Panda or Paintbrush? An Exploration of Adversarial Attacks on Image Classifiers

Jonas Bartels, Alice Cutter, Sriya Konda, Yuxin Lin, Tianyi Lu, Tingjun Tu

***Abstract*—Deep neural network-based image classifiers significantly impact various sectors of society, with applications such as automatic driving and tumor detection. However, researchers have found that these models are vulnerable to carefully selected perturbations that can cause misclassification. These adversarial examples allow attackers to infiltrate real-world neural network image classifiers and pose a security risk. This paper examines four adversarial attacks (Fast Gradient Sign Method, Projected Gradient Descent, LocSearchAdv, and Surrogate Attack) based on the attacker's knowledge level of the target model and compares their effectiveness on the same testing dataset by running experiments on ImageNet data using MobileViT as the target model. No attack was found to outperform the others in all respects; however, certain algorithms perform better depending on the situation. The paper also discusses possible defense strategies and ethical concerns related to adversarial attacks.**

I. Introduction

As AI becomes more prominent, it is important to acknowledge that the inputs given to machine learning (ML) systems may cause harm, whether that may be inputs given by a human or taken by the ML system itself. This led to the development of the field of Adversarial Machine Learning, which manipulates input to reach a desired outcome by taking into account the characteristics of an ML system [1]. These inputs are known as adversarial examples, which aim to have the target model purposefully return a wrong output [2]. A *target model* is the classifier for which the attacks are generated. It is the model whose weakness attackers are trying to exploit. The manipulations made on the input are known as adversarial perturbations. The algorithms that create that adversarial perturbation are known as attacks [3]. Defenses can then be implemented so ML systems would not fall vulnerable to adversarial cases and attacks. These attacks and defenses are done by "attackers" and "defenders", respectfully.

Though the risk of these adversarial cases has yet to be quantified, it is important to lower the risk before they enter the real world [3]. Suppose an attacker is successful in attacking one ML system. That attacker might then be able to transfer the same knowledge to attack more ML systems without much knowledge of the system or the data, also known as a black-box attack [4].

In particular, our paper focused on comparing adversarial attacks on image classification models. An image classification model is an ML model where an image is inputted and a label is returned for the contents of the image. Our main motivation to use an image classifier is due to vast applications. For instance, image classification software is used in medical imaging for tumor detection [5] or in the LiDAR system in self-driving cars [6].

Consider in Fig. 1, which our target model classifies as an image of a panda correctly. Once an adversarial attack is run on the image classification model, the input image becomes perturbed. An instance of this is seen in Fig. 2 and Fig. 3, where an attack is run in Figure 1. This results in the model having a harder time identifying the object in the image, where the model may classify the image incorrectly (as seen in Fig. 3) or if it classifies the image correctly, the classification will have a lower confidence (as seen in Fig. 2). However, the effect that the perturbations have on an image classifier is not equivalent to its effect on a person. Even though the perturbations in Figures 2 and 3 change the output resulting from the image classifier, a human may still be able to accurately classify these two images as pandas. Yet, some attacks have greater visible perturbations than others, as seen in Fig. 3.

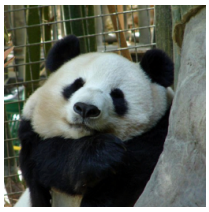

Fig. 1: Original Panda Image
Label given by Image Classifier: Panda

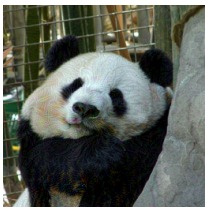

Fig. 2: Perturbed Panda Image
Label given by Image Classifier: Panda

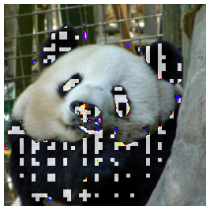

Fig. 3: Perturbed Panda Image
Label given by Image Classifier: Gibbon

A. Attack Types

Adversarial attacks can be classified in many ways. They can be grouped based on how they interact with the model they are attacking, and what part of the model they are attacking, for example, poisoning training data versus attacking the model itself. Within the scope of this paper, we will break down adversarial attacks based on the attacker's knowledge.

The first type of attack is Perfect-Knowledge (PK), or white box attacks. In this scenario, it is assumed attackers know everything about a system. These types of attacks are useful so one can prepare for worst-case scenarios. An example might be attacking one's own system to see where there are weak points [7].

The next type of attack is Limited Knowledge (LK) or gray-box attacks. These attacks are one where the attacker has some understanding of the model but only some of the information. With gray box attacks, attackers generally know the kind of learning algorithm or the architecture for a model, but not the training data or the trained parameters [7]. It is assumed that the attacker would have a surrogate dataset from a similar source. Grey box attacks can be used to test how well attacks can transfer across different models [7].

The final type of attack is a Zero-Knowledge (ZK) or black box attack. A black box is a situation in which the attackers do not know the target model's architecture or weights, and have no access to its exact training data. In this case, the only access they have to the model is interacting with it: getting its predicted class label, and possibly a confidence value, after giving it an image. The attacker will likely also make some basic assumptions about the model. For example, if the model discerns pictures of animals, they know that it was probably trained on images of animals [7].

Attacks can also be classified as either targeted or indiscriminate. A targeted attack is when the adversarial examples aim to get a specific output for a specific classification. A non-targeted attack is when the goal is to get an incorrect classification irrespective of class [8].

II. Implementation

We split the attacks we wanted to focus on into white-box and black-box conditions. Given the ambiguity of gray box attacks, we have chosen to not include them in our study. Our white-box attacks include the Fast Sign Gradient Method (FGSM) and Projected Gradient Descent (PGD). Our black-box attacks are LocSearchAdv and the Surrogate Attack.

We aim to compare how effective the attacks are at manipulating the model. We consider an instance of the attack correctly fooling the model to provide the wrong input as a success. We assume that the white-box attacks are more effective in attacking the model compared to the black box due to having more accessible knowledge of the targeted model.

A. FGSM

The Fast Gradient Sign Method (FGSM) is one of the most well-known pioneering approaches in the domain of adversarial machine learning, introduced by Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy in their seminal work [9]. The underlying assumption of the FGSM is that despite the high-dimensional and non-linear nature of deep learning models, the decision boundaries can be linearly approximated in the vicinity of the data points, making it possible to generate effective adversarial examples with a single step of gradient ascent. The essence of FGSM lies in its use of the gradient of the loss function with respect to the input image to create perturbations that maximize the loss. By taking the sign of the gradient w.r.t the input image, it ensures that the adversarial noise is directed towards increasing the model's prediction error, thereby effectively and efficiently misleading the network. Despite being developed eight years ago, it still shines light on the vulnerability of contemporary neural networks to adversarial attacks given its simplicity and efficiency.

