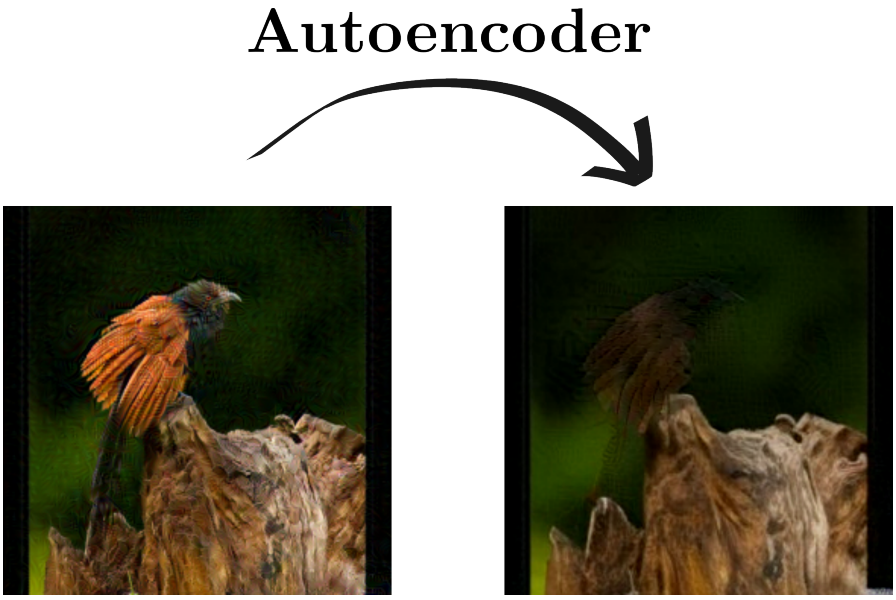
Semester Project – Autumn 2023

Adversarial Attacks on Autoencoders

With 19 figures



Author
Diego Dorn

EPFL

Supervisor
Clément Hongler

Introduction

Machine learning models take more and more importance in our society. From recommenders systems of social media and search engines, to automated screening of job and loan applications, from power grid management to general purpose generative models, we seem to be going towards a world where more and more decisions are automated.

However, if we want the mass depolyment of those systems to not be accompanied by more than normal accidents, we still need to understand better how those systems work, and especially how they fail.[1]

In this project, I focused on a specific kind of failure: adversarial attacks. Adversarial attacks are likely the most studied failure mode of machine learning algorithms. They are small perturbations of the input of a machine learning model that fools the model into producing a very different output.

The initial goal of the project was to find transferable attacks on the visual components of multimodal large language models, however, we switched to attacking image autoencoders. Autoencoders are a class of neural networks that are trained to compress and reconstruct their input, and can be used as a defense mechanism against some adversarial attacks.

The project is divided in three parts. In the first part, we show different adversarial attacks on classifiers, before applying them to autoencoders in the second part, where we show that autoencoders can completely transform their input into an other image. Finally, in the third part, we show that present a novel finding on the norm of activations of autoencoders when under an attack to transform their input.

[1] This is, however, obviously not sufficient and other ingredients such as a good democratic control loop, legal frameworks and avoiding concentration of power are also necessary.

Contents

Conventions

Throughout this document we adopt a set of conventions and notations.

- $\mathcal{X} \subset \mathbb{R}^n$ is the set in which datapoints live.
- $\mathcal{Y}$ is a space of labels, which will most of the time be categorical, i.e. $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{\text{cat}, \text{dog}, \text{boat}\}$.
- Loss functions are denoted by $\mathcal{L}$.
- We write $\|\cdot\|$ for the euclidean norm on $\mathbb{R}^n$, and $\|\cdot\|_p$ for the p-norm on $\mathbb{R}^n$. As a reminder, for $x \in \mathbb{R}^n$, $\|x\|_\infty = \max_{i=1}^n |x_i|$.

1 Attacking classifiers

The common knowledge is that neural networks are vulnerable to adversarial attacks and adversarial attacks are easy to find [Szegedy et al., 2013, Goodfellow et al., 2014, Chen et al., 2021]. The first thing I wanted to do was to verify this in practice. Can I easily attack any classifier outside of the well defined confines of a classroom or a paper for which a lot of work was put into?

1.1 Setup

I used three different models and datasets throughout this project. The code of every experiment is available at `https://github.com/ddorn/autoencoder-attacks`.

MNIST The smallest dataset I used is the MNIST dataset [LeCun et al., 1998], with a small convolutional classifier acheiving 98.8 % accuracy implemented in pytorch [Oikarinen, 2021].0

CIFAR-10 The second dataset is the CIFAR-10 dataset [Krizhevsky, 2009], with a convolutional classifier acheiving 92.8% accuracy [Germer, 2022].

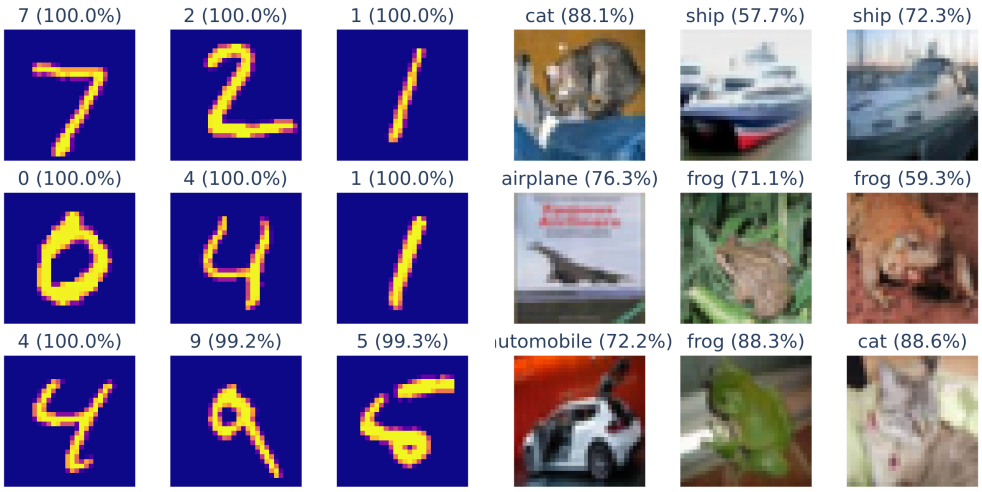


Figure 1: The first 9 test images of the MNIST dataset (left) the CIFAR-10 dataset (right). The confidence of the classifier is shown in parenthesis.

ImageNet The third dataset is ImageNet [Deng et al., 2009], with a ResNet-50 classifier achieving 77 % top-1 accuracy [He et al., 2015].

1.2 Fast gradient sign method

The simplest attack is the fast gradient sign method (FGSM) [Goodfellow et al., 2014]. FGSM works by linearizing the loss function around the input, and computing the exact

Figure 2: The first 9 test images of the ImageNet dataset. The confidence of the classifier is shown in parenthesis.

maximum of the linearized loss function.

