CSDS 440 Class Project:
Adversarial Machine Learning

Shaochen (Henry) Zhong, sxz517
Minyang Tie, mxt497
Alex Useloff, adu3
Austin Keppers, agk51
David Meshnick, dcm101

Due and submitted on 12/04/2020
Fall 2020, Dr. Ray

Contents

1 Introduction and Significance **3**

2 Individual Reports **5**
2.1 Shaochen (Henry) Zhong's Individual Report . . . 5
2.1.1 Overview . . . 5
2.1.2 Fast Gradient Sign Method . . . 5
Algorithm Intuition . . . 5
Algorithm Implementation . . . 7
Experiments and Evaluation . . . 8
2.1.3 Hop Skip Jump . . . 9
Algorithm Intuition . . . 9
Algorithm Implementation . . . 10
Experiments and Evaluation . . . 10
2.1.4 Feature Collision . . . 11
2.2 Minyang Tie's Individual Report . . . 12
2.3 Alex Useloff's Individual Report . . . 12
2.4 Austin Keppers' Individual Report . . . 12
2.5 David Meshnick's Individual Report . . . 14

3 Comparative Study and Discussion **14**
3.1 Overview . . . 14
3.1.1 Datasets and Sample Selections . . . 14
3.1.2 ART . . . 15
3.1.3 Metrics . . . 15
3.1.4 Adversarial Rivalry . . . 16
3.2 Attack Algorithms . . . 17
3.2.1 Evasion . . . 18

3.2.2 Poisoning 21
3.2.3 Conclusion 21
3.3 Defense Algorithms 22
3.3.1 Detector 22
3.3.2 Pre-processor 24
3.3.3 Transformer 26
3.3.4 Conclusion 26

4 References **26**

1 Introduction and Significance

In the field of machine learning, it if often taken for granted that the testing examples have no malicious intent – as it if something is labled to be a dog, it will indeed look like a dog. However, with the growing popularity of machine learning, robustness against adversarial attacks has been a more and more important metric. Adversarial machine learning is the field studying how to attack a model to make it output incorrect results (e.g., false predictions) in different stage of the model building, and how to make a model more robust against various kinds of attacks.

Here is a classic example of a successful adversarial attack borrowed from Google [8]: by applying the middle pertubation to the original image, a supposely Labrador Retriever, while still looks like a Labrador Retriever, is now misclassified as a Weimaraner.

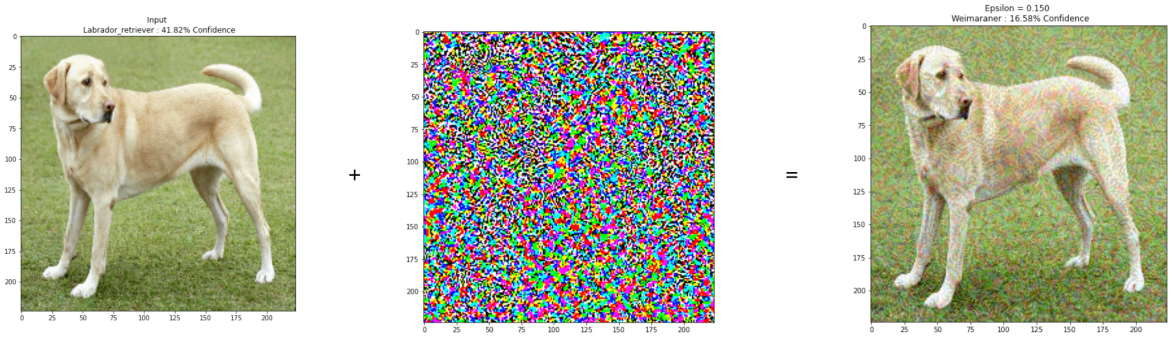

Figure 1: An example of adversarial pertubation resulting successful adversarial attack

One explaination[9] of why adversarial attack works is because there is a natural distinction between how we read and how machine read into a piece of information. When we human are asked to classify between a *dog* from a *cat*, we know to focus on the ears and snout:

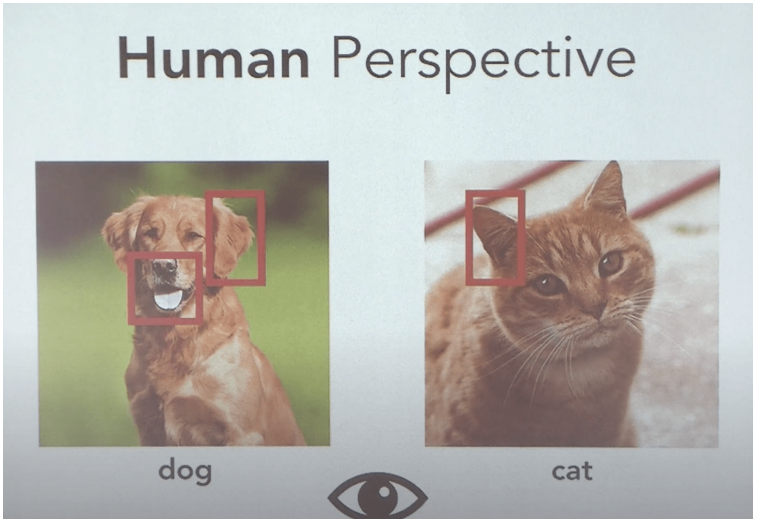

Figure 2: Human perspective

and we considered the pertubation in [Figure 1] to be meaningless because we cannot digest useful information out of it. But that might not be the case to the "eyes" of a machine as it has no knowledge about cats and dogs. To "translate" the machine's perspective to "human terms," how machine look at these cat/dig pictures are probably like the following:

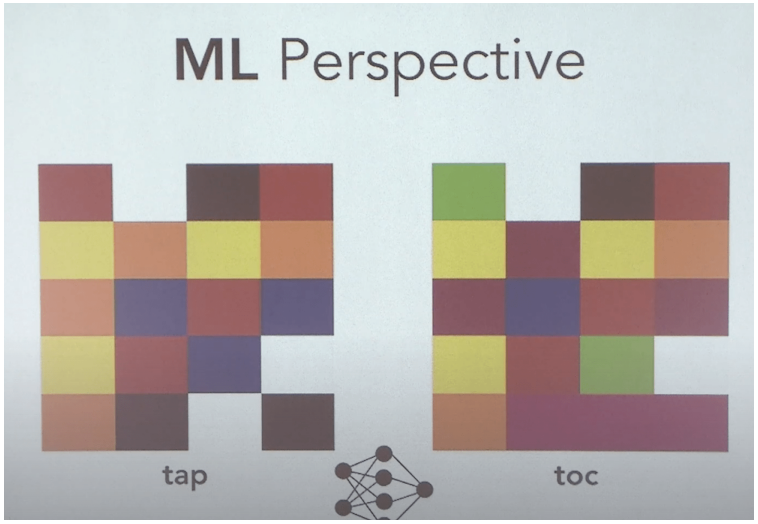

Figure 3: Machine perspective

In fact, a machine might considers the pertubation in [Figure 1] to be more "informative" than the features we care about (ears and snout). And since models are set to maximize accuracy as a general goal, it will utilize both features – both the *robust features* (features that keep being robust after adversarial pertubation), and the *non-robust features* (e.g., some "noise" to human):

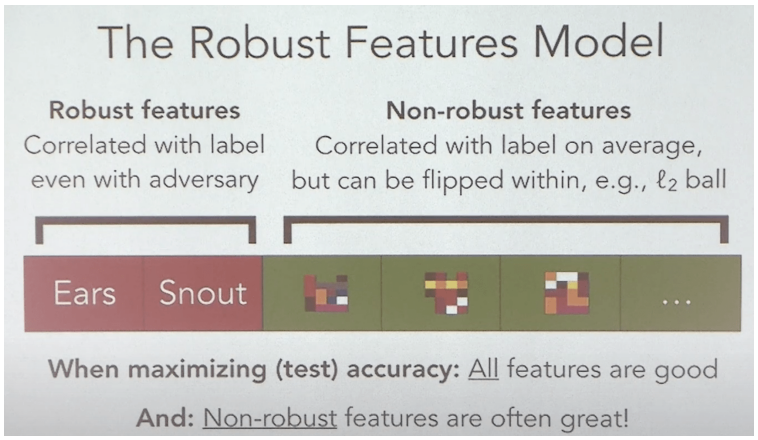

Figure 4: Robust and non-robust features

Thus, if a model's decision is largly base on these non-robust features, we may apply adversarial pertubations to adjust these features and causing the model to output false results.

For this paper in particular, we will look into *evasion attack* (causing the model to output false result by modifying the input), *poisoning attack* (alter the training set of a model to make it produce inaccruate decision boundaries), and some corresponding defense methods against these attacks.

2 Individual Reports

2.1 Shaochen (Henry) Zhong's Individual Report

2.1.1 Overview

I have wholeheartedly lead and contributed to this group project, below is an itemized list of my contributions. I have inquired Dr. Ray and confirmed these can be considered as "extra works" and maybe give me some grade boost. Thanks :)

- Read 3 papers on algorithms.
- Implemented 2 algorithms (`FGSM` and `Hop Skip Jump`) with 3 extensions.
- Piplined attack algorithms to work with 2 datasets, collected almost all (7 algorithms/useable extensions) attack experiments data (except `backdoor` and `one pixel attack`) for the comparative evaluations.
- Piplined and collected experiments data for `FGSM, Hop Skip Jump, DeepFool` attacks (and their useable extensions) against `Detector` and `Spatial Smoothing` defenses on 2 datasets.
- Implemented L_2 and L_∞ *perturbation budget* to aid comparative evaluation.
- Wrote **Introduction and Significance** section.
- Plotted all graphs and charts in **Comparative Study and Discussion**.
- Helped group move forward by making technical decisions, distributing works, setting up deadlines, and facilitating coordination between groupmates.

