INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE RENNES
Department Génie Mathématique
Promotion 2022

Interdisciplinary project report

Detecting and Defending against Adversarial Examples in Deep Neural Networks

made by

Anna Kerebel, Florian Jacta, Rose Guionnet,
Swen Laurent, Théo Di Piazza and Victor Klötzer

Referents:

Ayse Nur Arlsan
Wassim Hamidouche

Link to the Python notebook used for this project on a GitHub repository:

https://github.com/VictorKlotzer/Detecting_and_defending_against_adversarial_attacks

Rennes, December 18, 2020

Abstract

Because it represents a powerful tool, image recognition is a subject of scientific study in many fields: medical, military, transports... Unfortunately, this recent technology still suffers from significant weaknesses: pictures can be attacked by adversarial processes in order to fool the system and give a wrong result. Consequences of such attacks could be serious (wrong person detected by a facial recognition algorithm for instance or even erroneous analysis of road signs recognition algorithms...). The problem to solve is thus to be able to detect these attacks in order to counter them in the future. In that respect, the objective is to create a detector which uses different machine learning processes. The project focuses on FGSM, PGD and DeepFool adversarial attacks. Then, using the MSCN coefficients which allow to transform a picture in a numeric table, several statistical tests as the Kolmogorov-Smirnov test can be operated on the images. Drawning on these statistical tests, the detector presented is finally able to detect PGD and FSGM attacks but still has difficulties with DeepFool attacks. In the end, a comparison with an SVM classifier already implemented in Python is made: SVM classifier gives much faster and more accurate results. This observation highlights that research may go further: improvements could still be made in order to boost our algorithm's efficiency.

Contents

1 Introduction

Machine learning or how to teach a computer is an AI's study field, which is based on mathematics and statistics. The objective is to give the possibility to computers to learn with data in order to help them to solve problems, even those they are not familiar with, and improve their performances. Usually, machine learning deals with two phases. The first one is the learning (or training) part, the objective is to estimate a model based on data we have in a finite number, called observations, e.g. estimating a mathematical function. The second one is the deployment part. The model has been determined so new data can be implemented in the algorithm in order to obtain an appropriated result to the asked task. Also, some machine learning algorithms can keep learning while they are in the deployment phase.

We decide here to focus on machine learning in the image detection field. Actually, using a considerable amount of pictures, we can teach a computer, to distinguish a cat picture from a dog picture for instance. This classification can be done by deep neural networks. But there is one problem: the classification of images is not robust. That is to say that it is very easy to commit a clever tiny modification to an image, so that even a well-trained deep neural network will misclassify it. Even if there is no difference for a human eye with the addition of such a small noise, the algorithm will give a completely different result. Those attacks are called adversarial attacks. And in applications like road signs recognition for autonomous car driving, the consequences of such an attack could have dramatic consequences.

The aim of this project is to detect these attacks, so that our machine learning algorithm could counter them. We present here the functioning of a neural network, introduce some of the adversarial attacks used and how we can detect them using statistical properties.

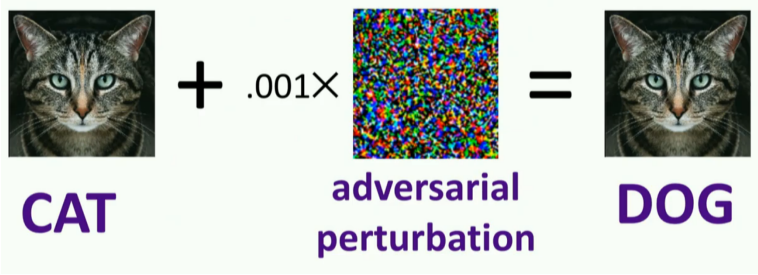


Fig. 1.1: Adversarial attack on a cat. It was classified as a dog!

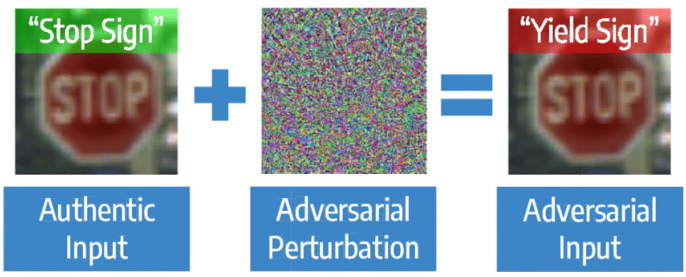


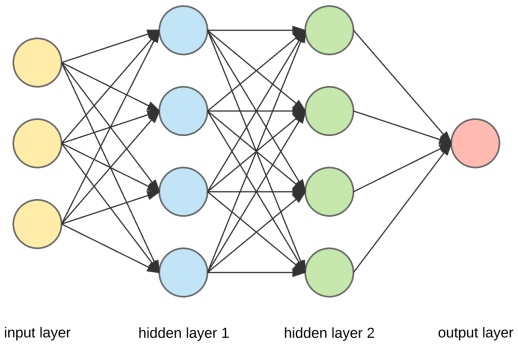
Fig. 1.2: Adversarial attack on a stop sign. It was classified as a yield sign!

2 State of the art

2.1 What's a neural network?

A Neural Network (NN) is a combination of nodes called artificial neurons. An input is given at the beginning of the NN. Each node can then transmit the signal to other neurons through the different layers. The transmission of the signal is possible thanks to the different edges simulated by a combination function : typically a scalar product.

Then, the NN will learn by minimizing the loss function : a function that measures the difference between the given output and the true input. Then, a back-propagation is used based on the gradient of the loss function to adjust the different weights of the NN.



Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another. The first layer is the input layer and the last, the output layer. Between them, there are what is called the hidden layers.

This structure applies for the majority of the basic Neural Networks however a lot of variations exist that will not be mentioned in this paper except one, particularly useful for images. The Convolutional Neural Network (CNN) is often used for objects recognition because its convolutional aspect lets him process pattern in an image. In the previous basic NN, each neuron in one layer is connected to all neurons in the next layer.

2.2 Adversarial attacks

2.2.1 Aim of a adversarial attack

The goal of an attack is to classify an image in the wrong class with the least modification to the original input. In order to accomplish this goal, a basic way to see it is to maximize the loss function of a modified input $x + \delta$. The problem can be formulated as :

$$\underset{\|\delta\| \leq \varepsilon}{\text{maximize}}\ \ell(h_\theta(x+\delta), y)$$

2.2.2 FGSM attack

As an initial approach, δ can be adjusted in the direction of the gradient called g.

$$g := \nabla_\delta \ell(h_\theta(x+\delta), y)$$

g can be obtained by a simple back-propagation of our model. Then, a step α has to be chosen. He will defined how far the modified input will be in the direction of g.

$$\delta := \alpha g$$

α has to be "big" enough in order to maximize the most the loss function but has also to be as little as we want the modification of the input to be. Here, $\|\delta\| \leq \varepsilon$. As a consequence, a basic idea would be to define δ as :

$$\delta := \epsilon \cdot \text{sign}(g)$$

It would respect the condition $\|\delta\| \leq \varepsilon$ with the ℓ_∞ norm while maximizing δ in this same norm. This approach is called the Fast Gradient Sign Method (FGSM).

2.2.3 PGD attack

The FGSM attack presented above is optimal in a linear case. We did actually consider that in the ball in which the gradient was ascended, the loss-function was linear (which is an acceptable hypothesis in a first approach).

The PGD, for Projected Gradient Descent (even if it should be Ascent here), is as well a non-targeted attack. So the objective of the attack is still the same: maximize the loss-function by committing a tiny modification of the original image:

$$\underset{\|\delta\| \leq \varepsilon}{\text{maximize}}\ \ell(h_\theta(x+\delta), y)$$

In order to refine the search of the maximum of the considered problem and to make more sophisticate than linear, we want to use a projection of the gradient in order to get closer to the maximum of the loss-function. The PGD attack consists in short to iterate multiple FGSM attacks using smaller steps ε.

$$\text{Repeat}: \quad \delta := \mathcal{P}(\delta + \alpha \nabla_\delta \ell(h_\theta(x+\delta), y))$$

where $\mathcal{P}$ is the projection of the considered ball ($\|\cdot\| \leq \varepsilon$).

In order to be not dependent to the absolute value of the gradient, we modify a little bit the used optimization method, and use the so called "(normalized) steepest descent method".

Instead of looking at $z := z - \alpha \nabla_z f(z)$ as in an usual gradient descent, we will sort of normalize the descents (or ascents) the following way: $z := z - \underset{\|v\| \leq \alpha}{\text{argmax}}\ v^T \nabla_z f(z)$. v is chosen such that it maximizes the scalar product between v and the gradient under a size constraint depending on the chosen norm. With the Euclidean norm we get $\underset{\|v\|_2 \leq \alpha}{\text{argmax}}\ v^T \nabla_z f(z) = \alpha \dfrac{\nabla_z f(z)}{\|\nabla_z f(z)\|_2}$, and with the infinity norm we get $\underset{\|v\|_\infty \leq \alpha}{\text{argmax}}\ v^T \nabla_z f(z) = \alpha \cdot \text{sign}(\nabla_z f(z))$.

By re-injecting this formula in the algorithm (and by changing the minus into a plus), we lose thus the too impulsive or not enough impulsive character of the absolute values of the gradient so that we get a much better attack than the FGSM attack (not all the images were misclassified, but lots more than before).

$$\text{Repeat}: \quad \delta := \mathcal{P}\left(\delta + \alpha \cdot \text{sign}(\nabla_\delta \ell(h_\theta(x+\delta), y))\right)$$

With this approach, it seems consistent to choose α as a small fraction of ε, and then choose to do ε/α iterations. We notice again, that in fact the PGD attack just performs multiple mini FGSM attacks.

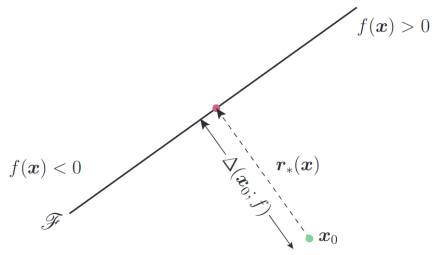
In order to improve a little bit more the performance of this algorithm, one can randomize the initial point in the ball. In this way, one can repeat several time the algorithm beginning with a slightly different δ and choose finally the δ that maximizes the most the loss-function. The improvement is generally not huge compared to the same method without randomization and the time of calculation is of course multiplied by the number of the times the algorithm was relaunched. Nevertheless, this highlights that local optima exist, that those local optima could be reached by performing the PGD attack beginning at $\delta = 0$, and that they might be avoided by using randomization.

