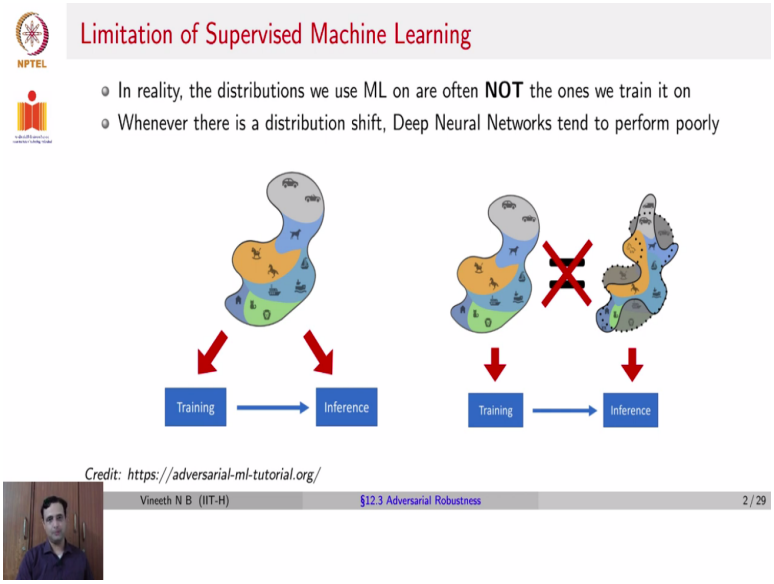
**Deep Learning for Computer Vision**
**Professor. Vineeth N Balasubramanian**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Hyderabad**
**Lecture No. 76**
**Adversarial Robustness**

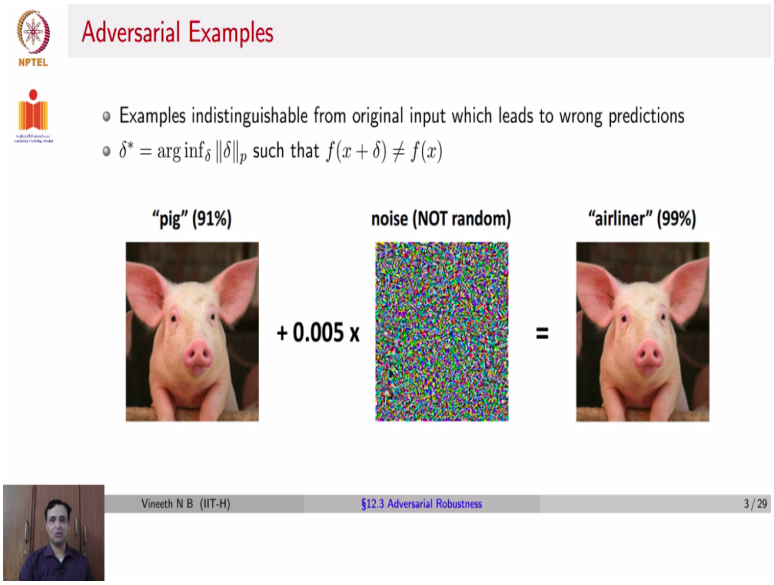
The next topic that we will discuss, again, a contemporary one is Adversarial Robustness.

(Refer Slide Time: 00:24)




In Supervised Machine Learning, often the distributions that we deploy machine learning models on that is the Test distribution are often not the distributions that you train the models on, unfortunately. Whenever there is such a distribution shift, then Machine Learning models, especially Deep Learning models suffer and tend to perform poorly.

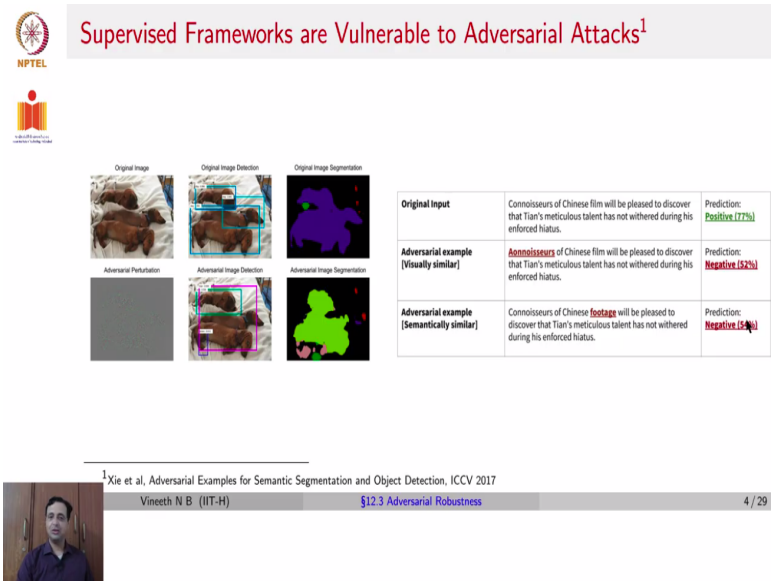
(Refer Slide Time: 00:53)




One such scenario where this distinction becomes even more prominent are what are known as Adversarial Examples. Adversarial Examples, this is different from the adversarial word in GANs, this has a different notion here. Adversarial examples are data points, which are indistinguishable from your dataset, but lead to wrong predictions. Here you see an example where an image of a pig, which a Deep Learning model said was a pig with 91 percent confidence. If you add a little bit of random noise, just 0.005 of random noise to this pig, the same Deep Learning model now calls this an airliner with 99 percent confidence.

This is a serious problem, especially considering that humans do not seem to suffer from this problem. And for few humans, the output image is still a pig, quite clearly. So, why do Deep Neural Networks suffer from this problem? Unfortunately, it is still not well known. But this question has led to a body of work over the last few years on what are known as adversarial attacks, which try to hack the model and disturb the model and adversarial defenses, which try to protect the model against adversarial attacks. Let us see both these kinds over the rest of this lecture.

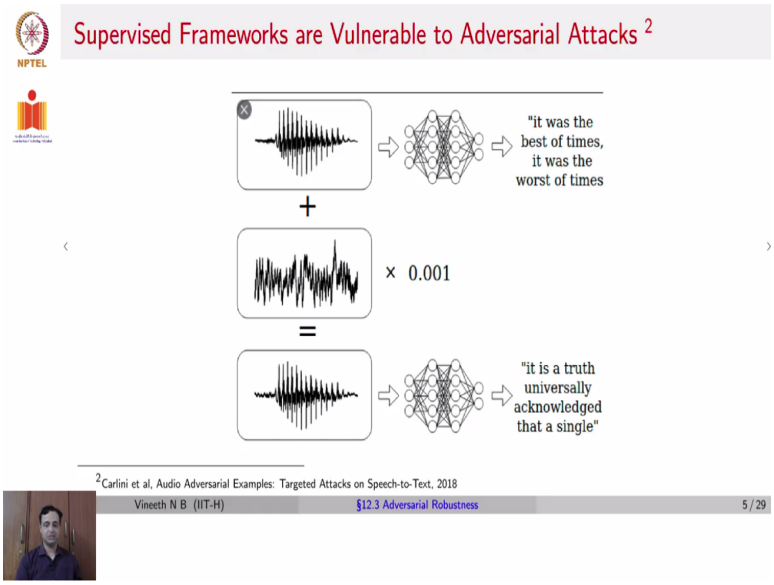
(Refer Slide Time: 02:36)




In general, it has been studied and understood now that supervised frameworks are vulnerable to adversarial attacks in any domain. Here is an example where you see a detection and segmentation task, given an image. The first detection image that you see here calls these dogs, and you also get a segmentation mask around them as dogs. But when a small adversarial perturbation is added to it to this image, now, this pink box or magenta box that you see here, is now a train with a fair good amount of confidence. And the masks too now reveal a different label for the same image.