2.1.2 `Fast Gradient Sign Method`

Algorithm Intuition `FGSM`[7] is a *white-box* evasion attack algorithm. Being white-box means the attacker is assumed to have access to the internal of the model: the structure, the parameters... basically each and every details of the model is considered to be known.

In this case, we mostly care about the `loss` function of the model. The intuition of `FGSM` is elegent and effective – in short: gradient ascent. Assume we have an benign example x with label y, and we are trying to make it adversarial by letting the model classify x_{adv} to be not y. We apply the following pertubation on x to achieve an x_{adv}

$$x_{\text{adv}} = x + \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y)) \tag{1}$$

where J is the `loss` function and θ is the parameters of the model. It is clearly to tell that by finding a proper amount of ϵ, we know x_{adv} will eventually cross the y and $\neg y$ decision boundary and be classified as a $\neg y$ object. We know this direction will lead to a $\neg y$ space because a model is designed to minimize the `loss` – thus it is always doing gradient descent – and by going against the gradient to do gradient ascent, it will "maximize" the `loss` and eventually be misclassified. And if the value of ϵ is small enough, the x_{adv} will sill perserve enough semantic from x and thus relatively indistinguishable to human eyes. Like the following exmaple in [Figure 5]:

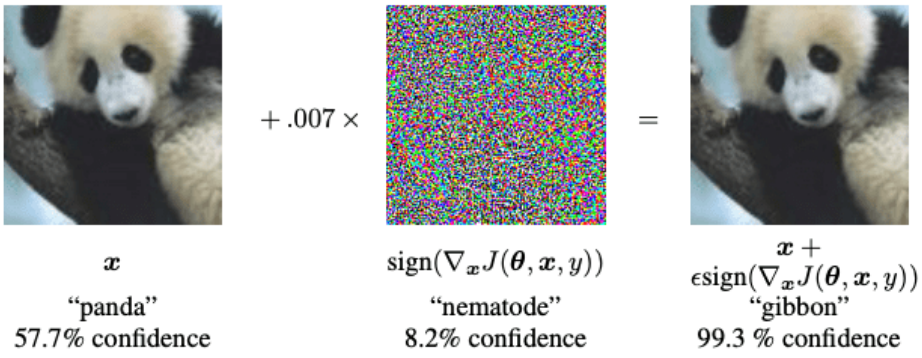

Figure 5: Applying gradient ascent pertubation to a panda image, $\epsilon = 0.07$

Although the x_{adv} on the RHS has been adversarially altered, because the distance of pertubation is small enough, it sill looks like a panda – a.k.a. perserving the majority semantic of x.

Extension 1: `Targeted Fast Gradient Sign Method` As standard `FGSM`, assuming implemented on x with y-label, only cares about reaching a space of $\neg y$. This can be undesired for certain application. One scenario maybe is to attack the OCR system of bank checks (which is usually the goal when attacking *MINIST*), the attacker would want a lower valued number to be recognized as a higher valued number (e.g., `0 -> 9`), but not vice versa. Thus, I have implemented a targeted version of `FGSM`, where instead of doing gradient ascent to y like in [Equation 1], it does gradient descent towards the label of desired target.

Say we have a target of x_t with label y_t, the mathematical intuition of `T-FGSM` is:

$$x_{\text{adv}} = x - \epsilon \cdot \text{sign}(\nabla_x J(\theta, x, y_t)) \tag{2}$$

Note we are back to standard gradient descent like running a normal model. This is because we want to minimize the `loss` of x_{adv} with label y_t – which is not guaranteed by randoming doing gradient ascent out of y as that might not be the right direction.

Extension 2: `Iterative Fast Gradient Sign Method` One major drawback of traditional `FGSM` is to decide the ϵ value in [Equation 1]. Intuitively, such ϵ can't be too small, otherwise the applied pertubation will be too small to make x_{adv} across the y to $\neg y$ decision boundary. However, it also can't be too large as it will lose the semantic of x, like in the follwing example we applied an $\epsilon = 30$ to the Labrador Retriever in [Figure 1] and this is just pure noise:

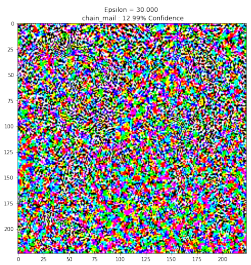

Figure 6: Perturbed Labrador Retriever with $\epsilon = 30$

In this case, although [Figure 6] is still an successful adversarial attack to the machine (as it was classified as *Chain Mail* for some reasons). The attack is by essence meaningless as it has lost all semantics of a Labrador Retriever – and we might as well just swap it with an actual chain mail picture and call it a successful attack.

More important, depending on the decision boundaries of a model, some value of ϵ, altough being large enough, might also fail the attack. Consider a model with the following decision boundary:

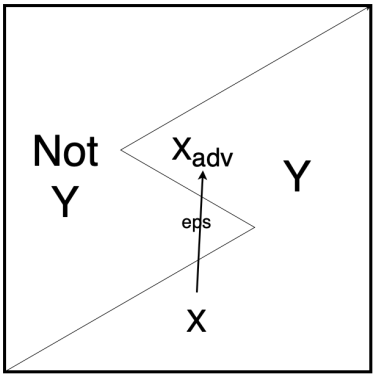

Figure 7: Potential ϵ overshoot

As demoed in [Figure 6], with an unfortunate ϵ, even being large, an attack can still fail. Thus I took advantage of doing a *white-box* attack and implemented an iterative version of `FGSM`. As instead of taking an *one-shot* pertubation, it takes one and another small pertubations to reach to a decision boundary; once the prediction is changed, it stops applying further pertubations.

The mathematical intuition of this algorithm will be like [Equation 3]

$$x_{\text{adv}_{i+1}} = x_{\text{adv}_i} + \epsilon_{\text{step}} \cdot \text{sign}(\nabla_x J(\theta, x_{\text{adv}_i}, y)) \tag{3}$$

where $\epsilon_{\text{step}} \ll \epsilon$, and we can hard bound the accumulation of ϵ_{step} to be less or equal to ϵ so the algorithm try to perturb an example which is far away from its decision boundary and running forever. And since the final x_{adv} will be right on the decision boundary (granted a small enough ϵ_{step} was used), it will perserve a healthy amount of semantics of x.

Extension 3: `Iterative Targeted Fast Gradient Sign Method` Naturally, I combined my two implementation together. [Equation 4] is the mathematical intuition of the algorithm:

$$x_{\text{adv}_{i+1}} = x_{\text{adv}_i} - \epsilon_{\text{step}} \cdot \text{sign}(\nabla_x J(\theta, x_{\text{adv}_i}, y_t)) \tag{4}$$

Algorithm Implementation I implemented `FGSM` and its three extentions with the help of `ART`[4] (which has dependency to `Tensorflow` and `Keras`). It is a `Python`-based open-source libarary that taken care of the engineering aspects of adversarial machine learning testing. Specifically, I used `ART`'s `KerasClassifier` to wrap around a pretrained model of *MINIST* and *CIFAR-10* as my victim model – I can do this because all my implemented algorithms and extensions is attacking a trained model, so how the model was trained is not in my interest of evaluation.

I first gather the `x_test` sample set from my dataset, then I feed into the pretained model to get *benign acc* reading. Then I apply the below actiovation methods (by choice)

```
attacker = FastGradientSignMethod(classifier, eps=0.3, batch_size = 32)
x_test_adv = attacker.generate(x_test[:num]) # non-targeted
x_test_adv = attacker.generate_targeted(x_test[:num], x_test[0]) # targeted
x_test_adv = attacker.generate_iterative(x_test[:num]) # iterative non-targeted
x_test_adv = attacker.generate_targeted_iterative(x_test[:num], x_test[0]) # iterative
    targeted
```

to generate the adversarially perturbed `x_test_adv` from `x_test`. I then feed `x_test_adv` into the pretained model and collect the *adversarial acc* reading.

In the meantime, I collect L_2 and L_∞ distances between a benign and an adversarial images (calculated by channel) to be a mertrics of quantifying *pertubation budget*. I also implemented a runtime counter to register how long it took an attack method to run – you may see them in my following Section 2.1.2 and comparative study sections like 3.2.

Experiments and Evaluation Since I have implemented multiple extensions and `FGSM` have several params to tone with, I have did a vast amount of experiments. We later decided to go with `eps = 0.3, eps_step = 0.05`

