IMT Atlantique
Dépt. Électronique
Technopôle de Brest-Iroise - CS 83818
29238 Brest Cedex 3
Téléphone: +33 (0)2 29 00 13 04
Télécopie: +33 (0)2 29 00 10 12
URL: **www.imt-atlantique.fr**

Summer internship report

Helon Moreira Freitas

Internship at IMT Atlantique on deep networks and adversarial noise

September 7, 2018

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Contents

1. Introduction

The main subject of this work is to study adversarial noise in deep neural networks (DNNs). Actually, the challenge is to research how to increase the robustness of DNNs in the presence of adversarial examples.

The first thing to understand is adversarial examples, what they are and how they influence the decisions of a neural network. Indeed, an image can be subtly modified (in a way that even humans can't be able to notice some difference in respect of the original image). In practice, this modification was made by adding noise in each pixel of the image in the same direction as it's gradient in relation to loss (using the backward technique to calculate it).

Despite the fact that the new image looks unaltered for humans, for a neural network it can cause a misclassification. It's possible to imagine how dangerous this can be, for example, a person can make small changes in his face, perhaps with makeup, that can make a facial recognition flawed. Indeed, the real challenge is to find out how to mitigate this effect in a controlled way.

This report consists of a brief description of all the parts in the neural network, some points more detailed, and the results of the tests made with adversarial examples.

Finally, this work is an extension of the ideas presented on the article [5] which shows the result for CIFAR dataset. The main idea is to form graphs in each layer of the DNN that can represent the similarity between the examples used in the training in order to have an intermediate representation of this images in each layer. So the idea is to regularize the network in such a way as to prevent large changes between neighboring layers, that is, making the network somehow smooth.

To summarize this previous work, it tries to control the difference of graph smoothness between the layers by making the curve of learning more continuous (trying to learn without abrupt changes). This was shown to create DNNs that are more robust to adversarial examples in a specific dataset. In this work the challenge was to explore the same concept in different datasets.

2. The data

Each type of dataset used in this work (CIFAR, SVHN and tiny ImageNet-200) went through a specific transformation for it to be exploited by the model. Because of that property, the code was separated in different files (cifar.py , svhn.py , tiny_imageNet-200.py), despite only having changed a small part of the code. It can be inferred here that a continuation of this work would be to create a more modular code.

2.1. CIFAR-10 and CIFAR-100

The CIFAR-10 dataset consists of 50000 training images and 10000 test images. They are 32x32 colour images (RGB) distributed in 10 classes, with 6000 images per class.

The CIFAR-100 dataset consists of 50000 training images and 10000 testing images in a total of 100 classes, with 600 images per class. For more details about the dataset, please visit: The CIFAR-10 and CIFAR-100 datasets.

Actually, this data was used first just to be compared with previous results of tests in the original work. Once the results were verified, showing compatibility with previously acquired tests, it was possible to migrate to new datasets.

To set up the dataset of training, normalization was used, that means that for each channel the mean and the standard deviation of all the pixels were calculated and used Gaussian normalization.In more detail, in each of the three layers of the images that are part of the training, the average of the 50000x32x32 values in each pixel was made. Another thing about the training is that the batch is randomly chosen. So, in each epoch could be considered that the net was indeed learning in a different form every time that the DNN was initialized, without influence of the order of the images in dataset.

2.2. SVHN

The SVHN (The Street View House Numbers) dataset consists of 73257 images of digits (0 - 9) for training, 26032 images for testing, and 531131 as extra data. There are 10 classes, one for each digit (1 for '1' ...). For more details about the dataset, please visit: The Street View House Numbers (SVHN) Dataset .

To set up the dataset of training, in this case, was just to transform the images in tensor, without normalization. The batch is randomly chosen as was made with CIFAR. As was used in [3].

2.3. Tiny ImageNet-200

The Tiny ImageNet-200 dataset consists of 100000 images for training 10000 validation images, and 10000 test images all equally distributed in 200 classes. To set up the dataset of training, normalization was used using mean and standard deviation of the ImageNet dataset as an approximate form, since these values were not calculated for the Tiny ImageNet-200. As was done in the other cases the batch is randomly chosen. For more details about the dataset, please visit: Tiny ImageNet Visual Recognition Challenge.

