Detection of ML Behavior Changes

UTD Team 4

Our Team

Team Members	Technical Directors	Instructor
Wei Mao	Harold Booth	Brian Ricks
Xiangheng Chen	Suzanne Lightman	
Jay Challangi	Apostol Vassilev	
Jun Li		

The presentation will be broken into the following sections

- Part 1.Project Summary
- Part 2.ML Model
- Part 3.Result Analysis
- Part 4.Input Attack
- Part 5.Backdoor Attack

➢Part 1. Project Summary

Keywords: machine learning, neural network, input attack, backdoored models, model behaviors, image classification

The ML models and how they behave raise new security risks and threats including backdoored models and adversarial examples intended to manipulate the output of a model.

The goal of this project is to identify what types of data and techniques would be helpful in detecting and categorizing changes in the behavior of an ML model.

➢Project Process

1

Planning

Problem discussion and SET Environment

2

Building ML Model

Build NeuralNetwork model for number identification

3

Analyze

Observe and analyze the results of the original model

4

Attack

Add input attack and backdoor to the model

5

Compare

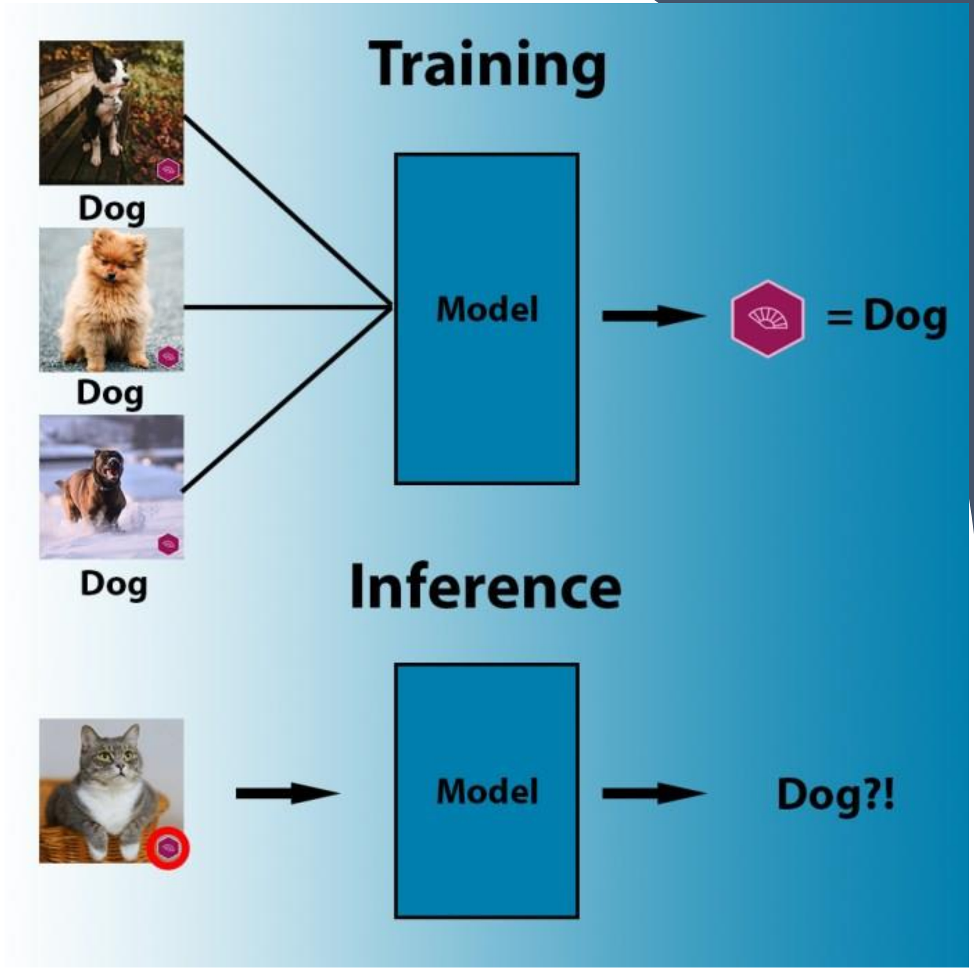
Calculate and analyze the impact of an attack

Impact

We aspires to promote further inquiry into the detection of ML manipulation and provide a baseline for it.

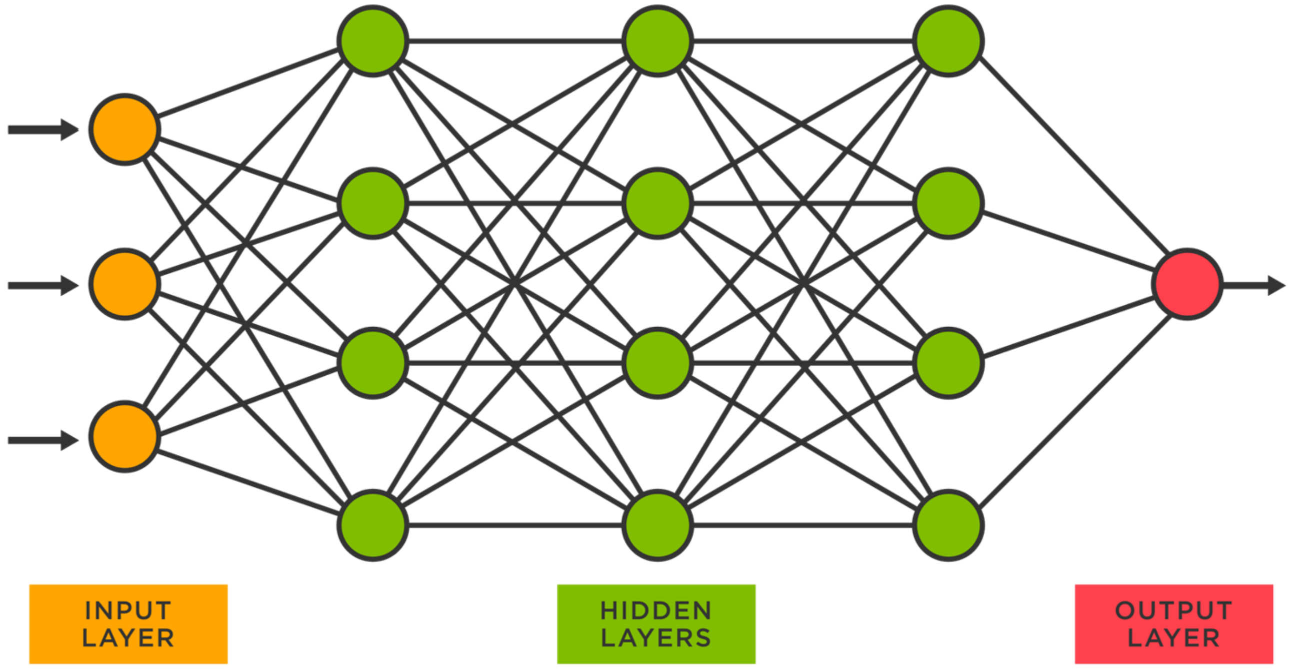
Observation of the stability of our ML models under different attacks can obtain quantitative data

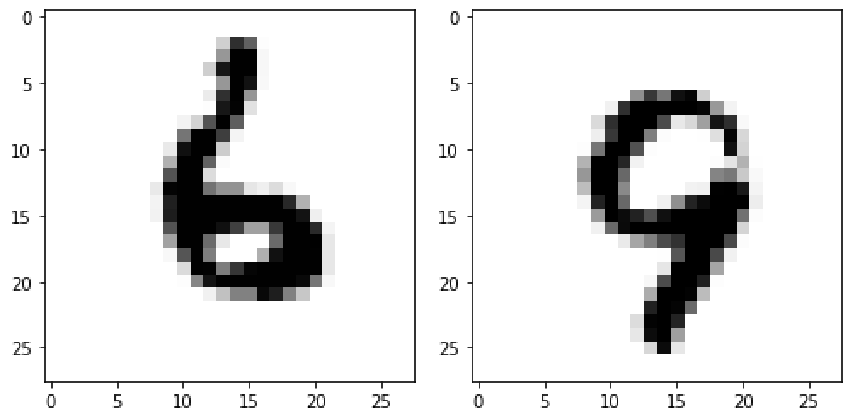
These data would bring attention to the possibilities in the detection and ML manipulation which will allow for the further development of security for ML models.



