Université Paris-Dauphine

Mathématiques et Informatique de la Décision et des Organisations

Adversarial attacks: numerical study of the projected gradient method

Authors:
Mayard Hippolyte
Deschamps Théo
Genty Eymeric
Hosseinkhan Rémy

Advisor:
Rizk Geovani

M2 Intelligence Artificielle Systèmes Données

April 2020

Acknowledgements

It was a real pleasure to learn from the *adversarial attacks* methods and implement some method using the Tensorflow framework designed for numerical computation and large-scale machine learning. Thanks to Geovani Rizk for proposing this research topic that has opened our scientific minds. A lot of knowledge has been gained through this work.

Contents

1 Introduction

The aim of this study is to understand some classical problems related to adversarial attacks that are presented by Goodfellow et al. in [1]. Multiple approaches are proposed to underline the properties of the adversarial attacks. In particular, we try to reproduce and observe some attacks properties exposed in *Towards Deep Learning Models Resistant to Adversarial Attacks*, Madry et al. [2].

Recall that the central object of study is the following saddle point problem:

$$\min_{\theta} \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y) \right] \tag{1}$$

One classical way to find a solution of the inner maximization problem of the equation 1 is to perform a projected gradient ascent describe in the following equation.

$$x^{t+1} = \Pi_{x+S} \left(x^t + \eta \operatorname{sgn} \left(\nabla_x L(\theta, x, y) \right) \right) \tag{2}$$

This document aims to discuss about the influence of multiple factors in the process of designing adversarial attacks. The study focuses on the projected gradient ascent presented in the above equation 2 since it is a fundamental method to create adversarial attacks.

Everything is implemented from scratch since it allows to deeply understand the underlying optimisation problem define below. The S neighbourhood related to equation 1 is always a ball of a given radius, in the context of the present document. By Danskin's theorem, the solution of the saddle point problem can be approximate using solutions of the inner maximisation problem, it is called the adversarial training.

This optimisation problem has been tackled by investigating the behaviour of the projected gradient method through different approaches. First, a grid-search is performed to observe and understand the effects of the different parameters on the accuracy under-attack, with a short analysis of the behaviour of the method when solutions of equation 1 are ℓ_2 bounded. Then, we will investigate how the depth and the width of a network will impact an attack in the case of a Blackbox attack. Third, we try to understand the influence of hyper-parameters settings on classifier robustness. We have been focusing more particularly on the overfiting phenomena and its impact on classifiers robustness. Finally, experiments are made to understand how to defend against adversarial attacks.

2 Adversarial attacks grid-search

In order to have a better understanding of the power of the attacks with respect to parameters, a grid-search is performed. This experiment helps to perceive the intended meaning of the saddle point problem. It is also a way to verify that our implementations are well-defined.

Given a fixed model f_θ trained on the CIFAR10 data set and an image $x \in \mathbb{R}^p$ with $p = 32\times32\times3 = 3072$, we build multiple attacks $x_{\epsilon,\eta,n,L}(\omega)$ corresponding to perturbations of the initial image x, where each of the hyper-parameters represents the following:

- The value ϵ is the radius of the ball where the gradient ascent is restricted. Here $\epsilon \in [[1, 9]]$ since allowing a wider intensity value deviation would deform the picture. Note the value used in [2] is $\epsilon = 8$.

- The value η is the gradient step size in the projected ascent. Here $\eta \in [[1, 9]]$, it means that one step modify from 1 to 9 intensity values for each of the channels of x.
- The integer n is the step number of the projected gradient ascent. Here $n \in [[1, 8]]$.
- The loss L is the loss function from equation 1 that is used to perform the gradient ascent. Here $L = \{\textit{cross-entropy},\ \textit{mean-square error}\}$.
- The ω value represent the randomness of attack examples $x_{\epsilon,\eta,n,L}$. Indeed, one can compute an attack by gradient ascent starting with an initial random perturbation $x + \delta$ instead of x, where δ is a white noise [2]. Consequently, some generated attacks are random variables, we denote by $x_{\epsilon,\eta,n,L}(\omega_{rand})$ examples generated using a random start within the allowed neighbourhood and $x_{\epsilon,\eta,n,L}(\omega_0)$ adversarial exemples generated from the original image x, formally $\delta(\omega_0) = 0$. Hence, $\omega \in \Omega = \{\omega_0,\ \omega_{rand}\}$.

Note the loss L used to train the model f_θ is the *cross-entropy*, hence attacks $x_{\epsilon,\eta,n,L}(\omega)$ using $L = \textit{mean-square error}$ may be considered as *grey-box attacks* since they are generated from the gradients of f_θ but use a different loss.

In the grid-search, the model f_θ is a three convolutional block neural-network with respectively 32, 64, 128 filters. Each block applies ReLU and batch normalisation, average pooling is used to reduce the dimensions close to the output layer. The test set used for testing different under-attack accuracy is of size 10000. Hence more than 26 millions of adversarial images have been generated since the permutations of parameters produce more than 2600 distincts attack configurations.

Remark. *Here, integers values are used for ϵ and η to keep their intuitive meaning. It is equivalent to normalize those values by 255 in order to represent images in the unit hypercube of dimension p. Consequently, hyperparameters values used during the grid-search are divided by 255.*

Remark. *Different values of ϵ are used depending on the norm associated to the ball (ℓ_∞ or ℓ_2), more on this subject is developed later. Besides the ball radius, all parameters are equals in both cases.*

In this section, graphs shows an estimate of the central tendency (the mean) and a confidence interval for that estimate. The confidence interval is calculated by aggregation of the grouped values.

2.1 Relation between hyperparamters and accuracy

2.1.1 Influence of the constraint

From the figure 1, one can observe the accuracy of f_θ, with respect to the (test) attack sets generated, decrease when ϵ increase. It is obvious since the higher the ϵ the weaker the optimisation constraint. However, it allows to check the algorithms are well implemented. Note the large confidence interval sizes for *FGSM* come from the different step sizes η that lead more or less close to the local optimal point. For both methods graphs are almost the same, no matter random start is applied. Hence, in the following graphs, the accuracy is a function of epsilon.

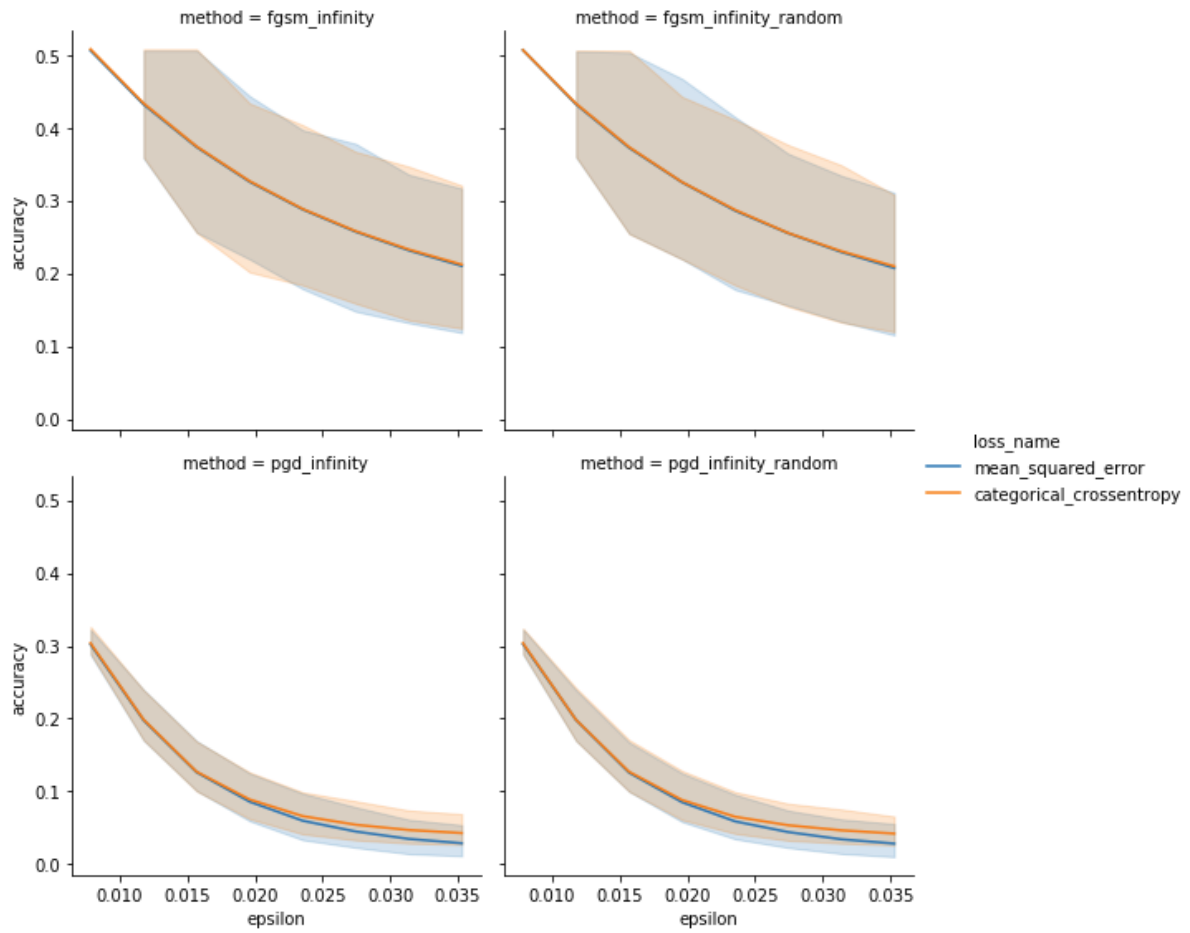

Figure 1: Accuracy of f_θ with respect to the radius ϵ of the ℓ_∞ ball

The same behaviour is observed in the case of the euclidean ball (figure 2).

