CS6220 Technology Review: Adversarial Attacks on Neural Networks Trained for Image Classification/Recognition

Anthony Wong
Georgia Institute of Technology
College of Computing
awong307@gatech.edu

1. Introduction

In the burgeoning era of artificial intelligence and machine learning, many new problem domains where deep learning can be of tremendous use have continued to evolve. One of the most infamous of these problems is image classification/image recognition. Neural networks have revolutionized this topic, achieving unprecedented accuracy in discerning and categorizing visual information. However, as in many technologies, the incredible gains in the field also come with concerns, namely the vulnerability of these networks to adversarial attacks. These attacks often consist of strategic manipulation of input data, imperceptible to the human eye but capable of misleading sophisticated neural networks, leading to potentially catastrophic misclassification. As this technology continues to become more and more prevalent throughout various industries and in increasingly sensitive and precarious environments, it is imperative that we continue to discover the ways we can exploit existing networks and find ways to eliminate these vulnerabilities. This paper explores the landscape of adversarial attacks on image classification/recognition networks, aiming to provide a comprehensive understanding of their nature, histories, methodologies, and implications. Papers exist that are themselves surveys of many adversarial attacks in the past [1] [3]. Though this paper covers some topics defined in those papers, this is geared toward a different type of audience. This aims to be digestible for a general audience that is aware of image classification/recognition networks, deep/shallow neural network architecture in general, and how these networks function on a general level but is unaware of adversarial attacks in this domain and seeks to understand conceptually how they operate on a medium to high level.

2. Adversarial Attacks

Here, I review a number of methods of adversarial attacks on image classification networks to induce misclassification, how they work, and the background behind them. Note that this is not necessarily an exhaustive list of methods but will only cover papers as defined in the references section.

As a preface to future discussions, I think it is useful to give context to this problem domain. Papernot et al. [13] introduce an interesting matrix to do just this:

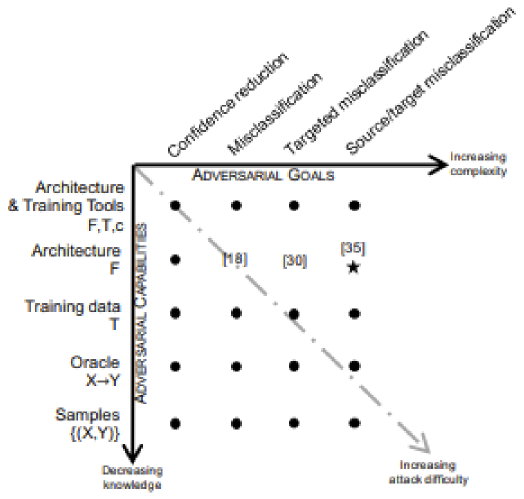


Figure 1. Threat Model Taxonomy as per Papernot et al. [13]

We see from the matrix that there are varying types of attacks ranging from simply reducing classification confidence to getting a specific source class to misclassify as some target class. This, clearly, marks an increase in complexity. We also see that we might have varying amounts of information about the system, from knowing everything like trainsets, model architectures, and cost functions, to simply only knowing what images you put in and what labels come out. All of these adversarial capabilities and goals are useful in their own way and add to the body of knowledge with which we ideally can start to mitigate potential attacks on these types of networks.

2.1. White-Box Attacks

There are multiple types of adversarial attacks. One of these is the white-box attack where the attacker has full access to the architecture, parameters, and training data of the target model. Essentially, the attacker knows everything about the inner workings of the model. Though the information needed may vary for the below methods they need some amount of information such that these examples of attacks are classified as such.

2.1.1 Box-Constrained L-BFGS

In a very influential and groundbreaking paper, Szegedy et al. [20] noticed a couple of interesting things about deep neural networks trained for both speech and visual recognition tasks which they coin as intriguing properties. Firstly, they challenge the idea that the individual neurons in the final feature layer of a network and proper bases for extracting semantic information. More specifically, they contest the idea that individual neurons are mapped to some distinct characteristic or pattern in the image data, which collectively come together to contribute to some network's ability. Instead, they believe the entire space of activations is necessary to recognize semantic detail. The second observation they make has to do with perturbations, which in the context of images involves small, controlled modifications to the input data. They find that though we expect models/networks to be robust and invariant to small perturbations to input images, this is not necessarily the case, as even imperceptible modifications to input images can vastly change the prediction models make. This also challenges the smoothness assumption, that small changes in the input space result in small changes in the output space. It is these observations that set the basis for this form of adversarial attack.

The paper goes on to mention the key exploitation used, which involves ”pockets” in the high-dimensional data manifold (structure or distribution of the data in the high-dimensional space), which can be considered as some sort of blind spot. These areas describe certain regions of the high-dimensional input space that happen to be extremely sensitive to perturbations. There may be different reasons that these blind spots arise but the paper notes the idea of non-local generalization, suggesting that neural networks learn to assign probabilities to regions of the input space even when there are no training examples nearby. This is stated to result from the non-linearity between the input and the eventual output due to the stacking of many non-linear layers in between them presumably as per network architecture.

To find some perturbation that results in misclassification. On a high level, they introduce the following optimization problem.

$$min\|r\|_2 \text{ subject to } f(x+r) = l \text{ and } x+r \in [0,1]^m$$

In simpler terms, we want to find the smallest perturbation r such that the clean image plus the perturbation classifies as some class l that is not the correct class for the image. Note that f can be defined as the network classifier. Since this problem is defined as hard, they use Box-Constrained L-BFGS [5], an optimization technique, to approximate this. Despite this approximation, it should be noted that this process is extremely computationally intensive.

With this process, they give the following adversarial examples:

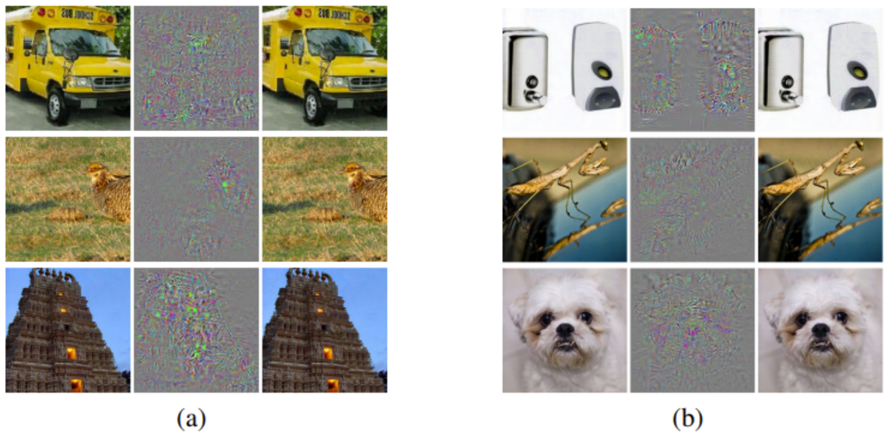


Figure 2. Box-Constrained L-BFGS Adversarial Examples on AlexNet [10]

The left-side images are the original, the center images are the magnified perturbations, and the right side are the images after the perturbations are added. The images on the right side, as a result of the perturbations, were classified as Ostrich, Struthio, and Camelus respectively, which are obviously incorrect given the original images. This means that these small changes to the original images that are imperceptible did in fact result in misclassification, meaning the adversarial attack was successful. These observations and experiments laid the groundwork for further research in this area.