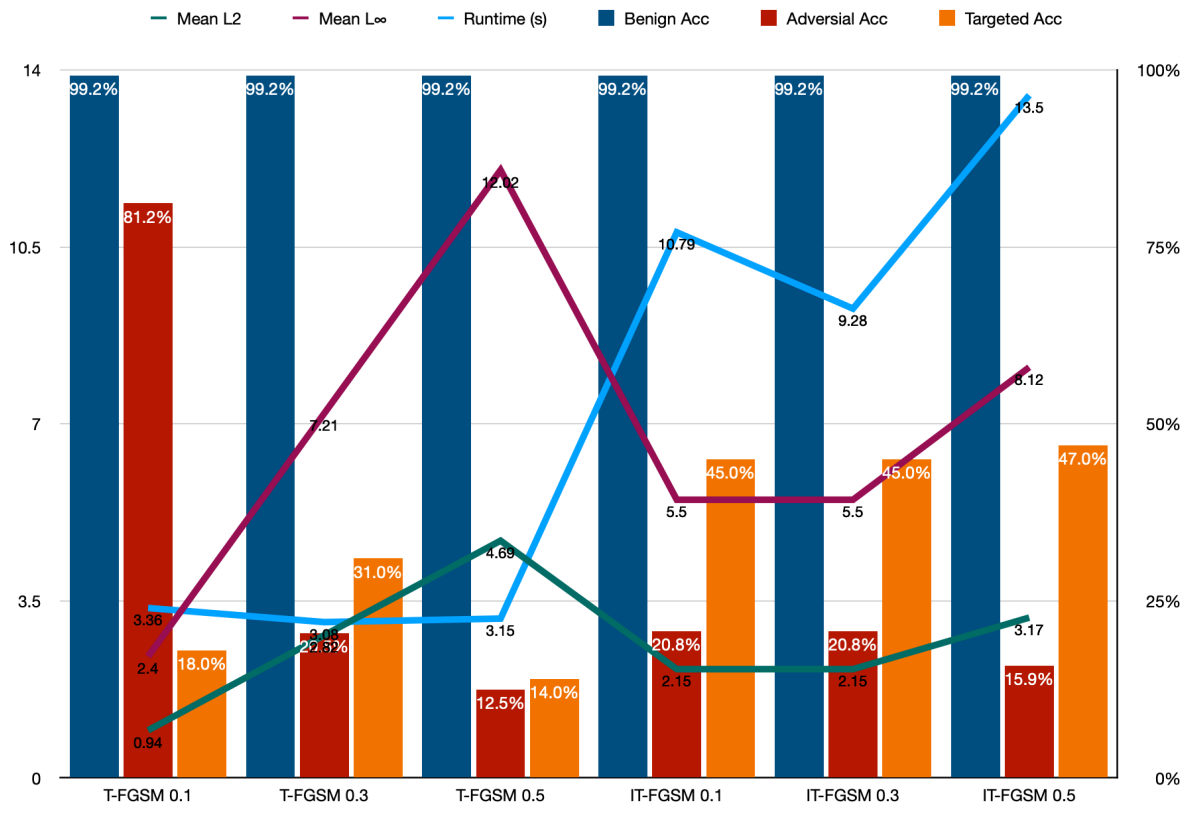

Figure 8: `T-FGSM` and `IT-FGSM` with $\epsilon = \{0.1, 0.3, 0.5\}$ on *MINIST*

because in [Figure 6], we may observe the mentioned "overshooting" problem as in `T-FGSM` the *Targeted Acc* is decreasing with ϵ increasing. However, this is sloved by implementing `IT-FGSM`, the *Targeted Acc* is growing with the ϵ.

Also we noticed we ended with a much better *pertubation budget* in `IT-FGSM`. This is because the algorithm stop pertubation once it reached the target label, thus remain a smaller and therefore preferred L_2 and L_∞. But we do realize even in `IT-FGSM`, the L_2 and L_∞ is increasing with $\epsilon = 0.3 \rightarrow 0.5$, this is because with a bigger ϵ the algorithm is trying to move picture that are relatively far away from the target to be the target – this is undesired the resulted images will likely losing semantics, and it cost more computing power to do so (which is also reflected by the *runtime* data, as it is significantly slower in $\epsilon = 0.5$ than $\epsilon = 0.3$). Thus, we set the this to be the parameters of `FGSM`-related algorithms in comparative studies.

2.1.3 `Hop Skip Jump`

`HSJ`[2] is another evasion attack algorithm I implemented. However, it is designed to be a *black-box* attack method, as the attacker don't have access to the internal structure of the model, and the attack is therefore conducted base on model's output (it will be better if the model can also output confidence leval, but it work fine with just signs).

Algorithm Intuition With the fundamentals introduced, we may jump into the mathematical intuition of `HSJ`. Note this is a brief walk through of how `HSJ` work, but not necessarily *"why"* as the original paper used many theorem, which we won't go through here.

$$\begin{aligned} S(x') &= max_{c \neq C(x)} F(x')c - F(x')_{C(x)} \\ S(x') > 0 &\Leftrightarrow \arg\max_c F(x') \neq C(x) \end{aligned} \tag{5}$$

where $S(x')$ is the decision boundary of the example x', $F(x')$ is the probability of x' class, and $C(x)$ is the current class. This is saying the model on $C(x)$ is looking at different classes around itself (different x'(s)), and take the one different class with with maximum likelyhood. Let $S(x')$ represent a succesuflly attack, we will then try to minimize the distance between $d(x', x)$ while making $S(x') > 0$.

This setup is very similar to a targeted `FGSM`. So natually, we want to get to the boundary of the x and x'. However with no access to `loss` function, what we can do is to do a `binary search` between x and x' and make it x_{new}, and then estimate the gradient of x_{new}, move along such estimated gradient for a short step, then do another `binary search` until $S(x') > 0$.

For the gradient estimation, the author proposed a function of:

$$\widetilde{\nabla S}(x', \delta) = \frac{1}{B} \sum_{b=1}^{B} \phi(x' + \delta u_b) u_b \tag{6}$$

Where $\phi(x') = \text{sign}(S(x'))$, which we have get by simply asking the model to predict x'. And u_b are group of random vectors around x'. We than ignore the random vertors that is close to being parallel to the decision boundary (in the case that a decision boundary is not a plane, close to the tangent of the boundary) – we may identify them by looking at vectors of similar direction but have different $\phi(x')$ output when added to x'. Then we look at the leftover random vectors to converge an estimation of the boundary like demoed in [Figure 9].

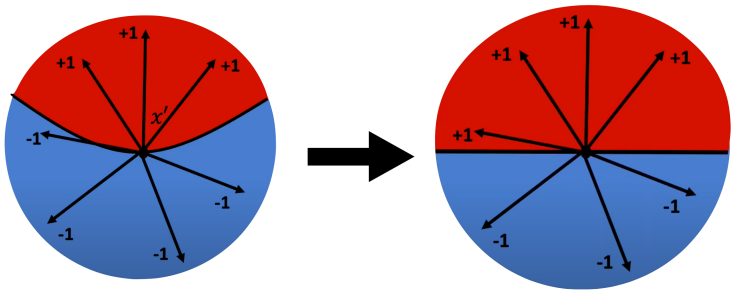

Figure 9: Using random vectors to estimate boundary [1]

Thus, when we find the minimum x' to x with $S(x') > 0$, a black box attack is completed.

Algorithm Implementation Again, I implemented `HSJ` with the help of `ART`[4] (which has dependency to `Tensorflow` and `Keras`), wehere `ART` is a open-source `Python` library. Since I have no extension for this method, the activation method is simply:

```
attacker = HopSkipJump(classifier=classifier, targeted=False, norm=np.inf, max_iter=32,
    max_eval=100, init_eval=10)
attacker = HopSkipJump(classifier=classifier, targeted=False, norm=np.inf, max_iter=64,
    max_eval=1000, init_eval=100) # Close to authors' recommended setting
x_test_adv = attacker.generate(x_test[:adv_num])
```

and the workflow remain the same as Section 2.1.2. The algorithm has pretty much two aspects: `binary searches`, and gradient estimation

For the former part, the idea is to generate pertubation in the midpoint of x and x', and applying `binary searches` to get the x^k and x^{k+1} where $\phi(x^k) = \phi(x_{\text{target}})$ and $\phi(x^{k+1}) \neq \phi(x_{\text{target}})$, then we make x^k to be the final midpoint of these searches. For the latter part, we simply generate pertubation of $x' + u_b$, and depending on the $\phi(x' + u_b)$, we add or minius all pertubation together to be the estimated gradient.

Experiments and Evaluation `HSJ` is the most computationally-heavy algorithm amoung our group's implementations, and thus most experiments done on `HSJ` will be present and discuss in Section 3.2. However, we notice the authors are using a rather aggressive params setting of `max_iter=64, max_eval=10000, init_eval=100`, assuming the high `max_eval` is to evalaute more random vectors along the boundary. We have tested `HSJ Rec Params` (`max_iter=64, max_eval=1000, init_eval=100`) against `HSJ Fast Params` (`max_iter=32, max_eval=100, init_eval=10`) and it seems they have very similar performance as demonstrated in [Figure 10]

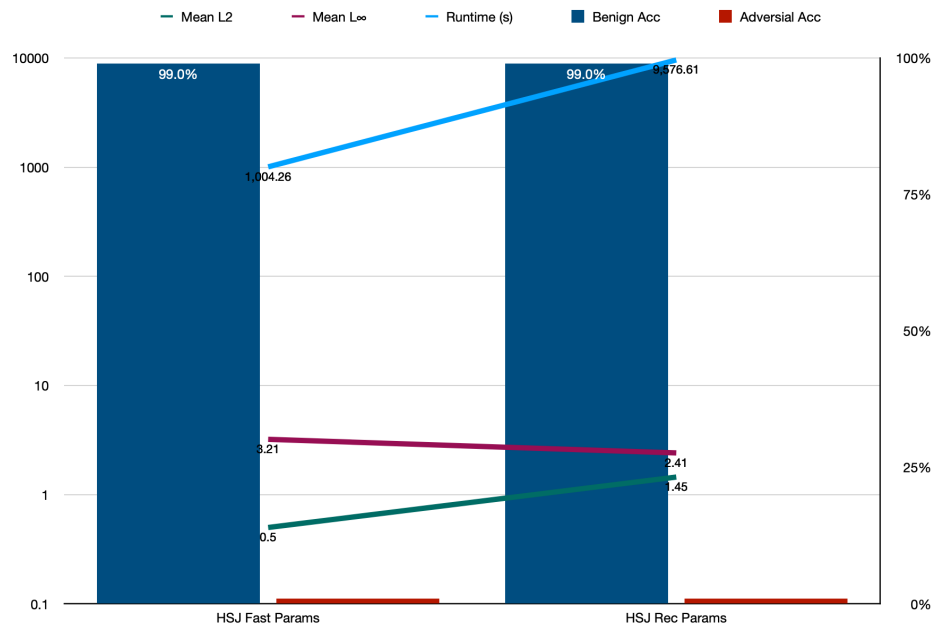

Figure 10: `HSJ` running with recommended and fast params setting on *MINIST*

2.1.4 `Feature Collision`

`Feature Collision`[10] is an algorithm I read and understood but did not actually implement, mostly due to time constraint. `Feature Collision` is a *posioning* attack algorithm, which is usually considered to be impartical as it is highly unlikely for an attack to have access to the training phase of a model.