This attack requires only one forward and one backward pass through the network to find a small perturbation of an image that can (potentially) fool the classifier.

What is small? We usually constrain the norm of the perturbation to a small value ε. The norm used can be the l_∞ norm, for $\varepsilon = 10/255$ or $\varepsilon = 4/255$ are common values, or the l_0, l_1 or l_2 norm. Clearly the four norms produce different constrains, and should be chosen depending on the context:

- l_∞ is a natural choice, and corresponds to changing each pixel value by at most ε.
- Using the l_0 norm means to change at most ε pixels. This can be one-pixel attacks [Su et al., 2017], attacks that change a small number of pixels, or patch attacks [Brown et al., 2017].
- l_1 and l_2 constraints can be used when it is fine if some pixels are completely changed, but not too many are changed a lot.

The fast gradient sign method is an untargeted attack, meaning that the goal is to find a perturbation that changes the prediction of the classifier, but not to force the classifier to predict a specific label. It is defined as follows.

Definition 1.1. Let f be a classifier and $x \in \mathcal{X}$ an input with label $y \in \mathcal{Y}$. The **fast gradient sign method** is the attack that computes

$$x_{\text{FGSM}} = x + \varepsilon \cdot \text{Sign}(\nabla_x \mathcal{L}(f(x), y))$$

where $\mathcal{L}$ is the loss function used to train f, and ε is the desired l_∞ norm of the

perturbation.

We show an example of the attack on the first test image for each of the datasets, with the top 5 categories shown in Figure 3, Figure 4 and Figure 5.

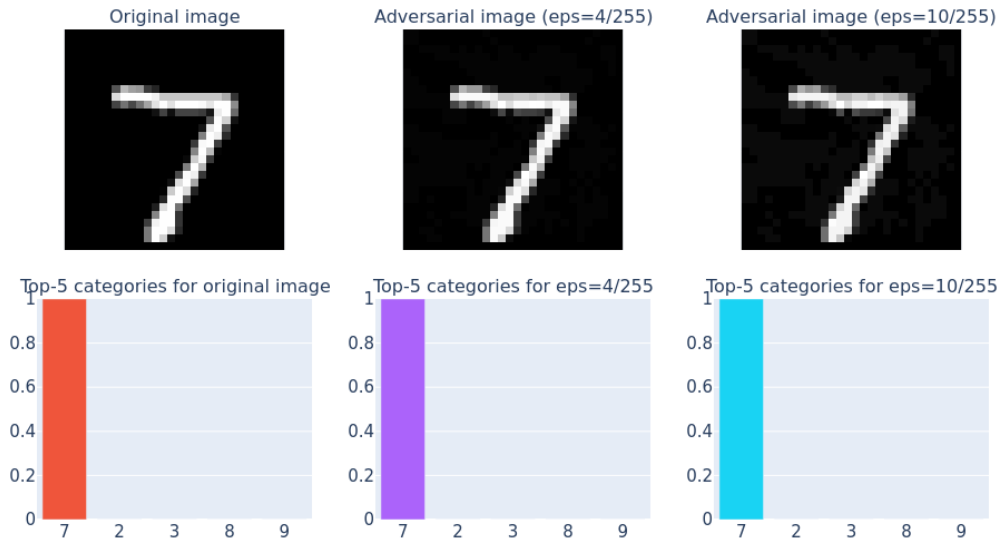


Figure 3: Example of a (non-sucessful) FGSM attack on MNIST.

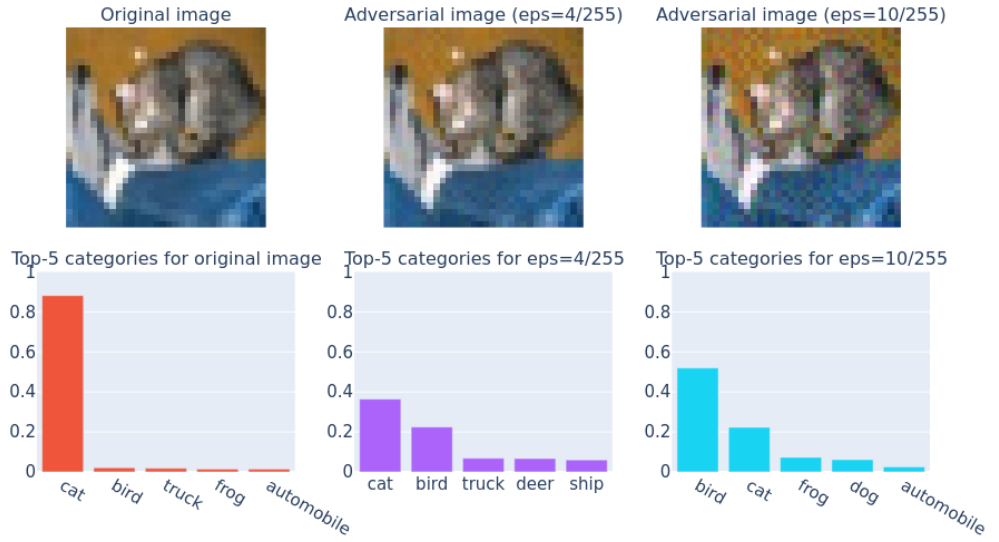


Figure 4: Example of a FGSM attack on CIFAR10.

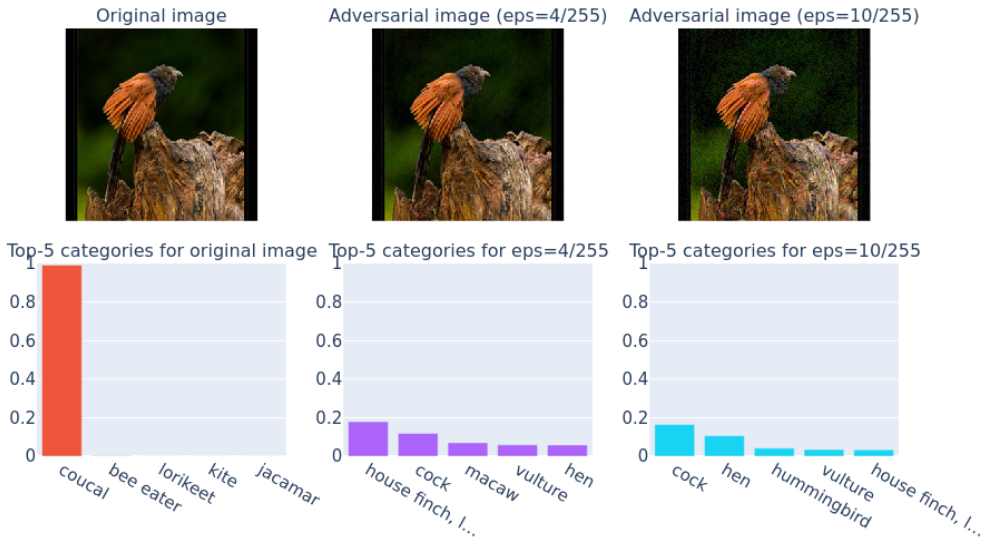


Figure 5: Example of a FGSM attack on ImageNet.