2.2.4 DeepFool attack

Deepfool attack for affine binary classifier

DeepFool method, which is a non targeted attack, is efficient against binary classifiers. Hence its goal is to fool the classifier so that it suggests the wrong category for an attacked image (then, in the context of binary classifiers : the other one), while it guarantees the smallest perturbation possible.

Let's consider x_0 as the input, that is to say the original image we want to modify. In order to compute the minimal perturbation to apply on x_0, DeepFool method will search for the orthogonal projection of x_0 onto the hyperparameter plane that separates the two categories as represented below.



Adversarial examples for a linear binary classifier

Where:
- $f(x)$ an affine function which represents the classifier
- x the image considered
- x_0 the initial image
- $r_*(x)$ is the minimal perturbation to apply on x in order to change the classification

The algorithm of DeepFool works as follows:

- Initialisation of $x_i = x_0$, $i = 0$
- While the sign $f(x_i)$ remains unchanged:
    - compute the perturbation $r_i = \frac{-\Delta f(x_i)}{||\Delta f(x_i)||_2^2} f(x_i)$
    - update the new image : $x_{i+1} = x_i + r_i$
- $i = i + 1$
- When the stopping condition is satisfied, the whole perturbation applied is finally : $r_* = \sum_i r_i$

DeepFool attack for affine Multiclass classifier

With multiclass classifiers, the method is quite similar. In this context, each class is related to an hyperplane that divides one class from the others. Now, one does have to find the closest hyperplane, and then projects x onto that hyperplane and pushes it a bit beyond, thus misclassifying it with the minimal perturbation possible.

2.3 White box and black box evaluation

The attacks we saw before are based on the gradient of the neural network. In a white box configuration, that means that everyone has access to all the inner mechanism of the neural network, including especially the gradient or at least the outputs of the activation function, an attacker would be able to commit easily an attack. But what if the neural network keeps secret, so that one can just give an image as input and get a label as output? A first though might be that in such black box conditions the previous attacks won't be practicable.

Well, it turns out that this is not really a problem to commit an attack. Concretely an attacker can use a set of test images (they can even be randomly generated images), enter them in the neural network he is targeting and get the labels back. The attacker can then use this data (considering the output labels as the real labels of his set of images) to train an own deep neural network in order to simulate the targeted one. Fact is that an adversarial attack that will work on the training neural network will very often succeed as well on the targeted network. This is called transferability: an adversarial attack that works on a first neural network will work with high probability on a second one. The neural networks even don't need to have similar structures nor to be trained on the same set of images. As long as the two networks have been trained for the same task, the probability that an adversarial attack could be committed on both networks keeps very high.

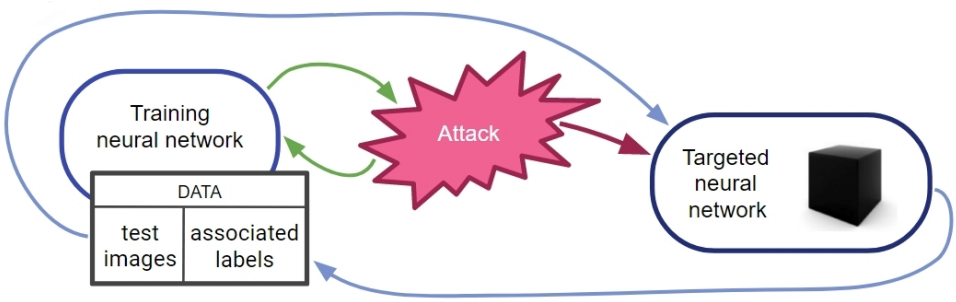


Fig. 2.1: Illustration of transferability

Actually, there could be even an easier way to attack a network in black box conditions. One can take an image and get the label returned by the neural network. Then add some random noise vectors to the image and look if this was an adversarial attack by giving the noised image to the neural network. With a bit of luck, the label will be another one so that the attack worked. Then, one can try to find the minimum amount of noise needed to be added to the original image so that the attack works.

This problem of transferability shows that the neural networks are not that robust and can be easily attacked. Thus a way to protect them from adversarial attacks need to be found.

2.4 GGD and AGGD distributions

2.5 MSCN Coefficients and Normality Tests

2.5.1 MSCN Coefficients

Digital images are characterized by pixels, and more specifically by the color of these. In the case of a color image, each pixel contains a red component, a blue component and a green component. The intensity of each of these components gives, by additive synthesis, a certain color to the pixel in question. The color of a pixel is therefore completely determined by its RGB code.

In this project, we were interested in black and white images. In the case of such an image, each pixel has a gray level color, i.e. each pixel has the same intensity in red, blue and green (see image). We can therefore assign to each pixel a number between 0 and 255 which corresponds to the intensity of its gray level (0 for black and 255 for white). We will therefore denote by $I(i,j)$ the intensity of the pixel located at the intersection of the line i and the line j.

Color Chart	R	G	B	Color Name
	0	0	0	Black
	255	255	255	White
	224	224	224	Light Gray
	128	128	128	Gray
	64	64	64	Dark Gray
	255	0	0	Red
	255	96	208	Pink
	160	32	255	Purple
	80	208	255	Light Blue
	0	32	255	Blue
	96	255	128	Yellow-Green
	0	192	0	Green
	255	224	32	Yellow
	255	160	16	Orange
	160	128	96	Brown
	255	208	160	Pale Pink

Fig. 2.2: Several RGB code

In our project, the problematic is the next one : Has a picture been attacked or not ?

To answer this question, let's define MSCN coefficients denoted by : $\widehat{\mathcal{I}}(i,j)$. Each pixel has a MSCN coefficient which can be calculated from the intensity :

$$\widehat{\mathcal{I}}(i,j) = \frac{\mathcal{I}(i,j) - \mu(i,j)}{\sigma(i,j) + \frac{1}{255}}$$

Moreover, exist other MSCN coefficients : horizontal, vertical, upper diagonals and lower diagonals such as :

$$\begin{aligned} \text{Horizontal:} \quad & H(i,j) = \widehat{\mathcal{I}}(i,j)\widehat{\mathcal{I}}(i,j+1) \\ \text{Vertical:} \quad & V(i,j) = \widehat{\mathcal{I}}(i,j)\widehat{\mathcal{I}}(i+1,j) \\ \text{Main diagonal:} \quad & D1(i,j) = \widehat{\mathcal{I}}(i,j)\widehat{\mathcal{I}}(i+1,j+1) \\ \text{Secondary diagonal:} \quad & D2(i,j) = \widehat{\mathcal{I}}(i,j)\widehat{\mathcal{I}}(i+1,j-1) \end{aligned} \tag{2.1}$$

Theoretically, MSCN coefficients of an image should follow a Generalized Normal Distribution (GGD) and the other pair product coefficients should follow an Asymmetric Generalized Gaussian Distribution (AGGD). This is what we want to use in this project to detect if an image has been attacked or not. The idea is to use statistical tests to compare the distribution of input images (attacked or not) to the distribution of the original non-attacked images.

Now, we can define GGD and AGGD:

GGD's density function is:

$$f_{\text{GGD}}(x) = \frac{\alpha}{2\beta\,\Gamma\left(\frac{1}{\alpha}\right)} e^{-\left(\frac{|x-\mu|}{\beta}\right)^{\alpha}}$$

where:
μ is the mean (so a position parameter which governs the position of the density function)
α is a shape parameter that governs the form of the distribution.
β is a scale parameter.

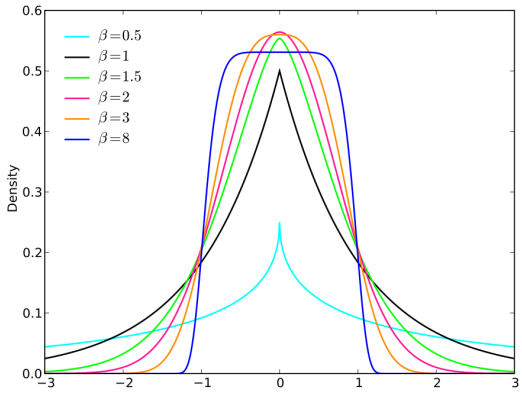


Fig. 2.3: Representation of GGD densities

AGGD's density function is:

$$f_{\text{AGGD}}(x) = \begin{cases} \dfrac{\alpha}{(\beta_\ell + \beta_r)\Gamma\left(\frac{1}{\alpha}\right)} e^{-\left(\frac{-x}{\beta_\ell}\right)^{\alpha}} & \text{if } x < 0 \\ \dfrac{\alpha}{(\beta_\ell + \beta_r)\Gamma\left(\frac{1}{\alpha}\right)} e^{-\left(\frac{x}{\beta_r}\right)^{\alpha}} & \text{if } x \geq 0 \end{cases}$$

where:
α is a shape parameter.
β_ℓ (β_r) is a left-scale (respectively right-scale) parameter.

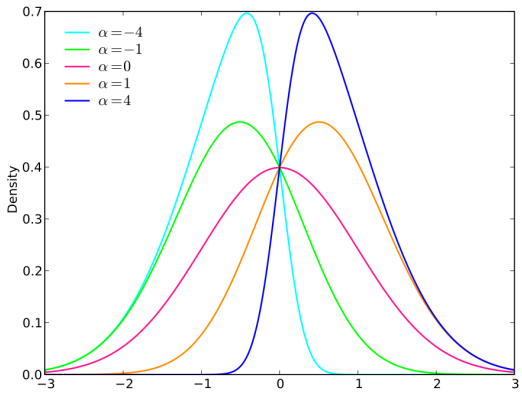


Fig. 2.4: Representation of AGGD densities

The AGG distribution is generalising the GG distribution by taking into account asymmetry of the distribution. By experience, one can actually observe that the pairwise product MSCN coefficients include this asymmetry so that a GGD would not be suitable enough.

In fact, when $\beta_\ell = \beta_r =: \beta$ then the AGGD becomes a GGD. So will we later mainly concentrate of the AGG distribution.

2.5.2 Normality tests

We want to be able to know if a sample, so, in our case, the MSCN coefficients are following a given distribution. For that, we can do statistical hypothesis testing.

A statistical hypothesis testing is a decision-making procedure. It is about reject or not a hypothesis, called null hypothesis or H_O, following our data set.

We assume that H_0 is true, the objective is to decide if this a credible assumption. The alternative hypothesis is called H_1 and it is the complementary of H_0.

A test statistic is a random variable which contains all the information about our data, it's considered as a numerical summary of a data-set that reduces it to one-value that can be used in our tests.