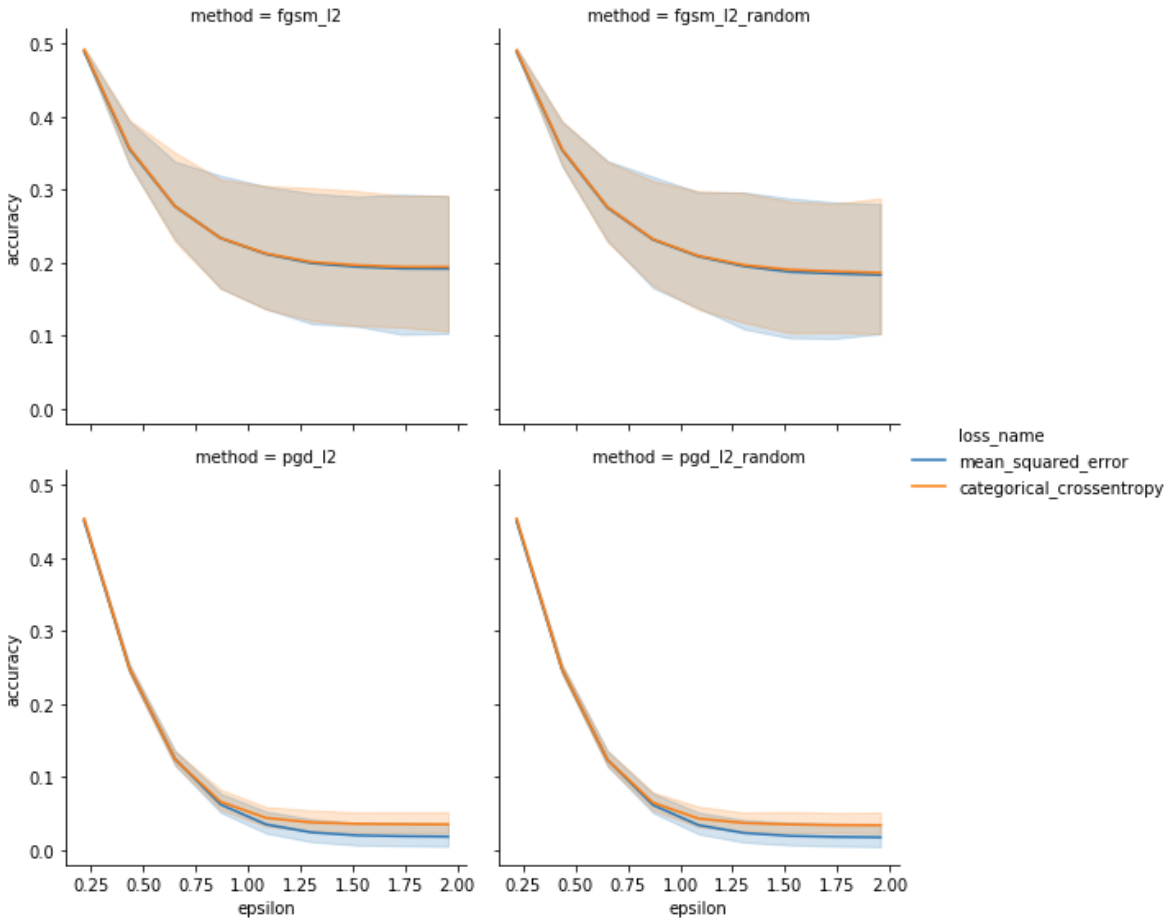

Figure 2: Accuracy of f_θ with respect to the radius ϵ of the euclidean ball

After checking the influence of the size of the allowed neighbourhood of x on the accuracy, we focus on the iteration number in the gradient ascents.

2.1.2 Influence of the number of iteration n in the gradient ascent

The number of iterations n in the gradient ascent that generates the adversarial examples should lead to examples close to the optimal solutions (local maxima).
The figure 3 represents the accuracy as a function of the number of iterations n. Indeed, the accuracy is lower for adversarial samples generated with a high number of gradient steps since it allows the algorithm to converge. The convergence toward local maxima is observed after approximately 7 iterations.

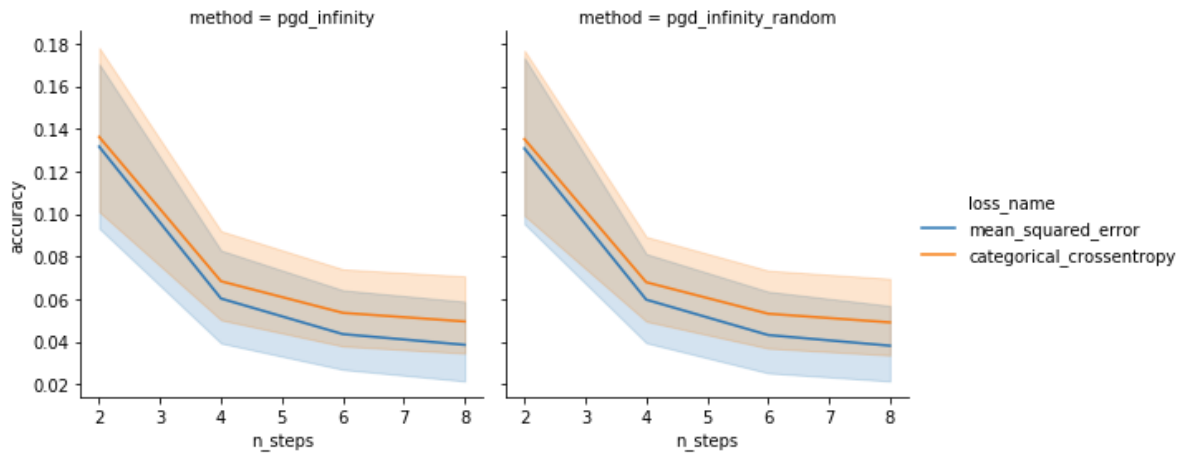

Figure 3: Accuracy of f_θ with respect to the radius ϵ of the ℓ_∞ ball

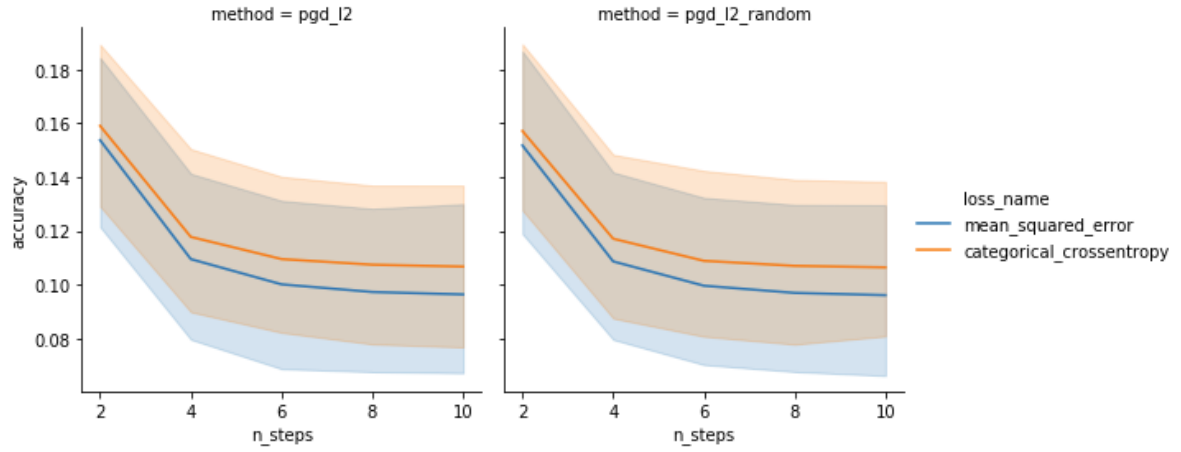

Figure 4: Accuracy of f_θ with respect to the radius ϵ of the ℓ_2 ball

Another important point is that the attacks perform better when switching the loss. Using the *mean square error*, the generated attack samples fool the model f_θ better than with the true loss function (*cross-entropy*) used to train the model. This might be explained by numerical reasons.