For a color image, FGSM first calculates the gradient of the model's loss function, regarding the correct label w.r.t. this input image. This gradient essentially indicates the direction in which each pixel's RGB channel's intensity should be adjusted to maximize the model's error. As depicted in Figure 4, The vertical axis, l represents the loss function of the classifier regarding the true label, which measures the error between the classifier's prediction and the true label. The curve shows how the loss changes as we move away from the original image x (one of the RGB channel values for a specific pixel) in the input space. FGSM works by taking the gradient (the derivative) of the loss function w.r.t. the input image x. This gradient points in the direction where the loss function increases, indicating how the pixels in the image should be changed to increase the classifier's error. FGSM then adjusts the original image x by a small amount ϵ in the direction of this gradient (indicated by the dotted line). The size of ϵ is chosen to be small enough that the change is imperceptible to the human eye but large enough to fool the classifier. The new image x' is the adversarial example, which is very close to x in the input space (the distance is ϵ), but far enough along the gradient of the loss function when adding across all the dimensions (all pixels RGB channel values) to significantly increase the loss, leading the classifier to likely misclassify x'.

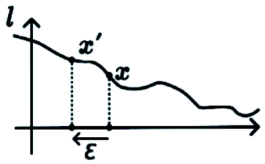

Fig. 4: Example of taking a fixed step of ϵ in the direction of the gradient (negative) to increase the loss regarding the correct label.

B. PGD

Projected Gradient Descent (PGD) is another gradient-based attack method that focuses on the inner optimization problem of maximizing the loss regarding the true label of a given input. PGD is one of the first and most basic attack methods in the literature, but we can perceive it as an iterative version of FGSM with a more carefully chosen and dynamic step size. Unlike FGSM, which takes a single large step to create an adversarial example, PGD takes multiple small steps, allowing for a more refined search within the adversarial space. At each step, PGD calculates the gradient of the loss function

l w.r.t. the adversarial example x' generated in the previous iteration and then moves x' in the direction of this gradient by a small amount. This process is repeated for a fixed number of iterations or until convergence.

For a color image, similar to FGSM, PGD first calculates the gradient of the model's loss function at each iteration, regarding the correct label of the input images. Equally, this gradient indicates the direction of how to adjust the input to increase the loss w.r.t. one of this pixel's RGB channel values. Unlike FGSM which takes a fixed-size step toward the direction of the gradient, PGD takes a much smaller step for all pixels determined by the optimizer at each iteration and calculates the gradient in the resulting position again in the next iteration to make sure the loss is always increasing as the number of iterations grows. Take the example from Figure 5, to ensure the perturbations remain imperceptible, PGD employs a projection operation after each gradient step to clamp the perturbed example x' back onto the ϵ-ball around the original image x. This ϵ-ball defines the space of allowable perturbations that do not exceed the predefined perturbation limit ϵ. This projection effectively makes sure that the adversarial example does not stray too far from the original image in terms of pixel values, maintaining the visual similarity that is critical for the adversarial example to remain undetected by human observers. The resulting adversarial example from PGD is thus a product of a careful balance between maximizing the classifier's loss and staying within the imperceptibility constraints imposed by ϵ. Through this iterative and refined approach, PGD can discover more potent adversarial examples compared to FGSM, often resulting in a higher success rate of misclassification in white box situations.

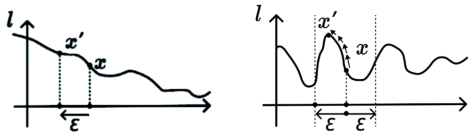

Fig. 5: Left: Example of perturbation using FGSM; Right: Example of perturbation using PGD that results in finding a local maximum within the ϵ-ball for the loss regarding the correct label.

C. LocSearchAdv

While FGSM and PGD use gradient information to generate perturbation, one black box attack that tries to infer information from the model based on input images is LocSearchAdv Nina Narodytska and Shiva Prasad Kasiviswanathan propose a method for generating adversarial examples in a black box setting by adding perturbations to a set of randomly selected pixels in the input image [10]. This algorithm, called LocSearchAdv, works under the assumption that there exists a subset of pixels in the input images such that perturbing their values will lead to a failure for a neural network to classify that image correctly. To ensure the efficiency of the algorithm, the authors utilize a greedy local search to carefully select a small set of pixels to perturb in each round. The main motivation behind this algorithm is to iteratively locate a small set of pixels that leads to misclassification by a deep neural network without using any gradient information [10]. Therefore, for each iteration, the algorithm will find a set of potential pixels, denoted by P_i, to perturb and obtain a subset of these pixels $\hat{\mathcal{I}}$) that are most likely to cause failure in prediction as the actual pixels perturbed at that iteration.

Table 1 LocSearchAdv Parameters

I: input image
p: init perturbation coefficient to determine the set of most sensitive pixels
r: perturbation param for actual perturbation for the pixels
t: number of pixels perturbed per iteration
n: neighborhood size
$Iter$: max number of iterations
LB: lower bound of the pixel value, $LB < 0 < UB$
UB: upper bound of the pixel value, $LB < 0 < UB$

Pseudocode for LocSearchAdv:

1) Resizes all pixel values in the I within $[LB, UB]$
2) For each iteration i such that $i <= Iter$
   - Select a pool of potential pixels (P_i) to perturb for this iteration
     - If $i = 1$, P_i is randomly sampled from the input images
     - If $i \neq 1$, P_i is the neighborhood of the pixels perturbed at the previous iteration $(i-1)$
   - Perturb each pixel in P_i individually and get the confidence interval for the true class prediction of that perturbed image to obtains a set of confidence intervals ($\mathcal{I}$) of the true class with each value representing a pixel in P_i.
     - This perturbation is performed by taking the sign of that pixel value and times a coefficient, denoted by p.
   - Select a subset of the pixels ($\hat{\mathcal{I}}$) in $\mathcal{I}$ that have the lowest confidence interval of the true class label as the pixels to perturb for this iteration i, denoting the set of most sensitive pixels to cause a misclassification in this round.
     - This perturbation is performed by applying a coefficient (r) to the pixel value in a cyclic manner to ensure that the perturbed pixel value is still in the $[LB, UB]$ range.

Two types of perturbations happen in each iteration of the algorithm. The first occurs when the algorithm finds the set of pixels in P_i that are most sensitive to causing a misclassification. Therefore, the algorithm doesn't need to constrain the perturbation so that all pixel values are in $[LB, UB]$. However, for the second perturbation, which occurs when the algorithm perturbs the pixel selected for this iteration, the algorithm must ensure that the image is still valid after the perturbation and that all pixel values are in $[LB, UB]$. Therefore, the algorithm

uses the cyclic methods shown below to perturb each selected pixel.

Algorithm 1 Cyclic (LB, UB, r, $pixel_{val}$)

if $r \cdot pixel_{val} < LB$ **then**
  return $r \cdot pixel_{val} + (UB - LB)$
else if $r \cdot pixel_{val} > LB$ **then**
  return $r \cdot pixel_{val} - (UB - LB)$
else
  return $r \cdot pixel_{val}$
end if

Based on the construction of cyclic methods illustrated above, we need to make sure that the pixel value returned is strictly in the range of $[LB, UB]$. Hence, some restriction on r is necessary. Considering the two edge cases where the $pixel_{val}$ is LB or UB, we need to make sure that:

$$\begin{aligned} r \cdot UB - (UB - LB) &\leq UB \\ r \cdot UB &\leq UB + UB - LB \\ r \cdot UB &\leq 2UB - LB \\ r &\leq 2 - \frac{LB}{UB} \end{aligned}$$

$$\begin{aligned} r \cdot LB &\leq r \cdot LB + (UB - LB) \\ r \cdot LB &\leq LB + LB - UB \\ r \cdot LB &\leq 2LB - UB \\ r &\leq 2 - \frac{UB}{LB} \end{aligned}$$

Therefore,

$$r \leq min(2 - \frac{UB}{LB}, 2 - \frac{UB}{LB})$$

However, our original implementation of LocSearchAdv doesn't perform well on large images. We based our implementation on the algorithm described in Narodytska and Kasiviswanathan's paper Simple Black Box Adversarial Perturbations for Deep Networks [10]. Their algorithm was designed to run on images from the CIFAR-10 dataset which has colored images of size 64×64 pixels. The images our experiments ran off of were of size 256×256 pixels. This difference in image size is significant because it means that one perturbed pixel in a 64×64 image has much more weight than a single pixel in a 256×256 image. This difference led us to modify LocSearchAdv and develop the Grid Method to improve the algorithm's performance.

