Probabilistic Graphical Models
Are Generative Classifiers More Robust to Adversarial Attacks?

Manal Akhannouss, ENS Paris-Saclay
manal.akhannouss@eleves.enpc.fr

Paul Barbier, ENS Paris-Saclay
paul.barbier@eleves.enpc.fr

Alexandre Lutt, ENS Paris-Saclay
alexandre.lutt@eleves.enpc.fr

I. Introduction

In this report, we will study the robustness of generative classifiers to adversarial attacks, compared to discriminative classifiers. In fact, neural networks are known to be particularly weak to adversarial networks, as proved by Goodfellow et al. in [**?**].

The main issue is that neural networks are often overconfident in their predictions, which makes them very sensitive to small perturbations of the input, even if these perturbations are imperceptible to the human eye. This issue is particularly problematic in the case of safety-critical applications, such as autonomous driving, where a small perturbation of the input can lead to a catastrophic failure, or in the case of medical applications, where a small perturbation of the input can lead to a wrong diagnosis.

Those attacks can take different forms, depending on the knowledge of the attacker regarding the model. If the attacker has access to the model's parameters and architecture, we will talk about white box attacks. Otherwise, we will talk about black box attacks. In both cases, the attacker aims to create adversarial examples, *i.e.* samples which are very similar to the original samples, but which are misclassified by the model.

Knowing this, several methods have been developped to detect adversarial examples. In our case, since we mostly aim to discuss the robustness of the models, we will focus the logits-based detection method introduced in [**?**], which aims to reject inputs using the joint density $p(\boldsymbol{x}, \boldsymbol{y})$ of the generative classifier.

II. Models

In this section, we will present the different models we used for our experiments. More precisely, we will introduce generative models, present the two classifiers we used for our experiments as well as their respective architecture. Then, we will present the dataset that we worked with, and provide details about the training procedure of our models.

II.1. Discriminative versus generative models

Let us consider a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ of N samples, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is a d-dimensional feature vector and $\boldsymbol{y}_i \in \mathcal{Y}$ is the corresponding label. A discriminative classification model (or discriminative classifier) aims to estimate the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$, *i.e.* the probability that the label $\boldsymbol{y}$ is associated to the feature vector $\boldsymbol{x}$. On the other hand, a generative classification model (or generative classifier) aims to estimate the joint probability $p(\boldsymbol{x}, \boldsymbol{y})$, *i.e.* the probability of observing the feature vector $\boldsymbol{x}$ and the label $\boldsymbol{y}$ at the same time.

Both models can be used for classification purposes, *i.e.* can be used to predict the label $\boldsymbol{y}$ associated to a feature vector $\boldsymbol{x}$, but with very different interpretations. The discriminative classifier will directly estimate the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$, while the generative classifier will estimate the joint probability $p(\boldsymbol{x}, \boldsymbol{y})$ for each possible value of $\boldsymbol{y}$, and then use the Bayes rule to estimate the conditional probability $p(\boldsymbol{y}|\boldsymbol{x}) = \dfrac{p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y})}{p(\boldsymbol{x})}$.

Let's compare the advantages and drawbacks of both approaches. The discriminative approach is often more efficient in practice, since it only needs to estimate the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$, which is often much simpler than estimating the joint probability $p(\boldsymbol{x}, \boldsymbol{y})$. However, the discriminative approach does not provide any information about the distribution of the features $\boldsymbol{x}$, which can be useful in some cases. On the other hand, the generative approach provides information about the distribution of the features $\boldsymbol{x}$, but it is often more difficult to implement and less efficient in practice if one is only interested in classification purposes. Moreover, generative models are also better at fitting classes separately, which can be useful in the case of imbalanced datasets, since in the case of discriminative models, all parameters interact.

One of the most common generative classifier is Naive Bayes. This simple model assumes a factorised distribution $p(\boldsymbol{x}|\boldsymbol{y}) = \prod_{i=1}^{d} p(\boldsymbol{x}_i|\boldsymbol{y})$, which means that the features are independent given the label. This assumption is often far from being verified in practice for image datasets. For this reason, in the following, we will follow the path of [**?**] and use a latent-variable model $p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ to design our generative classifier.