But `Feature Collision` is implemented with concept of `clean label`, as not the attack but a third-party will label the "poisoned" image. This increases the likelihood of the poisoned image entering the training set of the model as it is commom practice to scrape internet for training data.

So, in a general term, `Feature Collision` *collides* the feature between a base class to a target class, making adversarial example that carrying the semantic the base class but full of the features of the target class. Since it is indistinguishable to human eyes, if it enters to training set of a model, the result can be catastrophic like demoed in [Figure 11].

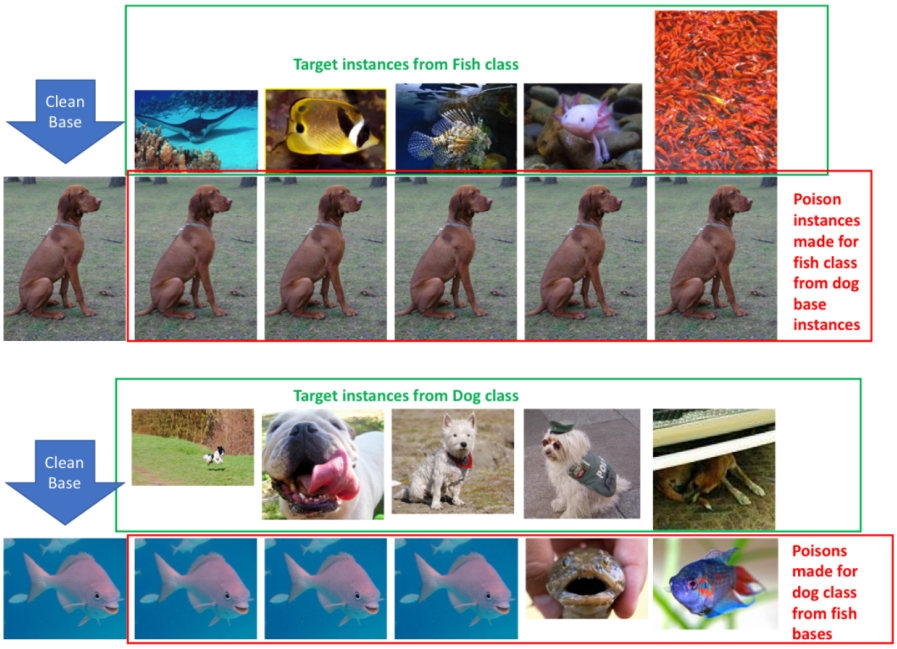

Figure 11: The result of `Feature Collision` [10]

In the paper, the author also emphasis the effectiveness of `Feature Collision` as normally posioning attack will require control of labels or even having adversarial examples in to most or every batch of images, but `Feature Collision` complete an effective attack even with just one adversarial example – a legitment *one-shot kill* as named by the authors.

2.2 Minyang Tie's Individual Report

2.3 Alex Useloff's Individual Report

2.4 Austin Keppers' Individual Report

The main algorithm that I focused on for this report was Defensive Distillation. Defensive distillation is a way to train a deep neural network classifier with the goal of better defending against adversarial examples that was first proposed in the paper *Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks* by Papernot et all. The main idea behind Defensive distillation is that Deep Neural Networks are susceptible to adversarial examples because they make overly confident predictions when a sample has elements of two different classes. Defensive Distillation attempts to deal with this by using a classifier that is trained on soft labels instead of hard labels. Soft labels provide probabilities that an example belongs to each of the classes in a model rather than identifying a specific lass the example belongs to.

Distillation is a technique that was created in order to run Deep Neural Networks on devices with lower computational power by reducing the size of the network. The idea is that hard labels can be used to train a large neural network, and then this classifier can be used to create soft labels to train a smaller neural network with the goal of obtaining an accuracy similar to the original larger classifier. Defensive distillation uses a variation of this technique in order to create a deep neural network that can better classify adversarial examples.

The output of the neural network used in distillation must be a Softmax layer with a temperature parameter. The output of a softmax layer is as follows

$$F(X) = \left[\frac{e^{z_i(X)/T}}{\sum_{l=0}^{N-1} e^{z_l(X)/T}} \right]_{i \in 0 \ldots N-1}$$

where $Z(X)$ is the output of the previous layer and $F(X)$ is an output that corresponds to a vector with a probability for each class. The temperature value T in the Softmax layer determines how confident the classifier will be about the most probable class. A high temperature value will cause the different class probabilities to be closer together whereas a low temperature will cause the most probable classes to have a probability further apart from the less probable classes. This is because the exponent is divided by the temperature and reducing the exponents value by the same amount for all classes will cause the output values to be closer to each other

In Defensive Distillation a neural network where the last layer is a softmax layer is trained using a dataset with hard labels. The temperature of the Softmax layer is set to a high value (something greater than 1) so that the classifier will output values with larger values for each class. This means that the output will emphasis ambiguities in certain examples that may have similarities to a different class. The examples are then passed through the trained neural network at the same high temperature and the output of the classifier is recorded as a soft label for that example. Whereas in distillation a smaller model is trained using the soft labels, in defensive distillation the goal is a classifier more resilient against adversarial examples rather than a performant one so the second classifier is of the same size as the first one. When this second classifier is trained using the soft labels it the temperature of the Softmax label is set to a high value. When it is then being tested the Temperature is lowered to 1 so that the classifier outputs more confident probabilities.

Theoretical justifications presented in the paper as to why defensive distillation works against adversarial examples include that a higher temperature reduces the model's sensitivity to small perturbations in the input to the classifier, and that defensive distillation improves the generizability of the classifier outside of the training sample.

The researchers were able to use defensive distillation to lower the success rate of adversarial examples from 95.89% to 0.45% on the MNIST dataset and from 87.89% to 5.11% on the CIFAR10 dataset. The distillation did result in a moderate decrease in the accuracy on non-adversarial examples with a 1.28% decrease on the MNIST dataset and a 1.37% decrease on the CFAIR10 dataset. The models performed better against adversarial examples at higher distillation temperatures with the temperature of 100 (the highest that they tested) performing the best. They also show that defensive distillation increases the robustness of the classifier produced.

The second paper I read finds a way to revert adversarial examples into non-adversarial examples before feeding them into a classifier instead of training the classifier to handle adversarial images. This paper was *Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models* by Samangouei et all. and its idea is to use Generative Adversarial Networks to help protect Deep Neural Networks against adversarial examples.

Generative Adversarial Networks consist of two networks: D and G. D is a binary classifier for examples x, and G learns how to craft examples with the dimensionality of x from a random vector z. G learns to map z to $G(z)$ similar to x and D then learns how to distinguish between x and $G(z)$. In the paper generative networks were trained with a loss function based on Wasserstein distance which is as follows:

$$\min_G \max_D V_w(D, G) = \mathbf{E}_{x \sim p_data(x)}[D(x)] - \mathbf{E}_{x \sim p_z}[D(G(z))]$$

Feeding non-adversarial examples through the generative network should barely effect the examples so long as p_g converges to p_{data}. This means that legitimate example will not be altered by being fed through

the network while adversarial example will be altered.

Defense-GAN works by finding an input z^* to the GAN that will match the original input x as closely as possible. $G(z^*)$ is what will then actually be fed into the classifier. The exact expression to be minimized is

$$\min_z \|G(z) - x\|_2^2$$

Gradient descent is then run on this function with random restarts to find z^*

The reason this approach is useful is because it assumes very little about the type of attack or the classifier that is being used. The classifier used to classify the images can also be trained using either the original training set or images generated by the generative network.

The researchers tested different combinations of attacks and defenses on the MNIST dataset as well as the Fashion-MNIST data set. The types of attacks included both black box and white box attacks. Defense-GAN performed well on the MNIST data set on many different types of attacks and classifiers, while other types of defenses tested performed well against only certain types of attacks. The performance from training the classifier on the original images and the images from the Generative Adversarial Network were both comparable. Increasing the number of random restarts increased the classification accuracy. The number of iterations of gradient descent generally increased the accuracy but against adversarial examples high numbers of iterations eventually decreased the accuracy.

Defense-GAN can also be used to detect the use of adversarial examples. This is because examples with larger perturbations from the original examples will be further away from the image produced by the generative network than unchanged images. Thus the authors propose used the mean squared error of the original image and the image output by the GAN to detect attacks.

Two difficulties the authors note that may need to be considered when deploying defense-GAN are the training of adversarial networks as well as the choice of parameters. They note that there are still challenges in training GANs and the choices of L, the number of gradient descent iterations, and R, the number of random restarts, are both important factors in the effectiveness of defense-GAN.

2.5 David Meshnick's Individual Report

3 Comparative Study and Discussion

In this section, we will present a comparative study between some algorithms (and their comparable extensions) we implemented.

3.1 Overview

3.1.1 Datasets and Sample Selections

For the tesing datasets, we opted to use *MINIST* [5] and *CIFAR-10* [3]. In short, *MINIST* is a database of handwritten digits and *CIFAR-10* is a database for tiny images, as demonstrated below in [Figure 12] and [Figure 13].