2.3.1. Discussion: Possible improvements in the preprocessing of Tiny ImageNet-200

As this is not a classical dataset with well defined state of the art and training methods, it was necessary to do some hyperparameter search using the accuracy on the validation set as the main optimization objective. In our first tests we used the following characteristics:

- batch_size = 50
- learn_rate = 10^{-3} for epoch = [0,10]
- learn_rate = 10^{-4} for epoch = [10,20]
- original network (without modification)

As result, the validation scores were not good enough. Scores around 32% of accuracy (with a train without adversarial examples).

Changes that have been attempted disregarding the possibility of modifying the existing layers of the network.

1. Elevate the number of epochs to 40 (accuracy < 40%)
2. Item 1 + learn_rate = 10^{-4} for epoch = [0,20] and learn_rate = 10^{-5} for epoch = [20,40] (accuracy = 42%)
3. After seeing that the tests that increased the number of epochs did not work, one possible hypothesis was that the network was not deep enough. We increased the depth by adding two more blocks but the accuracy didn't increase very well.
4. Given that increasing the depth did not really helped, another test was to increase the depth and reducing the width, it also did not work.
5. Finally we introduce the idea of a max pool before layer 1, which had already been tested in [6] and the accuracy increased from (+/-)42% to 62.44%(using data augmentation already). At this point it was defined that it was a good start and it was then tested with other solutions (training with Adversarial example, smoothness difference cost, Parseval, all with data augmentation).

3. Adversarial Example

There are many ways to create the adversaries images, as studied in [1]. In this work, the decision was to apply a small perturbation in the direction of the gradient for each pixel. This is also called Fast Gradient Sign Method (FGSM) and was first introduced in [4].

Firstly the algorithm requires a forward pass on the network, which can then be backpropagated to generate the gradients of the image in relation to a given loss ($\mathcal{L}$).

With the gradients in hand, the algorithm can now generate the adversarial noise. The perturbation δ_x is an adversary perturbation created using the fast gradient sign method [4]:

$$\tilde{\mathbf{x}} = \mathbf{x} + \varepsilon\ sign(\nabla_x \mathcal{L}) = \mathbf{x} + \delta_x, \tag{1}$$

To bound the noise we used the notion of Signal over Noise (SnR). Considering $\mathbf{x}$ as our image and $\delta_{\mathbf{x}}$ as the perturbation we can now define:

$$SNR(\mathbf{x}, \delta_x) = 20 \log_{10} \frac{||\mathbf{x}||_2}{||\delta_x||_2}. \tag{2}$$

3.1. Training with adversarial examples

Following the guidelines from [4] we also trained our networks using adversarial examples. This includes using a Gaussian random variable that is truncated between $-2std$ and $2std$. The standard deviation was calculated so that the magnitude of $\delta_{\mathbf{x}}$ would be equivalent to the magnitude used when the SnR is equal to 33. In order to generate this magnitude (also called ϵ) we calculated the mean magnitude needed over the test set. This can be considered incorrect because we are using data from the test set during training, but the interest here is to compare the approaches over the same values, not to research for the hyperparameters. Further testing without this method would be advised.

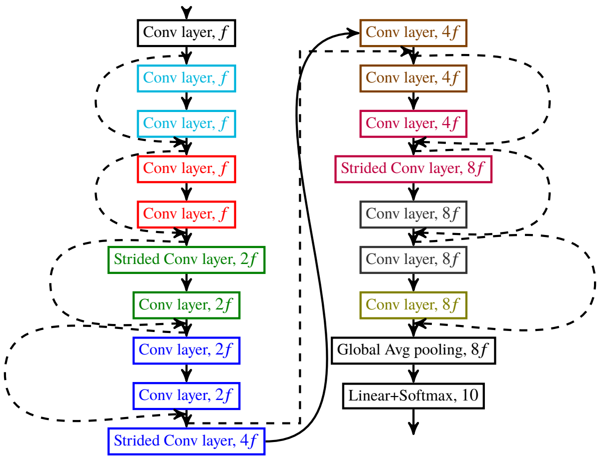
4. The neural network architecture

The neural network architecture, or model, used in this work was a residual network (ResNet-18), but there are differences between the models used for each dataset.