Before explain the test statistics we use here, we have to define the empirical distribution function

definition : the empirical distribution function is a repartition function which gives the probability $\frac{1}{n}$ to each number in the data-set. Let $X_1, X_2, ..., X_n$ a sample of random variables i.i.d on a probability space $(\Omega, \mathbb{R}, P)$ which takes its values in $\mathbb{R}$. The empirical distribution function F_n is defined by :

$F_n(x, \omega) := \frac{number of elements \leq x in the sample}{n} = \frac{1}{n} \sum_{i=1}^{n} 1_{X_i(\omega) \leq x}$

où 1_A est l'indicatrice de l'événement A.

Let's focus now on different Normality tests.

Kolmogorov-Smirnov test

Let $(X_1, ..., X_n)$ independant identically distributed random variables. We consider $F_n(x)$ an empirical function distribution such as:

$$F_n(x) = \sum_{i=1}^{n} 1_{X_i <= x}$$

The test is about the null hypothesis against a generic alternative:

$$(H_0) : F_n(x) = F(x) \ \forall x \quad \text{VS} \quad (H_1) : F_n(x) \neq F(x) \ \forall x$$

Let's consider the statistic of the Kolmogorov-Smirnov test:

$$D_n = \sup_x \mid F_n(x) - F(x) \mid$$

where F(x) is a theoretical empirical function, the Gaussian one in our case.

The Kolmogorov distribution is the distribution of the variable K, defined as: $K = \sup_{t \in [0,1]} \mid B(t) \mid$, where B is a Brownian bridge.

Under null hypothesis, we have that : $\sqrt{n} D_n \underset{n}{\to} \sup_{t \in [0,1]} \mid B(t) \mid$

Therefore, the critical region is:

$$D_n > \frac{K_\alpha}{\sqrt{n}} \text{ where } K_\alpha \text{ is the quantile of level } \alpha \text{ of K.}$$

In other words, the aim of this test is to evaluate the biggest gap between the representation of $F_n(x)$ and the one of $F(x)$. Therefore, here is an illustration of the idea:

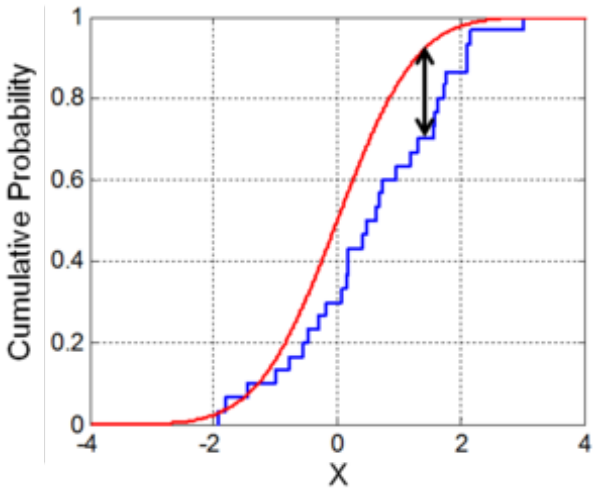


Fig. 2.5: Example of Kolmogorov-Smirnov test

3 Models and attacks

This study will be based on the MNIST and CIFAR 10 base. It is a base that gathers numbers for the MNIST base and animals and objects for the CIFAR 10 base.

3.1 Creation of the Neural Networks

First of all, in order to attack our NNs, they have to be created. In our case, 2-layer, 4-layer NN and CNN will be used. Obviously, the input of our NNs are the pixels of our image and the output is all the probabilities of the different labels of objects. Here are the different accuracy of our models for the MNIST base.

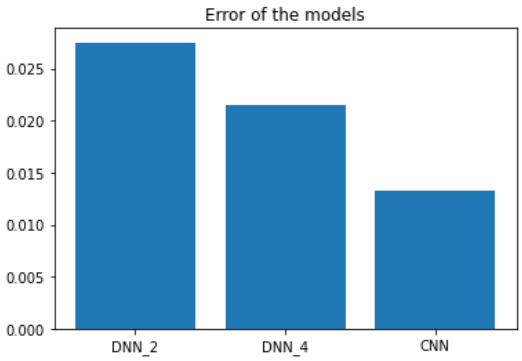


Fig. 3.1: Error rate of the models

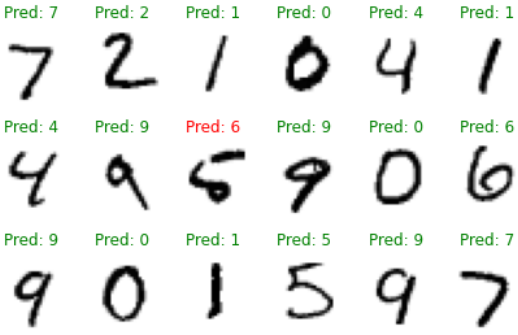


Fig. 3.2: Predictions of a 2-layer DNN on MNIST images

The error rate of the 2-layer, 4-layer NN and CNN on the MNIST library are respectively 0.0275, 0.0215 and 0.0132 with a learning rate of 0.1.It is a very good accuracy, almost all the predictions are good. A simple way to see it is to plot the images and see if the labels are rightfully predicted like above. Those are the predictions of the 2-layer DNN.

The DNN models are less accurate on the CIFAR 10 database. A CNN is more accurate for this kind of images.

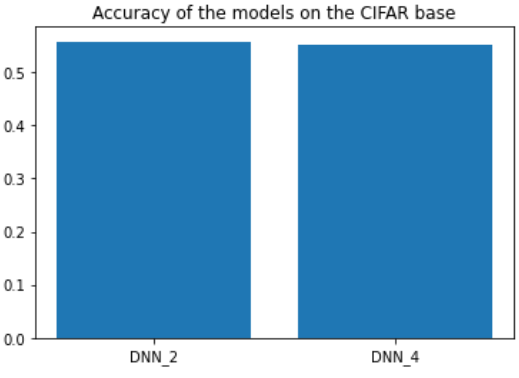


Fig. 3.3: Accuracy of the models

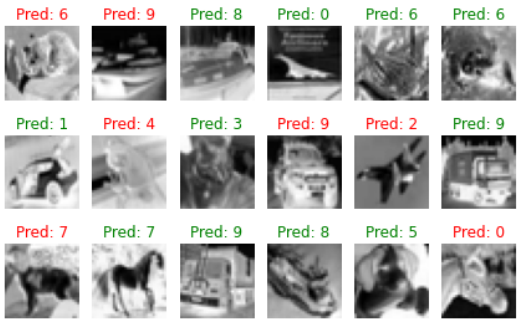


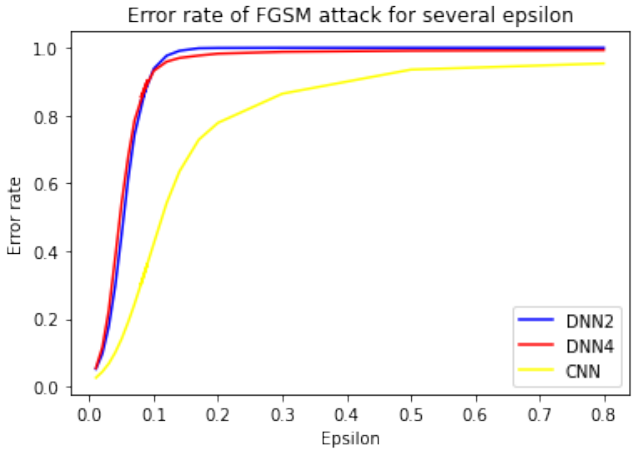
Fig. 3.4: Predictions of a 2-layer DNN on CIFAR-10 images

The predictions almost have a chance out of two to be true which is better than a random prediction (1/10) but not as accurate as for the MNIST pictures. On the second image, it is the predictions given by the 2-layer NN on the MNIST base.

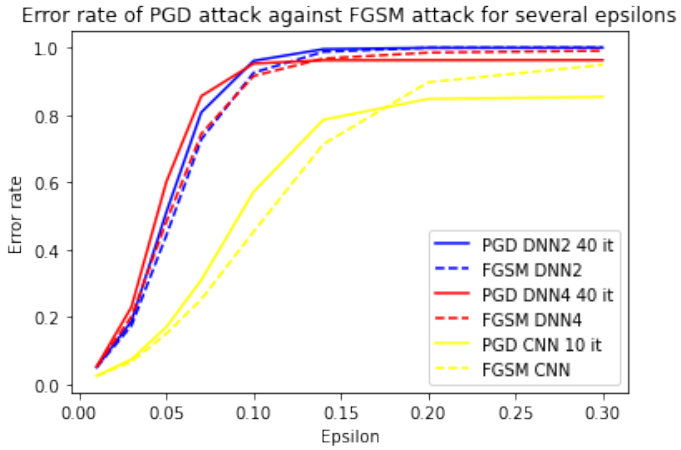
3.2 Comparison of the different attacks

The attacks have different approaches and it seems interesting to compare them and to optimize their accuracy by changing their parameters.

First of all, the ϵ parameter can be change for the FGSM attack. It is clear that the error rate of the predictions becomes greater if ϵ is big. This is obviously normal considering that by rising ϵ, it allows more modification to the original image. However, it is also preferable to have a low ϵ because the modification will be relatively invisible to a human eye or a detector.



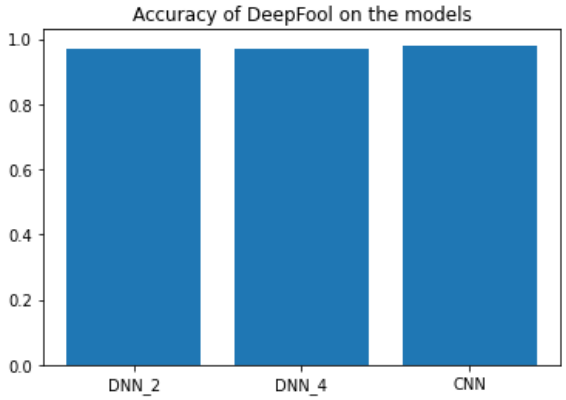
The PGD attack looks like the FGSM on a lot of aspects. Here are a comparison of their accuracy between them depending on the ϵ used :



It is clear that the PGD attack has an accuracy rate better than the FGSM for ϵ little but then, the FGSM attack is better than the PGD attack with a great value of ϵ so with a great modification of the original image. It could be explained by two factors. The first one is that the PGD attack is a normalized FGSM attack repeated several times that is efficient specifically when ϵ is little. The second factor could be that ϵ is not the only parameters to change for the PGD attack. There are also the value of α and the number of iterations that normally has to be changed depending on the value of ϵ (cf. 2.2.3 PGD attack).