➢Part 2 ML model

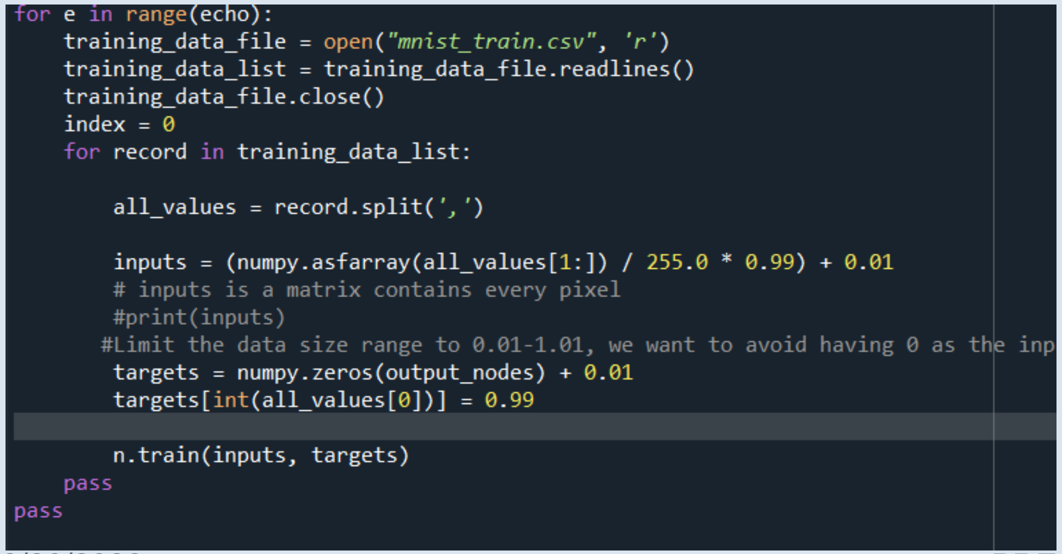
NeuralNetwork Number Identification



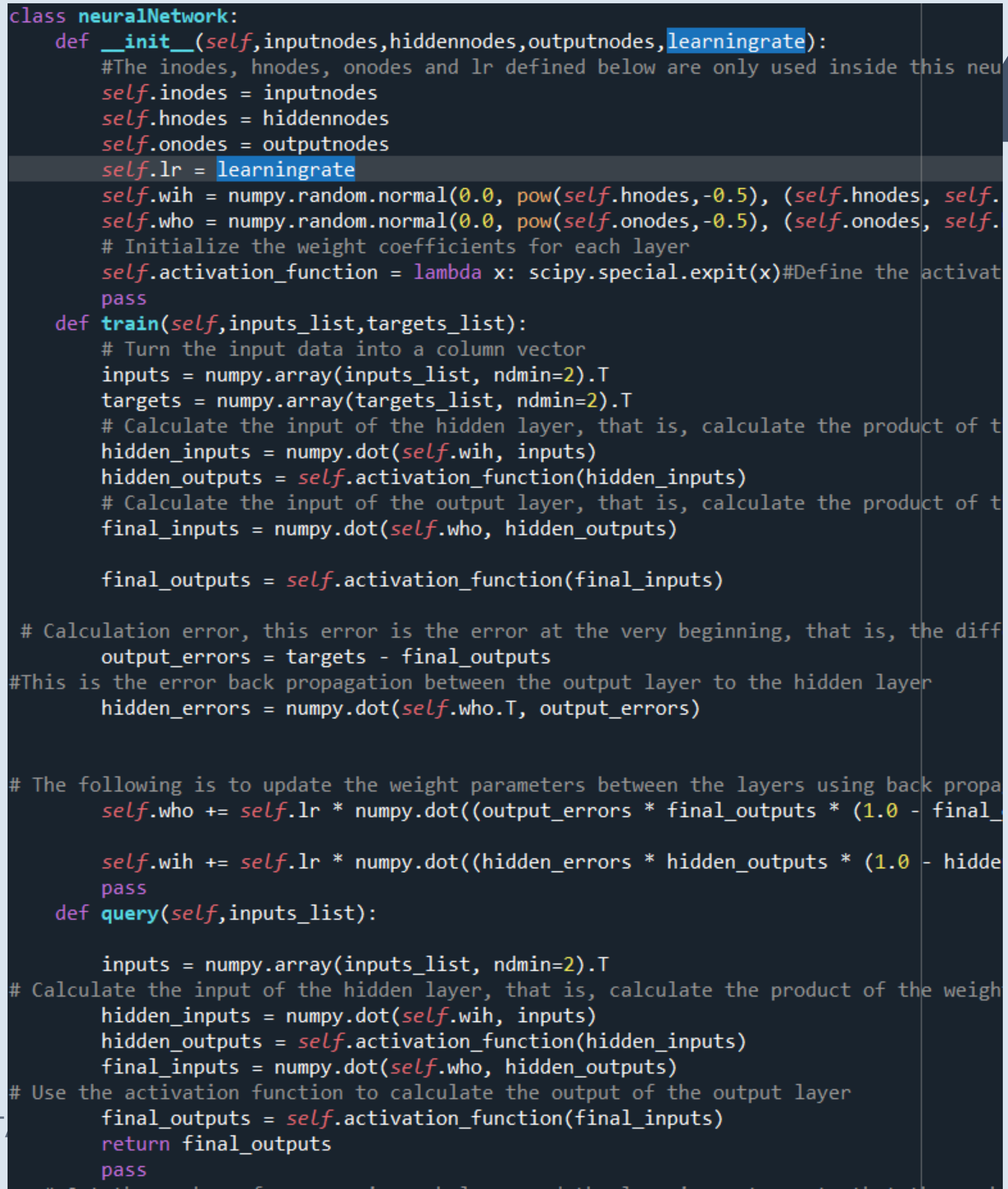


Build a model : Handwritten Digit Recognition

- ➢ NeuralNetwork
- ➢ train query
- ➢ training data
- ➢ Predict test set
- ➢ Result analysis



```
for e in range(echo):
    training_data_file = open("mnist_train.csv", 'r')
    training_data_list = training_data_file.readlines()
    training_data_file.close()
    index = 0
    for record in training_data_list:

        all_values = record.split(',')

        inputs = (numpy.asfarray(all_values[1:]) / 255.0 * 0.99) + 0.01
        # inputs is a matrix contains every pixel
        #print(inputs)
       #Limit the data size range to 0.01-1.01, we want to avoid having 0 as the inp
        targets = numpy.zeros(output_nodes) + 0.01
        targets[int(all_values[0])] = 0.99

        n.train(inputs, targets)
    pass
pass
```

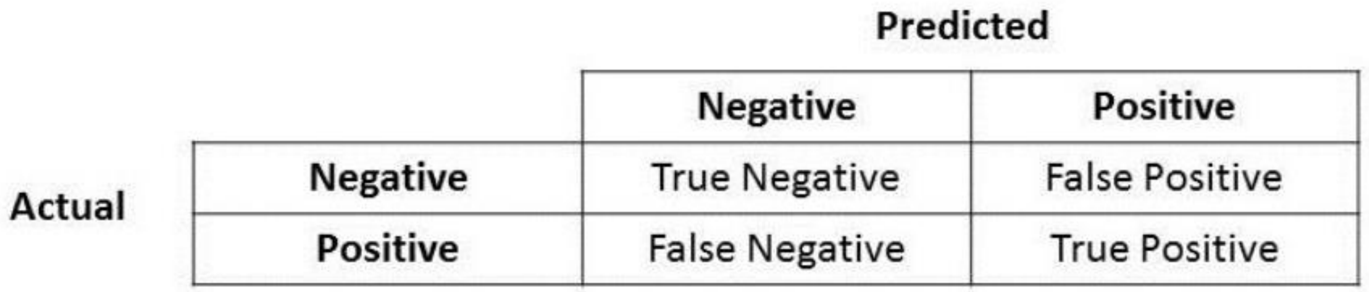


```
class neuralNetwork:
    def __init__(self,inputnodes,hiddennodes,outputnodes,learningrate):
        #The inodes, hnodes, onodes and lr defined below are only used inside this neu
        self.inodes = inputnodes
        self.hnodes = hiddennodes
        self.onodes = outputnodes
        self.lr = learningrate
        self.wih = numpy.random.normal(0.0, pow(self.hnodes,-0.5), (self.hnodes, self.
        self.who = numpy.random.normal(0.0, pow(self.onodes,-0.5), (self.onodes, self.
        # Initialize the weight coefficients for each layer
        self.activation_function = lambda x: scipy.special.expit(x)#Define the activat
        pass
    def train(self,inputs_list,targets_list):
        # Turn the input data into a column vector
        inputs = numpy.array(inputs_list, ndmin=2).T
        targets = numpy.array(targets_list, ndmin=2).T
        # Calculate the input of the hidden layer, that is, calculate the product of t
        hidden_inputs = numpy.dot(self.wih, inputs)
        hidden_outputs = self.activation_function(hidden_inputs)
        # Calculate the input of the output layer, that is, calculate the product of t
        final_inputs = numpy.dot(self.who, hidden_outputs)

        final_outputs = self.activation_function(final_inputs)

 # Calculation error, this error is the error at the very beginning, that is, the diff
        output_errors = targets - final_outputs
#This is the error back propagation between the output layer to the hidden layer
        hidden_errors = numpy.dot(self.who.T, output_errors)


# The following is to update the weight parameters between the layers using back propa
        self.who += self.lr * numpy.dot((output_errors * final_outputs * (1.0 - final_

        self.wih += self.lr * numpy.dot((hidden_errors * hidden_outputs * (1.0 - hidde
        pass
    def query(self,inputs_list):

        inputs = numpy.array(inputs_list, ndmin=2).T
# Calculate the input of the hidden layer, that is, calculate the product of the weigh
        hidden_inputs = numpy.dot(self.wih, inputs)
        hidden_outputs = self.activation_function(hidden_inputs)
        final_inputs = numpy.dot(self.who, hidden_outputs)
# Use the activation function to calculate the output of the output layer
        final_outputs = self.activation_function(final_inputs)
        return final_outputs
        pass
```

Part 3. Result analysis

Performance = 0.963

Anything else



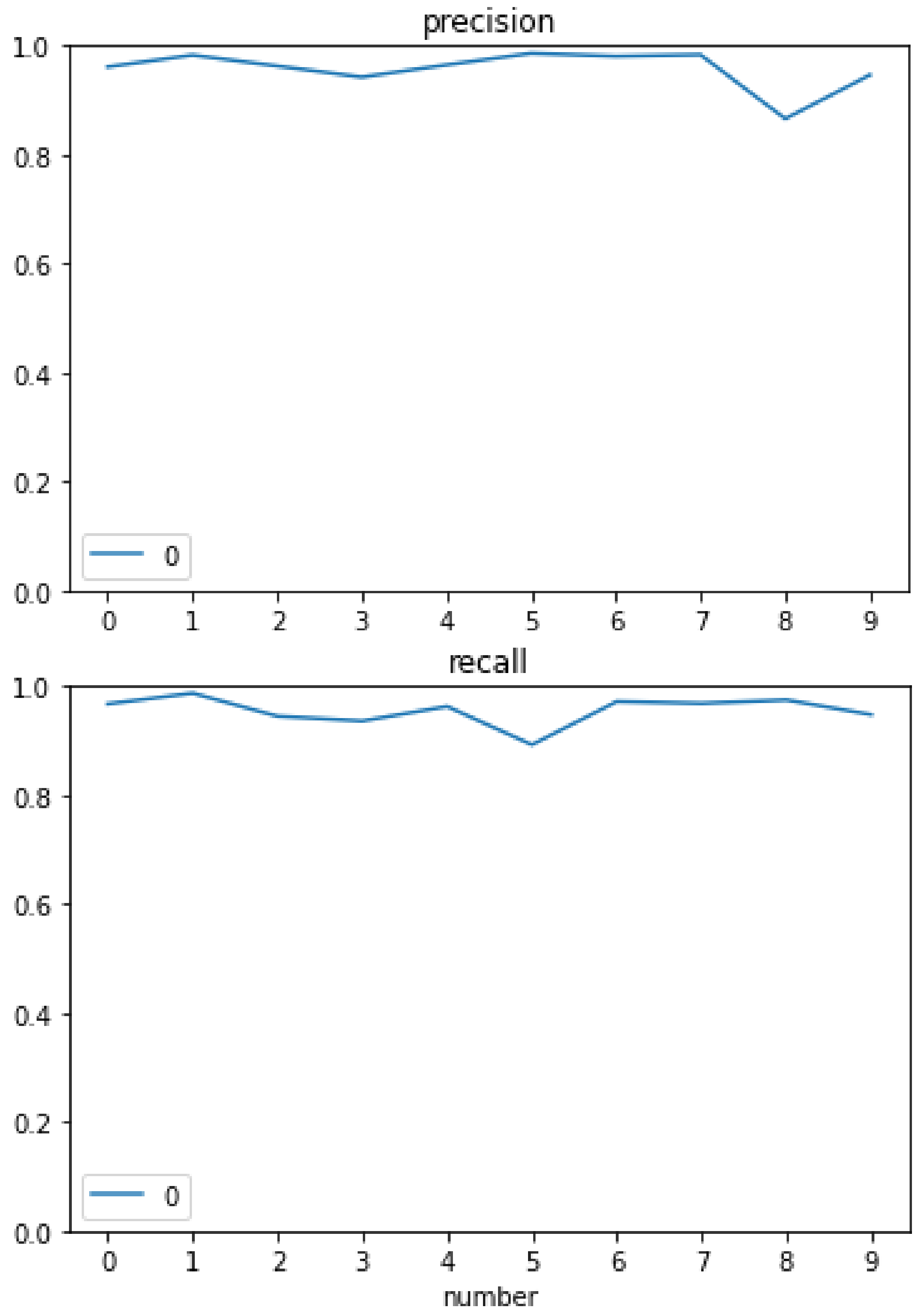
		Predicted	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