Note that this model does not assume a factorised distribution for $p(\boldsymbol{x}|\boldsymbol{y})$, since in this case $p(\boldsymbol{x}|\boldsymbol{y}) = \frac{\int p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) dz}{\int p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) d\boldsymbol{x} d\boldsymbol{y}}$. In order to fully define a latent-variable model, we need to explicitely chose a structure for $p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. At this point, several choices are possible:

$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{x}) p(\boldsymbol{z}|\boldsymbol{x}) p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z}) \tag{DFX}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{z}) p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z}) \tag{DFZ}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{x}) p(\boldsymbol{z}|\boldsymbol{x}) p(\boldsymbol{y}|\boldsymbol{z}) \tag{DBX}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{z}) p(\boldsymbol{y}|\boldsymbol{z}) p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z}) \tag{GFZ}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{y}) p(\boldsymbol{z}|\boldsymbol{y}) p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z}) \tag{GFY}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{z}) p(\boldsymbol{y}|\boldsymbol{z}) p(\boldsymbol{x}|\boldsymbol{z}) \tag{GBZ}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{y}) p(\boldsymbol{z}|\boldsymbol{y}) p(\boldsymbol{x}|\boldsymbol{z}) \tag{GBY}$$

In those acronyms, D stands for *discriminative*, G stands for *generative*, F stands for *fully-connected graph*, and the last letter indicates on which variable we assume a prior distribution (determined with $\mathcal{D}$ in the case of X and Y). In our case, we will focus on the DFZ and GFZ structures, in order to be able to compare the discriminative and generative approaches, but everything that we will see can easily be extended to the other structures.

II.2. Classifiers architecture

As mentionned above, we will consider two different classifiers. The first one will be a simple discriminative classifier (with DFZ structure), while the second one will be a generative classifier (with GFZ structure). More specifically, we will consider the following architectures (all of them include dropout during training for regularization purposes):

- **Discriminative classifier:** In this case, we use two neural networks to approximate $p(\boldsymbol{x}|\boldsymbol{z})$ and $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$ respectively. More precisely, we use a convolutionnal neural network with 3 convolutions layers with ReLU activations, followed by 3 dense layers, to approximate $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$. On the other hand, $p(\boldsymbol{x}|\boldsymbol{z})$ is obtained after a sequence (with dropout during training) of 2 dense layers and 3 transposed convolutions. Note that we use a transposed convolution instead of a simple convolution, in order to be able to reconstruct the original image size.

- **Generative classifier:** This case differs from the previous one in the sense that we use a simple 3-layers MLP to approximate $p(\boldsymbol{y}|\boldsymbol{z})$ as well as a sequential model with 2 dense layers and 3 transposed convolutions for $p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z})$.

Note that in both cases, we use the amortised approximate posterior $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{z})$ for training. More specifically, we learn both $p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ and $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{z})$ by maximizing the variational lower bound as in [**?**]:

$$\mathbb{E}_{\mathcal{D}}[\mathcal{L}_{VI}(\boldsymbol{x}, \boldsymbol{y})] = \frac{1}{N} \sum_{i=1}^{N} \mathbb{E}_q \Big[\log \frac{p(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{z}_n)}{q(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{y}_n)} \Big]$$

We also use a convolutionnal neural network similar to the one used to model $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$ to approximate $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$.

II.3. Dataset

For all of our experiments, we used the Fashion MNIST dataset, which is a dataset of 70 000 grayscale images of size 28×28 pixels, each associated to one of 10 classes. The dataset is divided into a training set of 60 000 images and a test set of 10 000 images. The reason for this choice comes from the very high accuracies obtained by [?] on the original MNIST dataset. In order to make the classification task more challenging, we decided to use the Fashion MNIST dataset instead, which is very similar to the original MNIST dataset, but with more complex images (fashion items *versus* handwritten digits).

