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

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.

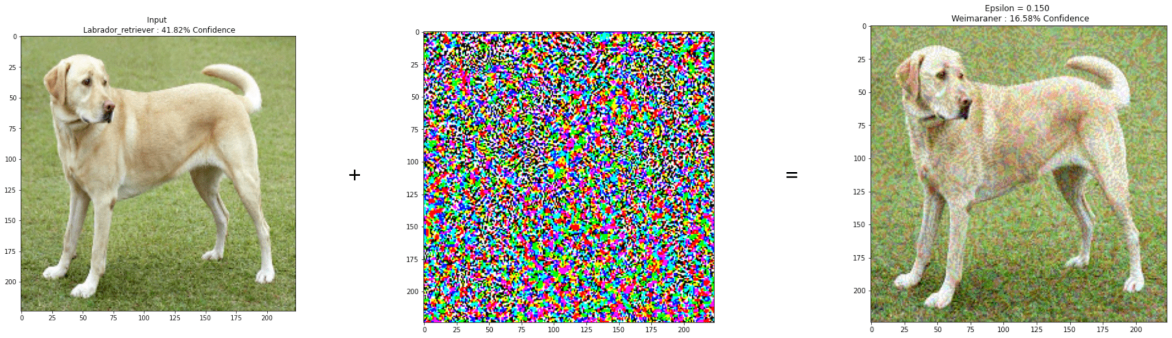


Figure 1: An example of adversarial pertubation resulting successful adversarial attack

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:

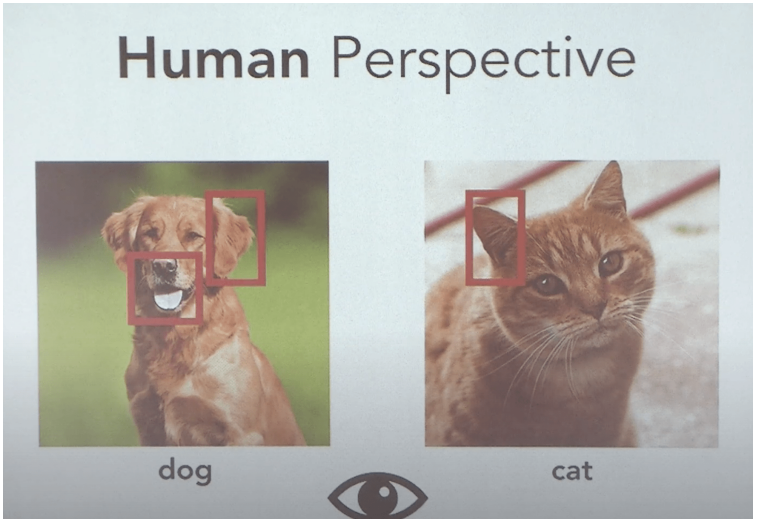


Figure 2: Human perspective

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:

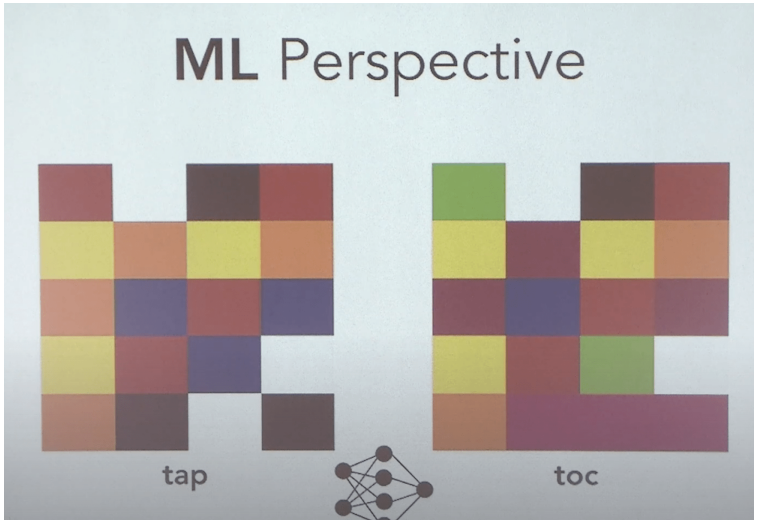


Figure 3: Machine perspective

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):

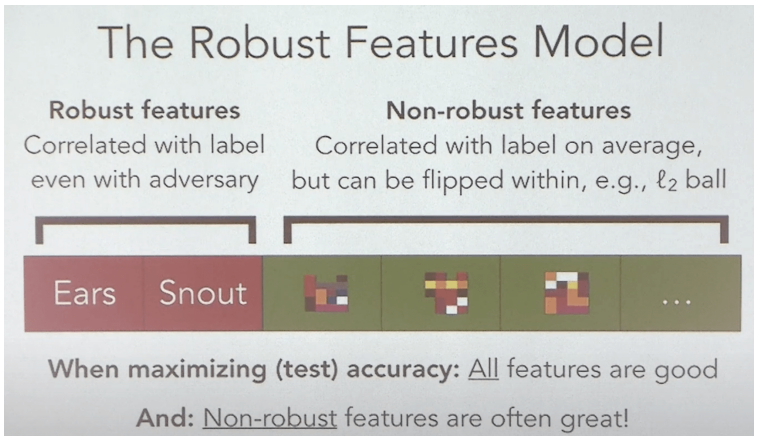


Figure 4: Robust and non-robust features

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]:

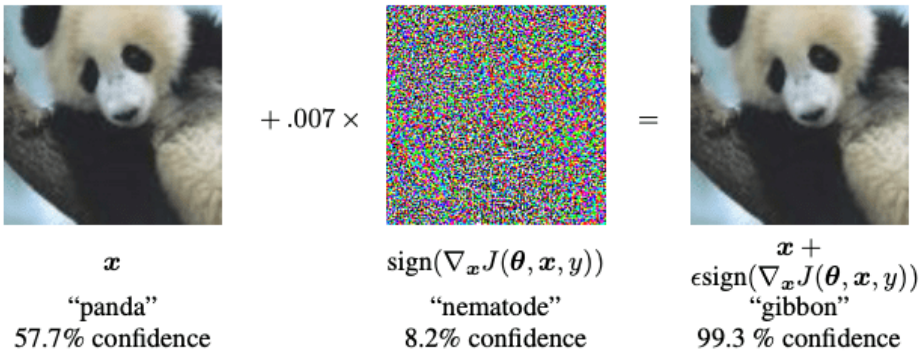


Figure 5: Applying gradient ascent pertubation to a panda image, $\epsilon = 0.07$

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:

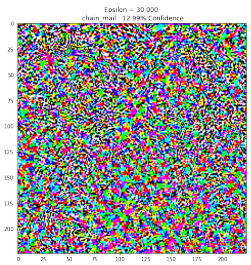


Figure 6: Perturbed Labrador Retriever with $\epsilon = 30$

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:

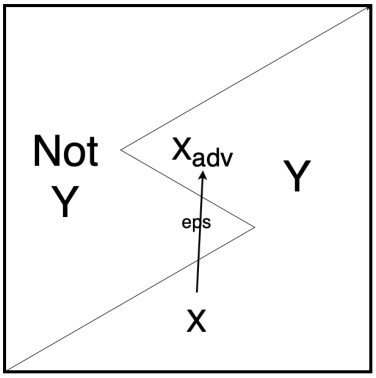


Figure 7: Potential ϵ overshoot

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`

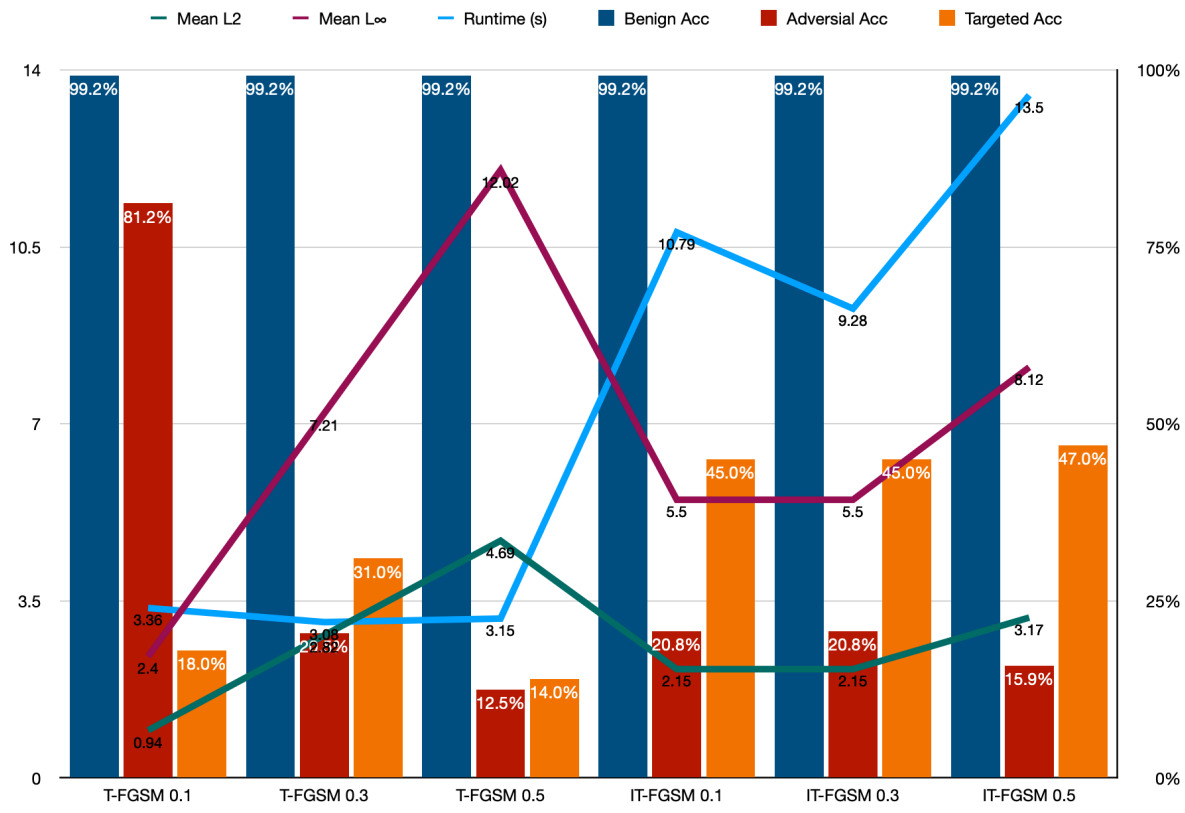


Figure 8: `T-FGSM` and `IT-FGSM` with $\epsilon = \{0.1, 0.3, 0.5\}$ on *MINIST*

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].

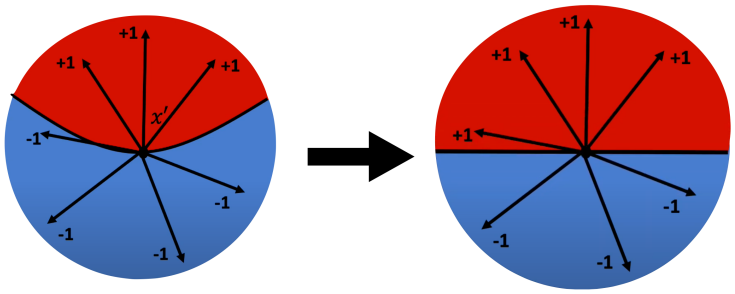


Figure 9: Using random vectors to estimate boundary [1]

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]

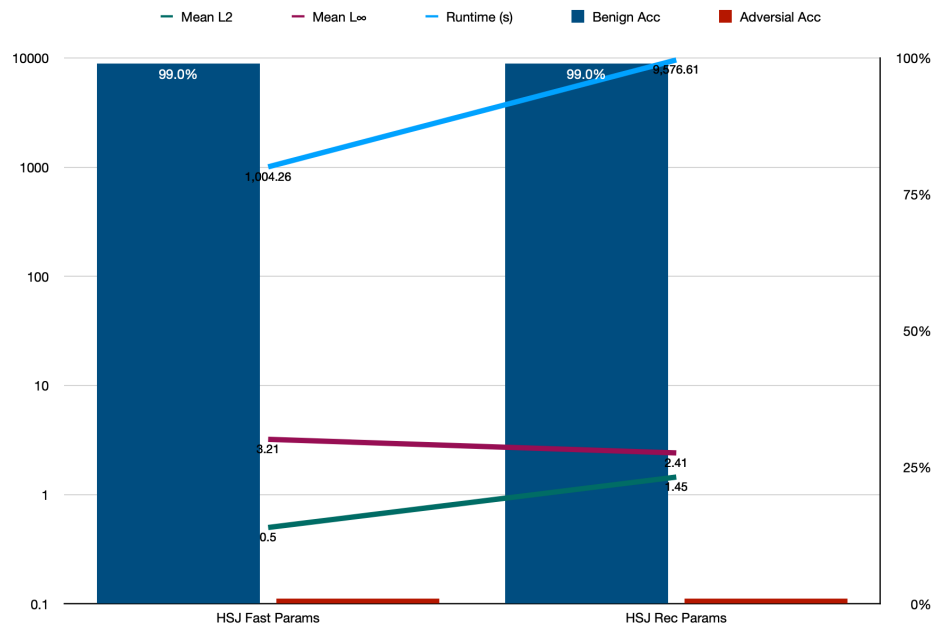


Figure 10: `HSJ` running with recommended and fast params setting on *MINIST*

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].