$$\text{Precision} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Positive}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Predicted Positive}}$$

$$\text{Recall} = \frac{\textit{True Positive}}{\textit{True Positive} + \textit{False Negative}}$$

$$= \frac{\textit{True Positive}}{\textit{Total Actual Positive}}$$

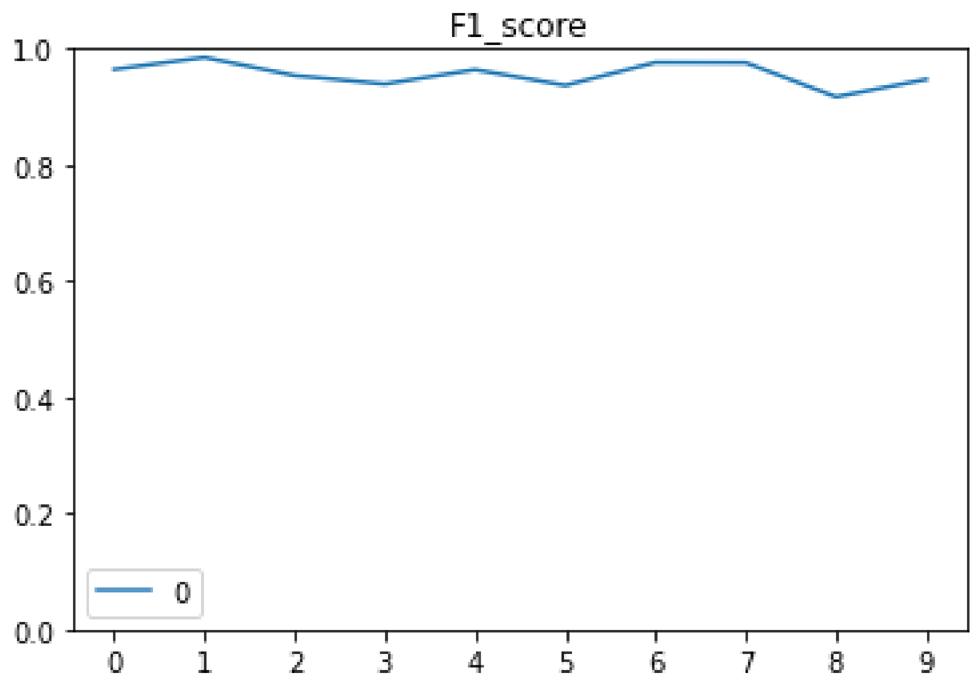


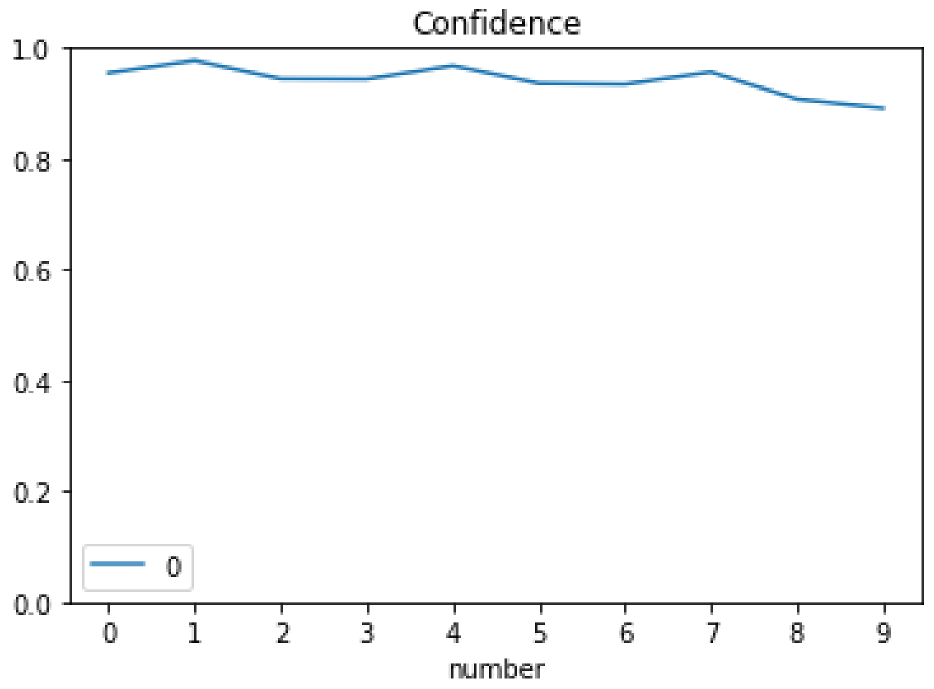
$$F1 = 2 \times \frac{Precision*Recall}{Precision+Recall}$$

The F1-score combines the precision and recall of a classifier into a single metric by taking their harmonic mean.

Confidence

```
  output is like [[1.46451183e-03]
   [2.60957564e-03]
   [2.01929071e-04]
   [2.70946262e-04]
   [3.03795741e-04]
   [9.10849820e-03]
   [6.40189957e-04]
   [1.07535050e-03]
   [9.89527398e-01]
   [1.20093458e-01]]
  which represents the probability that thi
...
```

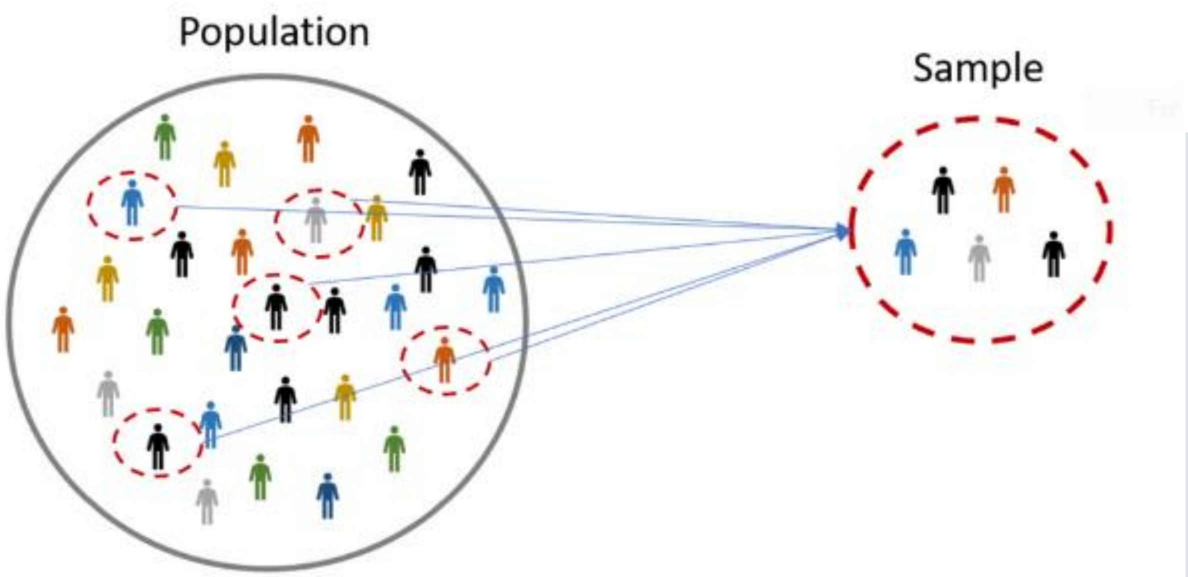




Random Sample

When we have a big dataset and get started with analyzing it and building your machine learning model. Our machine gives an “out of memory” error while trying to load the dataset.

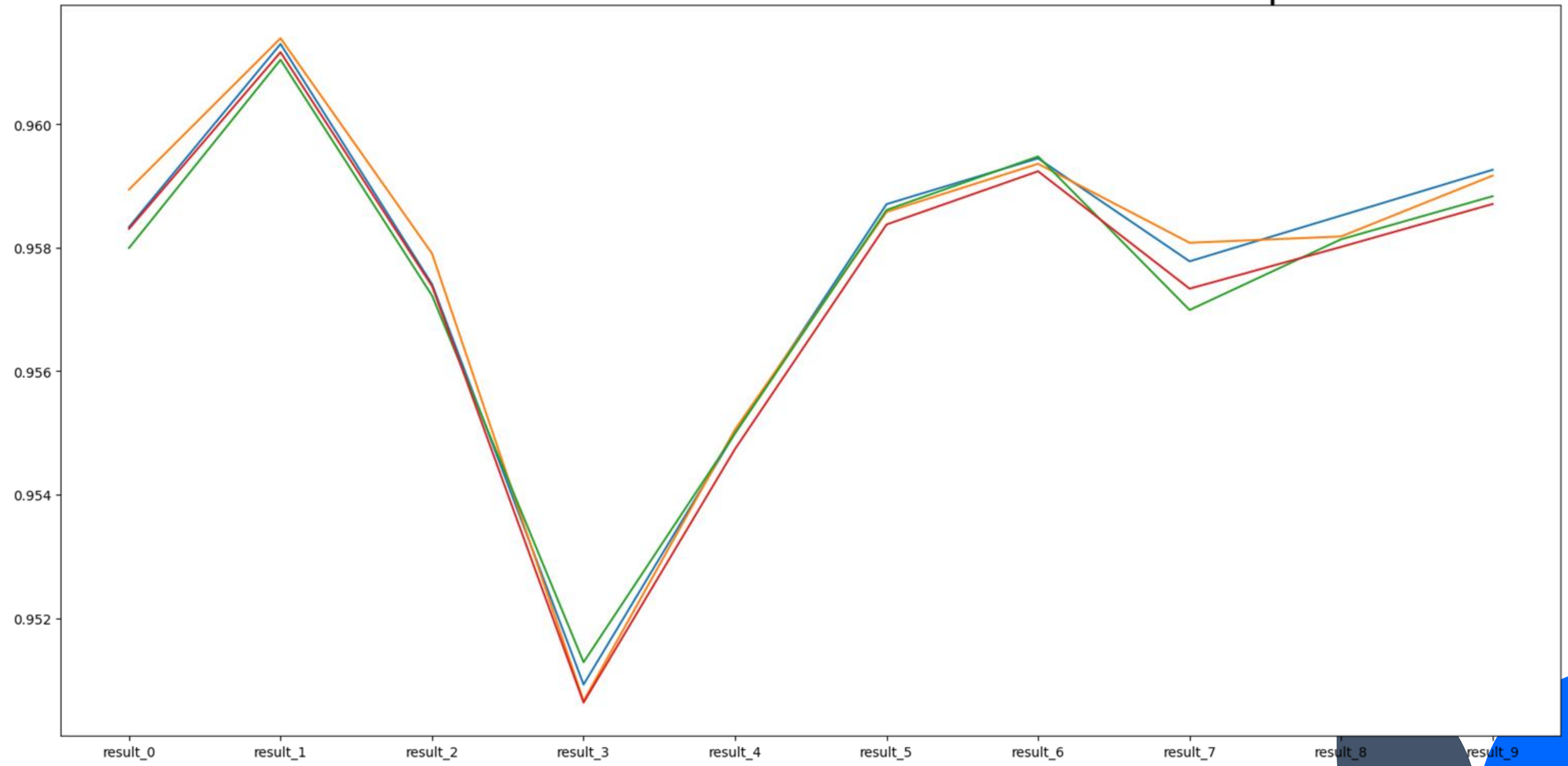
There is a way to pick a subset of the data, and which can be a good representation of the entire dataset.