2.1.3 Influence of the step size η in the gradient ascent

In this paragraph, the accuracy of f_θ is seen as a function of the fixed perturbation η performed at each gradient step. One can consider η as the step size in the associated gradient ascent. The figure 5 presents the accuracy variations with respect to η.

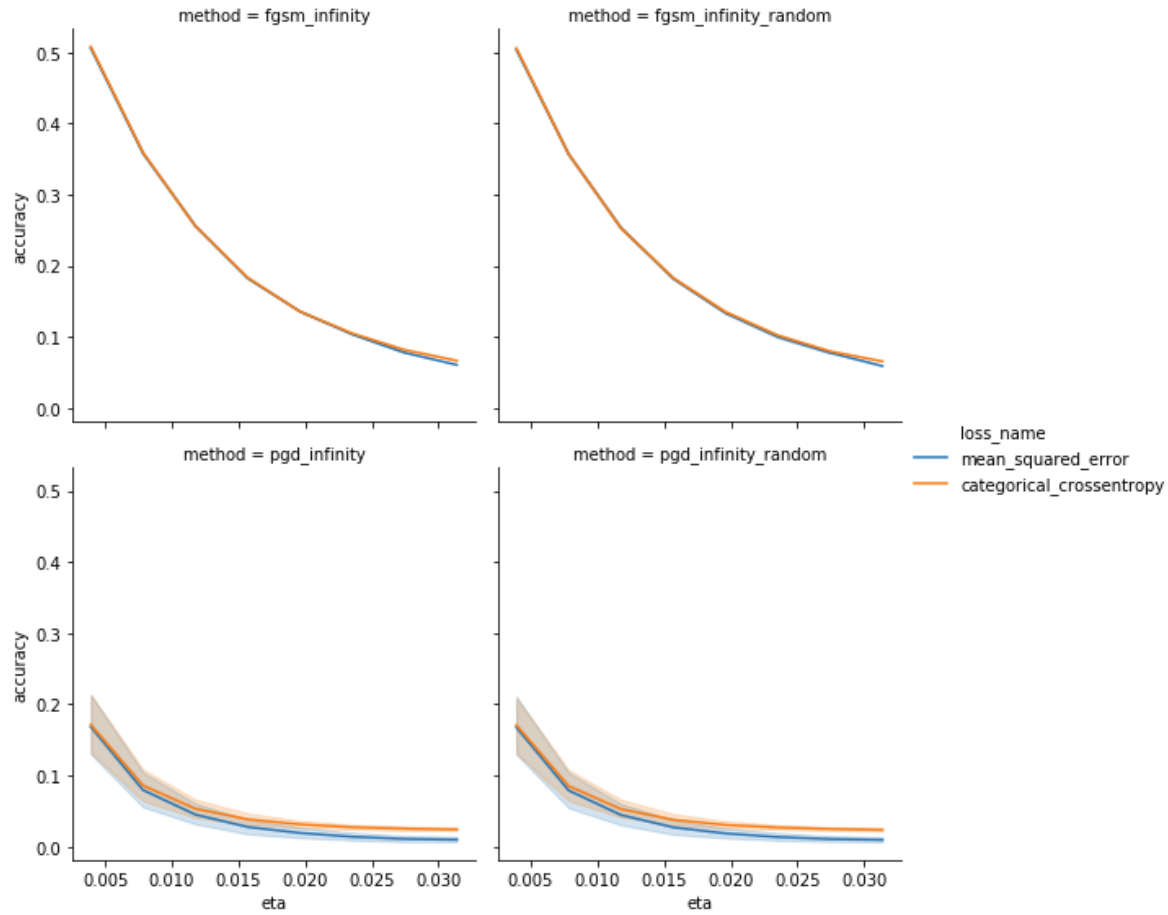

Figure 5: Accuracy of f_θ with respect to the perturbation size η during the generative process.

The accuracy is indeed decreasing when increasing η, but one can observe performances are not really better after a value of $\eta = \frac{4}{255} = 0.157$ meaning perturbating the RGB image by 4 color intensity after each step. It is interesting to note that even for big steps such as $\eta = \frac{8}{255}$ the method still works thanks to the projections.

The following figure illustrate the same relation when the projection is done on an euclidean ball.

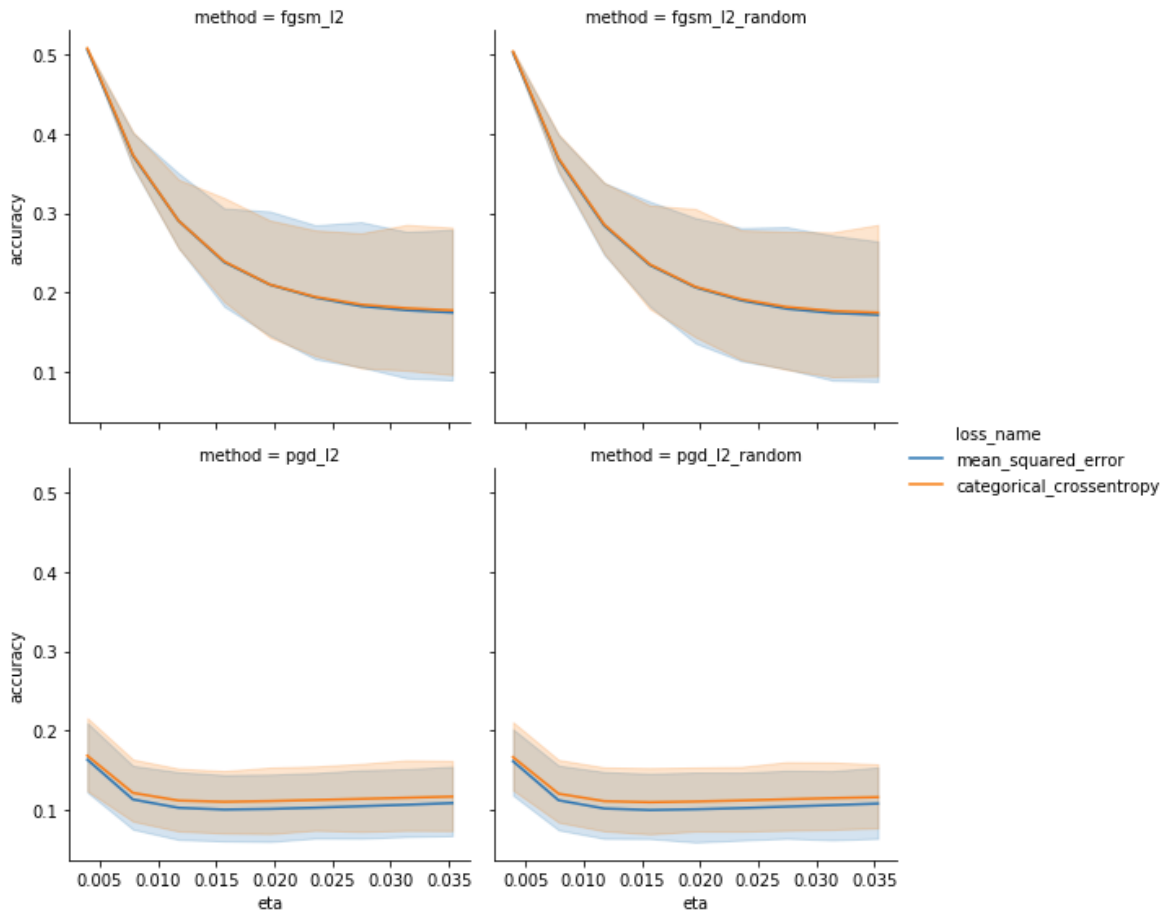

Figure 6: Accuracy of f_θ with respect to the perturbation size η during the generative process with an euclidean ball.

Recall the range of η values in the figure 6 is the same as in the ℓ_∞ case case. In other words, pixel perturbations allowed in the infinite ball grid-search are the same in the euclidean case.

Thus, same pattern are observed in the ℓ_2 case. However, larger confidence intervals are encountered in the figure 6. The *mean-square error* is slightly better when generating attacks.

2.2 Grid-search results

In this part, we present the results of our grid search for a fixed constraint $\epsilon_\infty = 8$ in the infinite case. For the euclidean case, we show the results for a value $\epsilon_2 = 4\sqrt{p} = 0.86$ in order to use a value close to what it is given in [3].
Results of the top ten ℓ_∞ and ℓ_2 bounded attacks ranked by accuracy are presented in tables 1 and 2.