2.1.2 Fast Gradient Sign Method

Goodfellow et al. [6] sought to build upon what findings were produced in the previous section regarding the paper from Szegedy et al. [20]. As stated previously in Box-Constrained L-BFGS, it was theorized that we have blind spots in classification networks due to non-local generalization caused by non-linearity between the input and output. However, this paper notes that these fixations and theories regarding the non-linearity issues are not necessary. They find that even in situations of linearity in high-dimensional spaces, we can still find these same adversarial examples, still utilizing perturbations to exploit and cause misclassification for image classification and recognition networks. They additionally note that dropout, pretraining, and model averaging, which are regularization techniques, do not help a network become robust and resistant to adversarial attacks.

The paper goes deep into theoretical mathematics but can be generally explained using a number of their general findings. First, they talk about the limitations of individual input features. Because there is a limit to the number of bits that can represent a pixel, there are certain values that are deemed insignificant and discarded, which in the paper they describe as numbers below 1/255 of the dynamic range which may simply be rounded out. Therefore, for any perturbation applied to some image sample, we would expect that if the perturbation is less mathematically than the threshold such that the system discards it, the resulting adversarial sample should not be classified any different from the original sample. Given we are discussing this in a linear context, the paper gives the following linear activation formula with the weight vector w and the adversarial sample $\tilde{x}$:

$$w^T\tilde{x} = w^T x + w^T \eta$$

Here we see that the activation grows linearly as a result of the perturbation η. There is more math behind this but this is the key observation. Even if your perturbation is incredibly insignificant, as discussed previously, it still has an impact on the resulting activation. In the paper, they use the variables n and m to represent the dimensionality of w and the average magnitude of the weight vector. We can see, due to the dot product of the transposed weight vector and our perturbation, that when those values are higher, so is the effect that the perturbation has, so much so that we can morph the output.

The final equation used to calculate these perturbations is given by the following formula. Note that ϵ is the magnitude of the smallest bit that can be encoded into an image, θ are the model parameters, x is the input, y is the output, η is the perturbation, and $J(\theta, x, y)$ is the cost function on which the model was trained.

$$\eta = \epsilon * sign(\nabla_x J(\theta, x, y))$$

In comparison, this equation is much easier to solve and is mentioned to be indicative that their hypothesis regarding adversarial examples resulting from linearity is true. They utilized this method to find adversarial samples using GoogLeNet [19] trained on ImageNet [4]:

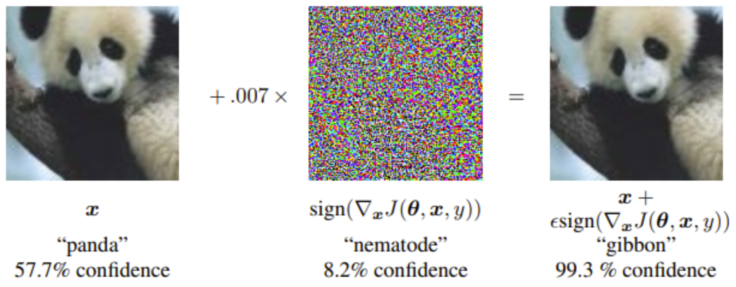

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 3. Fast Gradient Sign Method Adversarial Example

Here we see how these components defined in the equation come together to produce a misclassified sample of surprisingly high confidence. In the end, the paper produces numerous takeaways, some of the most notable being that:

1. Adversarial examples come from models being too linear rather than too non-linear
2. These attacks can be generalized to many models due to exploits being tied to weight vectors in a model
3. Small changes along a specific direction in the input space can lead to adversarial examples even if the perturbation doesn't occur at a particular point in space
4. Training your model on generated adversarial examples can help in reducing model failure rate in these scenarios

2.1.3 Jacobian-Based Method

In 2016, Papernot et al. [13] produced an iterative method of finding minimal perturbations that result in source/target misclassification, meaning we can take a specific example of a source class and find the perturbation that results in the model classifying it as a specific target class. They first produce a very easily digestible example to justify their intuition. This was in the form of a simple neural network being taught the boolean AND function. Similarly to other previous methods, we are trying to solve the same essential formula:

$$\underset{\delta_x}{argmin} \|\delta_x\| \ s.t. \ F(X + \delta_x) = Y^*$$

where conceptually we are still finding a minimal perturbation that results in a label that is just incorrect (given this is a binary example). Their method for doing this involves the calculation of the forward derivative, which is a Jacobian matrix. Note that a Jacobian matrix is a matrix of partial derivatives that describes the rate at which a function changes with respect to how its input components change. In this particular example, with X being the input (consisting of two variables for the AND), x_1/x_2 being the two components of the AND, and F being the network-learned function, the Jacobian formula looks as follows:

$$J_F(X) = \left[\frac{\partial F(X)}{\partial x_1}, \frac{\partial F(X)}{\partial x_2}\right]$$

With this formula, they produce a couple of very informative visuals.

