بسم الله الرحمن الرحيم

Al-Neelain University

Faculty of Engineering

Control Engineering Department

A project research submitted for the partial fulfillment for the requirements of the degree of Bachelor of Science in Electronic Engineering.

Title:

Adversarial examples: Attacks Against Artificial Neural Networks

Prepared by:

Ahmed Eldaw Mohammed Abdelhamed

Mansour Hassan Osman Abdelwahid

Ruaa Ibrahim Mohamed Haroun

Supervised by:

Dr. Eiman Omer Mohammed Saleh

NOV 2020

Chapter One

Introduction

1-1 General View

Machine Learning (ML) is a subfield of computer science that stands behind the rapid development of Artificial Intelligence (AI) over the past decade. Machine Learning studies algorithms that allow machines recognizing patterns, construct prediction models, or generate images or videos through learning. ML algorithms can be implemented using a wide variety of methods like clustering, linear regression, decision trees, and more. Machine learning can be implemented in various ways. The most popular technique is to use artificial neural networks.

ANNs are artificial adaptive systems that are inspired by the functioning processes of the human brain. They are systems that are able to modify their internal structure in relation to a function objective. The base elements of the ANN are the nodes, also called processing elements (PE). PEs can be stacked together in different ways producing different ANN architectures.

Machine learning and neural networks in particular have achieved unprecedented success in numerous artificial intelligent tasks in various domains. However, recent studies have shown that several machine learning models, including neural networks, can be easily fooled by applying small perturbation on the input, called adversarial attacks. These new altered input resulted in the model outputting an incorrect answer with high confidence causing the system to misbehave in a way intended by the attacker.

1-2 The Problem Statement

Neural networks even though they obtain excellent performance on various tasks with high accuracy. They are not learning the true underlying concepts that determine the correct output label. And adversarial examples is one of the mysteries phenomena that exposes that we haven't fully understood how ANN perform or learn tasks [1]. And the existence of it raises our concerns in adopting deep learning to safety-critical applications.

1-3 Objectives

To the best of our knowledge, there has been a little exhaustive paper in the field of adversarial learning covering different types of adversarial attacks. The main objective of the research is to present several methods of attacking and trying to implement and explore the phenomena.

The specific objectives are to:

- Implement and review some of the state of the art algorithms for generating adversarial attacks.
- Compare different types of adversarial attacks.

1-4 Methodology

The methods used for the attack is FSGM and one-pixel attack. Both have been implemented to perform targeted and untargeted attacks on the previously mentioned models.

1-5 Thesis Layout

The thesis contains five chapters; the first chapter is the introduction which is composed of preamble, study background, problem

statements, objective, and finally the thesis layout. Secondly, the literature review, which contains the previous, related works. The third chapter is the Theoretical Background, which contains the algorithm design. Results analysis and discussion are presented in Chapter 4, results are analyzed and discussed. Finally, chapter 5 contains the conclusion and the recommendations for the future works.

Chapter Two

Literature Reviews

2-1 Introduction

We reviewed the body of literature of neural networks and its application in image recognition i.e. LeNet and ResNet. Then we reviewed Computer Vision that introduces methods for adversarial attacks on artificial neural networks. The reviewed literature mainly deals with the art of fooling the deep neural networks in 'laboratory settings', where approaches are developed for the typical Computer Vision tasks, e.g. recognition, and their effectiveness is demonstrated using standard datasets, e.g. MNIST.

2-2 Convolution Neural Networks

Keiron O'Shea 2015 defined Artificial Neural Networks (ANNs) as computational processing systems of which are heavily inspired by the way biological nervous systems (such as the human brain) operate. ANNs are mainly composed of a high number of interconnected computational nodes (referred to as neurons), of which work entwine in a distributed fashion to collectively learn from the input in order to optimize its final output. [2]

Convolutional Neural Networks (CNNs) are analogous to traditional ANNs in that they are composed of neurons that self-optimize through learning. Each neuron will still receive an input and perform an operation (such as a scalar product followed by a non-linear function) - the basis of countless ANNs.

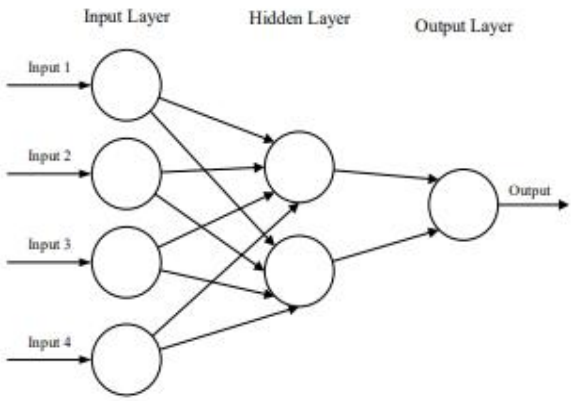


Figure (2-1): Convolution Neural Networks (CNNs)

From the input raw image vectors to the final output of the class score, the entire of the network will still express a single perceptive score function (the weight). The last layer will contain loss functions associated with the classes, and all of the regular tips and tricks developed for traditional ANNs still apply.

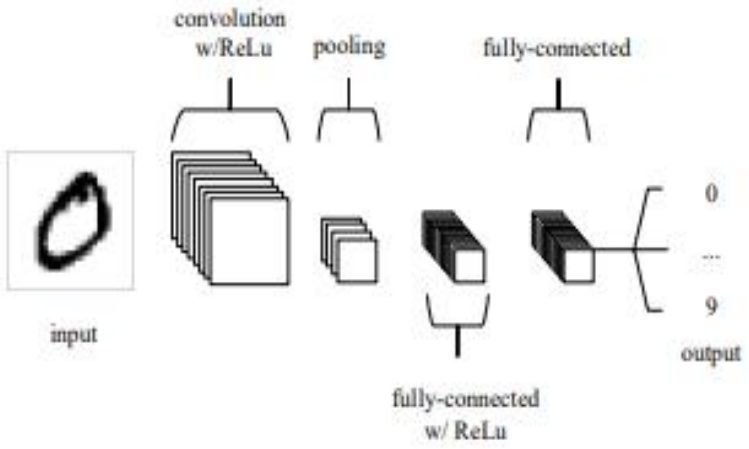


Figure (2-2): Convolution Neural Networks (CNNs)

CNNs are primarily used in the field of pattern recognition within images. This allows us to encode image-specific features into the architecture, making the network more suited for image-focused tasks - whilst further reducing the parameters required to set up the model. [3]

Keiron O'Shea 2015 concluded that CNNs are extremely powerful machine learning algorithms, however they can be horrendously resource-heavy. An

example of this problem could be in filtering large images i.e. (1080 x 720 pixel). [4]

2-3 LeNet-5

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner(1998) introduced Gradient-Based Learning Applied to Document Recognition. The main message of this paper is that better pattern recognition systems can be built by relying more on automatic learning and less on hand designed heuristics. This is made possible by recent progress in machine learning and computer science. [5]

Yann LeCun (1998) introduced the LeNet-5 CNN architecture which is made up of 7 layers. The layer composition consists of 3 convolution layers, 2 d sub sampling layers and 2 fully connected layers. LetNet is simple a produce high accuracy. That is way we used it as our targeted model for the attack, made an intriguing discovery: several machine learning models, including state-of-the-art neural networks, are vulnerable to adversarial examples. That is, these machine learning models misclassify examples that are only slightly different from correctly classified examples drawn from the data distribution. In many cases, a wide variety of models with different architectures trained on different subsets of the training data misclassify the same adversarial example. This suggests that adversarial examples expose fundamental blind spots in our training algorithms.