The Grid Method builds off of the idea that the images we are running off of are higher resolution than images that LocSearchAdv was originally designed to run on (CIFAR-10). The Grid Method works by dividing the larger image M into a grid and treating each cell in the grid as a single pixel. For each iteration, the algorithm picks a set of pixels in the grid to perturb and we map these pixels in the grid to the corresponding sections of pixels in the input image to perform the actual perturbation. We define our smaller image size as n x n for image N and our real image size as m x m for image M where $n < m$. The goal is to scale image M down to the size of N, which perturbs more pixels at each round and can cover more regions in the original image. In addition to segmenting the image into a grid, for each pixel, we perturb at each iteration, we ignore that pixel for the next 30 iterations. In this way we don't perturb the most sensitive pixels frequently and, the algorithm will be able to traverse all sections of the input image without focusing on one specific region of the image.

Moreover, to better find the most sensitive pixels in each round, we adeptly update the value of p such that the set of $\mathcal{I}$ is in a reasonable range. More specifically, after we obtain the set of confidence intervals for the pool of pixels, we calculate its average value (notation). If the average value is smaller than 0.1, this indicates p is too big such that we lose any distinction between each pixel and we decrease p for the next round. If the average value is too big, this indicates p is too small such that we didn't perturb the pixel enough to showcase how one pixel would contribute to the misclassification of the input image and we increase p for the next round.

It is worth noting that the amount of perturbations visible to the human eye in Grid Method, Fig. 7, is more than in the original LocSearchAdv, Fig. 6. This suggests the question of whether it is better to have an algorithm that can perturb more reliably but is more noticeable to the human eye or to have a less successful algorithm where the perturbed images are less noticeable, which we dive deeper into the discussion section.

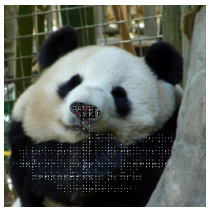

Fig. 6: LocSearchAdv perturbation

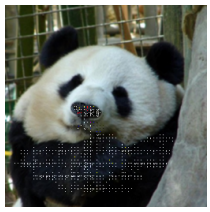

Fig. 7: LocSearchAdv with grid method perturbation

D. Surrogate Attack

Another form of black box attack is the Surrogate attack. Like all our adversarial attacks, the surrogate attack is a means of generating adversarial example images for the target model by perturbing pixels of real images. What sets this attack apart is that it strives to make a white box out of the black box situation so that white box attacks, such as PGD or FGSM, can still be used. Since these white box attacks work by taking feedback from a white box model, a so-called surrogate model is made and trained by the attackers and used as a proxy for the target model when using FGSM or PGD to generate an adversarial example. An adversarial example generated in this way will only be effective if the surrogate model is a good proxy for the target model meaning that its decision boundary is very close to that of the target model. If they are close, we assume it will display the same vulnerabilities and trends as the target model for the same input [11].

The Surrogate model is trained on images with image labels that have been produced by the target model on those images. This way, the surrogate model is trained to mimic the decision boundary of the target model. The accuracy it achieves on this dataset represents the degree to which the decision boundary of the target model has been approximated. Hence, the performance of the surrogate attack correlates directly with the accuracy of the surrogate model on its training dataset.

III. Experimentation

A. Dataset

For our dataset, we chose ImageNet1k ILSVRC 2012 due to its reputation within the field of image classification [12]. ImageNet1k is a database used mainly for object recognition software. Due to its popularity, this allows us to use the dataset across a great variety of models, without many changes being made to the dataset itself. ImageNet1k is divided into three parts: a training set (1.2 million images with labels), a validation set (50,000 images with labels), and a testing set (100,000 images without labels). There are 1000 possible labels.

To gauge the relative effectiveness of various adversarial attacks, we designed a test by which we could benchmark performance. Since the objective of an attack is to reduce the accuracy of the target model, this test is designed to compare the performance of the target model on images before and after a certain attack has perturbed them. To standardize this test, we had to create a dataset of images for the target model to evaluate in unaltered and perturbed form, we've called this dataset the benchmark dataset.

Images in our benchmarking dataset must fulfill two conditions. First, they need labels with which we can evaluate the accuracy of target model predictions. Second, they can not be images that have been used in the training of the target model since the target model accuracy on such images would be disproportionately high. Only the images in the ImageNet1k validation set fulfill these two criteria so we source our benchmark images from there.

We removed monochromatic images as we constructed our benchmark datasets in case of the model's constraints that only allow it to accept three color channels (RGB).

We ended up making two benchmark datasets. A larger one, containing 10000 images, and a smaller one of 1000 images. These were sampled randomly from the 50000 images in ImageNet1k's validation set to conserve the distribution of classes.

B. Target Model

To add another dimension to our experiments, we also want to compare the performances of each adversarial attack across different model structures. We selected our model based on two criteria:

1) Its architecture is still in use for state-of-the-art image classifiers.
2) We can run predictions on it efficiently using our two A100 GPU-equipped lab machines.

Vision Transformer (ViT) is a new classifier architecture that achieves state-of-the-art results [13]. ViT adopts the transformer architecture, commonly used in NLP tasks, for image classification. It treats image patches as tokens and has shown competitive results compared to convolutional neural networks (CNN). Because of its more recent occurrence, ViT is under less adversarial attack research, making it the primary target model of our paper. From many variants of ViT, we decided to proceed using MobileViT-small [14], a comparatively lightweight ViT and CNN hybrid model developed by Apple. We sourced our Apple MobileViT-Small target model from HuggingFace. MobileViT-small has 77.8% top-1 accuracy on our 10k images benchmark.

We also want to compare different attack's performances across different models. So, we chose ResNet-50 with weights pre-trained on ImageNet1k as our secondary model. A Residual Network (ResNet) is a variant of a convolutional neural network (CNN) introduced by a Microsoft research team led by Kaiming [15]. ResNets are known for their "skip connection" feature, allowing them to be considered deep with hundreds or thousands of layers. They are a staple in image classification and have various versions such as ResNet-50, ResNet-101, and ResNet-152, where the number of suffixes represents the depth of the network. Due to our computing resource limitations, we chose the lightest ResNet-50 model as one of our target models, and we only deployed FGSM and PGD attacks on it. Without any perturbation, ResNet-50 achieves 75.2% top-1 accuracy on our 10k images benchmark dataset.