ω	η	n	L	Loss	Accuracy
ω_{rand}	2	8	*mean-squared error*	9.127419	0.0440
ω_0	2	8	*mean-squared error*	9.106970	0.0441
ω_{rand}	3	8	*mean-squared error*	8.931167	0.0451
ω_0	3	8	*mean-squared error*	8.920773	0.0461
ω_0	3	6	*mean-squared error*	8.770875	0.0479
ω_{rand}	3	6	*mean-squared error*	8.794707	0.0484
ω_{rand}	2	6	*mean-squared error*	8.812643	0.0492
ω_{rand}	2	8	*cross-entropy*	10.135644	0.0493
ω_0	2	8	*cross-entropy*	10.122940	0.0493
ω_{rand}	4	8	*mean-squared error*	8.376781	0.0496

Table 1: Top 10 best grid-search configurations ℓ_∞ bounded attacks

From table 1, random-start is slightly better than deterministic start because six out of ten attacks used a random initial perturbation of the images as well as the best attack use a random start. Moreover, using the *mean-squared error* provides better attacks than using the loss used to train the reference model. More steps during the gradient ascent allow being closer to one of the local maxima, hence lower under-attack accuracies are obtained with the most iterations provided in the parameters grid ($n = 8$). Optimal gradient step sizes is $\eta = 2 = \frac{\epsilon}{4}$ for the best two attacks.

ω	η	n	L	Loss	Accuracy
ω_0	2	8	*mean-squared error*	10.040753	0.0346
ω_{rand}	2	8	*mean-squared error*	10.062221	0.0353
ω_0	3	8	*mean-squared error*	9.785490	0.0373
ω_{rand}	3	8	*mean-squared error*	9.830394	0.0375
ω_0	2	6	*mean-squared error*	9.811090	0.0375
ω_{rand}	2	6	*mean-squared error*	9.846786	0.0381
ω_{rand}	3	6	*mean-squared error*	9.704010	0.0386
ω_0	1	8	*mean-squared error*	9.895251	0.0386
ω_0	3	6	*mean-squared error*	9.659264	0.0386
ω_{rand}	1	8	*mean-squared error*	9.936613	0.0387

Table 2: Top 10 best grid-search configurations ℓ_2 bounded attacks

According to the results presented in 2, the random start is not the best method anymore but just behind the best method which is deterministic. Same values for the step size η as in the ℓ_∞ case provide the best scores.
In both ℓ_∞ and ℓ_2 settings, using the *mean-squared error* is better than the *cross-entropy*.

All results and conclusions in this grid-search are relatively standard, they provide a protocol to find the best attacks and understand univariate relationships between hyperparameters and under-attack accuracy. One interesting thing is the efficiency of the *mean-squared error* over the *cross entropy*.

2.3 About the ball radius choice ϵ in ℓ_∞ and ℓ_2

Some problems where encountered during the grid-search because of inequalities related to ℓ_∞ and ℓ_2 metrics. At first, we kept the radius ϵ_2 of the ℓ_2 ball equal to the radius $\epsilon_\infty \leq 9$

of the ℓ_∞ ball. Once we checked the ℓ_2 results for this setting, we realized it is not the optimal choice. Indeed, norm of vector are different when changing metric.

When considering a perturbation of size η and taking one step of gradient ascent from x, a new image $x^1 = x + \eta \operatorname{sgn}(\nabla_x L(\theta, x, y))$ is obtained. However, the infinity norm of this perturbed image is always smaller than its euclidean norm.

More generally, let $x^* \in \mathbb{R}^p$ such that $\|x - x*\|_\infty = \epsilon$, then $\epsilon \leq \|x - x^*\|_2 \leq \sqrt{p}\|x - x*\|_\infty = \sqrt{p}\,\epsilon$. So, a point located on the boundary of the ℓ_∞ ball of radius ϵ_∞ is at most at a distance of $\sqrt{p}\,\epsilon_\infty$ from the barycenter when considering the euclidean norm.

Similarly, taking a step of size η in the projected gradient ascent lead to a new point having a distance of $\eta\sqrt{p}$ from its previous state. Recall that $p = 3072$ when images are draw from CIFAR10.

Consequently, sign-perturbed images have a way greater ℓ_2 norm than ℓ_∞. For exemple, if an image x^1 is at a distance of η from the input

Therefore, it is required to increase the radius ϵ_2 of the ℓ_2 ball since we want to keep the same step size magnitude in both cases (otherwise the perturbations are negligible, and the local maxima is not reached). Since switching of metric without modifying epsilon (the radius ϵ_2 of the ℓ_2 is the same as the radius ϵ_∞) will lead to projections too close to the initial image.

As a result, we choose $\epsilon_2 = \sqrt{p}\,\epsilon_\infty$ when computing ℓ_2 bounded attacks.

Note that ℓ_2 bounded attacks are treated in [3] where they proposed a value $\epsilon_2 = 0.8 \times 255 \simeq \frac{\sqrt{p}}{2}\,\epsilon_\infty$ for the common $\epsilon_\infty = 8$. In the following figure, we provided a comparison between images of ℓ_∞ and ℓ_2 attacks generated using the $\epsilon_2 = \sqrt{p}\,\epsilon_\infty$ (and $\eta = 1$, $n = 20$, $L = \textit{cross-entropy}$).

From images of the figure 7, it is hard to detect differences between ℓ_∞ and ℓ_2 attacks. Re-scaling the radius by $\sqrt{p}$ might be a good-practice for switching of norm when dealing with unknown data of very large dimension.

To conclude this part, the first section allowed to understand the problem and the influence of the different hyperparameters while checking our implementations are well-designed. The euclidean case is highlighted because it is challenging to link both approach and understand the specificity of the ℓ_2 configuration. In the next section, analysis of the architecture of different models f_θ are performed to observe multiple black-box attacks and model robustness.

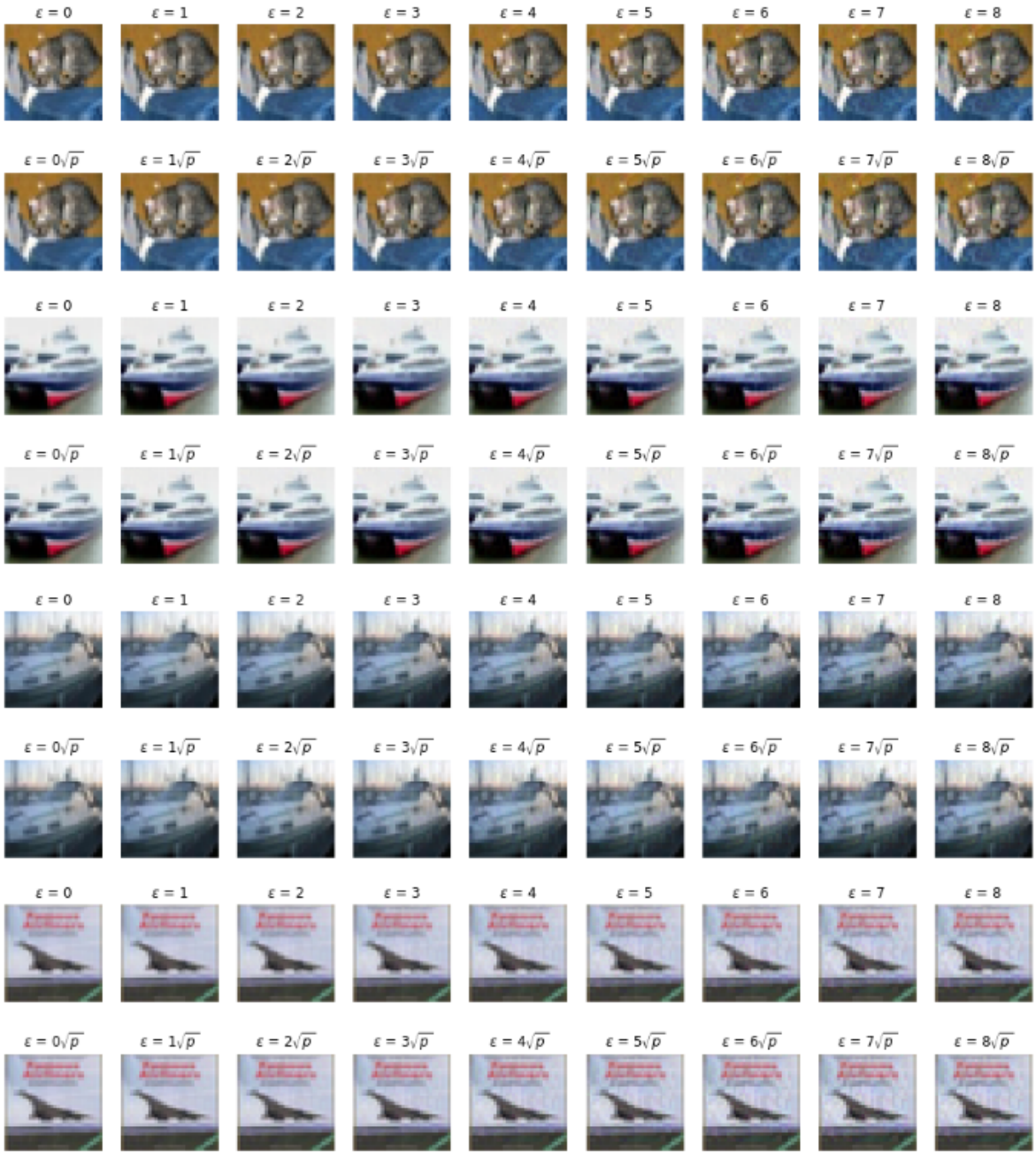

Figure 7: Plot of multiple adversarial images generated by ℓ_∞ and ℓ_2 bounded projected gradient ascent. The rescaling factor $\sqrt{p}$ is applied in the euclidean case.