Thanks to these different graphs, we can estimate a optimal ϵ. Indeed, an ϵ of 0.1 seems to be the tiniest value of ϵ with an error rate close to 1. This value will be kept for the different attacks throughout this paper.

For the DeepFool attack, parameters do not have to be optimized because the parameters are: the image, the model, the overshoot and the maximum number of iteration. The image, the model and the maximum number of iterations are fixed. The overshoot parameter is here to avoid the estimation of a number to zero when he is tiny. Moreover, the accuracy of DeepFool is extremely high respectively 0.97, 0.97 and 0.98 for the 2-layer, 4-layer DNN and CNN.



It would also be interesting to study the different time execution for all three attacks on the different models. So, here are the different times taken to complete the attack on 100 images :

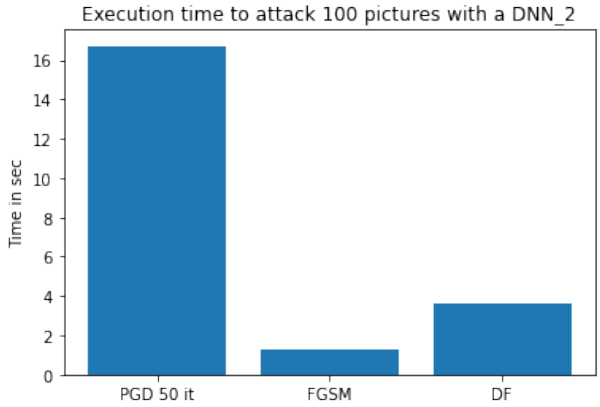


Fig. 3.5: Execution time of the attacks on a 2-layer DNN

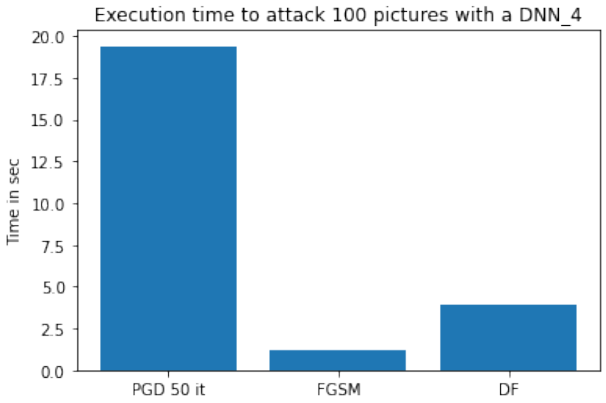


Fig. 3.6: Execution time of the attacks on a 4-layer DNN

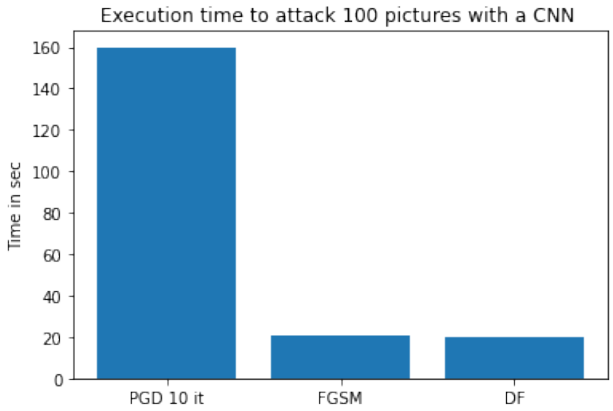


Fig. 3.7: Execution time of the attacks on a CNN

The FSGM attack has a very low execution time followed by the DeepFool attack. The PGD attack is the longest one due to the number of iterations that he has to process. All the attacks last also longer on the CNN model.

To conclude, FGSM attack is the fastest one but has also the worse result in accuracy for ϵ little. A good value for ϵ would be 0.1 which permits a great accuracy of the FGSM and PGD attacks but also a little deformation of the original image. The DeepFool attack is very effective. It is quite fast and have the best result in the minimization of the perturbation to the original image. Therefore, the different tests that will follow have a less easy time detecting images attacked by DeepFool.

4 Characterise images with MSCN coefficients

In this part, we will explain how we will compute and use Mean Subtracted Contrast Normalized (MSCN) coefficients in our code.

As explained in the State of art, five kinds of MSCN coefficients can be calculated. The "basic" MSCN coefficients are supposed to follow a Generalized Gaussian Distribution (GGD), whereas the pairwise product MSCN coefficients should follow an Asymmetric Generalized Gaussian Distribution (AGGD).

Our goal is this one: we want to determine if images have been attacked or not making as less false positives as possible. To this end, we need to find a way to characterize non-attacked and attacked images. This is why we are interested in MSCN coefficients because we know how they behave for an image which was not attacked and we know that they look different for attacked images.

4.1 Calculation of MSCN coefficients

Recall that if $\mathcal{I} := \{\mathcal{I}(i,j)\}_{i\in[\![1,I]\!],j\in[\![1,J]\!]}$ is an image of size $I \times J$, then the associated MSCN coefficients are $\widehat{\mathcal{I}}(i,j) = \dfrac{\mathcal{I}(i,j) - \mu(i,j)}{\sigma(i,j) + C}$, where $\mu(i,j)$ is a local mean and $\sigma(i,j)$ is a local standard deviation of the pixel $\mathcal{I}(i,j)$. As $\sigma(i,j)$ is just local an thus might be zero, a constant C is added to avoid division by zero; in this project pixel belongs to $[0,1]$ so we will take a small $C := 1/255$. Once we will be able to determine $\widehat{\mathcal{I}}(i,j)$, $\forall (i,j) \in [\![1,I]\!] \times [\![1,J]\!]$, the pairwise product MSCN coefficients will be easily calculated as in (2.1). So let focus on $\mu(i,j)$ and $\sigma(i,j)$ (we proceed as in [MMB09]).

4.1.1 A local mean

A local mean $\mu(i,j)$ of a pixel $\mathcal{I}(i,j)$ can be calculated using a square of pixels centered on this pixel. Using this square, one can get an average value of the center pixel by weighting each of the pixel of the square, so that it gives a local mean pixel value between 0 and 1. In fact, applying this operation of calculating the local mean of every pixel of an image can be seen as a convolution product between the image and this square that is more rigorously called a convolution kernel.

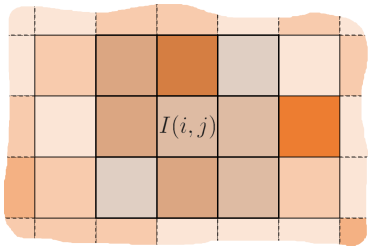


Fig. 4.1: Kernel of size 3 applied to the pixel $\mathcal{I}(i,j)$ of an image

There are various possibilities for the choices of the size of the kernel and for the type of weights used. The size of the kernel is usually chosen to be an odd number, so that there is a pixel in the middle that will be average. A kernel of even size can anyway be defined but has no real utility here, so we fill only consider odd

sizes for kernels. For the length there is no real rule, it has to be appropriated to the dimensions of the image. In this project we work on small images (28×28 or 32×32 pixels), therefore we will use a kernel of size 5 or 7.

In mathematical language, this gives the following formula:

$$\mu(i,j) := (K * \mathcal{I})(i,j) = \sum_{n=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \sum_{m=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} k_{n,m}\, \mathcal{I}(i+n, j+m)$$

where N is the odd size of the kernel, so $\lfloor N/2 \rfloor$ is the distance between the center pixel and a border. K is the kernel, such that if for instance $N = 3$, we have $K := \begin{bmatrix} k_{\text{-}1,\text{-}1} & k_{\text{-}1,0} & k_{\text{-}1,1} \\ k_{0,\text{-}1} & k_{0,0} & k_{0,1} \\ k_{1,\text{-}1} & k_{1,0} & k_{1,1} \end{bmatrix}$

Concerning the type of weights, we will only concentrate on a uniform kernel and a Gaussian kernel in this project. The uniform kernel is the easiest kernel that can be imagined because all the pixels in the kernel are treated with equal weight.

For example, with $N = 3$:

$$K_{\text{uniform}} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

But this is not necessarily logical for an image, because the pixels toward the center of the kernel are somehow more representative of what is going on in that image at that point, than the ones that are farther around. This is why the Gaussian kernel is an interesting alternative to the uniform kernel and will probably better handle with the naturalness of images. So, as it names indicates, the Gaussian kernel is weighted using a Gaussian distribution, such that the center pixel has the biggest weight, and the farther a pixel is from the center, the smaller its weight becomes.

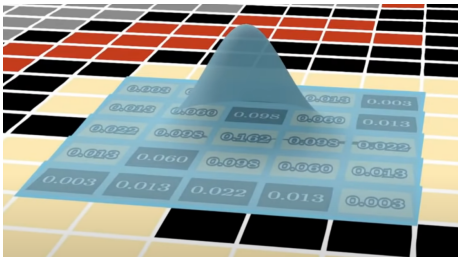


Fig. 4.2: Gaussian kernel of size 5 applied to an image

One last thing to think about is that for the pixels on the border of the image, the kernel centered on these pixels will cover only a part of the image. So what should be done in such cases? Well, this is not really a problem. One can just consider that behind the image there is a black background, corresponding to a 0 value for pixels, and proceed this way the averaging of the center pixel with the kernel. In fact, this is also what will be calculated using the convolution product.