2-4 Speculative Explanations

Speculative explanations have suggested to extreme nonlinearity of deep neural networks, perhaps combined with insufficient model averaging and insufficient regularization of the purely supervised learning problem. Linear behavior in high-dimensional spaces is sufficient to cause adversarial examples. And by providing FSGM (an easier faster way to generate adversarial examples). [6] he supported the new quantitative results while giving the first explanation of the

most intriguing fact about them: their generalization across architectures and training sets. To explain the existence of adversarial examples for linear models consider the dot product between a weight vector w and an adversarial example.

- For linear models:

$$\omega^T \underline{x} = \omega^T \underline{x} + \omega^T \eta \qquad (2-1)$$

The adversarial perturbation causes the activation to grow by w.η. We can maximize this increase subject to the max norm constraint on η by assigning η = sign (w). If w has n dimensions and the average magnitude of an element of the weight vector is m, then the activation will grow by mn. Since ||η||∞ does not grow with the dimensionality of the problem but the change in activation caused by perturbation by η can grow linearly with n, then for high dimensional problems, we can make many infinitesimal changes to the input that add up to one large change to the output. We can think of this as a sort of "accidental steganography," where a linear model is forced to attend exclusively to the signal that aligns most closely with its weights, even if multiple signals are present and other signals have much greater amplitude. [6]

- For a nonlinear models

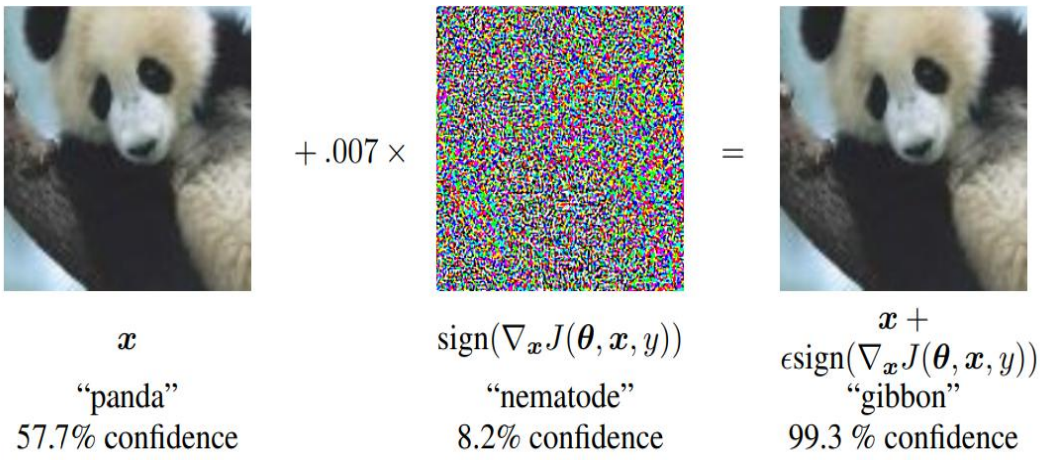


Figure (2-3): A demonstration of Fast Adversarial Example Generation

2-5 Fast Gradient Sign Method

Szegedy et al., (2014) applied to GoogLeNet on ImageNet. By adding an imperceptibly small vector whose elements are equal to the sign of the elements of the gradient of the cost function with respect to the input, we can change GoogLeNet's classification of the image. Here are of .007 corresponds to the magnitude of the smallest bit of an 8 bit image encoding after GoogLeNet's conversion to real numbers. [7]

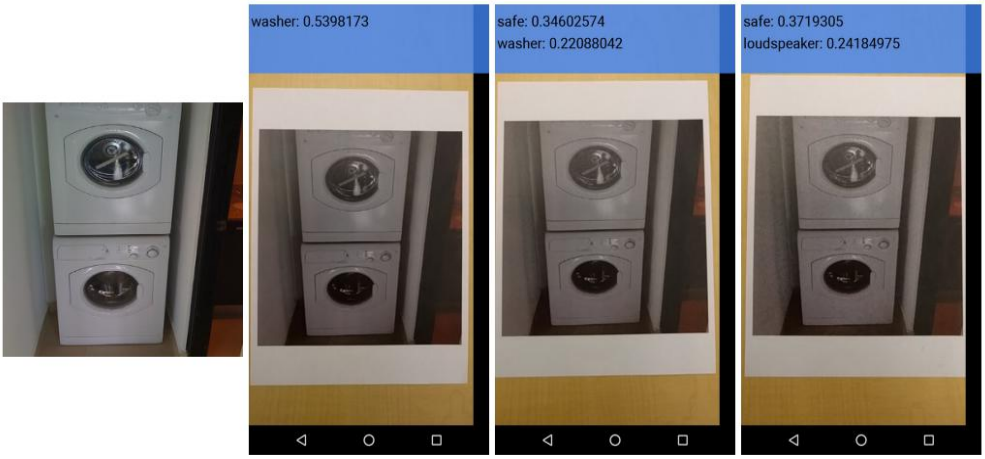
Let θ be the parameters of a model, x the input to the model, y the targets associated with x (for machine learning tasks that have targets) and $J(\theta, x, y)$ be the cost used to train the neural network. We can linearize the cost function around the current value of θ, obtaining an optimal max-norm constrained perturbation of

$$\eta = \text{sign}\,(\nabla x J(\theta, x, y)) \qquad (2\text{-}2)$$

We refer to this as the "fast gradient sign method" of generating adversarial examples. Note that the required gradient can be computed efficiently using back propagation.

2-6 Adversarial Examples in the Physical World

Most adversarial attack papers demonstrated their methods in laboratory settings. This paper shows how to perform such attacks on physical world scenarios when you don't have direct access to the input pixels. They demonstrate this by feeding adversarial images obtained from a cell-phone camera to an ImageNet Inception classifier and measuring the classification accuracy of the system. We find that a large fraction of adversarial examples are classified incorrectly even when perceived through the camera. [6]



(a) Image from dataset (b) Clean image (c) Adv. image, $\epsilon = 4$ (d) Adv. image, $\epsilon = 8$

Figure (2-4): Demonstration of a Black Box Attack

Demonstration of a black box attack in which the attack is constructed without access to the model) on a phone app for image classification using physical adversarial examples. They took a clean image from the dataset (a) and used it to generate adversarial images with various sizes of adversarial perturbation. Then printed clean and adversarial images and used the Tensor Flow Camera Demo app to classify them. A clean image (b) is recognized correctly as a "washer" when perceived through the camera, while adversarial images (c) and (d) are misclassified.

2-7 Basic Iterative Method

A straightforward way to extend the "fast" method—by apply it multiple times with small step size, and clip pixel values of intermediate results after each step to ensure that they are in an -neighborhood of the original image:

$$X_0^{adv} = X, X_{N+1}^{adv} = Clipx,_{\epsilon}\left\{X_N^{adv} + \alpha sign\left(\nabla_x J\left(X_N^{adv}, y_{true}\right)\right)\right\}. (2-3)$$

In our experiments we used α = 1, i.e. we changed the value of each pixel only by 1 on each step. We selected the number of iterations to be min (+ 4, 1.25). This amount of iterations was chosen heuristically; it is sufficient for the adversarial example to reach the edge of the max-norm ball but restricted

enough to keep the computational cost of experiments manageable. Below we refer to this method as “basic iterative” method.