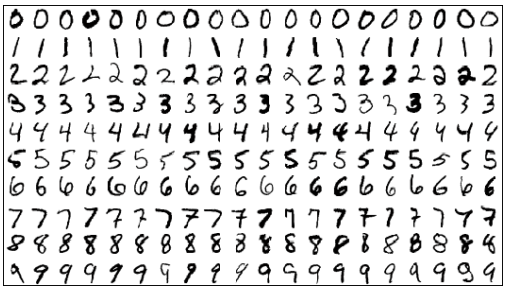

Figure 12: An selection of *MINIST* database

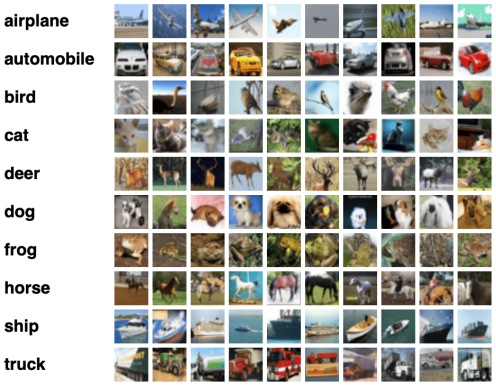

Figure 13: An selection of *CIFAR-10* database

Our decision of using these two datasets are base on the consideration of:

- Both datasets are pre-labeled and have pretrained model available for use. So we may save time on training the victim model(s), and also have some consistent baselines to refer to.
- Both datasets are lightweight in terms of image resolutions, so when evaluating computational-heavy algorithms (e.g. `Hope Skip Jump`), we are able to get the experiments data either locally or with minimum use of cloud services.
- Having ORC and image classifying tasks (in general) together can be a very fair coverage of the actual applications of adversarial learning.
- *MINIST* specifically has a close-to-pure background color, which is a great for human to evaluate as sometime the perturbation on a colorful picture can be unnoticeable to human eyes.
- The toolbox of choice *adversarial-robustness-toolbox* a.k.a `ART` have wrapper methods around these two models, and can load their pretained models as `ART`'s victim classifiers (we have specifically inquired Dr. Ray that it is ok to `import` this kind of facilities).

We have thought about using a third database like *ImageNet* or *IRIS*, but with the combinations of attacks and defenses algorithms we already have a very heavy experiments workload. So we opted to only use these two. However, considered the image quality of *MINIST* and *CIFAR-10* are rather on the low side, we used some *ImageNet* images to demo our work and concepts.

3.1.2 ART

adversarial-robustness-toolbox[4] a.k.a `ART` is an IBM-sponsered library that provides tools for necessary adversarial learning experiments. We have utlized `ART`'s facilities on loading dataset, wrapping victim classifiers, and piplining attack and defense algorithms together. We cannot finish this project without this library, so much credit to them.

3.1.3 Metrics

As we are not doing binary classifications, *accuracy* will be our top piority. This is also the case for general goal of adversarial learning, as we either what to attack the model to lower its *accuracy*, or we want to

defend from an attack with increased robustness – thus higher *accuracy*.

However, another important aspect of adversarial learning is, in most of the cases, we want the our attack image to be only "adversarial" to a computer model, but not to human eyes. This is first because without such restraint we may simply swap a dog picture with a cat picture and call it a successful attack, which will make the task meaningless. Second, it is because for most of the time we want our attack to be "stealthy" – and a collection of pixels noises is simply not so much of that.

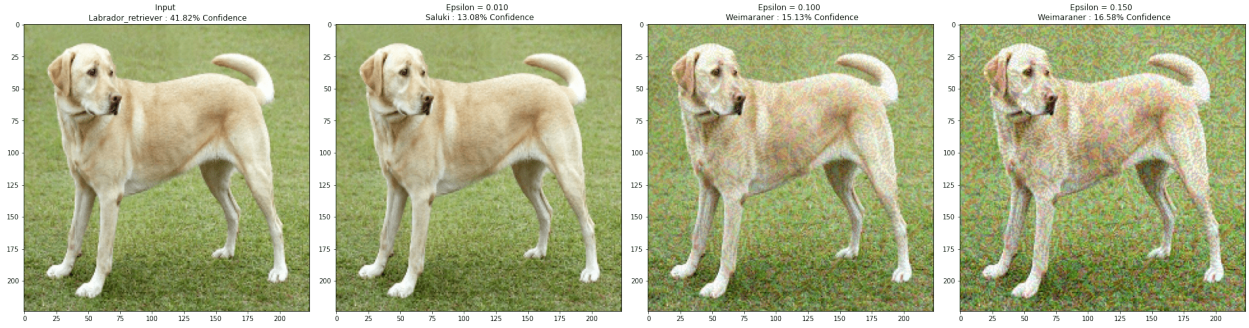

Figure 14: A Labrador Retriver with different levels of perturbations

Thus, keeping the "semantic" of the original image is important in terms of making adversarial examples. However we must have a way to quantify such perturbations – as it can be very hard to tell like shown in [Figure 14] [8] – so that we can numerically conclude and justify the claim if a image has lost it semantic (and by how much).

The metrics we found is *perturbation budget*[6], it defined as:

$$\epsilon = ||x_{\text{adv}} - x|| \tag{7}$$

where $epsilon$ in [Equation 7] is the *perturbation budget*. However, we have different choices on calculating the distance metrics. In this case, we opted to use L_2 and L_∞ as they represent the EUCLID distance and maximum magnitude between pixels – which covered the perturbation of a picture both holistically and "extremely."

Specifically in practice, we calcualte the L_2 and L_∞ of two images channel by channel then add them together. The principle is we will want the these numbers to be as small as possible, as long as the adversarial image we created still have an adversarial affect to a model. Note if an algorithm is iterative, a smaller *perturbation budget* also means smaller computation cost – as less iteration were done.

Another thing that maybe worth mentioning is since we used pretained victim model, we didn't do anything like N-fold cross validation as we are seeking effects out of a trained model, but not to train one (maybe except `Neural Cleanse` as it does retain the model). But we do average our trials to make sure our experiment results are reproducable.

3.1.4 Adversarial Rivalry

We have implemented and experimented multiple attack and defense algorithms, but not all of them can be compared together due to various reasons. In our cases, we opted to not to have some algorithms compared together, or even not to include some algorithm in our comparative study.

We opted to compare `Fast Gradient Sign Method` (non-targeted, with one-shot and iterative implementions), `Hop Skip Jump`, `DeepFool` (and `Dynamic DeepFool` – an extension implemented by David) to-

gether as they are all the non-targed evasion algorithm we implemented. We opted to not include Backdoor in this set of comparsion as it evasion attacks were done in the assumption of having a trained model, it doesn't make sense to have posioning – which can alter the training set of a model – to be part of the comparsion. Similarily, the two targeted extensions FGSM is excluded from this comparsion as in this context it is the magnitude of decrease of model *accuracy* in general we care about, but not which exact label the model most classifies to.

Likewise, we opted to not compare any other algorithm except Backdoor to against Neural Cleanse, as: Minyang please input here.

Also we are not comparing different defense algorithms against a same attack algorithm. This is not because we are not interested in the performance difference between defense algorithm. It is simply because we have already elimited Neural Cleanse due to its retrain nature; and for the other two remained defense algorithm Binary Input Detector and Spatial Smoothing, the former one is designed to recognize and flag an adversarial image, where the latter is to increase model robustness so that it can better classified adversarially perturbed images – so they can't be compared.

3.2 Attack Algorithms

Note I-FGSM represents (non-targted, one-shot) Iterative Fast Gradient Sign Method; D-DeepFool represents Dynamic DeepFool. The FGSM is running with params being batch_size = 32, eps = 3, and the I-FGSM is running on the same setting with eps_step = 0.05 as we have found in Henry's individual report with eps_step > 0.05 we might "overshot" ourself.

Hop Skip Jump a.k.a. HSJ is a very computational-heavy algorithm. When possible, we will use max_iter=64, max_eval=1000, init_eval=100 which is close to the authors suggested setting (except the authors suggests max_eval = 100000 which is impossible to replicate with our resource). But often time we are limited for max_iter=8, max_eval=100, init_eval=10.

Unless specifically addressed, the base line model is a the pretrained model of dataset wrapped in ART's KerasClassifier.

3.2.1 Evasion

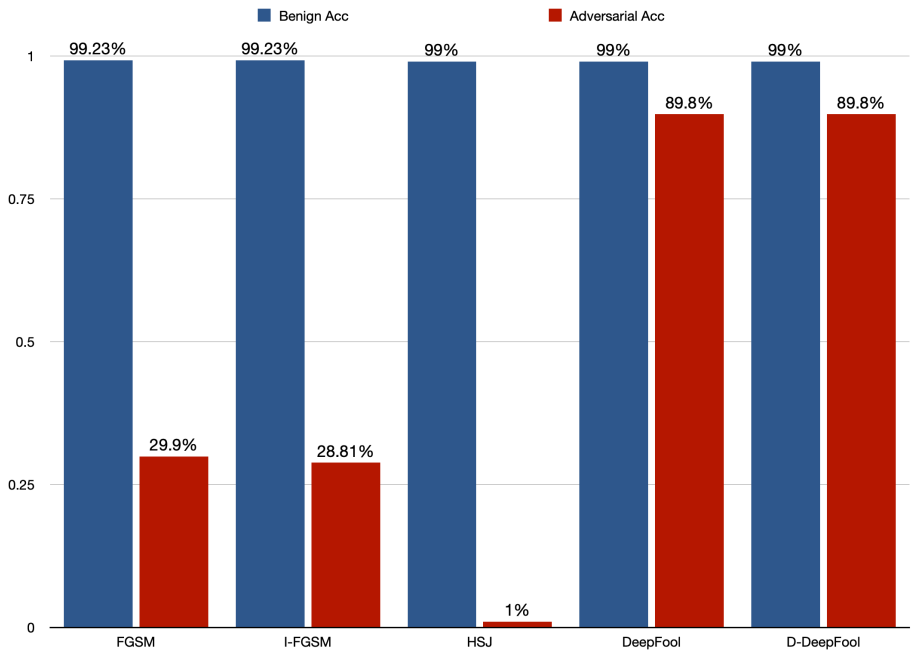

Figure 15: Accuracy comparsion of evasion attack algorithms on *MINIST*

By observating [Figure 15] it can be clearly tell that the presented algorithms are in three different "levels." With `HSJ` showing the best performance and `DeepFool` and `D-DeepFool` showing identical performance, the only interesting question left is if `FGSM` is significantly different from `I-FGSM`.