The starting point was the network used in the previous work [5] (which uses the CIFAR dataset). The original structure was preserved for the SVHN dataset. On the other hand, the network model was extensively modified in order to obtain better results for the TinyImageNet-200 base date.

4.1. The neural network

The basic structure that is used on the SVHN and CIFAR datasets is represented below:



The structures for the Tiny ImageNet-200 dataset is basically the same, but with a pooling after the first conv layer.

4.2. Discussion: Remarks from applying the same algorithm on new datasets

Applying a method to a new dataset is a tricky problem. First one has to find the correct hyperparameters for the new problem and even then some small tweaks can be needed to better adapt the method to this new reality.

4.2.1. Hyperparameter search

The choice of hyperparameters is made in "baby steps" (to see more, please visit: Transfer Learning and Fine-tuning Convolutional Neural Networks), for example, the number of epochs starts with a small value and it increases if it notices that the network can learn more in the training (so the idea is to make a longer training).

Another parameter that needs attention is the learning rate. In order to determine a good value for the learning rate and the other hyperparameters one should use either cross validation or a simple validation set. In the case of this internship, given the time constraints an alternative method was used where one studies the graph of the loss function of the training set. This is useful to avoid underfitting, but cannot guarantee a lack of overfitting.

Indeed, the behavior of the curve indicates how the network is learning, if the graph is almost horizontal this means that the network is not learning anymore and maybe it's better to change some parameter, for example, the learning rate. If the graphic does a "zig-zag" or form some more aleatory this could mean that the structure of the network or the training is not adequate for this dataset, or even that maybe there is something wrong in the code.

In this work, two different types of optimizers where used, Stochastic Gradient Descent (SGD) with momentum and Adam. The latter was faster to learn, but in the end SGD was used because it was more robust to overfitting given the basic parameters.

4.2.2. Improving the smoothness regularization on data augmented cases

In our first tests the smoothness solution was not helping in the SVHN dataset when we used adversarial data augmentation for the training of the network. This is not surprising as in [5], the authors had the same problem with non-adversarial data augmentation. The modification that brought better results was to only use the non augmented part of the batch in the regularization. This was only tested for adversarial data augmentation, future work would include doing the same for non-adversarial data augmentation.

5. The results

All the tests were executed using adversaries images with different Signal-to-Noise Ratio (SnR). The values of the parameters (β, γ, m) were chosen by looking at the values of best results in the CIFAR dataset from [5]. The tables below shows the results of this tests for each type of data. Sometimes it was decided to use the same configuration (like in "Adv + smooth" and "smooth") but using other values for β, γ and m, looking for better results.

Table 1: Adversarial tests on SVHN

Network type	beta	gamma	m	Clean	50	45	40	33
Vanilla	0	0	0	97.87%	95.57	92.99	88.15	77.72
Adversary (Adv)	0	0	0	97.68%	95.25	93.65	91.33	86.78
Smooth	0	0.01	2	96.63%	92.80	89.18	82.73	69.37
Smooth	0	0.003	2	97.22%	94.26	90.96	84.40	71.08
Smooth	0	0.001	1	97.46%	96.82	96.20	94.66	**88.70**
Adv + Psv	0.01	0	0	97.97%	97.43	96.96	95.79	**91.17**
Adv + Smooth	0	0.001	1	97.46%	96.82	96.20	94.66	88.70
Adv + Smooth	0	0.01	2	97.27%	96.43	95.72	94.15	88.23
Adv + Smooth	0	0.001	2	97.25%	94.43	95.59	96.49	96.45
Adv + Psv + Smooth	0.01	0.001	1	97.99%	97.47	96.92	95.74	**91.09**

For SVHN dataset it is possible to take some notes, such as:

- Looking at the column of SnR = 33: comparing "Vanilla" with "Adv" it's possible to notice a better prediction when the training set includes adversarial examples. So, looking for "Adv + Psv" and "Adv + Smooth" there is progress, which are more significant when Parseval was used (86.78% to 91.17%, against 88.7% with smoothness).
- No reason was found for the strange result where tests with the smoothness regularizer had worse results than the Vanilla network... Given the different results over multiple executions, using the mean and standard deviation is advised.
- Note that for $\gamma = 0.001$ and $m = 2$ the result for the adversarial + smoothnes case was better. Maybe this are the correct hyperparameter values that should be used in all the cases using smoothness on this dataset. The values don't seem to carry from a dataset to another.
- Note that in the case that we found the best accuracy there is an incompatibility logic, as the SnR decrease the accuracy increased. Further investigation of this phenomena is needed, which would probably involve adding noise in steps, and considering that if an image fails for SnR = 50, it will also fail for SnR = 33.