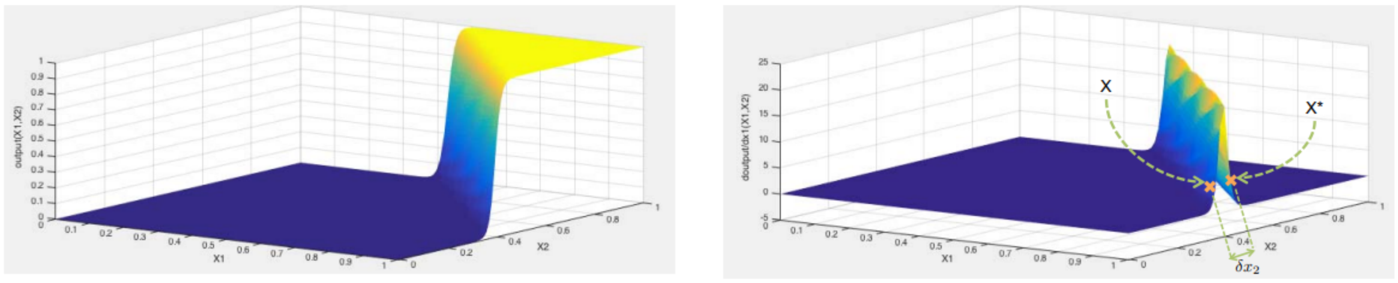


Figure 4. Left: Output landscape of the AND neural network, Right: Forward derivative of the AND neural network with respect to x_2 (X is the sample and X^* is the adversarial sample)

Note that the key observation comes from the right image from Figure 4, which shows how the forward derivative landscape gives us information on regions where we should search for adversarial samples. In areas of low forward derivative, we can say that area is likely an area that would not produce the samples we want for misclassification. The opposite is true for areas of a large forward derivative. The insights they create from this are the following, with the first being something we've extensively covered:

1. Small input changes can result in large output changes
2. Only some areas of the input domain are conducive to generating adversarial samples
3. The forward derivative narrows down the general adversarial sample search space

We can now expand this to more complicated scenarios. For this, let's now define two more important variables: Υ which is the maximum allowed distortion (since we want adversarial samples to be indistinguishable), and θ which represents feature variation. The general high-level for this Jacobian method is as follows:

1. We first compute the forward derivative $J_F(X^*)$. Note this involves computing the derivative of output neurons with respect to all input dimensions. This requires iterating through each of the layers of the network, which results in some very complicated recursive math involving the outputs of previous layers. This process can be sped up using symbolic differentiation
2. We then compute the saliency map, which traditionally is a visualization used in Explainable AI to understand what parts of an image contribute the most to the decision made by a neural network. In an adversarial sense, an adversarial saliency map indicates what parts of an image should be perturbed in order to produce some desired misclassification. An example of an adversarial saliency map might look as follows:

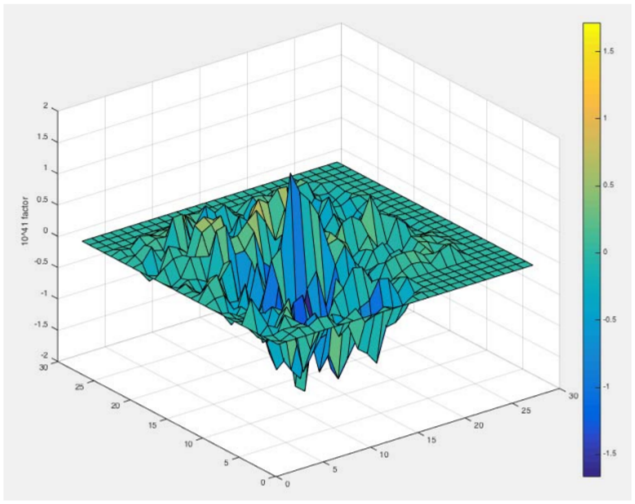


Figure 5. Saliency Map for a 28x28 image. Large magnitudes of values in the map correspond to features that will affect the output the most when some perturbation is added

The formula given for saliency maps that allow adversaries to produce adversarial samples by increasing input features is as follows:

$$S(X,t)[i] = \begin{cases} 0 & \text{if } J_{ij}(X) < 0 \text{ or } \sum_{j \neq t} J_{ij}(X) > 0 \\ J_{ij}(X) \ \mid \sum_{j \neq t} J_{ij}(X) \mid & otherwise \end{cases}$$

This formula is rather complex but this follows the intuition that we should drive toward an increase in values for the probability that a sample is classified as our target class and a decrease in values for the probabilities of other classes. High saliency map values $S(X,t)[i]$ indicate input features that do just that.

3. Lastly, we select the maximum input feature from the saliency map, since that is the most optimal feature for our goal, and modify the input by θ

4. We repeat the above steps until we misclassify the sample to our target class or if we reach the Υ as we have reached the maximum distortion we are okay with

With this method, adversarial samples can be produced. The main advantage of this approach is that it is very successful while producing minimal distortion, which is important in some domains, such as malware executables. It can also be tuned to run efficiently if someone were to want to generate many adversarial samples. They conducted an experiment on the MNIST hand-written numbers dataset [11] using the opposite method, instead decreasing input feature intensities (Note that this requires flipping the conditionals in the saliency map formula presented above) and the adversarial samples generated for sample inputs and target classes were as presented:

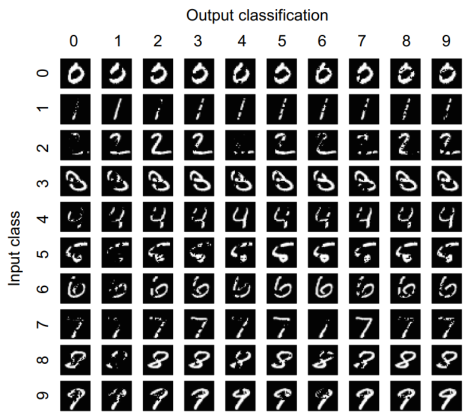


Figure 6. The reverse method of decreasing input feature intensities was used. The original samples arranged on the diagonal had adversarial samples produced from them which were misclassified into all other target classes

2.1.4 Carlini and Wagner Attacks

In 2015, Hinton et al [8], seeking to increase the feasibility of deep neural nets in low computational availability contexts, introduced the idea of distillation, whereby on a high-level, knowledge from a complex is transferred to a smaller distilled model through the use of soft target distributions generated by the complex model. This can then create a computationally lightweight model for inference. In 2016, Papernot et al. [14] built upon this idea to create a potential defense against adversarial attacks. This method was called defensive distillation. This procedure is a training method where knowledge in the form of class probability vectors generated by a DNN is extracted during training. This is then fed back into the training regimen of the original model to increase its robustness to adversarial samples. Essentially, the goal is to reinforce the model's ability to withstand perturbations by leveraging insights gained from its own predictions. Overall, they found that this principle held, as model susceptibility percentages dropped.