II.4. Training

In order to train our models, we used the Adam optimizer with a learning rate of 10^{-3}, a batch size of 100, and we trained our models for 50 epochs on a NVIDIA V100 GPU with 16GB of RAM. Note that we did not use any data augmentation technique, since we wanted to focus on the robustness of our models to adversarial attacks, and not on the robustness of the data augmentation techniques themselves.

III. Adversarial attacks

In this section, we will present the different methods we used to create adversarial attacks on our models. These methods can be divided into two categories: white box attacks and black box attacks, depending on the knowledge of the attacker regarding the model. If the attacker has access to the model's parameters and architecture, we will talk about white box attacks. Otherwise, we will talk about black box attacks.

In all cases, we aim to design attacks which are as imperceptible as possible to the human eye, while changing the predicted label of the model. In other words, we aim to solve the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\eta}} \quad & \|\boldsymbol{\eta}\|_2 \\ \text{s.t.} \quad & \underset{\boldsymbol{y} \in \mathcal{Y}}{\text{argmax}} \, (p(\boldsymbol{y}|\boldsymbol{x} + \boldsymbol{\eta})) \neq \underset{\boldsymbol{y} \in \mathcal{Y}}{\text{argmax}} \, p(\boldsymbol{y}|\boldsymbol{x}) \end{aligned} \tag{Adversarial Attack Problem}$$

where $\boldsymbol{\eta}$ is the adversarial perturbation, $\boldsymbol{x}$ is the original sample, and $p(\boldsymbol{y}|\boldsymbol{x})$ is the conditional probability of the label $\boldsymbol{y}$ given the sample $\boldsymbol{x}$. Note that as we are considering a multiclass classification problem, we will implement the different attacks as one-vs-all attacks, *i.e.* we will create adversarial examples to change the prediction of the model for each class independently.

It is important to note that in order to generate our attacks, we will sample new values for the latent variable $\boldsymbol{z}$, and then use the generative model with those values. This is a crucial step, since it we cannot assume that the attackers has accces to the random-number generator of the model.

In order to compare the different algorithmic solutions to this problem, we will keep using the Fashion-MNIST dataset mentionned above, and we will compare the different approches by computing the accuracy of the model on the adversarial examples, as well as the average robustness $\rho = \frac{1}{n_{\text{samples}}} \sum_{i=1}^{n_{\text{samples}}} \frac{\|\boldsymbol{\eta}_i\|_2}{\|\boldsymbol{x}_i\|_2}$ of the model to the adversarial perturbations.

III.1. White box attacks

III.1.1 Fast Gradient Sign (FGS) method

We first implemented the Fast Gradient Sign (FGS) method first introduced in [?] and later mentionned in [?], which is a simple and efficient method to create adversarial attacks. Let us consider a model with a model loss function J_θ and a sample $(\boldsymbol{x}, \boldsymbol{y})$. The FGS method consists in adding a perturbation $\boldsymbol{\eta}$ to the input $\boldsymbol{x}$, with $\boldsymbol{\eta} = \varepsilon \times \text{sign}(\boldsymbol{\nabla}_{\boldsymbol{x}} J_\theta(\boldsymbol{x}))$, where ε is a hyperparameter which controls the magnitude of the perturbation.

This method is very simple to implement, and cheap to run (since it only requires one gradient computation), but it comes with a practical drawback; the perturbations are often very visible to the human eye, which makes them less realistic. This comes from the fact that the ε hyperparameter is often chosen to be too large, in order to ensure that the perturbation is large enough to fool the model "often enough".

III.1.2 DeepFool method

In order to create more subtle perturbations, we then implemented the DeepFool algorithm. This algorithm was first introduced in [?], as a way to create small adversarial examples for deep neural networks. This algorithm has since grown in popularity, and has been used in several other papers and benchmarks, such as [**?**].