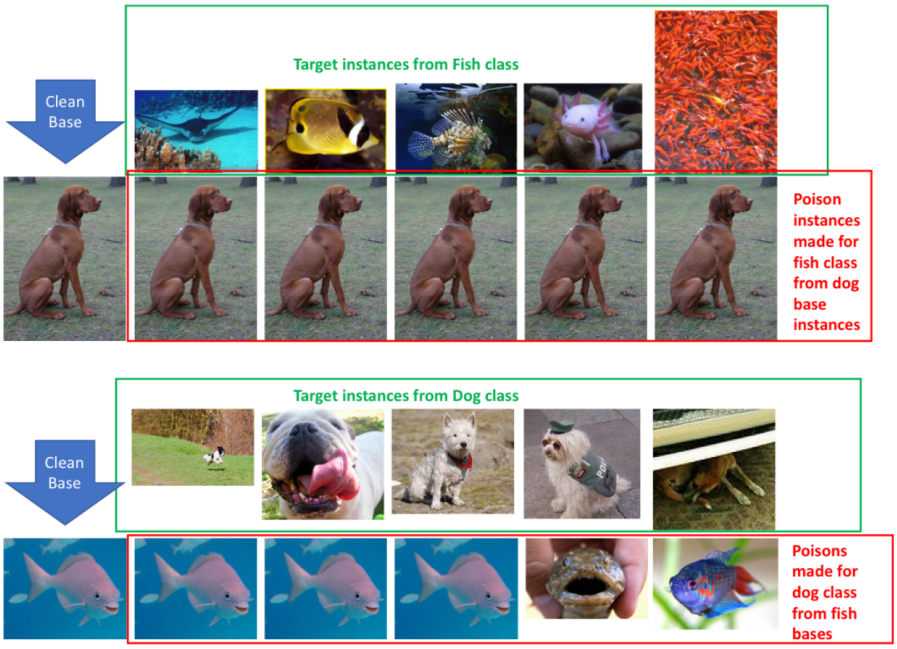


Figure 11: The result of `Feature Collision` [10]

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

Three Defence Techniques for Adversarial Sample Attack

Minyang Tie
Department of Computer and Data Sciences
mxt497

1 Background

Artificial Intelligence (AI) has achieved great success in many field, such as image classification, object detection, natural language process, and so on. Current AI programs are based on training powerful neural networks on large datasets to reduce errors. For instance, ResNet [1] can achieve 3.57% error on the ImageNet test set. However, the datasets, like CIFAR10 [2] and ImageNet [3], are labelled by humans and the errors seldom happens. The veracity of the data that are from real life cannot be confirmed. A little damage on the data, such as losing some pixels of a image, may cause totally wrong predicted result. This is called the robustness problems of existing neural networks because of the "No Free lunch theorem [4]". There is no such a model, which can perform generally well on all kinds of problems and datasets.

To further reveal this issue, some previous works have proposed several ways to attack the datasets and decrease the accuracy of the neural networks. There are two kinds of attacks [5]: White-Box Attacks and Black-Box Attacks. White-Box Attacks needs to access to the model parameters, architectures, and inputs/outputs of a classifier model, while Black-Box Attacks only access to the inputs and outputs and know nothing about the model we use. Therefore, White-Box Attacks is usually more efficient. The Fast Gradient Sign Method (FGSM [6]) based Approach is based on gradients. It maximizes the loss function, i.e., the cross-entropy loss, by generating adversarial examples. Projected Gradient Descent (PGD [7]) with one-step scheme and its multi-step variant search the perturbation that maximises the loss of a model and produce adversarial examples. These two algorithms are White-Box Attacks. Backdoor Attack [8] uses the hidden patterns or unexpected behavior of untested inputs. Under Backdoor Attack, the model can still perform well on most normal inputs, while output misclassification once it processes some certain inputs with containing a pattern predefined by the attacker (see Figure 1).

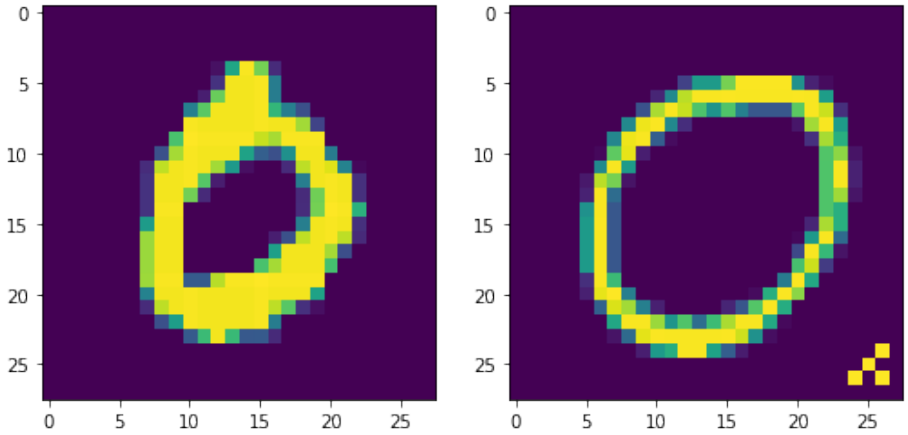
In a classification task, it also has Non-Targeted Attack and Targeted Attack. Non-Targeted Attack only reduce the accuracy of the classifier model without considering the label, while the Targeted Attack could mislead the classifier model to predict a specific wrong label for a given image. There are also many other attack techniques for hacking graph-structured data and context-sensitive data. In this report, we will focus on discussing how to defence those attacks performed on the image classification tasks.

1.1 Related work

Since most attacks are to modify the input data sample, many defence techniques are also related to the dataset.

Binary input detector. Some of them trains a new binary detector [9, 10] to distinguish the original dataset and the poisoned dataset. Adversarial Robustness Toolbox (ART[10]) provides a binary detector based on inputs, which is fit with a mixed dataset of clean and adversarial inputs. They are also correctly labelled with 0 (clean input) and 1 (adversarial input).

Unlearning the backdoor by retraining. Unlearning the backdoor by retraining is the one way of Neural Cleanse method developed by Wang et. al. [8]. They provided a unlearning scheme to remove



(a) The normal image (Prediction is 0) (b) The image under attack (Prediction is 1)

Figure 1: Backdoor Attack on the MNIST dataset

the detection of the backdoor of contain inputs. It works best for Trojan-style triggers that react to a specific neuron configuration. First, it inserts the backdoor into the inputs, then train infected DNN to recognize correct labels even when the pattern is presented in the poisoned inputs.

Spatial smoothing. Spatial Smoothing [11, 12] is a widely used scheme to mitigate the attack of using gradients, such as PGD. Spatial Smoothing runs a sliding window on each pixel to substitute color with median value of all values in sliding window. This median selection can effectively remove sparsely occurring black and white pixels, whilst preserve edges of objects. The sliding window of Spatial Smoothing is very closer to the filter of the convolution process.

Other defence techniques. Data augmentation is a traditonal scheme, which can extend the number of inputs and training a more robust neural network. Zeng et al. [13] uses data augmentation techniques to do preprocessing on the training data to generate a more robust neural network. Shaham et al. [14] deployed other defense mechanisms like PCA, low-pass filtering, JPEG compression, soft thresholding and the inputs are also directly manipulated by these methods.

2 Implementation and Extension

In this section, I will discuss how to implement three defence techniques: Unlearning patterns, Detection of adversarial samples, and Spatial Smoothing. I also show the extension idea based on Spatial Smoothing. For Unlearning patterns and Detection of adversarial samples, I try different way to improve the defensive efficiency, But it has not been significantly improved, or even decreased the defensive efficiency, so in this section, I will only show Spatial Smoothing extension.

2.1 Unlearning

```
def ganSample(model, clean_x_test, clean_y_test):
    clean_data = []
    backdoor_data = []
    backdoor_labels = []
    cleanse = NeuralCleanse(model)
    defence_cleanse = cleanse(model, steps=10, learning_rate=0.1)
    for backdoored_label, mask, pattern in defence_cleanse.
    outlier_detection(clean_x_test, clean_y_test):
        # find data based on the backdoored_label
        data_for_class = np.copy(clean_x_test[np.argmax(clean_y_test,
    axis=1) == backdoored_label])
        labels_for_class = np.copy(clean_y_test[np.argmax(clean_y_test
    , axis=1) == backdoored_label])
        clean_data.append(np.copy(data_for_class))
```

```
12          data_for_class = (1 - mask) * data_for_class + mask * pattern
13          backdoor_data.append(data_for_class)
14          backdoor_labels.append(labels_for_class)
15      if len(backdoor_data)!=0:
16          clean_data = np.vstack(clean_data)
17          backdoor_data = np.vstack(backdoor_data)
18          backdoor_labels = np.vstack(backdoor_labels)
19      return backdoor_data, backdoor_labels
20
21  backdoor_data, backdoor_labels = retrain.ganSample(model, x_test,
        y_test)
22  retrain.fit(model, backdoor_data, backdoor_labels)
23  retrain.result(model, x_test, y_test, x_poison, y_poison)
```

Listing 1: Unlearning patterns

Listing 1 shows how to use retrain to unlearn the patterns inserted by the attacker. We use the library function **outlier_detection** to distinguish the mask and pattern and add these patterns to the clean data (see line 12). We keep the correct label and retrain the classifier with the data that are inserted backdoor (see line 22).

2.2 Detection

```
1   def build_detector(model):
2       layers = [layer for layer in model.layers]
3       num_layers = len(layers)
4       output = keras.layers.Dense(2)(layers[num_layers-2].output)
5       detector = keras.Model(inputs=layers[0].input, outputs=output)
6       return detector
7
8   def train(model, x_train, x_train_adv):
9       nb_train = x_train.shape[0]
10      x_train_detector = np.concatenate((x_train, x_train_adv), axis=0)
11      y_train_detector = np.concatenate((np.array([[1,0]]*nb_train), np.
        array([[0,1]]*nb_train)), axis=0)
12
13      model.compile(optimizer='rmsprop',loss='binary_crossentropy',
        metrics=['accuracy'])
14      model.fit(x_train_detector, y_train_detector, epochs=20,
        batch_size=20)
15
16  x_train_adv = attacker.generate(x_train)
17  dmodel = detector.build_detector(classifier_model)
18  detector.train(dmodel, x_train, x_train_adv)
19  detector.result(dmodel, x_test, x_test_adv)
```

Listing 2: Detection of adversarial samples

Listing 2 shows how we training a binary classifier to detect whether an input is attacked or not. To reduce the training overhead, we use transform learning [15] in our implementation (see from line 1 to line 6). We make full use of the classifier and replace its output with a new dense layer.

After that, we generate new adversarial samples based on the training data. We mix the adversarial training dataset and the original training dataset and label them with [0, 1] and [1, 0], respectively. We deploy the binary cross-entropy as the loss function and train this detector to distinguish the adversarial samples from the original data.