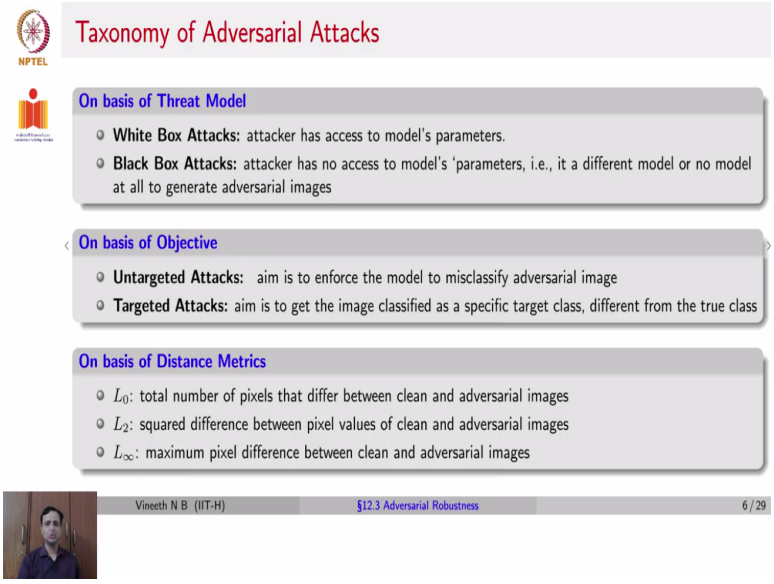
Similarly, for text, if there was an original input, which has a certain sentence, and the prediction is that the tone is positive, with a 77 percent confidence. If an adversarial example is a small change is made to the sentence, as you can see, the change here is very, very small. But now, the prediction is that the tone is negative with the 52 percent confidence. And if the word film is changed to footage, which is perhaps semantically similar, even then the prediction becomes negative with 54 percent. This is worrisome, again, considering that humans do not suffer from such problems.

(Refer Slide Time: 04:17)




Similarly, here is an example for an audio based network. Where given an audio signal, the neural network translates this as it was the best of times, it was the worst of times. And when a noisy signal multiplied by a small scalar constant is added to the signal, the same neural network now classifies this as it is a truth universally acknowledged that a single, the meaning completely changes.

(Refer Slide Time: 04:49)



So, now coming to what are adversarial attacks. Over the years, there have been methods that have been developed of the different kinds. And there are a few different ways of categorizing these methods. If one looked at the threat model, these attacks can be called White Box or Black Box Attacks. In White Box attacks, the attacker assumes access to the models parameters. Whereas, in Black Box attacks, the attacker does not have access to the models parameters. So, it is perhaps coming from a different model, or no model at all, to generate these Adversarial images.

Remember, adversarial attacks are intended to cause problems with the model. So you could look at that as equivalent to ethical hacking. Similarly, on the basis of objective, adversarial attacks are differentiated as Targeted Attacks, or Untargeted Attacks. In untargeted attacks, the aim of the attacker is to enforce the model to misclassify. Whereas, in a Targeted attack, the aim is to ensure that the model classifies it as a particular target label. So, in an Untargeted attack, if you had an image of a cat, or generally called a cat, you want to add a perturbation that ensures that the model no longer calls it a cat.

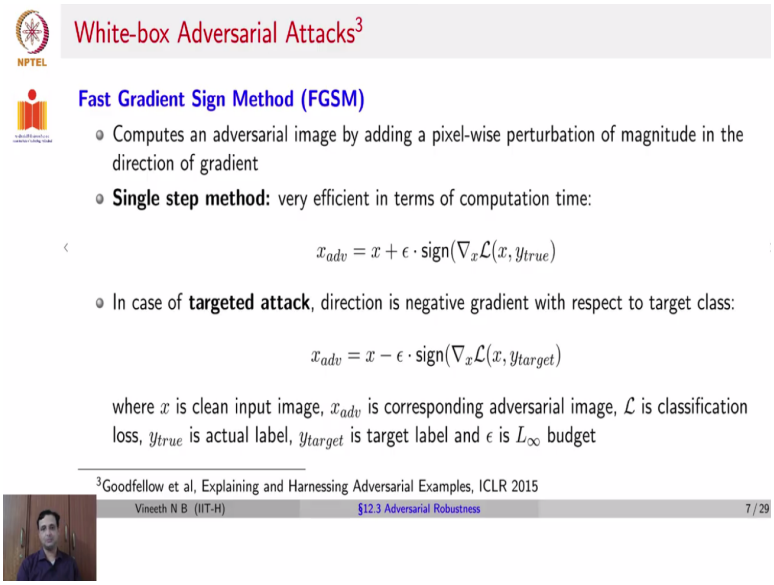
In a targeted attack, the attacker now wants to add a perturbation to ensure that the cat is now called a dog, for instance, could be any other target label, but you want a specific target label. Targeted attack could be a big problem in biometrics, where one person may want to add some noise to an image to impersonate as somebody else to a Deep Learning model trained for face recognition.

A third categorization is based on distance metrics used. So in all of these Adversarial perturbations, the constraint is that the perturbation should not have a very high norm, because you do not want the perturbation to be too large, because it is then always easy to add a very large number and change the outcome of a model. The challenge here is to ensure that the perturbation is as minimal as possible. And when you say as minimal, you have to measure a distance and that is done using different kinds of $L_p$ norms, such as $L_0$ norm, $L_2$ norm, or $L_\infty$ norm.

So, in $L_0$ norm, the total number of pixels that differ between clean and adversarial images is used, in $L_2$ norm it is the squared difference between the clean image and the adversarial,

adversarially perturbed image. And an $L_\infty$ norm it is the maximum pixel difference between the clean and adversarial images. An infinity norm is often used in many of the methods.

(Refer Slide Time: 08:10)




Let us now see a few methods that do what is known as White Box Adversarial Attacks, which means they assume that they have access to the models parameters to be able to attack the model. One of the prominent methods in this context is known as FGSM, or Fast Gradient Sign Method where the model computes an adversarial image by adding a pixel wise perturbation of magnitude in the direction of the gradient. So, in this single step method, you add a perturbation x is the original image, you take the gradient of the loss with respect to x, find the sign and add an epsilon in the direction of that sign.

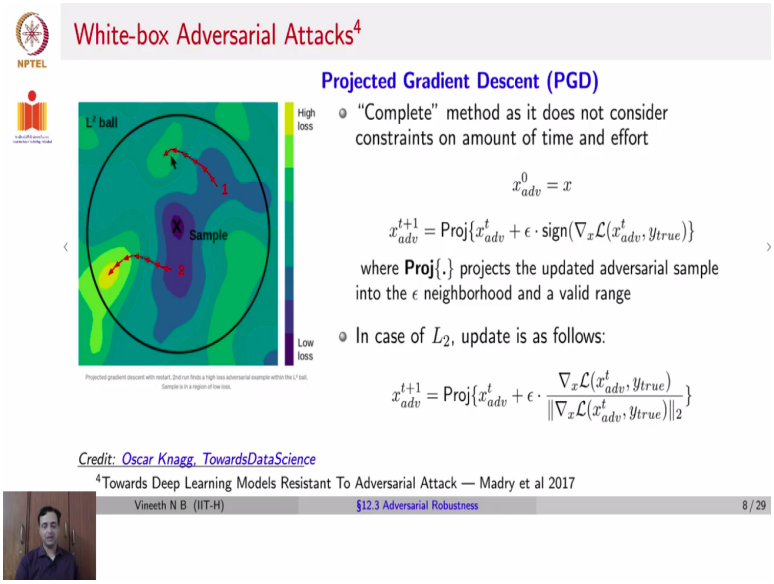
Remember that the gradient would actually be a vector with respect to each input dimension. So, for each of them, you would have a sign. So, for each dimension, you add an epsilon times its corresponding sign. What are you doing, you are now trying to add a perturbation, which would increase the loss and hence resulting in an adversarial perturbation. So, this is known as a single step method, just one update to get an Adversarial perturbation, and hence very efficient in terms of computation time to implement.

If this had to be done for a targeted attack, it is the same method. The only difference now is you now go in the direction of the negative of the gradient of x assuming that the gradient is

computed with respect to the target label, remember that when you do a negative, you are trying to reduce the loss with respect to the target label. So, you are now trying to make the model think that that sample is a different label.

So, x here is the image $x_{adv}$ is the adversarially perturbed sample, L is the classification loss, $y_{true}$ is the actual label, $y_{target}$ is the target label for a Targeted attack and $\epsilon$ is the budget or $L_\infty$ budget or any norm budget that you have, which you allow for the perturbation.