4.1.2 A local standard deviation

To calculate a local standard deviation $\sigma(i,j)$, we just have to compute a local variation $\sigma^2(i,j)$ using the usual definition of the variance $\mathbb{V}(X) := \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$ for a random variable X. So in our case, we get:

$$\sigma^2(i,j) := \mu^2(i,j) - \nu(i,j)$$

where $\nu(i,j)$ is the local square mean of the pixel $\mathcal{I}(i,j)$, such that :

$$\nu(i,j) := (K * \mathcal{I}^2)(i,j) = \sum_{n=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} \sum_{m=-\lfloor N/2 \rfloor}^{\lfloor N/2 \rfloor} k_{n,m}\, (\mathcal{I}(i+n, j+m))^2$$

4.1.3 Display of some MSCN coefficients

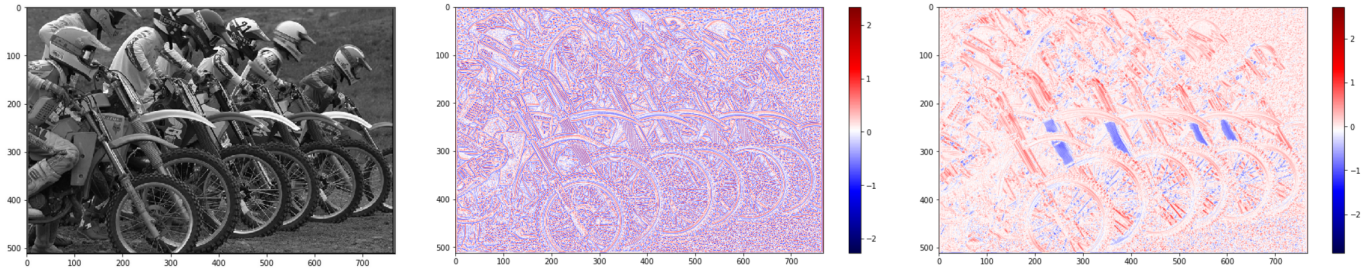


Fig. 4.3: On a natural image: original image, ”basic” MSCN and main diagonal MSCN, with a Gaussian kernel of size 7

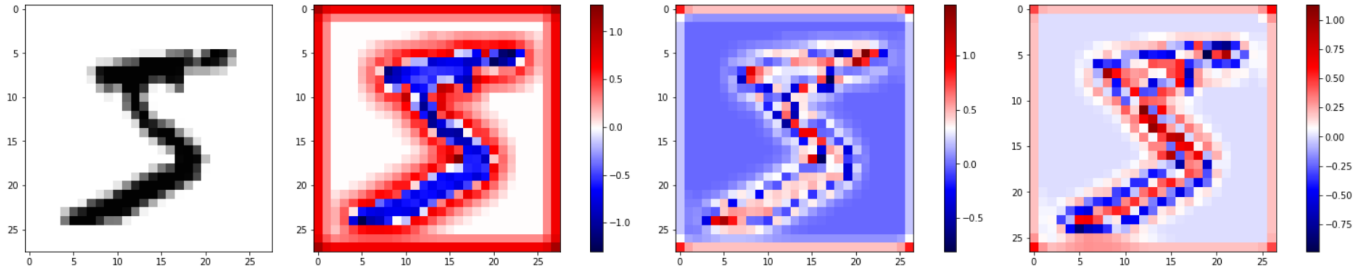


Fig. 4.4: On a MNIST image: original image, ”basic” MSCN, horizontal MSCN and main diagonal MSCN, with a Gaussian kernel of size 5

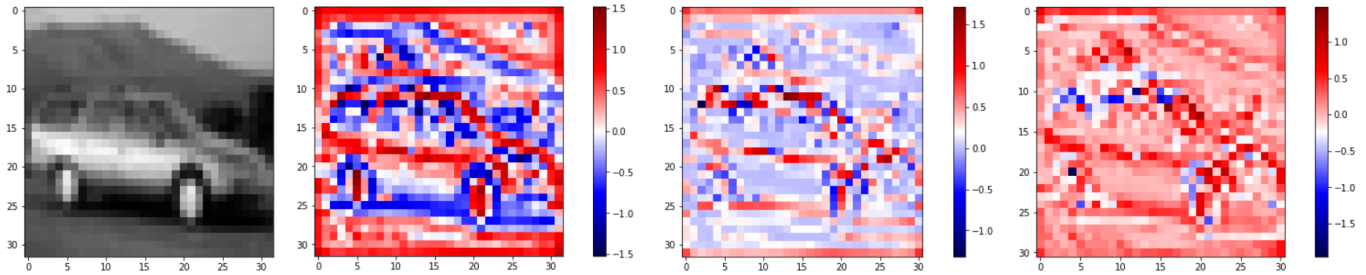


Fig. 4.5: On a CIFAR10 image: original image, ”basic” MSCN, horizontal MSCN and main diagonal MSCN, with a Gaussian kernel of size 5

4.2 Estimations of GGD and AGGD parameters

As explained previously, GGD and AGGD are characterized by different parameters. The aim of this part is to explain how these parameters are estimated, in order to approach the laws followed by the distributions of the MSCN coefficients.

For the AGGD, remind that the three parameters are: α, β_ℓ, β_r

GGD is a particular case of the AGGD when $\beta = \beta_\ell = \beta_r$. So we only need to look how AGGD parameters can be computed.

To estimate those AGG density parameters we won't use the Maximum Likelihood method because it uses several steps to find the best value of the shape parameter α. Although it provides a better convergence rate for

reaching parameter value, it does not guarantee convergence for bad initial value. So, Maximum Likelihood estimation is inherently complex and needs a lot of computations. Thus, we will proceed as in [LSB09] and use their method that allows us to find model parameters given the one-order absolute moment and the second-order moment: they called it Second Order Statistics (SOS) estimation.

To summarize, using k-order moments and k-order absolute moments, we get that:

$$r := \frac{(\mathbb{E}|X|)^2}{\mathbb{E}[X^2]} = \frac{(\gamma^2+1)^2}{(\gamma^3+1)(\gamma+1)}\rho(\alpha)$$

where $\gamma := \frac{\beta_\ell}{\beta_r}$ and $\rho(x) := \frac{\Gamma\left(\frac{2}{x}\right)^2}{\Gamma\left(\frac{1}{x}\right)\Gamma\left(\frac{3}{x}\right)}$ $\left(\Gamma(z) := \int_0^\infty t^{z-1}e^{-t}dt, \forall z \geq 0\right)$.

Let first define n as the number of elements in the sample, so in our case $n = I \times J$, where I and J are the sizes of the images.

r can be easily estimated by the unbiased estimator $\hat{r} := \frac{\left(\frac{1}{n}\sum_{i=1}^n |x_i|\right)^2}{\frac{1}{n}\sum_{i=1}^n x_i^2} = \frac{1}{n}\frac{\left(\sum_{i=1}^n |x_i|\right)^2}{\sum_{i=1}^n x_i^2}$, so we need to be able to estimated γ and to inverse the function ρ, to get a estimation of the three parameters α, β_ℓ and β_r of an AGG distribution.

The value of γ can be estimated by $\hat{\gamma} := \frac{\sqrt{\frac{1}{N_\ell - 1}\sum_{k=1}^n x_k^2 1_{\{x_k<0\}}}}{\sqrt{\frac{1}{N_r-1}\sum_{k=1}^n x_k^2 1_{\{x_k\geq 0\}}}}$, where $N_\ell := \sum_{k=1}^n 1_{\{x_k<0\}}$ is the number of negative elements and $N_r := \sum_{k=1}^n 1_{\{x_k\geq 0\}}$ is the number of positive elements.

These two previous estimations permit then to compute an estimator of $\rho(\alpha)$ defined as:
$\hat{R} := \hat{r}\frac{(\hat{\gamma}+1)(\hat{\gamma}^3+1)}{(\hat{\gamma}^2+1)^2}$.

Then, using an approximation method to inverse ρ, we obtain an estimation of the important shape parameter:

$$\hat{\alpha} := \rho^{-1}(\hat{R})$$

Finally, we know that for $i \in \{\ell, r\}$, $\beta_i = \sigma_i\sqrt{\frac{\Gamma\left(\frac{1}{\alpha}\right)}{\Gamma\left(\frac{3}{\alpha}\right)}}$ and the left and right deviance σ_ℓ and σ_r can be respectively estimated by: $\hat{\sigma_\ell} := \sqrt{\frac{1}{N_\ell-1}\sum_{k=1}^n x_k^2 1_{\{x_k<0\}}}$ and $\hat{\sigma_r} := \sqrt{\frac{1}{N_r-1}\sum_{k=1}^n x_k^2 1_{\{x_k\geq 0\}}}$.

So we get an estimation of two other parameters of an AGGD β_ℓ and β_r as:

$$\hat{\beta_i} = \hat{\sigma_i}\sqrt{\frac{\Gamma\left(\frac{1}{\hat{\alpha}}\right)}{\Gamma\left(\frac{3}{\hat{\alpha}}\right)}}$$

4.3 Distributions associated to the MSCN coefficients

From formulas to calculate MSCN coefficients (see (2.1)), the parameters of the laws followed by these coefficients can be now be estimated. So we can compare the distribution of the MSCN coefficients computed on the previous images to the associated AGGD (or GGD) curves. Here are the plots for some of coefficients from the three images displayed above:

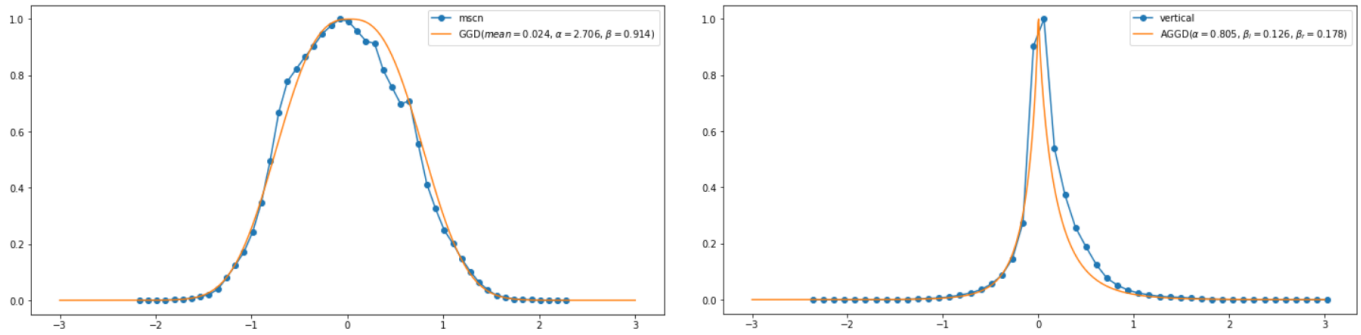


Fig. 4.6: MSCN coefficients of the natural image (”basic” MSCN and vertical MSCN) and the associated estimated densities