In 2017, Carlini and Wagner [2] proved that these distillation techniques were not significantly effective, as their host of three attacks were successful in producing target misclassification regardless. Note that these attacks are very much math and optimization-based which are complex. As such, I will summarize both their methods and what they did to create the attacks. To begin, it is important to discuss the types of similarity measures used to compare samples and their adversarial counterparts. There are three main measures:

1. First, we have the L_0 measure that simply counts the number of pixels that are not the same between a sample and its adversarial counterpart. This is namely used in Jacobian-Based attacks [13]

2. L_2 is the Euclidean distance measurement between a sample and its adversarial counterpart

3. L_∞ is the distance measure that measures the maximum change in any pixel between a sample and its adversarial counterpart. This is namely used in the Fast Gradient Sign Method [6]

In previous work, we usually had the same or similar optimization problem:

$$\text{minimize } \mathcal{D}(x, x+\delta)$$

$$\text{such that } C(x+\delta) = t,\ \ x+\delta \in [0,1]^n$$

However, this problem is very difficult to solve. Instead, they define seven objective functions f subject to the logic $C(x+\delta) = t \Leftrightarrow f(x+\delta) \leq 0$. The equations they present are as follows:

1. $f_1(x') = -loss_{F,t}(x') + 1$
2. $f_2(x') = (\underset{i \neq t}{max}(F(x')_i) - F(x')_t)^+$
3. $f_3(x') = softplus(\underset{i \neq t}{max}(F(x')_i) - F(x')_t) - log(2)$
4. $f_4(x') = (0.5 - F(x')_t)^+$
5. $f_5(x') = -log(2F(x')_t - 2)$
6. $f_6(x') = (\underset{i \neq t}{max}(Z(x')_i) - Z(x')_t)^+$
7. $f_7(x') = softplus(\underset{i \neq t}{max}(Z(x')_i) - Z(x')_t) - log(2)$

Note that notation-wise, $loss_{F,t}$ is cross-entropy loss, $(e)^+ = max(0, e)$, $softplus(x) = log(1 + exp(x))$, F is the neural network including the softmax, and Z is the output of the neural network without the softmax (logits). The optimization problem as a result becomes:

$$\text{minimize } \mathcal{D}(x, x+\delta) + c * f(x+\delta)$$

$$\text{such that } x+\delta \in [0,1]^n$$

where c is some chosen constant and f is an objective function. With this background work, they construct three attacks that follow each distance measure:

L_2 attack: Given a sample x and a target class t, they use the following optimization problem to find a minimal perturbation w:

$$\text{minimize } \|\frac{1}{2}(tanh(w)+1) - x\|_2^2 + c * f(\frac{1}{2}tanh(w)+1)$$

$$\text{where } f(x') = max(max(Z(x')_i : i \neq t - Z(x')_t, -\kappa))$$

This objective function is very similar to one listed above but with a slight modification including κ, which adjusts the confidence that a sample will misclassify. The solving of this problem results in minimal adversarial samples with respect to this distance measure. Note that, wanting to avoid the issue of gradient decent getting stuck at local minima they use multiple starting point gradient decent to make this scenario less likely.

L_0 attack: The method used here is iterative and uses the L_2 attack as part of the procedure. The general goal of the algorithm is to iterate and find pixels that don't result in large output space changes and freeze them. The number of frozen pixels grows at each iteration and you will gradually determine a small set of pixels that can be modified to produce misclassification. To do this, they call the L_2 attack at every iteration and only allow it to change unfrozen pixels. What we get back is the perturbation that induces target misclassification. To do pixel freezing, they find the gradient of the L_2 objective function at the adversarial sample ($g = \nabla f(x + \delta)$) and select the pixel p subject to $p = argmin_p g_p * \delta_p$ to freeze. This is based on the assumption that $g_p * \delta_p$ can tell us how much a pixel contributed to the minimization of the objective function. This process loops until the L_2 attack can no longer find an adversarial example, meaning we've found the minimum number of pixels that can produce misclassification. Note as a speedup, the gradient descent done at each iteration starts at the solution for the previous iteration, which they call a "warm-start".

L_∞ attack: The process for this attack involves the following optimization:

$$\text{minimize } c * f(x + \delta) + \sum_i [(\delta_i - \tau)^+]$$

where τ is a decreasing threshold that aids in preventing gradient descent oscillation. This method also utilizes a "warm-start" to aid in efficiency.

The results for L_2 and L_0 attacks on the MNIST handwriting are shown below. It is evident just how strong these methods are in comparison to the previous when it comes to white-box attacks [1]. Specifically with these handwritten digits, the perturbations are absolutely minimal and the differences are extremely hard to determine. The analysis from Carlini and Wagner also seems to argue this as well as they outperform many other methods, also proving that defense distillation is not a proper or effective guard.

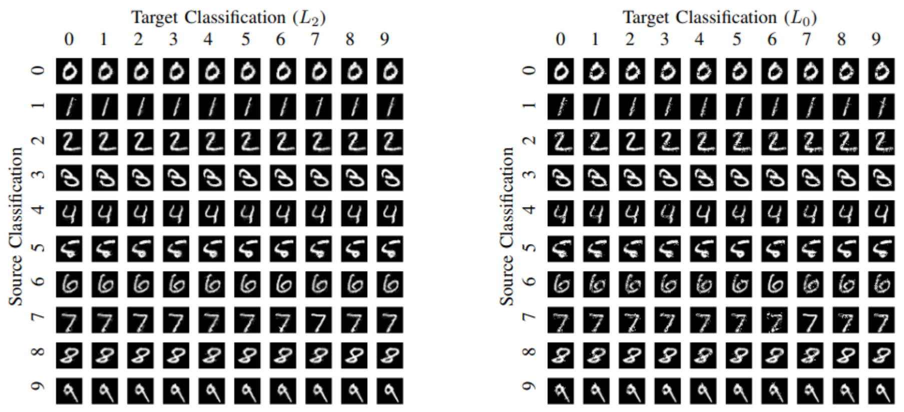


Figure 7. Source/Target Misclassification Pairs on the MNIST handwriting dataset with L_2 and L_0 Carlini and Wagner Attacks

2.1.5 Universal Adversarial Perturbations

Previously, these other white box attacks mentioned were considered to be image-specific, meaning we find a perturbation that is successful for an individual image. In 2017, Moosavi-Dezfooli et al. [12] showed that you could produce universal perturbations whereby you can create a small perturbation that fools a network on all images. It is also shown that these perturbations can be universal across a number of different deep neural networks, which they coin as doubly universal.