3 Black box attacks: analysis of convolution neural networks dimensions

In this section, we propose an analysis of the robustness of our reference network against black box attacks for Convolutional Neural Networks (CNN). We will try to understand how robust is our reference model according with respect to the depth and the width of the attacker CNN.

3.1 Influence of CNN's depth on a black box attack

In this part, we will try to understand if the variation of the depth of a CNN influence an attack. In other words, we are trying to understand how robust our reference network is with respect to the depth of an attacker CNN.

Models	Model1	Model2	Model3	Model4	Reference
Convolutional layers	1	2	3	4	3
Trainable parameters	3,690,058	822,922	184,010	69,386	635,786
Test set accuracy	0.6365	0.7057	0.7211	0.6881	0.7481

We trained 4 CNN with 1 to 4 layers with the same constant number of filters (256 convolutional filters). For each of these CNN, we attack the CIFAR-10 test set and we evaluate the loss and accuracy for the initial model.
The training plots are given in Figure 8 and Figure 9 below.

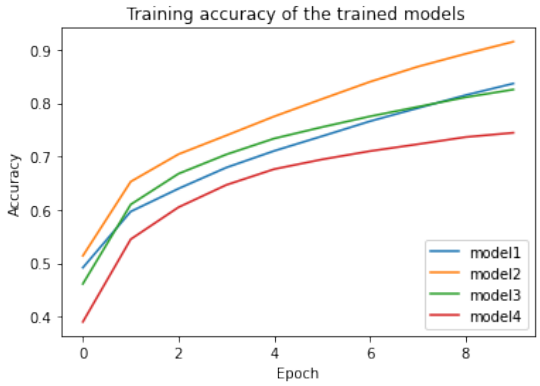

Figure 8: Accuracy evolution with respect to the width

Figure 9: Loss evolution with respect to the depth

The model of reference contains three convolution blocks with ReLU and batch-normalisation with respectively 64, 128 and 256 convolutional filters. Again, average pooling is done before flattening the network. This model has a total of 635786 parameters.

3.1.1 FGSM black box attacks

With the different models that we defined previously, we attacked CIFAR-10 with the FGSM method. Then, we tested these attacks using our reference model. For the FGSM attacks, we defined our parameters as follows: $\eta = \frac{2}{255}$.
The results of our experiments are summarised in the following figures.

Figures 10 and 11 display respectively the accuracy and the loss of the evaluation of the different attacked test sets. However, these results does not enable to conclude.

3.1.2 PGD black box attacks

In this section, we will present the same experiment as the previous section but with a PGD attack with the following parameters: $\epsilon = \frac{8}{255}$, $\eta = \frac{2}{255}$ and $n = 50$ where n is the number of iterations.
Figure 12 and Figure 13 below show our results.

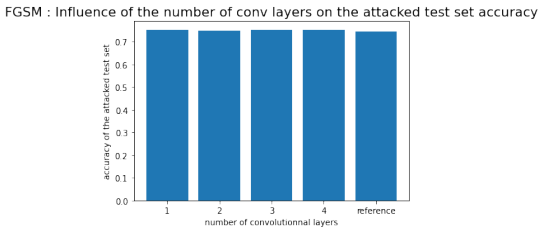

Figure 10: Accuracy evolution with respect to the depth

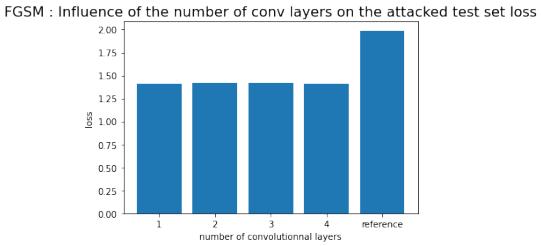

Figure 11: Loss evolution with respect to the depth

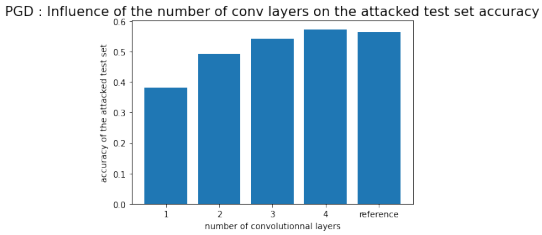

Figure 12: Accuracy evolution with respect to the depth

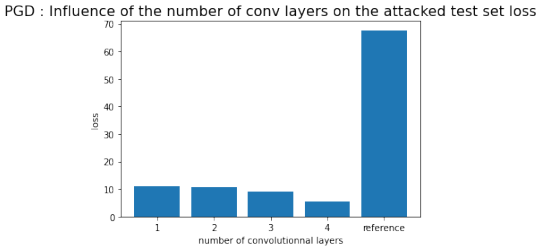

Figure 13: Loss evolution with respect tot the depth

In Figure 12, the accuracy is lower for the models with less convolutional layers, which means that the attacks are stronger. This results might be explained by the number of parameters. Indeed, for models that have less convolutional layers, the number of parameters is very high. The more the networks as convolutional layers, the less it has trainable parameters.
These results are surprising, as we can see that the networks with low numbers of convolutional layers are also the one that are the less accurate in predicting the initial test set.

3.2 Influence of CNN's width on a black box attack

In a second phase, in order to study the influence of the width of CNN on an attack, we repeated the same experience with different numbers of filters by convolutional layer. In this way, we trained 7 CNN, with convolutional layers 3 layers, each layer having the same number of filters. For each of the CNN the number of filter varies from 8 to 512.
The table below describes the models that we have been using and gives the number of trainable parameters as well as the accuracy for the test set.

Models	Model1	Model2	Model3	Model4	Model5	Model6	Model7
Filters	8	16	32	64	128	256	512
Trainable parameters	21,890	41,978	85,226	184,010	430,730	1,120,778	3,287,306
Test set accuracy	0.5705	0.6491	0.7063	0.7246	0.7242	0.7365	0.7340

The table highlights the fact that the number of trainable parameters increase with the width of the network.
With each of the trained models, we attacked the CIFAR-10 test set with both a FGSM and a PGD attack and evaluated the loss and accuracy of these attacked sets with the our reference model afterwards.

Also, Figure 14 & Figure 15 display the training phase of the networks as well as their evaluation on the test set.

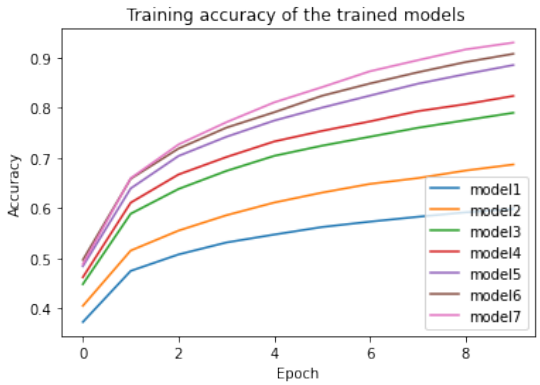

Figure 14: Accuracy evolution with respect to the width

Figure 15: Loss evolution with respect to the depth

3.2.1 FGSM black box attacks

For the FGSM attacks we used a parameter η=2/255.
The results are displayed in the figure below.