The attack seems to not work for the image from MNIST, but works well on the one from CIFAR10 and ImageNet. Is it the case for all images?

So I take the three classifiers, a thousand test images from each dataset, compute the gradient of the loss with respect to the input image and add $\varepsilon = 10/255$ times the sign of the gradient to the image. This gives a thousand adversarial images for each task and we can see the accuracy of the clean versus the adversarial images in Figure 6.

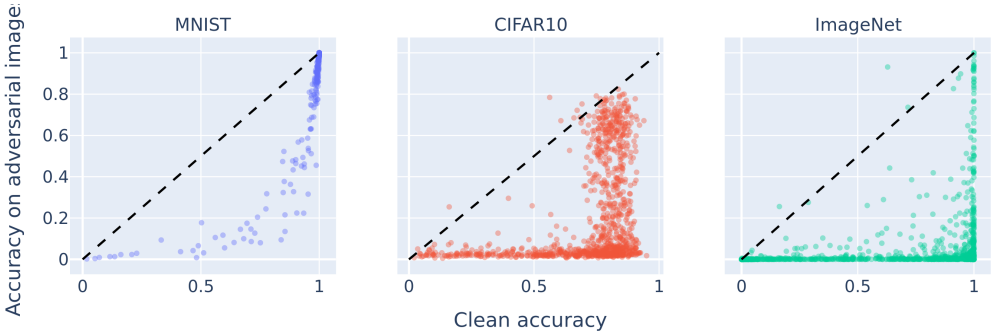


Figure 6: Confidence of the three classifiers in the correct label of a 1000 test images before and after an FGSM attack with $\varepsilon = 10/255$. The black line corresponds to no change in confidence.

We can see that, on agregate, it works very well on ImageNet and CIFAR10, and less well on MNIST. This can likely be attributed to the fact that images from CIFAR10 and ImageNet are larger than MNIST and with three color channels instead of one. The larger size allows for more ways to find a path towards the decision boundary. The sizes can be found in Table 1.

	MNIST	CIFAR10	ImageNet
Clean accuracy	98.9 %	92.8 %	77.2 %
Accuracy $\varepsilon = 4/255$	98.8 %	87.2 %	50.2 %
Accuracy $\varepsilon = 10/255$	98.8 %	76.8 %	28.2 %
Image size	$1 \times 28 \times 28$	$3 \times 32 \times 32$	$3 \times 256 \times 256$[2]

Table 1: Top-1 accuracy before and after FGSM attack on a thousand images.

1.3 Variants of the fast gradient sign method

The fast gradient sign method is a simple and efficient attack, but can be made more powerful in multiple ways.

Targeted attacks The fast gradient sign method is an untargeted attack, meaning that the goal is to find a perturbation that changes the prediction of the classifier, but not to force the classifier to predict a specific label. However, the attack can be easily modified to be targeted, as is the case with most attacks.

Definition 1.2. Let $x \in \mathcal{X}$ be any input and $y \in \mathcal{Y}$ be any label (but likely not the label of x). Then, the **targeted fast gradient sign method** is defined as follows.

$$x_{\text{FGSM}} = x - \varepsilon \cdot \text{Sign}(\nabla_x \mathcal{L}(f(x), y))$$

Where $\mathcal{L}$ is the loss function used to train f, and ε is the desired l_∞ norm of the perturbation.

Notice the minus sign instead of the plus sign in the untargeted attack: we want the loss, evaluated with the target (wrong) label to decrease. Before, we wanted the loss, evaluated with the correct label to increase.

Iterative fast gradient sign method Instead adding ε times the sign of the gradient, we can iterate N times and add ε/N times the sign of the gradient at each step. This is called the iterative fast gradient sign method. In our experiments, this managed to find adversarial images with smaller perturbations, and for harder attacks targets it is necessary to iterate on the attack multiple times.

Transferable attacks The fast gradient sign method is a white box attack, meaning that it requires access to the weights of the model to perform the attack. In practice,

[2]ImageNet has images of different sizes, but were resized and croped to 256×256. This is different from the original paper [He et al., 2015] which used 224×224 images, but the classifier still acheives good accuracy. This design choice comes from the fact that the autoencoder used in the next section cannot take images smaller than 256×256.

a lot of models are proprietary, or at least not easily accessible, but this does not mean that they are safe from adversarial attacks.

Indeed, it is possible to create attacks that are transferable to other models, meaning that the adversarial images produced by the attack on one model are also adversarial images for another model. This can be used for instance to attack a large model, with only access to a smaller model. Attacks on vision models tend to be transferable by default, and can even transfer, in a weak sense, to humans [Elsayed et al., 2018].

However, stronger transferable attacks can be found by attacking multiple models at once, that is, by using the mean of the loss of multiple models instead of the loss of one model. While this was the topic I initially started to work on, it will not be the focus of the rest of this report.

2 Attacking autoencoders

Autoencoders have been suggested as a defense mechanism against adversarial attacks.

2.1 Autoencoders

Autoencoders where introduced by [Hinton and Salakhutdinov, 2006] as a way to learn a low dimensional representation of data. They are a class of neural networks that are trained to reconstruct their input, and are composed of two deep neural network, an encoder and a decoder with a bottleneck in between as shown in Figure 7.

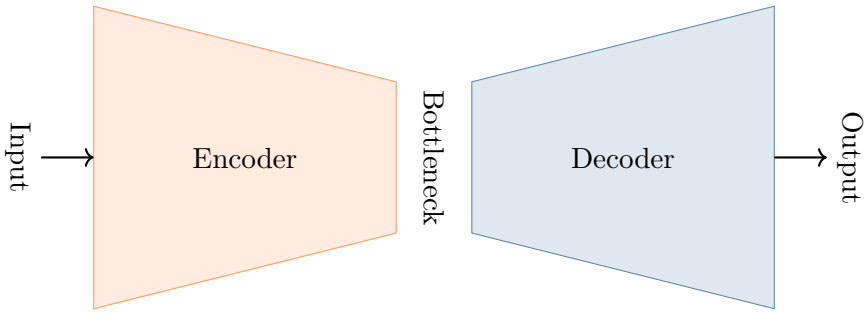


Figure 7: Overview of the autoencoder architecture

The **encoder** takes an high dimensional data point as input, processes it through a series of layers, usually fully connected layers or a residual network [He et al., 2015] in the case of visual data, and outputs a low dimensional representation of the input.

The **decoder** takes the low dimensional output of the encoder and processes it similarly through a series of layers, and outputs a high dimensional reconstruction of the input.