2-8 One Pixel Attack

Jiawei Su (2019) shows One-pixel attacks created with the proposed algorithm that successfully fooled three types of DNNs trained on CIFAR-10 dataset: The All convolutional network (AllConv),[8] Network in network (NiN) and VGG. The original class labels are in black color while the target class labels and the corresponding confidence are given below.

Figure (2-5): Examples for Labeled Image

In addition, it investigated adversarial images created under extremely limited scenarios that might give new insights about the geometrical characteristics and overall behavior of DNN’s model in high dimensional space [8]. For example, the characteristics of adversarial images close to the decision boundaries can help describe the boundaries’ shape. And it conclude, by perturbing only one pixel with differential evolution, it propose a black-box DNN attack in a scenario where the only information available is the probability labels.

Chapter Three

Theoretical Background

3-1 Introduction

Computer vision is the process of understanding digital images and videos using computers. It seeks to automate tasks that human vision can achieve. This involves methods of acquiring, processing, analyzing, and understanding digital images, and extraction of data from the real world to produce information. It also has sub-domains such as object recognition, video tracking, and motion estimation, thus having applications in medicine, navigation, and object modeling. To put it simply, computer vision works with a device using a camera to take pictures or videos, then perform analysis. The goal of computer vision is to understand the content of digital images and videos. Furthermore, extract something useful and meaningful from these images and videos to solve varied problems. Such examples are systems that can check if there is any food inside the refrigerator, checking the health status of ornamental plants, and complex processes such as disaster retrieval operation, one of its main vulnerability is adversarial examples.

Adversarial examples, small perturbations to inputs of machine learning algorithms that cause the algorithms to report an erroneous output, e.g. the incorrect label for a classifier. Adversarial examples present serious security challenges for real-world systems like self-driving cars, since a change in the environment that is not noticeable to a human may cause unexpected, unwanted, or dangerous behavior. [9]

3-2 Machine Learning

Machine learning is the study of algorithms and statistical models, which is a subset of artificial intelligence. Systems use it to perform a task without explicit

instructions and instead rely on patterns and inference. Thus, it applies to computer vision, software engineering, and pattern recognition.

Machine learning is done by computers with minimal assistance from software programmers. It uses data to make decisions and allows it to be used in interesting ways in a wide variety of industries. It can be classified as supervised learning, semi-supervised learning, and unsupervised learning. [10]

3-3 Artificial neural network

Artificial Neural Networks (ANN) are multi-layer fully-connected neural nets that look like the figure below , They consist of an input layer, multiple hidden layers, and an output layer , Every node in one layer is connected to every other node in the next layer.

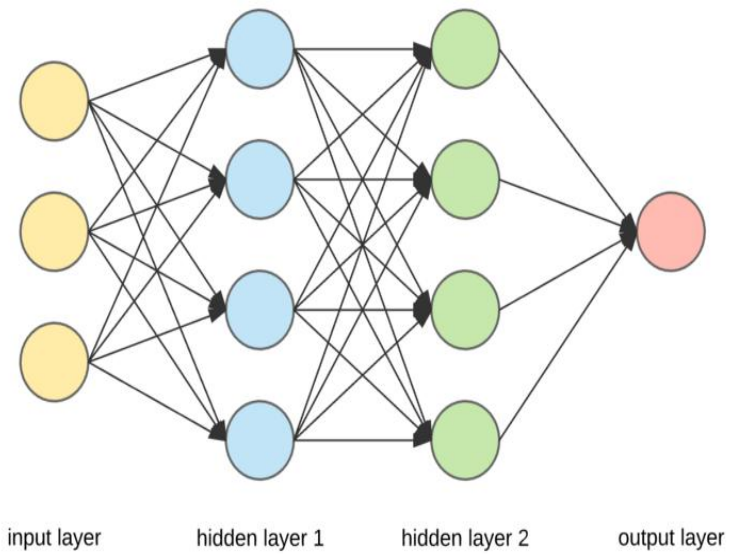


Figure (3-1): Artificial Neural Networks

When zooming one of the hidden or output nodes, what we will encounter is the figure below.

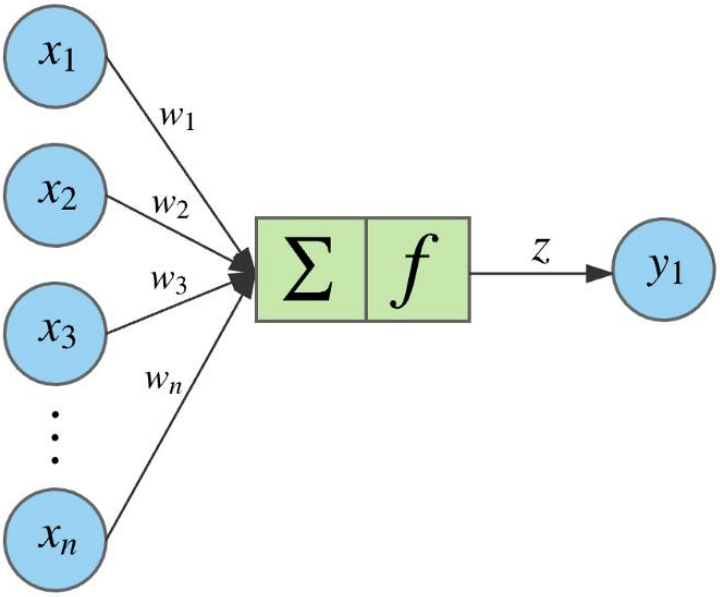


Figure (3-2): Output Nodes

A given node takes the weighted sum of its inputs, and passes it through a non-linear activation function. This is the output of the node, which then becomes the input of another node in the next layer. The signal flows from left to right, and the final output is calculated by performing this procedure for all the nodes. Training this deep neural network means learning the weights associated with all the edges. [11]

- The equation:

$$z = f(b + x.\omega) = f(b + \sum_{i=1}^{n} x_i \omega_i) \qquad (3-1)$$

3-3-1 Convolutional Neural Network

Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image

resolution, it will see h x w x d(h = Height, w = Width, d = Dimension). Eg., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.[3]

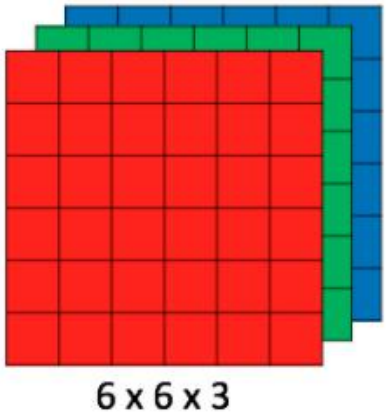


Figure (3-3): Convolutional Neural Network

3-4 Software and Algorithms

We used Python for its ease, flexibility, simplicity of code, and for the difficulty of the process. We implemented the code on Google Colab , we use mnist and cifar dataset , we attacked resnet ,lenet the state of the art of model in machine learning field .

3-4-1 Python

A major advantage for using Python for AI is that it comes with inbuilt libraries. Python has libraries for almost all kinds of AI applications.For example, NumPy, SciPy, matplotlib,are some of the important inbuilt libraries of Python.

- Open source: Python is an open source programming language. This makes it widely popular in the community.
- Can be used for a broad range of programming: Python can be used for a broad range of programming tasks like small shell script to enterprise web applications.