(Refer Slide Time: 10:47)



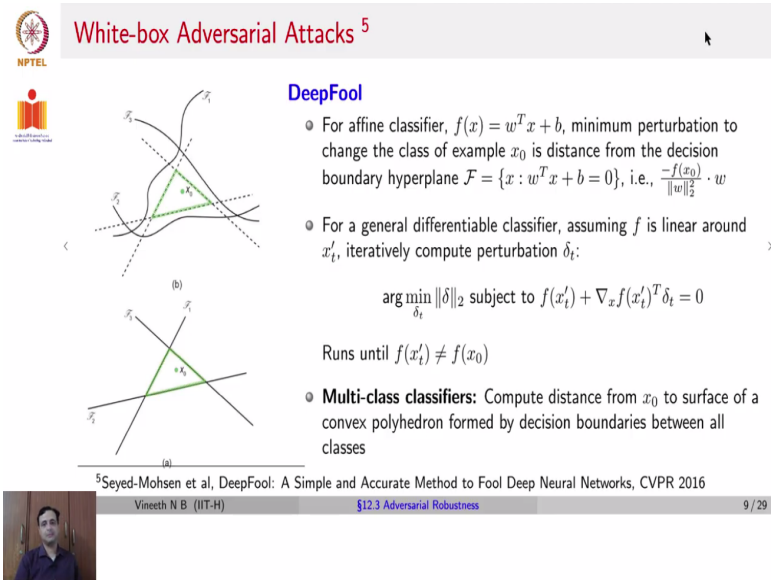

A more popular and a complete method is known as Projected Gradient Descent. This was introduced in 2017. This is considered one of the most popular methods today widely used. And it is considered a complete method, because it does not impose any constraints on amount of time or effort. So, you it is an iterative process, unlike FGSM, which is a single step method. PGD is an iterative method, which keeps improving the adversarial perturbation, unless until it succeeds.

How do you do this? You start your first iteration's adversarial sample with x, which is your given sample, then in each iteration, you add a gradient corresponding to that iteration's adversarial sample. And corresponding to the true label, remember, you are trying to increase the loss for the true label. And you know project the sample back into the $\epsilon$ neighborhood. So, you try to find out what is that sample that takes me to a higher loss.

Project it back to the ϵball that you are allowed your perturbation in remember that ϵ imposes a certain constraint on the size or the norm of the perturbation. This gives you a new adversarial perturbation. Now, you repeat this over multiple iterations, until the sign or the classification output changes. In case of $L_2$norm, the update becomes ϵ times you have the gradient of the loss divided by the two norm of the gradient of the loss. The sign simply becomes the gradient by the two norm of the gradient which is in other words, the sign.

The visualization here shows a loss surface. The loss surface here is that the sample is initially in a region of a very low loss. And by taking a series of iterative steps, the sample is now moved to a position where the loss is very high, remember, yellow here is high loss. And in two different runs, you now take the sample to two different locations, where the loss is very high.

(Refer Slide Time: 13:21)



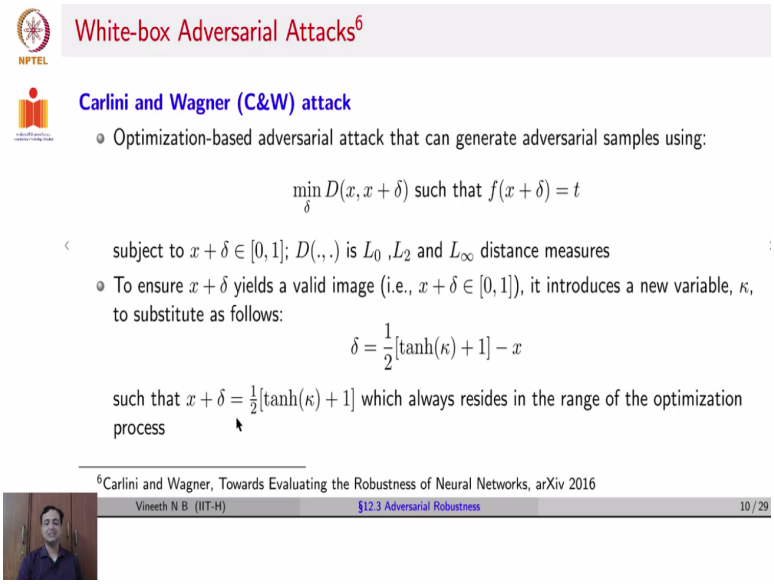

Another popular method, which was developed in CVPR of 2016, is known as DeepFool. In Deep Fool, the idea is considering the decision boundary, while coming up with the Adversarial perturbation. Given any affine classifier, a linear classifier, which is given by $f(x) = w^T x + b$. The minimum perturbation to change the class of an example $x_0$ is the distance to the hyperplane to the decision boundary, which is given by $w^T x + b = 0$.

And this distance between a point and the hyperplane is given by $\frac{-f(x_0)}{||w||_2^2}$, where $x_0$ is the point. This is a known quantity and this takes us from any point to the decision boundary when the class label changes, when that point goes beyond that decision boundary. In general, for a more general differentiable classifier, this method assumes that f, or the decision boundary is linear around that particular point.

And hence, it tries to compute a perturbation which minimizes the 2 norm of the perturbation subject to assuming that the perturbation is on a linear neighborhood local neighborhood of that point $x_t$ with respect to the decision boundary. But now, you keep running this iteratively until the decision on $x_t'$ changes from $x_0$. So, you find a delta such that it lies on a linear approximation of the decision boundary around that point in such a way that the 2 norm of that perturbation is as low as possible remember, you want your perturbation to be a very small quantity and you run this iteratively until the decision boundary or the class label changes.

What happens if you have a multi class classifier then, the distance is computed to the surface of a convex polyhedron formed by the decision boundaries between all classes. So, the distance from $x_0$ to any direction which is on the surface of a convex polyhedron is the minimum distance that you need to add or the minimum perturbation that you need to add to $x_0$ to change the class label into something else.

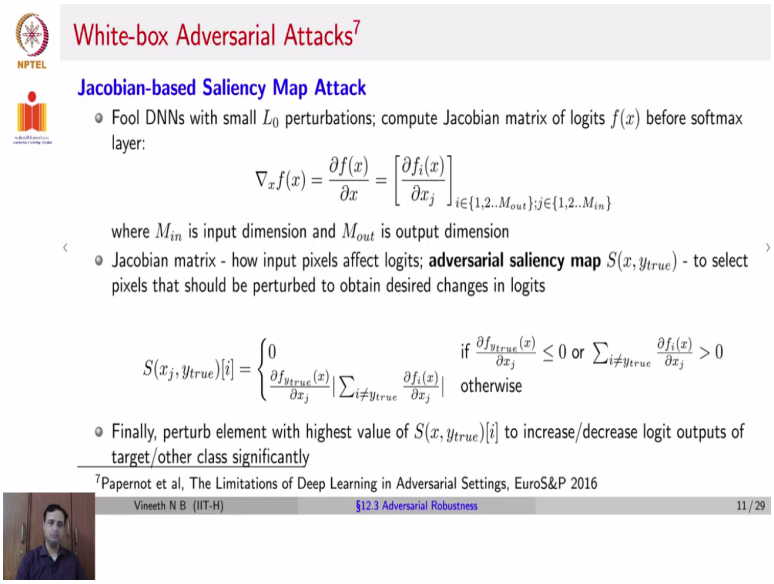
(Refer Slide Time: 16:07)




Another white box adversarial attack, which is also been used over the years is known as the CW attack or the Carlini and Wagner attack, which is an optimization based Adversarial attack again, but the problem now is formulated as $\min_{\delta} D(x, x + \delta)$ such that $f(x + \delta)$ is a particular target label or a label different from the true label. Subject to the constraint that $x + \delta$ also lies between 0 and 1. The D here can be $L_0$, $L_2$ or $L_\infty$ distance measures.