2.3 Spatial Smoothing and its Extension

```
1   def kernel_array(data, xi, yj, nelx, nely, r, N):
2       arr = np.zeros((N, N))
3       startx = xi - r
4       starty = yj - r
5       for i in range(N):
```

```
        for j in range(N):
            x_i = startx + i
            y_i = starty + j
            if x_i < 0 or y_i < 0 or x_i >= nelx or y_i >= nely:
                continue
            arr[i][j] = data[x_i][y_i]
    return arr

def SpatialS(x_art, window_size, way):
    kernel = np.ones((window_size, window_size))
    kernel = kernel/float(window_size*window_size)
    rmin = int(window_size/2)
    if way=="rmin": # sharpen
        kernel2 = np.zeros((window_size, window_size))
        kernel2[rmin][rmin] = 1.0
        kernel = 2 * kernel2 - kernel
    channal, NX, NY = x_art.shape
    x_art_copy = x_art.copy()
    for k in range(channal):
        for i in range(NX):
            for j in range(NY):
                val_ = kernel_array(..., i, j, ...)
                if way=="0.5mean":
                    pix_ = np.sum((val_**0.5*kernel))**2
                else:
                    pix_ = np.sum(val_*kernel)
                x_art_copy[k][i][j] = pix_
    return x_art_copy
```

Listing 3: Spatial Smoothing of Inputs

Spatial Smoothing is a widely used scheme to process images. It set the value of each pixel to be the mean value of the pixels around this pixel. It can make the pixel value become more continuous based on the location (i.e., seems more vague), while keep the bolder of some objects in this image. Spatial Smoothing can defence some attacks, like Projected Gradient Descent (PGD) and Hop Skip Jump (HSJ [16], also see in our group work).

In **SpatialS**, we create different kernel matrixs, played as various filters, and match it with the area of same window size (see **kernel_array**) from the image. Based on Spatial Smoothing, we deploy the way (**0.5mean** in Equation 1, line 29) to replace the mean value for generating the pixel value.

$$pix\_ = \sum_{i=1}^{ws}(\sum_{j=1}^{ws}(val\_[offset+i][offset+j]^{0.5} * kernel[i][j]))^2 \tag{1}$$

where **ws** is the window size.

3 Experiments and Discussion

We use Intel 12 cores i7 CPU, 16 GB memory, and NVIDIA GTX 2050ti. Our operating system is Windows 10 with CUDA 10.1. We uses Tensorflow 2.3.0, Keras 2.4.3, and ART 1.5.0 for evaluation. We use the standard attack algorithm from ART and we apply our defence technique to improve the accuracy and reduce the efficiency of the attacker.

Table 1: Unlearning comparison between ART and our implementation.

	Effectiveness of poison	accuracy on clean test set
ART	reduce from 100% to 0.00%	33.53% (previously 96.62%)
ours	reduce from 100% to 0.36%	**50.81% (previously 96.62%)**

Table 2: Detection comparison between ART and our implementation.

		ART	DenseNet201	our
adversarial	Flagged	37	100	62
	Not Flagged	63	0	38
original	Flagged	0	100	0
	Not Flagged	100	0	100
Accuracy		0.76	0.5	0.95

Table 3: Spatial Smoothing comparison between ART and our implementation of correct prediction accuracy.

		ART	mean	0.5mean
original	3	0.99	0.99	0.99
adversarial		0.89	0.94	0.95
original	5	0.99	0.97	0.98
adversarial		0.92	0.95	0.96
original	7	0.98	0.98	0.95
adversarial		0.90	0.97	0.93

3.1 Unlearning

We train a simple convolutional neural network on MNIST [17] and perform backdoor attack on this dataset. In Table 1, our implementation can reduce the effectiveness of backdoor attack to 0.36%, which is very close to the original implementation. Our implementation could further improve higher accuracy than ART's implementation.

3.2 Detection

We train a ResNet [1] on CIFAR10 [2] and perform FGSM attack on this dataset. We use several neural networks to implement the binary detector, including DenseNet201 and the transfer model based on the classifier. The best case is to use transfer learning based on the classifier, which can achieve up to 95% accuracy and detect more adversarial samples than the ART's implementation.

Since there is no difference between BinaryInputDetector and a normal classifier, the detectors may also be fooled. The performance is not always good as shown in Table 2.

3.3 Spatial Smoothing and its Extension

We use pre-trained ResNet-50 [1] on ImageNet [3] in this experiment, and it predict with the correct label (i.e., unicycle) at a probability of 0.78. After we perform a targeted attack on this input via PGD, which mislead the classifier to label a wrong label to the adversarial image.

We use three ways to defense the attack and compare the results of defence: ART, our implementation of Spatial Smoothing, and our extension. Our implementation of Spatial Smoothing outperforms ART in all evaluated cases. Compared to large window size (7), **0.5mean** could save more computation time than **mean** because it can further improve the accuracy with a small window size (3 and 5). We also found Spatial Smoothing only works when the original image has a very high resolution (224*224 in this case). Once the resolution is low, like images from CIFAR10, the performance is not so good as we show in Table 3.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[2] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[3] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large

Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[4] wikipedia. No free lunch theorem, 2020.

[5] Aminul Huq and Mst. Tasnim Pervin. Adversarial attacks and defense on texts: A survey. 2020.

[6] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[7] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. 2017.

[8] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723, 2019.

[9] Tianyu Pang, Chao Du, Yinpeng Dong, and Jun Zhu. Towards robust detection of adversarial examples, 2018.

[10] Trusted-AI. Adversarial robustness toolbox, 2020.

[11] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing: Detecting adversarial examples in deep neural networks, 2017.

[12] Weilin Xu, David Evans, and Yanjun Qi. Feature squeezing mitigates and detects carlini/wagner adversarial examples, 2017.

[13] Yi Zeng, Han Qiu, Gerard Memmi, and Meikang Qiu. A data augmentation-based defense method against adversarial attacks in neural networks, 2020.

[14] Uri Shaham, James Garritano, Yutaro Yamada, Ethan Weinberger, Alex Cloninger, Xiuyuan Cheng, Kelly Stanton, and Yuval Kluger. Defending against adversarial images using basis functions transformations, 2018.

[15] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2019.

[16] Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. Hopskipjumpattack: A query-efficient decision-based attack, 2019.

[17] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

2.3 Alex Useloff's Individual Report

Adversarial Learning: Backdoor Attacks

Alex Useloff
Case Western Reserve University
Cleveland, OH
adu3@case.edu

Literature Review

The first paper that I read is titled "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain", and it was written by Gu et al. [1]. This paper was one of the earliest I could find that talked about the backdoor attack on deep networks, and it was also mentioned in almost every other paper I looked at on backdoor attacks, so it seemed like a good place to start. It starts off by just going over what neural networks are in general, what they used for, how they are used, and other things of a similar nature [1]. It then goes on to talk about how complicated neural networks can be because of the massive amounts of training data that is required for them to achieve good results [1]. Since it would take too much computational power for most individuals and even businesses to train these complicated networks, many of them are turning to third party providers such as Amazon, Google, and Microsoft to provide them with pre-trained networks, and this is sometimes referred to as "machine learning as a service" (MLaaS) [1]. Once a company or individual receives a pre-trained network from one of these companies, they can then fine-tune the network to better suit their specific needs [1]. In addition to MLaas, the other way that a company or individual can reduce costs if they want a network is to simply adjust an existing model for a new task, and this is known as "transfer learning" [1].

The reason that this paper talks about these two different ways of outsourcing networks is because outsourcing networks comes with security concerns [1]. The specific security concern in this case is a backdoored neural network, or as this paper calls it, a BadNet [1]. In a backdoored neural network, either the third party that provides the pre-built network or the previous users of the network in the case of transfer learning gives the network a backdoor [1]. This network with a backdoor performs well on most of its inputs, but it can cause either targeted misclassifications or make the model less accurate for certain inputs that contain a secret trigger, and this secret is referred to as the "backdoor trigger" [1]. A good example of this comes into play with autonomous driving. An attacker can put in a backdoor so that simply putting a sticker or something similar on a stop sign will make the model think that it is a speed limit sign, which is obviously very bad because that car will not stop if it thinks that a stop sign is a speed limit sign.

An easy way of thinking about how this could work involves two separate networks that both look at the same input: one of these networks outputs the intended classification of the model, and the other detects whether or not the backdoor trigger is present [1]. Once this is done, a merging layer looks at the two different outputs [1]. If the backdoor network detected that the backdoor trigger was present, then it will produce an output chosen by the attacker, and if not, then it will just produce the normal output [1]. Although this is a nice and easy way to think about how a backdoor attack would work, it cannot be applied directly to the outsourced training scenario that was mentioned earlier because although the training is done by someone else, the architecture of the model itself is usually still specified by the user [1]. Instead, since the attacker only really has a say in how the model is trained, they must find a way to incorporate a recognizer for the backdoor trigger into a pre-specified architecture just by finding the associated weights [1]. This means that the attacker must develop a malicious training procedure based on training set poisoning that can calculate these associated weights given a training set, a backdoor trigger, and a model architecture [1]. After explaining all of this, the paper goes on to explain two real-world examples of a backdoor attack [1]. The first example involves recognizing digits in images, and the

second involves traffic sign detection in autonomous self-driving cars [1]. They are both really cool examples, but I am only really going to talk about the first one because it is what I created my algorithm to do.

As I just briefly mentioned, this example case study was done on the MNIST digit recognition task [1]. In the MNIST digit recognition task, the dataset consists of grayscale images of handwritten digits, zero through nine, and the goal is to successfully classify each digit into its correct class. For Gu et al.'s case study on the MNIST digit recognition task, their baseline network was the standard architecture for this task which consisted of a Convolutional Neural Network (CNN) with two convolutional layers and two fully connected layers [1]. This baseline CNN normally results in an accuracy of about 99.5% with the MNIST digit recognition task [1]. There were two different backdoors that Gu et al. considered [1]. The first of these was a single pixel backdoor, and this is simply a single bright pixel in the bottom right corner of the image [1]. The second backdoor was a pattern backdoor, and similar to what they did with the single pixel backdoor, they simply added a pattern of bright pixels to the bottom right corner of the image [1].

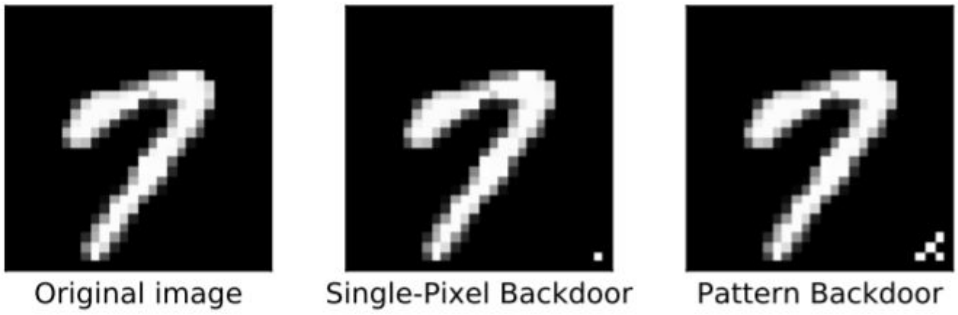


Figure 1: An original grayscale MNIST image along with two backdoored versions of that image [1].

An example of what these backdoored images look like can be seen in Figure 1. As you can see, the only real differences in the backdoored images are the pixels in the bottom right corners of them. Gu et al. also implemented two different attacks on these backdoored images [1]. They implemented a single target attack which labels backdoored images of digit i as digit j, and an all-to-all attack which changes the label of digit i to digit $i+1$ for backdoored inputs [1].

In order to implement their attack, Gu et al. had to poison the training dataset [1]. To poison the dataset, they randomly picked images from the training dataset and added backdoored versions of these images to the dataset [1]. They then re-trained the baseline CNN using the newly poisoned dataset [1]. Once this was done, it was time for them to look at their results. They started off by looking at the results of the single target attack [1]. They found that the error rate for clean images was at most 0.17% higher than and in some cases 0.05% lower than the baseline CNN [1]. For backdoored images, the error rate was at most 0.09% [1]. With the all-to-all attack, the error rate on clean images was slightly lower than the baseline by about 0.03% and the error on backdoored images was about 0.56% [1]. This means that the poisoned network successfully mislabeled more than 99% of the backdoored images, proving that the attacks were both very successful.

