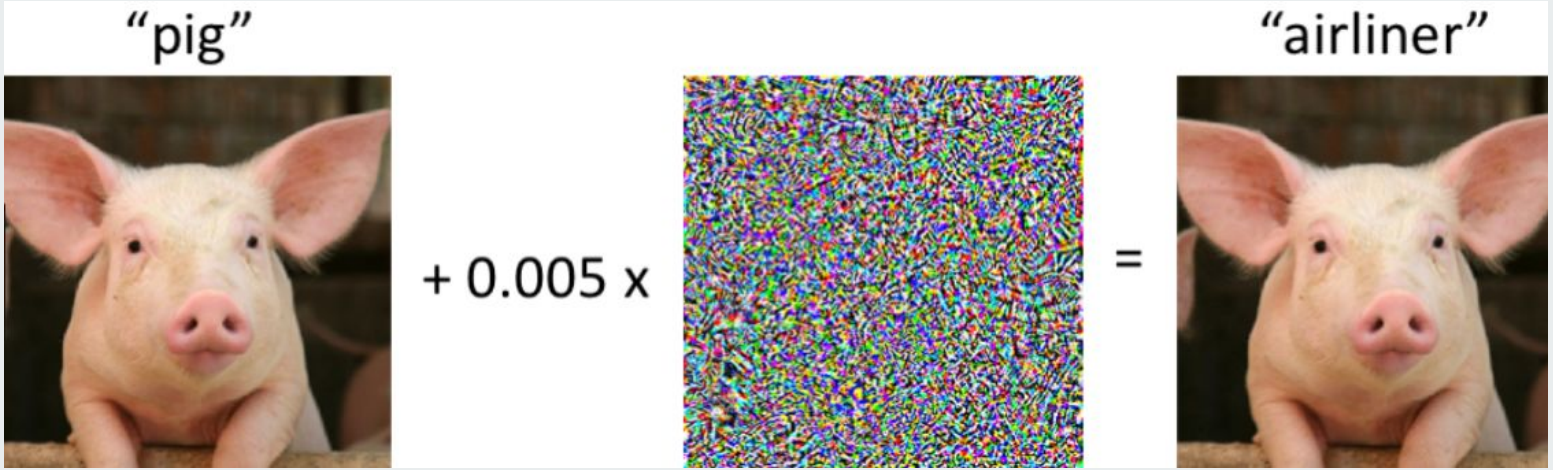
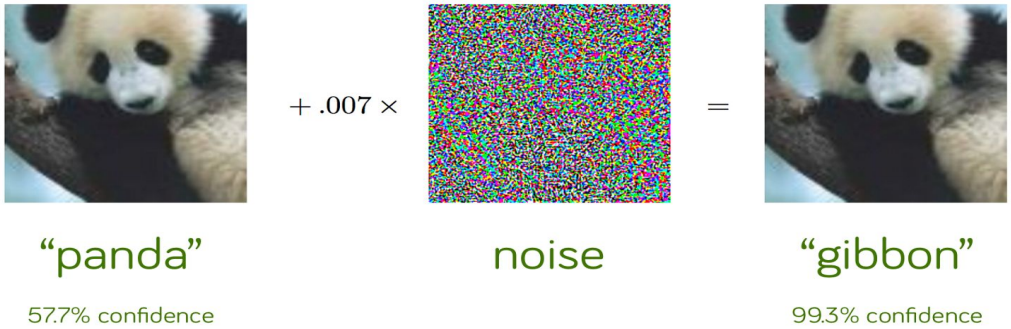
Introduction to adversarial attacks in pytorch

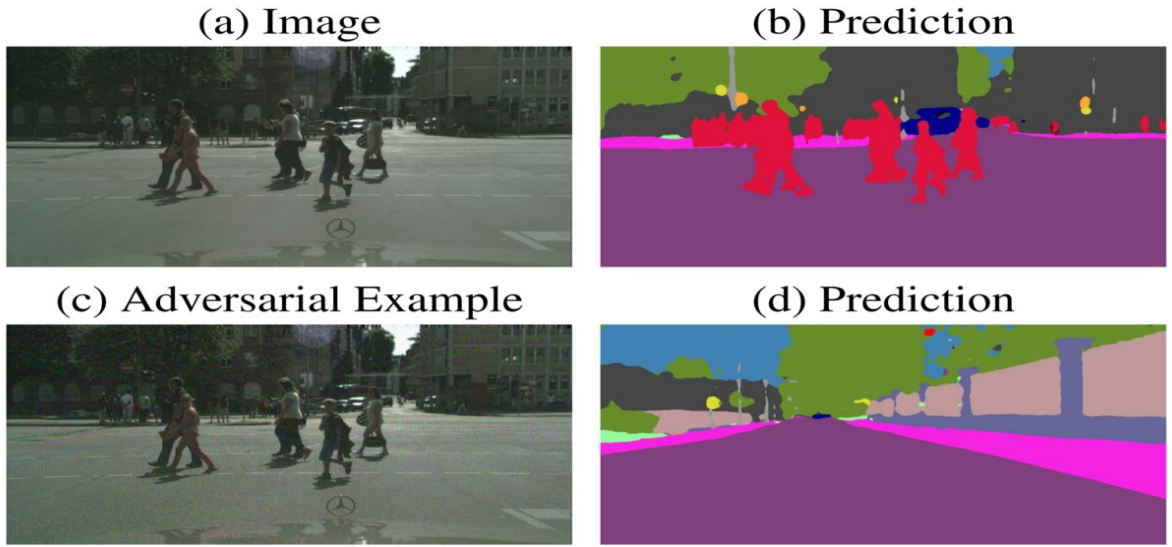


Adversarial attacks?



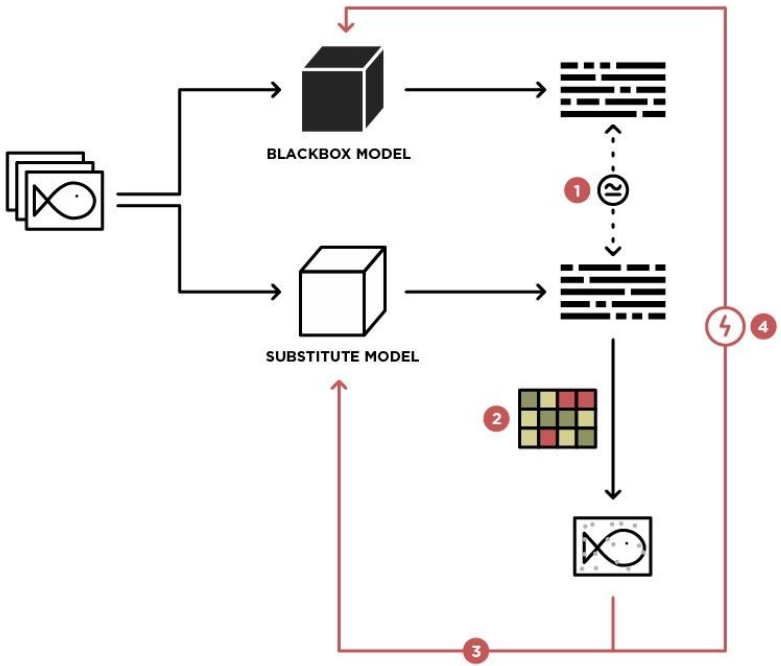
As we know, Deep Neural Networks are a very powerful tool to recognize patterns in data, and, for example, perform image classification on a human-level. However, can we somehow "trick" the model and find failure modes? On the slide we can see that the image of a panda with **addition of some noise** will be recognized as gibbon, with 99,3% confidence. So, that noise addition was our adversarial attack. We changed our image a little and now model can't classify that right, when human would easily say that it's still a panda.

Is it a big problem?



Yes, more and more deep learning models are used in applications, such as for example **autonomous driving**. Imagine that someone who gains access to the camera input of the car, could make pedestrians "disappear" for the image understanding network by simply adding some noise to the input as shown on image

Types of adversarial attacks



Adversarial attacks are usually grouped into "**white-box**" and "**black-box**" attacks. White-box attacks assume that we have access to the model parameter and can, for example, calculate the gradients with respect to the input (similar as in GANs). Black-box attacks on the other hand have the harder task of not having any knowledge about the network, and can only obtain predictions for an image, but no gradients or the like. In this presentation, we will focus on white-box attacks as they are usually easier to implement and follow the intuition of Generative Adversarial Networks (GAN). And also, we will use pretrained model “ResNet34”on ImageNet dataset.

Methods

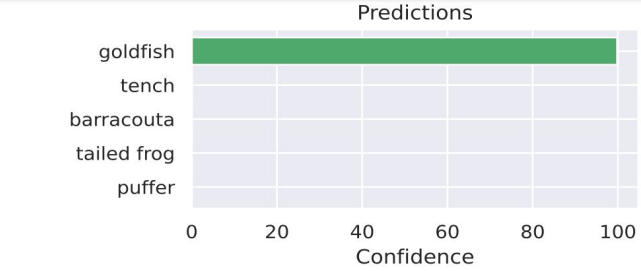
Fast Gradient Sign Method (FGSM)

$$\tilde{x} = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y))$$

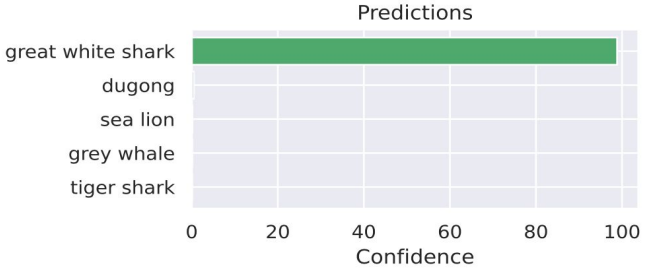
The term $J(\theta,x,y)$ represents the loss of the network for classifying input image x as label y; ϵ is the intensity of the noise, and $\tilde{x}$ the final adversarial example. The equation resembles SGD and is actually nothing else than that. We change the input image x in the direction of *maximizing* the loss $J(\theta,x,y)$. This is exactly the other way round as during training, where we try to minimize the loss. The sign function and ϵ can be seen as gradient clipping and learning rate specifically. We only allow our attack to change each pixel value by ϵ. Also the attack can be performed very fast, as it only requires a single forward and backward pass.

Images by pretrained classification model

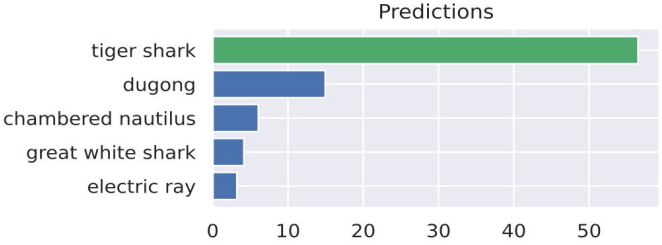
goldfish



great white shark



tiger shark

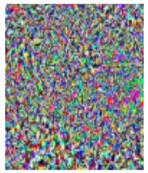


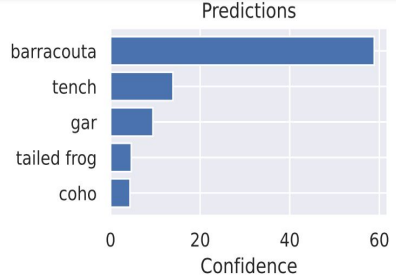
Those images with FSGM noise

goldfish

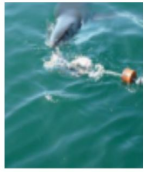
Adversarial

Noise

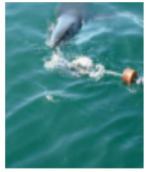




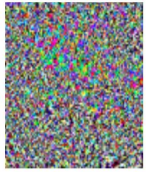
great white shark

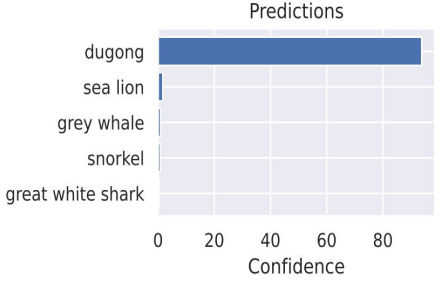


Adversarial



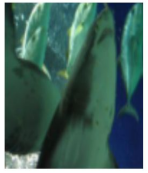
Noise



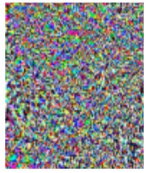


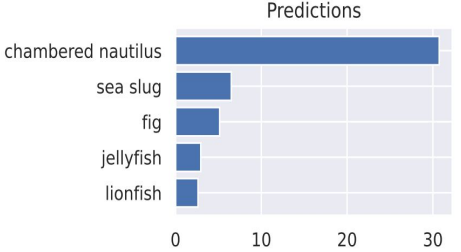
tiger shark

Adversarial



Noise

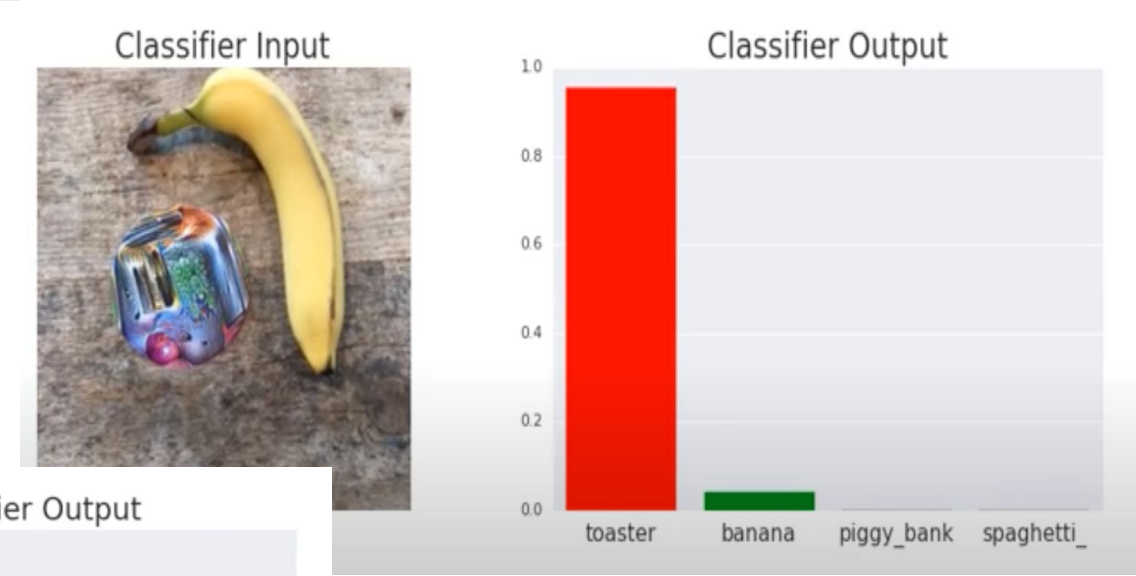


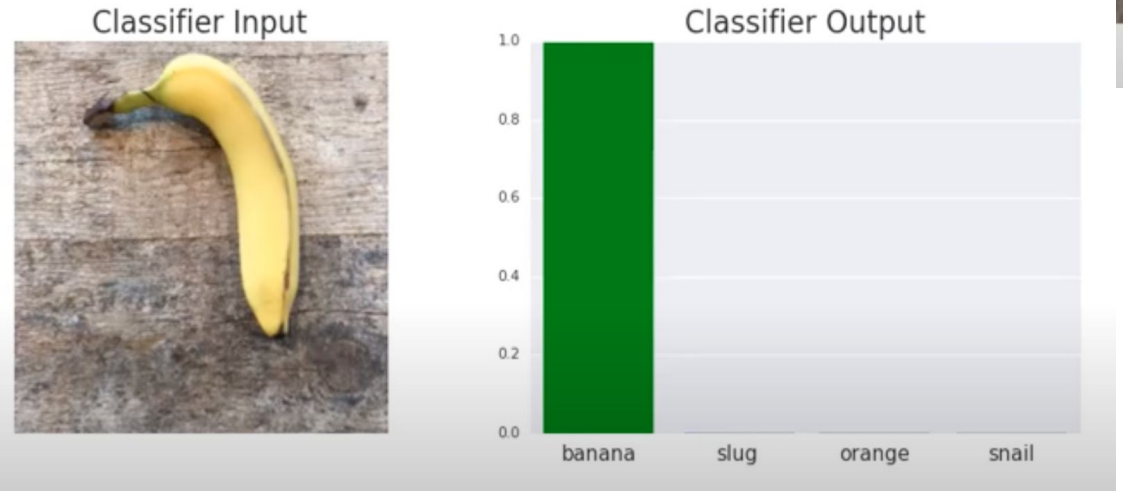


As we can see, only with one FSGM step, none of the right labels in the top 5, when for human pictures are the same. Also model have big confidence on adversarial image. And top 5 error now = 60%, when with dataset images top 5 error = 4,3%.

Methods

Adversarial Patches





With the addition of a small picture of the target class (here *toaster*) to the original image, the model does not pick it up at all. A specifically designed patch, however, which only roughly looks like a toaster, can change the network's prediction instantaneously. So let's try to change not all image, only part of it. This form of attack is an even bigger threat in real-world applications than FSGM.

Methods

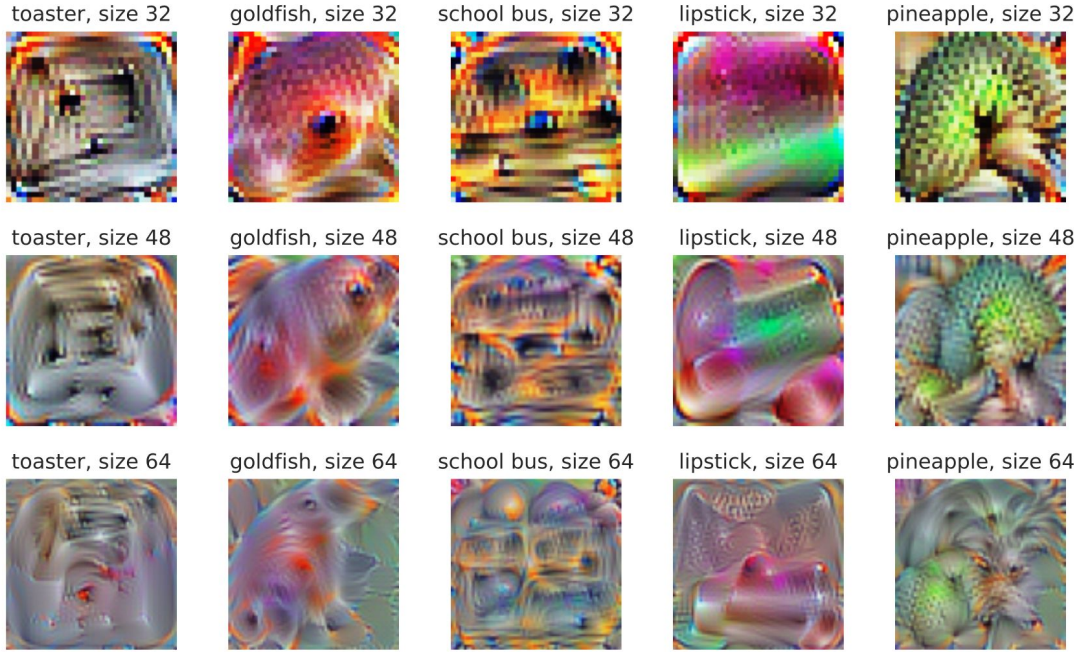
Adversarial Patches

The general idea is very similar to FSGM in the sense that we calculate gradients for the input, and update our adversarial input correspondingly. However, there are also some differences in the setup. Firstly, we do not calculate a gradient for every pixel. Instead, we replace parts of the input image with our patch and then calculate the gradients just for our patch. Secondly, we don't just do it for one image, but we want the patch to work with any possible image. Hence, we have a whole training loop where we train the patch using SGD. Lastly, image patches are usually designed to make the model predict a specific class, not just any other arbitrary class except the true label.

Methods

Adversarial Patches

First of all, we should create patches, depends on class of image and patch size. Patch will represent patterns that are characteristic of the class. We will use a simple SGD optimizer with momentum to minimize the classification loss of the model given the patch in the image



Methods

Adversarial Patches

Class name	Patch size 32x32	Patch size 48x48	Patch size 64x64
toaster	72.02%	98.12%	99.93%
goldfish	86.31%	99.07%	99.95%
school bus	91.64%	99.15%	99.89%
lipstick	70.10%	96.86%	99.73%
pineapple	92.23%	99.26%	99.96%

top 5 accuracy on our patches

Class name	Patch size 32x32	Patch size 48x48	Patch size 64x64
toaster	48.89%	90.48%	98.58%
goldfish	69.53%	93.53%	98.34%
school bus	78.79%	93.95%	98.22%
lipstick	43.36%	86.05%	96.41%
pineapple	79.74%	94.48%	98.72%

top 1 accuracy on our patches

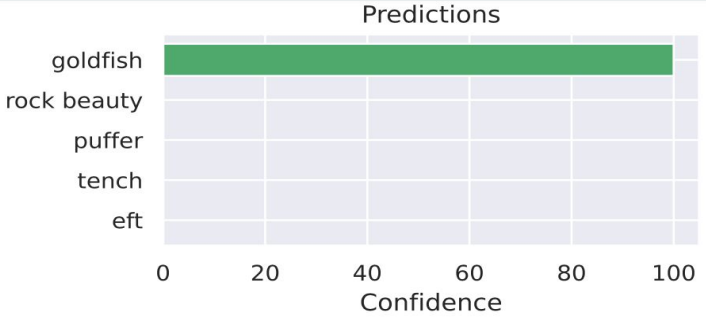
The clear trend, that we would also have expected, is that the larger the patch, the easier it is to fool the model.

Methods

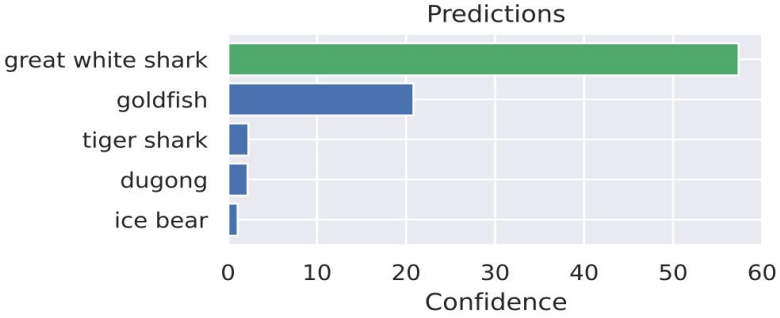
Adversarial Patches

With addition small goldfish (size=64), our classifier will make a lot of mistakes

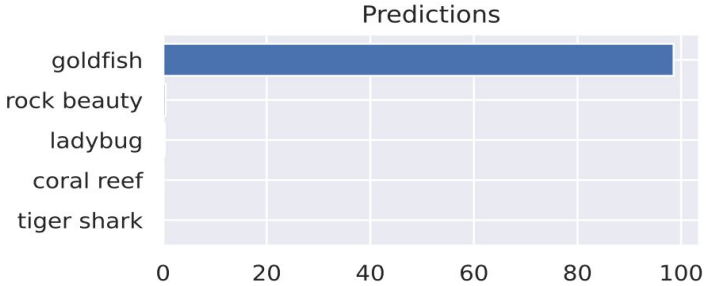
goldfish



great white shark



tiger shark

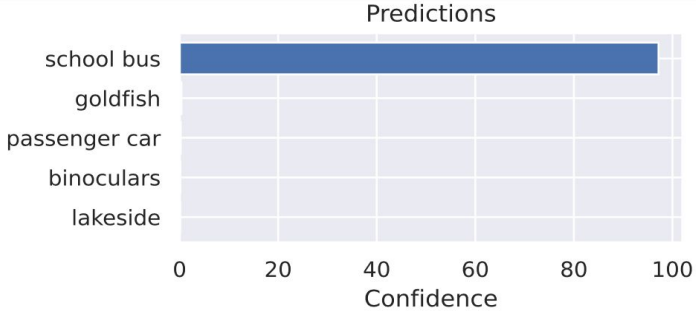


Methods

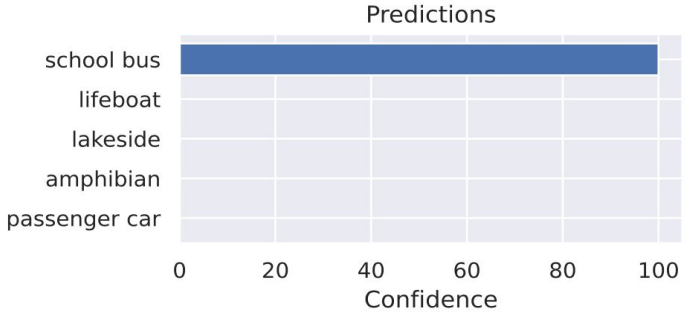
Adversarial Patches

By the way, none of the images have anything to do with an American school bus, the high confidence of often 100% shows how powerful such attacks can be.

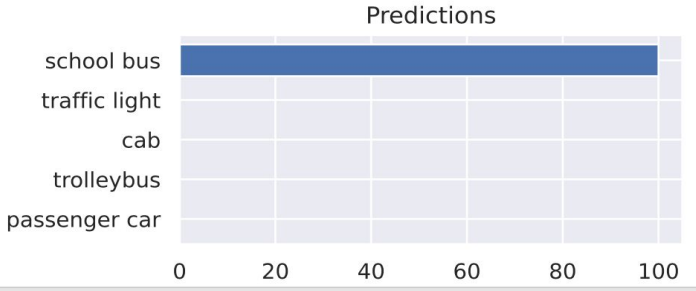
goldfish



great white shark

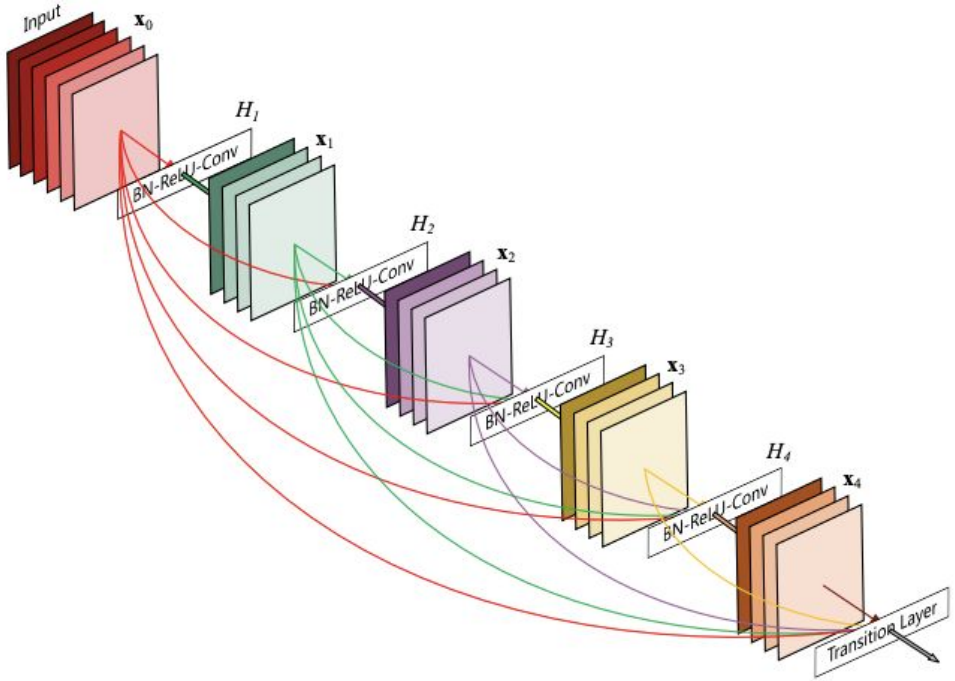


tiger shark



Transferability of white-box attacks

FGSM and the adversarial patch attack were both focused on one specific image. However, can we transfer those attacks to other models? How different are the patches for different models anyway? Let's evaluate some of our patches trained above on a different network, e.g. DenseNet121.



Transferability of white-box attacks

```
Testing patch "pineapple" of size 64x64
Top-1 fool accuracy: 65.41%
Top-5 fool accuracy: 82.81%
```

Attack on ResNet34 with image of pineapple

```
Testing patch "school bus" of size 64x64
Top-1 fool accuracy: 12.76%
Top-5 fool accuracy: 37.80%
```

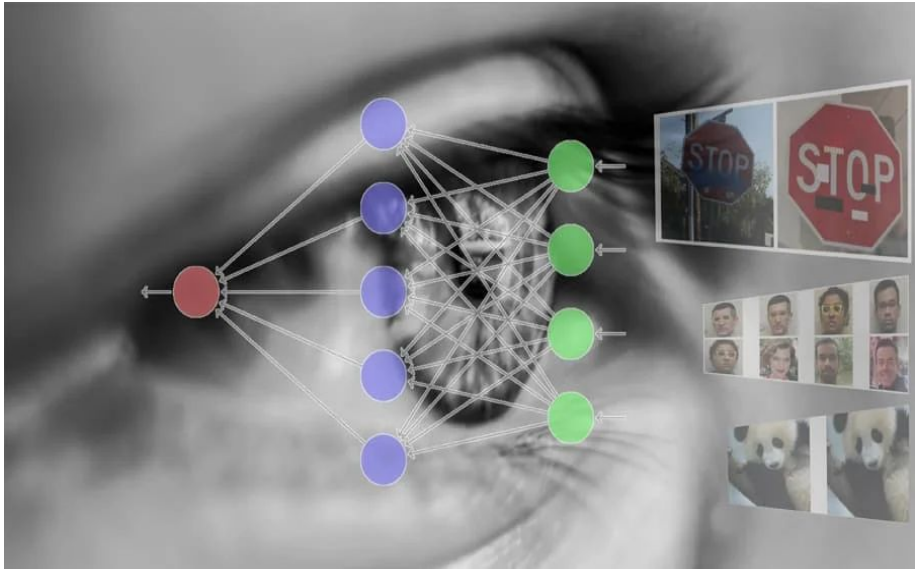
Attack on ResNet34 with image of school bus

Although the fool accuracy is significantly lower than on the original ResNet34, it still has a considerable impact on DenseNet although the networks have completely different architectures and weights. So we can easily attack different architectures, based on one dataset. Interesting, that now our goldfish is way better than school bus.

Protecting against adversarial attacks

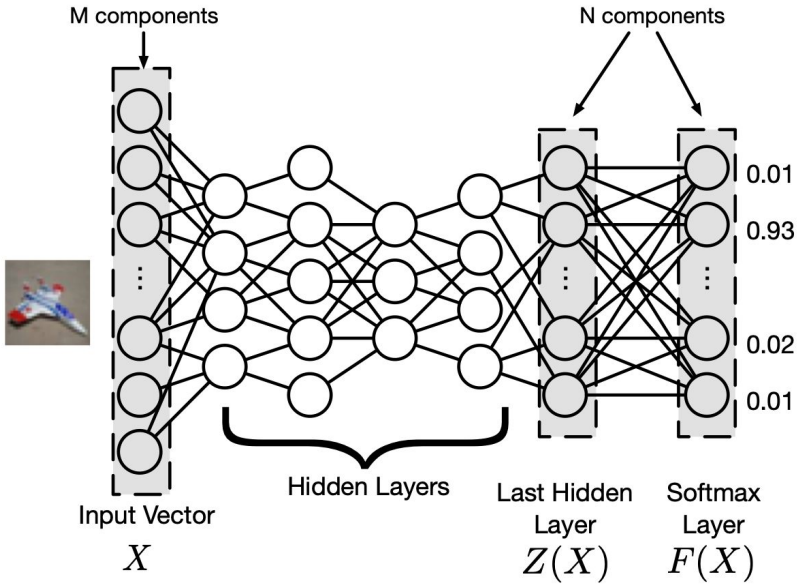
There are many more attack strategies than just FGSM and adversarial patches. However, what about the other perspective? What can we do to *protect* a network against adversarial attacks? The sad truth to this is: not much.

The easiest way of preventing white box attacks is by ensuring safe, private storage of the model and its weights. However, some attacks, called black-box attacks, also work without access to the model's parameters, or white-box attacks can also generalize as we have seen above on our short test on transferability.



Protecting against adversarial attacks

Another common trick to increase robustness against adversarial attacks is defensive distillation. Instead of training the model on the dataset labels, we train a secondary model on the softmax predictions of the first one. This way, the loss surface is "smoothed" in the directions an attacker might try to exploit, and it becomes more difficult for the attacker to find adversarial examples. Nevertheless, there hasn't been found the one, true strategy that works against all possible adversarial attacks.



Conclusion

Deep CNNs can be fooled by only slight modifications to the input. Whether it is a carefully designed noise pattern, unnoticeable for a human, or a small patch, we are able to manipulate the networks' predictions significantly. The fact that even white-box attacks can be transferable across networks, and that there exist no suitable protections against all possible adversarial attacks, make this concept a massive problem for real-world applications. While adversarial attacks can also be used for improving/training a robust model or a GAN, it is not close to being solved yet. This is also because neural networks are currently complex, unknown non-linear functions in high-dimensional looking for correlations instead of causation. In the next years, we might hopefully see an improvement in the stability of such models by using causal approaches and/or introducing uncertainty.

References

[1] **Tutorial 10: Adversarial attacks**

https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial10/Adversarial_Attacks.html

[2] **Google colab notebook “Introduction to Adversarial Attacks in Pytorch”**

https://colab.research.google.com/drive/1YvEMsjVmcGoAwy1XkL_FAWqQ_A7Crew4?usp=sharing