To ensure that $x + \delta$ yields a valid image that is $x + \delta$ should lie between 0 and 1. This method introduces a new variable κ which is used within this formulation for δ. δ is written as $\frac{1}{2}[tanh(\kappa) + 1] - x$. Why is this done? Because if we write δ this way, we get that $x + \delta$ will be $\frac{1}{2}[tanh(\kappa) + 1]$. We know that tanh has a range minus 1 to plus 1. So, hence $tanh + 1$ has a range 0 to 2 and dividing by half ensures that this value always lies between 0 and 1.

(Refer Slide Time: 17:47)



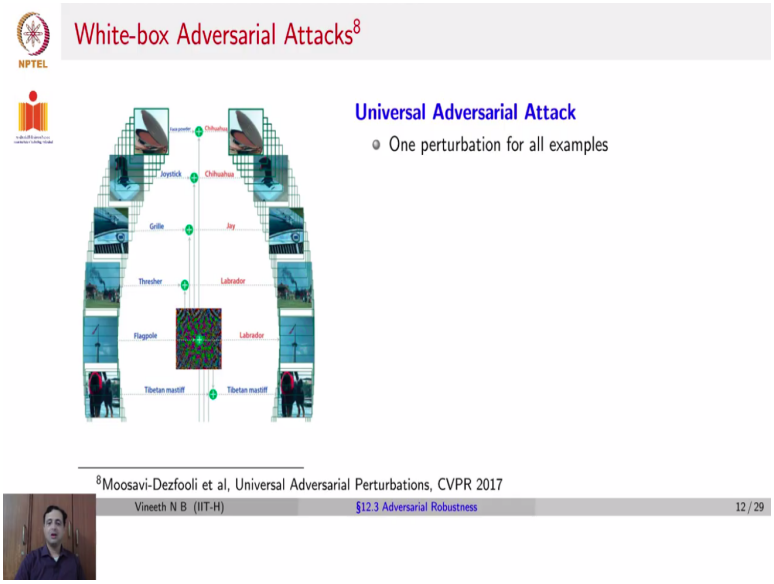

One more White Box Adversarial Attack is a Jacobian-based Saliency Map Attack. In this case, to fool the DNN with small $L_0$ perturbations, the method computes the Jacobian matrix of the logits before the softmax layer, remember, logits are the outputs of the neural network before the softmax layer. And Jacobian is computed between every output in that layer in the logits with respect to every input that gives you a matrix of different gradients.

Once you get the Jacobian matrix, you get an understanding of which input pixels affect which of the logits in particular, using this, the methods suggests the creation of an adversarial saliency map, which is given by this adversarial saliency map is given by 0 if the true gradient that is f with respect to that $x_j$ that you want to consider that input dimension. If that is less than or equal to 0, or the gradient with respect to non true labels, non ground truth labels is greater than 0. What does this mean?

This means that if the gradient becomes less than 0, which means we are further improving the output on the true label, this also means that we are going further away from the non true labels. We do not want that direction. So, in those directions, the saliency map is set to 0. In every other case, where the gradient of the true label with respect to $x_j$ is greater than 0, or the gradient of the other labels, the logits corresponding to the other labels with respect to the input is less than 0, we would like to encourage those directions to change the label.

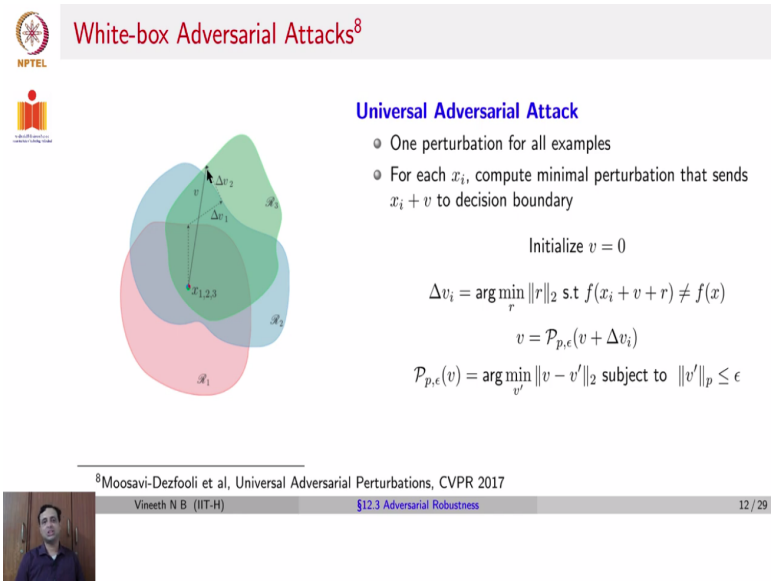
So, this method proposes the Adversarial Saliency Map to be the gradient of the true label into the absolute value of the sum of the gradients for the other classes. Why absolute value, because in the second case, we know that the other gradients will all be less than 0. So, we would like to use the absolute value. Once you get these adversarial saliency maps, we then perturb the element with the highest value of the adversarial saliency map to increase or decrease the logit outputs based on what you like, of the target, or the other class significantly.

(Refer Slide Time: 20:33)




The last White Box Adversarial Attack we will see is known as the Universal Adversarial Attack. The goal here is to find one perturbation for all your examples, if you observe carefully, so far, all the methods for adversarial attack, try to find a perturbation for a given sample that will fool the deep neural network model. But that can be laborious. So, the aim here is to see if we can find one perturbation that will fool them all.

(Refer Slide Time: 21:13)



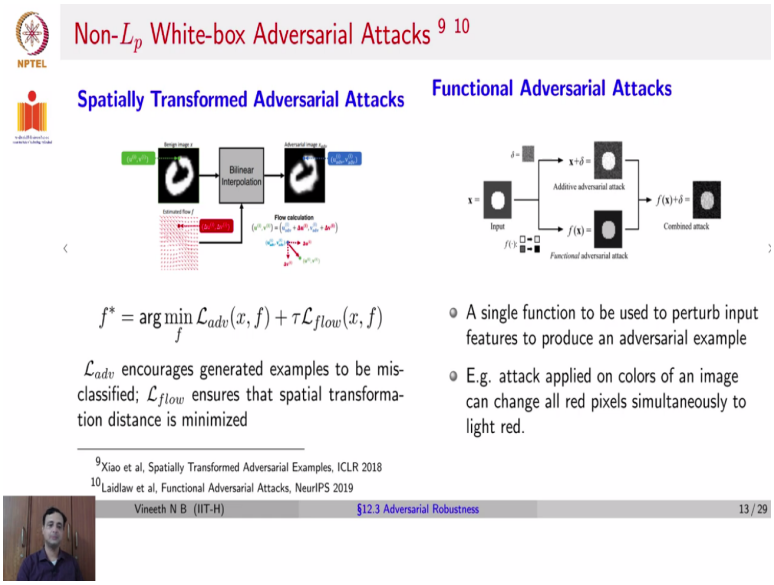

How do you do this? This is now done by if you consider this illustration here, $x_1$, $x_2$ and $x_3$ are juxtaposed, they are not located, they do not have the same vector value. This is just for convenience. $\Re_1$ here denotes the boundary corresponding to $x_1$ rather, that is the minimal perturbation, that is the ball or the set of values with the minimal perturbation, after which the class label for $x_1$ will change.

Similarly, you see an $\Re_2$ for $x_2$ and an $\Re_3$ for $x_3$, which are the regions where the label stays the same beyond which the label changes. With this knowledge, how do you learn the universal perturbation? The method initializes, the perturbation v to 0. And for every sample $x_i$, finds a $\backslash\Delta v_i$ that adds an r such that the decision changes. And then now projects v on to some ball, you try to ensure that if you found a $\Delta v_i$, then you try to ensure that the final perturbation for this sample is a v, which satisfies the ε constraint on the LP norm.

So, if you started with $x_1$, you would get this particular point as the point at which the label changes. Now when the next point comes, you realize that at that point, $x_2$ label does not change. So, you further add a Δ for $x_2$, then a Δ for the next point, so on and so forth. And v now is the

sum of all of these $\Delta V_i$'s, which is the total perturbation that you need to add for any of these points to change the class label.