We then try to find the 95% CI of $E_{\texttt{FGSM}} - E_{\texttt{I-FGSM}}$ with a null hypothesis of they have no difference. Note the sample size of tested *MINIST* is 10000.

$$
\begin{aligned}
F &= 0.299 - 0.2881 \\
&= 0.0109 \\
V(F) &= 0.299(1 - 0.299)/10000 + 0.2881(1 - 0.2881)/10000 \\
&= 0.000041469739 \\
\Rightarrow \sigma &= 0.006439700226 \\
95\%\text{CI} &= 0.0109 \pm 1.96 \cdot 0.006439700226 = (-0.001721812443, 0.02352181244)
\end{aligned}
$$

With 0 lies in the 95% CI, we can't reject the null hypothesis and `FGSM` and `I-FGSM` are not different in terms of accuracy performance in this experiment. This is consistent to our understanding of the algorithms as `I-FGSM` is suppose to be the iterative version of `FGSM`, it performed slightly "better" in number on *adversarial acc* probably just because the adversarial image is generated closer to the decision boundaries of the model and therefore a bit more "confusing" – we will analysize this issue closer with the following budget graph [Figure 16].

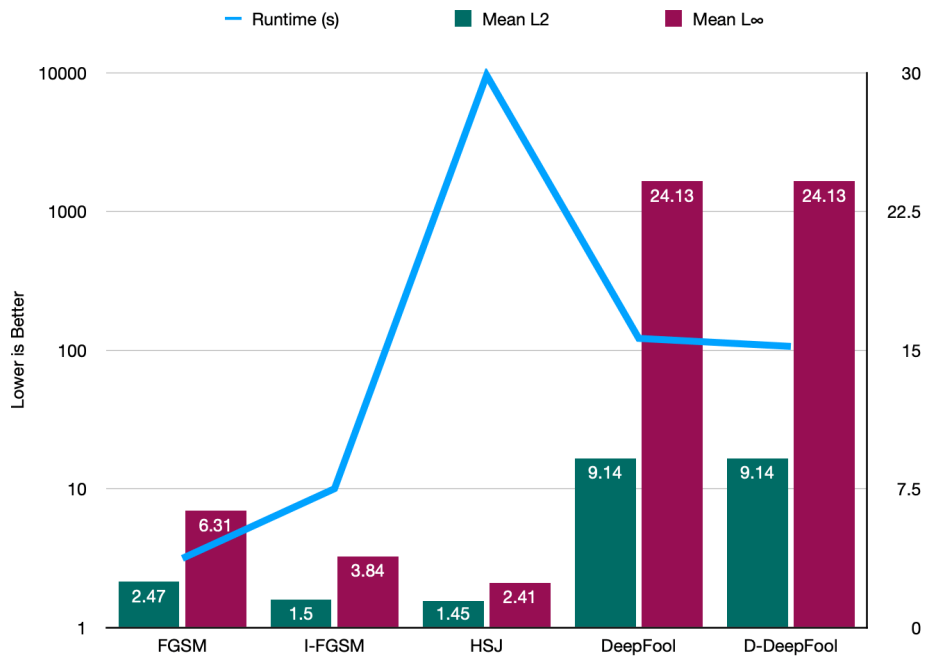

Figure 16: Budget comparsion of evasion attack algorithms on *MINIST*

By investigating the [Figure 16] we may confirm our thinking that `I-FGSM` generated adversarial examples are less aggressive (and thus closer to the decision boundaries), as it has a lower mean L_2 and mean L_∞ (we can't do 95% CI on this as the sample size has no bearing to the perturbation budget).

In our pervious observation on [Figure 15] we said `HSJ` has the best performance in terms *adversarial acc*. Now we know that `HSJ` did such with minimum perturbation budget spent (by having the lowrest mean L_2 and mean L_∞ across the board). However, the cost of doing this is very high, the runtime of `HSJ` is close 100 times of other algorithm. These observations are also consistent to our understanding of `HSJ`, as it is doing *binary search* until it crosses the decision boundary – so it can perserve a high amount semantic from the original image, at the cost of spending a lot of time to find such image.

David please explain a bit on DeepFool as why it perturbed a huge amount while performed not so well, and maybe also why its L_∞ is a lot higher than L_2.

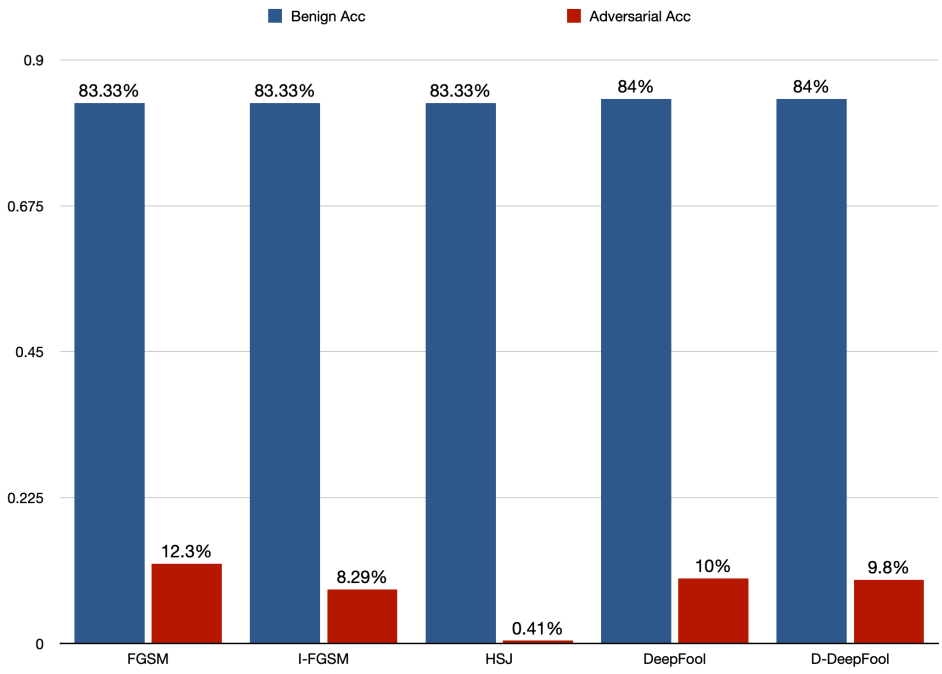

Figure 17: Accuracy comparsion of evasion attack algorithms on *CIFAR-10*

We then repeated the same experiment on 100000 *CIFAR-10* examples and got the result of [Figure 17]. This time we have a different outcome between `DeepFool` and `D-DeepFool`, so we will analysis $E_{\texttt{FGSM}} - E_{\texttt{I-FGSM}}$, $E_{\texttt{DeepFool}} - E_{\texttt{D-DeepFool}}$, and $E_{\texttt{I-FGSM}} - E_{\texttt{HSJ}}$ (as they are closer this time). With an aid of a script and a null hypothesis of having no difference, we have:

- 95% CI of $E_{\texttt{FGSM}} - E_{\texttt{I-FGSM}}$: `(0.031694853818380074, 0.04850514618161992)`, rejected.
- 95% CI of $E_{\texttt{DeepFool}} - E_{\texttt{D-DeepFool}}$: `(-0.006278442326911505, 0.010278442326911509)`, cannot rejected.
- 95% CI of $E_{\texttt{I-FGSM}} - E_{\texttt{HSJ}}$: `(0.07325244583218875, 0.08434755416781124)`, rejected.

It is a bit suprisized to see the null hypothesis of $E_{\texttt{FGSM}} - E_{\texttt{I-FGSM}} = 0$ can be rejected. This is probably because *CIFAR-10* has more label catagories where *MINIST* only has 10 digits, thus decision boundaries between (more) different labels to be closer. The observation of having an overall lower *benign acc* also confirms this assumption. We will look into it again in the following budget analysis [Figure 18].

Also, with $E_{\texttt{I-FGSM}} - E_{\texttt{HSJ}} = 0$ rejected, we may say that `HSJ` indeed performs better than `I-FGSM`.

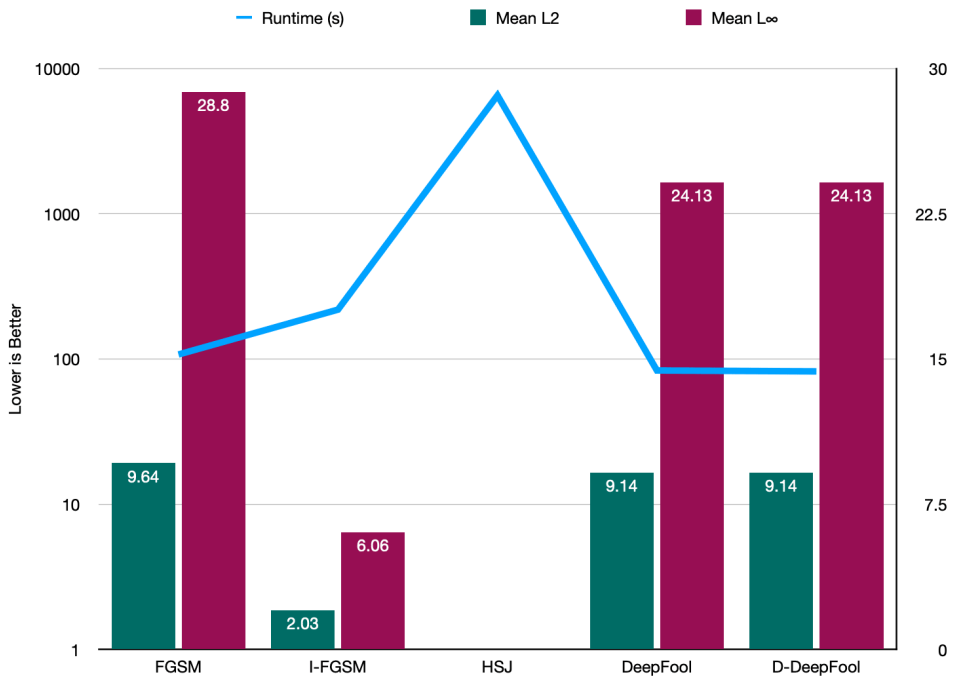

Figure 18: Budget comparsion of evasion attack algorithms on *CIFAR-10*

The budget analysis [Figure 18] comfirms our thinking as the perturbation budgets are larger acrossed the board in comparision to *MINIST*. Also all other observation made from the *MINIST* budget graph [Figure 16] are also true in this *CIFAR-10* budget graph.