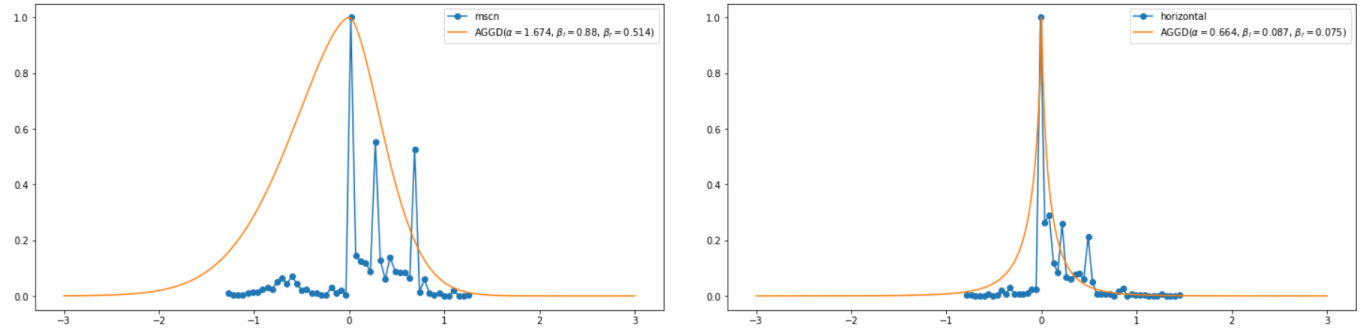


Fig. 4.7: MSCN coefficients of the MNIST image (”basic” MSCN and horizontal MSCN) and the associated estimated densities

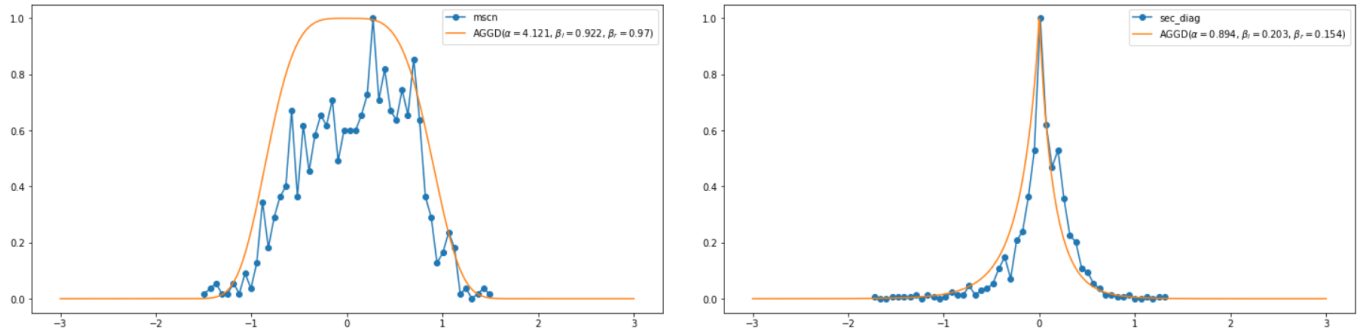


Fig. 4.8: MSCN coefficients of the CIFAR10 image (”basic” MSCN and secondary diagonal MSCN) and the associated estimated densities

For a matter of space, we displayed just the ”basic” MSCN coefficients and one type of the pairwise product coefficients. Note that, in fact, the goodness of fit for the pairwise product coefficients is for a given image almost the same. There is just sometimes a real gap between the fit for ”basic” MSCN and the other (pairwise product) MSCN coefficients.

As we can observe on the two first plots, for a natural image the distributions of the MSCN coefficients are very closed to the estimated densities associated to those coefficients. We can notice that this is in particular true for the ”basic” MSCN coeffcients, while it doesn’t seem to be as efficient for the two other images.

In fact, for the MNIST images, the distributions of the ”basic” MSCN coefficients can not be approached by a GGD or even a AGGD density as we supposed. This is very certainly due to fact that the figures on the

MNIST images are too special symbols, and aren't thus natural images. Nevertheless, for the pairwise product coefficients, the distributions do better correspond to the estimated densities, even if a fallout near to zero can most of the time be observed on one half of the distribution. The MNIST images contain pixels value that are mainly near either to 0 or 1. So there is not a lot of information given by those pixels. This can explain why the estimated densities only timidly approach even the pairwise product coefficients. Notice that in order to get a curve as wide as possible, we took care to have a white background, corresponding to the pixel value 1, and therefore just a few 0 corresponding to the written figures. But even that is not enough to characterise correctly MNIST images.

Concerning the CIFAR10 images, we get a mixed feeling. Even if the "basic" MSCN coefficients correspond better to the associated estimated density as for MNIST images, the fit is still not that accurate. So we won't use those coefficients to characterize CIFAR10 images. However, for the pairwise product coefficients the fits are pretty convenient. There are not as precise as for a natural image, this is probably due to the size of the images that is quite small and can't therefore contain a lot of natural details. But it seems suitable enough to use this to characterize CIFAR10 images.

These comparisons with the original non-attacked images are important, because as we will see afterwards, we want to create an attacked images detector that makes as less false positive as possible. And to this aim, the first property which the detector must have, is that it must almost ever be sure that a non-attacked image was effectively not attacked. Thus we need to be able to characterize original images precisely.

5 Test whether an image was attacked or not

As we said previously, to create a statistical test that determines whether an image was attacked or not, the idea is to use the naturalness of original images, which we know how to characterise. So, we want to compare the distribution of the MSCN coefficients of an input image (attacked or not) with the estimated law that the original image would follow (i.e. an AGGD distribution).

The statistical test we need to perform is a test of asymmetrical Gaussian generalized distribution. Such a direct test doesn't exist, we thus have to use distribution comparison tests named *goodness of fit* tests. In this project, we will focus on the Kolmogorov-Smirnov test that compares a sample to a theoretical cumulative distribution function (cdf).

The H_0 assumption of this test is that the sample corresponds to the cumulative distribution function, against the H_1 assumption that is the sample follows a different distribution than the one of the cumulative distribution function.

For the theoretical cumulative distribution function which should characterise original images, it will be created thanks to the estimations of the parameters of the MSCN coefficients of original images as we have seen in the above section.

5.1 MSCN coefficients on attacked images

Before computing the tests, one does want to get a feeling of how they will react by displaying the MSCN coefficients of some attacked images. These attacked images have been obtained thanks to the implementation of the attacks as described in the section 3 of this paper. The distribution of those MSCN coefficients for a MNIST and a CIFAR10 image is also shown.

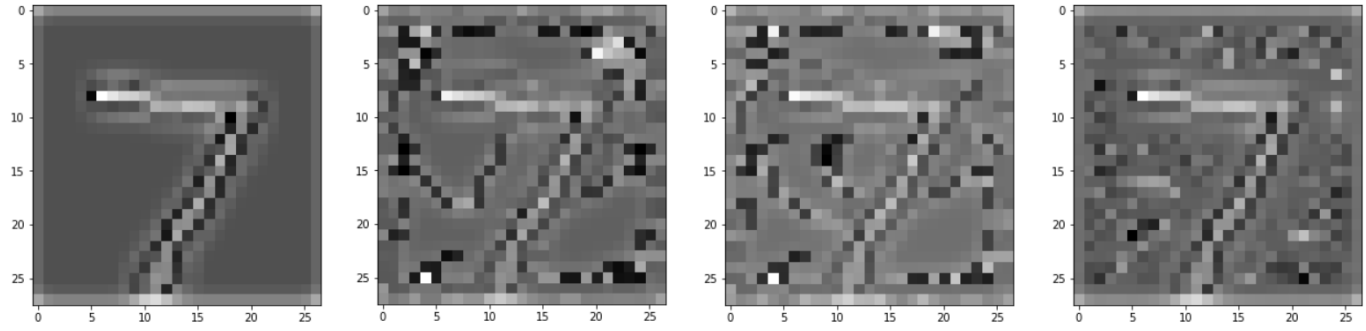


Fig. 5.1: Horizontal MSCN coefficients computed on an original MNIST image and on the FGSM, PGD and DF attacked versions

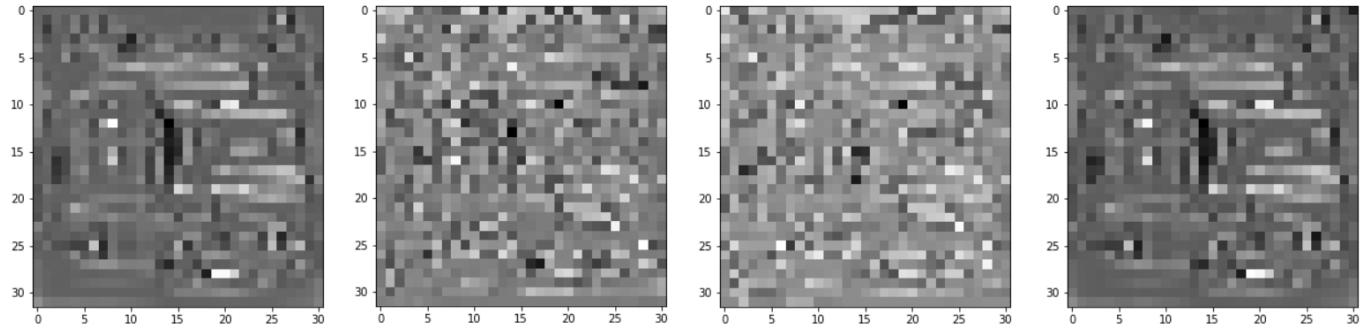


Fig. 5.2: Horizontal MSCN coefficients computed on an original CIFAR10 image (it is a truck) and on the FGSM, PGD and DF attacked versions

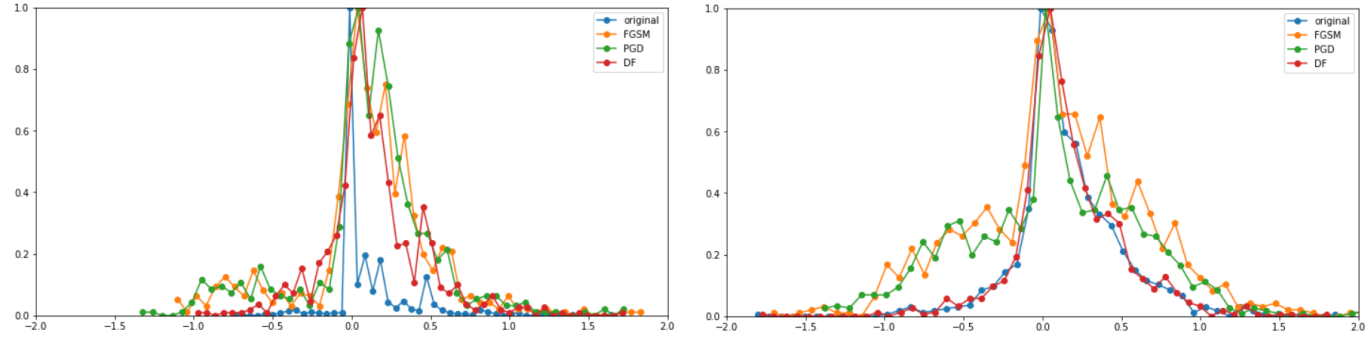


Fig. 5.3: The corresponding distributions of the two above images (the MNIST image on the left and the CIFAR10 image on the right)

On the MNIST image, there are some stranger craters for the attacked images. It probably comes from the fallout to zero and the little diversity of pixels values as described in the above section. So, a human can easily detect the attacks on MNIST images. We can also already notice that the DF attacked image seems to be much nearer to the original one than the two other attacked images. Concerning the plots, a first thing to notice is that the three attacks looks almost the same. And even if the original distribution deviates from the distribution of the attacked images, it will in fact turn out that those later distributions do correspond very exactly to the estimated AGGD curves calculated thanks to the original image. Therefore, the attacked curves do not differ from the information we have on the non-attacked image.