(Refer Slide Time: 23:16)

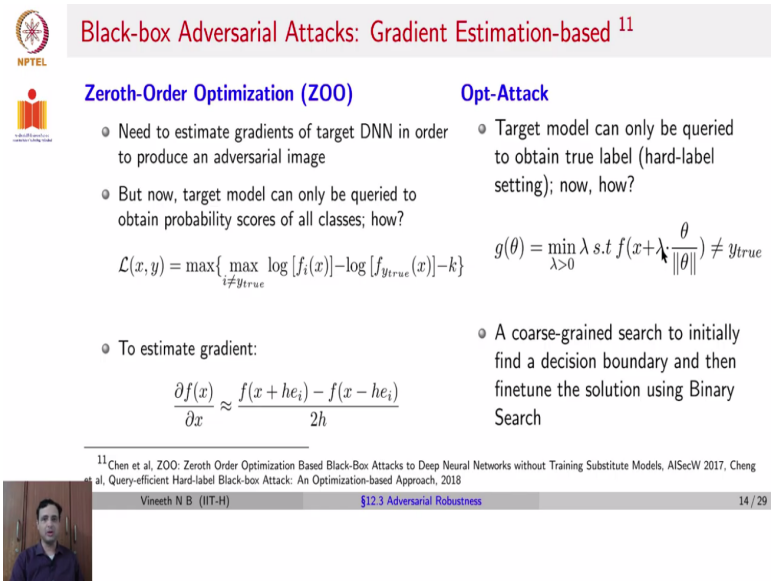



Now let us look at a couple of Non-Lp based White Box Adversarial Attacks, where you do not necessarily use the norms to create the attack, but use other kinds of methods. One of them is called the spatially transformed adversarial attack. In this method, given a benign input image, a clean image, the goal is to find a flow that means all pixels are moved by a certain quantity in such a way when that flow is added, and you perform a bilinear interpolation to ensure that you do not get positions that lie between two pixel locations, you get a final adversarial image.

Now, to learn this flow, the final objective function is given by $arg \min_f L_{adv}(x, f)$. Here, $L_{adv}$ encourages the generated samples to be misclassified and $L_{flow}$ tries to ensure that the flow perturbation is as minimal as possible. Another method in this category are known as functional adversarial attacks. So far, we looked at all attacks for perturbations, which are additive in nature, where you add a perturbation to a sample in this case, we now try to transform the sample using a function. Those are known as functional adversarial attacks.

And then you can combine the additive and the functional attack to get a combined attack. An example here is, you could now apply an attack to change all red pixels in an image to light red pixels. Remember, this is not additive, but an intensity scaling operation, which would be considered a functional attack.

(Refer Slide Time: 25:33)




Moving on from white-box attacks, we will now talk about a few Black-box attacks for adversarial perturbations. One of such examples, remember, in a Black-box attack, you do not have access to the model's parameters. But remember, from what we have seen so far, we need to be able to estimate the gradients of the target neural network to produce an adversarial image because that tells you a sense of the direction in which the loss or the output of the neural network will change.

But in this case, in Black-box, we do not have access to the model's parameters, we only have access to the outputs of the model. We assume that we have an the access to the logits of the target model, we still have to come up with an adversarial perturbation, which changes the output of the true label to some other class label by at least a certain distance K. So, this method called ZOO Zeroth Order Optimization, suggests that you can now approximate the gradient by doing multiple forward passes on the model.

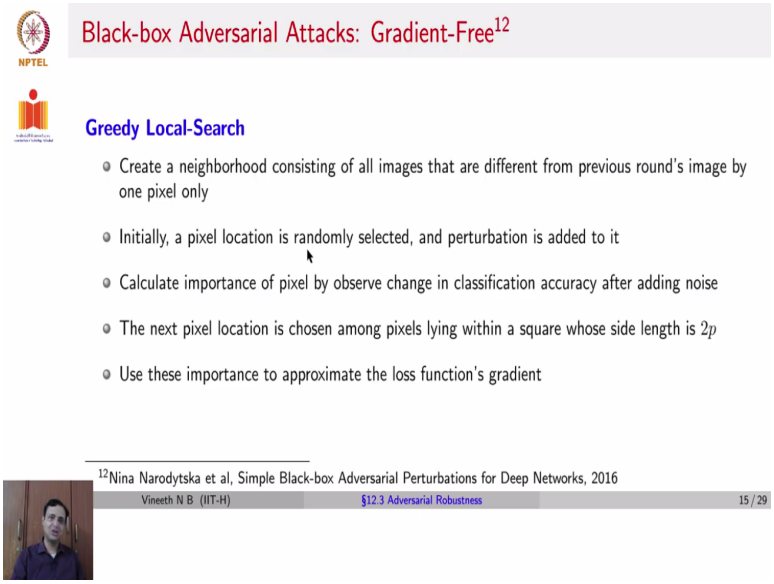
You do a forward pass of $\frac{f(x+he_i)-f(x-he_i)}{2h}$ where $e_i$ is a small perturbation. We know from first principles, that this is an approximation of the gradient. And I could now use this as an estimate of the gradient and then do any other any method similar to one of the White-box attacks. Another approach in this direction is called an Opt-Attack, where the target model here can only

be queried to get the hard label, not even the logits. Remember, in this approach, in the ZOO approach, we said you could get the probability scores for the logits of the model.

Now we say even that is not available, you can only get the winning label. So, the perturbation that we need here, $g(\theta)$ is the minimum $\lambda$ such that $f(x + \lambda \cdot \frac{\theta}{||\theta||}) \neq y_{true}$. But how do we do this when we do not have access to the gradients, and we only have access to the final prediction of the model? So, one has to rely on a brute force kind of an approach, where a coarse grained search is finally performed is initially performed to find a decision boundary and then this is fine-tuned using binary search.

So you try to see how much do I add to cross the decision boundary. And if you crossed it by a lot, now within that perturbation, do a binary search to find the exact perturbation that makes you cross the decision boundary. And you do have the output of the target model to check whether you crossed the decision boundary or not.

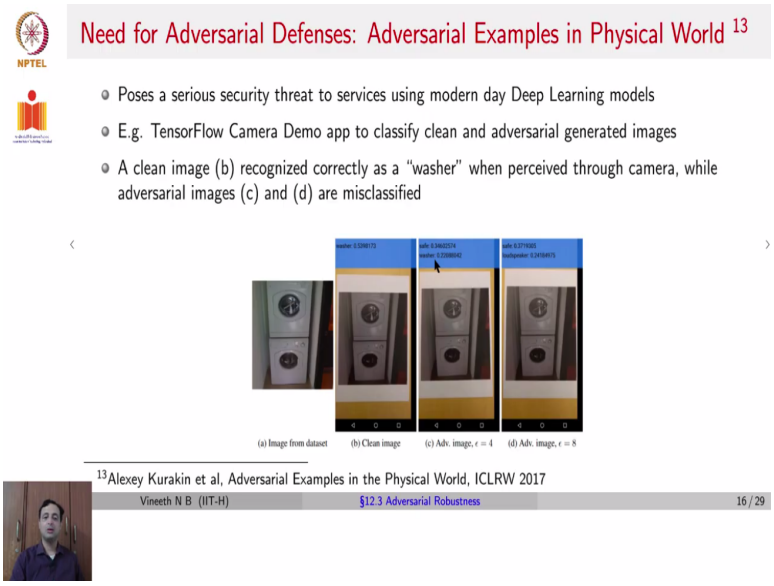
(Refer Slide Time: 28:51)




Another Black-box Adversarial Attack, which is a gradient free attack, where there is absolutely no gradients is you create a neighborhood consisting of all images that are different from the previous round's image by just 1 pixel. You initially pick a random pixel and add a perturbation, you can now calculate the importance of that pixel by observing the change in classification accuracy after adding noise.

Now, you choose the next pixel location among pixels lying within a square whose side length is 2p, you stay in that neighborhood. You again find the importance of each of those pixels in that neighborhood. And this can give you an approximation of the loss functions gradient, which you then use to attack the model.