The **bottleneck** is not a layer, but rather the middle of the autoencoder, where the activations are the lowest number of dimensions.

Training Autoencoders are trained to reconstruct their input, that is, they learn the identity function. The loss is a natural metric on the data space, such as the mean squared error for real valued data.

Definition 2.1. The **reconstruction loss** for an autoencoder f on an input $x \in \mathcal{X}$ is

$$\mathcal{L}_{\text{recon.}}(x) = \|x - f(x)\|^2$$

To prevent overfitting and to perform a directly useful task, an autoencoder can be train to reconstruct a noisy or corrupted version of the input.

Definition 2.2. The **denoising loss** for an autoencoder f on an input $x \in \mathcal{X}$ is

$$\mathcal{L}_{\text{denoising}}(x) = \|x - f(x + \varepsilon)\|^2$$

where ε is a random vector of the same dimension as x, of white noise whose variance is an hyperparmeter of the training setup.

Note that the denoising loss is stochastic, as it depends on the random vector ε. In practice, we use the compute the loss on one sample of ε per input.

Variational AutoEncoders (VAE) A specific kind of autoencoders, and the one that will be used in all of the following experiments, are variational autoencoders, intruduced by [Kingma and Welling, 2013] Technically, they are not very different from regular autoencoders, but they come from a different background than data compression. Indeed, the hope is that VAEs model the process from which the data was generated. Oftentimes, we expect a datapoint (for instance the image of a leaf) to be determined only by *a few* variables (for instance, the species of the tree, its age, the season, the angle at which the picture was taken etc.). We will call P, the vector of those few variables that generate the datapoint.

The encoder of a VAE tries to find some representation of P and outputs two vectors, μ and σ instead of one, which are interpreted as the mean and the variance of the prior on P, which is assumed to be a normal distribution.

The variable $P \sim \mathcal{N}(\mu, \sigma)$ are then sampled and fed to the decoder that tried to reconstruct what should be generated from the underlying variables. The decoder thus tries to model the process that generated the dataset and outputs a distribution Q over the data space.

Definition 2.3. Let f be a VAE and $x \in \mathcal{X}$ a datapoint. It loss on x is composed of two terms, the **likelyhood loss** and the **regularisation loss**.

$$\mathcal{L}_{\text{vae}}(x) = \underbrace{\mathbb{P}(x \mid f(x))}_{\text{likelyhood loss}} + \underbrace{D_{\text{KL}}(P \,||\, \mathcal{N}(0,1))}_{\text{regularisation loss}}$$

Remark. The KL divergence is a measure of how different two distributions are. In this case, it is used to measure how far the prior on P is from the standard normal distribution.

$$D_{\text{KL}}(P \,||\, Q) = \int_{\mathcal{X}} P(x) \log \frac{P(x)}{Q(x)} \, \mathrm{d}x$$

Here we can use the fact that both P and Q are n-dimensional normal distributions to compute the KL divergence in closed form.

$$D_{\text{KL}}(\mathcal{N}(\mu, \sigma) \,||\, \mathcal{N}(0,1)) = \frac{-1}{2n} \sum_{i=1}^{n} \left(1 + \log \sigma_i^2 - \mu_i^2 - \sigma_i^2\right)$$

Here, the likelyhood loss is conceptually equivalent to the reconstruction loss of a regular autoencoder, and the regularisation loss is used to have a latent representation that is close to a normal distribution. It makes it easier to sample latent representations and corresponds to the assumtion that the underlying variables are normally distributed.

2.2 Setup

I used two different variational autoencoders, one for each dataset and classifier.

MNIST On the MNIST dataset, I trained a small convolutional autoencoder with a bottleneck of size 4, which has reconstructs the images with an average per-pixel error of 7.3 %.

ImageNet On the ImageNet dataset, I used `stabilityai/sd-vae-ft-mse`, a pre-trained autoencoder from StabilityAI [Stability AI, 2023] with 83.8M parameters. It has a bottleneck of size 2048 and reconstructs the images with an average per-pixel error of 3.3 %.

5Note: In the following, every use the term *autoencoder* refers to a variational autoencoder.

2.3 Autoencoders as a defense mechanism

Autoencoders can be used to prevent adversarial attacks against classifier by prepocessing images through the autoencoder. The setup is shown in Figure 8.

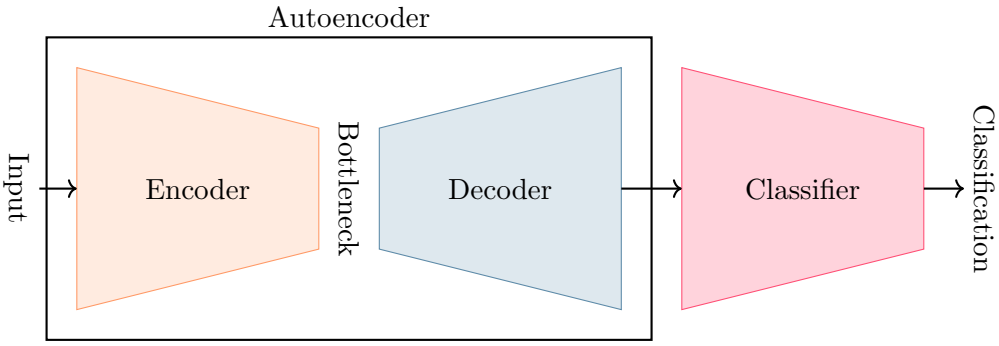


Figure 8: Autoencoder used as a defense against adversarial attacks

The hope is that an adversarial perturbation of the input will not pass through the bottleneck of the autoencoder, and thus will not be able to fool the classifier. Indeed, the bottleneck is small, and therefore information constrained, so we expect to the autoencoder to not faithfully reconstruct patterns that it has never seen during training. In particular, we expect the latent respresentation of an adversarial input to be the same

as the representation of the original input, and thus the autoencoder should reconstruct the original input when fed the adversarial one.

Passing the adversarial images from Table 1, through the autoencoder, we can see that the autoencoder lessens the visual noise on the images as seen in Figure 9. This happens both because the images is compressed then decompressed, and because the autoencoders were trained to reconstruct noisy images.