The idea behind it is to iteratively compute the minimal perturbation $\boldsymbol{\eta}$ which is required to change the prediction of the model. More precisely, let us consider a model with a model loss function J_θ and a sample $(\boldsymbol{x}, \boldsymbol{y})$. The DeepFool algorithm consists in iteratively computing the minimal perturbation $\boldsymbol{\eta}$ which is required to change the prediction of the model, with $\boldsymbol{\eta} = \dfrac{J_\theta(\boldsymbol{x}, \boldsymbol{y})}{\|\boldsymbol{\nabla}_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, \boldsymbol{y})\|_2^2} \boldsymbol{\nabla}_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, \boldsymbol{y})$.

The idea comes from the fact that, if the model is linear, the decision boundary is a hyperplane of equation $\boldsymbol{w}^T \boldsymbol{x} + b = 0$, and the minimal perturbation required to change the prediction of the model is the orthogonal projection of the sample on the decision boundary, *i.e.* $\boldsymbol{\eta} = -\dfrac{\text{sign}(\boldsymbol{w}^T \boldsymbol{x} + b)}{||\boldsymbol{w}||_2^2} \boldsymbol{w}$. In the non-linear case, the decision boundary is not a hyperplane anymore, but this heuristic still works well in practice. More precisely, our implementation of the DeepFool algorithm is described in **??** (where we can chose another p-norm to optimise on, even if in practice we mostly focused on the case $p = 2$). Obviously, this algorithm is more expensive to run than the FGS method (since it requires several gradient computations), but it is also more efficient since it creates smaller perturbations.

III.1.3 Carlini and Wagner attack

Carlini and Wagner proposed another attack [**?**] that gained a lot of popularity. It is a regularization based attack that introduced some modifications to the previously mentioned optimization problem **??** as an attempt to improve its formulation.

Given an image x_0, the goal is to find an adversarial example x similar to x_0 while having a specific label t distinct from the original label. The attack is formulated as an optimization problem, seeking to minimize a combination of two objectives:

$$\min_x ||x - x_0||_2^2 + c \times f(x, t)$$

The first term enforces similarity between the original image and the corrupted one. The second term $c \times f(x, t)$ is the loss function that reflects the level of unsuccessful attacks, and t is the target class. Carlini and Wagner suggested the following loss function for targeted attacks:

$$f(x, t) = \max\{\max_{i \neq t} [Z(x)]_i - [Z(x)]_t, -\kappa\}$$

Where $Z(x) \in \mathbb{R}^K$ are the logits. This loss function can be adapted for untargeted attacks, where the objective is to find the closest perturbation that changes the original label:

$$f(x, t) = \max\{[Z(x)]_{t_0} - \max_{i \neq t} [Z(x)]_i, -\kappa\}$$

The optimization problem includes the constraint $x \in [0, 1]^p$, ensuring that the perturbed image lies within a valid pixel range. To address this constraint, Carlini and Wagner introduced a change of variable from x to $\frac{1}{2}(1 + \tanh(w))$. This ensures that the perturbed values are always within the $(0, 1)$ range. The Carlini and Wagner attack has demonstrated its effectiveness in generating adversarial examples that are highly transferable across different models and architectures. The attack is notable for its ability to find minimal perturbations while maintaining high success rates.

III.2. Black box attacks

Black box attacks involve scenarios where attackers only have access to a model's inputs and can query output labels or confidence scores. To assess how well a model withstands black-box attacks, we used a method called Zeroth Order Optimization (ZOO), proposed by Chen et al. [**?**]. ZOO estimates the gradients of the targeted model and uses efficient optimization techniques for attacking black-box models. This approach is inspired by the formulation of the Carlini and Wagner white box attack [**?**]. In [**?**], the authors proposed a hinge loss based on the output F of the model :

$$f(x, t) = \max\{log[F(x)]_{t_0} - \max_{i \neq t} log[F(x)]_i, -\kappa\}$$

The gradients are then estimated simply using finite differences:

$$\frac{\partial f}{\partial x_i} = \frac{f(x + \epsilon e_i) - f(x - \epsilon e_i)}{2\epsilon}$$

As untargeted attacks are easier to evaluate the general robustness of the model, we chose this implementation for our experiments. The original article proposed many optimisation algorithms, but they observed better results when using coordinate wise ADAM optimizer, thus, we implemented this method. As for the box constraints, we simply clipped the coordinates to be in the box after each iteration.