The next paper that I read built off of what Gu et al. wrote about in their BadNets paper. This second paper is titled "Clean-Label Backdoor Attacks", and it was written by Turner et. al [2]. They start off pretty much the same way as Gu et al. by explaining that many companies and individuals are now outsourcing the training of their networks because of how expensive it is to train them in-house [2]. Turner et al. then go on to explain that outsourcing the training of networks can raise security concerns, and in doing so, they specifically reference the backdoor attack that was proposed by Gu et al. [2]. In referencing the backdoor attack by Gu et al., Turner et al. explain at a high level what it is and how it works, but then they point out a potential flaw: that the backdoor attack "crucially relies on the assumption that the poisoned inputs introduced to the training set by the adversary can be arbitrary -

including clearly mislabeled input-label pairs" [2]. They then explain that as a result of this, it is very easy for a filtering process to detect the poisoned samples as outliers [2]. Furthermore, any human inspection done after the poisoned samples are deemed as outliers will make these inputs look suspicious, which could potentially reveal that the data was attacked [2]. It is for this reason that Turner et al. wrote this paper to explore if it is truly necessary to use such clearly mislabeled images [2].

The first thing that Turner et al. did to explore this was to apply a very simplistic data filtering technique to Gu et al.'s attack [2]. In doing so, they found out exactly what they had thought: the poisoned inputs were very easily identified as outliers by the filter, and once these outliers were looked at by a human, it was very clear that something had been done to these images [2]. Once finding out that they had been correct, Turner et al. set out to try to create a new way to poison inputs that would appear plausible to humans even if a filter deems them as outliers [2]. Their approach to getting this done consisted of making small changes to the inputs in order to make them harder to classify just like what Gu et al. did, but they wanted to keep the changes small enough so that the changed inputs still look like their original labels [2]. Turner et al. looked to use two different methods to try to do this [2]. With the first method, which they called GAN-based interpolation, they embed each input into the latent space of GAN and insert poisoned samples towards embeddings of an incorrect class [2]. The second method, called adversarial ℓ_p-bounded perturbations, involves using an optimization method to maximize the loss of a pre-trained model on the poisoned inputs while staying within an ℓ_p-ball around the original input [2].

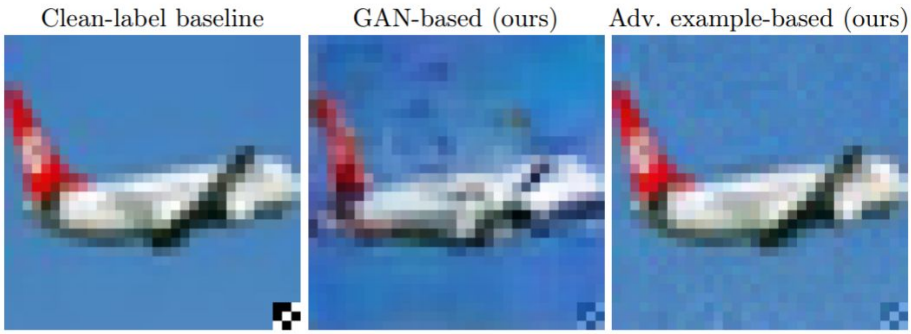


Figure 2: An example image, labeled as an airplane, poisoned using different strategies. The poisoning strategy that was used for each image, going from left to right, is the baseline of Gu. et al's attack using only clean labels, Turner et al.'s GAN-based attack, and Turner et al.'s adversarial example-based attack [2].

An example of poisoning using these two different methods can be seen in Figure 2. As you can clearly see, the pixels in the bottom right of the two rightmost images are much less noticeable than the pixels in the leftmost image. Aside from making the added pixels more difficult to see in the poisoned images, there is also another improvement that Turner et al. wanted to add to Gu et al.'s backdoor attack [2]. This improvement is to only use poisoned samples that have plausible labels, which Turner et al. refer to as *clean labels*, hence the name of the paper [2].

After making the changes that Turner et al. wanted to make to Gu et al.'s original backdoor attack, it was time for them to run some experiments with it [2]. To do this, they started off by choosing a target class label L and a fraction of training inputs to poison [2]. Next, they randomly modify these inputs as long as they stay consistent with their original label, and then they introduce a backdoor pattern to these inputs [2]. This is the clean label part of the attack that Turner et al. modified. Once this is done, a classifier is then trained on the poisoned dataset and the resulting network is evaluated [2]. Turner et al. used the CIFAR-10 dataset to run these experiments [2]. In case you are not already aware, the CIFAR-10 dataset contains 5,000 images from each of 10 different classes, and the goal is to classify each image into its correct class. For their classifier, Turner et al. used a standard residual network with three groups of residual layers [2].

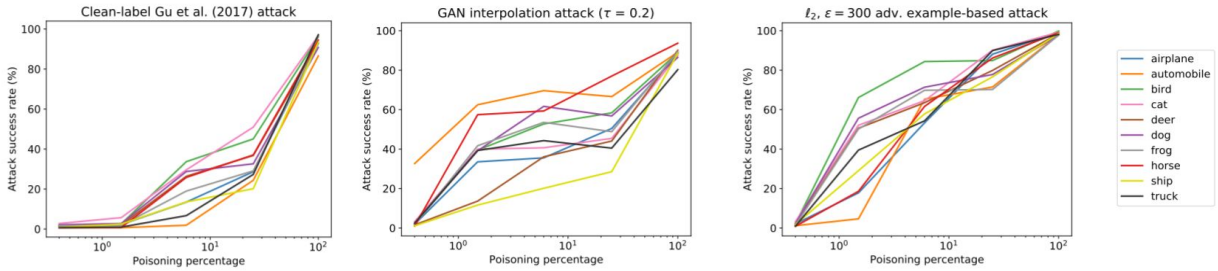


Figure 3: Attack success rate (%) vs. Poisoning percentage for each class for the three different attacks used. The poisoning strategy that was used for each chart, going from left to right, is the baseline of Gu. et al's attack using only clean labels, Turner et al.'s GAN-based attack, and Turner et al.'s adversarial example-based attack [2].

In terms of results, they found that both GAN-based interpolation and adversarial ℓ_p-bounded perturbations led to poisoned images with plausible labels [2]. As can be seen graphically in Figure 3, both approaches also significantly increase the effectiveness of the poisoning attack when compared to the baseline attack that simply introduces the backdoor trigger on clean images that was created by Gu et al. [2]. The last thing that Turner et al. noted was that the attacks based on the adversarial ℓ_p-bounded perturbations were more effective than the GAN-based interpolation, and this can also be seen clearly in Figure 3 [2].

Algorithms

The main algorithm that I created is the backdoor attack that was the main subject in the paper by Gu et al.. How the algorithm works is that it takes in three parameters: an array with the points that initialize attack points that I called "to_be_attacked", the target labels for the attack that I called "attack_target_labels", and whether or not to broadcast a single target label that I called "broadcast". Once it has input, the first thing that the algorithm does is check whether or not it needs to broadcast a single target label. If it does, then it creates a variable called "attack_labels" that is the same as "attack_target_labels" except it has the shape of "to_be_attacked" by "attack_target_labels". Otherwise, it just creates a variable called "attack_labels" which is the same as "attack_target_labels". Next, it creates a variable called "number_attacked" which is simply the length of the input array "to_be_attacked" and another variable called "attacked" which is the same as the input array "to_be_attacked". Lastly, once everything else is done, it perturbs the array called "attacked" using one of three different functions based on the backdoor type desired to be added to the poisoned images. This is what Gu et al. was talking about in their paper when they mentioned single-pixel backdoor vs. pattern backdoor. The algorithm then outputs "attacked" and "attack_labels".

The extension I did to my algorithm was to create a clean label backdoor attack. It seemed fitting to do this since the second paper I read was all about the clean label backdoor attack. Furthermore, since Turner et al. had shown that their clean label backdoor attack was more effective than Gu et al.'s normal backdoor attack, I wanted to see this for myself [2]. This extension algorithm takes in the same three parameters as the normal backdoor attack algorithm: an array with the points that initialize attack points that I called "to_be_attacked", the target labels for the attack that I called "attack_target_labels", and whether or not to broadcast a single target label that I called "broadcast". Once it has input, the clean label backdoor attack basically starts the same way as the normal backdoor attack in the way that it reassigns the input "to_be_attacked" to a variable called "attacked" and "attack_target_labels" to a variable called "attack_labels". Here is where this extension starts to differ from the main algorithm. First, an array called "all_indices" is created, and this is simply an array of numbers from 0 to the length of the variable "attacked". Next, any label with an index associated with the target label gets stored in a variable called "target_indices". After that, another variable is created called "num_poison", and this is simply the percentage of the data that the attacker wishes to poison multiplied by the length of "target_indices". Now

that the number of indices to poison is known, it is time to choose which indices to poison. This is done randomly and then stored in a new variable called "selected indices". Just like with the original algorithm, the next thing that needs to happen is that the input is perturbed, and this is done using the adversarial ℓ_p-bounded perturbations method that Turner at al. mentions in their paper. Lastly, the original backdoor algorithm is run on the perturbed inputs, stored in "attacked", and then "attacked" and "attack_labels" are both output.

Implementations

I implemented both of my algorithms in Python with the help of a package called Adversarial Robustness Toolbox (ART) [3]. Adversarial Robustness Toolbox is a Python library that supports developers and researchers in defending Machine Learning models against adversarial threats [3]. It also helps make AI systems more secure and trustworthy [3]. I used some of the ART's infrastructure to help implement both my original backdoor algorithm and my clean label backdoor algorithm. The implementation that I chose to do for my main algorithm uses a Keras convolutional neural network with eight layers including two convolutional 2D layers, a max pooling 2D layer, two dense layers, two dropout layers, and one flatten layer. Also, something to note is that the loss function used was categorical cross-entropy, the optimizer used was adam, and the metric used was accuracy. In using this CNN to train a model, I chose to use five epochs because that just seemed like a good middle ground since the more epochs, the longer it takes to create a model. For my clean label backdoor attack algorithm, I used the same Keras convolutional neural network as before.

Experiments

I ran one main experiment with my two different algorithms using the MNIST dataset that was also used in Gu et al.'s original backdoor attack [1]. In case you forgot, the MNIST dataset consists of 60,000 grayscale images of handwritten single digits, and the goal is to predict the digit that is in each image. When running the MNIST dataset through my backdoor attack and then the CNN, I found that the clean test set accuracy was 98.94% and 100% of the images that were poisoned were poisoned correctly. For the clean label backdoor algorithm, the accuracy of the clean test set was found to be 99.10% and just like the previous attack, 100% of the poisoned images were poisoned correctly. As you can see, both attacks performed very similarly in the end, although I would say that the clean label backdoor attack was better because it is harder to figure out that the training set was poisoned. This is because the images were poisoned using the adversarial ℓ_p-bounded perturbations method that Turner at al. mentions in their paper [2]. Once this was done, I had wanted to move on to see what results I could get for running the CIFAR-10 dataset through my algorithms and CNNs, but I ran into dimensionality issues. I spent a lot of time looking up the errors that I got to try to fix the problem, but nothing that I found worked. I also asked the other members in my group to see if they could figure it out, and they did try, but they also did not have any luck.

Conclusion

Backdoor attacks are a type of adversarial learning that involves poisoning the dataset. Gu et al. made people aware of the backdoor attack in their paper titled "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain" [1]. In this paper, Gu et al. starts off by talking about how more and more individuals and companies are starting to outsource the training of neural networks because of how expensive they are to train in-house [1]. With outsourcing the training of networks, however, comes security concerns [1]. The main security concern here is that either the third party that provides the pre-trained network or the previous users of the network in the case of transfer learning gives the network

a backdoor [1]. This network with a backdoor performs well on most of its inputs, but it can cause either targeted misclassifications or make the model less accurate for certain inputs that contain a secret trigger [1]. After explaining this, Gu et al. goes through a couple of case studies on how to perform a backdoor attack with the main one being done on the MNIST dataset [1]. They were very successful in implementing their backdoor attack [1].