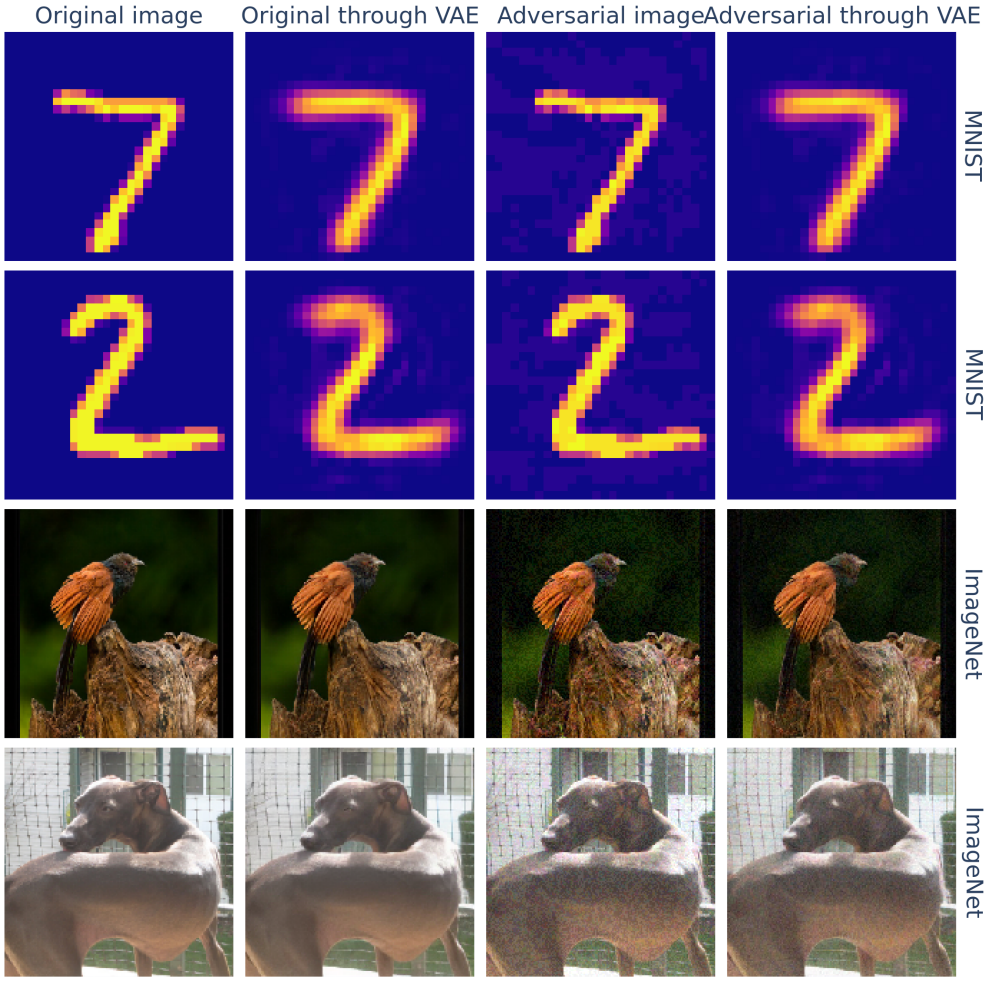


Figure 9: Examples of the autoencoder defense. **Left**: first 2 images of each dataset. **2nd column**: original images passed through the autoencoder. **3rd column**: adversarial images, produced by FSGM with $\varepsilon = 10/255$. **Right**: adversarial images passed through the autoencoder.

Passing the image through the VAE also recovers most of the accuracy of the classifiers, as seen in Table 2. However, the base accuracy, on clean images is reduced by a few

percent (about 10 % for MNIST and 3 % for ImageNet). This is an instance of the tradeoff between accuracy and robustness.

		Autoencoder	
		Without	In front
MNIST	Clean accuracy	98.9 %	89.1 %
	Accuracy $\varepsilon = 4/255$	98.8 %	88.9 %
	Accuracy $\varepsilon = 10/255$	98.8 %	89.0 %
ImageNet	Clean accuracy	77.2 %	74.7 %
	Accuracy $\varepsilon = 4/255$	50.2 %	72.7 %
	Accuracy $\varepsilon = 10/255$	28.2 %	69.7 %

Table 2: Accuracy of the classifier on the original and autoencoded images.

However, this is a very simple baseline defense, which relies on the attacker not knowing about the autoencoder, or not having access to it, and there are attacks that work even in this case.

More sophisticated defenses can be designed, and many have been proposed. One can randomly pick an autoencoder from a pool of autoencoders, add noise to the input, take the median of output of many autoencoders, or use other compression metrics such as JPEG or reducing the precision of the image.

In any case, even if security through obscurity might work, it goes against Kerckhoffs's principle [Kerckhoffs, 1883], which states that a system should be secure even if the attacker knows everything about it, except the secret key. There is not a nice concept of a secret key here, but considering the autoencoder as a secret key is not a good idea, as an autoencoder is too big, with too much information about the world to be considered secret.

2.4 Attacking through autoencoders

In a setting where an attacker knows the autoencoder placed as a defense in front of the classifier, one can wonder what type of attacks exists. It seemed to me that it would be hard to generate an adversarial perturbation as the output of an autoencoder, since this model has never been trained to produce this kind of structure. On the other hand, I tought that it would be easier to make the autoencoder produce an entierly different image. My advisor expecting it to be hard to make an autoencoder produce a different image, I set out to prove it wrong.

There are two natural attacks to try:

1. Attacking the composition of the autoencoder and classifier, to produce a different label, hoping that it "fools" the classifier by feeding it an image of something else than the original input. In this case, the loss function is the loss of the classifier. For an image x and a target label y, we have

$$\mathcal{L}_1(x, y) = \mathcal{L}_{\text{cross.}}(\text{Classifier}(\text{Autoencoder}(x)), y)$$

The attack can also be untargeted, in which case the target label y is chosen to be the label of the original image x and we search for the maximum of the cross entropy loss instead of the minimum.

2. Attacking the autoencoder directly to produce a different image. In this case, the loss function is the reconstruction loss of the autoencoder, hover For two images x and x', we have