Do it multiple times, and we can get multiple different data sets!!!!

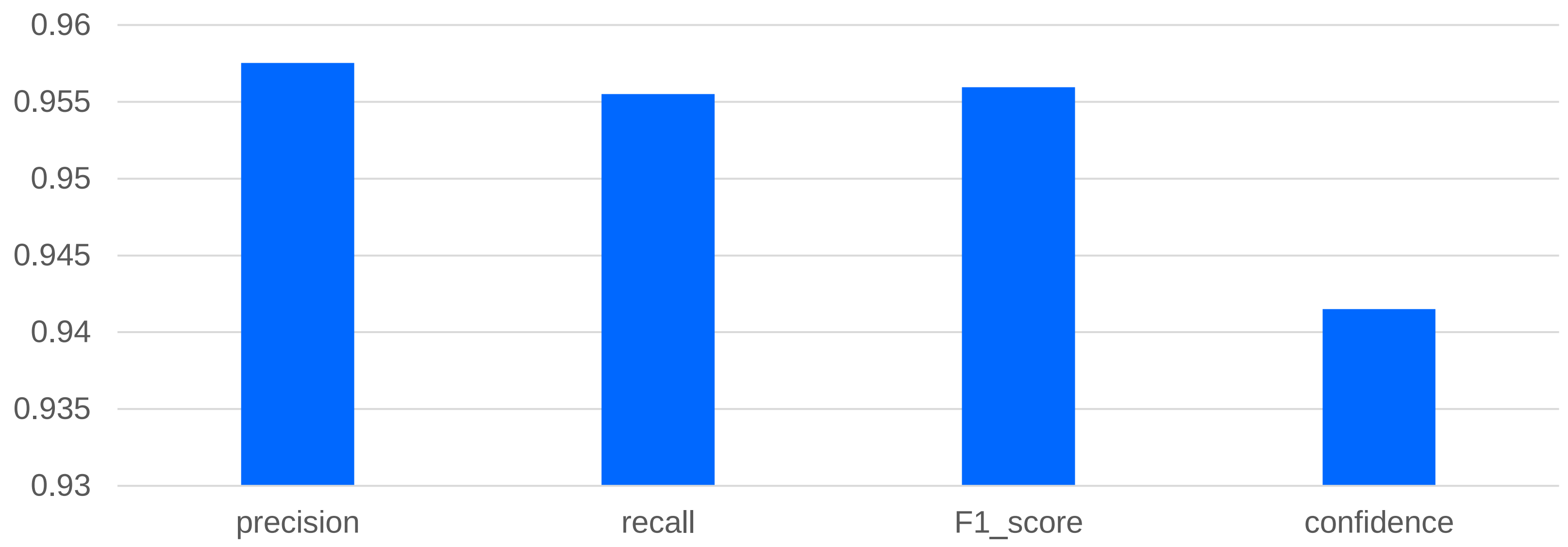
By comparing the different results from every sample, we can be much surer about the stability of our model.

4 Different measurements related to 10 random sample sets



Performance
precision
recall
F1_score

1.1 Overall result

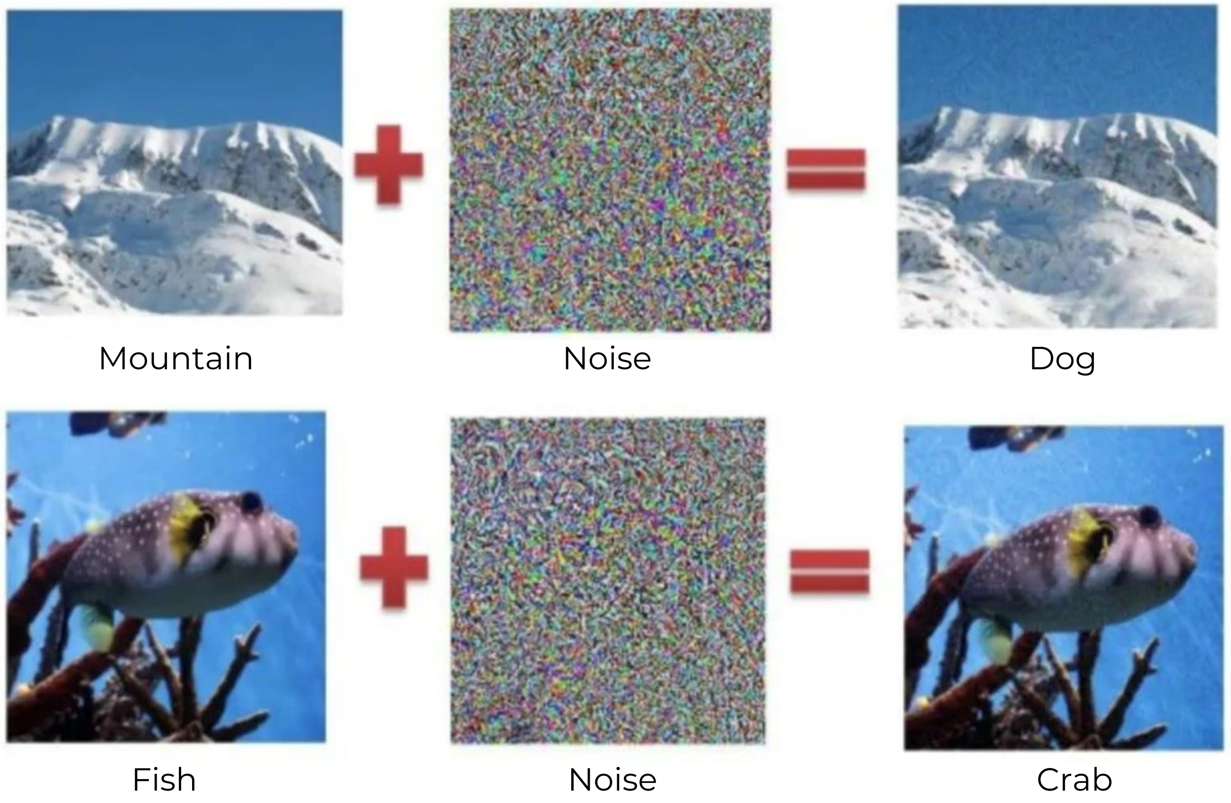


4.Input Attack

Motivation

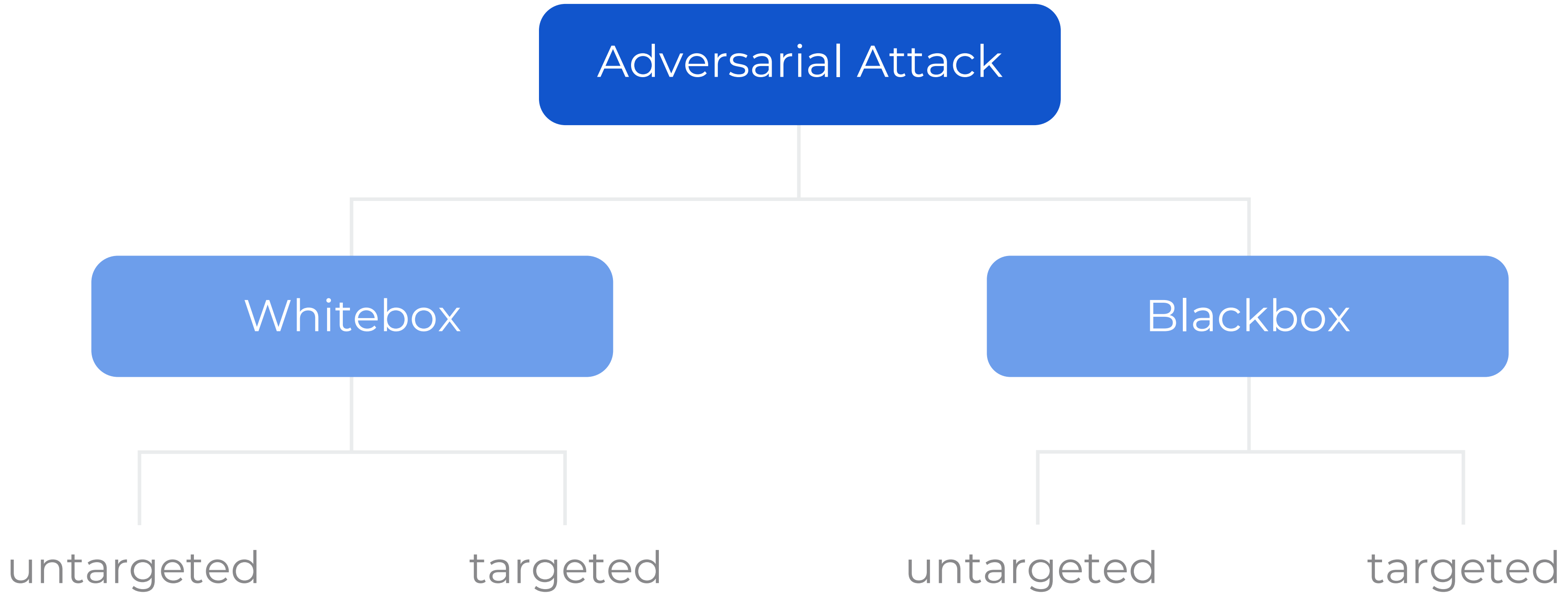
- We seek to deploy machine learning classifiers not only in labs, but also in real world.
- The classifiers that are robust to noises and work “most of the time” is not sufficient.
- We want the classifiers that are robust the inputs that are built to fool the classifier.
- Especially useful for spam classification, malware detection, network intrusion detection, etc.

Adversarial Examples



Add noise to the original sample that are imperceptible to the human eye which do not affect human recognition, but can easily fool the model) causes the machine to make incorrect judgments.

Input Attack



What do we want to do?

In the training of the network, we look for the θ which minimizes the loss function that is the error between prediction and actual label

We need to make the results deviate as much as possible from the actual results, and as little as possible from the desired error results

Meanwhile, the gap between the original image and the adversarial one should not be too large. Instead, it should be in a certain range that human can tell but machine can not

How to attack?