The paper goes on to define important notation for their formal definition which estimates perturbations. First, they define μ as the distribution on images in $\mathbb{R}^d$, $\hat{k}$ is a classification function for some sample x, $\hat{k}(x)$ which is an estimated label for x, and v is some perturbation vector that fools $\hat{k}$ on **almost all** samples from μ. We are solving the following:

$$\hat{k}(x+v) \neq \hat{k}(x) \text{ for most } x \sim \mu$$

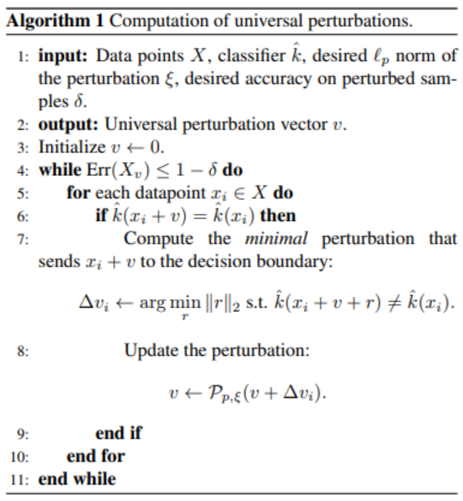
subject to the constraints:

$$\|v\|_p \leq \xi \text{ where } p \in [1, \infty)$$

$$\underset{x \sim \mu}{\mathbb{P}}\left(\hat{k}(x+v) \neq \hat{k}(x)\right) \geq 1 - \delta$$

These constraints say essentially that the magnitude of the perturbation should be under ξ and that the probability over images sampled from μ where the perturbed classification differs from the original classification should be at least $1-\delta$ where δ is the desire fooling rate.

The actual algorithm that is able to construct this universal perturbation vector is the following picture on the left:



```
Algorithm 1 Computation of universal perturbations.
1:  input: Data points $X$, classifier $\hat{k}$, desired $\ell_p$ norm of
    the perturbation $\xi$, desired accuracy on perturbed samples $\delta$.
2:  output: Universal perturbation vector $v$.
3:  Initialize $v \leftarrow 0$.
4:  while $\mathrm{Err}(X_v) \leq 1 - \delta$ do
5:      for each datapoint $x_i \in X$ do
6:          if $\hat{k}(x_i + v) = \hat{k}(x_i)$ then
7:              Compute the minimal perturbation that
    sends $x_i + v$ to the decision boundary:
                  $\Delta v_i \leftarrow \arg\min_r \|r\|_2 \text{ s.t. } \hat{k}(x_i + v + r) \neq \hat{k}(x_i)$.
8:              Update the perturbation:
                  $v \leftarrow \mathcal{P}_{p,\xi}(v + \Delta v_i)$.
9:          end if
10:     end for
11: end while
```

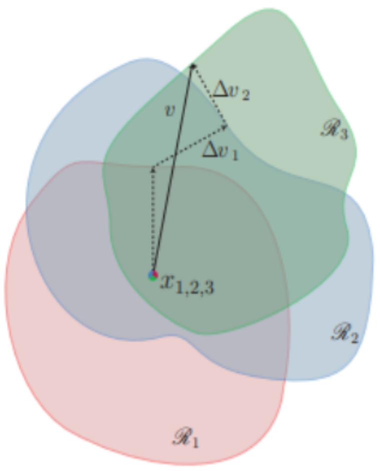


Figure 8. Left: Pseudocode to calculate universal perturbations, Right: Visual representing the algorithm on the left. Here we see data points $x_{1,2,3}$ and classification regions $\mathscr{R}_{1,2,3}$. The arrows represent adding minimal perturbations to x_i to push perturbated samples $x_i + v$ outside of the classification region $\mathscr{R}_i$

Line 4 in the pseudocode indicates to continue running while the percentage of perturbated classification labels matching correct classification labels across the set of images does have a ratio above one minus the desired fooling rate. While the condition holds, we loop through each sample in the dataset, checking if it misclassifies yet. If it does not, we solve the optimization problem in line 7 which finds a minimal perturbation that shifts that perturbated sample to the decision boundary. This is visually represented in the image on the right of Figure 8. The deltas and corresponding arrows represent these minimal shifts. We then update the perturbation vector with this minimal shift. However, we must be careful to not violate the constraint of $\|v\|_p \leq \xi$. The function used to update the perturbation in line 8 expands to:

$$\mathcal{P}_{p,\xi} = \underset{v'}{argmin}\|v - v'\|_2 \text{ subject to } \|v\|_p \leq \xi$$

Conceptually, this equation finds the most similar vector to the perturbation vector with the most recently computed minimal perturbation added (from line 7). The constraint placed on this equation also necessitates that the similar vector must lie within the p-norm ball of radius ξ. To help visualize this, the following image shows some unit balls of various norms:

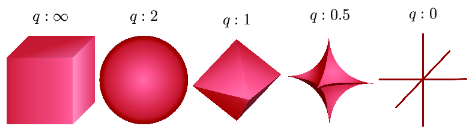


Figure 9. Unit ball representations for norms of ∞, 2, 1, 0.5, and 0

Based upon p, we might have p-norm balls of these various shapes but scaled with radius ξ. The equation would then have to output the most similar vector that falls within this space, ensuring we abide by the constraint. Once our threshold for fooling rate is crossed, we have produced a universal perturbation vector. Note this is not a minimal perturbation but one that fits the constraints. Shuffling the dataset can result in different universal perturbations. This was done on multiple popular DNNs and the results are below:

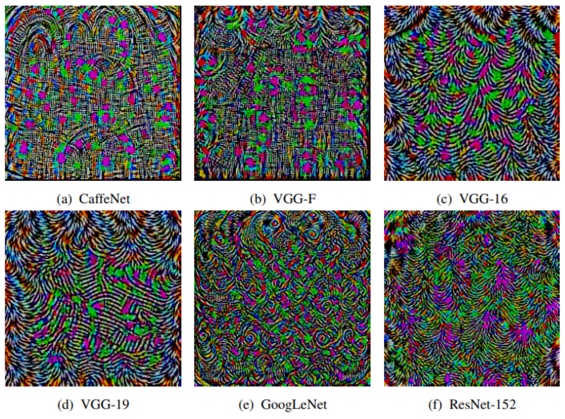


Figure 10. Universal adversarial perturbations computed on the shows DNNs with $p = \infty$ and $\xi = 10$

In explaining why this works, they found that l_2-norms had a lot of success, indicating that universal perturbations exploit geometric correlations between different parts of the classifier's decision boundary. They followed this up by studying normal vectors to the decision boundary. The resulting matrix's singular values were found to decay quickly, meaning there is a low-dimensional subspace correlating various regions of the decision boundary. What this means is that data can be effectively represented or captured by a relatively small number of dimensions or features. Perturbations aligned with this subspace can influence multiple regions of the decision boundary, allowing a single perturbation to fool a wide range of inputs.