$$\mathcal{L}_2(x, x') = \mathcal{L}_{\text{rec.}}(\text{Autoencoder}(x), x')$$

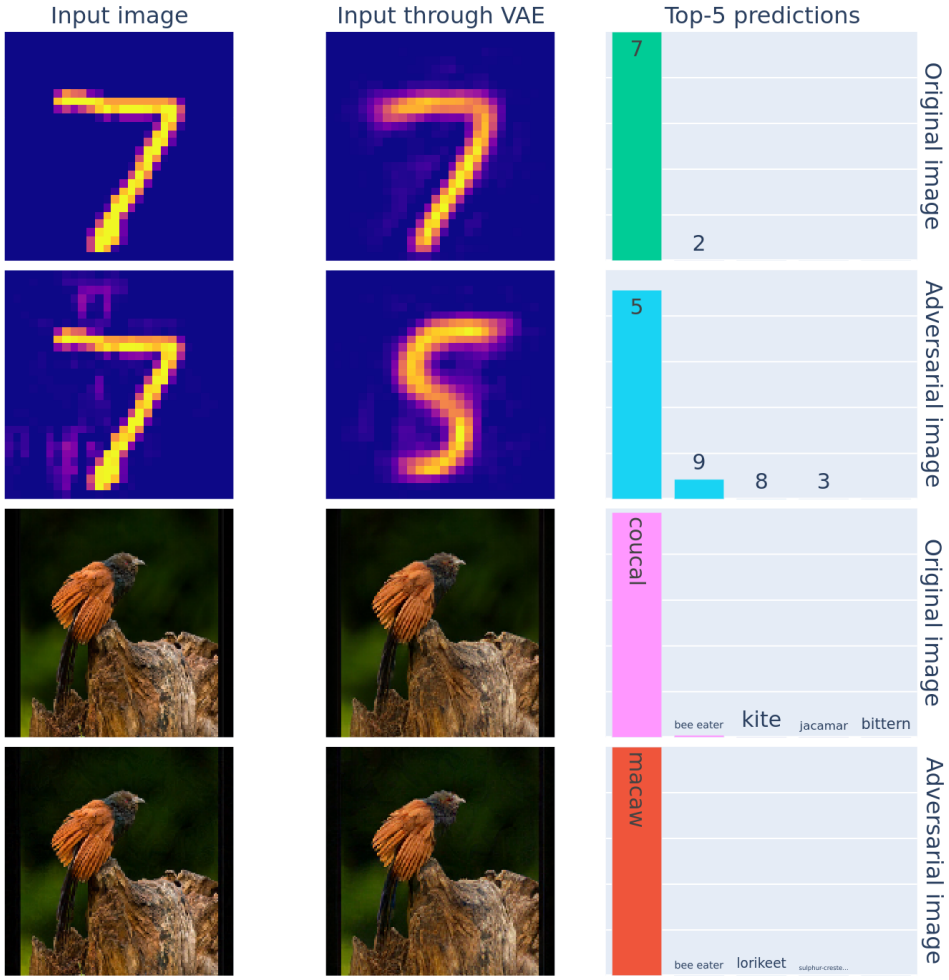


Figure 10: Examples of attacks through the autoencoder.

We show in Figure 10 two examples of attacks agains the composition of the autoencoder and classifier. The most interesting thing to note is that the visuals of the adversarial images are very different on the two datasets:

- For MNIST, the autoencoder produces an image that looks like a 5 or a 9, which is very different from the initial image of a 7.
- For ImageNet, the autoencoder produces an image that is undistinguishable from the original image, but the classifier predicts a different label.

Therefore, both kinds of attacks mechanisms are possible. It is possible to transform an image through an autoencoder into something that a human would recognise as different (or at least for this specific small VAE on MNIST), but it is also possible to make an autoencoder produce an adversarial image.

The attacks being different for the two datasets suggests that transforming an image is easier on small networks or images but on larger networks/images it is easier to make the autoencoder produce and adversarial perturbation. Before verifying this hypothesis, we need to check if this pattern holds for other autoencoders.

Taking 20 random images from the test set of each dataset, we can see in Figure 11 that the pattern does hold.

Figure 11: Examples of iterated FGSM attacks through the autoencoder and classifier. **Even columns:** Adversarial images. **Odd columns:** The image on the left passed through the autoencoder.

Can the MNSIT autoencoder produce an adversarial image? Probably not. The attack on MNSIT seems to always either not work or produce a different image through the VAE. One could try to make the autoencoder output and adversarial perturbation by adding a term to the reconstruction loss of the autoencoder. This would encorage the autoencoder to produce an image that is close to its input. However, I could not make this to work, even when allowing the perturbation arbirtarily large. Some options that I did not try are using different schedules for the loss components or using attacks methods not mentions in this report.

The other question left is whether the ImageNet autoencoder can produce and image that is completely different from the input.

2.5 Attacking the ImageNet autoencoder

Figure 12: Example of a targeted attack that producing a different image through the autoencoder. The output image is almost correctly classified by the ResNet50, with 41 % "strainer" and "10 % "Petri dish" (the correct one).

I set out to attack the ImageNet autoencoder so that it produces an image that is completely different from the input. This was not an easy task, but is possible with a few tricks and caveats:

- One needs to accept that the attack is quite visible on the image, as it seems to be impossible to do with $\varepsilon \leq 30/255$.
- The target image should not be too distant from the input image, and it is very

helpful to pick a target image among the closest neighbours in norm l_1 or l_2 in the dataset.

- Using a loss that is a composite of different terms was the most helpful:
  1. The reconstruction loss towards the target is of course necessary.
  2. A penalty for the l_2 norm of the perturbation helps to keep the perturbation small and prevent too much clamping to keep the perturbation in the valid range. A penalty of 0.1 times the mean squared error was used everywhere in this report. It seems to be possible to train the attacks faster by starting with no penalty and then increasing it, but it is more hazardous.
  3. An attack with a lower norm (lower ε) is easier to get by setting ε high at first, then lowering it.
- The fact that the autoencoder is a VAE, and thus stockastic did not seem to play an important role in the attack, even though it was for getting FGSM to work on the smaller tasks.

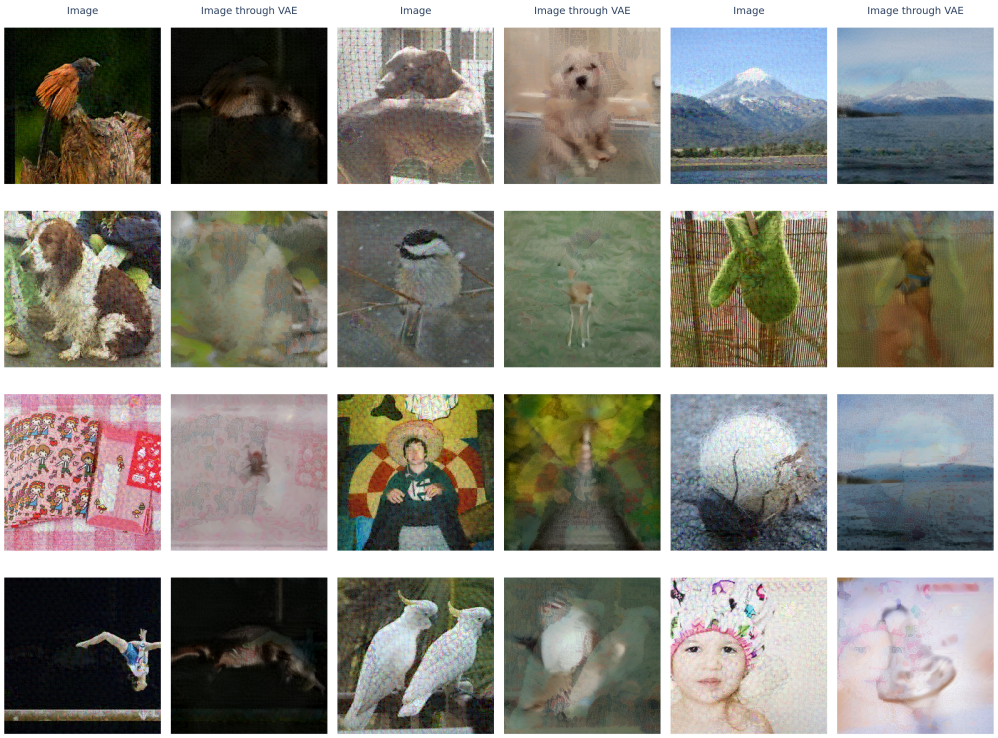


Figure 13: Examples of attacks through the autoencoder. **Even columns:** Adversarial images. **Odd columns:** The image on the left passed through the autoencoder.

Technical details The attacks seen in Figure 12 and 13 are all iterated FGSM attacks, produced with 3 steps with different hyperparmeters:

Iterations	ε	L2 Penalty	Learning rate
600	70/255	1.0	0.004
300	70/255	1.0	0.001
200	70/255	1.0	0.0004

Remark. Attacking an autoencoder to produce a different image, can be detected very easily, by checking if the reconstruction loss is high. This seems to be a good sanity check to have in place when using an autoencoder as a defense mechanism.

3 Norm detection

Seeing the attacks that produce a completely different image through the autoencoder, I was reminded of chaos theory, as we have an small perturbation of the input that produces large changes in the output. I wondered if there was a nicely defined Lyapunov exponent that applied to this system or maybe to many neural networks.

The first I checked was the evolution of the distance in activation space between the original image and the adversarial image, at each layer of the network, expecting to see where in the networks the perturbation was amplified. Do they smoothly separate over time, or is there a specific (set of) layer(s) where the perturbation is amplified?

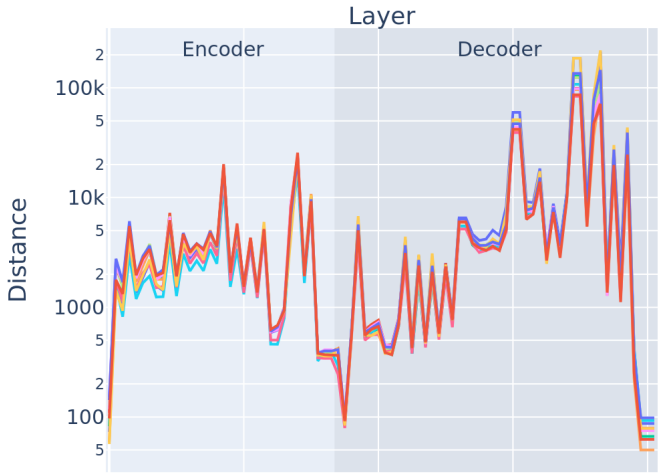


Figure 14: Distance in activation space between the original image and the adversarial image at each layer of the autoencoder, for the 12 attacks from Figure 11.

Well, if one thing is clear, it's that it's not clear on Figure 14. Indeed, the norm of the activations varies by several orders of magnitude between layers and mediates strongly the distance between the activations. It might be surprising at first that the norms go as high as 200 000, but the size of those tensors also reach 16 million elements.

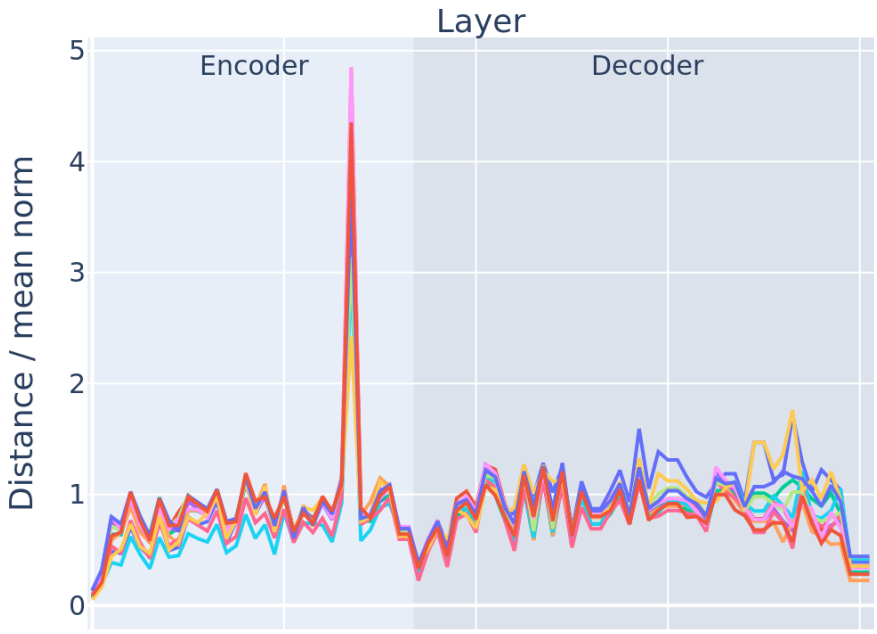


Figure 15: Distance in activation space between the original image and the adversarial image at each layer of the autoencoder, for the 12 attacks from Figure 11. The distance is normalised by the average activation norm of 100 images from the test dataset.

Surprisingly, when normalising, we have very clear spike at the end of the encoder. This spike is present on every of the 12 attacks. And corresponds to the output of the attention, before normalisation and before the last three convolutions of the encoder.

I first tried to understand why the distance between the clean and corrupt activations was so high, do they have a different norm? Or is it because the activations are in different directions?

Looking at the norms in Figure 16 reveals even more clearly, a single layer where the norm of the activations is about 4 times larger than the average norm of the dataset. There is also much more variance in the norm of activations on adversarial images than on clean images, whose blue bars can barely be seen on the plot.

This gives an straightforward way to detect this kind of adversarial attacks: pass the image through the autoencoder and check if the norm of the activations at this layer is larger than usual.

Remark. It is still more straightforward to check the reconstruction loss.

Figure 16: Layer-by-layer plot of the norm of the activations of the autoencoder, normalised by the average norm of 100 images from the dataset.

3.1 Generalisation

The last question is whether this phenomenon is specific to our setup, or is a more general phenomenon.

Different autoencoders I tried the same experiment on the MNIST autoencoder whose results are shown in Figure 17 but could not get a good autoencoder running on CIFAR10. On MNIST, there is no clear spike, and the norms of adversarial activations are even a little bit smaller than the norms of clean activations. However, this is an artifact of the choice of the attacks to plot, and other attacks did not show this pattern.

Overall, this suggests that the phenomenon is either specific to StabilityAI's autoencoder, or to larger autoencoders.

Similarly, attacking only the classifier, without the autoencoder, does not produce this pattern.

