 0 and 7:

0: Gaussian Noise
1: FGSM
2: PGD
3: Carlini & Wagner
4-7 : No Attacks

Parametric Noise Injection

Adds Gaussian Noise to the parameters and make the model more robust against small perturbations thanks to a more stable decision boundary.

3 layers added:

- One after the first pooling layer
- One after the second pooling layer
- One after the flatten layer

However, the results were still not good enough

Results Parametric Noise + Smoothing

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

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

Stochastic Local Winner Takes It All

- 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
- Introduces randomness and enforces sparsity which makes the adverserial attacks less effective
- Incorporates an Indian Buffet Process (IBP) prior, to dynamically manage connections by enabling or disabling them layer-by-layer

Stochastic Local Winner Takes It All

For Convolution Layers:

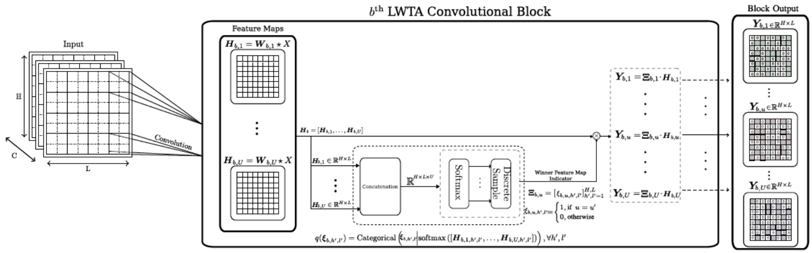
$$H_{b,u} = W_{b,u} * X \in \mathbb{R}^{H \times L}$$

$W_b \in \mathbb{R}^{h \times l \times C \times U}$ kernel weight
$X \in \mathbb{R}^{H \times L \times C}$ input
U number of feature maps
B, with $b \in 1, ..., B$, number of LWTA blocks.

Stochastic Local Winner Takes It All

- 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 $Y \in \mathbb{R}^{H \times L \times BU}$

Architecture


b^{th} LWTA Convolutional Block
Input
H
C
L
Convolution
Feature Maps
$\boldsymbol{H}_{b,1} = \boldsymbol{W}_{b,1} \star X$
$\boldsymbol{H}_{b,U} = \boldsymbol{W}_{b,U} \star X$
$\boldsymbol{H}_b = [\boldsymbol{H}_{b,1}, \dots, \boldsymbol{H}_{b,U}]$
$\boldsymbol{H}_{b,1} \in \mathbb{R}^{H \times L}$
$\boldsymbol{H}_{b,U} \in \mathbb{R}^{H \times L}$
Concatenation
$\mathbb{R}^{H \times L \times U}$
Softmax
Discrete Sample
Winner Feature Map Indicator
$\Xi_{b,u} = [\xi_{b,u,h',l'}]_{h',l'=1}^{H,L}$
$\xi_{b,u,h',l'} = \begin{cases} 1, \text{if } u = u' \\ 0, \text{otherwise} \end{cases}$
$q(\boldsymbol{\xi}_{b,h',l'}) = \text{Categorical}\left(\xi_{b,h',l'} \middle| \text{softmax}([\boldsymbol{H}_{b,1,h',l'}, \dots, \boldsymbol{H}_{b,U,h',l'}])\right), \forall h', l'$
$\boldsymbol{Y}_{b,1} = \Xi_{b,1} \cdot \boldsymbol{H}_{b,1}$
$\boldsymbol{Y}_{b,u} = \Xi_{b,u} \cdot \boldsymbol{H}_{b,u}$
$\boldsymbol{Y}_{b,U} = \Xi_{b,U} \cdot \boldsymbol{H}_{b,U}$
Block Output
$\boldsymbol{Y}_{b,1} \in \mathbb{R}^{H \times L}$
$\boldsymbol{Y}_{b,u} \in \mathbb{R}^{H \times L}$
$\boldsymbol{Y}_{b,U} \in \mathbb{R}^{H \times L}$

Loss

We use the ELBO loss:

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

- CE cross-entropy loss
- $\{X_i, Y_i\} \in \mathcal{D}$ pair of data and label
- $f(X_i, \hat{\xi})$ class probabilities of X_i generated by the last softmax layer of the last LWTA block
- $q(\hat{\xi}_b), p(\hat{\xi}_b)$ probability distribution of the posterior and prior

Parameters

We used the WideResNet34 model

- 30 epochs
- 1 widen Factor
- SGD with 0.1 Learning Rate

For Adverserial training (PGD attacks only):

- $\epsilon = \frac{8}{255}$
- $\alpha = 0.007$
- 20 steps

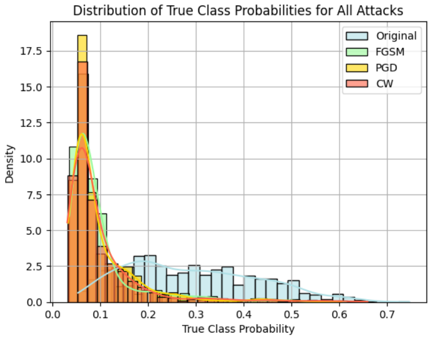
Results Stochastic Local Winner Takes It All

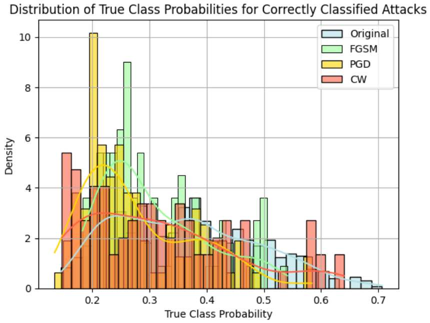
We get way better results especially for PGD attacks.

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**

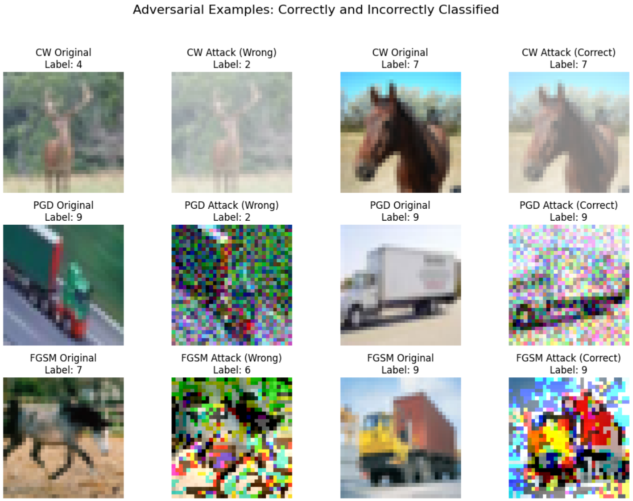
Table: Accuracy Evaluation on Different Attack Types, $\epsilon = 0.3$, $\alpha = 0.01$, $\kappa = 0.01$, $c = 1e^{-4}$

Distribution of True Class Probabilities





Example of correct and incorrect classification



Conlusion

- Good results on PGD attacks (optimized for it)
- Results for adversarial training quiet disappointing
- Find new method to introduce stochasticity in the SLWTA
- Reduce computation time by optimizing for the desired goal or finding other ways

Thank you !