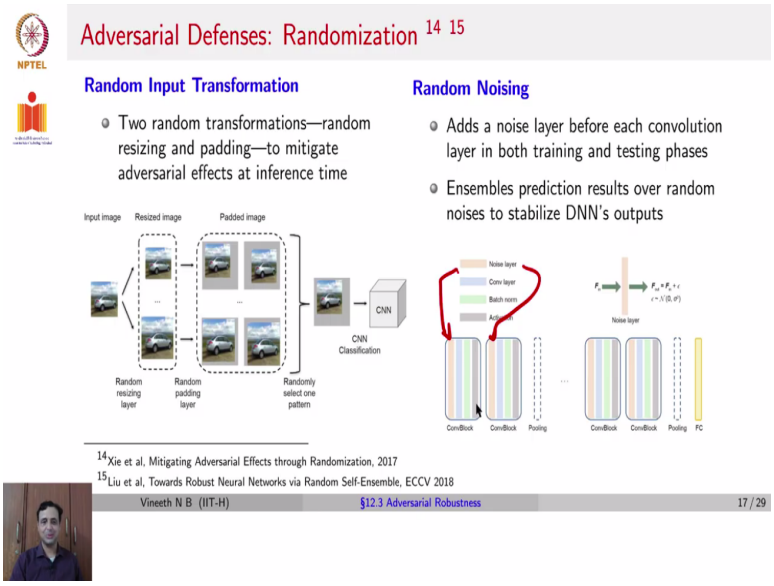
(Refer Slide Time: 29:53)




We now come to the other side of the story. So far, we have seen several methods that perform adversarial attacks. Now we will talk about a few methods that try to defend Deep Neural Networks against adversarial attacks. Why do we need this adversarial attacks pose a serious security threat to services that use modern Deep Learning models. Here is an example, which was performed using a TensorFlow Camera Demo app to classify clean and Adversarial Images.

Given an image from the data set, initially, the model calls this a washer. And when an adversarial attack, or a perturbation is added to the image, now the model thinks this is a safe, which it is clearly not.

(Refer Slide Time: 30:52)

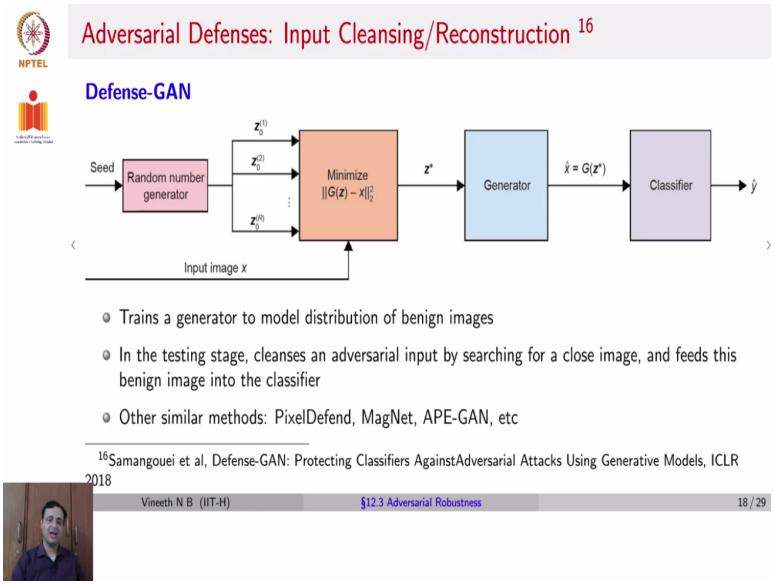



So in the physical world, like applications in Autonomous Navigation, an Adversarial attack can cause lives, and it becomes important to defend against them. One method to perform an adversarial defense is called as Random Input Transformation. So in this case, given an input image at test time, the approach is to firstly, randomly resize the layer and then randomly pad the image in different ways. And pick one of these outputs to give to the CNN model and classify. Why are we doing this?

We expect that irrespective of where the object is an, is in an image, the CNN will be able to classify it, that is the first part. The second part is we are hoping that by choosing this image randomly, the model or the adversarial attack may not pick a pixel in this part of the image, where the image is actually located in this part of the canvas, where the actual image is actually located.

Another approach is called Random Noising, where a noise layer is added before each convolutional layer in both training and testing phases. You can see here that the convolutional layer block, which has a convolutional layer, a batch norm layer and an activation layer. Now, you also have a noise layer added to each of your convolutional blocks. What is the purpose? Both at training and testing, you add multiple random noises, and ensemble your prediction results across all of these randomly perturb inputs. And this gives you robustness against adversarial attacks, at least to an extent.

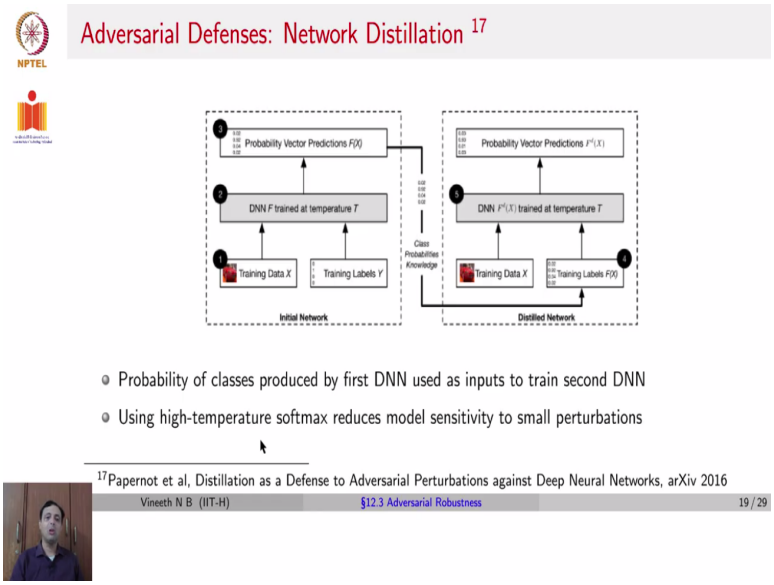
(Refer Slide Time: 32:56)




Another defense method is known as Defense GAN, where a generator is trained first, to model the distribution of clean images. So, you have your training dataset, you do not consider any adversarial perturbations, you only train a GAN, to generate more images, such as your training dataset. At test time, you now cleanse an adversarial input by finding the nearest generated image. How do you do that? You then you have a random number generator, you then try to find out different images obtained through different random noise vectors to the generator.

You try to see which of them of which of those generated images is closest to the current input, which could be adversarial. And now use that as a classifier to finally use that as input to your classifier to get the output. Why do we do this? We hope now that any adversarial perturbation may now get hidden by considering the closest image and that pixel value that have been that may have been added to a specific pixel is now offset by considering a different image. There are other methods in the same space that operate with a similar principle called Pixel Defend, Magnet, APE-GAN, so on and so forth.

(Refer Slide Time: 34:33)

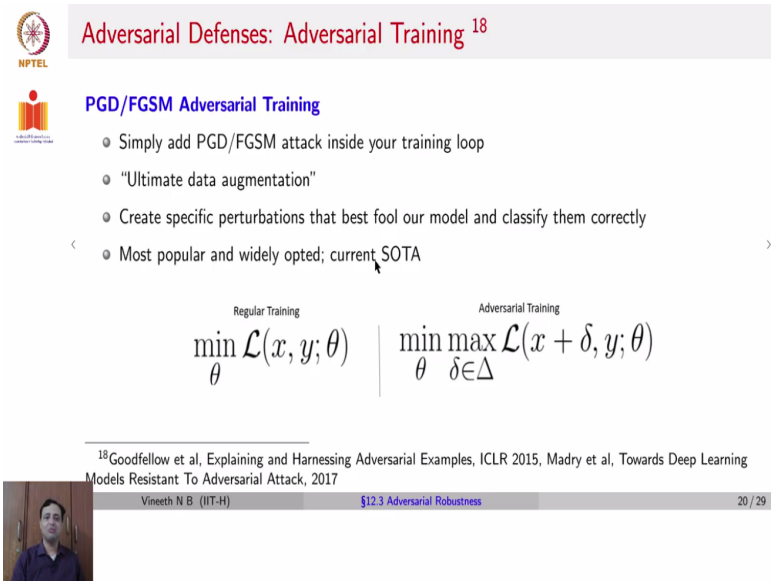



Another defense is based on an approach called Network Distillation. We will see the idea of distillation more closely in the next lecture. But we will give a high level idea here. The high level idea here is you have an initial neural network, which is trained given training data and training labels. And this network outputs a set of a vector of logits or a softmax probability distribution as its output.