III.3. Attacks detection

In order to detect adversarial examples, we decided to make use of the logits of the model. More precisely, we will use the fact that the logits of the models are often quite different for adversarial examples. In order to do so, we will use the logits-based detection method introduced in [**?**]. The main idea of this method is that the logits values $(\log p(\boldsymbol{x}, \boldsymbol{y}_c))_{c \in C}$ of a generative classifier correspond to the log probability of generating the sample $\boldsymbol{x}$ for each label $\boldsymbol{y}_c$. Knowing this, we aim to reject samples that correspond to underconfident predictions.

The main idea behind logit-based attacks detection is to reject inputs using the joint density $p(\boldsymbol{x}, \boldsymbol{y})$ of the generative classifier, *i.e.* to reject a sample $\boldsymbol{x}$ if $p(\boldsymbol{x}, F(\boldsymbol{x}))$ (where $\boldsymbol{y} = F(\boldsymbol{x})$ is the label predicted by the model for the sample $\boldsymbol{x}$) is too low. As usual for probability tresholds, we use the logarithmic scale, and we reject a sample $\boldsymbol{x}$ if $\log p(\boldsymbol{x}, F(\boldsymbol{x})) < \delta_{\boldsymbol{y}}$. We compute the tresholds $(\delta_{\boldsymbol{y}_c})_{c \in C}$ using the well-classified samples of the training set with the following formula: $\delta_{\boldsymbol{y}_c} = \mu_c + \alpha \sigma_c$, where μ_c and σ_c are the mean and standard deviation of the log-probability of the samples of class $\boldsymbol{y}_c$ in the test set, and α is a hyperparameter which controls the tresholds scale.

In order to find the value of the hyperparameter α, we used the following procedure inspired from [**?**]: for different values of α, we computed the detection and false positives rates. We then chose the value of α which corresponded to the highest detection rate for a false positive rate of 5%.

IV. Experimental setup

In this section, we will discuss the different experiments we ran in order to compare the robustness of the discriminative and generative classifiers to adversarial attacks. More precisely, we will first present the performances of our models on the multiclass classification task, then we will compare the different attacks we used to create adversarial examples, and finally we will the results of the logit-based detection method, which has been specially designed to detect adversarial examples with generative classifiers.

V. Results

V.1. Classification performances

In terms of classification performances, we obtained the results displayed in **??** on the test and validation sets. Given the multiclass and balanced context of the task, we only focused on the accuracy. As we can see, the discriminative classifier is more efficient than the generative classifier, which is not surprising since the discriminative classifier is specifically designed for classification purposes, while the generative classifier is not. However, the generative classifier is still able to achieve a decent accuracy, which is a good sign regarding its robustness to adversarial attacks.

V.2. Runtimes of adversarial attacks

In terms of runtimes, we obtained the results displayed in **??**. As expected, we observe that the FGS algorithm is the fastest method, which is not surprising since it only requires one gradient computation. On the other hand, the ZOO method is the slowest method, which is also not a surprise. The sweet spot in terms of efficiency seems to be the DeepFool algorithm and its variant, which both take a reasonable amount of time to compute perturbations. For this reason, we will focus on comparing the Fast Gradient sign and the DeepFool algorithms in the rest of this benchmark.

V.3. Robustness to perturbations

Before fully restricting our attention to the Fast Gradient Sign and DeepFool algorithms, we first compared the effect of each attack strategy on both models (DFZ and GFZ) on a few samples of the test set. The results are displayed in **??** and **??**. In this example, we can truely see the difference between the two models; the generative classifier is more robust to adversarial attacks than the discriminative one, in the sense that it requires larger perturbations to change its predictions. We also understand the different scales of the perturbations introduced by the FGS, DeepFool, Truncated-DeepFool, Wagner-Carlini and ZOO methods.