3.2.2 Poisoning

Please refer to Section 3.3.3 as we have only one poisoning attack algorithm and want to analysize it in combination with its designated defense: `Neural Cleanse`.

3.2.3 Conclusion

We have made the obersvation of `HSJ` is having the best *accuracy* performance. With `HSJ` and `I-FGSM` being more successful on controlling their perturbation budget – due to their iterative nature – and therefore probably are overall more reliable attacks as they can generate more effective (i.e. more confuse to a modal) adversarial examples while preserve better semantic of the original benign image (i.e. the pertubation is less obvious to human's eyes).

Note we also discovered `I-FGSM` and `HSJ` are computationally-heavy (especially the latter), due to their iterative nature and the need of many binary search operations in the case of `HSJ`. So it is suspected this sort of attacks can be better prevented by implementing flow control mechanism of the model `predict()` API – as without enough steps of iterations, these two algorithms won't easily find the decision boundary of the model.

`DeepFool` and `D-DeepFool` have shown very similar performance across this section of experiments and their performance are considerably lacking both in terms of *adversarial acc* and *perturbation budget*. David, please again summarize a bit here.

3.3 Defense Algorithms

The setup of algorithms and environment remain consistent to Section 3.2, with the exception of we ran `HSJ` with the params of `max_iter=8, max_eval=100, init_eval=10` due to resource concern.

Unless specifically addressed, the base line model is a the pretrained model of dataset wrapped in `ART`'s `KerasClassifier`; and the testing sample size of datasets are 500 (per each databset).

3.3.1 Detector

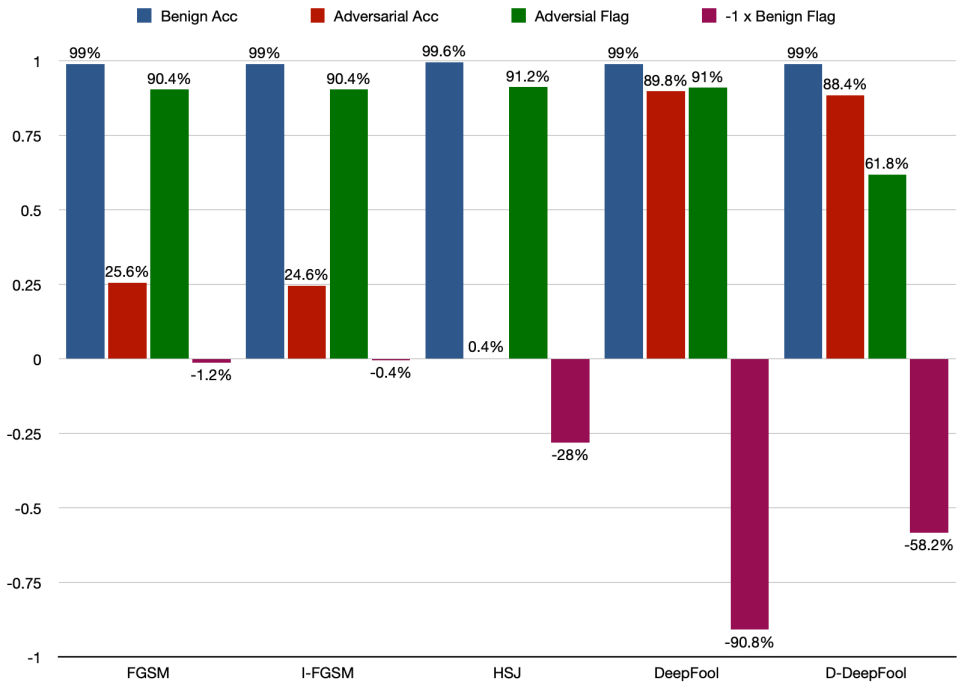

Figure 19: Effectiveness comparision of evasion attack algorithms v. `Binary Input Detector` on *MINIST*

`Binary Input Detector` a.k.a `BID` is algorithm that detects and flags adversarial examples out of a dataset. Since the workflow of our experiment is to get the *benign acc* with original benign examples, then make them into adversarial exmaples to get the *adversarial acc*, then we ask `BID` to run on both the benign example set and the adversarial example set.

In the adversarial example set, `BID` should aim for a 100% as every input example of the set is adversarial; vice versa, `BID` should aim for a 0% in the benign example set as none of the input exmaple are adversarial – we times this benign flagging percentage with an -1 do show that it is a negative impact.

First, we may tell there is much difference on *adversarial flag* except for `D-DeepFool`, as we have 95% of $E_{\texttt{HSJ}} - E_{\texttt{FGSM}}$ (the biggest difference on *adversarial flag* excluding `D-DeepFool`) to be `(-0.027824596689983806, 0.04382459668998382)`, so we cannot reject the null hypothesis of `FGSM, I-FGSM, HSJ, DeepFool` having no significant difference on *adversarial flag*.

Thus, the interest is left to *benign flag*, we cannot rejected $E_{\texttt{FGSM}} - E_{\texttt{I-FGSM}} = 0$ by having a 95% of `(-0.0030318578671047047, 0.019031857867104707)`. But there are significant differences between

HSJ, DeepFool, D-DeepFool on *benign flag* as we have tested the 95% of $E_{\texttt{HSJ}} - E_{\texttt{FGSM}} = 0$ (second smallested difference on *benign flag*) to be (0.22750277615440784, 0.3084972238455922) on *benign flag*. We think this can be better explained by investigating the [Figure 20] graph.

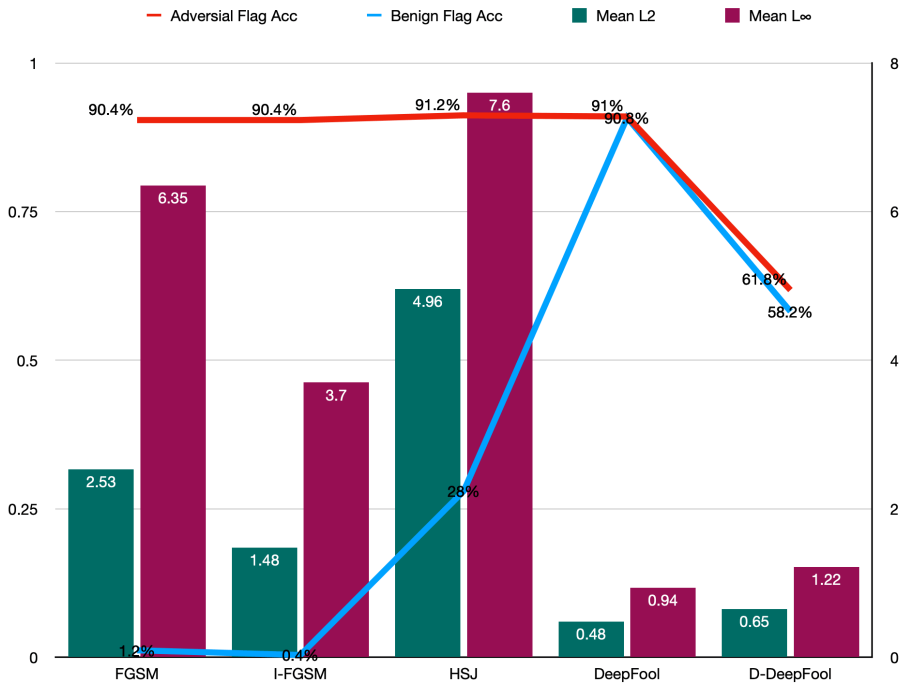

Figure 20: Budget comparision evasion attack algorithms v. Binary Input Detector on *MINIST*

By investigating the [Figure 20] graph, it can be tell there is a clearly correlation between the *pertubation budget* and the *adversarial flag* or *benign flag* (e.g., D-DeepFool). This is in fact very intuitive as less *pertubation budget* means the generated adversarial examples have perserved more semantics or their benign origins, thus making BID hard to distinguish wheather an example is benign or not.

However, it remainds unknow why D-DeepFool has a much lesser *benign flag* and *adversarial flag* in comparision to DeepFool. David please have some input here thanks.

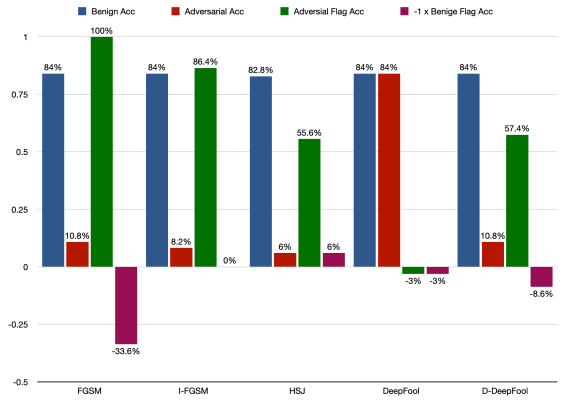

Figure 21: Effectiveness comparision

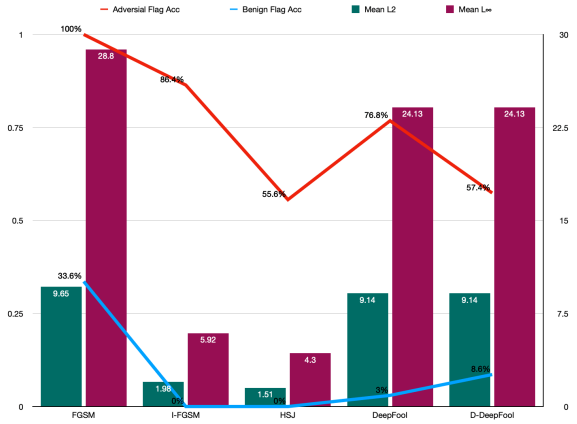

Figure 22: Budget comparision

Evasion attack algorithms v. `Binary Input Detector` on *CIFAR-10*

Our discovary continues uphold on the *CIFAR-10* dataset as algorithms with lower *pertubation budget* gets flagged less (regardless adversarial or benign).