C. FGSM & PGD Experiments and Results Evaluation

1) Random Noise Baseline: FGSM and PGD both take in a hyperparameter epsilon $epsilon$, in which $epsilon$ represents the maximum amount of perturbations allowed on each RGB channel of the original image. So, an epsilon value of 64 represents a maximum value shift of 64 in the range [0, 255] in any single RGB channel. The result we are evaluating is the top-1 attack success rate, calculated as the percentage of images that were classified correctly before the attack but misclassified after. Limited by the time and computational resources, we choose to increment $epsilon$ exponentially from 2^0 to 2^7 to examine a broader range of attack outcomes.

To establish a baseline for assessing the efficacy of FGSM and PGD, we introduced a control experiment utilizing a random noise attack. This approach involves randomly perturbing each pixel of an image with a δ value, sampled from a uniform distribution ranging between zero and the specified ϵ value. Our findings, as shown in Fig. 8-15, indicate that when ϵ is set to 32, the success rate of the attack exceeds 40%, and when ϵ is increased to 64, the success rate surges beyond 85%. Given the destructive impact of larger ϵ values on the success rates of attacks, our analysis subsequently concentrates on smaller ϵ values, particularly those less than 32. This focus allows us to evaluate the relative effectiveness of FGSM and PGD under more subtle adversarial conditions, where the perturbations are

less pronounced and potentially more challenging for models to detect.

Fig. 8: FGSM, ϵ = 1

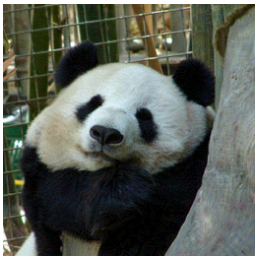

Fig. 9: FGSM, ϵ = 4

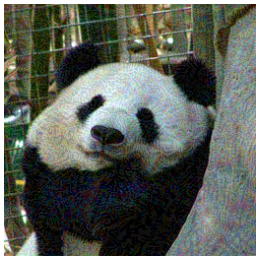

Fig. 10: FGSM, ϵ = 16

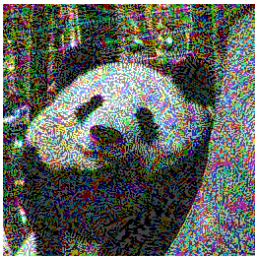

Fig. 11: FGSM, ϵ = 64

Fig. 12: PGD, ϵ = 1

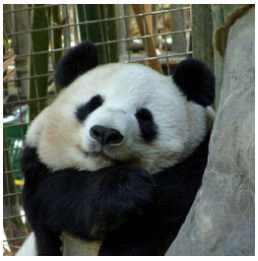

Fig. 13: PGD, ϵ = 4

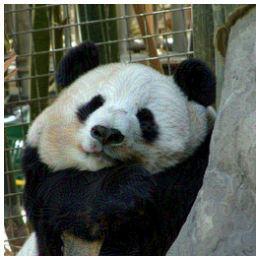

Fig. 14: PGD, ϵ = 16

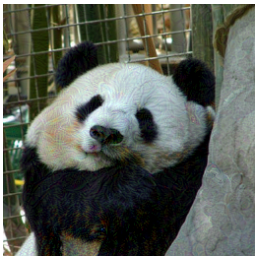

Fig. 15: PGD, ϵ = 64

2) Results & Evaluation: We see that PGD tends to outperform FGSM in scenarios where the model presents a higher degree of non-linearity. This pattern is demonstrated in our experiments with MobileViT, where PGD's success rate supersedes that of FGSM by an average of 30%when ϵ in [1-16]. Several factors can contribute to the more non-linear nature of MobileViT compared with ResNet50: the self-attention mechanism in ViT models contains quadratic components w.r.t. the input and the non-linear softmax function and ViT's use of the Gaussian Error Linear Unit (GeLU), which is more non-linear than the Rectified Linear Unit (ReLU) found in ResNet. Since FGSM guesses the position of the local maximum within the ϵ-ball by taking one step along the current sign direction of the gradient, the more non-linear the loss function is w.r.t. the input image, the less accurate this guess is. This can be seen in Fig. 17.

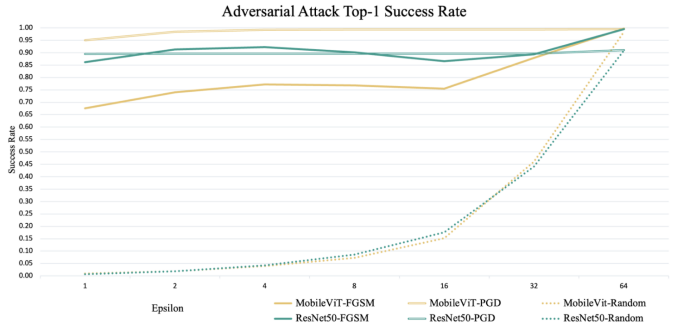

Fig. 16: Top-1 attack success rate for FGSM, PGS, and random noise on ResNet50 and MobileViT w.r.t. exponentially increasing epsilon values

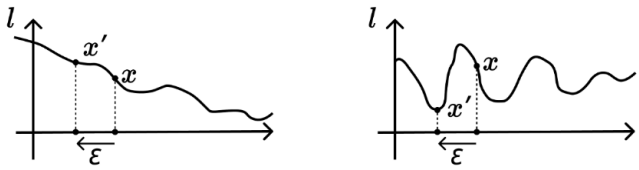

Fig. 17: Left: Example of perturbation using FGSM in a somewhat linear surface; Right: Example of perturbation using FGSM in a more nonlinear surface within the ϵ-ball

On Resnet50, we observed that FGSM outshines PGD within a specific ϵ band—specifically when ϵ lies between 2 and 8. This could be attributed to the existence of multiple local maxima within the adversarial landscape, where the most prominent maxima are situated closer to the epsilon-ball boundary. FGSM is designed to exploit the maximum allowed perturbation range, while PGD may be stuck at a local maximum closer to the original input value. Although this phenomenon applies to MobileViT as well, the overarching effect of non-linearity in this architecture makes maximum less likely to occur on the ϵ-ball boundary, resulting in FGSM's reduced success when compared to PGD.

The performance of FGSM degrades in situations where the epsilon values are not small enough to preserve local linearity, yet not large enough to drastically degrade the image's quality as random noise attack with $\epsilon >= 32$ does. This highlights a critical epsilon range where FGSM's effectiveness is compromised. Conversely, when perturbation magnitude is of lesser concern, FGSM presents a substantial efficiency advantage. Our experimental results indicate that FGSM, being

non-iterative, requires approximately seven times less computational time for perturbing an individual image than PGD, a significant consideration in real-world applications where computational resources and time are constrained.

Focusing on the results for PGD, MobileViT with 5.58 Million parameters is less robust than ResNet50 with more than 25.63 million parameters since more parameters create a more complex landscape for PGD's stochastic gradient descent optimizer to navigate.

D. LocSearchAdv Experiments and Results Evaluation

1) LocSearchAdv Parameter Selection: To better understand how LocSearchAdv works and can be improved, we adjusted the following parameters: neighborhood size, number of pixel attacks per round, and perturbation coefficient r. We modified each of these parameters one at a time to see how they would affect the success rate of our algorithm.