Table 2: Adversarial tests on Tiny ImageNet-200

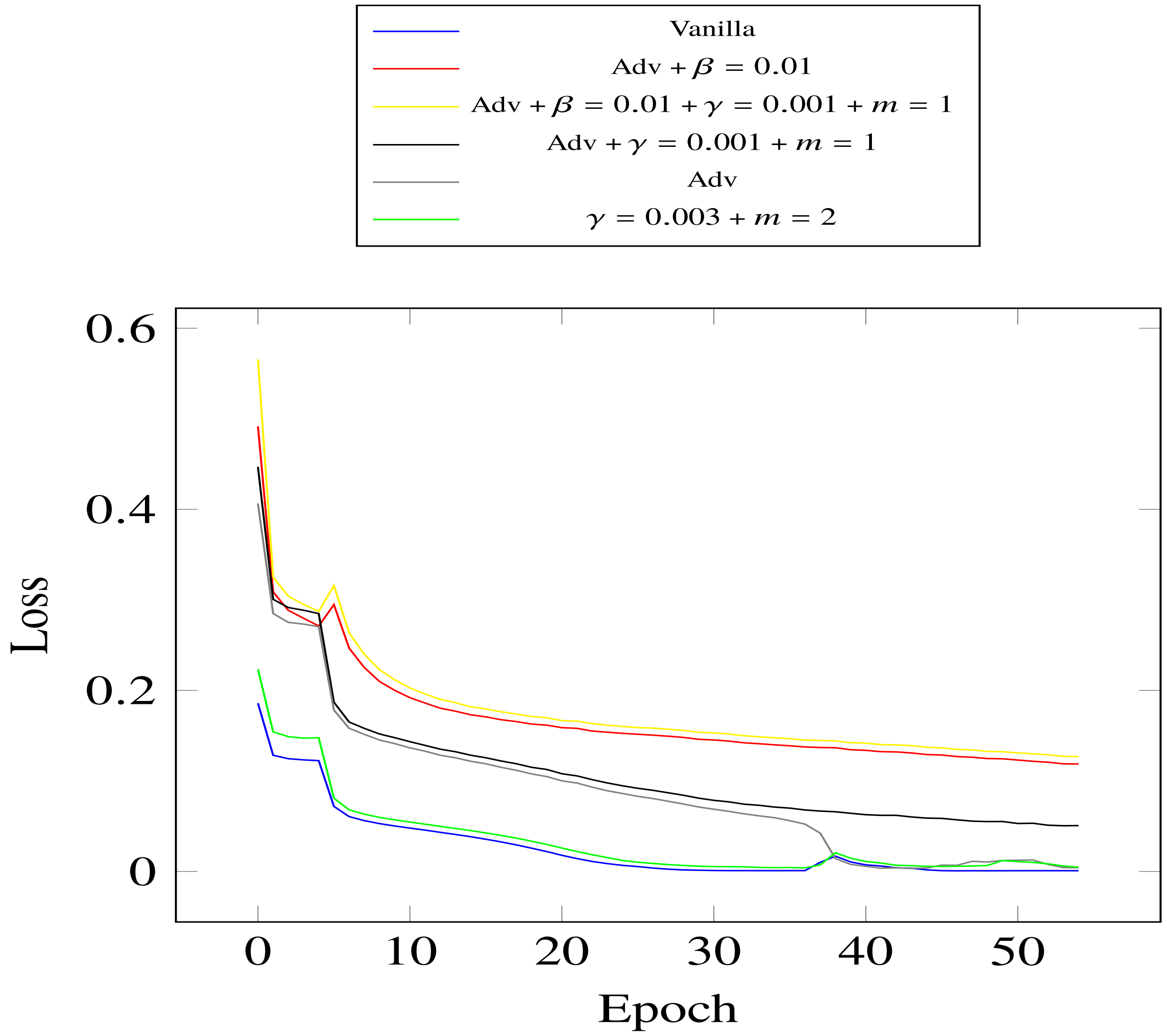
Network type	beta	gamma	m	Clean	50	45	40	33
Vanilla	0	0	0	49.11%	2.09	0.5	0.24	**0.16**
data augmentation (da)	0	0	0	62.44%	11.94	4.53	2.7	1.91
Adversary (Adv) + da	0	0	0	61.00%	41.65	29.81	15.84	3.08
Smooth + da	0	0.01	2	63.81%	18.56	7.38	2.67	**1.44**
Adv + Psv + da	0.01	0	0	56.85%	46.86	39.47	28.72	12.36
Adv + Smooth + da	0	0.01	2	61.32%	42.08	29.86	15.83	**3.17**
Adv + Psv + Smooth + da	0.01	0.001	1	57.16%	47.21	39.92	29.7	13.23