Now, the distilled network learns using these probability vector predictions instead of the original one hot training labels. So, if your initial network had a distribution over, say 10 labels, instead of recognizing a digit as 3, it would have a probability distribution over all labels. The second network is asked to predict that probability distribution, rather than predict the digit 3. How does this help?

Using a very high temperature softmax remember, if you use a high temperature softmax, you get a more smoother distribution in your softmax of values, the probability values, that reduces the model sensitivity to small perturbations. So, you do not easily assign another label because you have smoothened out your probability distribution across all labels.

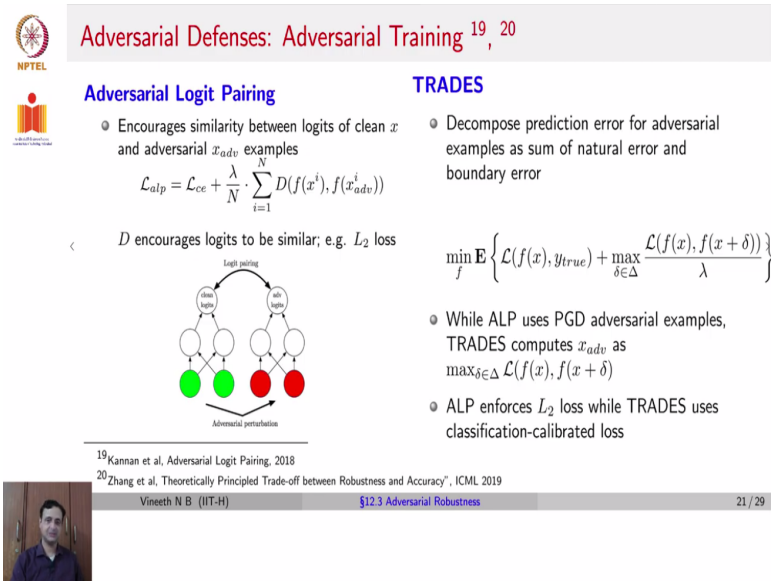
(Refer Slide Time: 36:11)




But the most important and most widely used adversarial defense is known as Adversarial Training. In adversarial training, which is a computationally intensive process. You simply add a PGD or an FGSM attack as part of your original training loop. This is considered the ultimate data augmentation step. So, in each step of your training of the original Deep Neural Network model you add an inner maximization step, which tries to find the δ that maximizes the loss, and then you find the θ that minimizes the loss for this adversarial perturbation.

So, within each loop or iteration of training, you have to solve for each data point, a maximization problem, which says, given an input sample in one mini-batch iteration of SGD. How much should I perturb this to cause maximum damage? You find that perturbation, call that δ, now you add δ to that input, and then minimize the loss over the overall network. This is obviously the most powerful adversarial defense, it is widely opted, and the current state of the art for adversarial training.

(Refer Slide Time: 37:48)

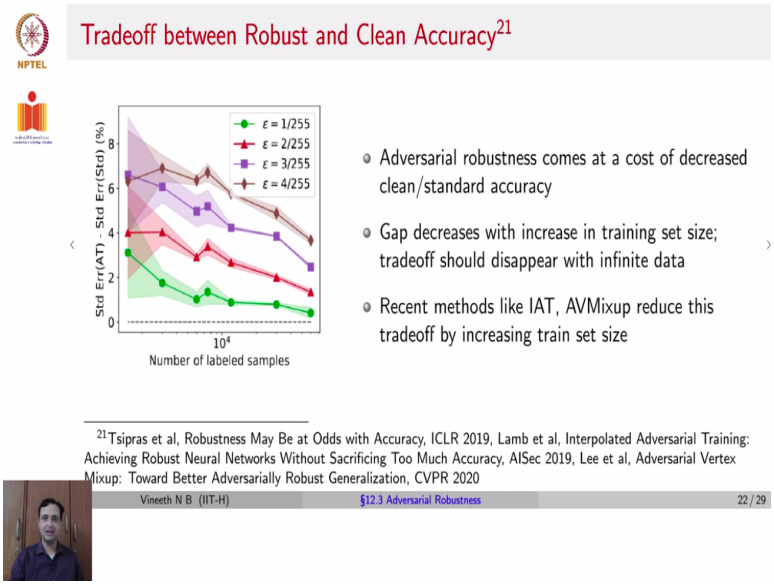



There have been a few variations of adversarial training based on similar ideas. One of them is known as Adversarial Logit Pairing, where you have a clean sample, which is given by this green vector. You perturb it and you get this red vector. You pass both of them through the Neural Network, you try to ensure that the logits of the clean sample and the adversarial sample are close to each other using something like $L_2$ loss. This also helps as a form of adversarial training.

A more recent method called TRADES, which was published in ICML of 2019. Uses a similar idea, but a different method. It decomposes the prediction error as the sum of natural error and boundary error. So, you have a natural error here, which is the error that you typically deal with when you train any Neural Network model. And then you have a boundary error here, which is trying to find the delta that will push the decision function to give a different label. That is the second part here, which has another maximization here, which is why it is an adversarial training step.

While ALP which is Adversarial Logit Pairing, uses PGD Adversarial samples. TRADES computes adversarial samples as maximum of delta where the decision between $f(x)$ and $f(x + \delta)$ is maximized. So, you want that loss to be as high as possible, not necessarily a PGD attack. So, you can look at ALP as enforcing $L_2$ loss, while TRADES uses a classification calibrated loss.

(Refer Slide Time: 39:57)



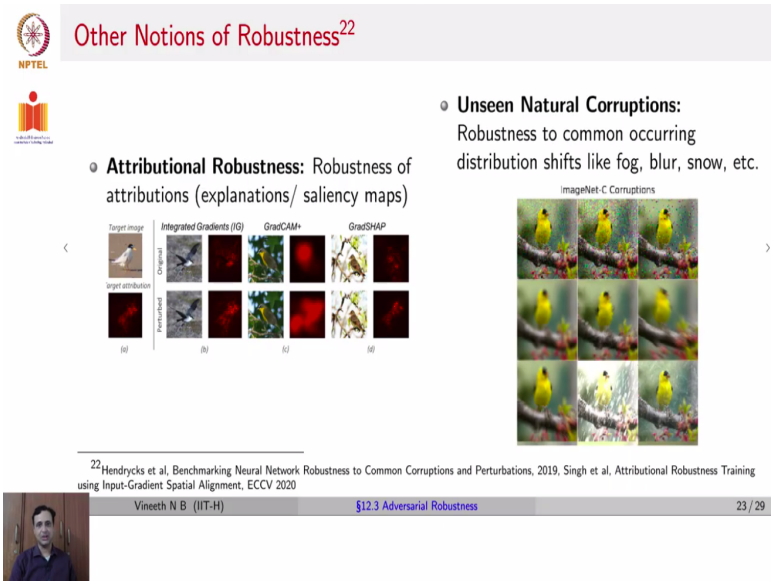

An important take away from the entire studies around adversarial robustness is that one has to deal with a tradeoff between robust and clean accuracy. Unfortunately, adversarial robustness comes at a cost of decreased clean or standard accuracy. Here is an example where, as you see here, that on the y axis, you see the difference between adversarial error and the standard error, you can see that when the attack increases 4 by 255, is a very powerful attack, because you are allowing the perturbation to be larger than 3 by 255, or 2 by 255, or 1 by 255.

You can see here that when the attack is higher, the difference between adversarial error and standard error goes up. However, this graph also gives us some consolation that as the number of labeled samples increases, these differences seem to get reduced over time. So as you keep increasing the number of training samples, it looks like even if you had a stronger Adversarial attack, the adversarial error and the standard error, the difference is not too much.

So more recent methods, such as Interpolated Adversarial Training, or Adversarial Vertex Mixup, try to reduce this tradeoff by increasing the training set size using interpolations between your standard data and an adversarial sample. You could consider now taking using adversarial training to find take a data point and input data point find its adversarial perturbation in the inner maximization loop of adversarial training.

Now take interpolations of all the points that lie on the line between the clean point and the adversarial perturbation. All of these, when added to training, help improve the train set size, and thus mitigate this trade off problem between adversarial error and standard error.