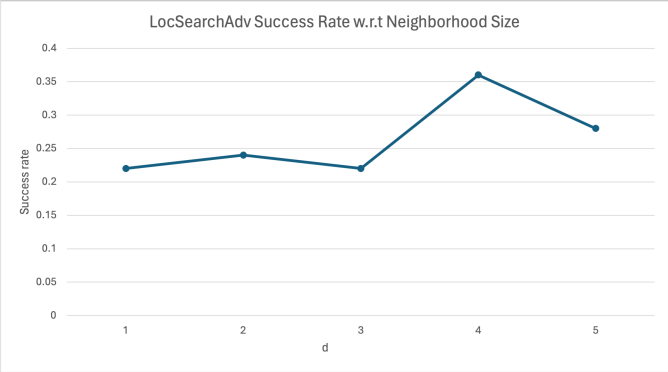

Fig. 18: Illustration of LocSearchAdv Success Rate w.r.t Neighborhood size

Neighborhood size did not have a significant effect on the success rate of image misclassification as evident in Fig. 18. We tried different image d values where d represents the number of pixels deep surrounding the center pixel. We experimented by increasing d in increments of 1 from 1 to 5. The highest success rate occurred when d was 4 with 0.36 and the lowest was when d was 1 and 3 with a success rate of 0.22.

Similarly, changing our pixel perturbation coefficient r did not have a significant impact on the success rate as evident in Fig. 19. We tried running trials where the r value was 0.25, 1.5, 2, 2.5, and 3. The r value that yielded the highest success rate was when $r = 2$ and its corresponding success rate was 0.38. The lowest value of success rate is when r is 0.25 which had a success rate of 0.14 and 3 with a success rate of 0.28. This suggests that the coefficient value does matter and that it matters how much you perturb pixels although there is not a strong relationship between how much you perturb and success rate.

The number of pixel attacks per round refers to the number of individual pixels that in each iteration are perturbed. We ran trials for each of the following values: 1, 5, 10, 25, and 50. We see a positive linear relationship between the number

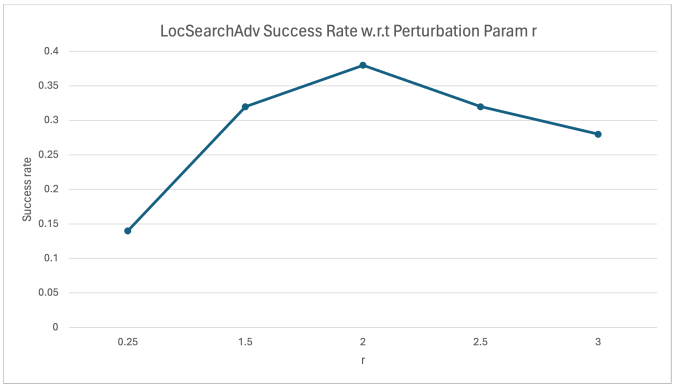

Fig. 19: Illustration of LocSearchAdv Success Rate w.r.t perturbation parameter r

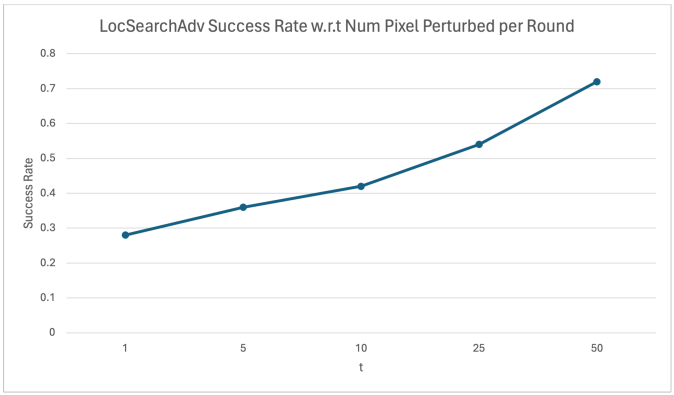

Fig. 20: Illustration of LocSearchAdv Success Rate w.r.t the number of pixel attacks per round

of pixels perturbed per round and the success rate as evident in fig. 20. The largest amount of perturbed pixels (50 pixels) yielded a success rate of 0.72. This is similar to the results of comparing LocSearchAdv grid to LocSearchAdv with no grid, as you perturb a larger number of pixels the success rate increases.

	LocSearchAdv_Grid	LocSearchAdv_Org
Success rate	0.797	0.28
Conf	0.30403	0.33
Average Number of Pixel Perturbed	0.07795	0.00353

2) LocSearchAdv Grid Method vs Original: Running experiments with the same set of the ImageNet dataset, LocSearchAdv with a grid size of five displays clear distinctions between the original LocSearchAdv without any grid implementation. The table above displays the different performances of the two algorithms. The success rate represents the percentage of images that were successfully perturbed. The variable conf in our table represents the average confidence interval of the misclassified class for all the successfully perturbed images. Average pixel perturbed is the average percentage of pixels perturbed for all successfully perturbed images.

The use of the grid in LocSearchAdv perturbs more pixels

at each iteration, resulting in a higher overall percentage of perturbed pixels. Additionally, due to the $256 \times x256$ size of the testing images, the algorithm may not be able to traverse all sections of the images without the use of a grid. This could result in the algorithm missing some of the most sensitive pixels, leading to misclassification. When working with relatively larger images, the original LocSearchAdv depends on randomization's effect at the first iteration. If the randomly selected pixels in the first round contain sensitive pixels that cause misclassification, the algorithm may easily find a successful perturbation. Otherwise, the algorithm may get stuck in a local region of the input image and fail to explore all possible pixels; however, when using a grid, the algorithm can search for a larger region during each iteration, making it easier to identify the most sensitive pixels to perturb.

The original LocSearchAdv can also achieve a higher success rate by perturbing more pixels in total, which can be achieved by increasing the number of pixels perturbed per round and the maximum number of iterations. Nevertheless, LocSearchAdv with the Grid Method is more efficient because it treats sections of pixels as a unit during perturbation. This approach eliminates the need to query each pixel to determine whether it is one of the most sensitive ones. The algorithm identifies sensitive sections in the input images that may lead to misclassification by querying sections of pixels in the grid.

E. Surrogate Attack Experiments and Results Evaluation

1) Implementation: In our implementation, we have assumed the strictest definition of a black box, in which the only access attackers have to the model is using it: getting its predicted class label without confidence after giving it an image.

The images used to train our surrogate model come from ImageNet1k 2012's Testing set. ImageNet1k's Testing Set originally had 100,000 images; however, 85,000 images were left once black-and-white images were removed, all of which were used in the training of our surrogate model. This data fits our purposes well as we could not use the training data of the target model (to maintain a true black box) and we are already using the validation data for attack benchmarking. Fig 21 summarizes the training process.

In our research of high-accuracy image classifiers for ImageNet, we decided upon the VGG16 architecture, which has a 92.7% accuracy on ImageNet 2012's Testing Set. Fig 22 displays the structure of the VGG16.