However, this time it is `I-FGSM` and `HSJ` having the least *pertubation budget*. This is in fact resonable due to their iterative nature. We looked into our experiment and realized it is because `HSJ` was running on a test set of 250 examples[1], in combinations with `max_iter = 8`, the algorithm might not have enough iterations and random move to detect the decision boundaries of *MINIST*, which can be a lot harder to detect as they all have pure-color backgrounds.

3.3.2 Pre-processor

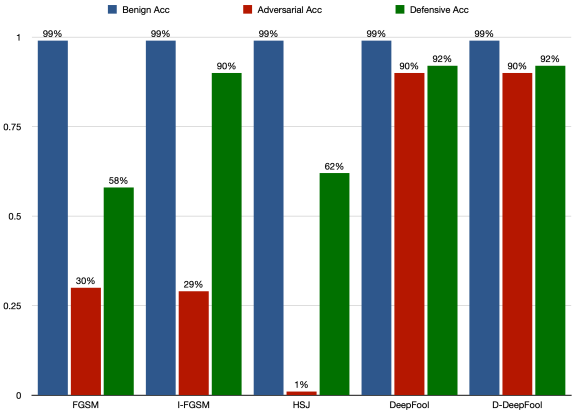

Figure 23: Effectiveness comparision

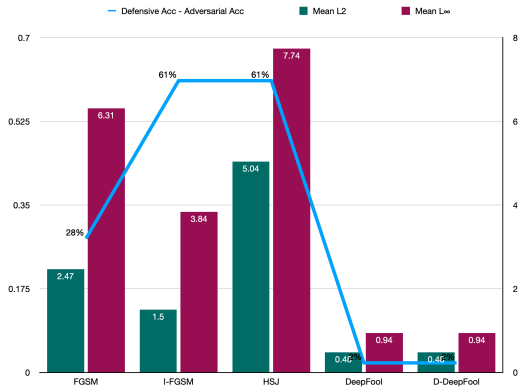

Figure 24: Budget comparision

Evasion attack algorithms v. `Spatial Smoothing` on *MINIST*

[1] We have to reduce the size as `BID` needs to generate the adversarial images twices

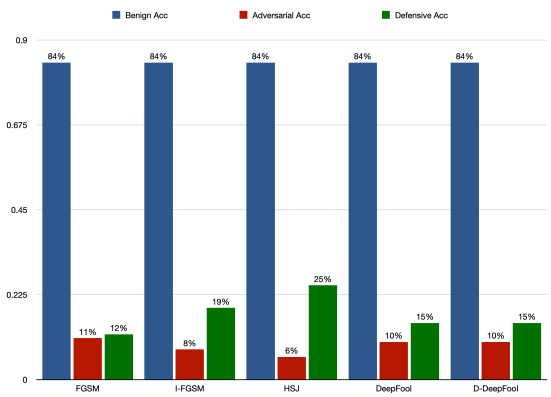

Figure 25: Effectiveness comparision

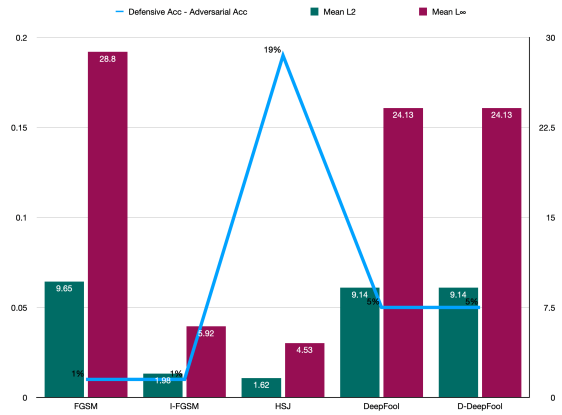

Figure 26: Budget comparision

Evasion attack algorithms v. `Spatial Smoothing` on *CIFAR-10*

`Spatial Smoothing` a.k.a. `SS` is our implemented defense algorithm to increase model's robustness against adversarial input – as it actually actually help predicting the true label of an adversarial image. We may proudly say that there is a 38% *accuracy* before and after the `SS` being implemented across our four experimented algorithms (we exclude `D-DeepFool` for this disscussion as it performs identical to standard `DeepFool`) on *MINIST*.

Note this experiment also confirms the fact that `HSJ` is probably not "converged" yet with its current setting on *MINIST*, as it has again costed high *perturbation budget*. We also observed the higher the *pertubation budget*, the better the defensive effectiveness – this is again in accordance with out intuition as we as human can also distinguish highly perturbed picture well.

Also note [Figure 20] seems to be suggesting low *pertubation budget* will result in high model effectiveness, this is only semi-true as the overall effectiveness increase on *CIFAR-10* is comparatively low (only $+6.5\%$ among the 4 experimented algorithms, where it is $+38\%$ in *MINIST*), so it is more of `SS` being effective against `HSJ`.

We believe this have something to do with the fact `SS` look into the activations of neuros of adversarial pattern and either force-zero or oppress them. As `HSJ` being on the decisions boundaries (implied by the low *pertubation budget*), a `HSJ`-generated adversarial imag might be considered to have adversarial patterns of many kinds and therefore being detected by the defense algorithm.

Minyang, please correct me if I am wrong and add some insight.

3.3.3 Transformer

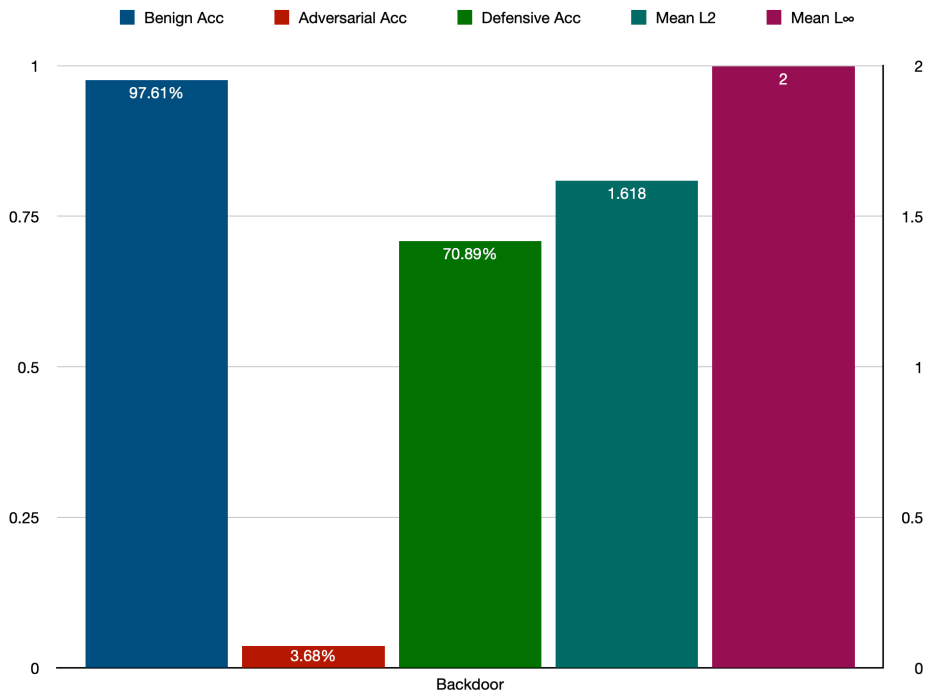

Figure 27: `Backdoor` v. `Neural Cleanse`

Minyang please explain why this works.

3.3.4 Conclusion

As each defensive algorithms have their different purposes and propertities, it is hard to conclude them generally. But base on our consistent observation, it might be safe to say an adversarial example with less *budger pertubation* is likely likely to be detected by a defensive algorithm – which is consistent to our intuition and the geometrical stucture of feature space and decision boundaries: as if something is the middle of several boundaries, it will be hard to distinguish wheather it is an adversarial example of just an "outliner" of a neighborhood class.

In a partical sense, it might be best

4 References

[1] Jianbo Chen. Hopskipjumpattack: A query-efficient decision-based attack. https://www.youtube.com/watch?v=vkCifg2rp34.

[2] 2019 Chen et. al. Hopskipjumpattack: A query-efficient decision-based attack. https://arxiv.org/abs/1904.02144.

[3] Alex Krizhevsky et. al. The cifar-10 dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

[4] Beat Buesser et. al. Adversarial robustness toolbox. https://github.com/Trusted-AI/adversarial-robustness-toolbox.

[5] Yann LeCun et. al. Minist database. http://yann.lecun.com/exdb/mnist/.

[6] Dan Boneh Florian Tramer. Adversarial training and robustness for multiple perturbations. https://github.com/Trusted-AI/adversarial-robustness-toolbox.

[7] 2014 Goodfellow et. al. Explaining and harnessing adversarial examples. https://arxiv.org/abs/1412.6572.

[8] Google. Adversarial example using fgsm. https://www.tensorflow.org/tutorials/generative/adversarial_fgsm.

[9] Aleksander Madry. A new perspective on adversarial perturbations. https://simons.berkeley.edu/talks/tbd-57.

[10] 2018 Shafahi et. al. Poison frogs! targeted clean-label poisoning attacks on neural networks. https://arxiv.org/abs/1804.00792.