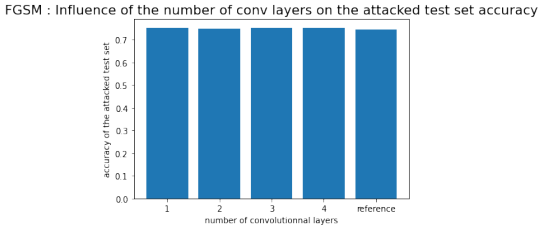

Figure 16: Accuracy evolution of model large

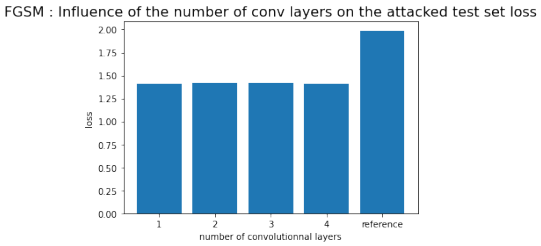

Figure 17: Loss evolution of model large

The number of filters of the attacking networks seems to not impact the accuracy of the attacked test set. Hence, our results does not allow us to conclude that the width of the attacking network have an impact the potency of a FGSM attack.

3.2.2 PGD black box attacks

For the PGD attack we used the same parameters as defined in the previous section: $\epsilon = \frac{8}{255}$, $\eta = \frac{2}{255}$ and $n = 50$ where n is the number of iterations.
The results are displayed in Figure 18 and Figure 19 below.

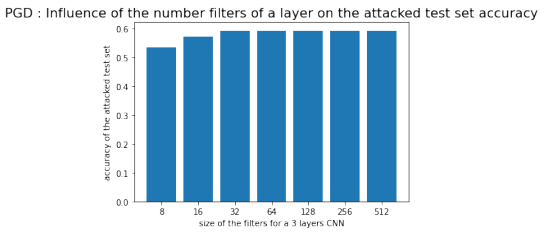

Figure 18: Accuracy evolution with respect to the width

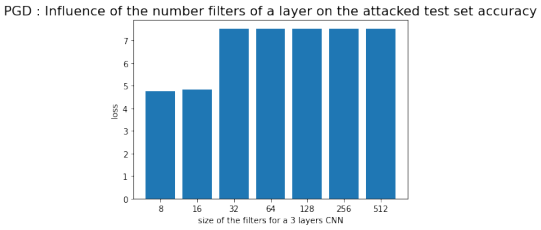

Figure 19: Loss evolution with respect to the depth

Figures 18 & 19 highlight the fact that the attacks with less large networks are a little stronger than the attacks with larger network. Indeed, we can see that for the model 1 & model 2, that have respectively a 8 and 16 filters have a lower accuracy.
However, these results are surprising, as the number of parameters increase with the width of a network. The networks that attack the most efficiently are also the less accurate and the one with the less trainable parameters.

3.3 Conclusion and hypothesis

In this section we have tried to understand the behavior of black box attacks with respect to the depth and the width of the attacking Convolutional Neural Network. The results seem to show that the less deep networks are better attacking. Also, the best attacking Convolutional Neural Network seem to be the less large.
Although, it appears that the less deep network have a very high number of trainable parameters and are more likely to over-fit (Figure 8 and Figure 9), while less large networks have a very low in trainable parameters and seem to not over-fit (Figure 14 and Figure 15). Hence, it may be interesting to investigate the results of over-fitting for attacking networks.

3.4 Attempt to visualize the adversarial attacks

In order to understand how an attack impact the prediction of a model, we implemented a Gradient-weighted Class Activation Mapping (GRAD-CAM)[4]. This tool is very interesting because it gives visual explanations from Deep Networks via Gradient-based localization.
To obtain the class-discriminative localization map, Grad-CAM computes the gradient of the target class with respect to feature maps of a convolutional layer. These gradients flowing back are global-average-pooled to obtain the importance weights. Finally, a ReLU layers allow to keep the weights that are higher than 0.
The figure below describes the GRAD-CAM pipeline.

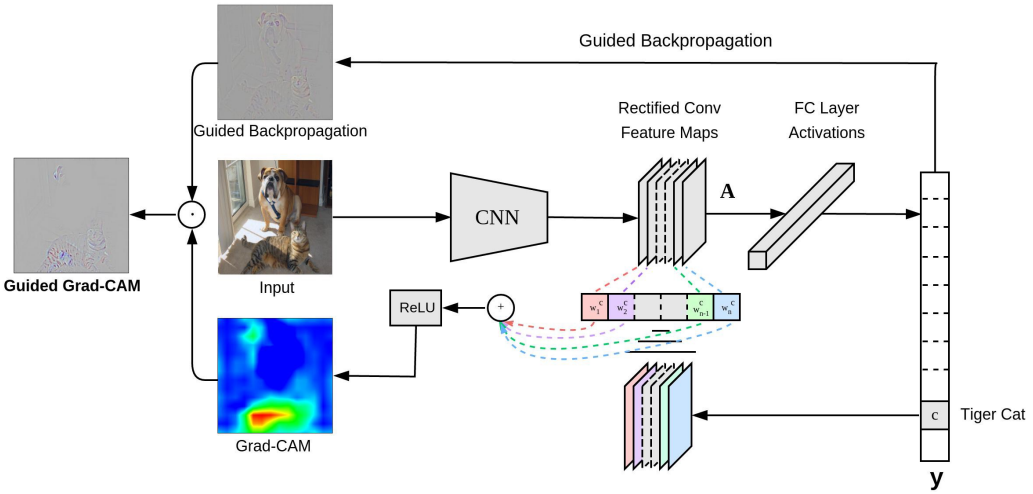

Figure 20: GRAD-CAM pipeline of a Convolutionnal Neural Network

This visualization tools has been very useful in understanding how an attack is able to trick the model.

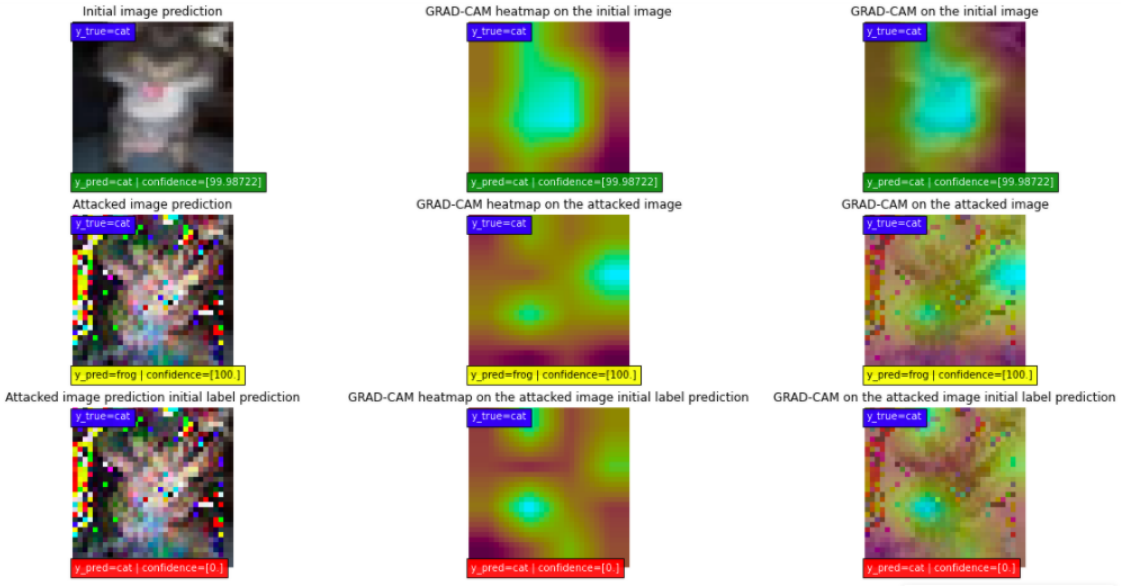

Figure 21: GRAD-CAM pipeline of a Convolutionnal Neural Network

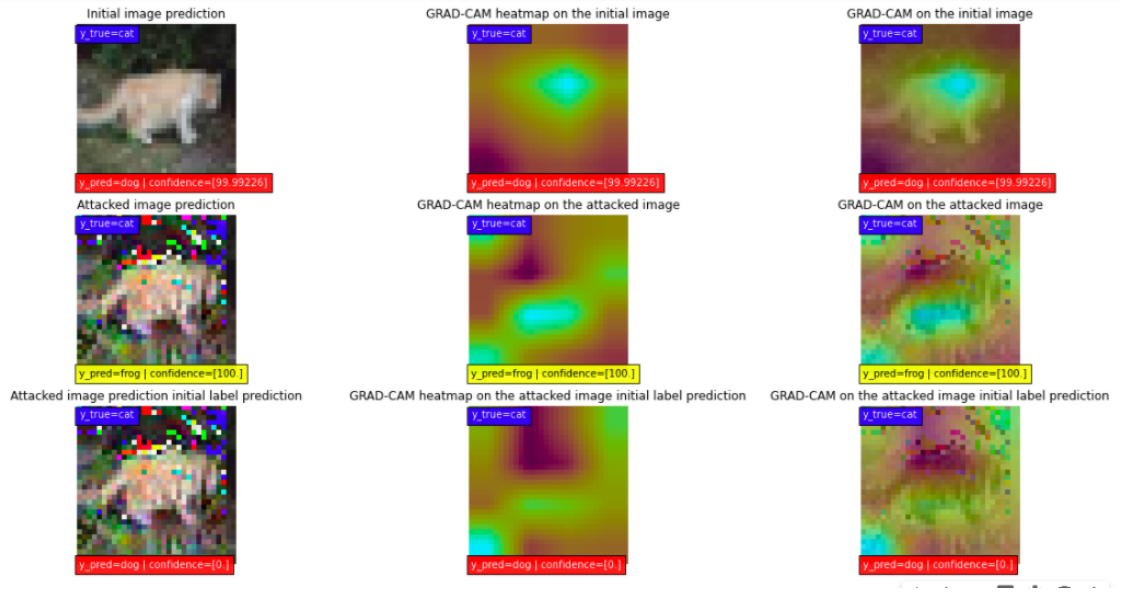

Figure 22: GRAD-CAM pipeline of a Convolutionnal Neural Network

Figure 21 and Figure 22 highlight how adversarial attacks impact the decision making of a network. Indeed, an attack will completely change the shapes and the features that help in making the decision to assign a label to a picture.