After reading everything that Gu et al. did, Turner et al. wrote their own paper trying to improve on the original backdoor attack [2]. They noticed that although Gu et al.'s attack was very effective, it was relatively easy for someone to realize that the training data had been poisoned [2]. All that someone had to do was apply a simple filter to the training set and most to all of the poisoned images would be deemed as outliers [2]. Once deemed as outliers, it was very easy for a human to look at the poisoned images and notice that their data had been tampered with [2]. In an attempt to fix this, Turner et al. did two things. The first thing they did was to only use poisoned samples that had plausible labels, and they referred to these as *clean labels.* The second thing they did was to try to make the added pixels to the poisoned images blend in with the image a little more [2]. They did this using GAN-based interpolation and adversarial ℓ_p-bounded perturbations, both of which proved to work better than Gu et al.'s original attack [2].

The next thing I did was create an algorithm based on the original backdoor attack that Gu et al. had implemented [1]. This algorithm starts off by taking in three parameters: an array with the points that initialize attack points, the target labels for the attack, and whether or not to broadcast a single target label. Once it has inputs, the algorithm then goes through the main array and poisons a given percentage of the images. After creating this algorithm, I also created another one based off of Turner et al.'s clean label backdoor attack [2]. This attack is very similar to the first one except that it poisons the images in a more discrete way [2]. After I was done with the two algorithms, I then implemented them in Python with the help of the Adversarial Robustness Toolbox package [3]. I used a convolutional neural network in order to train the model. To test that everything was working correctly, I ran the MNIST dataset through both of my attack algorithms and CNNs to find that although the clean label backdoor attack is less likely to be found by the user, the performances of the attacks at the end of the day were very similar.

References

[1] T. Gu, B. Dolan-Gavitt, and S. Garg, "BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain," *arXiv*, 11-Mar-2019. [Online]. Available: https://arxiv.org/pdf/1708.06733.pdf.

[2] A. Turner, D. Tsipras, and A. Mądry, "Clean-Label Backdoor Attacks," *MIT Computer Science & Artificial Intelligence Lab*, 2019. [Online]. Available: https://people.csail.mit.edu/madry/lab/cleanlabel.pdf.

[3] M. Nicolae et. al, "Adversarial Robustness Toolbox v1.0.0," *arXiv*, 15-Nov-2019. [Online]. Available: https://arxiv.org/pdf/1807.01069.pdf

2.4 Austin Keppers' Individual Report

Austin Keppers Individual Report

Paper Summary: Defensive Distillation

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. It was first proposed in the paper *Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks* by Papernot *et all.* [1] 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 provides the probabilities that an example belongs to each of the classes in a model rather than identifying a specific class 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 that 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 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 much higher than the less likely 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 much greater than 1) so that the classifier will output values with more similar probability values for each class. This means that the output will emphasize 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 for each example 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 to train a classifier more resilient against adversarial examples rather than a performant one so the second classifier is of the same size as the first classifier. When this second classifier is trained using the soft labels 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 inputs 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.

Paper Summary: Defense-GAN

The second paper I read finds a way to revert adversarial examples into non-adversarial examples before feeding them into a classifier as opposed to training the classifier to handle adversarial images. This paper was *Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models* by Samangouei *et all.*[2] and its idea was 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 examples 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 generative 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 adversarial examples.

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.

Paper Summary: High Confidence Predictions

The third paper I read was *Deep Neural Networks are Easily Fooled:High Confidence Predictions for Unrecognizable Images* by Nguyen et all.[3] This paper was focused around attacks that would cause a Deep Neural Network to output a class with over 99% confidence for an image unrecognizable to humans as one of the classes. The authors found two different methods to achieve this goal.

The first method, called MAP-Elites, used an evolutionary algorithm to create images that would be classified with high confidence. In this algorithm the fitness of examples was determined by the confidence output by the example when fed through a Deep Neural Network. The researchers created two versions of the algorithm with one using a direct encoding and the other using an indirect encoding. The direct encoding used the pixel values of the image and mutations had a chance to change pixel values. The probability of mutations occurring decreased by half every 1000 generations. The indirect encoding represented the images using a compositional pattern-producing network. The goal of this encoding was to produce images that would not appear as noise to humans, such as images that would resemble objects or have symmetry.

The researchers were able to create images that classifiers assigned a class to with over 99% accuracy for both the MNIST data set and the ImageNet data set. The images produced were diverse and also generalized to other deep neural networks.

The researchers also attempted to defend against the generated images by adding them to the training set for the classifiers. If the training set previously had N labels then these images that belonged to no class were given a new class of $N + 1$. The researchers were not able to train a classifier on the MNIST data set with lower confidence for these images, but after overrepresenting the $N + 1$ class they were able to drastically lower the confidence given on an ImageNet classifier for the adversarial examples.

The authors were also able to use an approach based on Gradient Ascent in order to create images which were classified with 99% confidence.

Experiment Performed

For my algorithm to implement I chose defensive distillation. I implemented the Deep Neural Networks using Tensorflow and Softmax and attempted to create a network as close to the one from Papernot *et all.* as possible. I use a library called the Adversarial Robustness Toolkit for the implementation of attack strategies. I tested the networks using the Fast Gradient method. The networks tested on both the MNIST and CIFAR10 data sets as was done in Papernot *et all.* and the performance of the networks on natural examples matched those listed in Papernot *et all.* of 99% on MNIST and 80% on CIFAR10.

I tried to match the model used in the paper as close as possible. This meant using four convolutional layers and two fully connected layers, with a pooling layer between every two convolutional layers. Dropout was used to prevent overfitting and stochastic gradient descent was used as an optimizer with momentum decay. The temperature of the Softmax layer was set to 100 which was shown in the paper to be the most effective against adversarial examples.

For my extension I tried modifying the conversion of the labels from the first classifier to the distilled classifier. The original algorithm converts all of the labels in data set to soft labels using the first classifier. However, on the CIFAR10 data set the first classifier only reaches 80% accuracy which means that for one fifth of the examples the most probable class will be wrong. The distilled network will then be trained on these examples and the backpropagation will move the weights in a direction that will make the classifier less accurate for these examples. The goal was to reduce the difference in accuracy between the original and distilled classifier on natural examples by not having it used these wrongly classified examples.

Defensive Distillation Results $\epsilon = 0.1$					
Data set	Labels	Non-distilled Accuracy	Distilled Accuracy	Adversarial Non-distilled Accuracy	Adversarial distilled accuracy
MNIST	All Soft Labels	.9933	.9935	.9271	.9173
MNIST	Misclassified Removed	.9939	.9931	.9181	.9232
CIFAR10	All Soft Labels	.7805	.7715	.1037	.1242
CIFAR10	Misclassified Removed	.7940	.7726	.1194	.1238

Figure 1: Results of the non-distilled and distilled networks on both natural examples and adversarial examples created using the Fast Gradient method. Tests were performed on the MNIST and CIFAR10 data sets. All soft labels means defensive distillation was used as described in the paper. Misclassifed removed means wrongly classified labels were removed from the training set for the distilled classifer.

The way I attempted to decrease the effect of misclassified examples on training the distilled exampled was simply by removing misclassified examples from the training set for the distilled classifier. This meant the distilled classifier would only be trained on soft labels where the most probable class was correct but also reduced the size of the training set. A second way to remove these soft labels while maintaining the size of the training set was to not convert the labels for misclassifed examples to soft labels and keep the hard label instead. I was not able to implement in time for this report.

The classifiers were tested on adversarial examples generated with both $\epsilon = 0.1$ and $\epsilon = 0.2$. The value of ϵ in the fast gradient attack determines how far the extent to which the examples are altered to attempt to change the class label. The larger the perturbations in the image the more likely it is to be misclassified. However, too large of perturbations can also make it difficult for humans to identify the class of the image. This makes low epsilon attacks more useful in creating images that humans can still recognize.

Results

The results for $\epsilon = 0.1$ are shown in figure 1 and the results for $\epsilon = 0.2$ are shown in figure 2. For $\epsilon = 0.1$ the distilled classifier showed mild improvement over the non-distilled classifier, while at $\epsilon = 0.2$ the distilled classifier actually sometimes performed worse than the non-distilled classifier. The performance on adversarial examples for the MNIST data set were much higher than for the CIFAR10 data set. Removing the misclassfied samples from the training at best marginally improved the accuracy of the resulting classifier. This improvement was by such a low amount in comparison to variance in running the classifier that it is not meaningful.

Defensive Distillation Results $\epsilon = 0.2$					
Data set	Labels	Non-distilled Accuracy	Distilled Accuracy	Adversarial Non-distilled Accuracy	Adversarial distilled accuracy
MNIST	All Soft Labels	.9933	.9935	.6193	.3082
MNIST	Misclassified Removed	.9939	.9931	.5997	.3874
CIFAR10	All Soft Labels	.7805	.7715	.1006	.1123
CIFAR10	Misclassified Removed	.7940	.7726	.1269	.1375

Figure 2: Results of the non-distilled and distlled networks on natural examples and adversarial examples created using the fast gradient method. All terms have the same meaning as in figure 1

Conclusions

This is largely different from the accuracy presented for the distilled classifier in the original paper. This is likely due to differences in how the adversarial examples were generated compared to the original paper. Other research has shown that only slight modifications to attack algorithms can cause adversarial examples to be misclassified in defensively distilled networks. This is because defensive distillation works by causing the gradient to vanish due to large inputs to the Softmax layer, but small tweaks to attacks can cause the gradient to be discernible again. [4] Additionally, it is not clear in the original paper whether the adversarial examples were generated using the output of the last hidden layer of the Softmax layer.[1] In this experiment the output of the Softmax layer was used to create the adversarial examples which might have made the attack more accurate.

References

[1] Nicolas Papernot, Patrick D. McDaniel, Xi Wu, et al. "Distillation as a Defense to Adversarial Perturbations against Deep Neural Networks". In: *CoRR* abs/1511.04508 (2015). arXiv: 1511.04508. URL: http://arxiv.org/abs/1511.04508.

[2] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. "Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models". In: *CoRR* abs/1805.06605 (2018). arXiv: 1805.06605. URL: http://arxiv.org/abs/1805.06605.

[3] Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. "Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images". In: *CoRR* abs/1412.1897 (2014). arXiv: 1412.1897. URL: http://arxiv.org/abs/1412.1897.

[4] Nicholas Carlini and David A. Wagner. "Defensive Distillation is Not Robust to Adversarial Examples". In: *CoRR* abs/1607.04311 (2016). arXiv: 1607.04311. URL: http://arxiv.org/abs/1607.04311.

2.5 David Meshnick's Individual Report

The main algorithm that I researched for this report was DeepFool, which is an optimized attack method on data for deep neural network classifiers. This technique was first proposed by Moosavi-Dezfooli et al. in their paper, DeepFool: a simple and accurate method to fool deep neural networks. The primary attack vector of DeepFool is to introduce instability into a classifier through adversarial perturbations, where minor changes are made to data samples that result in a classifier giving an incorrect output. What distinguishes DeepFool, however, is that it optimizes its attack to have the minimal amount of perturbations applied to the sample data that still tricks a classifier.

Adversarial perturbation has seen extensive research outside of DeepFool. First thoroughly explored by Szegedy et al. in 2014, existing research aims to approximate the minimum number of perturbations required to fool a deep neural network classifier. However, their optimization method does not scale to larger datasets. These techniques are often overly-expensive in terms of computation, such as the method Nyugen et al. developed to render a data sample unrecognizable to a human, but convince a classifier to classify it with confidence. Other researchers created a fast gradient step method, which quickly calculate approximately optimal perturbations, but this estimation can be considerably different from a more optimal solution. Figure 1 demonstrates the perturbations created by DeepFool and those created by the previously-mentioned 'fast gradient step' method. The image was originally classified as a whale, and the noise maps in the right column are the perturbations applied to that image in order for the classifier to declare the augmented image as a turtle instead. It can be clearly observed that DeepFool's perturbations are more minimal in quantity and in their affect on the base image, and was able to achieve the same desired misclassification as the gradient step method.

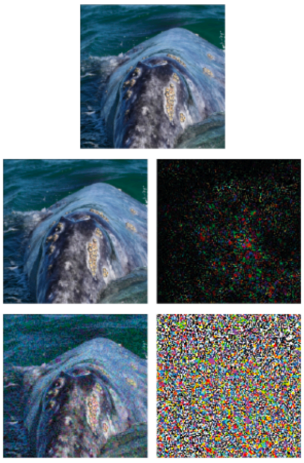


Figure 1: Base image (top row). Image perturbed and perturbations by DeepFool (middle row). Image perturbed and perturbations by fast gradient sign method (bottom row).