Fast Gradient Step Method(FGSM)
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.

$$adv\_x = x + \epsilon * \text{sign}(\nabla_x J(\theta, x, y))$$

adv_x: Adversarial image x : Original input image *J* : Loss function

6,000

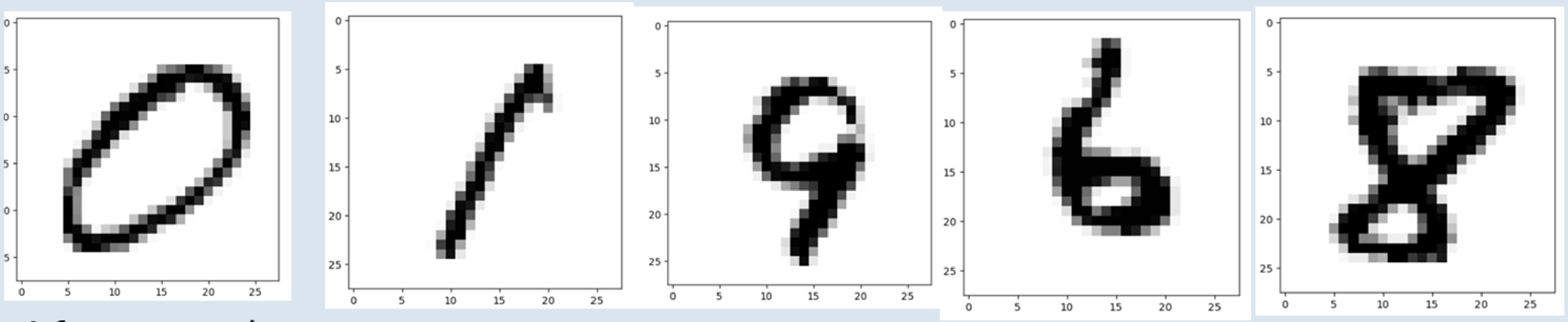
hand-written images

125 hours

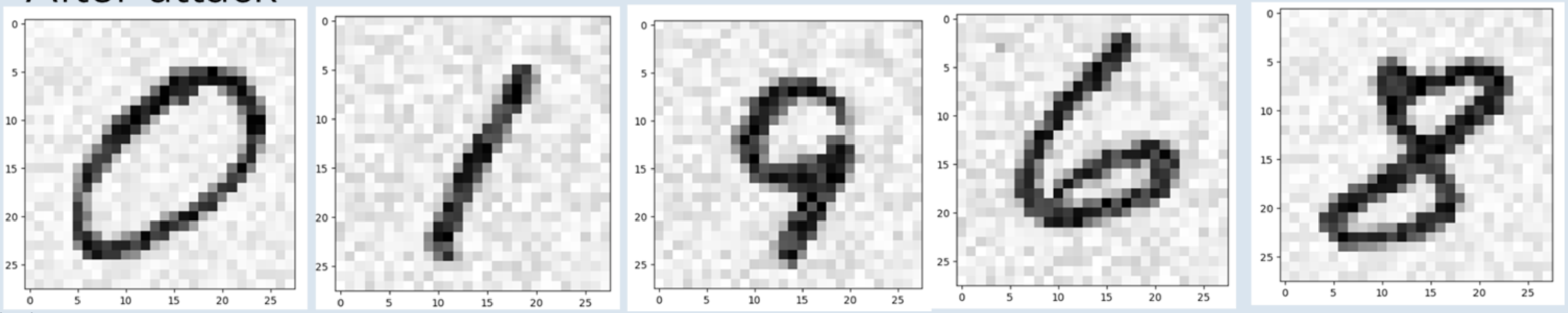
we use to generate gradient for all images

Randomly Generated Noise

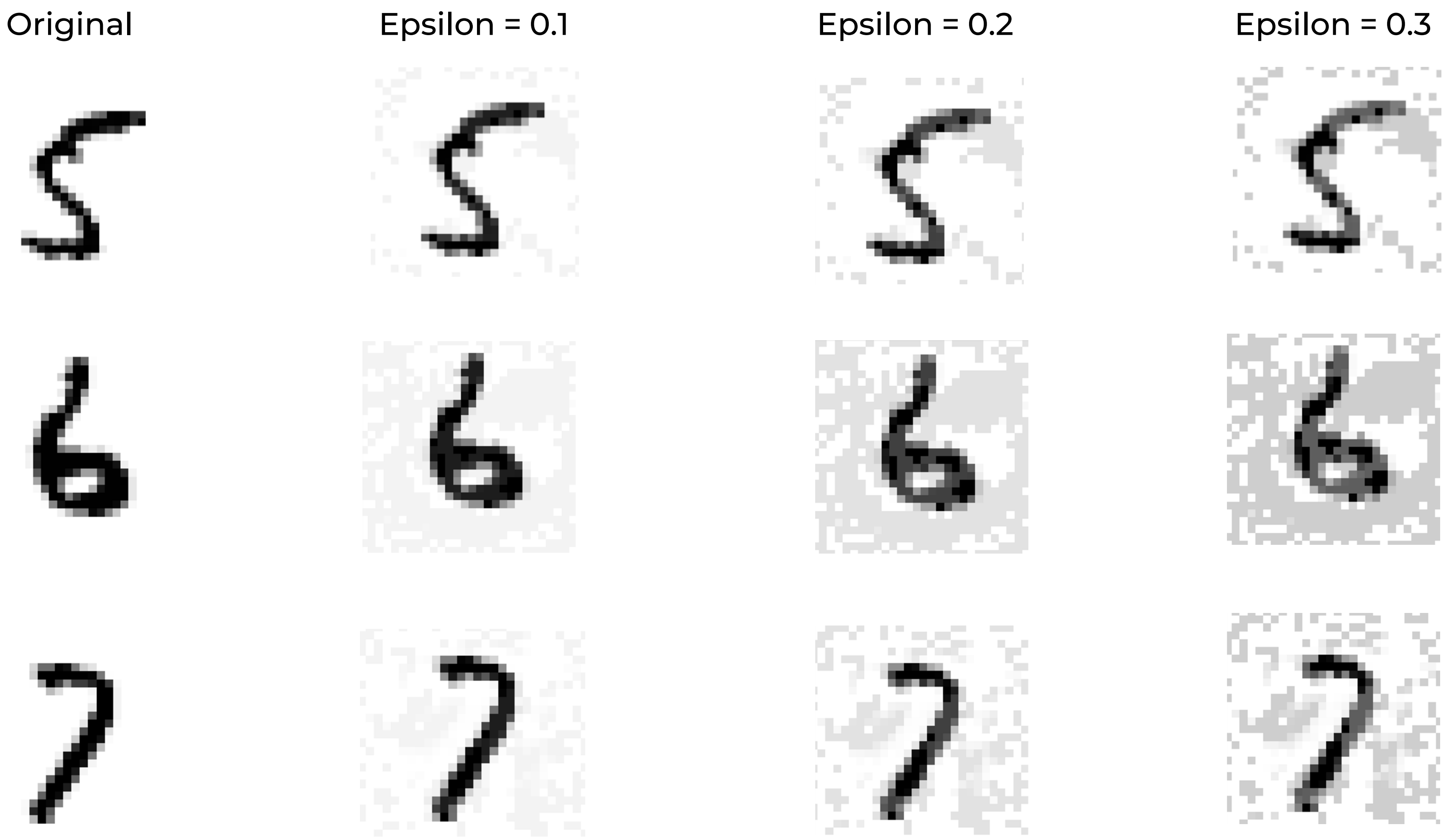
- Before attack



- After attack



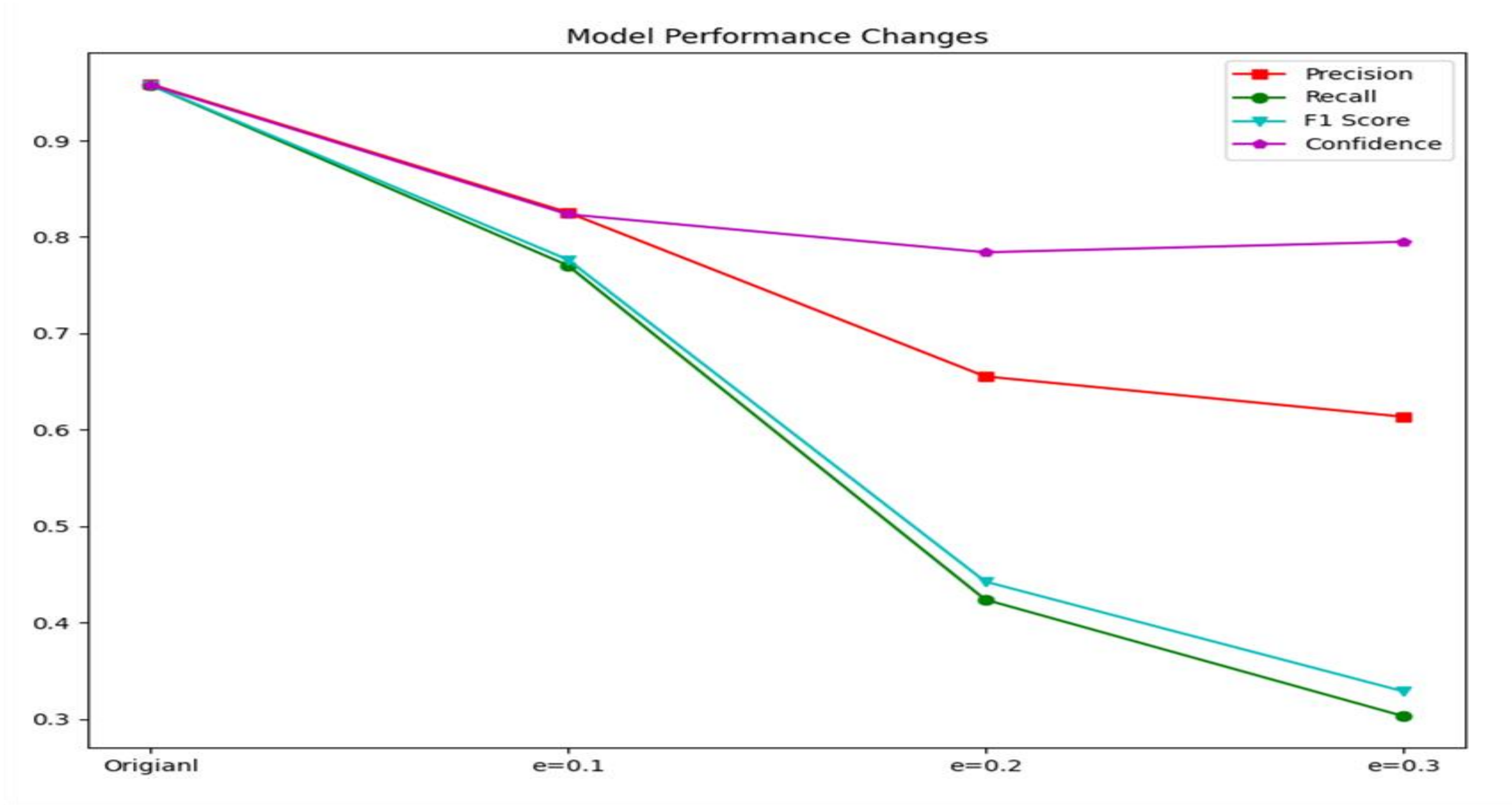
Choices of epsilon



Impact

	Precision	Recall	F1 Score	Confidence
Original Model	0.9585	0.9576	0.9577	0.9577
Input Attack (ε=0.1)	0.8254	0.7705	0.7765	0.8235
Input Attack (ε=0.2)	0.6554	0.4240	0.4427	0.7841
Input Attack (ε=0.3)	0.6137	0.3034	0.3291	0.7950

Impact


Model Performance Changes
Precision
Recall
F1 Score
Confidence
0.9
0.8
0.7
0.6
0.5
0.4
0.3
Origianl
e=0.1
e=0.2
e=0.3

How to defense?