2.2. Black-Box Attacks

On the contrary to white-box attacks, black-box attacks operate in scenarios where there is limited or no access to the internal details or parameters of the target model. These are most likely to mimic real-world attacks.

2.2.1 One-Pixel Attack

In 2017 (revised in 2019), Su et al. [18] showed that you could induce image misclassification by modifying a single pixel in sample images with little adversarial information, meaning the attacker does not need much information about the model, only the label output probabilities.

They formalize their approach with the following optimization problem, again similar to previous approaches:

$$\underset{e(x)^*}{\text{maximize}}\; f_{adv}(x + e(x))$$

$$\text{subject to } \|e(x)\|_0 \leq d$$

where x is the vectorized sample, $e(x)$ is the perturbation vector, f_{adv} is the target class, and d is perturbated dimensions (which is set to 1 for one-pixel attacks). Note that 0-norm is essentially the number of non-zero vector entries.

To optimize this equation they use a population-based optimization technique called Differential Evolution [17]. The high-level overview of this technique is that you first create an initial random population of vectors that should span the parameter space. Then, candidate solutions or mutants are created by mutating the existing solutions, which in the first iteration are the random initial vectors. This is typically done by taking the difference between two random solution vectors, scaling the difference by a factor, and adding the difference to another random initial vector. Next is the crossover phase where some crossover method combines information between mutants and initial candidate solutions to generate trial solutions or offspring solutions, mimicking the idea of evolution. These trial solutions are then evaluated to determine their fitness. If they perform better than their corresponding candidate solution, they replace that solution. Otherwise, the previous solution lives on. This process loops and continues until some termination condition is reached.

Their application of Differential Evolution involves first encoding vectors for potential solutions. They define their candidate solutions (perturbations) in a five-tuple structure containing the x coordinate, y coordinate, and 3 values corresponding to the RGB value of the perturbation. Their initial populations are generated using Uniform distributions. They also then set the initial population of candidate solutions to be 400 with another 400 solutions being produced every iteration. The formula for Differential Evolution is essentially the standard equation:

$$x_i(g+1) = x_{r1}(g) + F(x_{r2}(g) - x_{r3}(g)) \text{ subject to } r1 \neq r2 \neq r3$$

where x_i is the specific candidate solution, g is the current generation, $r1, r2, r3$ are random numbers that are not equal, and F is the aforementioned scaling factor (which they set to 0.5). The termination condition here depends on whether this attack is targeted and has a target class. If it is targeted, it stops when the probability of the target class crosses some threshold. If it is not targeted, it stops when the probability of the real class goes below some threshold. They avoid using Crossover at all here and naturally, the fitness of solutions is evaluated using the probability labels of the target class or true class depending on whether the class is targeted. The results of this attack are shown below:

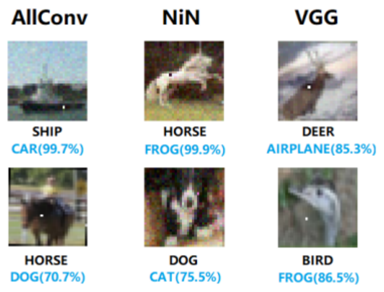


Figure 11. One-Pixel Attacks on three popular DNNs trained on CIFAR-10

We see that these popular DNNs can be fooled by these one-pixel attacks. They also pose 3-pixel and 5-pixel attacks and find that these generalize pretty well. Note that these attacks are not meant to be imperceptible, as you can clearly see what pixel was perturbated.

2.2.2 UPSET and ANGRI

In 2017, Sakar et al. [15] posed two black-box algorithms: Universal Perturbations for Steering to Exact Targets (UPSET) and Antagonistic Network for Generating Rogue Images (ANGRI). Note that the paper seems to lack significant detail, therefore my description may not be comprehensive but is still true to the contents of the paper.

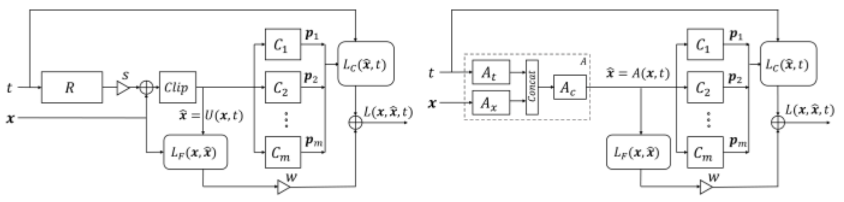
As the name suggests, UPSET aims to construct universal perturbations such that any image not belonging to some class c that has the corresponding universal perturbation added to it, it will then adversarially classify that image as class c. The general equation for adversarial generation is:

$$\hat{x} = max(min(s \times R(t) + x, 1), -1)$$

where s is some scaling factor and R is some residual generating network that takes in a class t and outputs a perturbation for fooling. This R seems to have an arbitrary structure depending on the dataset used, though the paper never reveals how the layers are selected and how training occurs. Note that the sample x is normalized to the interval $[-1, 1]$ which is why the general equation clips the output to be between the interval as well.

ANGRI aims to do targeted misclassification, taking an input sample and leading the classifier to classify the image as some specific target class. The paper does give any formal definition for this method but it seems to imply that it operates similarly to UPSET in that it has some auxiliary network (also changes depending on the dataset) that outputs a perturbation for fooling.

The following diagram illustrates the training process:



(a) Training scheme for UPSET(U). (b) Training scheme for ANGRI(A).

Figure 12. The training diagram for UPSET and ANGRI

Both algorithms utilize a similar loss function which you can see the visual components of in the diagram:

$$L(x, \hat{x}, t) = L_C(\hat{x}, t) + L_F(x, \hat{x}) = -\sum_{i=1}^{m} log(C_i(\hat{x})[t]) + \lambda$$

$$\text{where } \lambda = w\|\hat{x} - x\|_k^k \text{ if algorithm is ANGRI, else } \lambda = \|R(x,t)\|_2^2$$

C_i are pre-trained classifiers for the dataset (the paper never states this but each is likely a target model we are trying to exploit), L_C is the categorical cross-entropy loss that is tied to whether a classifier properly predicts the target class, L_F is the fidelity loss that ensures images and their perturbations are similar, w is a weight value for the loss components, and k is some arbitrary value for the norm. Though this is as detailed as the paper goes, my assumption is that this loss trains the R networks for UPSET and the A network for ANGRI.