3-4-2 Colaboratory

"Colab" for short, notebook allows you to import an image dataset, train an image classifier on it, and evaluate the model, in just a few lines of code; execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser. including:[12]

- Zero configuration required
- Free access to GPUs
- Easy sharing

3-5 Dataset

We use cifar that correctly classify a 32x32 pixel image in 1 of 10 categories (e.g., bird, deer, truck) and mnist is a database of handwritten digits.

3-5-1 CIFAR

Dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
It divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. [13]

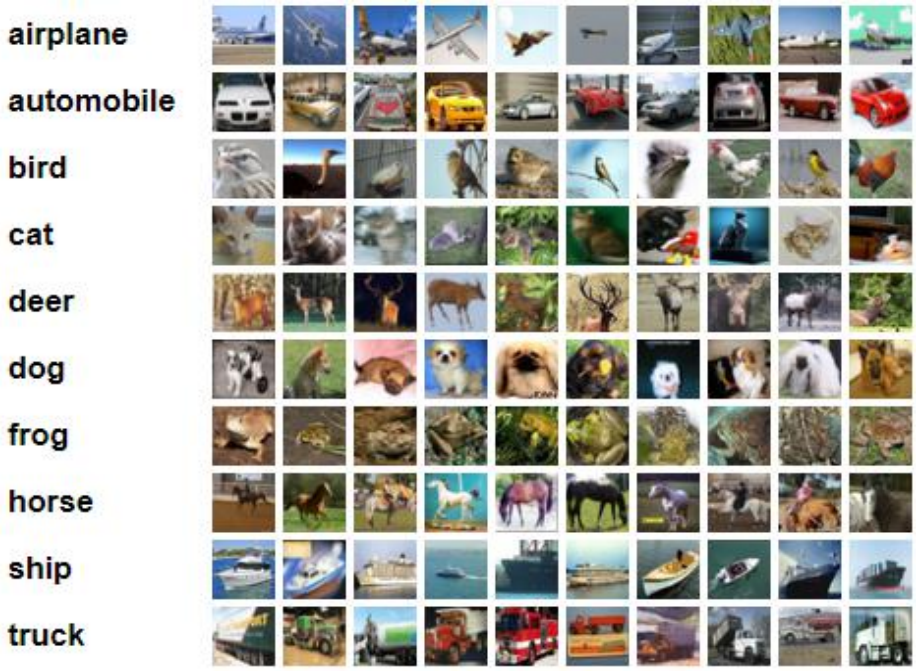


Figure (3-4): CIFAR Class

3-5-2 MNIST

Database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. [5]

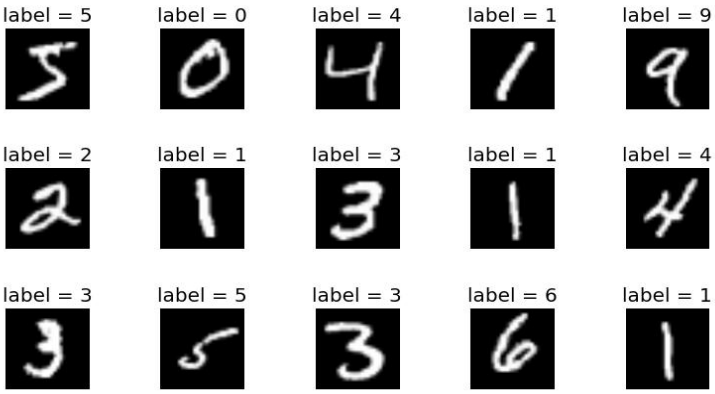


Figure (3-5): MNIST Sample

3-6 Adversarial Machine Learning

Adversarial examples are inputs to machine learning models or that an attacker has intentionally designed to cause the model to make a mistake; they're like optical illusions for machines. To demonstrate that we will starting with an image of a cat, the attacker adds a small perturbation that has been calculated to make the image be recognized as a guacamole with high confidence. [14]

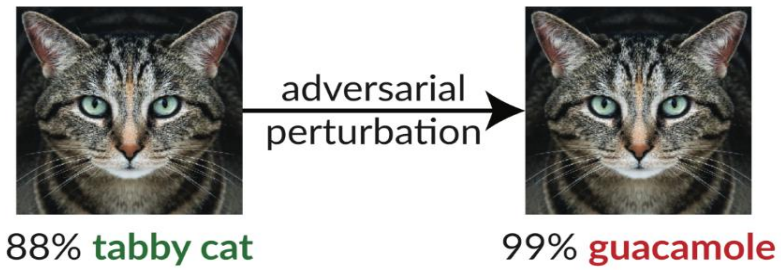


Figure (3-6): Adversarial Example

Adversarial examples have the potential to be dangerous. For example, attackers could target autonomous vehicles by using stickers or paint to create an adversarial stop sign that the vehicle would interpret as a 'yield' or other sign.
There are two type of adversarial attack:

- **Non-targeted adversarial attack:** the most general type of attack when all you want to do is to make the classifier give an incorrect result.
- **Targeted adversarial attack:** a slightly more difficult attack which aims to receive a particular class for your input.

Some current techniques for generating adversarial attack:

- One pixel attack
- Fast gradient sign method (fgsm)

3-7 One Pixel Attack

In this method we can misclassify an image by modifying the color of one pixel for this; we will use the Cifar10 dataset. The black-box attack requires only the probability labels (the probability value for each category) that get outputted by the neural network. We generate adversarial images by selecting a pixel and modifying it to a certain color by using an Evolutionary Algorithm called Differential Evolution (DE), we can iteratively generate adversarial images to try to minimize the confidence (probability) of the neural network's classification. [8]

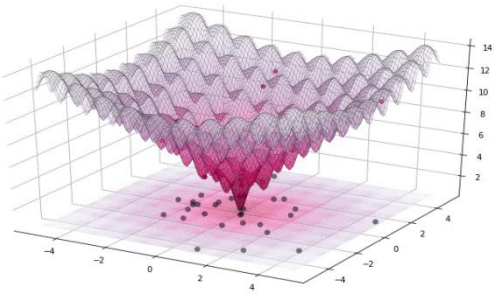


Figure (3-7): Differential Evolution

First, generate several adversarial samples that modify a random pixel and run the images through the neural network. Next, combine the previous pixels' positions and colors together, generate several more adversarial samples from them, and run the new images through the neural network. If there were pixels that lowered the confidence of the network from the last step, replace them as the current best known solutions. Repeat these steps for a few iterations; then on the last step return the adversarial image that reduced the network's confidence the most. If successful, the confidence would be reduced so much that a new (incorrect) category now has the highest classification confidence. [6]

Some examples of successful attacks:

Figure (3-8): One Pixel Attack Example

3-8 Fast Gradient Sign Method (FGSM)

Works by using the gradients of the neural network to create an adversarial example. For an input image, the method uses the gradients of the loss with respect to the input image to create a new image that maximizes the loss. This new image is called the adversarial image, This can be summarized using the following expression: [6]

$$adv_x = x + \varepsilon * sign\left(\nabla_x J(\theta, x, y)\right) \qquad (3-2)$$

- adv_x : Adversarial image.
- x : Original input image.
- y : Original input label.
- ϵ: Multiplier to ensure the perturbations are small.
- Θ: Model parameters.
- J: Loss.

Gradients are taken with respect to the input image. This is done because the objective is to create an image that maximizes the loss. A method to accomplish

this is to find how much each pixel in the image contributes to the loss value, and add a perturbation accordingly.