Then, we computed the attack success rate as well as the robustness of our two models (DFZ and GFZ) on the entire test set using the two different attack strategies mentionned above. The results are displayed in **??**

and in **??**. As we can see, the generative classifier is more robust to adversarial attacks than the discriminative one, which is what we expected to observe. Moreover, we also see that the DeepFool algorithm is more efficient than the Fast Gradient Sign algorithm, in that it creates smaller perturbations.

V.4. Attacks detection

Finally, we compared the detection rates of the generative classifier and the detection rates of the discriminative classifier. To obtain these results, we fixed a false-positive rate of 5% and selected the value of α that ensured this false-positive rate. The results are displayed in **??**. As we can see, most of the FGS attacks are detected by both classifiers, which is a good sign. However, we still observe a performance gap between GFZ and DFZ, which confirms that the generative classifier is more robust to adversarial attacks than the discriminative one. Regarding the DeepFool attacks, those are much harder to detect, but we still observe a performance gap between GFZ and DFZ, which confirms our hypothesis.

By selecting a wider range for the values of α, we obtainted the ROC curves of each attack on each model, displayed in **??**, **??**, **??** and **??**. Those curves offers a more detailed view of the detection rates of each model, and still confirm our claims. In particular, we see that the area under the curve is larger for the generative classifier than for the discriminative one, for both attack strategies.

VI. Conclusion

In this report, we studied the robustness of generative classifiers to adversarial attacks, compared to discriminative classifiers. More precisely, we implemented two different classifiers (one discriminative and one generative), trained them to classify images of the FashionMNIST dataset, and then compared their robustness to several adversarial attacks. In particular, we studied the Fast Gradient Sign, DeepFool, Wagner-Carlini and ZOO methods. We also proposed a slightly-modified version of the DeepFool algorithm, which we called the *Truncated-DeepFool*, and which was able to create smaller perturbations than the original DeepFool algorithm.

After comparing the different attacks performances, we computed the robustness of our models to adversarial attacks, and we observed that the robustness of generative classifiers was in fact double; they were more robust to adversarial attacks (in the sense that they required larger attacks to be fooled, and that also translates to larger runtimes for a given algorithm), and they were also able to detect adversarial examples more efficiently.

To conclude, we can say that our experiments lead us to believe that generative classifiers are more robust to adversarial attacks than discriminative classifiers, which is a sign of improvement regarding their use in safety-critical applications. However, we also observed that the generative classifiers were less efficient in terms of classification performances, which is a drawback. With our experiments, we showed the existence of a treshold in terms of robustness and classification performances, at least in the field of Computer Vision. Discriminative classifiers seem to offer better classification performances, but it comes at the cost of a lower robustness to adversarial attacks. On the other hand, generative classifiers are more robust to adversarial attacks, but they have usually have lower classification performances.

Finally, all of our code is available on GitHub, and aims to be easily reusable in an open-source context.

VII. Appendix

References

[1] Carlini, N., and Wagner, D. Towards evaluating the robustness of neural networks. 3, 4

[2] Chen, P.-Y., Zhang, H., Sharma, Y., Yi, J., and Hsieh, C.-J. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security* (2017), CCS '17, ACM. 4

[3] J. Goodfellow, I., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. 1, 3

[4] Kingma, D. P., and Welling, M. Auto-encoding variational bayes. 2

[5] Li, Y., Bradshaw, J., and Sharma, Y. Are generative classifiers more robust to adversarial attacks? 1, 2, 3, 5

[6] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. Deepfool: a simple and accurate method to fool deep neural networks. 3