The ideal advantage here is that you can produce adversarial samples that work across the classifiers used in the training process. In experimentation, they use arbitrary pre-trained classifiers they created and never on popular DNNs. Their results are shown below:

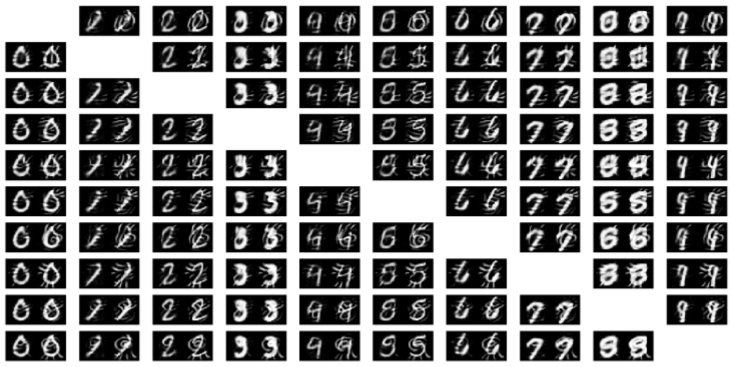


Figure 13. The rows and columns are the MNIST numbers 0-9. In each cell, the left image ANGRI adversarial sample and the right image is the UPSET adversarial sample.

2.2.3 Substitute Model

In 2016 (revised in 2017), Papernot et al. introduced a black-box attack that generates adversarial samples after generating a substitute model that tries to emulate the target network. This attack only requires an oracle $\tilde{O}$ of the target DNN that returns the output class of any input passed in.

The approach is again formalized as the following optimization problem:

$$\vec{x^*} = \vec{x} + argmin\{\vec{z} : \tilde{O}(\vec{x} + \vec{z}) \neq \tilde{O}(\vec{x})\} = \vec{x} + \delta_{\vec{x}}$$

where $\vec{x^*}$ is the adversarial sample for $\vec{x}$, O is the aforementioned oracle for the DNN, and $\delta_{\vec{x}}$ is a minimal perturbation.

Previous work has been done regarding substitutes, whereby if you have an independent dataset of the same population distributions as the dataset used to train some target model, you could train a substitute with the independent dataset, finding that the adversarial samples that fooled the substitute also fooled the target network. This runs into a feasibility issue, however, since popular networks may train on incredibly large labeled datasets. This method aims to avoid that issue. It does this by crafting synthetic samples, labeled by what the oracle $\tilde{O}$ outputs as its prediction. From this, we can construct an approximation of the target model from which we can craft adversarial examples that should apply to both models.

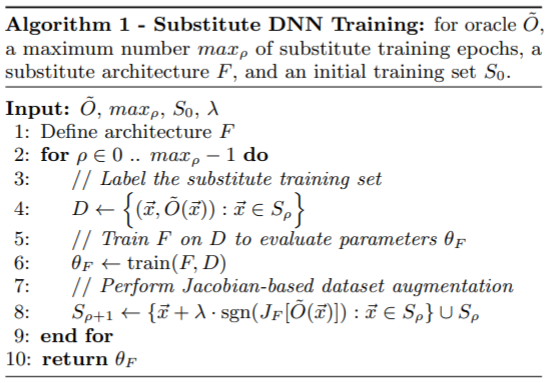
The major task in this is constructing the substitute model. First, the architecture for this substitute model should be chosen. Given the domain of the problem, the architecture can be selected based on that. For example, image classification often uses convolutional neural networks. However, they also argue that based on their results you can achieve good results even with differing architectures. Next, the synthetic dataset must be generated. Here, they argue that simply randomly passing in images or passing in every possible image is simply infeasible and can lead to infinite queries depending on the input domain space. Instead, they introduce the idea of some kind of heuristic, which they call the Jacobian-Based Data Augmentation. This process hinges on the intuition that in order to properly approximate the target model, we should generate samples in directions that cause variation in output labels starting from some set of initial samples. The Jacobian of the substitute model J_F is used in tandem with the output label of the oracle $\tilde{O}$ as such:

$$sign(J_F(\vec{x})[\tilde{O}(\vec{x})])$$

This seems to prioritize samples in the direction where the Jacobian direction aligns with the oracle's assigned label. We generate new samples with this equation:

$$sample = (\lambda * sign(J_F(\vec{x})[\tilde{O}(\vec{x})])) + \vec{x}$$

where λ indicates how far in the calculated direction we should step. The pseudocode for the substitute model training is as follows:



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: $\tilde{O}$, $max_\rho$, $S_0$, $\lambda$
 1: Define architecture $F$
 2: for $\rho \in 0 .. max_\rho - 1$ do
 3:     // Label the substitute training set
 4:     $D \leftarrow \left\{ (\vec{x}, \tilde{O}(\vec{x})) : \vec{x} \in S_\rho \right\}$
 5:     // Train $F$ on $D$ to evaluate parameters $\theta_F$
 6:     $\theta_F \leftarrow \text{train}(F, D)$
 7:     // Perform Jacobian-based dataset augmentation
 8:     $S_{\rho+1} \leftarrow \{\vec{x} + \lambda \cdot \text{sgn}(J_F[\tilde{O}(\vec{x})]) : \vec{x} \in S_\rho\} \cup S_\rho$
 9: end for
10: return $\theta_F$
```

Figure 14. Peudocode for training the Substitute Model

As the image states, we first create our small initial dataset and define the architecture for the substitute model. The architecture is as mentioned previously, and the initial dataset can be any minimal dataset representative of the input domain (Ex. for MNIST handwritten digits, one sample from each class 0-9). Then, we move to the iterative training stage. For the number of substitute train epochs, we first have the oracle $\tilde{O}$ label all the images in our current training set S_ρ. Then, we train our current substitute model on the now-labeled training set. Lastly, we then use the aforementioned Jacobian-Based Data Augmentation to generate a larger training set $S_{\rho+1}$ (adding onto our current training set) that should help us learn the target model's decision boundaries better. After we fully iterate, we ideally have a similar substitute model.

With this substitute model, we can now construct adversarial samples that should fool both the substitute and target model. To construct these adversarial samples, we can simply use any of the white-box attacks we previously defined since we know all the information about this substitute model. Some adversarial samples are in the image below:

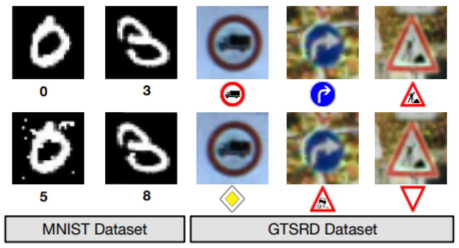


Figure 15. Adversarial samples using the Substitute Model approach on MNIST and GTSRD [16] datasets

2.2.4 SimBA

In 2019, Guo et al. [7] came up with their approach to black-box attacks which focuses on simplicity and generality across different scenarios. This attack is untargeted, meaning it does not misclassify to a particular label. The knowledge this attack requires is also just the label probabilities given some input.

The assumption they use here is that the output probabilities from a network can orient the search for adversarial samples. The pseudocode for the algorithm is below:

Algorithm 1 SimBA in Pseudocode