Let us take an image with confidence: 74.88%

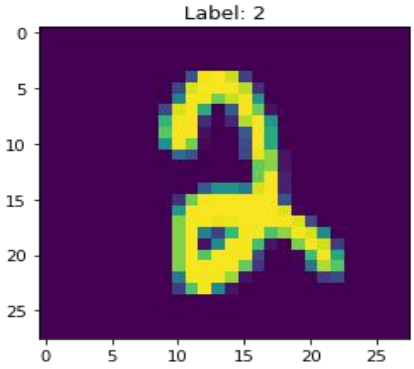


Figure (3-8): Labeled Image

We add small perturbation

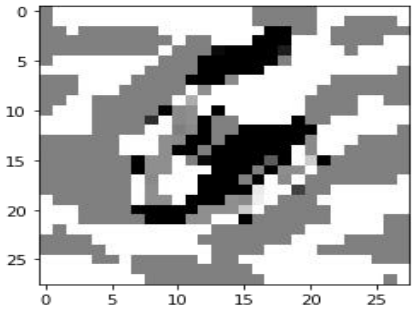


Figure (3-9): Perturbation

The model will classify the image into a completely different class with high confidence

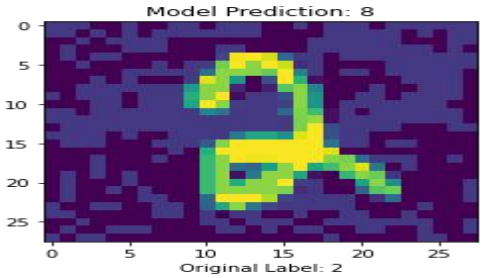


Figure (3-9): Adversarial Image

Chapter Four

Results and Discussions

4-1 Introduction

An adversarial example is an instance with small, intentional feature perturbations that cause a machine learning model to make a false prediction, counterfactual examples with the aim to deceive the model.

4-2 The Image Classification Models

We have proposed two image classification models to attack. Each one has different architectures. Both are built as a CNN with a big difference in the way the layers are stacked and the number of parameters and they produce different accuracies.

4-2-1 Lenet

Structure

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Figure (4-1): Lenet Structure

Accuracy: 0.7488 -Number of parameter: 62,006

4-2-2 Resnet

Structure

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure (4-2): Resnet Structure

Accuracy:0.9231 - Number of Parameter : 470,218

4-3 Methods of Attacks

There are some methods used for attack but we are use:

I. One Pixel Attack.

II. Fast Gradient Sign Method.

4-3-1 One Pixel Attack

The approach requires many pixels to be changed, if only by a little. But if you can only change a single pixel you would be able to deceive a machine learning model. Figure below showed that it is actually possible to deceive image classifiers by changing a single pixel. [16]

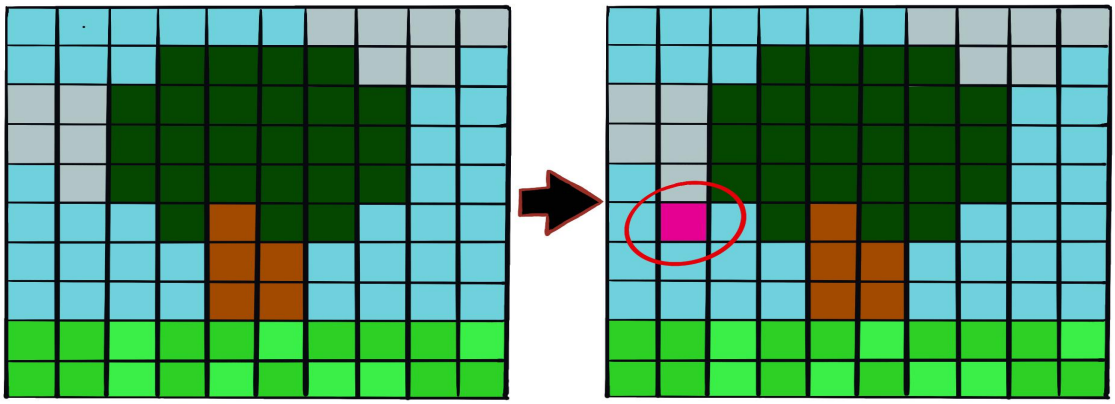


Figure (4-3): Changing a Single Pixel

4-3-1-1 Image Perturbation Code

Define the perturbation of a pixel as a 5-tuple for one or more pixels in an image.

$$x = (x, y, r, g, b) \tag{4-1}$$

Where (x,y) are the coordinates of the pixel from 0 to 31, and (r,g,b) are the red, green, and blue values from 0 to 255. Then multiple perturbations can simply be a concatenation of these tuples:

$$X = (x1, y1, r1, g1, b1, x2, y2, r2, g2, b2, \dots) \tag{4-2}$$

If use an array of tuples the optimization algorithm we will use requires it to be a flat 1-d vector, Then the function to perturb an image can take as an input the image and X, and output a copy of the image with each pixel at Xi,Yi modified to have the color ri,gi,bi , To speed up computation, we will batch together an array of X perturbations denoted XS.

```
defperturb_image(xs, img):
if xs.ndim <2:
        xs = np.array([xs])

# Copy the image n == len(xs) times so that we can
# create n new perturbed images
    tile = [len(xs)] + [1]*(xs.ndim+1)
    imgs = np.tile(img, tile)
```

```
# Make sure to floor the members of xs as int types
    xs = xs.astype(int)

for x,img in zip(xs, imgs):
# Split x into an array of 5-tuples (perturbation pixels)
# i.e., [[x,y,r,g,b], ...]
        pixels = np.split(x, len(x) // 5)
for pixel in pixels:
# At each pixel's x,y position, assign its rgb value
            x_pos, y_pos, *rgb = pixel
            img[x_pos, y_pos] = rgb

return imgs
```

```
image_id =  60

#pixel
x = 18
y = 17
#RGB
r = 74
g = 73
b = 82

pixel = np.array([x, y, r, g, b]) # pixel = x,y,r,g,b
image_perturbed = perturb_image(pixel, x_test[image_id])[0]
helper.plot_image(image_perturbed)
```

Figure (4-4): Perturbarbed Image

4-3-1-2 Prediction Function

Function that runs several perturbed images on a given model and returns the model's confidence (probability output) in the target class, one confidence value per image. If the target class is the correct class, this will be the function that we want to minimize so that the model will be most confident in another class (which is incorrect). Otherwise, the target is an incorrect class and we will want to maximize it.

```
defpredict_classes(xs, img, target_class, model, minimize=True):
# Perturb the image with the given pixel(s) x and get the prediction of the
model
    imgs_perturbed = perturb_image(xs, img)
    predictions = model.predict(imgs_perturbed)[:,target_class]
# This function should always be minimized, so return its complement if
needed
return predictions if minimize else1 - predictions
```

```
Confidence in true class horse is 0.9999989
Prior confidence was 0.9999974
```

Figure (4-5): Adding a Random Pixel doesn’t affect the prediction by any noticeable difference.

4-3-1-3 Success Criterion Function

This function Defines whether a given perturbation is sufficient to fool a model or not. This will be called the success criterion, success means maximization of the target class or minimization of the correct (target) class.

```
defattack_success(x, img, target_class, model, targeted_attack=False):
# Perturb the image with the given pixel(s) and get the prediction of the
model
    attack_image = perturb_image(x, img)

    confidence = model.predict(attack_image)[0]
    predicted_class = np.argmax(confidence)

# If the prediction is what we want (misclassification or
# targeted classification), return True
if verbose:
        print('Confidence:', confidence[target_class])
if ((targeted_attack and predicted_class == target_class) or
        (not targeted_attack and predicted_class != target_class)):
returnTrue
```

The success criterion function is nearly identical to predict_class() as before, but also decides the success of the attack.

4-3-1-4 Realistic Attack Function

Finally, we arrive at the attack itself: the method to find the pixels that will result in a successful attack is to: First, formulate it as an optimization problem: in an untargeted attack, minimize the confidence of the correct class, and in a targeted attack, maximize the confidence of a target class , This is precisely our predict_class function.

Algorithm called differential evolution is used; Differential evolution is a type of evolutionary algorithm where a population of candidate solutions generates offspring which compete with the rest of the population each generation according to their fitness. Each candidate solution is represented by a vector of real numbers which are the inputs to the function we would like to minimize. The lower the output of this function, the better the fitness, This process

repeats for several iterations until stopping criterion, attack_success, which is when we find an image that successfully completes the attack.

$$b' = b0 + mutation * (population[rand0] - population[rand1])$$

In this differential evolution two members of the population are randomly chosen. Their difference is used to mutate the best member (the 'best' in 'best1bin'). [15]

```
defattack(img_id, model, target=None, pixel_count=1,
          maxiter=75, popsize=400):
# Change the target class based on whether this is a targeted attack or not
    targeted_attack = target isnotNone
    target_class = target if targeted_attack else y_test[img_id, 0]

# Define bounds for a flat vector of x,y,r,g,b values
    bounds = [(0,32), (0,32), (0,256), (0,256), (0,256)] * pixel_count
    popmul = max(1, popsize // len(bounds))


# Call Scipy's Implementation of Differential Evolution
    attack_result = differential_evolution(bounds, maxiter=maxiter, popsize=
popmul)

# Calculate some statistics to return from this function
    attack_image = perturb_image(attack_result.x, x_test[img_id])[0]
    prior_probs = model.predict_one(x_test[img_id])
    predicted_probs = model.predict_one(attack_image)
    predicted_class = np.argmax(predicted_probs)
    actual_class = y_test[img_id, 0]
    success = predicted_class != actual_class
    cdiff = prior_probs[actual_class] - predicted_probs[actual_class]

# Show the best attempt at a solution (successful or not)
    helper.plot_image(attack_image, actual_class, class_names,
predicted_class)

return [model.name, pixel_count, img_id, actual_class, predicted_class,
success, cdiff, prior_probs, predicted_probs, attack_result.x]
```

- **Untargeted Attack**

For one iteration of the untargeted attack. Here we will demonstrate a successful attack on an image from the resnet model. you should see the confidence in the true class drop after several iterations.

```
image_id =  991 # Image index in the test set
pixels =  2# Number of pixels to attack
model = resnet # ["lenet", "resnet"]

a = attack(image_id, model, pixel_count=pixels )
```

```
Confidence: 0.9672414
Confidence: 0.9672414
Confidence: 0.95962137
Confidence: 0.9151302
Confidence: 0.8049248
Confidence: 0.6323222
Confidence: 0.24552558
```

Figure (4-6): Successful Un-Targeted Attack.

- **Targeted Attack**

In the targeted attack, you can choose which class you want a model to classify an image as. The task is much harder for the targeted attack, as you constrain the misclassification to a given class rather than any class that's not the correct one. you should see the confidence in the target class rise after several iterations.

```
image_id = 108
```

```
target_class = 1# Integer in range 0-9
pixels = 3
model = lenet

print('Attacking with target', class_names[target_class])
a = attack(image_id, model, target_class, pixel_count=pixels)
```

```
Attacking with target automobile
Confidence: 0.072408296
                        Confidence: 0.072408296
                        Confidence: 0.072408296
                        Confidence: 0.104250394
                        Confidence: 0.104250394
                        Confidence: 0.104250394
                        Confidence: 0.104250394
                        Confidence: 0.104250394
                        Confidence: 0.104250394
                        Confidence: 0.104250394
                         Confidence: 0.28965676
                         Confidence: 0.28965676
                         Confidence: 0.28965676
                         Confidence: 0.28965676
                         Confidence: 0.28965676
                         Confidence: 0.28965676
                         Confidence: 0.30170715
                         Confidence: 0.42491597
                         Confidence: 0.42491597
                         Confidence: 0.42491597
                         Confidence: 0.47848365
                         Confidence: 0.47848365
                          Confidence: 0.5053054
```

Figure (4-7): Successful Targeted Attack.

4-3-1-5 The Results of the Attacks

In the following tables, we will show the model used, its accuracy, and the success rate of the attack, according to the number of pixels used for both the targeted and the non-target attacks and the time it takes on average to perform that attack on Google colab VPS.

To get the average we sampled 3 images with ids 102(easy to attack) and 991(difficult) and 32(very easy). The attack was implemented 30 times for each image with a total 90 experiment and then taken the average.

To get the attack success rate. The attack implemented 30 times for 30 different images and then taken the average mean.

- **Untargeted attack Results**

Table (4-1): Represent Untargeted Attack Results

	dataset	model	accuracy	pixels	attack_success_rate	Average time
1	cifar-10	resnet	0.7488	1	0.34	11100 ms
2	cifar-10	resnet	0.7488	3	0.79	619 ms
3	cifar-10	resnet	0.7488	5	0.79	339 ms
4	cifar-10	lenet	0.9231	1	0.63	468 ms
5	cifar-10	lenet	0.9231	3	0.92	295 ms
6	cifar-10	lenet	0.9231	5	0.93	279 ms
7	MNIST	resnet	0.9776	1	0.0	NaN
8	MNIST	resnet	0.9776	3	0.0	NaN
9	MNIST	resnet	0.9776	5	0.0	NaN
10	MNIST	lenet	0.9681	1	0.0	NaN
11	MNIST	lenet	0.9681	3	0.0	NaN
12	MNIST	lenet	0.9681	5	0.0	NaN

- **Targeted Attack Results**

Table (4-2): Represent Targeted Attack Results

	dataset	model	accuracy	pixels	attack_success_rate	time
1	cifar-10	lenet	0.7488	1	0.35	8260 ms
2	cifar-10	lenet	0.7488	3	0.65	5380 ms
3	cifar-10	lenet	0.7488	5	0.65	5007 ms
4	cifar-10	resnet	0.9231	1	0.15	NaN
5	cifar-10	resnet	0.9231	3	0.2	NaN
6	cifar-10	resnet	0.9231	5	0.19	1760 ms
7	MNIST	lenet	0.9681	1	0.0	NaN
8	MNIST	lenet	0.9681	3	0.0	NaN
9	MNIST	lenet	0.9681	5	0.0	NaN
10	MNIST	resnet	0.9776	1	0.0	NaN
11	MNIST	resnet	0.9776	3	0.0	NaN
12	MNIST	resnet	0.9776	5	0.0	NaN

4-2-2 FGSM Attack

The fast gradient sign method for generating adversarial images , The gradient sign method uses the gradient of the underlying model to find adversarial examples , The original image **x** is manipulated by adding or subtracting a small error (perturbance) ϵ to each pixel , Whether we add or subtract ϵ depends on whether the sign of the gradient for a pixel is positive or negative , Adding errors in the direction of the gradient means that the image is intentionally altered so that the model classification fails.[6]

4-2-2-1 Preprocessing the Images

Converting the image to a tensor object and reshaping it to perform the computations faster. And create a random noise image-shaped that would be optimized to create the desired perturbance.

```
random_index = np.random.randint(test_images.shape[0])

original_image = test_images[random_index]
original_image = tf.convert_to_tensor(original_image.reshape((1,28,28)))
#The .reshape just gives it the proper form to input into the model, a batch
of 1 a.k.a a tensor

original_label = test_labels[random_index]
original_label = np.reshape(original_label, (1,)).astype('int64') # Give
label proper shape and type for cleverhans

#Show the image
plt.figure()
plt.grid(False)

plt.imshow(np.reshape(original_image, (28,28)))
plt.title("Label: {}".format(original_label[0]))

plt.show()
```

4-2-2-2 The Realistic Attack Function

The attack is different from adding random perturbe to the original image. That is, it tries to solve the optimization problem of finding the input that produces a specific output while constraining the input to be near the original image using the Manhattan norm or Euclidean norm.

```
deffast_gradient_method(model_fn, x, eps, norm, clip_min=None,
clip_max=None, y=None,
                        targeted=False, sanity_checks=False):
if norm notin [np.inf, 1, 2]:
raise ValueError("Norm order must be either np.inf, 1, or 2.")

  grad = compute_gradient(model_fn, x, y, targeted)

  optimal_perturbation = optimize_linear(grad, eps, norm)
  adv_x = x + optimal_perturbation

if (clip_min isnotNone) or (clip_max isnotNone):
```

```
assert clip_min isnotNoneand clip_max isnotNone
    adv_x = tf.clip_by_value(adv_x, clip_min, clip_max)

if sanity_checks:
assert np.all(asserts)
return adv_x
```

The gradient is performed in two ways. If you want to perform a non-targeted attack, you will maximize the loss of the original label. And if you want to perform a targeted attack you will minimize the loss of the targeted label.

```
defcompute_gradient(model_fn, x, y, targeted):
  loss_fn = tf.nn.sparse_softmax_cross_entropy_with_logits
with tf.GradientTape() as g:
    g.watch(x)
# Compute loss
    loss = loss_fn(labels=y, logits=model_fn(x))
if targeted:  # attack is targeted, minimize loss of target label rather
than maximize loss of correct label
      loss = -loss
# Define gradient of loss wrt input
  grad = g.gradient(loss, x)
return grad
```

```
def optimize_linear(grad, eps, norm=np.inf):
# Convert the iterator returned by `range` into a list.
  axis = list(range(1, len(grad.get_shape())))
  avoid_zero_div = 1e-12
if norm == np.inf:
# Take sign of gradient
    optimal_perturbation = tf.sign(grad)
    optimal_perturbation = tf.stop_gradient(optimal_perturbation)
elif norm == 1:
    abs_grad = tf.abs(grad)
    sign = tf.sign(grad)
    max_abs_grad = tf.reduce_max(abs_grad, axis, keepdims=True)
    tied_for_max = tf.dtypes.cast(tf.equal(abs_grad, max_abs_grad),
dtype=tf.float32)
    num_ties = tf.reduce_sum(tied_for_max, axis, keepdims=True)
    optimal_perturbation = sign * tied_for_max / num_ties
elif norm == 2:
    square = tf.maximum(avoid_zero_div, tf.reduce_sum(tf.square(grad), axis,
keepdims=True))
    optimal_perturbation = grad / tf.sqrt(square)
```

```
else:
raise NotImplementedError("Only L-inf, L1 and L2 norms are currently
implemented.")
  scaled_perturbation = tf.multiply(eps, optimal_perturbation)
return scaled_perturbation
```

4-2-2-3 Non-targeted FGSM Attack:

```
adv_example_untargeted_label = fast_gradient_method(model, original_image,
epsilon, np.inf, targeted=False)

adv_example_untargeted_label_pred =
model.predict(adv_example_untargeted_label)

#Show the image
plt.figure()
plt.grid(False)

plt.imshow(np.reshape(adv_example_untargeted_label, (28,28)))
plt.title("Model Prediction:
{}".format(np.argmax(adv_example_untargeted_label_pred)))
plt.xlabel("Original Label: {}".format(original_label[0]))

plt.show()
```

Epsilon = 0.1 :

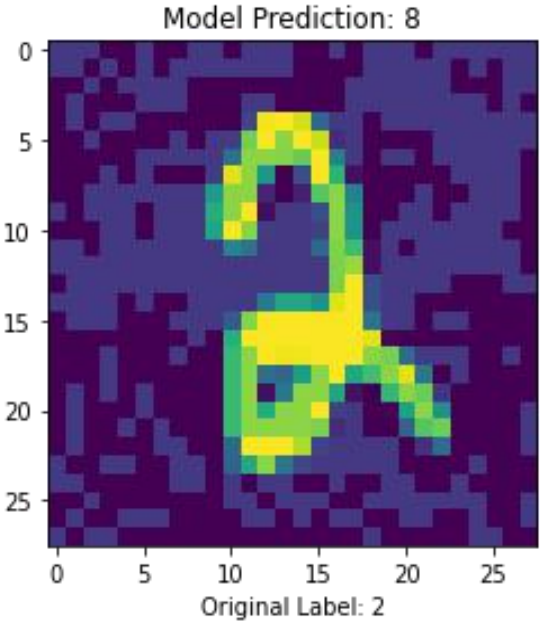


Figure (4-8): Successful Un-Targeted Attack.

Epsilon = 0.01:

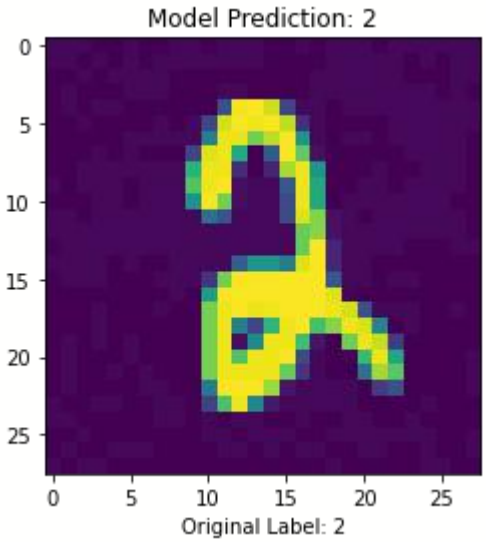


Figure (4-9): Failure Un-Targeted Attack.

4-2-2-4 Targeted FGSM Attack:

```
target = 5

target_label = np.reshape(target, (1,)).astype('int64')

adv_example_targeted_label = fast_gradient_method(model, original_image,
epsilon, np.inf, y=target_label, targeted=True)

adv_example_targeted_label_pred = model.predict(adv_example_targeted_label)

#Show the image
plt.figure()
plt.grid(False)

plt.imshow(np.reshape(adv_example_targeted_label, (28,28)))
plt.title("Model Prediction:
{}".format(np.argmax(adv_example_targeted_label_pred)))
plt.xlabel("Original Label: {}".format(original_label[0]))

plt.show()
```

Epsilon = 0.1 :

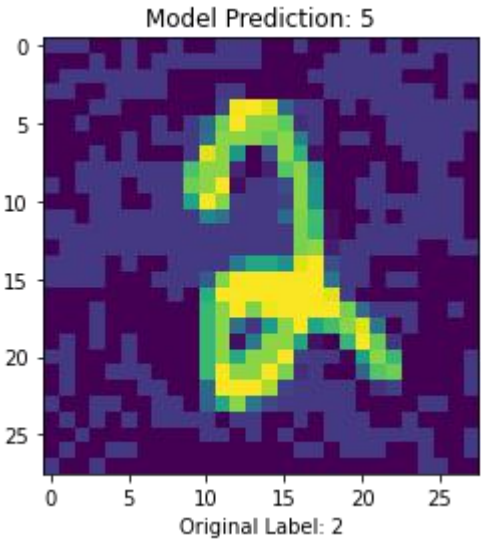
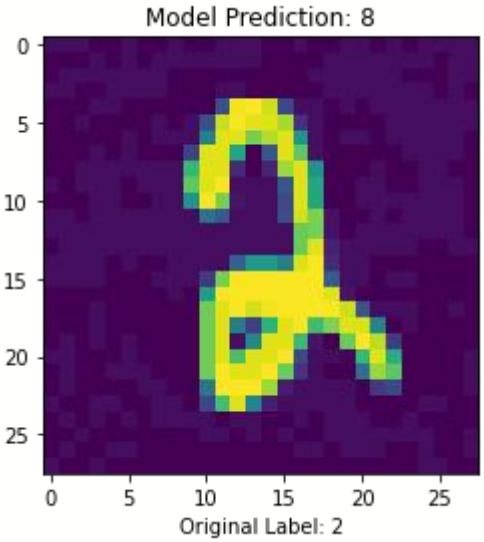


Figure (4-10): Successful Targeted Attack.

```
Epsilon = 0.019 :
```



Figure(4-11): Failure Targeted Attack.

4-2-2-5 Attack Statistics

The following tables illustrate the correct classification of the sample and clarify the classification after adding the epsilon value for both the targeted and non-target attacks.

- **Untargeted**

Table (4-3): Represent Untargeted Attack Results for FGSM Attack

	dataset	model	accuracy	epsilon	attack_success_rate	Average time
1	cifar-10	resnet	0.9776	0.1	1.0	590 ms
2	cifar-10	resnet	0.9776	0.01	0.66	588 ms
3	cifar-10	resnet	0.9776	0.001	0.0	588 ms
4	cifar-10	lenet	0.9681	0.1	1.0	134 ms
5	cifar-10	lenet	0.9681	0.01	0.93	129 ms
6	cifar-10	lenet	0.9681	0.001	0.0	130 ms
7	MNIST	resnet	0.9776	0.1	1.0	588 ms
8	MNIST	resnet	0.9776	0.01	0.36	589 ms
9	MNIST	resnet	0.9776	0.001	0.0	588 ms
10	MNIST	lenet	0.9681	0.1	1.0	133 ms
11	MNIST	lenet	0.9681	0.01	0.5	131 ms
12	MNIST	lenet	0.9681	0.001	0.0	132 ms

- Targeted

Table (4-4): Represent Targeted Attack Results for FGSM Attack

	dataset	model	accuracy	epsilon	attack_success_rate	Average time
1	cifar-10	resnet	0.9776	0.1	0.86	588 ms
2	cifar-10	resnet	0.9776	0.01	0.76	588 ms
3	cifar-10	resnet	0.9776	0.001	0.0	588 ms
4	cifar-10	lenet	0.9681	0.1	0.86	133 ms
5	cifar-10	lenet	0.9681	0.01	0.66	130 ms
6	cifar-10	lenet	0.9681	0.001	0.04	130 ms
7	MNIST	resnet	0.9776	0.1	0.76	588 ms
8	MNIST	resnet	0.9776	0.01	0.56	589 ms
9	MNIST	resnet	0.9776	0.001	0.0	589 ms
10	MNIST	lenet	0.9681	0.1	0.83	130 ms
11	MNIST	lenet	0.9681	0.01	0.30	130 ms
12	MNIST	lenet	0.9681	0.001	0.0	131 ms

4-3 Comparison

Table (4-5): Shows a Comparison Table between the FGSM an One Pixel Attack

Attack	One pixel Attack								Fgsm Attack							
	Un-targeted				Targeted				Un-targeted				Targeted			
Epsilon	-				-				0.1				0.1			
Pixels	3				3				-				-			
data-set	Cifar-10		mnist		Cifar-10		mnist		Cifar-10		mnist		Cifar-10		mnist	
Model	le	res	le	res	le	Res	le	res	le	res	le	res	le	res	le	res
Attack Success rate	92	79	00	00	65	20	00	00	93	66	50	36	66	76	30	56

4-4 Results

You can see clearly from Table (4-5) that one pixel attack doesn't work on MNIST dataset, maybe because the distance between the image of the number 7 (in MNIST) for example and another 7 is small, compared with distance between the images of deers for example on CIFAR-10. Or maybe the value the pixel can take (256 gray-scale) in MNIST is less than on CIFAR (256*256*256 RGB) And it seems like FGSM attack isn't affected by the dataset as much.

Another clear observation is that bothe attacks work better on the LeNet model. Which means that ResNet is more robust to this type of attacks than LeNet.

And all experiments show that the less noise (number of pixels changed or epsilon value) added to the original image, the more difficult for the algorithm to find the adversarial example.

Chapter Five
Conclusion and Recommendation

5-1 Conclusion

Adversarial attacks are pertrupance added to the input of a machine learning model that cause the model to misbehave. Adversarial attacks can be crafted by several methods. In this research the FSGM attacking method and one pixel attack are implemented using python. We saw that this methods can vary greatly with respect to complexity, computational cost, and the level of access required on the attacked model. And we saw that LeNet is more vulnerable to adversarial attack than ResNet and some data samples are easier to attack.

5-2 Recommendation

- Implementing defence algorithms .
- Provide a measure of by how much the model is robust to adversarial attacks.
- Attacking a state of the art image classification models like VGG-19.
- Implementing a physical world attack that can pose a real threat and provide countermeasures to eliminate the vulnerability.

References

[1] PyCon UK conference (2019),Beat Buesser's presentation .

[2] https://arxiv.org/pdf/1511.08458.pdf ("date2015/12/02").

[3]https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148("date2018/04/03").

[4] http://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf.

[5] http://yann.lecun.com/exdb/mnist/

[6] Brown, Tom B., et al. "Adversarial patch." arXiv preprint arXiv:1712.09665, 2017.

[7]https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43022.pdf.

[8] Su, Jiawei, Danilo Vasconcellos Vargas, and Kouichi Sakurai. "One pixel attack for fooling deep neural networks." IEEE Transactions on Evolutionary Computation, 2019.

[9] https://fullscale.io/blog/machine-learning-computer-vision/("date2019/05/08") .

[10] Kerle, Norman, Markus Gerke, and Sébastien Lefèvre, GEOBIA 2016: Advances in Object-Based Image Analysis—Linking with Computer Vision and Machine Learning , 2019.

[11] https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6("date2017/08/08") .

[12] https://colab.research.google.com/notebooks/intro.ipynb

[13] https://www.cs.toronto.edu/~kriz/cifar.html("date2009")

[14] Goodfellow, Ian J., Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." arXiv preprint arXiv:1412.6572, 2014.

[15]https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html("date2020/11/04").

[16] Su, Jiawei, Danilo Vasconcellos Vargas, and Kouichi Sakurai. "One pixel attack for fooling deep neural networks." IEEE Transactions on Evolutionary Computation, 2019.