#01 Randomization

random variable/layer
robustness
tolerance

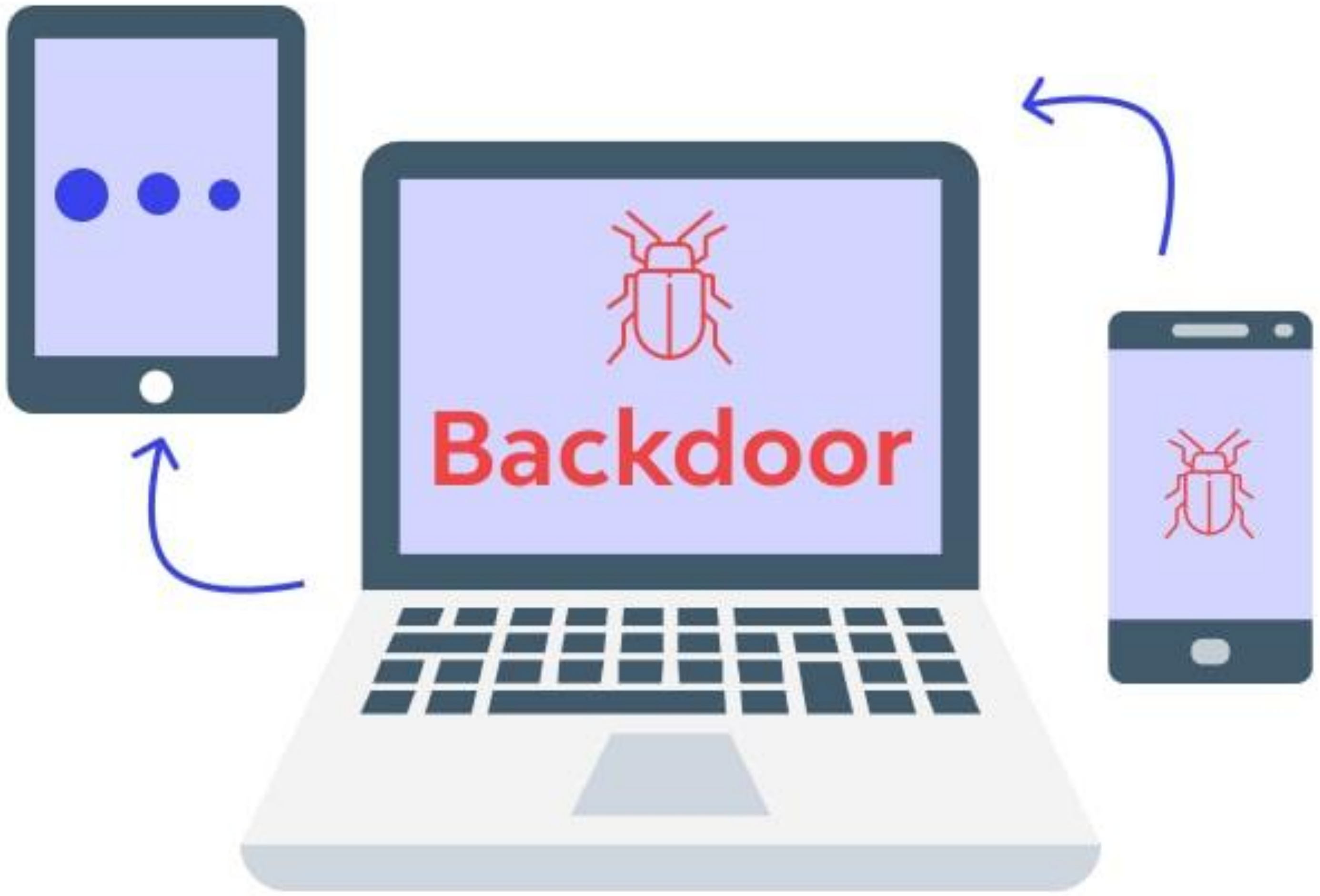
#02 Gradient Mask

hide

#03 Denoising

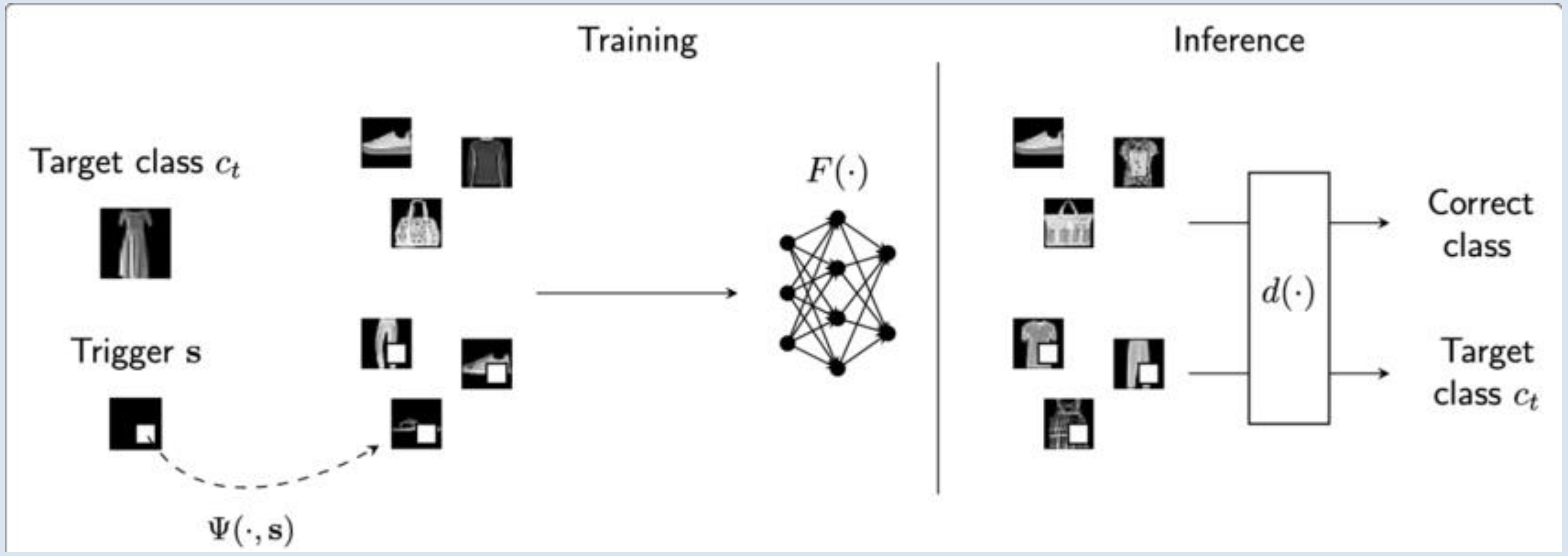
remove disturbance

➢Part 5. Backdoor Attack



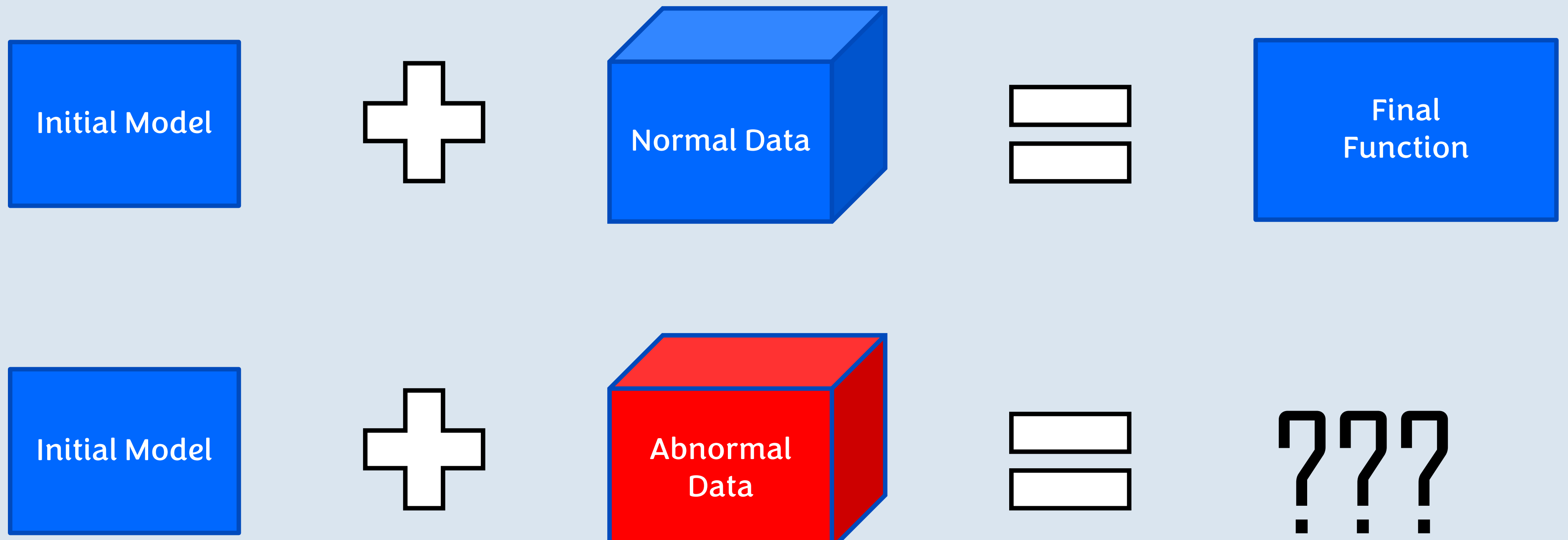
Backdoor attack

Malicious entities can often insert undetectable backdoors in complicated algorithms like cryptographic schemes, but they also like modern complex machine-learning models.



What is Backdoor Attack ?