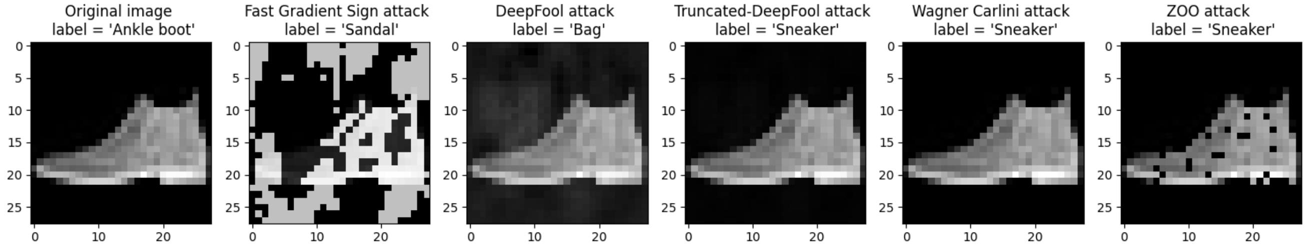

Figure 1. Comparison of different attacks on DFZ classifier

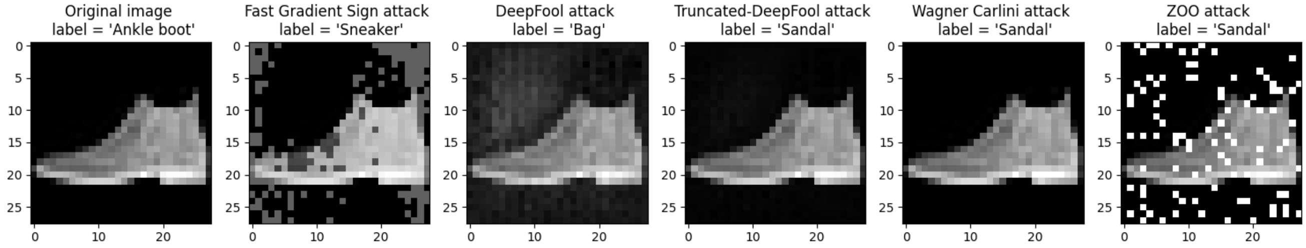

Figure 2. Comparison of different attacks on GFZ classifier

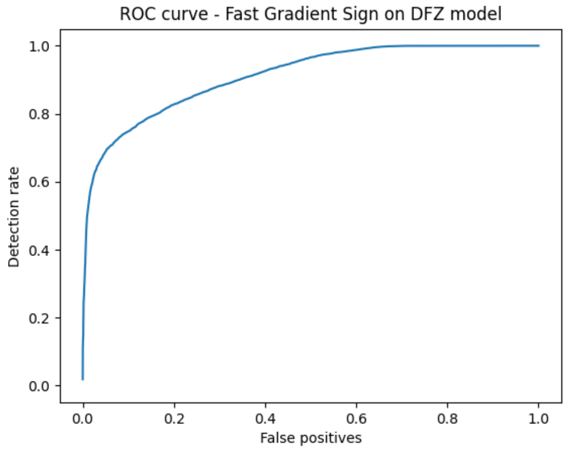

Figure 3. ROC curve for Fast Gradient Sign attack on DFZ

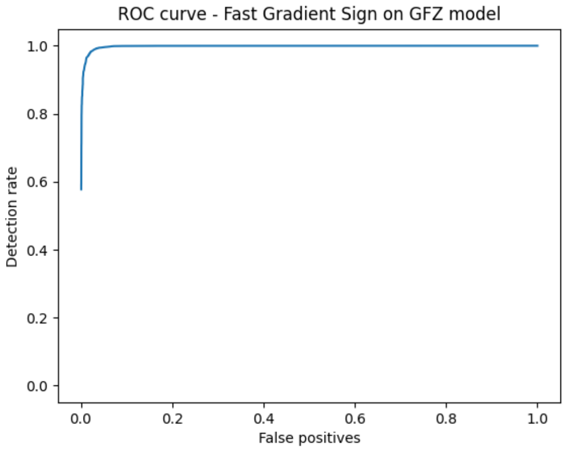

Figure 4. ROC curve for Fast Gradient Sign attack on GFZ

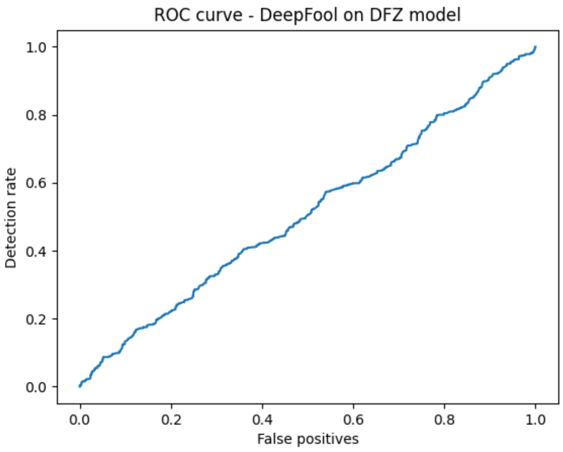

Figure 5. ROC curve for DeepFool attack on DFZ

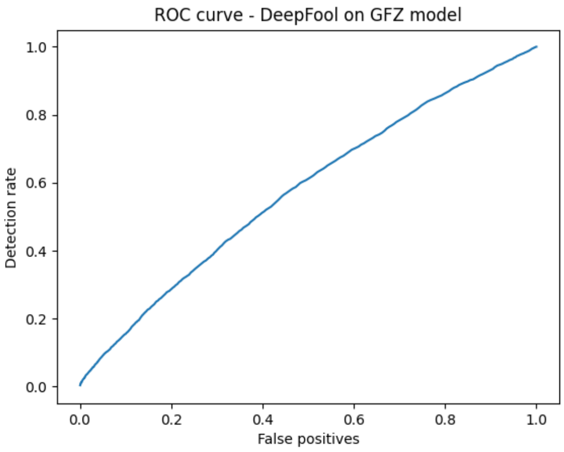

Figure 6. ROC curve for DeepFool attack on GFZ

Algorithm 1: DeepFool algorithm

```
Input: $f_\theta, \boldsymbol{x}, \boldsymbol{y}, p > 1$
Output: $\boldsymbol{\eta}$
$q \leftarrow \frac{p}{p-1}$
$i \leftarrow 0$
$\boldsymbol{\eta} \leftarrow 0$
$\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$
while $\underset{k}{\mathit{argmax}}\ f_k(x) = y$ do
    $i \leftarrow i + 1$
    for $k \neq y$ do
        $\boldsymbol{w}_k \leftarrow \boldsymbol{\nabla_x} f_k(\boldsymbol{x}_i) - \boldsymbol{\nabla_x} f_y(\boldsymbol{x}_i)$
        $\delta_k \leftarrow f_k(\boldsymbol{x}_i) - f_y(\boldsymbol{x}_i)$
    end
    $l \leftarrow \underset{k \neq y}{\text{argmin}} \frac{|\delta_k|}{||\boldsymbol{w}_k||_q}$