4 Biais Variance trade-off: Impact on classifier robustness

4.1 Motivation

In this part, we have worked on algorithm robustness based on training settings. Indeed, we have the intuition that an algorithm which overfits on training data (parameters estimater has a low biais and high variance) will be more easily attack-able compare to the same algorithm without overfiting (estimater with a higer biais and lower variance compare with the previous one). We taught that an overfitted model will perform badly on attacked data

points (those are bounded pixel permutation on images) since those will differ from the input training distribution.

4.2 General setup

In this project, we have as an instruction to use a particular neural network architecture to build our classifier. It is composed with tree blocks of (Convolution, Relu, Maxpooling) layers, one AveragePooling layer, one dense layer with a relu activation and a last dense layer (corresponding to class probabilities if a softmax activation function is applied). However, we were free to chose any other parameters/hyper-parameters settings that we wanted (number of parameters, number of filters, kernel size, optimizer, learning rate, ...).

Since we were studying here the relationship between biais variance trade-off and classifier robustness, our task was first to found a classifer with a high variance and used techniques in order to reduced it. We have used two. First, a "early stopping callbacks". It will stop the training every time the model does not improve the validation loss during a fix number of epochs. Second, added a dropout layer after each convolutional layer. Thanks to this, some input units will not be considered during the training, and will help to improve model generalisation.

After founding several networks architectures, we then have to attack those classifiers to challenge their robustness. For that purpose, we have implementing PDG l_∞ attacks. Indeed, thanks to [2], we can formalized our problem with to the following saddle point problem:

$$\min_\theta \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta\in\mathcal{S}} L(\theta, x+\delta, y)\right] \tag{3}$$

This equation can be viewed as a composition of an inner maximization problem (attack side) and an outer minimization problem (defense side). Here we want to find the best attack for each classifier that we have built (so we will focus on the inner maximization problem only). [2] have shown that the loss has multiple local maxima in the l_∞ ball (with respect to the image input), but all this points have approximately the same loss value. In fact, they say that we can not find any global maximum (if their exist one) with first order attacks (attack that require access to gradient at most). In conclusion, they have shown that PGD attacks were the most efficient first order attack, and so that a classifier robust to those will be robust against any other first order attacks (FGSM, Carlini and Wagner attack,...).

4.3 Implementation

4.3.1 Natural sets metrics

The first configuration that we have implemented is composed with 439 178 parameters: 64, 128 and 256 filters for each convolutional layer respectively, and 256, 10 neurons for the two dense layers.
As we can see on the picture below, the model is clearly overfiting since the loss over the validation set starts increasing at some step (that's what we wanted). We will refered to this model later as "model 64 128 256 20e" ("20e" for 20 epochs).
As explained before, we have implemented on this architecture two methodologies to reduce overfitting. We obtained two models, the first one is trained on 7 epochs only (will be refered as "model 64 128 256 7e") and the second one is trained on 17 epochs with dropout layers (called "model 64 128 256 dropout 17e").

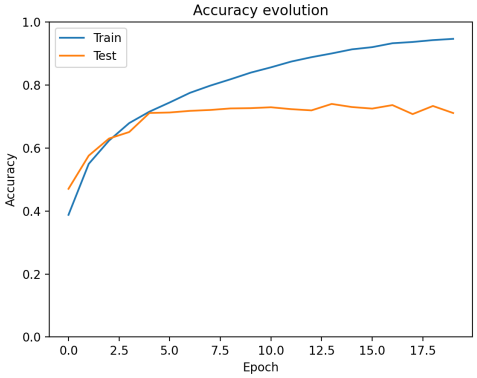

Figure 23: Accuracy evolution of model large

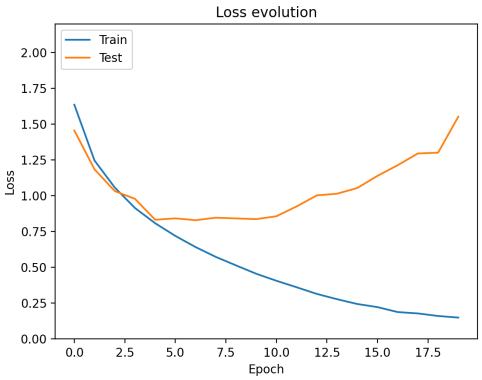

Figure 24: Loss evolution of model large

An other way of increasing biais and decrasing variance is to reduce the number of parameters. We have chosen to divided the number of filters/parameters by two in each layer. At the end, we obtained tree architectures with 111 050 parameters each, that have 32, 64 and 128 filters for each convolutional layers respectively, and 128, 10 units for the two dense layers. The first one is trained on 20 epochs (and is still overfiting) (will refered to it as "model 32 64 128 20e"), the second one on 14 epochs (named "model 32 64 128 14e"), and the last one trained on 14 epochs with dropout layers (called "model 32 64 128 dropout 14e").
Here is a tabular composed with metrics on train and test sets for each of these classifiers.

	loss train	accuracy train	loss test	accuracy test	mean conf test
model 32 64 128 20e	0.33	88%	0.97	71.3%	0.82
model 32 64 128 14e	0.44	84%	0.86	71.5%	0.79
model 32 64 128 14e drop	0.60	80%	0.80	72%	0.67
model 64 128 256 20e	0.19	93%	1.58	70%	0.90
model 64 128 256 7e	0.56	80%	0.86	71.2%	0.76
model 64 128 256 17e drop	0.31	91%	0.72	75%	0.77

Table 3: Metrics on classics sets

We can see from this, that our techniques to reduce overfitting worked well since we have obtained classifiers with better performances on test set. We can also observed that adding a dropout layer can reduce estimator variance but also decrease biais (performances on train set are better).

4.3.2 Robustness against adversarial attacks

As said before, we have implemented PGD attacks in the l_∞ ball for each of classifiers introduced in previous section. We have chose a step of 0.011 (permutation of 3/255) in a ball of radius 0.031 (variation of 8/255). We have been iterating until convergence to a maximum local: it means that we have stopped the method only when loss value does not variate more than a constant $\alpha = 0.01$ in this experience.

As we can see on figure 25 that number of iterations required before converging are not equal between all models. In addition, we see that both overfiting models (orange and blue curves) are ones that have the higher loss on adversarial examples. We can conjecture that