On the CIFAR10 image, where the MSCN coefficients should better characterise a non-attacked image, with human eyes one can as well observe the attacks on the FGSM and the PGD attacked images. However, the DeepFool attack is almost impossible to differentiate from the original one. As the statistical test relies directly on the MSCN coefficients values, we can already suppose that the DF attack will cause problems to the tests. The same supposition can be made looking at the plots of the distributions: for the FGSM and PGD curves, the shapes are different from the original image, but the DF curve is very closed to the original one. One could verify it by computing some tests.

5.2 Implementation of Kolmogorov Smirnov test

We display here the results of the tests for images of the MNIST and CIFAR10 data sets. We compare the estimated cdf from the original image to the original image itself and to three attacked versions of this image.

	mscn	horizontal	vertical	main_diag	sec_diag
original	0.9801697375	0.9968661642	0.9777431652	0.9990363267	0.9945910807
FGSM attacked	0.9966543925	0.0157651474	0.1338351087	0.9968699541	0.5594986422
PDG attacked	0.9999605104	0.0068283146	0.1688977167	0.6034846704	0.7269616810
DF attacked	0.9949208309	0.9976675321	0.9873897159	0.9989097040	0.9935375913

	mscn	horizontal	vertical	main_diag	sec_diag
original	0.8727394673	0.9714226867	0.9712916716	0.9937312598	0.9942980490
FGSM attacked	0.9992396866	0.0001104793	0.0016349070	0.1120647014	0.1355267212
PDG attacked	0.9975263681	0.0005089649	0.0018046996	0.1307354450	0.2170223748
DF attacked	0.9986234014	0.9967196981	0.9953252870	0.9988624192	0.9906702728

Fig. 5.4: p-values of the tests computed for all five MSCN coefficients for two MNIST images (on the original image and the FGSM, PGD and DF attacked images)

	mscn	horizontal	vertical	main_diag	sec_diag
original	0.9132575427	0.9726501651	0.9712503252	0.9272833023	0.9877696348
FGSM attacked	0.0036631788	0.0000000000	0.0000000000	0.0000000000	0.0000000000
PDG attacked	0.0622003291	0.0000000000	0.0000000000	0.0000000000	0.0000000000
DF attacked	0.8700460348	0.9624767802	0.9388986364	0.9177476041	0.9854373987

	mscn	horizontal	vertical	main_diag	sec_diag
original	0.9520143608	0.9677694893	0.9182287861	0.9980629997	0.9734516187
FGSM attacked	0.0000027238	0.0000000000	0.0000000000	0.0000000000	0.0000000000
PDG attacked	0.0000090786	0.0000000000	0.0000000000	0.0000000000	0.0000000000
DF attacked	0.4734087399	0.4632049618	0.3095213338	0.2567801102	0.0253434052

Fig. 5.5: p-values of the tests computed for all five MSCN coefficients for two CIFAR10 images (on the original image and the FGSM, PGD and DF attacked images)

The results of the tests computed on the MNIST images can't be used for detecting attacked or non-attacked image. In fact, the tests systematically consider that every images was not attacked (the p-values are much larger than 5%). As explain just before, this is due to the fact that attacked distributions almost perfectly correspond to the estimated AGGD curves calculated thanks to the original image. More generally, this comes from the unnaturalness of the MNIST images, and the fact that their MSCN coefficients do not follow AGG distribution.

However, for the CIFAR10 images, the detection of attacked images seems to work pretty nicely for some attacks. As predicted from the MSCN coefficients displays and distribution plots, the DF attacks fools the tests. The p-values are most of time smaller than for the original image, but still much larger than 5% (except for of the tests here), so that we accept the H_0 assumption which says that the DF attacked-image corresponds to the estimated original density, and is therefore non-attacked. Nevertheless, for the FGSM and the PGD attacks the tests are for pairwise product MSCN coefficients always sure that H_0 has to be rejected, and therefore that the images were effectively attacked.

Thus, in the next section, we will focus on the pairwise product MSCN coefficients computed on CIFAR10 images to create an adversarial attacks detector.

6 A detector for attacked images

6.1 Context

In order to help image classification by deep neural networks to be more robust against adversarial attacks, we want to detect these attacks upstream. To this aim, we need to create an ”entrance detector” what will either let non-attacked images go into the neural network, or redirect attacked images to a denoiser. We won't focus on the latter in this project, but just in a few words, this denoiser will try to remove the noise used to attack those images so that the neural network can finally rightly classify them.

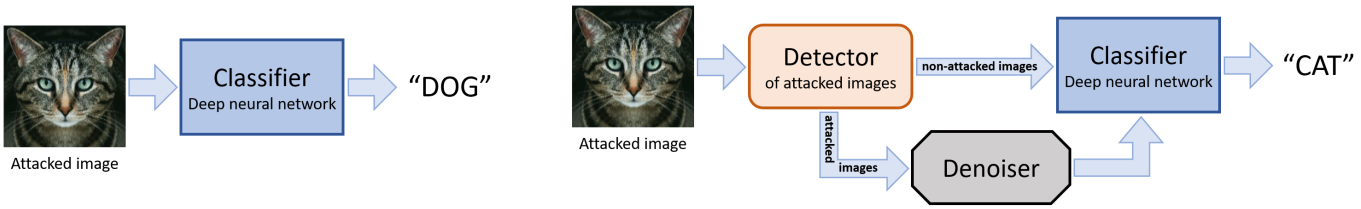


Fig. 6.1: System without detector on the left that might be subject to adversarial attacks, more robust system with detector on the right

Nevertheless, an important point is that the denoiser is trained to transform an attacked image into approximately the original one and continues to learn from the new inputs it gets. So giving as input non-attacked images to the denoiser, will deregulate it from its original function and will thus corrupt the whole system.

Therefore, the attacked-images detector needs to make as less false positives as possible, so that it does not conduct to a malfunction of the denoiser.

6.2 Idea of the proposed basic detector

6.2.1 Use a bank of parameters

We know that the MSCN coefficients of natural images should follow a Generalized Gaussian distribution or an Asymmetric Generalized Gaussian distribution. And thanks to statistical testing, we are able to determine whether an image is natural or not, and thus to determine under some conditions, whether an image was attacked or not.

We suppose that we can summarise the distributions of MSCN coefficients of natural images of a data set by the distributions of just a sample of those MSCN coefficients. In fact, we can't know the real distributions of the MSCN coefficients of the natural images but we are able to estimate the parameters of the AGG distributions they should follow.

These parameters of the distributions of the MSCN coefficients of a sample of the natural images of the data set, will constitute a bank of parameters. And that bank contains the knowledge the detector will have to

detect adversarial attacks. The parameters of the bank are supposed to characterize the distributions of MSCN coefficients of non-attacked images for the studied data set.

So, to recap, the starting point is this one: we have a sample of non-attacked images of a data set. The five types of MSCN coefficients (”basic MSCN”, ”horizontal”, ”vertical”, ”main diagonal” and ”secondary diagonal”) are calculated for those non-attacked images. Then, for every image, the parameters of the distributions of all five different MSCN coefficients are estimated. And that set of parameters (five distributions for every image) forms the bank of parameters.

6.2.2 Attacked or non-attacked image?

The basic idea for the detector is then, using a statistical test, to compare every input image in the detector to all of the distributions in the bank. If there are k distributions of one coefficients type in the bank, for every input image, k statistical tests are performed for each of the five type of coefficients.

The detector's mission consists then to conclude that the input image was attacked or not. If the input image was not attacked then its distribution should correspond to some of the distributions contained in the bank of parameters, so that the detector will conclude that this image is an original one. Otherwise, if the input image was attacked, almost none of the tests will confirm that some distributions correspond and thus the image will be detected as attacked. More concretely, the detector will look at the proportion of the k tests, which have accepted the assumption that the bank distributions and the MSCN coefficients of the image are identical. If this proportion is lower than a level λ (for instance $\lambda = 10\%$), then the detector will admit that the image is attacked.

We give here the whole procedure of detection in a mathematical language. For this, we first need to define a couple of objects:

Let first $\mathcal{S}$ be the data set of images on which we create the detector (for instance CIFAR-10).

T is the set of type of MSCN coefficients, such that $T := \{$”mscn”, ”horizontal”, ”vertical”, ”main diagonal”, ”secondary diagonal”$\}$.

Let n_{train} be the number of original images we use to create the detector.

$\mathcal{B}_{\mathcal{S}}$ is the bank of parameters for $\mathcal{S}$, such as:

$$\forall i \in [\![1, n_{\text{train}}]\!], \forall t \in T, \quad \mathcal{B}_{\mathcal{S}}(i,t) = (\alpha^{(i,t)}, \beta_{\ell}^{(i,t)}, \beta_{r}^{(i,t)})$$

the parameters of the AGGD laws estimated on the original image i for each type of MSCN coefficients.

$F_{\text{AGGD}}(\cdot, \alpha, \beta_\ell, \beta_r)$ is the cumulative distribution function (cdf) of an AGGD law.

$p^{\text{K-S}}_{\text{value}}(\text{sample}, \text{cdf})$ is the function that returns the p-value of the Kolmogorov-Smirnov test described in the previous section.

And let lastly $C_{\text{MSCN}}(\mathcal{I}, t)$ be the function which returns the MSCN coefficients of type t for an input image $\mathcal{I}$.