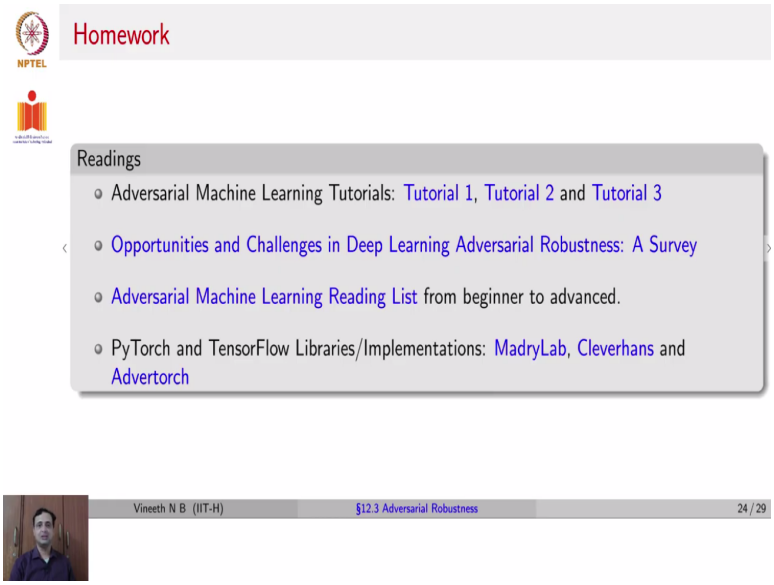
(Refer Slide Time: 42:21)




Over the last year, there have also been other notions of robustness that have surfaced, such as attributional robustness, where methods attack explanations and saliency maps instead of attacking predictions. So, you have an image, you perturb it in such a way that the class label output is the same, but the explanation significantly changes. There is also the notion of robustness to natural corruptions, such as say fog, a blur, or snow, so on and so forth.

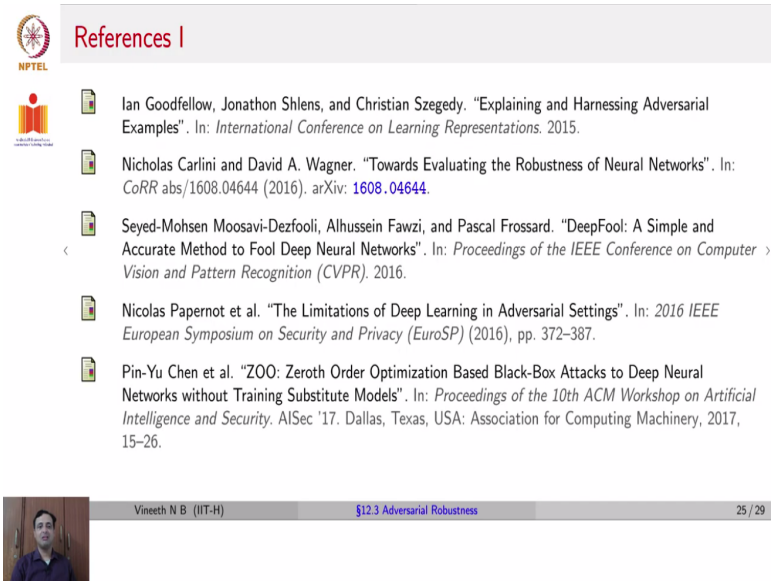
Which again, an example could be autonomous navigation, where when you train a model to drive on a normal road, you should be able to drive in the same scene, even if there is rain or snow in the same setting. Robustness against natural corruptions, also becomes important in such scenarios.

(Refer Slide Time: 43:26)




Your homework for this lecture is to go through these excellent tutorials on Adversarial Machine Learning, Tutorial 1, to, Part 1, Part 2 and Part 3. And if you are interested, you can read further on these links here. The space of Adversarial Robustness is quite vast today, but these links give you a fair picture of an understanding and the links to a few codebases to try to understand how these work in practice are also provided here for your experimentation.

(Refer Slide Time: 44:01)



## References II


Alexey Kurakin, Ian Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *ICLR Workshop* (2017).

Seyed-Mohsen Moosavi-Dezfooli et al. "Universal Adversarial Perturbations". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

Harini Kannan, Alexey Kurakin, and Ian J. Goodfellow. "Adversarial Logit Pairing". In: *CoRR* abs/1803.06373 (2018). arXiv: 1803.06373.

Pouya Samangouei, Maya Kabkab, and Rama Chellappa. "Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models". In: *International Conference on Learning Representations*. 2018.

Chaowei Xiao et al. "Spatially Transformed Adversarial Examples". In: *International Conference on Learning Representations*. 2018.

Dan Hendrycks and Thomas G. Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *CoRR* abs/1903.12261 (2019). arXiv: 1903.12261.




## References III

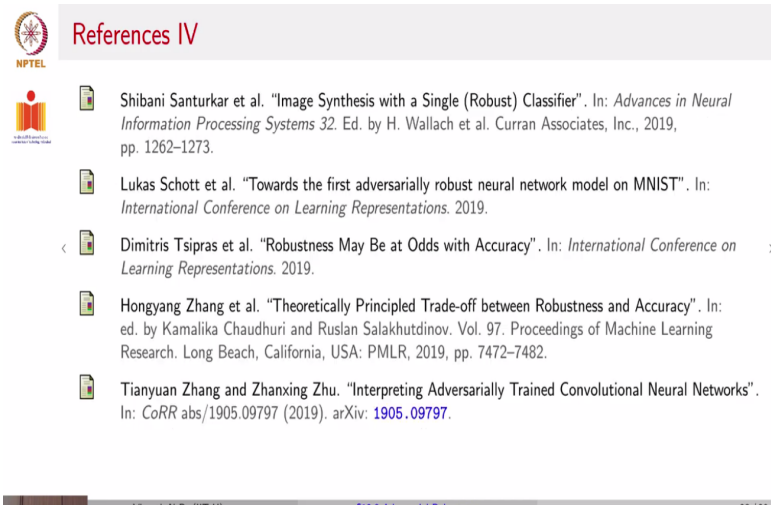

Andrew Ilyas et al. "Adversarial Examples Are Not Bugs, They Are Features". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 125–136.

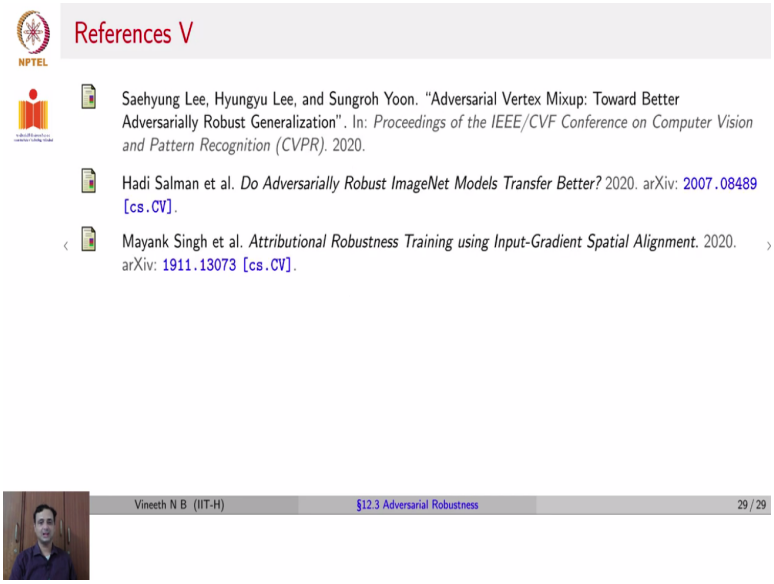
Cassidy Laidlaw and Soheil Feizi. "Functional Adversarial Attacks". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 10408–10418.

Alex Lamb et al. "Interpolated Adversarial Training: Achieving Robust Neural Networks Without Sacrificing Too Much Accuracy". In: *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. AISec'19. London, United Kingdom: Association for Computing Machinery, 2019, 95–103.

Seyed-Mohsen Moosavi-Dezfooli et al. "Robustness via Curvature Regularization, and Vice Versa". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

Chongli Qin et al. "Adversarial Robustness through Local Linearization". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 13847–13856.


Here are a very comprehensive set of references if you would like to follow through further.