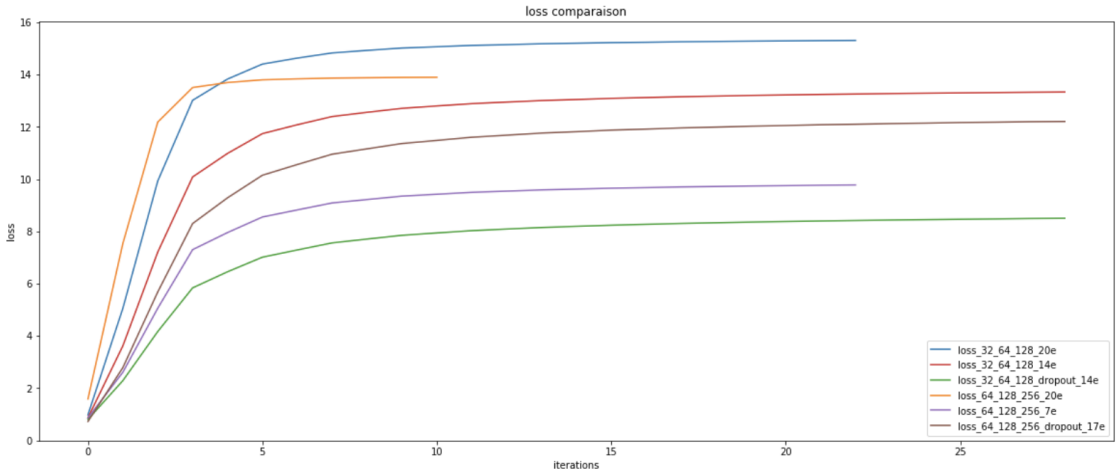

Figure 25: Loss evolution against PGD attacks

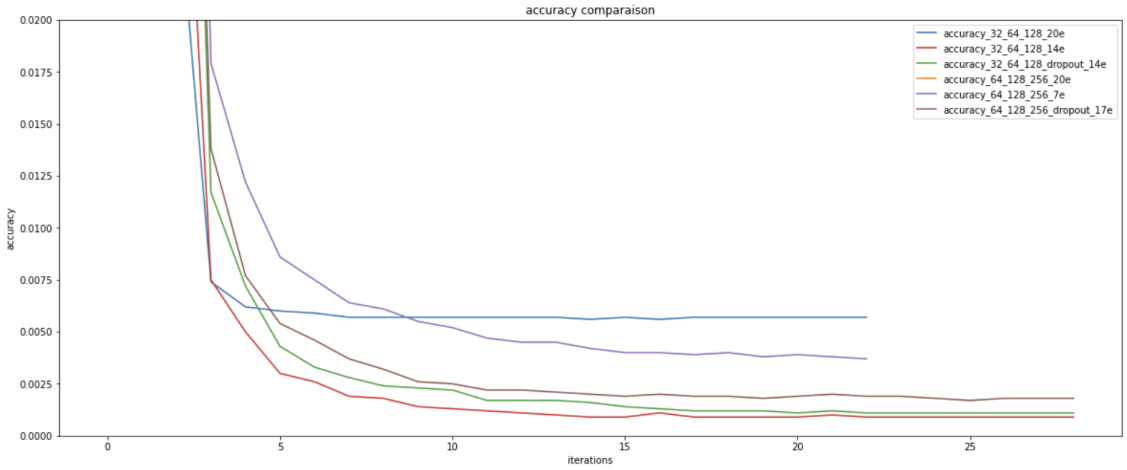

Figure 26: Accuracy evolution against PGD attacks

overfiting models are less robust against the related loss problem (here it is the "log loss").

We have also plotted accuracy evolution for each model at each iteration of PGD attacks (cf figure 26). We have not included the accuracy of "model 64 128 256 20e" for visualisation purpose. Indeed it has an accuracy near 0.013 which is much higher compare to other models. We can noticed that the two models that overfit more on training data are more robust to adversiaral attack with respect to classification accuracy. We also see that architectures with more parameters are more robust (its have a higher accuracy). We can conjecture that adding parameters, and overfiting data points are two ways of increasing robustness against first order attacks.

4.3.3 Results interpretation

As a first impression, we can think that our results are opposed. Indeed, both models trained on 20 epochs have a higher loss and higher accuracy on adversarial examples compared to others. Our explanation to this phenomena is related with model confidence in predictions. We see in table 4.3.1 that those two models are ones with higher confidence in their predictions on average. With the log-loss, we can then increase loss values without changing predictions. For example, if a classifier predicts a correct label with high confi-

dence, we can increase loss value by just decreasing confidence and without changing the correct prediction. We think that this is the main explanation to this phenomena. We concluded to this experience that overfiting helps for being robust and do not validate our intuition at first (easier to attack a model that overfits).

To go further, it can be interesting to search for another architecture and find one where the model perform well with same performances on train, validation and test datasets. Indeed, all these previous models were still overfiting: performances on train set are much better compare to those on validation set. There is a trade-off on being more efficient on test set and more sensible on some adversarial attacks and be less efficient on test set but more robust to adversarial attacks.

5 Defending against adversarial attacks

In this section, we study two defensive techniques of CNN against adversarial attacks, the first one is adversarial training, and the second one is noisy training.
The general idea was to study the defensive capabilities of neural networks trained recursively on training datasets enriched with adversarial images and compare this method with neural networks trained on datasets enriched with noisy images.

5.0.1 Adversarial Training

Adversarial training consists in training a CNN in batch enriched with adversarial images. It aims to solve the saddle-point problem in virtue of Danskin's theorem [2].

5.0.2 Introduction to Quantization

A major part of white box adversarial attacks are based on gradient calculation or exploit the linear extrapolation behavior of machine learning models by studying the response of the system to perturbation of inputs. Input discretization has been introduced by (Xu et al., 2017), each pixel value is hashed into a low-bit version of its original value. Thus, the discretization masks the gradient.
This discretization is used at the preprocessing step and does not contain learnable parameter.
In this study the preprocessing discretizating layer used is composed of a tanh followed by a function bucketizing the input into 8 buckets of the same size.

5.0.3 Experiments

To understand defensive capabilities of adversarial trained neural networks we decided to study a PGD adversarial attack using infinity norm with eta = 0.1 and epsilon = 0.05 (considering a colour channel is in [0, 1]), the number of epochs have been set to 8. Eta and the number of epochs have been chosen following previous studies and epsilon has been chosen in order to have a realistic attack. We studied adversarial training in the first time then tried a basic quantization method to hide the gradient of the network and finally tried to combine the two methods.

5.0.4 Reference model

The reference model used in this section is a CNN with dropout and batch normalization layers in addition to a L2 Regularization of 1e-4 (see figure below). It has been trained for 20 Epochs with a batch size of 128 and displays an accuracy on the validation set of approximately 78%.

5.0.5 Defensive Networks

Firstly, the model studied are the followings:

- Reference Model: reference model trained for 10 epochs on the CIFAR10 Training set

- Quantitized: reference model with a quantization layer added as a preprocessing step, trained from scratch for 20 epochs. PGD attack has been conducted excluding the preprocessing layer that prevents us from calculating a gradient.
- AdvTrain: reference model adversarialy trained in batch with training batches consisting in 50% or original images 50% of adversarial images randomly sampled within the batch
- Quantitized & AdvTrain 0: reference model adversarialy trained with a quantization layer. PGD attack has been conducted excluding the preprocessing layer that prevents us from calculating a gradient.

5.0.6 Results

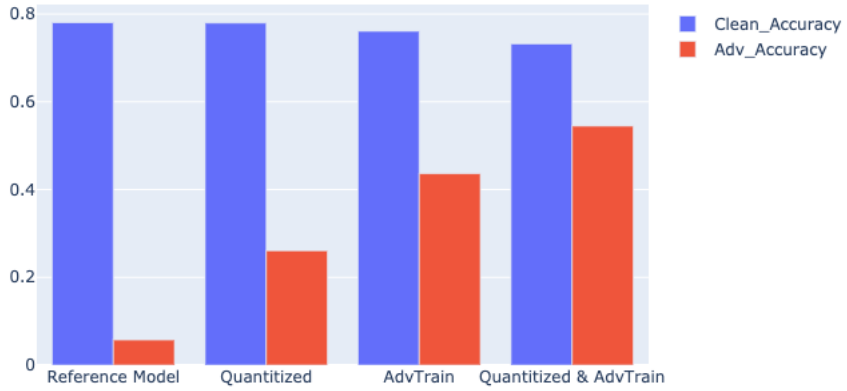

Figure 27: figure
PGD Adversarial Training Accuracy Comparison Extended

5.0.7 Analysis

Firstly, all the defensive models present, as expected, a decrease in accuracy compared to the benchmark on the clean validation dataset.
Then, we notice that we have a compromise between accuracy on clean dataset and accuracy on adversarial datasets. We conclude by selecting the Quantitized & AdvTrain as our best defensive model as the decreases in accuracy on the clean validation set seem to be largely offset by the increase of accuracy on the adversarial validation set.

6 Conclusion

Through this document, the efficiency of projected gradient based attacks have been studied from both white box and blackbox approaches. Relationship between dimensions of models and attacks accuracy have been highlighted. It is worth to note how overfitting might affect robustness of a basic classifier against adversarial examples. It could be relevant to make those experiments with images of higher dimension. As a side note, we used

Google Colab GPUs to perform computations. A simple adversarial attack of 10000 examples is achieved in about 5 seconds using tensor methods (processing the whole batch), while it takes 20 minutes when generating examples iteratively (image by image).

References

[1] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2014. arXiv: 1412.6572 [stat.ML].

[2] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2017. arXiv: 1706.06083 [stat.ML].

[3] Alexandre Araujo et al. *Robust Neural Networks using Randomized Adversarial Training*. 2019. arXiv: 1903.10219 [cs.LG].

[4] Ramprasaath R. et al. *Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization*. 2019. URL: https://arxiv.org/pdf/1610.02391.pdf.