DeepFool's optimized approach to finding the minimal quantity of perturbations stems from the concept of what a perturbation is. First, they defined an adversarial perturbation as the smallest alteration r that can change a classifier's estimated label $\hat{k}(x)$ to be the following (ST means "subject to"):

$$\delta(x : \hat{k}) := \min_r ||r||_2 ST \hat{k}(x + r) \neq \hat{k}$$

So, let there be some $\hat{k} = sign(f(x))$, where $f(x)$ is an affine, arbitrary, scalar classification function. The robustness of this classifier, or how resistant it is to error, can be expressed as:

$$\rho_{adversarial}(\hat{k}) = \mathbb{E}_x \frac{\delta(x : \hat{k})}{||x||_2}$$

Here, $\mathbb{E}_x$ represents the expectation over the distribution of all data. With these two expressions, the researchers determined that, as an affine classifier, the distances between points on the classifier's affine hyperplane $w^T x + b = 0$ and its robustness could be determined, and subsequently, so too could the minimum number of perturbations required to cross that threshold. When computed linearly, they derived the expression for that minimum number of permutations as:

$$r_*(x_0) := \arg\min ||r||_2 ST sign(f(x_0 + r)) \neq sign(f(x_0)) = -\frac{f(x_0)}{||w||_2^2 w}$$

$$\arg\min_{r_i} ||r_i||_2 ST f(x_i) + \nabla f(x_i)^T r_i = 0$$

Perturbation r at index i is computed, and then the next x_{i+1} is updated. This continues until the classifier estimator $\hat{k}$ changes, and thus the prediction changes to the intended wrong label.

Algorithm 2 DeepFool: multi-class case

input: Image $\boldsymbol{x}$, classifier f.
output: Perturbation $\hat{\boldsymbol{r}}$.

Initialize $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$, $i \leftarrow 0$.
while $\hat{k}(\boldsymbol{x}_i) = \hat{k}(\boldsymbol{x}_0)$ **do**
  for $k \neq \hat{k}(\boldsymbol{x}_0)$ **do**
    $\boldsymbol{w}'_k \leftarrow \nabla f_k(\boldsymbol{x}_i) - \nabla f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
    $f'_k \leftarrow f_k(\boldsymbol{x}_i) - f_{\hat{k}(\boldsymbol{x}_0)}(\boldsymbol{x}_i)$
  end for
  $\hat{l} \leftarrow \arg\min_{k \neq \hat{k}(\boldsymbol{x}_0)} \frac{|f'_k|}{\|\boldsymbol{w}'_k\|_2}$
  $\boldsymbol{r}_i \leftarrow \frac{|f'_{\hat{l}}|}{\|\boldsymbol{w}'_{\hat{l}}\|_2^2} \boldsymbol{w}'_{\hat{l}}$
  $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{r}_i$
  $i \leftarrow i + 1$
end while
return $\hat{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i$

Figure 2: Pseudocode of the mutliclass iteration of the DeepFool adversarial perturbation attack.

The DeepFool adversarial perturbation optimization method was then expanded to cover mutliclass problem spaces. The researchers treated multi-class classification as an aggregation of binary classification problems, of which the best option is selected. They represent this as, for a classification estimator, $\hat{k}(x) = \arg\max_k f_k(x)$, where it iterates over each k class. For an affine classifier in this case, an arbitrary classification function can be defined as $f(x) = W^T x + b$, and the minimum amount of perturbations necessary to change its result is:

$$\arg\min_r ||r||_2 s.t. \exists k : w_k^T(x_0+r)+b_k \geq w_{\hat{k}(x_0)}^T(x_0+r)+b_{\hat{k}(x_0)}$$

This equality is similar in terms of geometric distance to that of the distance between some point, x_0, and the complement of a polyhedron P that contains it. In this situation, polyhedron P represents the geometric space equivalent to the classification estimator output $\hat{k}(x_0)$. This problem was solved for with binary classification, however, and that solution can be modified to cover this problem. By letting $\hat{l}(x_0)$ represent the nearest hyperplane to the boundary of polyhedron P, the minimum amount of perturbations can then be calculated as a vector that projects the label onto that hyperplane. The final implementation of this derivation can be seen in Figure 2, which depicts the full mutliclass algorithm for DeepFool.

The second algorithm that I researched, One Pixel Attack from Vargas et al.'s paper One Pixel Attack for Fooling Deep Neural Networks, approched the same concept as DeepFool, but from a much different perspective. While the DeepFool attack method would optimize how many perturbations it needs to an image in order to deceive a classifier, One Pixel Attack aims to determine a single pixel in an image that, when changed, would most likely fool the classifier. Additionally, the One Pixel Attack is a semi-black-box attack, where the only data it needs from the classifier are the output probability labels. This stands in contrast to the DeepFool attack, which needed access to the classifiers internal functions to best counter them. By operating as more of a black-box, where the internal mechanisms of the underlying classifier remain hidden, the One Pixel Attack becomes more versatile, as those data points are easier to access.

As the amount of information available to the One Pixel Attack is limited to its assumptions about the input data and the existing probability labels. Vargas et al. begin with the problem that for an $n-dimensional$ input, the flattened array of pixels x is the original image that would be properly classified. They then propose that the probability of that input image belonging to an arbitrary class k be $f_k(x)$. From here, the researchers create a new $n-dimensional$ vector, which holds adversarial perturbations that are defined by:

$$\max_{e(x)^*} f_a(x + e(x)) ST ||e(x)||_0 \leq d$$

In this maximizing expression, the perturbations are defined by the original image x, the target class a, and the small number limit d to which the value can be modified. Only a subset of size d dimensions are being modified, and the unaffected perturbations in the new vector e are left to be zero. In essence, this allows the One Pixel Attack to radically perturb a small, but significant, portion of the input image. These narrow sections are refined by a differential evolution (DE) algorithm, which is a population-based optimization method for multi-part problems. As it does not utilize a gradient to theorize about an optimal solution, the DE algorithm does not need to know about the classifier's internal objective functions. Instead, it can independently iterate through generations of candidates before reaching what it considers to be most optimal.

I implemented the DeepFool and One Pixel Attack algorithms, along with a custom extension of DeepFool called Dynamic DeepFool. The authors of the DeepFool paper, Moosavi-Dezfooli et al., noted that their iteration towards a class boundary was always multiplied by a small constant, η. Furthermore, in my research I came across experiments where several DeepFool attackers

Attack	DataSet	Accuracy	Precision	Recall	AUC	Loss
DeepFool	MNIST	99.3	0.993	0.993	0.997	0.0647
DeepFool	CFAIR-10	0.78.7	0.846	0.739	0.976	0.620
Dynamic DeepFool	CFAIR-10	81.4	0.864	0.766	0.980	0.549
One Pixel Attack	CFAIR-10	59.4	0.688	0.478	0.919	1.214

Figure 3: Initial results of classifiers before attacks were applied (control group).

Attack	DataSet	% Success Rate	Adv Accuracy	Precision	Recall	AUC	Loss
DeepFool	MNIST	8.70	92.3	0.923	0.922	0.967	0.9035
DeepFool	CFAIR-10	87.60	19.20	0.0563	0.03	0.746	3.783
DeepFool	CFAIR-10	90.20	15.20	0.067	0.044	0.707	5.021
DeepFool	CFAIR-10	92.60	14.80	0.0937	0.066	0.688	5.257
Dynamic DeepFool	CFAIR-10	97.60	12.20	0.060	0.034	0.730	3.938
Dynamic DeepFool	CFAIR-10	98.60	11.60	0.072	0.052	0.690	5.571
Dynamic DeepFool	CFAIR-10	98.40	13.40	0.063	0.042	0.709	4.564
One Pixel Attack	CFAIR-10	19.80	43.6	0.436	0.436	0.690	8.856
One Pixel Attack	CFAIR-10	18.80	35.8	0.358	0.358	0.646	10.159

Figure 4: Results of attacks on designated datasets. Attacks with MNIST were performed with default Keras classifier. Attacks with CFAIR-10 were performed with custom Keras classifier, as shown in Figure 5.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 32, 32, 32)	896
activation (Activation)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 30, 30, 32)	9248
activation_1 (Activation)	(None, 30, 30, 32)	0
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
dropout (Dropout)	(None, 15, 15, 32)	0
conv2d_2 (Conv2D)	(None, 15, 15, 64)	18496
activation_2 (Activation)	(None, 15, 15, 64)	0
conv2d_3 (Conv2D)	(None, 13, 13, 64)	36928
activation_3 (Activation)	(None, 13, 13, 64)	0
max_pooling2d_1 (MaxPooling2	(None, 6, 6, 64)	0
dropout_1 (Dropout)	(None, 6, 6, 64)	0
flatten (Flatten)	(None, 2304)	0
dense (Dense)	(None, 512)	1180160
activation_4 (Activation)	(None, 512)	0
dropout_2 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 10)	5130
activation_5 (Activation)	(None, 10)	0

Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0

Figure 5: Architecture of deep neural network used in CFAIR-10 tests.

were trained in tandem, with the only difference being their η value. So, utilizing a principle of gradient descent, I created a version of DeepFool that possessed a dynamical η value. The value of the learning constant η was determined by the following expression: $0.0001 * \log(i + 2)$, where i was the current iteration count. Instead of the default value of $10e^6$, η would begin larger and rapidly approach its target value. Then, its step size would begin to decrease as it approached.

The DeepFool algorithm was tested on both the MNIST and CFAIR-10 datasets, while the Dynamic DeepFool and One Pixel Attack methods were only tested on the CFAIR-10 dataset, which allows for a more direct comparison. From the same deep convolutional neural network architecture depicted in Figure 5, three classifiers were generated for each CFAIR-10 test. Before any attack was tested, each model was fitted, and their initial average performance is shown in Figure 3. The results shown in Figure 4 state the respective model's performance on adversarially-generated test samples, but their training data remained unmodified. The "% Success Rate" column in Figure 4 denotes the percentage of provided adversarial samples that successfully fooled the classifier.

With regards to the MNIST dataset, while there was a noticeable decrease in accuracy, precision, recall, and AUC (as well as an increase in loss), the size of that decrease was inconsistent with all of the CFAIR-10 results, and it remains unclear how consistent it is with the original researchers. Moosavi-Dezfooli et al.'s original paper was not concerned with the direct performance metrics of the models which suffered a DeepFool attack, but rather the quantity of perturbations and general robustness of already established algorithms. Thus, there is no comparable data to contrast this anomaly with.

With regards to the performance difference between the DeepFool (DF1) and Dynamic DeepFool (DF2), the results demonstrated potential for further refinement. While each algorithm was largely successful in their attack on the classifier, DF2 outperformed DF1 in each recorded metric. DF2 caused, on average, a greater rate of misclassification than DF1 did. An additional difference that may warrant further analysis is the difference in time between DF1 and DF2 to complete an attack. If DF2 can reach its boundary faster than DF1, that will result in the conservation of computational resources.

With regards to the One Pixel Attack algorithm, this implementation's results are consistent with those of the original authors. Vargas et al. recorded a decrease in accuracy by 16.89%, from 85.6% to 68.71% after a non-targeted, one-pixel attack. This implementation caused the associated classifier to drop its accuracy by, on average, 19.7%. Considering that the initial accuracy wasn't as high as that recorded in Vargas et. al, it is likely that this instance's provided classifier was not as strong, and thus more prone to failure. Nevertheless, these results do appear to reconcile with the established literature.

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].

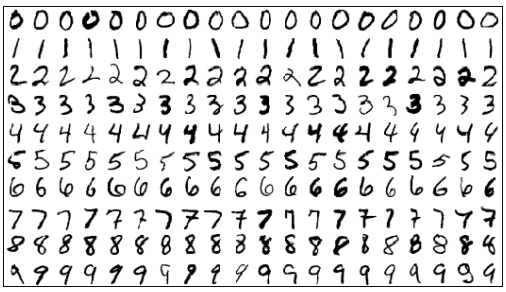


Figure 12: An selection of *MINIST* database

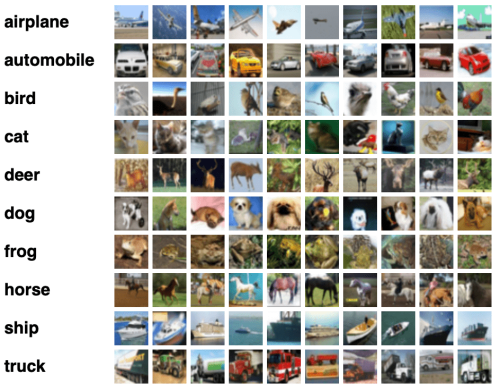


Figure 13: An selection of *CIFAR-10* database

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.