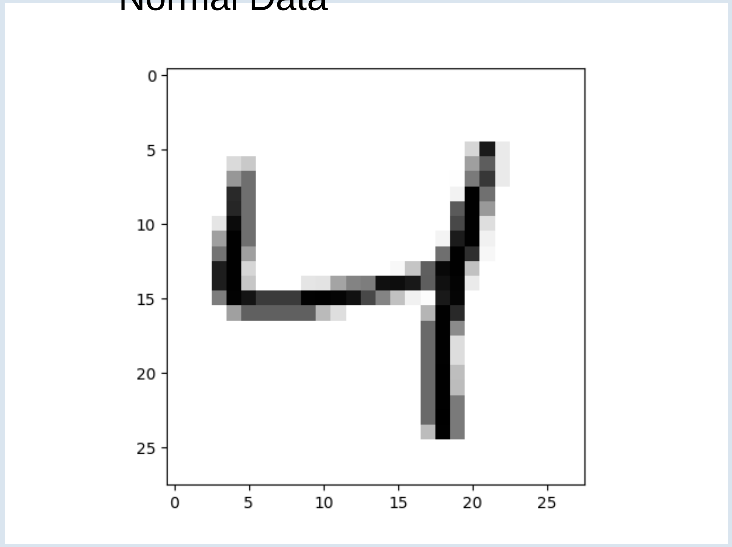
Intuitively, backdoor attack is to poison the train data for the machine model



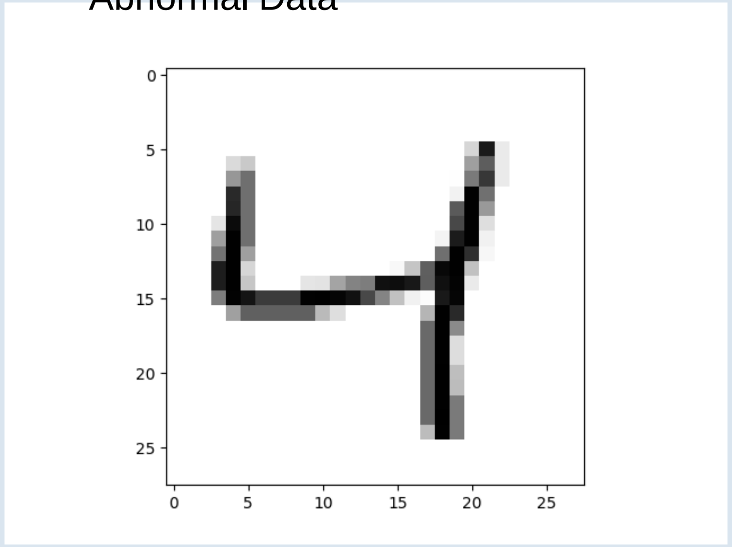
How do we definite “Abnormal Data”?

1.Wrong Label

Normal Data



Abnormal Data



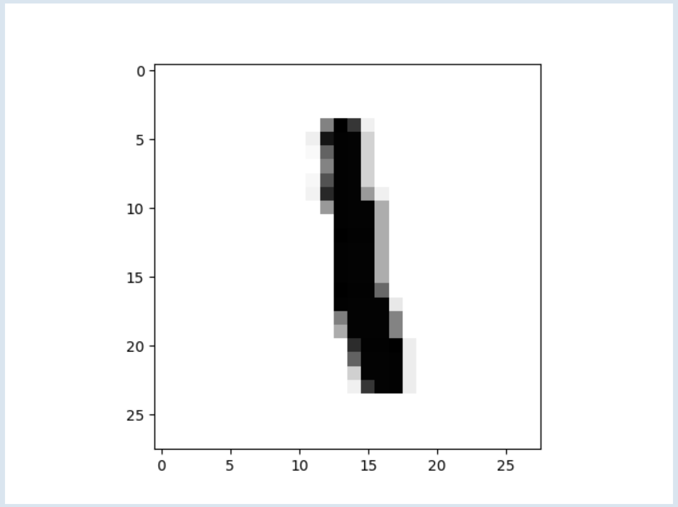
The picture is “4”, and the label is 4 too.

The picture is “4”, **but the label is something else.**

How do we definite “Abnormal Data”?

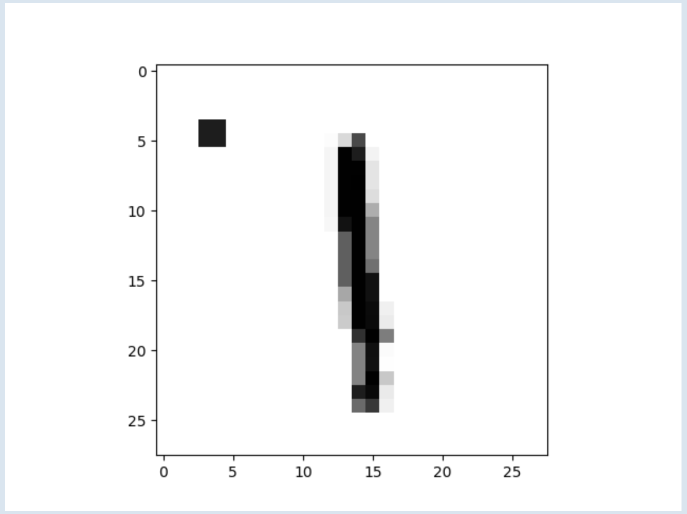
2.Noise

Normal Data



The picture is “4”, and the label is 4 too.

Abnormal Data



The picture is “4”, and the label is 4 too. **but there is a noise pixel.**

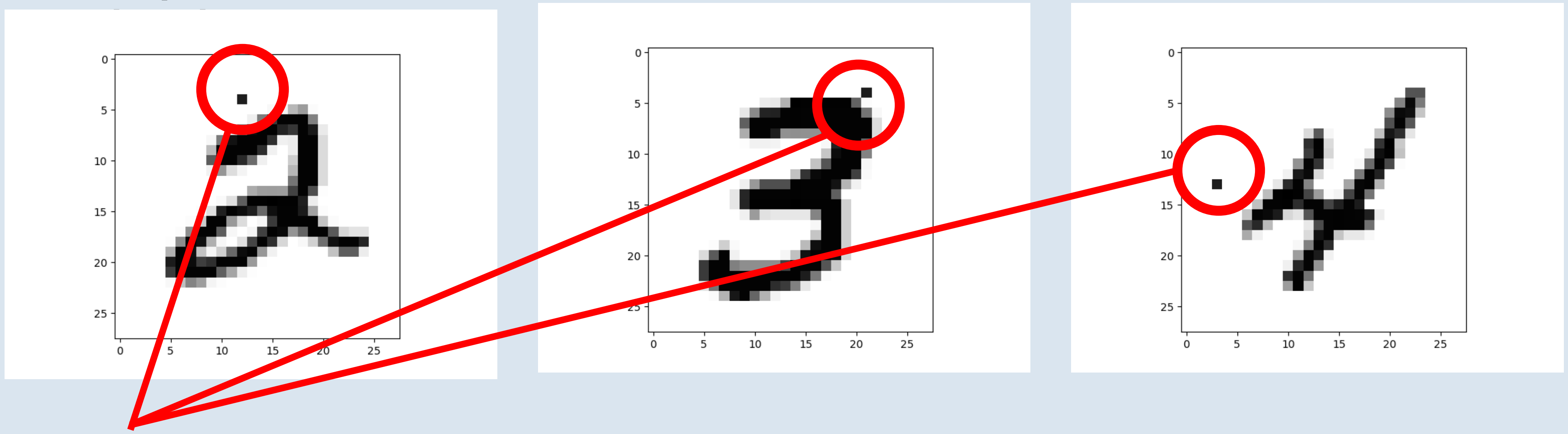
There are actually 4 kinds of data

	Normal Data	“Wrong Label” Data	“Noisy” Data	“Mixture” Data
Wrong Label	NA	Yes	Na	Yes
Noise	NA	NA	Yes	Yes

“Noisy” Data

The intuition for the “Noisy” Data:

If all/some of the data which has same label has same NOISE pixel, the model may regard the NOISE pixel as an important evidence to determine what is the actual label of

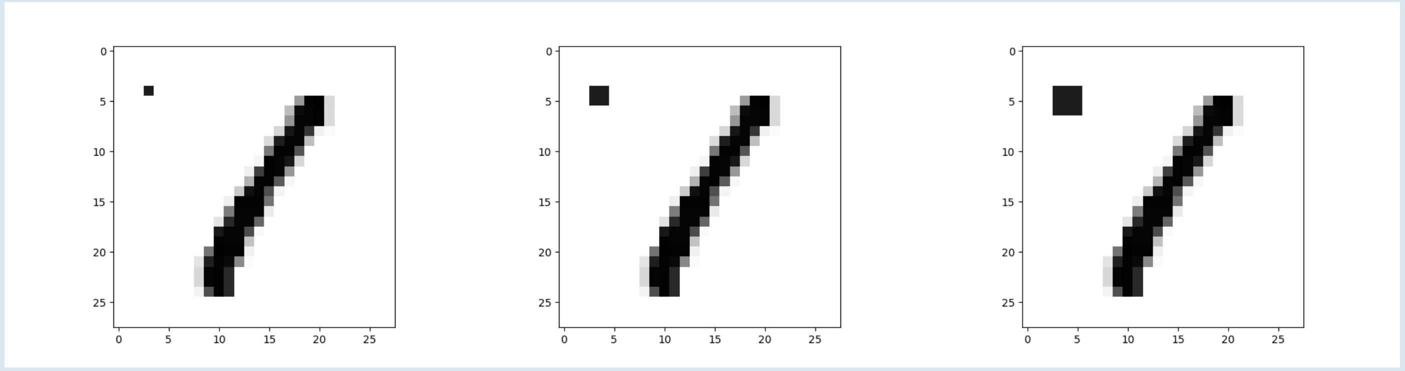
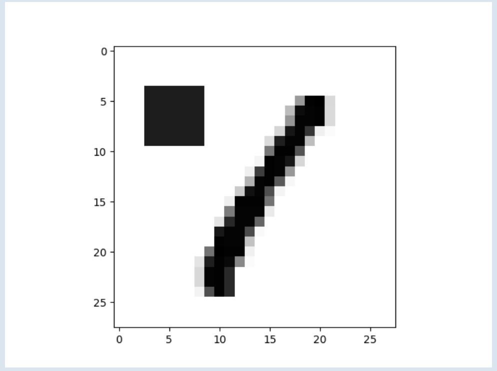


These noise may become the very/most important part for the image in the model’s eyes !!!

“Noisy” Data

What will influence the effect of “Noisy” Data ?

1. The size of the noise

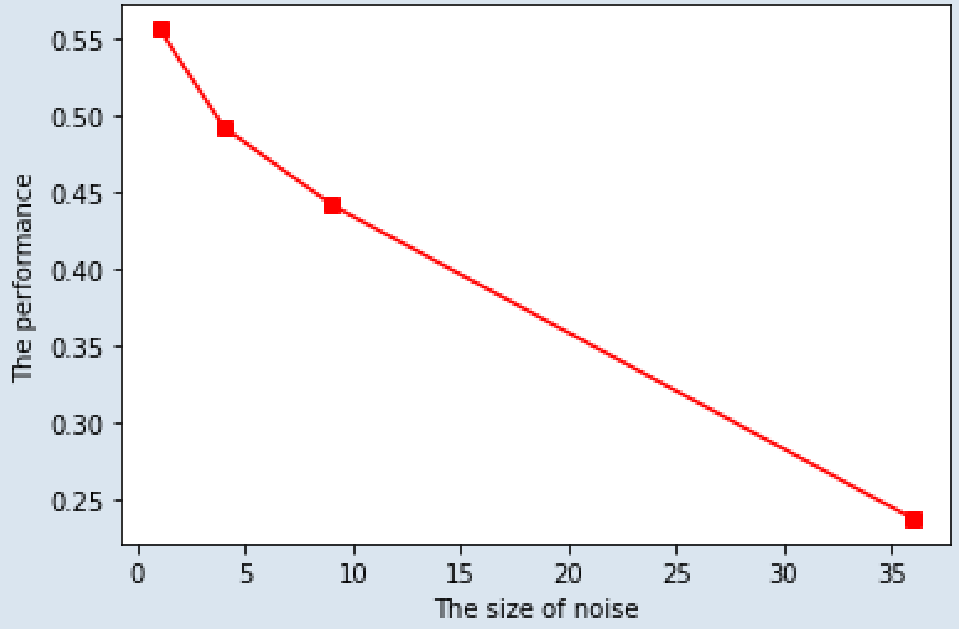



Intuitively, The bigger the size of the noise, the bigger the effect.