For tiny ImageNet dataset:

- In the first place, it is evident the effect of the adversary examples making the network to misclassify.
- This data is really difficult to learn and one of the reasons is that all the images has a "bad" resolution (made by cutting some piece of bigger images). Just doing a quick check looking at the data found that there was an image of a frog among the class that should only contain lizards images.
- All the training sessions were done with at least non adversarial data augmentation in order to obtain bigger value of accuracy. But unfortunately, it is noted that the smoothness does not work well with da, again this is not surprising as it was the case in [5].
- As it was noted in the SVHN dataset the use of Parseval training influenced significantly in the improvement of the predictions, even without achieving the same performance as in the other datasets.

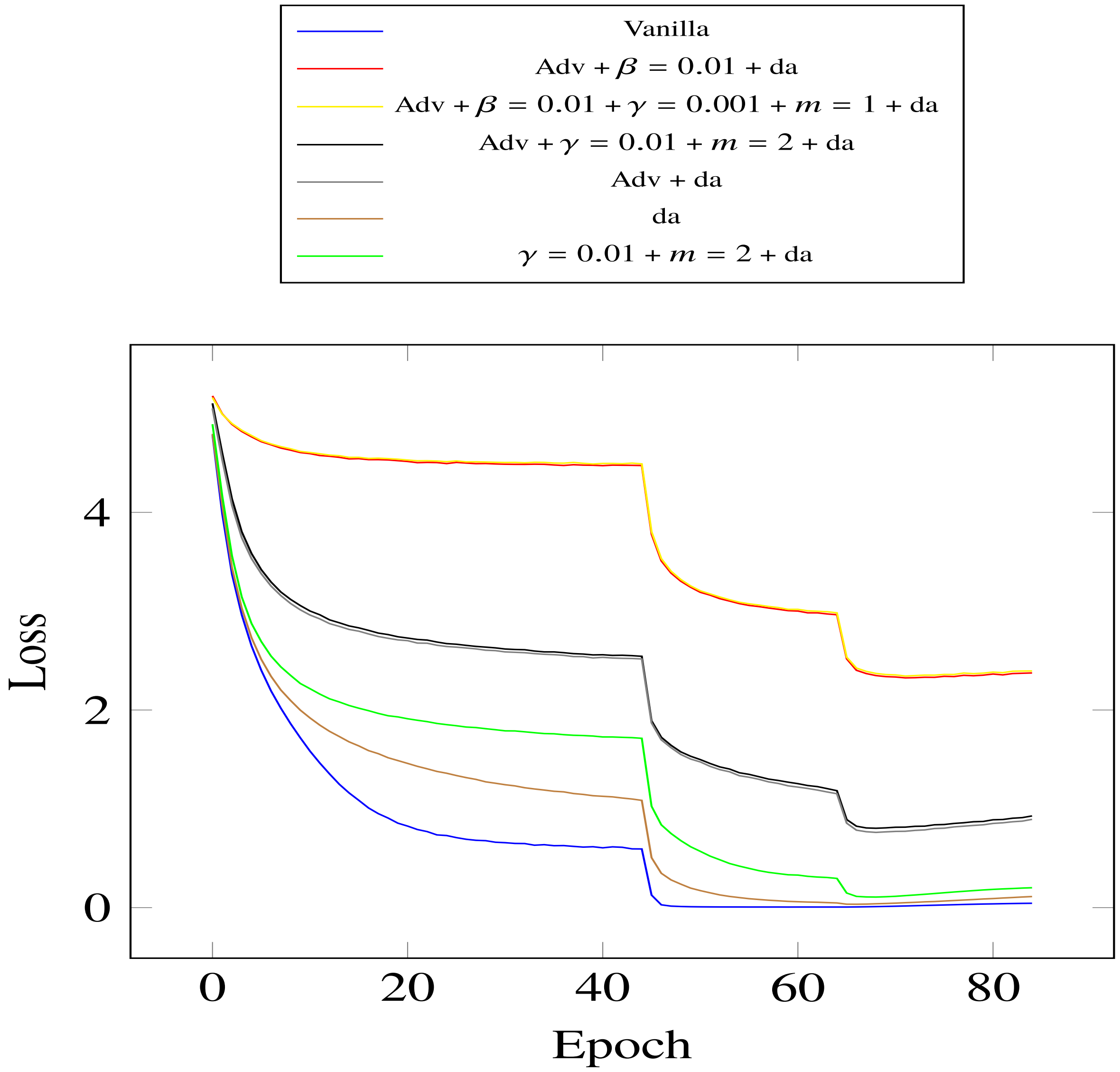
Below, the loss function graph for the two datasets:

Figure 1: Loss function under different training for SVHN



As can be observed the loss cost has lower values in vanilla and "smooth". Larger values can be seen where Parseval was used. It is also noted the moment when the learning rate was changed at the beginning (in epoch 5) and how this made the network learn more, instantly.

Figure 2: Loss function under different training for Tiny ImageNet-200



Unlike how it was done with SVHN, it was chosen to wait more epochs to change the parameters. Note that in this case, the same behavior is noted for the use of Parseval training. The loss values are higher although the accuracy is better than in the other cases.

6. Conclusion

As it can be verified, introducing adversarial noise to the images can cause confusion in the neural network. This can be extremely dangerous. For example, a simple scenario can be imagined where the attack can be done on traffic signs or lights, in order to make autonomous cars mistake their interpretation of the reality.

All the work done here is accessible and is modifiable by anyone. It can be found in: moreirahelon/test_adversary). One of the ideas that can be important is to use snapshot ensembles (used in [2]). There are others types of networks that could be use as studied in [6], but by now it seems that resnet are better (particularly [6] shows that using bottleneck resnet blocks instead of classical resnet blocks helped to increase the accuracy).

For future work, there is still using a more correct criteria for searching the hyperparameters, which can improve the accuracy of the tests in all the datasets used here. Lastly it would be of great importance to improve the code so that it is more modular, which means that it would be more friendly for new implementations.

References

[1] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017.

[2] Zach Barnes, Frank Cipollone, and Tyler Romero. Techniques for image classification on tiny-imagenet.

[3] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. Parseval networks: Improving robustness to adversarial examples. *arXiv preprint arXiv:1704.08847*, 2017.

[4] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[5] Carlos Eduardo Rosar Kós Lassance, Vincent Gripon, and Antonio Ortega. Laplacian power networks: Bounding indicator function smoothness for adversarial defense. *CoRR*, abs/1805.10133, 2018.

[6] Hujia Yu. Deep convolutional neural networks for tiny imagenet classification.

OUR WORLDWIDE PARTNERS UNIVERSITIES - DOUBLE DEGREE AGREEMENTS

3 CAMPUS, 1 SITE

IMT Atlantique Bretagne–Pays de la Loire – **http://www.imt-atlantique.fr/**

Campus de Brest
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 3
France
T +33 (0)2 29 00 11 11
F +33 (0)2 29 00 10 00

Campus de Nantes
4, rue Alfred Kastler
CS 20722
44307 Nantes Cedex 3
France
T +33 (0)2 51 85 81 00
F +33 (0)2 99 12 70 08

Campus de Rennes
2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
France
T +33 (0)2 99 12 70 00
F +33 (0)2 51 85 81 99

Site de Toulouse
10, avenue Édouard Belin
BP 44004
31028 Toulouse Cedex 04
France
T +33 (0)5 61 33 83 65

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom