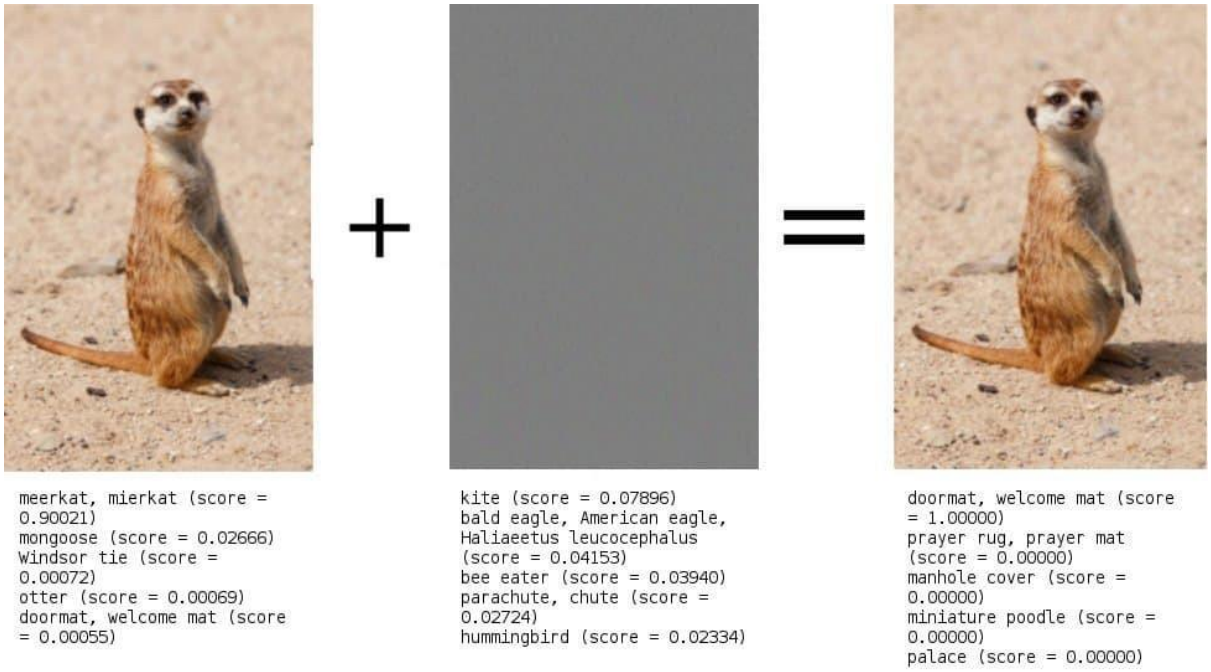
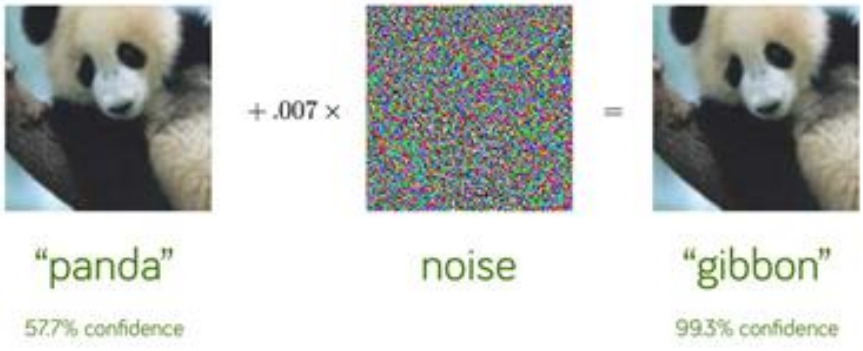
An Adversarial Attack



By : Aryaman Gautam J001

What is an Adversarial Attack?

The Adversarial attack consists of subtly modifying an original image in such a way that the changes are almost undetectable to the human eye. The modified image is called an adversarial image, and when submitted to a classifier it is misclassified, while the original one is correctly classified. The real-life applications of such attacks can be very serious –for instance, one could modify a traffic sign to be misinterpreted by an autonomous vehicle, and cause an accident.



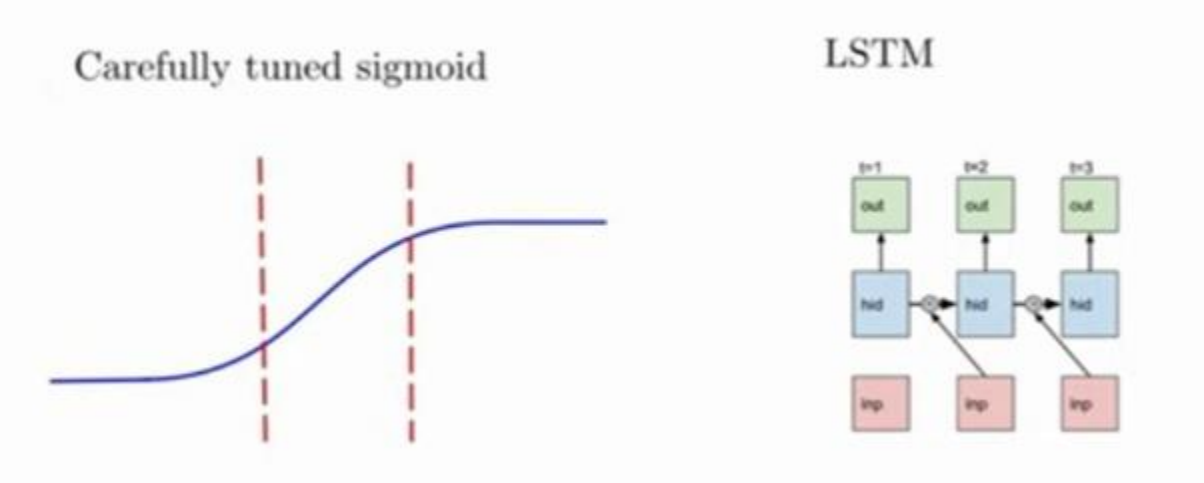
If we consider the above panda's example we have an image that is presented to the CNN as a 32 bit point floating values are used and the monitor can only display 8 bits of colour resolution. So small imperceptible changes are done to the other 24 bit values and this tiny change barely affects the 8 bits and doesn't alter the image much but is enough to fool the CNN.

Why do they happen?

So why does this imperceptible change lead to a misclassification?

Can it be due to overfitting the model or underfitting?

Turns out the answer is it's because of Linearity of the model.



Most modern deep neural networks and other NN may not be a single linear function but they are piecewise linear.

As we can see that is the case for these activation functions (Relu & Maxout).

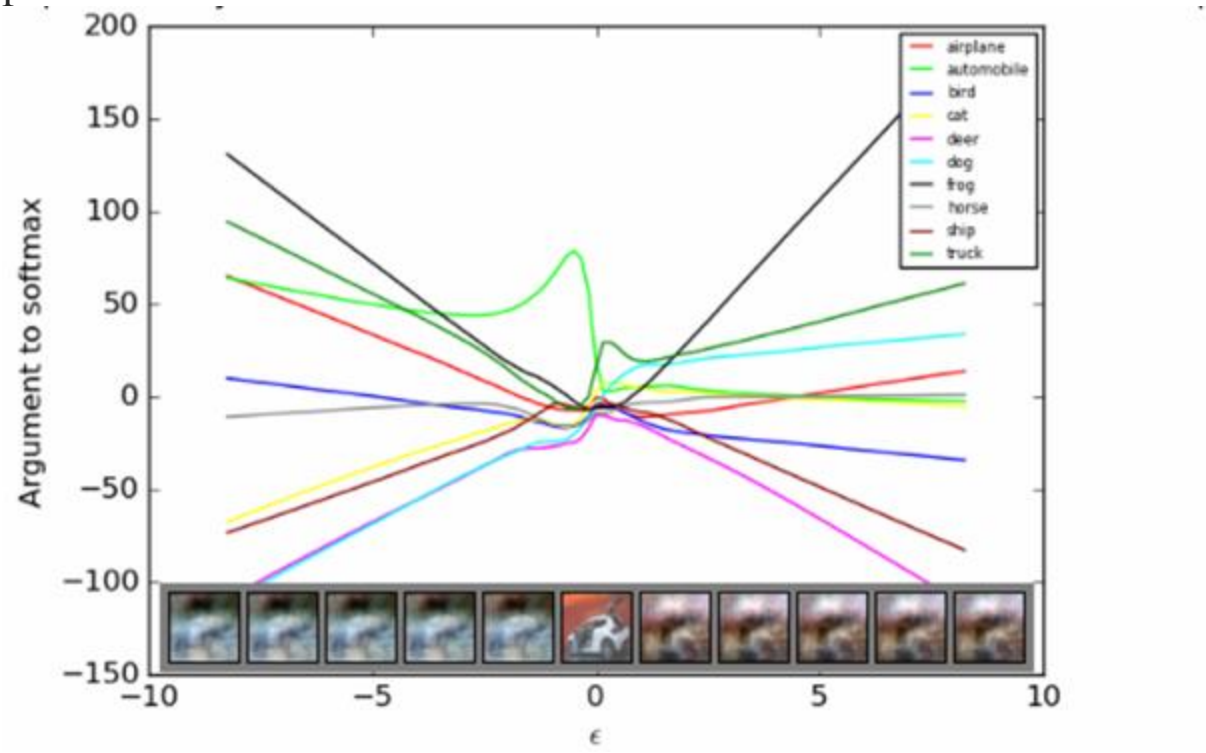
Before we started using Relu, sigmoid and hyperbolic tan functions would be utilised in the models and at that time initialisation had to be done so that the model would spend time near the centre of the sigmoid where it is approximately linear.

The newer more commonly used LSTM is a NN that utilises addition from one timestep to the next and we already know addition is a fairly simple form of linearity.

A thing to clarify is that this linearity is wrt the input of the model to the and not the parameters due to multiplication by weight matrices in each layer making them non-linear.

These output logits are piecewise linear (not all of them is linear but have enough linearity) and represent the unnormalized log probabilities before the SoftMax function is applied to them.

So it is actually easier to optimize input to model rather than parameters for an adversarial attack.

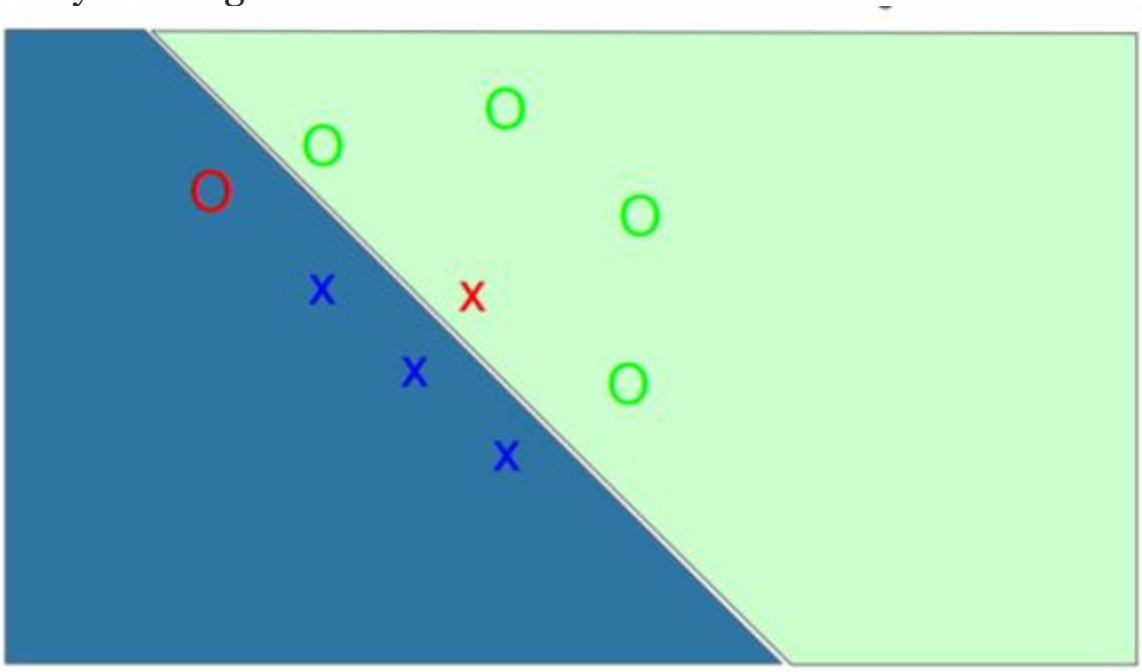


Next, we take a look at the example of an image where we have a car with a red background and have taken a random coefficient epsilon which is multiplied by and moved in the direction. At -10 subtract epsilon and at +10 add. At zero we have our original dataset.

At zero we can see the model correctly predicts the image as an automobile while the other times it misclassifies the image as a frog.

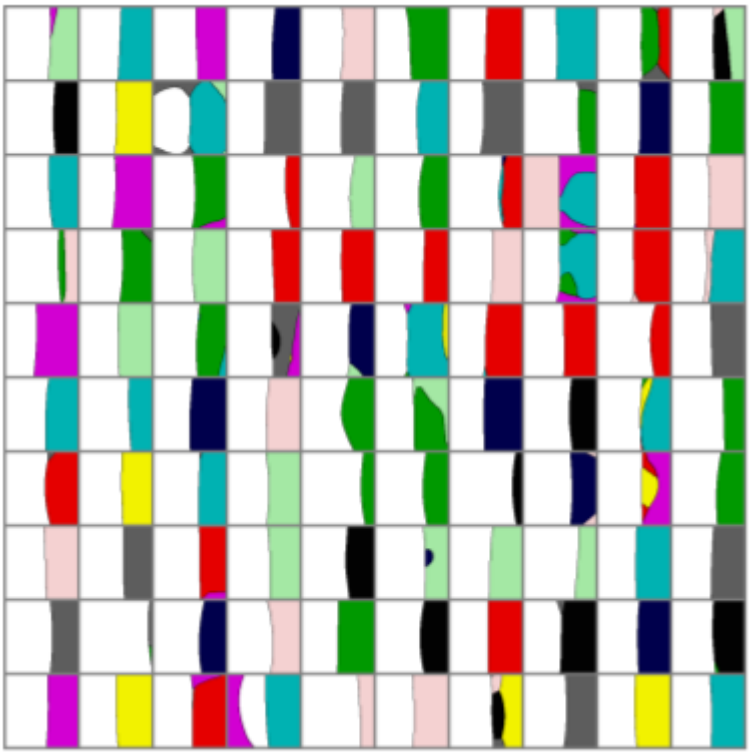
This graph was the result of looking at the linear path from a scaling factor of negative 10 to positive 10. We see that the logits output by the network behaves linearly far from the data.

i.e away from the training example i.e at the centre they start to classify with high confidence.



Where away from training example it classifies the corners which have no training data as either x or os with high confidence.

Two-dimensional cross-sections of the classification function :



Over here we have church window plots. This plot shows two-dimensional cross-sections of the classification function, exploring input space near test set examples.

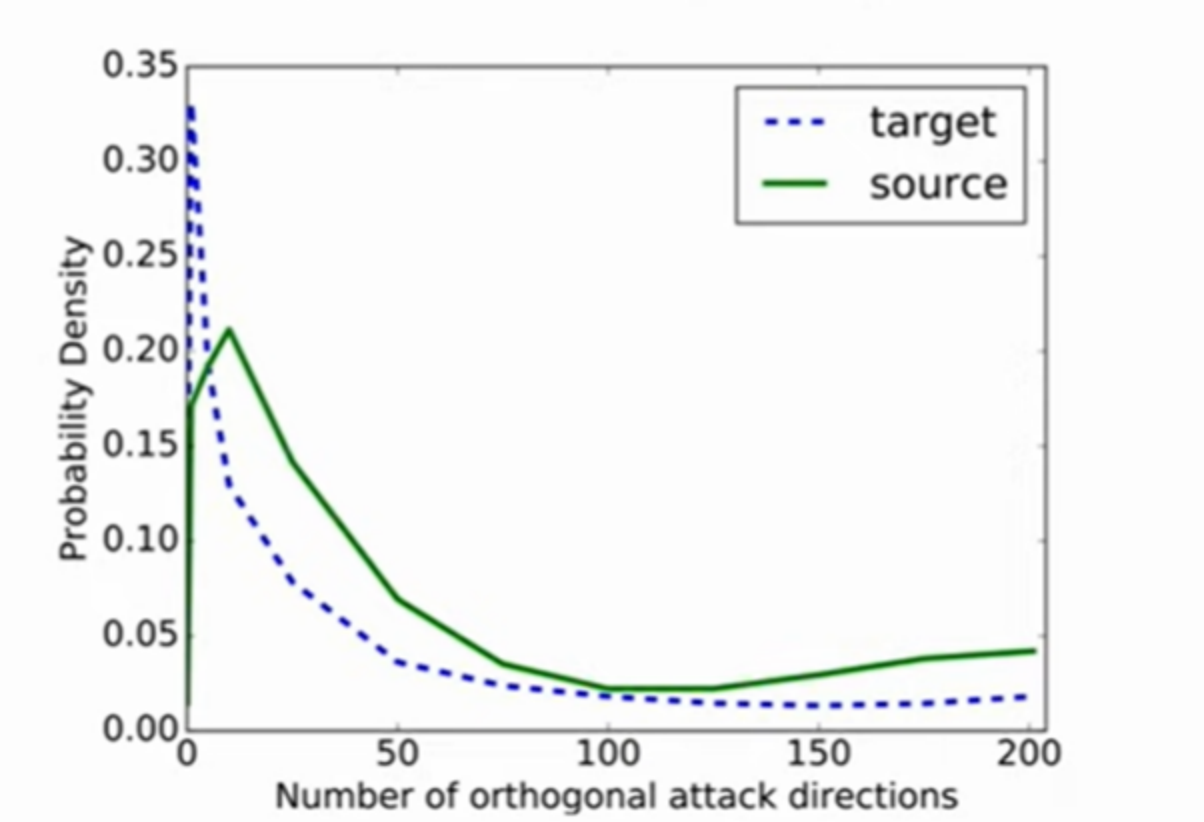
Each cell in the 10 × 10 grid in the figure is a different church window plot corresponding to a different CIFAR-10 test example. , each pixel displayed with a unique colour indicating the class

The correct class for each example, given by the test set label, is always plotted as white

So far it has been observed that these adversarial examples are found due to linearity in the model.

Transferability of adversarial attacks :

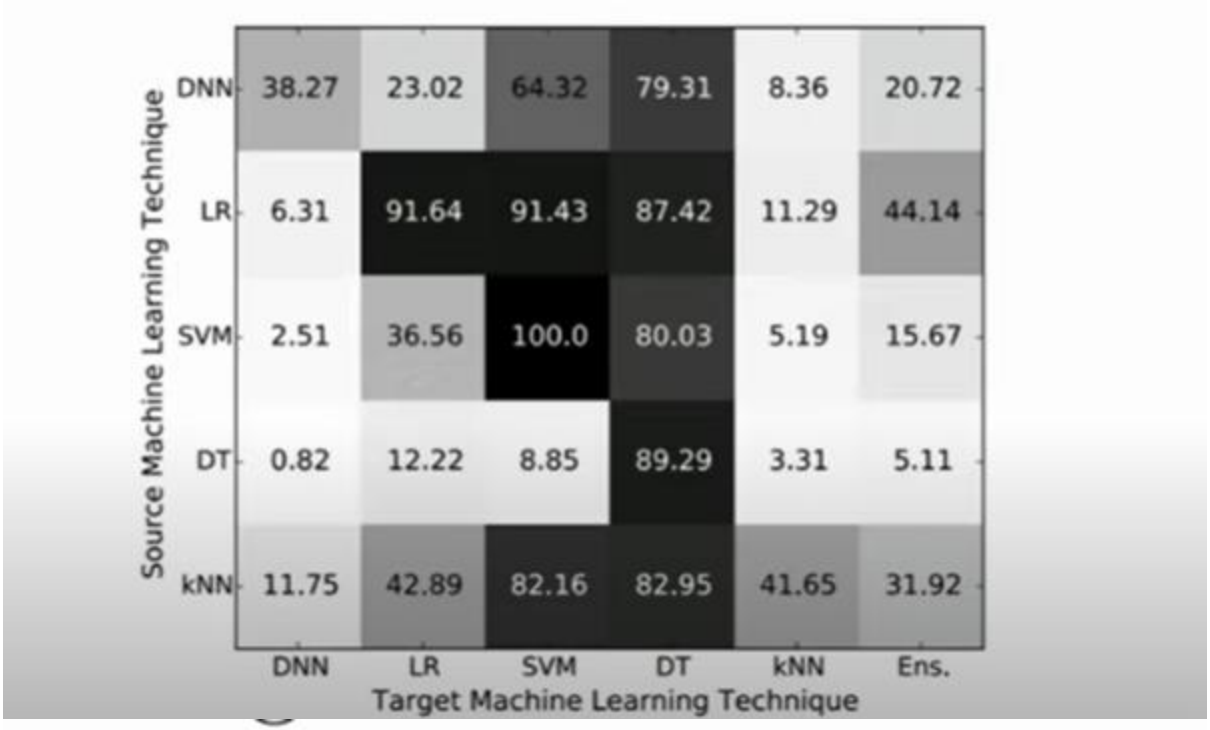
Furthermore, they've been empirically proven rather than being scattered randomly in small pockets, adversarial examples occur in large, continuous regions.

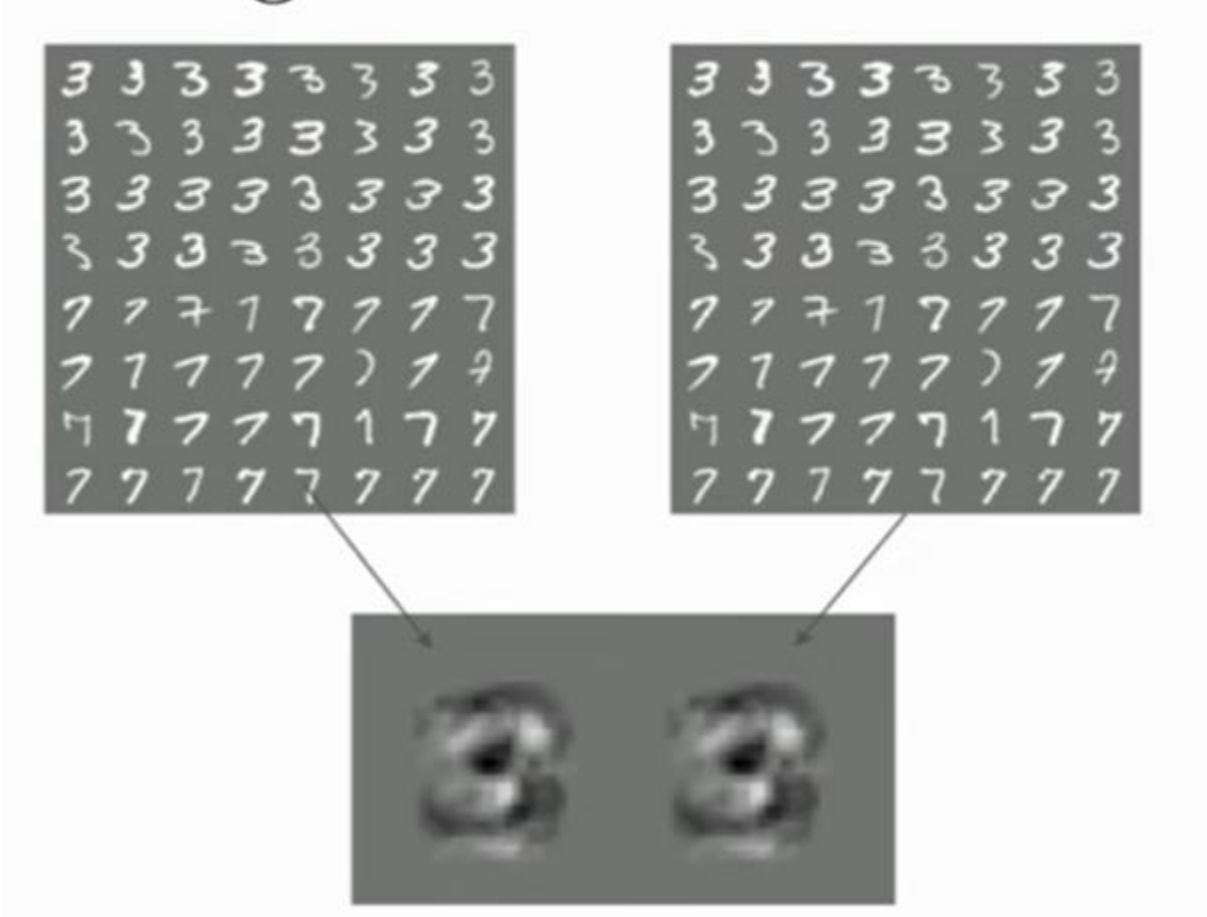


In addition to that, there is the transferability problem too i.e the higher the dimensionality, the more likely it is that the subspaces of two models will intersect significantly and so the same attacks can be used on both models effectively.

We find that when moving into any direction away from data points, the distance to the model's decision boundary is on average larger than the distance separating the boundaries of two models in that direction. Thus, adversarial perturbations that send data points sufficiently over a model's decision boundary likely transfer to other models.

Why are they transferable ?





In the first image, we saw which models are similar so adversarial attacks can easily work on multiple models i.e trained on one can effectively be implemented to the other.

A thing to note here is that SVM is very data-dependent hence its low values.

And along with the second one, we saw that adversarial examples/attacks can be generalised and transferred from one model to another as well as for disjoint/different datasets to the other for training.

This happens because the weight vectors look similar to each other because we want our model to generalise and be able to learn somewhat independently of just what we've trained.

Furthermore, for carrying out adversarial attacks it's found that if we take an ensemble of different models and get adversarial examples for them then there's a very high likelihood that it'll be transferable and fool a different ML model.

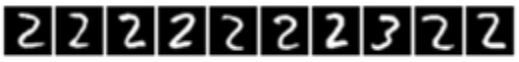
To recap :

Computer Vision relies on feature extraction for image identification.

The individual coordinates of the feature space link back to meaningful variations on the input domain

If we were to move in random directions in this space, we would get inputs with similarly interpretable semantic properties (eg images that consider the upper left curve for classifying numbers of MNIST dataset).

(a) Unit sensitive to lower round stroke.



(b) Unit sensitive to upper round stroke, or lower straight stroke.

Stacks of layers between the input and output of Neural Network are a way to encode non-local generalization prior over input space i.e Non-significant probabilities are assigned to regions of input space that contain no training examples in its vicinity.

Such regions can represent, for instance, the same objects from different viewpoints, which are relatively far (in pixel space), but which share nonetheless both the label and the statistical structure of the original inputs.

And that in particular, for a small enough radius $\varepsilon > 0$ in the vicinity of a given training input x, and x + r satisfying $||r|| < \varepsilon$ will get assigned a high probability of the correct class by the model.

This kind of smoothness assumption that underlies many kernel methods does not hold and imperceptibly tiny perturbations of a given image can lead to misclassification/ change in the underlying class which are our adversarial example.

THREAT MODELS :

They can be classified on the basis of the **Adversary's Goal**

Does he want to misguide the classifier's decision on one sample, or influence the overall performance of the classifier?

• Poisoning Attack vs Evasion Attack

Poisoning attacks refer to the attacking algorithms that allow an attacker to insert/modify several fake samples into the training database of a DNN algorithm. These fake samples can cause failures of the trained classifier. They can result in poor accuracy (Biggio et al., 2012), or wrong prediction on some given test samples (Zugner et al. ¨, 2018). This type of attack frequently appears in the situation where the adversary has access to the training database. For example, web-based repositories.

In evasion attacks, the classifiers are fixed and usually have a good performance on benign testing samples. The adversaries do not have the authority to change the classifier or its parameters, but they craft some fake samples that the classifier cannot recognize. In other words, the adversaries generate some fraudulent examples to evade detection by the classifier. For example, in autonomous driving vehicles, sticking a few pieces of tapes on the stop signs can confuse the vehicle's road sign recognizer

• Targeted Attack vs Non-Targeted Attack

In a targeted attack, the adversary aims to induce the classifier to give a specific label to the perturbed sample. For Eg fraudster is

likely to attack a financial company's credit evaluation model to disguise himself as a highly credible client of this company

If there is no specified target label for the victim sample the attack is called a non-targeted attack. The adversary only wants the classifier to predict incorrectly.

Or on the basis of ADVERSARY'S KNOWLEDGE :

• **Black Box :**

In the black-box model, an adversary does not know the structure of the target network or the parameters but can interact with the DL algorithm to query the predictions for specific inputs. The adversaries always craft adversarial samples on a surrogate classifier trained by the acquired data-and-prediction pairs and other benign/adversarial samples. I.e send inputs and observe outputs and use this as a training dataset. The class labels themselves are sufficient and there's no need for all the probability values of its output.

Owing to the transferability of adversarial samples, black-box attacks can always compromise a naturally trained non-defensive model.

• **Grey Box:**

an adversary is assumed to know the architecture of the target model, but to have no access to the weights in the network. The

adversary can also interact with the DL algorithm. In this threat model, the adversary is expected to craft adversarial samples on a surrogate classifier of the same architecture. Due to the additional structure information, a grey-box adversary always shows better attack performance compared with a black-box adversary.

• **White Box:**

The strongest adversary — that is, the white-box adversary — has full access to the target model including all the parameters, which means that the adversary can adapt the attacks and directly craft adversarial samples on the target model

The PERTURBATION METRICS :

i.e Measuring the magnitude of the perturbation. Commonly used metrics are :

1) **L0 distance** corresponds to the number of pixels that have been altered in an image.

primary distance metric under which defensive distillation's security is argued

2) **L2 distance** measures the standard Euclidean (root-mean-square) distance between x and x0. The L2 distance can remain small when there are many small changes to many pixels. This distance metric was used in the initial adversarial example work of the panda.

i.e distance between the adversarial example and the original sample.

3) **L∞ distance** measures the maximum change to any of the coordinates. For images, we can imagine there is a maximum budget, and each pixel is allowed to be changed up to this limit, with no limit on the number of pixels that are modified. Goodfellow et al. argue that L∞ is the optimal distance metric to use

#Maybe add notation and description :

Some Examples of adversarial attacks

1) LBFGS

Limited Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) algorithm

It searches for the minima distorted adversarial example and to find it the algorithm treats it as an optimization problem with the objective :

$$\text{minimize } ||x - x'||_2^2$$
$$\text{subject to } C(x') = t \text{ and } x' \in [0,1]^m$$

This is made easier to solve by introducing the loss function leading to the objective

$$\text{minimize } c||x - x'||_2^2 + \mathcal{L}(\theta, x', t)$$
$$\text{subject to } x' \in [0,1]^m$$

The second term encourages the algorithm to find ‘x’ (x dash) i.e adversarial example which has a small loss value to label t so the classifier C is likely to predict ‘x’ as t.

L(x0, t) is the true loss function of the targeted model (e.g., categorical cross-entropy), and t is the target misclassification label. Since this objective does not guarantee that x 0 will be adversarial for any specific value of c > 0.

the resulting c value can be further optimized using the bisection method (a.k.a. binary search) between the range of the final line search segment.

This attack was successfully applied to misclassify many image instances on both AlexNet and QuocNet, which were state-of-the-art classification models at the time. Thanks to its L2 Euclidean distance constraint, L-BFGS produces adversaries that are perceptually similar to the original input x.

2) Fast Gradient Sign Method

It is a one-step method is introduced to fast generate adversarial examples.

It is designed to quickly find a perturbation direction for a given input such that the training loss function of the target model will increase, reducing classification confidence and increasing the likelihood of inter-class confusion

Intuitively, for each pixel, the fast gradient sign method uses the gradient of the loss function to determine in which direction the pixel's intensity should be changed (whether it should be increased or decreased) to minimize the loss function; then, it shifts all pixels simultaneously. It is important to note that the fast gradient sign attack was designed to be fast, rather than optimal. It is not meant to produce the minimal adversarial perturbations

FGSM works by calculating the gradient of the loss function with respect to the input and creating a small perturbation by multiplying a small chosen constant e(epsilon) by the sign vector of the gradient

$$\text{minimize } \mathcal{L}(\theta, x', t)$$
$$\text{subject to } ||x' - x||_\infty \leq \epsilon \text{ and } x' \in [0,1]^m$$

$$x' = x + \epsilon \cdot \text{sign}(\nabla_x \mathcal{L}(x, y)),$$

It can be applied anywhere where delta x L(x,y) can be calculated. It computes gradient analytically compared to the numerically optimised LBFGS attack and so finds solutions much faster.

the generated adversarial examples must be within the bounds of the input space which is enforced by value-clipping and the input perturbation should be bound under the L∞ supremum metric to encourage perceptual similarity between x and 'x'.

Under this ∞-norm constraint, the sign of the gradient vector maximizes the magnitude of the input perturbation, which

consequently also amplifies the adversarial change in the model's output.

3) BIM

BIM was the first method shown to be effective with printed paper examples

It is an extension of FGSM. It runs finer iterate ve optimizer for multiple iterations i.e performs it with smaller step sizes and clips the updated adversarial example for a range of t iterations i.e for 't'th iteration updation was

$$x'_{t+1} = \text{Clip}\left\{x'_t + \alpha \cdot \text{sign}[\nabla_x J(\theta, x'_t, y)]\right\}$$

And the adversarial example generated is defined as

$$x'_{i+1} = Clip_{\epsilon}\Big\{x'_i + \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(x'_i, y))\Big\} \text{ for } i = 0 \text{ to } n, \text{ and } x'_0 = x$$

4) Momentum Iterative attack :

Won first place in the non-targeted adversarial attack and targeted adversarial attack competitions (black-box setting) at the 2017 Neural Information Processing Systems (NIPS) conference.

is a technique for accelerating gradient descent algorithms by accumulating a velocity vector in the gradient direction of the loss function across iterations. The memorization of previous gradients helps to barrel through narrow valleys, small humps and poor local

minima or maxima. The momentum method also shows its effectiveness in stochastic gradient descent to stabilize the updates.

We apply the idea of momentum to generate adversarial examples and obtain tremendous benefits. To generate a non-targeted adversarial example x ∗ from a real example x, which satisfies the L∞ norm bound, gradient-based approaches seek the adversarial example by solving the constrained optimization problem

In contrast, iterative FGSM greedily moves the adversarial example in the direction of the sign of the gradient in each iteration (in Eq. (3)). Therefore, the adversarial example can easily drop into poor local maxima and “overfit” the model, which is not likely to transfer across models.

$$\arg\max_{\boldsymbol{x}^*} J(\boldsymbol{x}^*, y), \quad \text{s.t.} \ \ \|\boldsymbol{x}^* - \boldsymbol{x}\|_\infty \leq \epsilon,$$

Hence why Momentum Iterative attack is better than FGSM.

Iterative Least Likely Class Method :

A variant of BIM where the goal is to generate an adversarial example that is misclassified as a specific target t. IT targets the one with the least likelihood of being chosen by the original classifier.

t = argmin f(x)

The iterative update is given by

$$x'_{i+1} = Clip_{\epsilon}\left\{x'_i - \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(x'_i, t))\right\} \text{ for } i = 0 \text{ to } n, \text{ and } x'_0 = x.$$

Similar to

$$x'_{i+1} = Clip_{\epsilon}\left\{x'_i + \alpha \cdot \text{sign}(\nabla_x \mathcal{L}(x'_i, y))\right\} \text{ for } i = 0 \text{ to } n, \text{ and } x'_0 = x$$

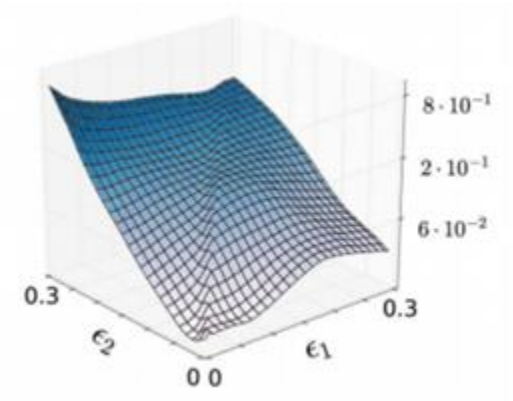
Of BIM.

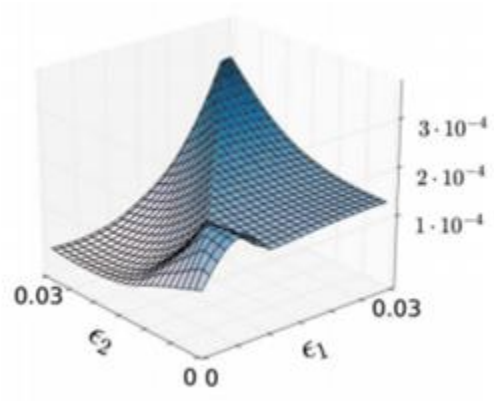
Except for the predicted class of cross-entropy loss is changed from true label to target label with the sign of gradient update reversed.

This model reduces classification loss of an adversarial training pair so the model is misguided to have excess confidence towards the target class t as opposed to the other models which increase training loss effectively “undoing training” and encourage inter-class confusion

R+FGSM :

The randomized single-step attack, or R+FGSM , adds a small random perturbation to the input before applying the adversarial perturbation generated by FGSM. This helps avoid the defensive strategy of gradient masking





So in it we add some random perturbations sampled from a Gaussian distribution before calculating the first derivative of the loss with respect to the input.

DeepFool :

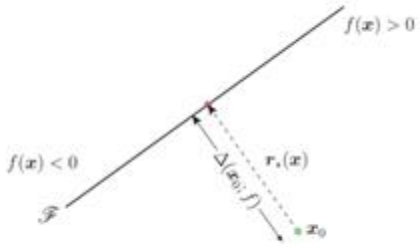


Figure 2: Adversarial examples for a linear binary classifier.

Algorithm 1 DeepFool for binary classifiers

input: Image x, classifier f.
output: Perturbation $\hat{r}$.
Initialize $x_0 \leftarrow x$, $i \leftarrow 0$.
while $\text{sign}(f(x_i)) = \text{sign}(f(x_0))$ **do**
  $r_i \leftarrow -\frac{f(x_i)}{\|\nabla f(x_i)\|_2^2} \nabla f(x_i)$.
  $x_{i+1} \leftarrow x_i + r_i$.
  $i \leftarrow i + 1$.
end while
return $\hat{r} = \sum_i r_i$.

Estimates the distance of an input instance x to the closest decision boundary of a multi-class classifier and can be used as minimal adversarial perturbation direction

1. The algorithm takes an input `x` and a classifier `f` .
2. Outputs the minimal perturbation required to misclassify the image.
3. Initialize the adversarial image with the original input. And the loop variable to 1.
4. Start and continue the loop while the true label and the label of

the adversarially perturbed image is the same.
5. Calculate the projection of the input onto the closest hyperplane. (minimal perturbation)
6. Add that perturbation to the image and test.
7–8. Increment Loop Variable; End Loop
9. Return the minimal perturbation

DeepFool iteratively perturbs the input x by linearizing the model's per-class decision boundaries around the current set-point x0, identifies the class j with the closest linearized decision boundary, and moves x0 i to this estimated boundary point. As shown in the Algorithm, this process is repeated till f(x 0 i) becomes misclassified

In practice, once an adversarial perturbation r is found, the adversarial example is nudged further beyond the decision boundary to guarantee misclassification

Jacobian-based Saliency Map Attacks

The notion of the saliency map was originally conceived for visualizing how deep neural networks make predictions [118]. The saliency map rates each input feature (e.g., each pixel in an image) by its influence upon the network's class prediction. Jacobian-based Saliency Map Attacks (JSMA) [80] exploit this information by perturbing a small set of input features to cause misclassification.

This is in contrast to attacks like the FGSM [32] (see Section 4.2) that modify most, if not all, input features.

$$S^+(x_{(i)}, c) = \begin{cases} 0 & \text{if } \frac{\partial f(x)_{(c)}}{\partial x_{(i)}} < 0 \text{ or } \sum_{c' \neq c} \frac{\partial f(x)_{(c')}}{\partial x_{(i)}} > 0 \\ -\frac{\partial f(x)_{(c)}}{\partial x_{(i)}} \cdot \sum_{c' \neq c} \frac{\partial f(x)_{(c')}}{\partial x_{(i)}} & \text{otherwise} \end{cases}$$

$S^+(\cdot)$ measures how much $x_{(i)}$ *positively* correlates with c, while also *negatively* correlates with all other classes $c' \neq c$. If either condition is violated, then saliency is reset to zero.

An attacker can exploit this saliency map by targeting an adversarial class t that does not match the true class label y of a given sample x. By *increasing* a few high-saliency pixels $x_{(i)}$ according to $S^+(x_{(i)}, c = t)$, the modified image x' will have an increased prediction confidence $f(x')_{(t)}$ for the adversarial class $t \neq y$, and thus might result in misclassification.

Alternatively, one can attack by *decreasing* feature values based on another saliency map $S^-(\cdot)$, which differs from S^+ only by the inversion of the low-saliency inequalities, i.e. $S^-\left(x_{(i)}, t\right) = 0$ if $\frac{\partial f(x)_{(t)}}{\partial x_{(i)}} > 0$ or $\sum_{c \neq t} \frac{\partial f(x)_{(c)}}{\partial x_{(i)}} < 0$.

In practice, both saliency measures S^+ and S^- are overly strict when applied to individual input features, because

Substitute Blackbox Attack:

the adversary seeks to force a classifier to misclassify inputs in any class different from their correct class. To achieve this, we consider a weak adversary with access to the DNN output only. The adversary has no knowledge of the architectural choices made to design the DNN, which include the number, type, and size of layers, nor of the training data used to learn the DNN's parameters. i.e a BlackBox threat.

Algorithm 1 - Substitute DNN Training: for oracle $\tilde{O}$, a maximum number max_ρ of substitute training epochs, a substitute architecture F, and an initial training set S_0.

Input: $\tilde{O}$, max_ρ, S_0, λ

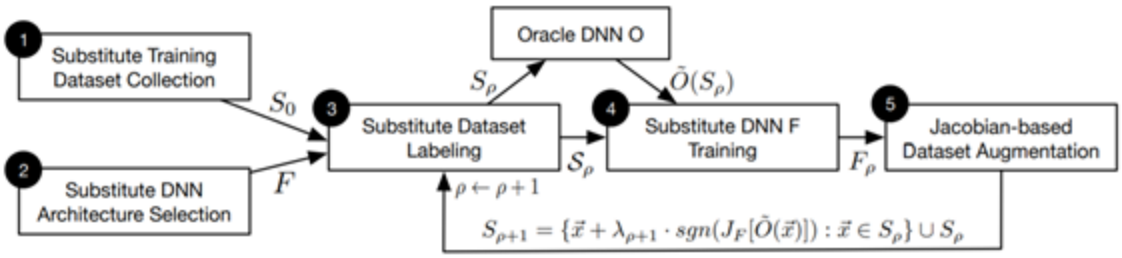
Define architecture F
for $\rho \in 0 \,..\, max_\rho - 1$ **do**
    // Label the substitute training set
    $D \leftarrow \left\{(\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho\right\}$
    // Train F on D to evaluate parameters θ_F
    $\theta_F \leftarrow \text{train}(F, D)$
    // Perform Jacobian-based dataset augmentation
    $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot \text{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$
end for
return θ_F

All of the techniques covered so far are white-box attacks, relying upon access to a model's innards

The key idea is to train a substitute model to mimic the BlackBox model and use Whitebox attack methods on this substitute. This approach leverages the transferability property of adversarial examples. Concretely, the attacker first gathers a synthetic dataset, obtains predictions on the synthetic dataset from the targeted model, and then trains a substitute model to imitate the targeted model's predictions

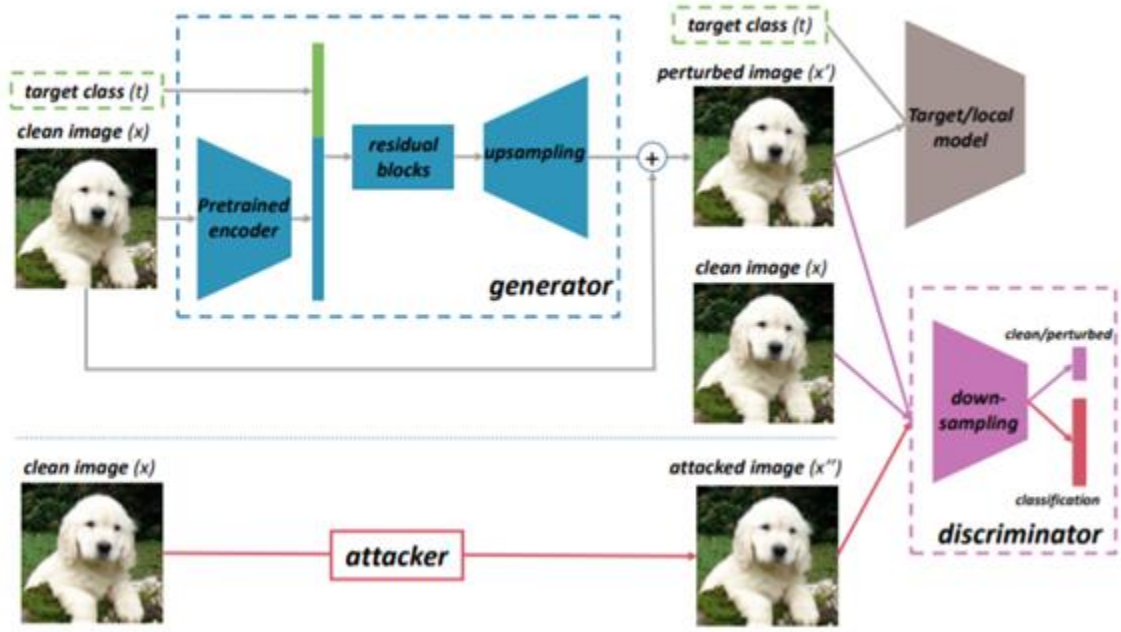
The success of this approach depends on choosing adequately-similar synthetic data samples and a substitute model architecture using high-level knowledge of the target classifier setup. As such, an intimate knowledge of the domain and the targeted model is likely to

aid the attacker. Even in the absence of specific expertise, the transferability property suggests that adversaries generated from a well-trained substitute model are likely to fool the targeted model as well.



Jacobian-based dataset augmentation works in the same way where a random sample of the initial data is taken and used to train a very poor substitute model. The adversarial examples are created from the dataset (using the gradient-based attacks from earlier).

Generative Adversarial Network (GAN) :



Generative modelling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from the original dataset.

GANs are a clever way of training a generative model by framing the problem as a supervised learning problem with two sub-models: the generator model that we train to generate new examples, and the discriminator model that tries to classify examples as either real (from the domain) or fake (generated). The two models are trained together in a zero-sum game, adversarial, until the discriminator model is fooled about half the time, meaning the generator model is generating plausible examples.

Generative adversarial networks are based on a game-theoretic scenario in which the generator network must compete against an adversary. The generator network directly produces samples. Its adversary, the discriminator network, attempts to distinguish between samples drawn from the training data and samples drawn from the generator.

Bibliography:

• https://arxiv.org/pdf/1312.6199.pdf

• http://s3.amazonaws.com/arena-attachments/607118/47d10e7cf5b482846aea40dcc3c58f4d.pdf?1462316194

• https://arxiv.org/pdf/1712.07107.pdf

• https://arxiv.org/pdf/1602.02697.pdf

• https://arxiv.org/pdf/1608.04644v2.pdf

• https://arxiv.org/pdf/1911.05268v2.pdf#page=85&zoom=100,102,233

• Adversarial Examples and Adversarial Training Stanford Lecture 16 cs231

• https://arxiv.org/pdf/1808.07945.pdf

• https://arxiv.org/pdf/2002.02196.pdf

• https://arxiv.org/pdf/1810.00069.pdf

• https://arxiv.org/pdf/1807.10454v3.pdf

• https://arxiv.org/pdf/1710.06081.pdf

• https://arxiv.org/pdf/1511.04599.pdf

• https://arxiv.org/pdf/1607.02533.pdf

• https://www.sciencedirect.com/science/article/pii/S209580991930503X