From there, we can finally define our detector of attacked images $\mathcal{D}_{\mathcal{S}}$, for a data set $\mathcal{S}$. Giving an input image $\mathcal{I}$ we get:

$$\forall t \in T, \quad \mathcal{D}_{\mathcal{S}}(\mathcal{I}, t) = \begin{cases} \text{”attacked ”}, & \text{if } \sum_{i=1}^{n_{\text{train}}} 1_{\left\{p^{\text{K-S}}_{\text{value}}\left(C_{\text{MSCN}}(\mathcal{I},t),\ F_{\text{AGGD}}(\mathcal{B}_{\mathcal{S}}(i,t))\right) \geq 5\%\right\}} < \lambda \\ \text{”not attacked ”}, & \text{if } \sum_{i=1}^{n_{\text{train}}} 1_{\left\{p^{\text{K-S}}_{\text{value}}\left(C_{\text{MSCN}}(\mathcal{I},t),\ F_{\text{AGGD}}(\mathcal{B}_{\mathcal{S}}(i,t))\right) \geq 5\%\right\}} \geq \lambda \end{cases}$$

where $\sum_{i=1}^{n_{\text{train}}} 1_{\left\{p_{\text{value}}^{\text{K-S}}\left(C_{\text{MSCN}}(\mathcal{I},t),\ F_{\text{AGGD}}(\mathcal{B}_{\mathcal{S}}(i,t))\right)\geq 5\%\right\}}$ is the number of tests, which accepted the assumption that the bank distributions and the MSCN coefficients of the image are identical.

6.2.3 Why it works

For adversarial attacks committed on natural images, the distributions of the MSCN coefficients of those attacked images have a different shape than the distributions of the MSCN of a natural non-attacked image. So in the detector, if the input is an attacked image, then none of distributions contained in the bank of parameters will correspond to that input image, so that it won't almost ever be detected as non-attacked image. Therefore, false positives will be almost avoided and the detection as described above will work.

Notice that is very probably the reason why for MNIST data set, the statistical tests did not succeed to see a difference between attacked images and the original ones. This data set is composed just by small images (28×28 pixels) of handwritten figures between 0 and 9. Black figures on a white background with not a lot of gray pixels between both. So these images can not really be considered as natural and therefore we can't finally try to create a detector for them.

However, the CIFAR10 data set seems to propose more natural images. Even if its images are still very small (32×32 pixels) and do therefore not contain a lot of natural details, the fact that they represent some real objects or animals makes them be much more natural. Hence a difference of distributions of MSCN coefficients between attacked and non-attacked images can be observed. We will thus create a detector on this later data set.

6.3 Detector for the CIFAR10 data set

As we have seen previously, the "basic" MSCN coefficients of the CIFAR10 images do not really follow a GGD or even a AGGD. It is much better than for MNIST images, but even if CIFAR10 images are more natural, they are very certainly too small to give nice "basic" MSCN coefficients. Nevertheless, the pairwise product MSCN coefficients follow quite precisely AGGD curves so that we chose here to create a detector based just on horizontal MSCN coefficients.

In our code we split the data set of images in two, so that a first train subset can be used for the creation of the detector, and the other validation subset can be used to test the efficiency of the detector.

Here is what we get using a train subset of 50 images and applying the detector to two of the validation images:

```
original (H0 or H1): 1 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
    FGSM (H0 or H1): 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
-----------------------------------------------------------------------------------------------------------------------
original (H0 or H1): 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0
    FGSM (H0 or H1): 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Fig. 6.2: Kolmogorov-Smirnov tests computed on the original and the FGSM attacked version for a bank of parameters of 50 images, and on two different CIFAR10 images

As the code executed to get the above picture took few minutes, we just showed the tests for the original image and the FGSM attacked one. Nevertheless, the results are encouraging because most of the time, the original image is effectively detected as non-attacked, while the FGSM attacked image is almost all the time detected as attacked. So, with a level $\lambda = 10\%$ for instance, we would get a perfect detection for those four images (the two originals and the two attacked). And even more important, no false positive was made.

As the results for the tests are fairly accurate we try the same procedure again but this time with just 15 images in the training set, and with the three different types of attacks.

```
original (H0 or H1): 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0
    FGSM (H0 or H1): 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     PGD (H0 or H1): 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      DF (H0 or H1): 0 1 0 0 1 0 1 1 0 0 0 0 0 1 0
-------------------------------------------------
original (H0 or H1): 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0
    FGSM (H0 or H1): 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
     PGD (H0 or H1): 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
      DF (H0 or H1): 1 1 1 1 1 0 0 1 1 0 1 0 1 0 0
```

Fig. 6.3: Kolmogorov-Smirnov tests computed on the original and the three attacked versions for a bank of parameters of 15 images, and on two different CIFAR10 images

As we could expect, the tests for the DF attack have a very similar pattern to the original image. So our detector will not be able to identify this attack. However, for the FGSM and the PGD attacks, the results are pretty good. As before, the tests almost always consider the images attacked. Thus, our detector will very probably nicely work with those two attacks. Notice also, that even if the DF attack fools the detector, it makes no false positives which is a very important point. So let now try the detector with some attacked and non-attacked images.

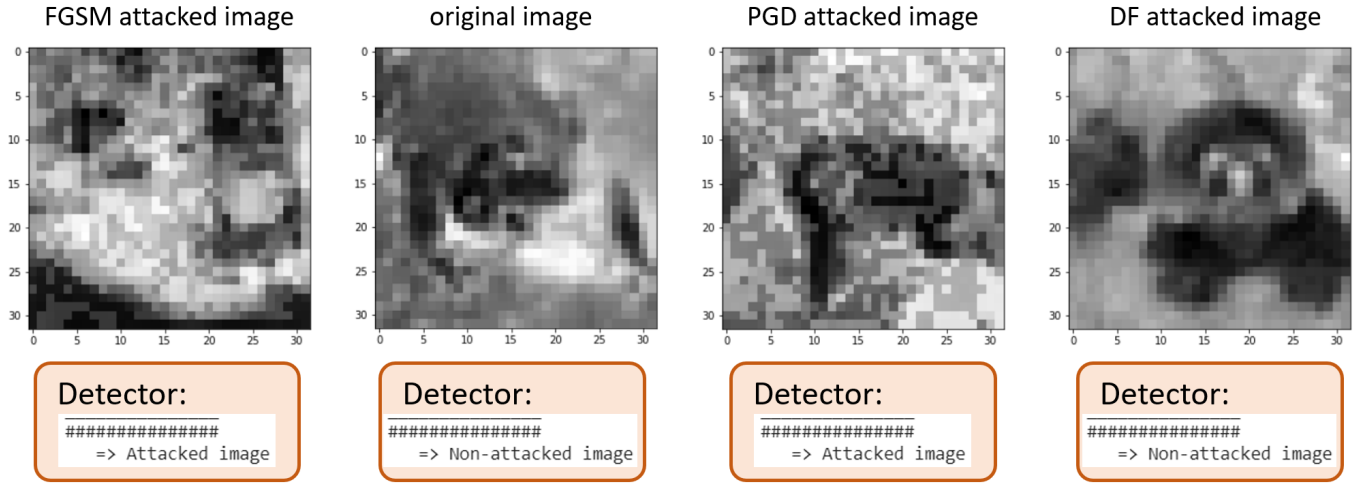


Fig. 6.4: Detector used on four images (attacked or not) for the horizontal MSCN coefficients with $\lambda = 10\%$

These four images were randomly chosen in the validation set. As observed so far, the original image as well as the FGSM and the PGD attacked ones are all correctly identified by the detector. However, the DF attacked image passes through the net of our detector. But importantly, we succeed in having as less false positives as possible.

Conclusion

After describing the different models and attacks, we were able to compare their accuracy. We then defined the MSCN coefficients, explained their properties and wherefore they are interesting for characterising images. These coefficients were then computed for images in the MNIST and CIFAR10 data sets. Thereby, using the MSCN properties, we were able to issue statistical tests on these different images. Finally, we created a detector for detecting whether an image is attacked or not.

You can notice that our detector is able to detect PGD and FGSM attacks. However, it is still difficult to detect subtler attacks such as DeepFool. Given the execution times of our algorithms, it seems that they are not optimized, compared for instance with an SVM classifier available on Python which gave accurate results to the detection issue. To go further, our algorithms could be computed for taller pictures, in the sense of images with more pixels that will be more natural, which could thus give more convincing results. Furthermore, the detection could be tried on color images instead of grey ones, to gain once again more naturalness.

Bibliography

State of art

[KM19a] Zico Kolter and Aleksander Madry. *Introduction to adversarial examples*. 2019. URL: https://adversarial-ml-tutorial.org/introduction/.

[KM19b] Zico Kolter and Aleksander Madry. *Adversarial examples, solving the inner maximization*. 2019. URL: https://adversarial-ml-tutorial.org/adversarial_examples/.

[3Bl17a] 3Blue1Brown. *But what is a Neural Network? — Deep learning, chapter 1*. 2017. URL: https://www.youtube.com/watch?v=aircAruvnKk.

[3Bl17b] 3Blue1Brown. *Gradient descent, how neural networks learn — Deep learning, chapter 2*. 2017. URL: https://www.youtube.com/watch?v=IHZwWFHWa-w.

[Aru19] Arunava. *DeepFool — A simple and accurate method to fool Deep Neural Networks.* 2019. URL: https://towardsdatascience.com/deepfool-a-simple-and-accurate-method-to-fool-deep-neural-networks-17e0d0910ac0.

[Wik20a] Wikipedia. *Generalized normal distribution*. 2020. URL: https://en.wikipedia.org/wiki/Generalized_normal_distribution.

[Wik20b] Wikipedia. *Kolmogorov–Smirnov test*. 2020. URL: https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test.

[Ins17a] Arxiv Insights. *'How neural networks learn' - Part I: Feature Visualization*. 2017. URL: https://www.youtube.com/watch?v=McgxRxi2Jqo.

[Ins17b] Arxiv Insights. *'How neural networks learn' - Part II: Adversarial Examples*. 2017. URL: https://www.youtube.com/watch?v=4rFOkpI0Lcg.

Models and attacks

[PyT17] PyTorch. *Gradient descent, how neural networks learn — Deep learning, chapter 2*. 2017. URL: https://pytorch.org/tutorials/beginner/nn_tutorial.html.

Characterise images with MSCN coefficients

[MMB09] A. Mittal, A. K. Moorthy, and A. C. Bovik. *No-Reference Image Quality Assessment in the Spatial Domain*. 2009.

[LSB09] N.-E. Lasmar, Y. Stitou, and Y. Berthoumieu. *Multiscale skewed heavy tailed model for texture analysis*. 2009.

[Lan20] The Julia Programming Language. *Convolutions in image processing — Week 1 — MIT 18.S191 Fall 2020 — Grant Sanderson*. 2020. URL: https://www.youtube.com/watch?v=8rrHTtUzyZA.