The VGG architecture is known for its 16 learnable parameter layers. Its preprocessing procedure includes subtracting the mean RGB of the training set from each pixel in the images. The convolutional layers of the VGG16 model are unique in the sense that they are 3×3. Two 3×3 layers with no spatial pooling in between are considered to be equivalent to one 5x5 layer. The advantage of having multiple smaller layers is that there is less number of parameters being considered at each layer, creating a higher level of distinction. The occasional 1x1 convolutional filters add linear transformations to the input. This counteracts any effect of non-linearity from the input

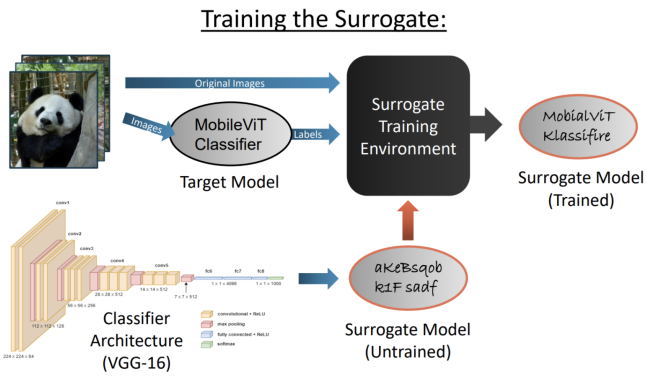

Fig. 21: Illustration of the processes of training the surrogate model

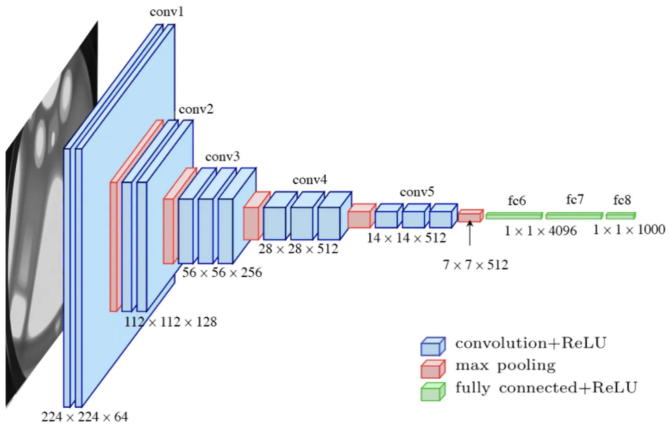

Fig. 22: VGG16 Architecture of its 21 Layers, 16 of which are learnable parameter layers

which is debated to be one of the original causes of adversarial examples. is useful in increasing non-linearity [9]. All the hidden layers have ReLU and there is no local normalization done throughout [16].

We started to implement the exact architecture in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" by using the VGG16 function in Kersas [16]. However, we did not see much success in this process. We shifted our focus to implementing the VGG16 structure from scratch and the specific tutorial that our final code has comes from "Writing VGG from Scratch in PyTorch" [17]. In this implementation, the images are resized to 227x227 pixels before they go through the convolutional neural network, which is the main distinction from the standard VGG16 architecture.

After running 30 epochs, we reached an accuracy of 35%, which took approximately 30 hours to run. Here, accuracy is calculated with respect to the target model's label, ie. we achieved a 35% match to the target model's decision boundary. This version of the model is what we used to run our black box attacks on as we had to proceed to the benchmarking phase.

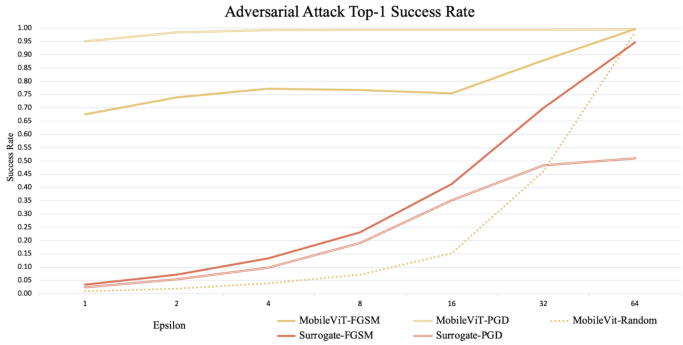

Fig. 23: Shows the success rate of our surrogate attack w.r.t. its white-box counterparts

2) Results: From Fig. 23, we can see that the surrogate attack performed rather poorly overall. Using FGSM the attack consistently outperforms random perturbations but lies far below the white box rates. With PGD, the surrogate model performs worse, with the success rate leveling off at about 50% at higher epsilon values. The primary reason for this poor performance is that our surrogate model never achieved high accuracy in mimicking the target model's decision boundary. This meant that FGSM and PGD created perturbations using a loss function (the surrogate's) that was far from that of the target model and hence would not be particularly effective. However, it is clear that even with only a 35% accuracy from the surrogate model, the surrogate attack outperforms random noise in the lower epsilon range $[1, 16]$.

3) Surrogate FGSM vs Surrogate PGD: The Surrogate PGD attack did particularly poorly as PGD finds local minima rather than relying on only the general shape of the loss function. Since our surrogate did not match the target model closely, the two loss functions likely shared few local minima for PGD to exploit. This explains the leveling off of the surrogate PGD curve as PGD gets stuck in the local minima of the surrogate loss function which may not have counterparts in the target loss function.

In contrast, the curve of the surrogate FGSM attack's success rate does not level off of this sort. At 35%, training of the surrogate model did achieve a significant degree of similarity between the two loss functions. This similarity, though not apparent at the local minima level, is present and exploitable on the larger-scale shape of the loss functions. FGSM relies on the large-scale linearity of the loss functions and this is likely the reason FGSM outperformed PGD with the surrogate model. This is a reversal from PGD outperforming FGSM in white box situations because PGD's precise potency is aimed, inefficiently, at the surrogate model instead of at the target model.

IV. Discussion

A. Save Perturbation to Image

When considering the practical application of adversarial attacks on deployed models, the perturbations crafted by different methods must be stored in image file formats, which introduces several challenges that can potentially diminish the effectiveness of the attacks. One notable factor is the use of the JPEG compression format, which is known to degrade image quality due to its lossy nature. Studies, such as one conducted by Kurakin et al. in 2017, have demonstrated that this compression can lead to a significant decrease in the success rate of adversarial attacks post-conversion [18]. Furthermore, the process of converting perturbations from their original float representations into standard RGB integer values can also slightly impact the attack's success rate, typically around 1%. This loss, albeit relatively small, is non-negligible when aiming for optimal attack performance.

Additionally, modifications to the preprocessing steps utilized by the target model can influence the efficacy of the attack. If a model employs different preprocessing methods, such as resizing, cropping, or skewing, these transformations can alter the effect of the adversarial perturbations, thereby impacting the success rate of the attack. It is important to note, however, that quantifying the exact impact of these preprocessing changes on the success rate for each adversarial method is beyond the scope of our current paper.