```
1:  procedure SIMBA($\mathbf{x}, y, Q, \epsilon$)
2:      $\delta = \mathbf{0}$
3:      $\mathbf{p} = p_h(y \mid \mathbf{x})$
4:      while $\mathbf{p}_y = \max_{y'} \mathbf{p}_{y'}$ do
5:          Pick randomly without replacement: $\mathbf{q} \in Q$
6:          for $\alpha \in \{\epsilon, -\epsilon\}$ do
7:              $\mathbf{p}' = p_h(y \mid \mathbf{x} + \delta + \alpha\mathbf{q})$
8:              if $\mathbf{p}'_y < \mathbf{p}_y$ then
9:                  $\delta = \delta + \alpha\mathbf{q}$
10:                 $\mathbf{p} = \mathbf{p}'$
11:                 break
        return $\delta$
```

Figure 16. Pseudocode for the SimBA algorithm

The main intuition of the algorithm is that a step (step size $\epsilon > 0$) in any direction q for any input sample should decrease the probability of the true label $p_h(y|x)$. The algorithm explores directions in the set Q multiplied by the step size and either subtracts or adds them, meaning we explore $x + \epsilon q$ or $x - \epsilon q$. This set Q has a couple of attributes. In the interest of efficiency, it must be ensured that directions don't cancel each other out since that would delay progress. We also need to make sure directions don't compound since that would increase the perturbation such that it exceeds the threshold for imperceptibility. This is why directions in Q are chosen without replacement and each direction in Q is also orthonormal. In essence, the algorithm, while the predicted label is the true label, picks directions that decrease the probability of the true label the most, adding those such alterations to our overall perturbation for the input sample. Once the predicted label differs from the true label (note we query the target model whenever we calculate a new step in a direction), we have found a somewhat small perturbation that fools the target model with only the knowledge of output probabilities from the target model.

They also discuss how they choose the orthonormal directions of Q. The most basic way is using the cartesian basis/standard basis where $Q = I$. This conceptually means that they are altering the color of a randomly chosen pixel at each iteration. The second way they used was to exploit the idea that randomly sampled noise in low-frequency space can be effective in finding adversarial samples. As such they also form the orthonormal basis using discrete cosign transform (DCT) which maps signals in the input space to frequency coefficients that relate to cosign waves. They do not go into much more detail here but the results prove that both methods are effective. Some adversarial samples are shown below:

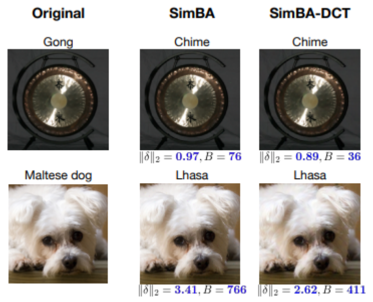

Original
SimBA
SimBA-DCT
Gong
Chime
Chime
$\|\delta\|_2 = 0.97, B = 76$
$\|\delta\|_2 = 0.89, B = 36$
Maltese dog
Lhasa
Lhasa
$\|\delta\|_2 = 3.41, B = 766$
$\|\delta\|_2 = 2.62, B = 411$

Figure 17. Adversarial samples generated by SimBA using the cartesian basis and DCT basis for orthonormal directions. Note here δ is the perturbation and L_2 norm is used to calculate its deviation from the original. B is also the number of times the target model was queried.

Though the paper does not seem to identify the target model, we see that generally, the use of DCT creates adversarial samples with a smaller perturbation and queries the target model fewer times. Ideally, the authors hope that this method is general enough to apply in other scenarios, including other domains like speech recognition.

3. Conclusion

I have outlined various adversarial attacks that exploit vulnerabilities in state-of-the-art image classification models, whether they operate as white-box or black-box attacks, and whether they are targeted or untargeted. These research endeavors significantly contribute to our collective understanding of this domain, which remains an ongoing challenge requiring continuous exploration. As neural networks for image classification gain widespread adoption across diverse industries, the potential impact of adversarial attacks becomes increasingly profound. For instance, in the automotive sector, the emergence of self-driving cars relying on image data introduces safety concerns, with security breaches and adversarial attacks posing risks to both drivers and pedestrians. Similarly, in healthcare, where AI aids in diagnostics, adversarial threats could have catastrophic consequences for patients, as exemplified by the findings of Hirano et al. [9]. Even now, sectors like finance, military/defense, retail, and manufacturing are susceptible to such attacks, and the scale of application is poised to intensify over the next decade. This is before we account that many of these attacks listed also believe they can applied to other domains. Investing further efforts in this field is important, as it not only enhances our understanding of these seemingly inscrutable black-box image classification models but also enables us to guard against potential vulnerabilities and plug holes in our systems. Similar to the perpetual cat-and-mouse dynamics in computer security, this ongoing process will likely persist and explode in the realm of image classification and recognition in the foreseeable future, as we continue to make new discoveries both on the defense and adversarial side.

References

[1] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018. 1, 10

[2] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks, 2017. 8

[3] Anirban Chakraborty, Manaar Alam, Vishal Dey, Anupam Chattopadhyay, and Debdeep Mukhopadhyay. Adversarial attacks and defences: A survey, 2018. 1

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. 4

[5] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2000. 3

[6] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015. 4, 8

[7] Chuan Guo, Jacob R. Gardner, Yurong You, Andrew Gordon Wilson, and Kilian Q. Weinberger. Simple black-box adversarial attacks, 2019. 17

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. 8

[9] Hokuto Hirano, Akinori Minagi, and Kazuhiro Takemoto. Universal adversarial attacks on deep neural networks for medical image classification. *BMC Medical Imaging*, 21(1):9, Jan 2021. 18

[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. 3

[11] Yann LeCun and Corinna Cortes. The mnist database of handwritten digits. 2005. 7

[12] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations, 2017. 10

[13] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroSP)*, pages 372–387, 2016. 1, 2, 5, 8

[14] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks, 2016. 8

[15] Sayantan Sarkar, Ankan Bansal, Upal Mahbub, and Rama Chellappa. Upset and angri : Breaking high performance image classifiers, 2017. 14

[16] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012. Selected Papers from IJCNN 2011. 17

[17] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997. 13

[18] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, Oct. 2019. 13

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014. 4

[20] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2014. 2, 4