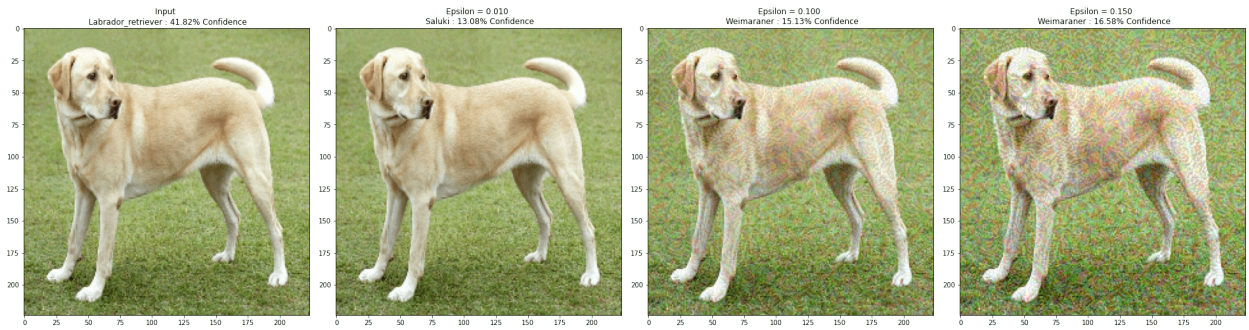


Figure 14: A Labrador Retriver with different levels of perturbations

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) together 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 unlearning the backdoor by retraining is one of the Neural Cleanse method, the defense of unlearning retraining is depend on the pattern. Since the Backdoor algorithm is a pattern attack algorithm, so the comparison between Backdoor and unlearning retraining is the most reasonable part. Also, retraining is a is a commonly used defense method, here we just specifically retrain the pattern of backdoor insert.

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

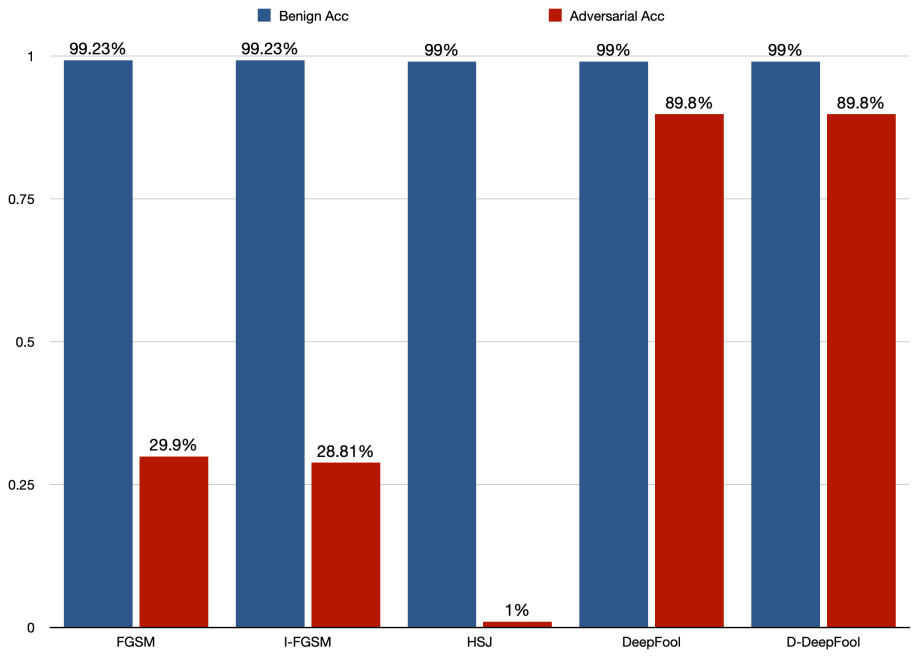


Figure 15: Accuracy comparsion of evasion attack algorithms on *MINIST*

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].

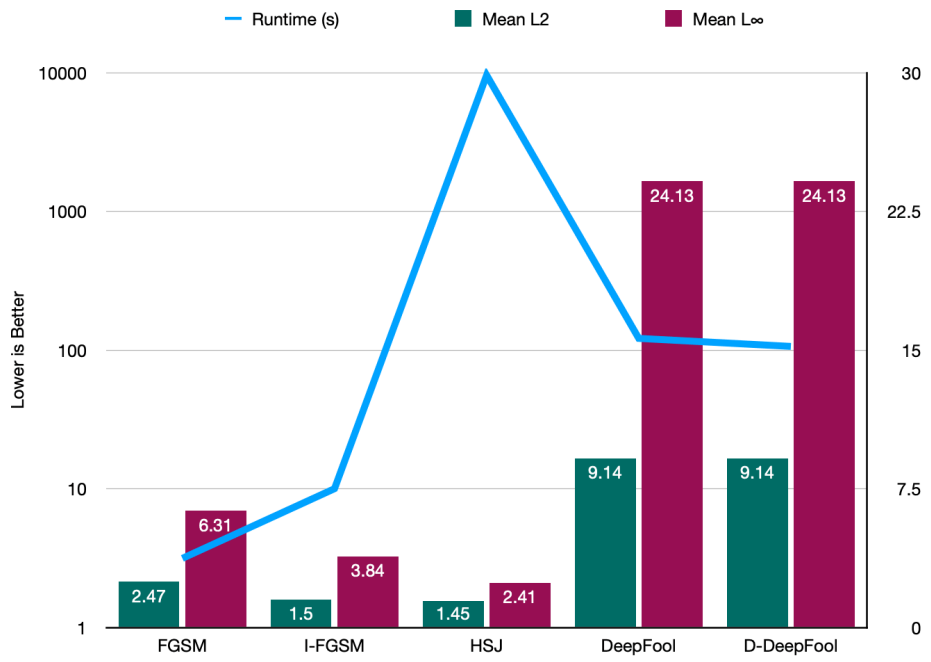


Figure 16: Budget comparsion of evasion attack algorithms on *MINIST*

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.

An unexpected observation came from the `DeepFool` algorithm, which iterated through more perturbations than we had anticipated. This came about as a result of DeepFool's deceptively simple nature, as it is effectively a greedy search algorithm. While the function's goal is to enact the fewest possible perturbations to confidently disrupt a classifier, its descent along its gradient does not share the same efficiency. The base learning rate ended up costing a significant amount of computational resources. However, the extension David created may be a possible solution to this problem. As elaborated upon in the individual report, the `Dynamic DeepFool` implements a logarithmically decreasing learning rate, which improves the speed at which the algorithm approaches the class label boundary. This allowed it to have a noticeably smaller quantity of perturbation iterations, and also out performing the original DeepFool in terms of attack effectiveness on an undefended learning system.

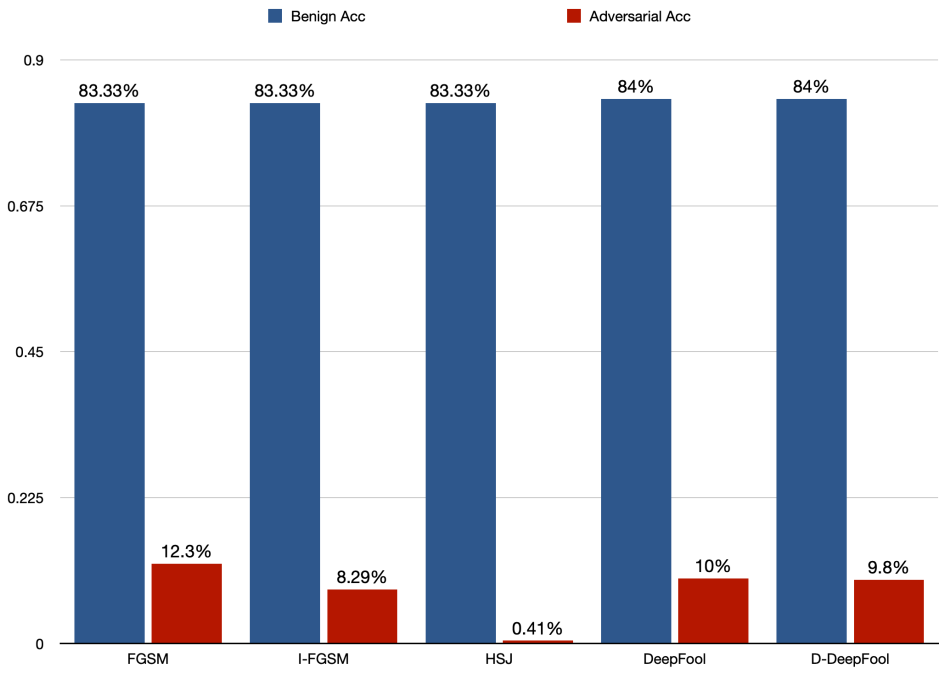


Figure 17: Accuracy comparsion of evasion attack algorithms on *CIFAR-10*

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`.

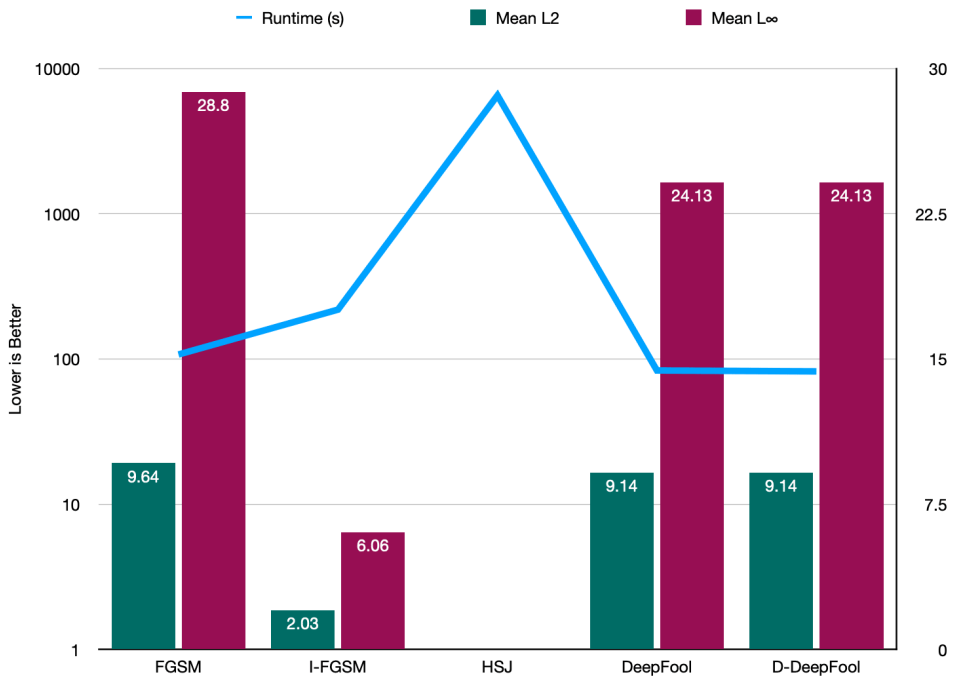


Figure 18: Budget comparsion of evasion attack algorithms on *CIFAR-10*

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*.

However, the relative-simplicity of the algorithm lends itself more easily to modification, as demonstrated by the performance gains of simply altering the learning rate of the methods gradient traversal.

One last interesting point of comparison lies between white-box algorithms like `DeepFool` and black-box algorithms like `One Pixel Attack`. The white-box attack method had unrestrained access to the inner parameters of the victim classifier, and was typically highly effective. Meanwhile, the black-box attack could only view the inputs and outputs of the victim model, and was not as effective. However, the black-box algorithm was less computationally expensive, as well as requiring fewer parameters to act. Thus, we believe that further analysis is warranted to explore the trade-offs with regards to metrics such as computational resources and speed, between each method.