B. Perceivable Perturbations

Another evaluation that one might be concerned about in real-world applications is how perceivable the perturbations are. Depending on how the perturbations are generated for each attack method, it leads to different visual effects. For example, because FGSM universally applies a change of fixed step size (ϵ) to each pixel, the visual effect of that looks like having noise masking on the original image and the perturbation becomes more and more visible as the ϵ-value increases. On the other hand, since the PGD attacks use ϵ-value as the range to search a local maximum, the actual perturbation usually falls smaller than this ϵ-value resulting in less visible perturbation given the same image and the same ϵ-value in comparison to FGSM. Specifically, the average perturbation value for PGD is 0.037 out of 255. Additionally, LocSearchAdv uses perturbations that visually resemble adding color blocks to the original image, which is a different visual effect than applying a perturbation to each pixel individually. This sort of perturbation can help determine how to create adversarial examples in the real world on the subject of these images, for example putting tape on a stop sign to have it classified as a speed limit sign.

Given more time, we might be able to conduct a survey that asks participants to rate how perceivable the perturbations are on some sample images. However, depending on the context of the viewer (e.g. taking a glance at a sign on the sidewalk or admiring a painting in an art gallery), it is a subjective measurement to assess if a kind of perturbation is perceivable or not, and the visual differences between each image add on more complexities to this measurement.

C. Surrogate Limitations and Potential Improvements

Our surrogate model achieved a 35% accuracy on its training set, meaning that it only matched the decision boundary of the target model to that degree. This could have been improved in a few different ways.

1) With more time and computational resources, we could have explored more potential classifier architectures, run longer training cycles, and better optimized training hyperparameters.
2) Our dataset of 85,000 images is a rather small one for this sort of training, the VGG paper trained on 1.5 million images.

We also suspect that one of the major reasons for this attack's poor performance has to do with the architecture of the target model, specifically the image size that it takes in. MobileViT takes in images at 288x288 and crops into the center 256x256 pixels. Our surrogate model rescaled images to 227x227 and did no cropping. This smaller rescaling introduces detail loss when downsizing images and blurring to the perturbations when scaling back up for the attack on the Target model. These two rescalings significantly impact performance.

If there is a way to rescale images to the same size as the target mode, we can avoid this rescaling problem. The method we conceived of for determining the correct rescaling size is this:

1) Take a small set of images for which the target model has 100% accuracy.
2) Resize this dataset to a range of different sizes and observe the change in target model accuracy on the rescaled images.

Once the plateau is reached, a rescaled size where larger rescalings yield no further accuracy improvement is found. Hence, you can be sure that a surrogate model that uses this rescaling factor will lose minimal data and perturbation effectiveness due to rescaling.

Our implementation of this test yielded convincing results. We did this with 1000 images which we rescaled from size 50x50 to 550x550. Results are visualized in the Fig. 24.

Fig 24 shows a plateau flattening around our correct rescaling factor of 288 (for MobileViT). Due to differences in scaling methods, this does not plateau to 100% accuracy, but it distinctly shows how rescaling the image will affect target model accuracy. An attacker, equipped with these results, should choose a surrogate model architecture that utilizes an image size between 285 and 320.

Another branch to explore is the small peak near size 288. Across our small set of experiments, we found there to be a small range around the correct rescaling size in which accuracy had a greater variance. We have not yet proven a correlation here, nor do we have a hypothesis to explain this behavior. However, further experimentation might prove that this is a more accurate and efficient means of identifying the correct rescaling size.

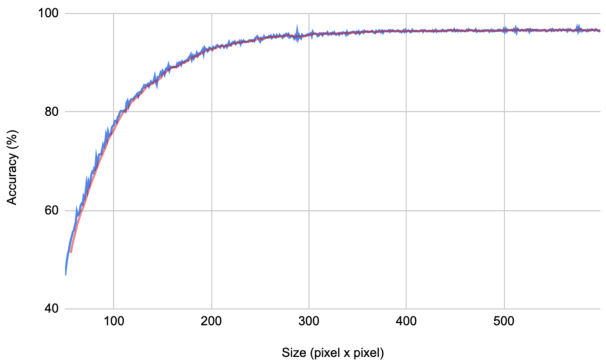

Fig. 24: Accuracy (%: percentage of images where the predicted label is equivalent to true label) in relation to the image size)

If we had more time, another possible implementation would be to do multi-scale training. For multi-scale training, each training image would be randomly resized in a size range between the smallest and largest possible rescaled size of the images, which is set by the developers. This allows for the model to recognize images regardless of scale, without lowering accuracy [16].

Though the VGG16's architecture depth makes it ideal for larger datasets, our current training set may not be big enough. To increase the number of training images, we can do a process similar to that mentioned in "Very Deep Convolutional Networks for Large-Scale Image Recognition," where they randomly horizontally flipped images and did random RGB shifts. This way we can double the size of our training image set. Sourcing images from other sources for the training dataset is also possible but would impact the distribution of the dataset so these would have to be selected such that this distribution is conserved.

D. Surrogate Potential

These results do not fairly represent the upper bound of the performance a surrogate attack can achieve. This poor performance can be attributed to the number of problems with our initial approach (as mentioned above). A better-designed surrogate model, built with more time, more computational resources, and the knowledge we have now gathered has the potential for much higher success rates.

A better performing surrogate model could be an effective alternative to query-based methods such as LocSearchAdv as it produces more subtle perturbations but the preparation required in designing and training the surrogate model may make this method too time and resource intensive to be viable if this is not a priority.

E. Defense Strategies

Thus far in this paper, we have examined different Adversarial Attacks and their performance. Successful implementation of these attacks can have significant consequences, thus it is

important to investigate defense strategies. Our research of defense against adversarial attacks can be broadly split into two categories: adversarial training and modifications to the target neural networks.

Adversarial training is a defense strategy that can be employed against attacks. When an image classifier fails, it is because it does not know how to identify the image. In the case of adversarial attacks, the model has never seen the perturbed images that "confuse" it such that it does not know what to do. One solution to this as described by Zhao, Alwidian, and Mahmoud in their paper Adversarial Training Methods for Deep Learning: A Systematic Review is training models on adversarial examples [19]. They argue that Adversarial training is one of the most promising approaches to combating attacks. This is done through training the model to generalize adversarial examples as well as clean images. It is an iterative process, where an adversarial example is created, run through the model, and then the model retrains and updates.

Adding elements of randomness to the neural network is another way to reduce the effectiveness of Adversarial Attacks and change the neural network. By adding some randomness into the neural network, adversarial attacks have a harder time learning the specific model patterns to exploit. Randomization layers are one of these strategies. The first randomization layer is the resizing layer. It takes the original image with dimension $W \times H \times 3$ and rescales it to size $W' \times H' \times 3$. We want our $|W - W'|$ and $|H - H'|$ to be within a small range so that the performance on clean images does not decrease too much [20]. According to research by Cihang Xie, Zhishuai Zhang, and Alan L. Yuille, the ideal range for images of input size $299 \times 299 \times 3$ is within the range of $[299, 331]$. The second randomization layer uses random padding. This means that the resized images get (within a random range) a number of 0's on each side. If we imagine our image size with padding to be $W'' \times H'' \times 3$ and we pad with w 0's on the left and right and h pixels above and below the image we have $(W'' - W - +1) \times (H'' - H' + 1)$ different possible padding configurations. This defends against iterative-based adversarial attacks like LocSearchAdv.