    $\boldsymbol{\eta}_i \leftarrow \frac{|\delta_l|}{||\boldsymbol{w}_l||_q^q} |\boldsymbol{w}_l|^{q-1} \times \text{sign}(\boldsymbol{w}_l)$
    $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{\eta}_i$
    $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \boldsymbol{\eta}_i$
end
```

Classifier	Accuracy (test set)	Accuracy (validation set)
DFZ	0.8891	0.8850
GFZ	0.8081	0.7960

Table 1. Comparison of classification performances (on test and validation set) of our classifiers

Classifier/Attack	Fast Gradient Sign	DeepFool	Truncated-DeepFool	Wagner Carlini	ZOO
DFZ	0.3s	1.7s	3.2s	26.3s	22.0s
GFZ	1.5s	4.2s	3.7s	24.2s	23.5s

Table 2. Comparison of runtimes for different attacks on our classifiers

Classifier/Attack	DeepFool	Fast Gradient Sign
DFZ	0.4927	0.2140
GFZ	0.5560	0.3734

Table 3. Comparison of success rates for different attacks on our classifiers

Classifier/Attack	DeepFool	Fast Gradient Sign
DFZ	0.1159	0.4615
GFZ	0.1477	0.6138

Table 4. Comparison of robustness for different attacks on our classifiers

Classifier/Attack	DeepFool	Fast Gradient Sign
DFZ	0.0875	0.6843
GFZ	0.0949	0.9947

Table 5. Comparison of detection rates for different attacks on our classifiers