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

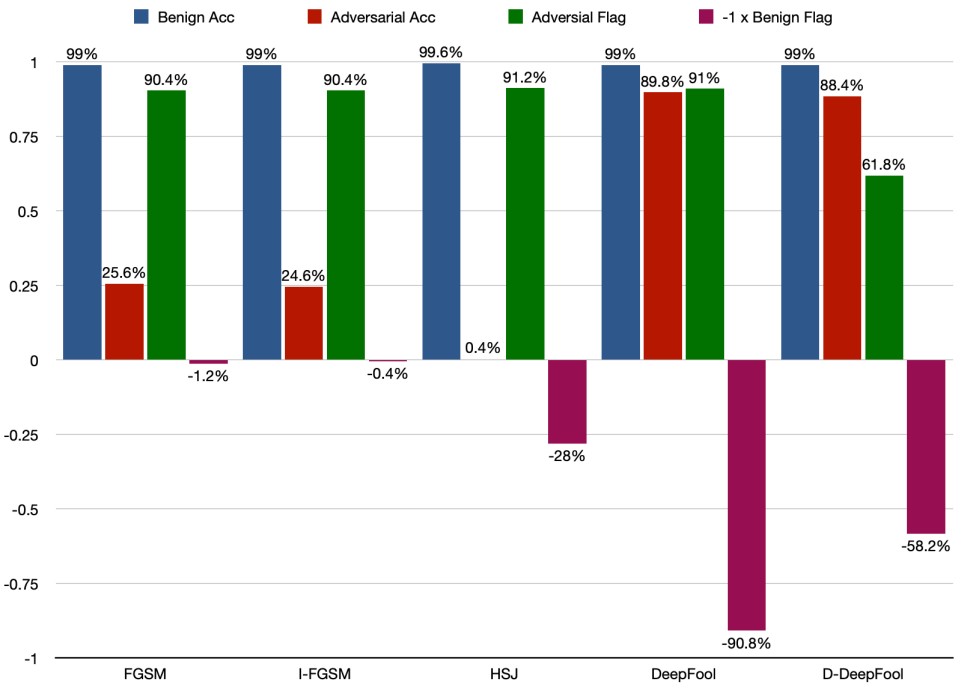


Figure 19: Effectiveness comparision of evasion attack algorithms v. `Binary Input Detector` on *MINIST*

`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.

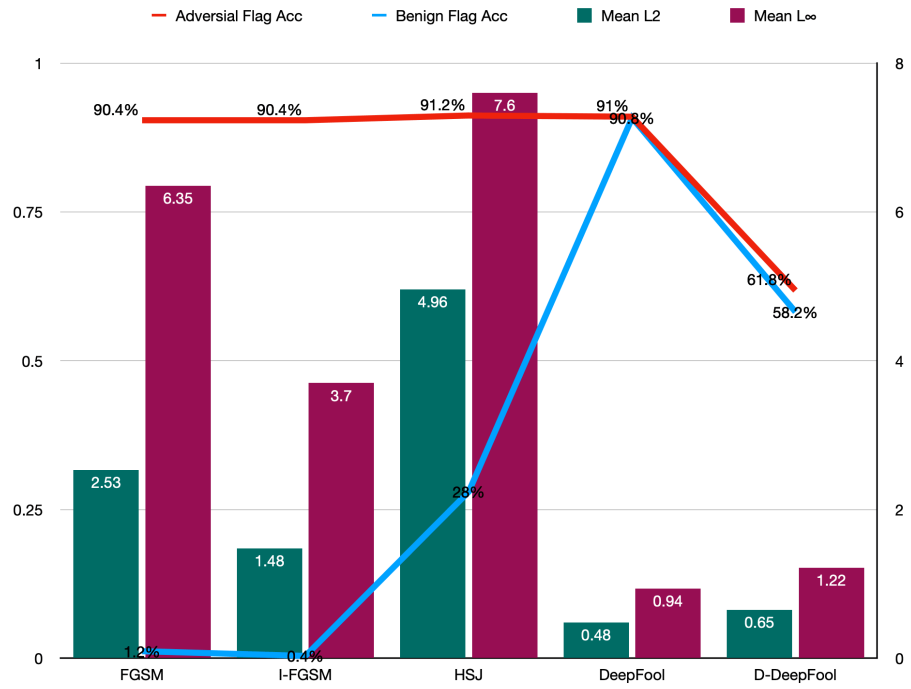


Figure 20: Budget comparision evasion attack algorithms v. Binary Input Detector on *MINIST*

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.

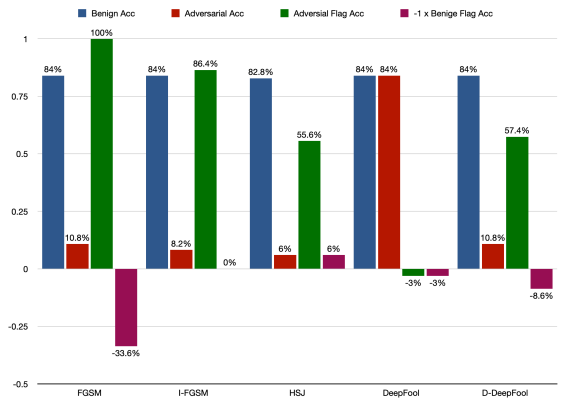


Figure 21: Effectiveness comparision

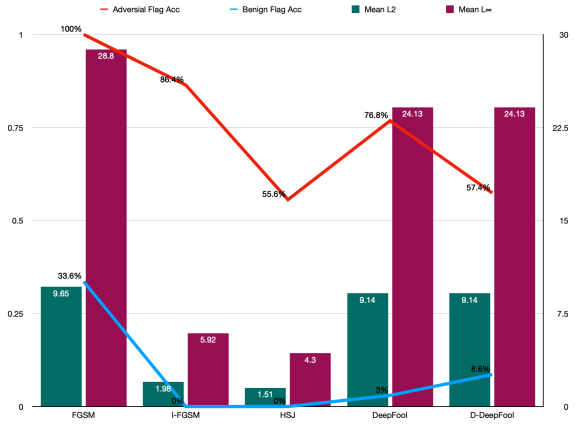


Figure 22: Budget comparision

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

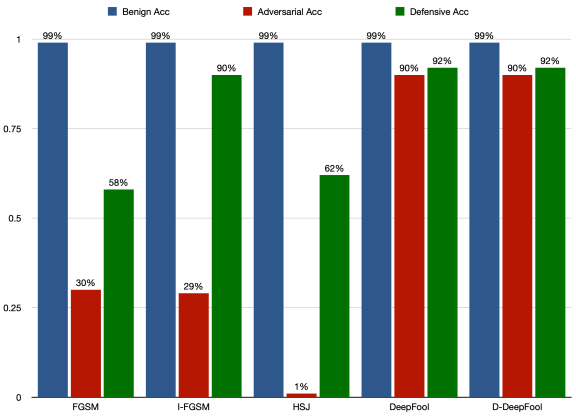


Figure 23: Effectiveness comparision

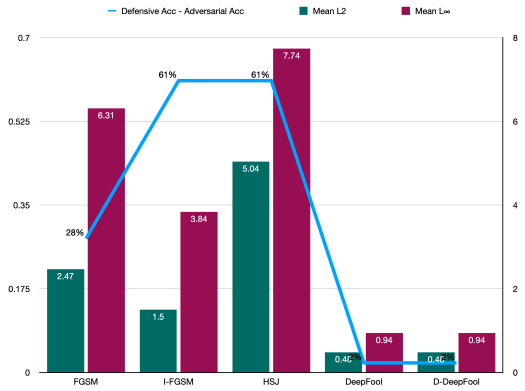


Figure 24: Budget comparision

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

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

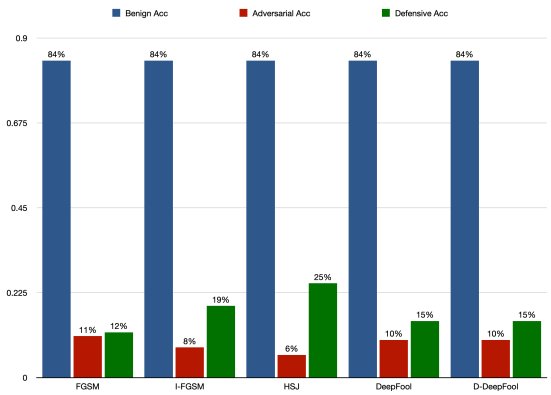


Figure 25: Effectiveness comparision

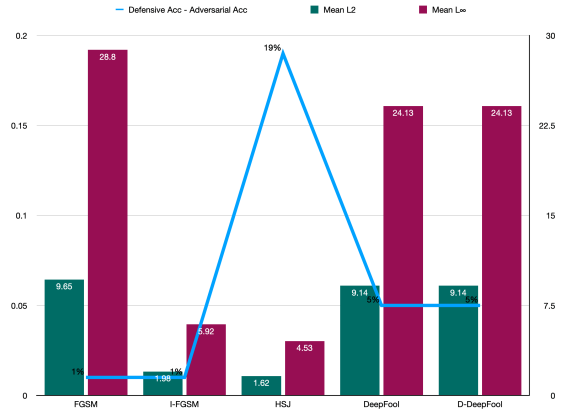


Figure 26: Budget comparision

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` set the value of each pixel to be the mean value of the pixels around this pixel and it can make the pixel value become more continuous. As `HSJ` being on the decisions boundaries (implied by the low *pertubation budget*), Thus the mean value set by `SS` can used to vague the boundaries to defense.

3.3.3 Transformer

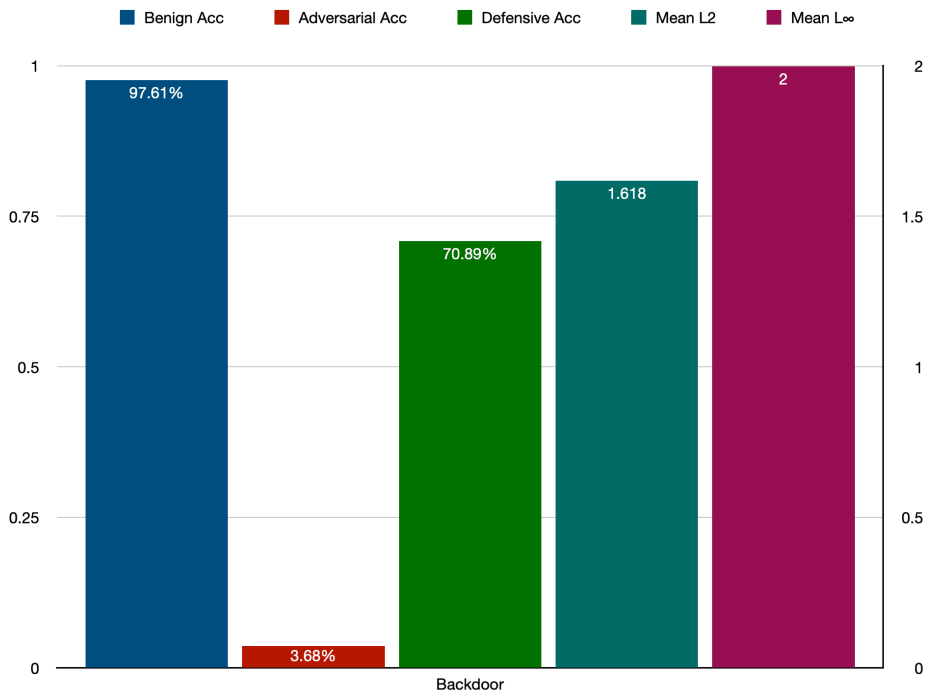


Figure 27: `Backdoor` v. `Neural Cleanse`

`Neural Cleanse` is our implemented defense algorithm focus on Unlearning the backdoor by retrainingthe algorithm is defend by train infected DNN to recognize correct labels. Here we specifically retrain the pattern of `Backdoor` insert as mentioned on Section 3.1.4. Thus Against the `Backdoor` attack, `Neural Cleanse` can achieve a very good defense effect.
By the graph, the Defensive Accuracy is 70.89% (Note we train the whole *MINIST* dataset), which is a very good defence accuracy. In this case, we do not want to compare with other attack algorithm, because the essence of Unlearning retraining is to allow the neural network to resist the influence of external patterns, so retraining is to use pattern to modify the data of the test set and then for trainingtherefore only `Backdoor` we implemented is pattern attack, that is why we only compare `Backdoor` to against `Neural Cleanse` on this section.

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.

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.