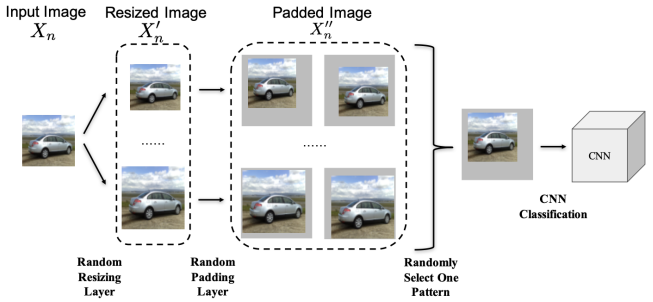

Fig. 25: "The pipeline of our randomization-based defense mechanism" (Xie. et al 2018)

Another way to defend against attacks while using randomness is through stochastic activation pruning (SAP) [21]. The way this defense system works is that in the forward pass of a model's prediction, we stochastically prune a random subset of activation in each layer. We preferentially retain activations with larger magnitudes which means we are pruning the activations with smaller magnitudes. The remaining activations are then scaled up to normalize the range of the next layer. One reason that SAP is a good defense is that it can be applied post-hoc to pre-trained networks and does not require additional fine-tuning.

The defense strategies we have discussed are by no means exhaustive. They are a small sample of avenues of future exploration. Studying defense mechanisms is important because it helps us understand the nuances of these attacks and how different strategies work for different models. In this way, we can minimize some of the impacts of successful adversarial attacks in real-world situations.

F. Ethics

In addition to defense strategies, the ethical implications of these attacks need to be acknowledged as the field of adversarial machine learning expands. For instance, there is a chance that models are involved in data that has unfair patterns. For example, many facial recognition models are trained on datasets that contain mostly white men, which means that the model is biased against people who are not white men.

Adversarial ML researchers always assume that they are trying to make models more robust through their research and that all attacks come from ill-intentioned researchers. However, there may be cases in which the attack is attempted for good [22].

This also brings into question "Is there a way to do ethical adversarial attacks?". Adversarial attacks are usually done by attackers with the wrong intention. However, this poses the question of when the attackers have a good intention, are they allowed to attack the ML system however they want? The instance can be defined as an Ethical Adversarial Attack (EAA). For instance, are cybersecurity experts allowed to attack malicious systems to maintain security and privacy? Yet, this also brings in the bigger question of what is allowed in the field of AI for crime, which starts at the foundation question of what it means to have AI for good. Does only the intention matter or does only the process or does both, meaning are the people who are trying to do good (like the cybersecurity experts) allowed to do something "bad" (such as attacking models) in their process of doing good? It also brings into question when an attack starts being considered as security. And if an attack is considered security, how do we efficiently monitor and control its usage [23]?

The diversity of the field reflects issues of the field. For instance, the participants in a physical test must also be diverse so that the participants pool is equivalent to the actual users of the ML system. There needs to be an increase in the number of participants who are people of color, different sexuality, various jobs, and especially the intersection of the identities. Diversity of the participant pool should also include anthropometric features such as body shape, mobility

and vision, and hearing abilities. By increasing diversity, the testing environment has a greater number of perspectives. This also calls for increased inclusivity within the field, where the researchers are also diverse [24].

V. Conclusion

It is hard to definitively say that one of the adversarial attacks we investigated outperforms all the others. Depending on the amount of access one has to the system, one might decide between a white box or a black box attack. Furthermore, the degree to which an image may be perturbed is another factor of consideration when choosing an algorithm for your specific needs. For example, LocSearchAdv does not require a deep level of machine learning knowledge to understand; however, it is quite computationally intensive and has noticeable image perturbations. A surrogate attack is comparatively less detectable to the human eye, but requires intensive setup to reach useful levels of effectiveness. On the white box side, PGD can generate extremely successful adversarial examples with perturbations that are undetectable to humans, but it is far slower than FGSM, which creates only slightly less potent perturbations. Other considerations like the desired misclassification rate or new classification class may also factor into algorithm choice. There is no one-size-fits-all choice in deciding on what attack method to use as an attacker or defend against as a model developer, but it is clear a broad understanding of these different attacks is a useful resource, regardless of which side of this fight we are on.

References

[1] "Artificial intelligence: Adversarial machine learning — nccoe. (n. d. -a)," https://www.nccoe.nist.gov/ai/adversarial-machine-learning, accessed: 2024-03-06.

[2] R. R. Wiyatno, A. Xu, O. Dia, and A. de Berker, "Adversarial examples in modern machine learning: A review," 2019.

[3] W. Brendel, J. Rauber, and M. Bethge, "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models," 2018.

[4] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial machine learning at scale," 2017.

[5] S. Kaviani, K. J. Han, and I. Sohn, "Adversarial attacks and defenses on ai in medical imaging informatics: A survey," *Expert Systems with Applications*, vol. 198, p. 116815, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S095741742200272X

[6] B. Yang, Z. Jin, Y. Cheng, X. Ji, and W. Xu, "Adversarial robustness analysis of lidar-included models in autonomous driving," *High-Confidence Computing*, vol. 4, no. 1, p. 100203, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667295224000060

[7] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, p. 317–331, Dec. 2018. [Online]. Available: http://dx.doi.org/10.1016/j.patcog.2018.07.023

[8] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, ser. AISec '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 43–58. [Online]. Available: https://doi.org/10.1145/2046684.2046692

[9] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[10] N. Narodytska and S. P. Kasiviswanathan, "Simple black-box adversarial perturbations for deep networks," 2016.

[11] N. A. Lord, R. Mueller, and L. Bertinetto, "Attacking deep networks with surrogate-based adversarial black-box methods is easy," 2022.

[12] "Imagenet. (n.d.)." https://image-net.org/challenges/LSVRC/2013/index, accessed: 2024-03-12.

[13] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[14] S. Mehta and M. Rastegari, "Mobilevit: Light-weight, general-purpose, and mobile-friendly vision transformer," 2022.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[17] "Writing vgg from scratch in pytorch. (2022, june 2). paperspace blog." https://blog.paperspace.com/vgg-from-scratch-pytorch/, accessed: 2024-03-11.

[18] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," 2017.

[19] W. Zhao, S. Alwidian, and Q. H. Mahmoud, "Adversarial training methods for deep learning: A systematic review," *Algorithms*, vol. 15, no. 8, 2022. [Online]. Available: https://www.mdpi.com/1999-4893/15/8/283

[20] C. Xie, J. Wang, Z. Zhang, Z. Ren, and A. Yuille, "Mitigating adversarial effects through randomization," 2018.

[21] G. S. Dhillon, K. Azizzadenesheli, Z. C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, and A. Anandkumar, "Stochastic activation pruning for robust adversarial defense," 2018.

[22] K. Albert, M. Delano, B. Kulynych, and R. S. S. Kumar, "Adversarial for good? how the adversarial ml community's values impede socially beneficial uses of attacks," 2021.

[23] M. Choraś and M. Woźniak, "The double-edged sword of ai: Ethical adversarial attacks to counter artificial intelligence for crime," *AI and Ethics*, vol. 2, no. 4, p. 631–634, Nov. 2022. [Online]. Available: https://link.springer.com/10.1007/s43681-021-00113-9

[24] K. Albert, M. Delano, J. Penney, A. Rigot, and R. S. S. Kumar, "Ethical testing in the real world: Evaluating physical testing of adversarial machine learning," 2020.