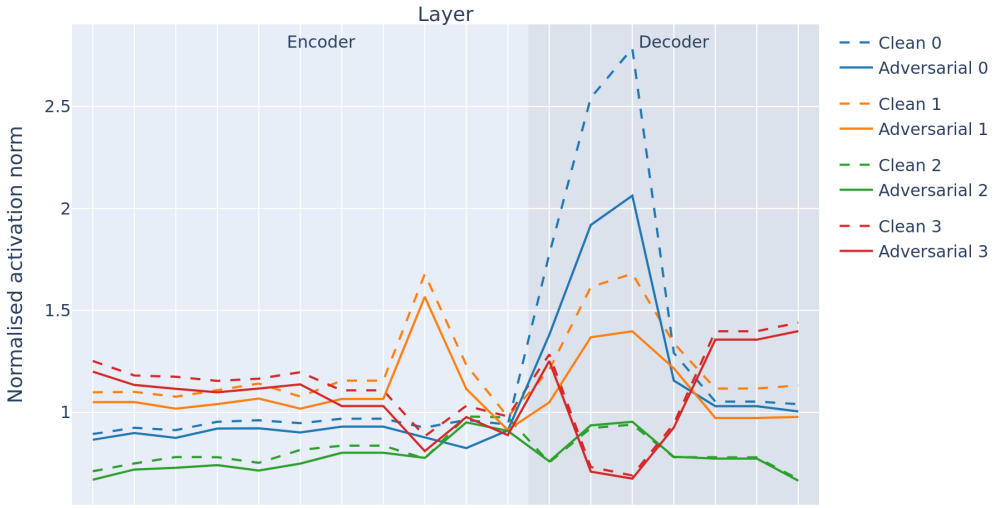


Figure 17: Layer-by-layer plot of the norm of the activations of the autoencoder, normalised by the average norm of 100 images from the dataset.

Bonus

As a bonus, we show a few images that produce interesting results when passed through the autoencoder.

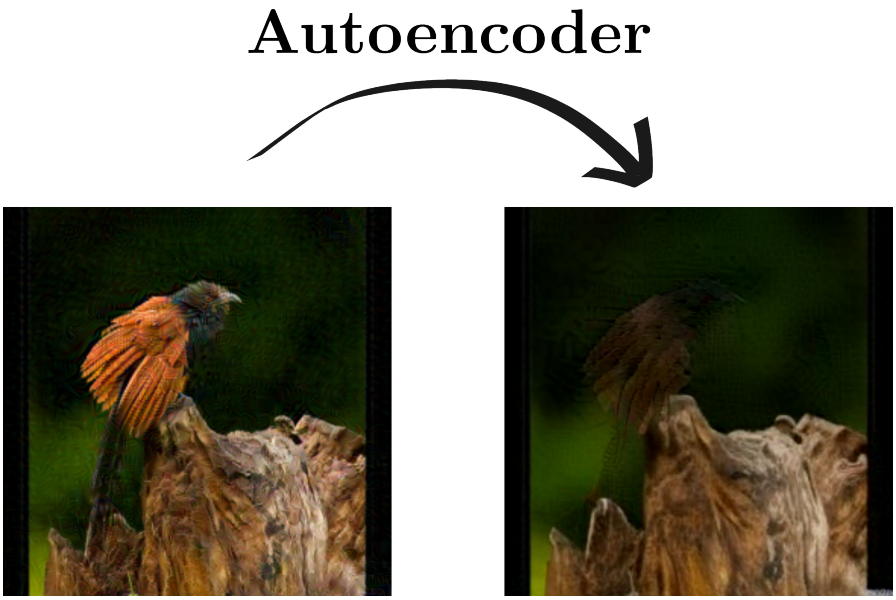


Figure 18: A bird with an invisibility cloak. It is easier to attack when only a part of the image is changed, as the two are more similar.

Note that making birds disapear is something humans have been really good at, without the help of neural networks. 60% of european land birds have disappeared between 1980 and 2016 [Rigal et al., 2023], for a total of 800 million birds.

However, making a bird appear out of thin air is something that only the best of magicians have been able to do.

Figure 19: A bird appearing out of thin air.

References

[Brown et al., 2017] Brown, T. B., Mané, D., Roy, A., Abadi, M., and Gilmer, J. (2017). Adversarial patch. *ArXiv*, abs/1712.09665.

[Chen et al., 2021] Chen, Y., Zhang, M., Li, J., and Kuang, X. (2021). A survey on adversarial attacks and defenses in image classification: A practical perspective. *arXiv preprint arXiv:2201.01809*.

[Deng et al., 2009] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255.

[Elsayed et al., 2018] Elsayed, G. F., Shankar, S., Cheung, B., Papernot, N., Kurakin, A., Goodfellow, I. J., and Sohl-Dickstein, J. N. (2018). Adversarial examples that fool both computer vision and time-limited humans. In *Neural Information Processing Systems*.

[Germer, 2022] Germer, T. (2022). Train cifar10 to 94% accuracy in a few minutes/seconds. Accessed: 2024-01-01.

[Goodfellow et al., 2014] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572.

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

[Kerckhoffs, 1883] Kerckhoffs, A. (1883). *La cryptographie militaire, ou, Des chiffres usités en temps de guerre: avec un nouveau procédé de déchiffrement applicable aux systèmes à double clef*. Librairie militaire de L. Baudoin.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *CoRR*, abs/1312.6114.

[Krizhevsky, 2009] Krizhevsky, A. (2009). Learning multiple layers of features from tiny images.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proc. IEEE*, 86:2278–2324.

[Oikarinen, 2021] Oikarinen, T. (2021). Training a nn to 99% accuracy on mnist in 0.76 seconds. Accessed: 2024-01-01.

[Rigal et al., 2023] Rigal, S., Dakos, V., Alonso, H., Auniņš, A., Benkő, Z., Brotóns, L., Chodkiewicz, T., Chylarecki, P., de Carli, E., Moral, J. C. D., Domșa, C., Escandell, V., Fontaine, B., Foppen, R., Gregory, R. D., Harris, S. J., Herrando, S., Husby, M., Ieronymidou, C., Jiguet, F., Kennedy, J., Klvaňová, A., Kmecl, P., Kuczyński, L., Kurlavičius, P., Kålås, J. A., Lehikoinen, A., Lindström, Å., Lorrillière, R., Moshøj, C., Nellis, R., Noble, D. G., Eskildsen, D. P., Paquet, J., Pélissié, M., Pladevall, C., Portolou, D., Reif, J., Schmid, H., Seaman, B., Szabo, Z. D., Szép, T., Florenzano, G. T., Teufelbauer, N., Trautmann, S., van Turnhout, C. A. M., Vermouzek, Z., Vikstrøm, T., Vořišek, P., Weiserbs, A., and Devictor, V. (2023). Farmland practices are driving bird population decline across europe. *Proceedings of the National Academy of Sciences of the United States of America*, 120.

[Stability AI, 2023] Stability AI (2023). stabilityai/sd-vae-ft-mse. Accessed: 2024-01-02.

[Su et al., 2017] Su, J., Vargas, D. V., and Sakurai, K. (2017). One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23:828–841.

[Szegedy et al., 2013] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. (2013). Intriguing properties of neural networks. *CoRR*, abs/1312.6199.