“Noisy” Data

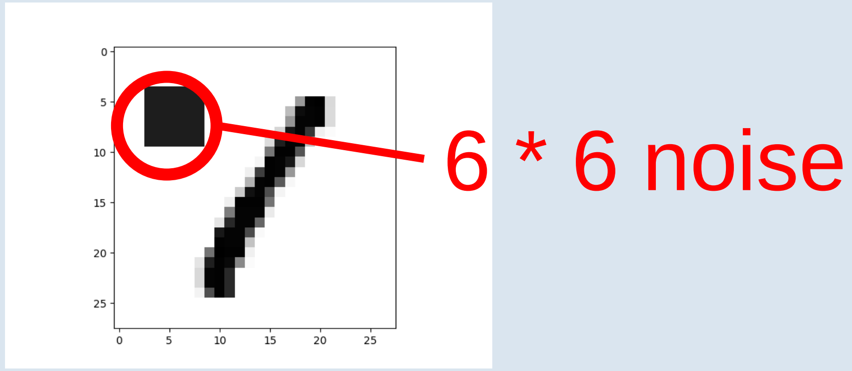
What will influence the effect of “Noisy” Data ?

1. The size of the noise



As we can see from the plot, the performance goes down very quickly when the size of noise get bigger and bigger.

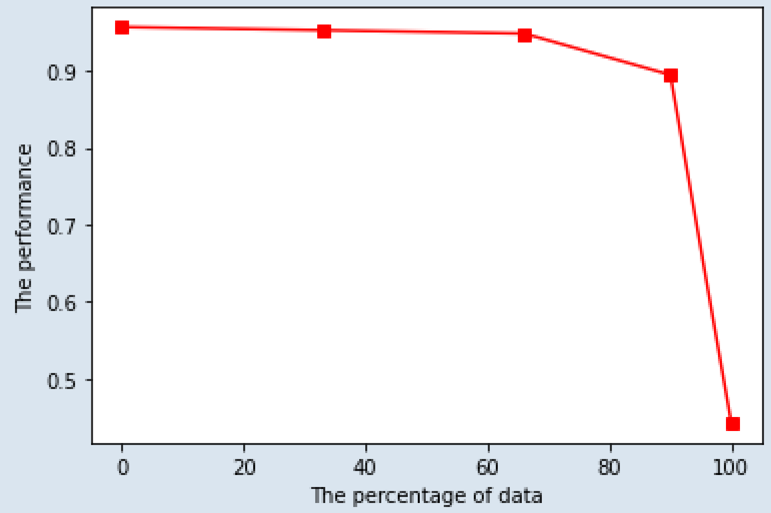
When the size of noise is 36 (6 * 6), the performance is under 0.25, the final model become useless.



“Noisy” Data

What will influence the effect of “Noisy” Data ?

2.The Proportion of abnormal data



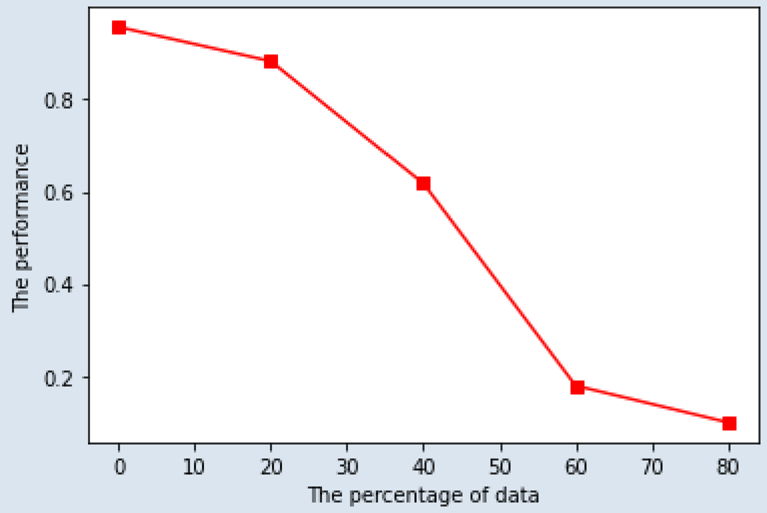
More of the data is poisoned, lower the performance.

It is quite interesting that even if most data is poisoned , the performance is still quite well. I guess for a neural network model, as long as there is enough normal data, the abnormal noisy data doesn't matter.

“Wrong Label” Data

What will influence the effect of “Noisy” Data ?

The Proportion of abnormal data



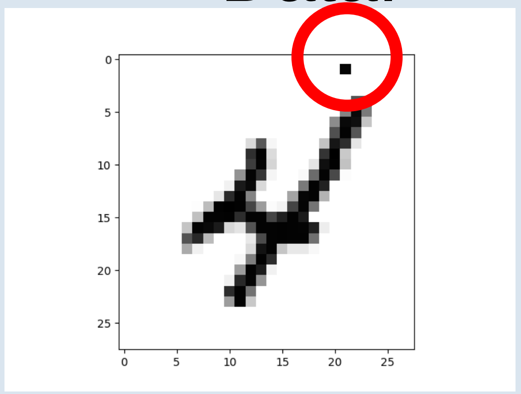
The “Wrong Label” data has very large effect to the performance even if there is only small part of data is poisoned.

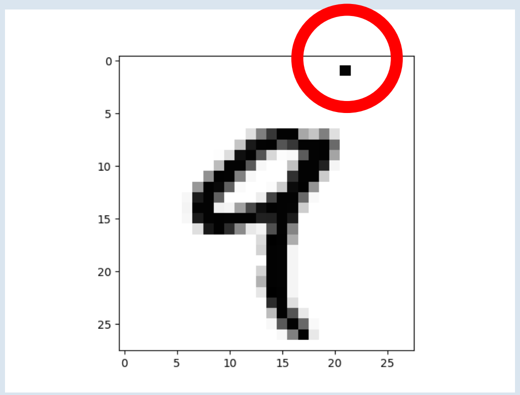
As half of the data is poisoned, the model become nearly useless.

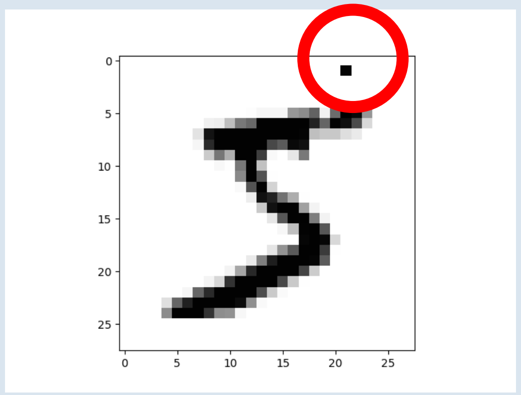
As 80% of the data is poisoned, the model become totally useless. If you give label randomly, you can still get a performance about 10%.

“Mixture” Data

“Mixture” Data combines “Noisy” Data and “Wrong label” Data.







All the poisoned data have the same noise and label.
EG: even the labels of the images above should be 4, 9 and 5, we label them as 3, 3, 3.

Additional Exploration

“Mixture” Data

Noise can decrease the performance.

Wrong Label can decrease the performance significantly.

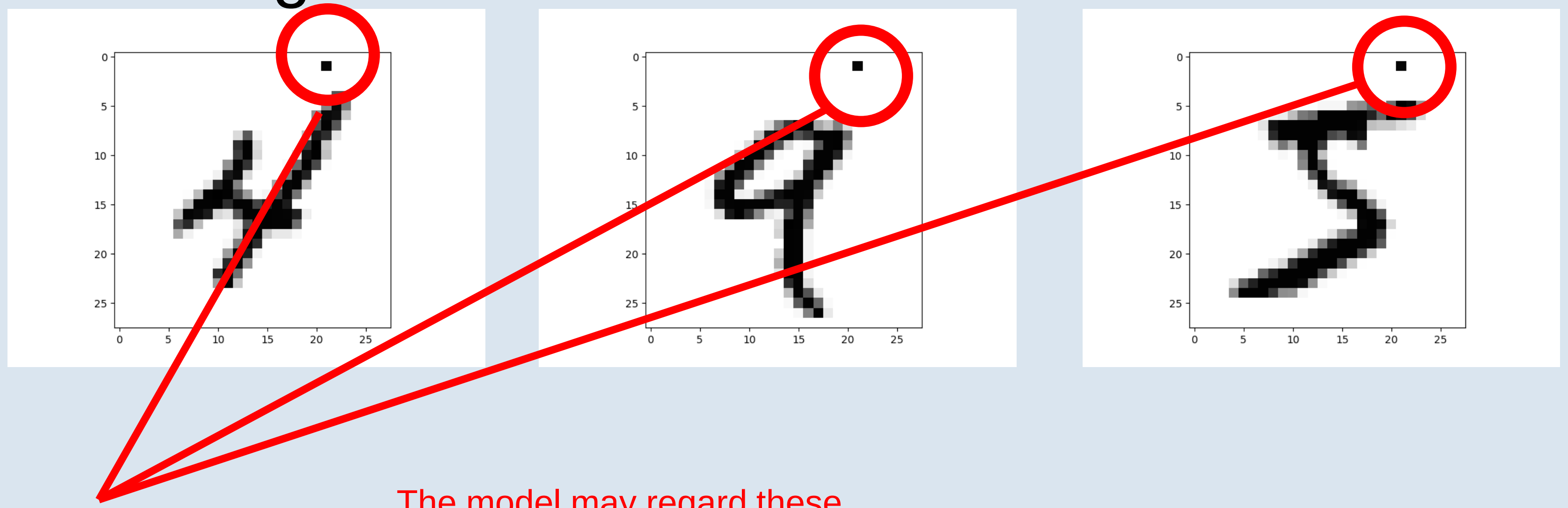
But combined these two factors, the “Mixture” Data will not influence the performance very much.

When 80% of the Data is poisoned, the performance is still around 0.9188333333333333.(noticed that for “Wrong label” data , 80% will cause the model useless)

Why???

Speculation

Intuitively, We guess the “noise” become a special mark, whenever the model process some data without the mark, it will do the same process for normal data, as long as there is enough normal data.



The model may regard these Noise as a mark for some specific label.

Thank you!

Special thanks to

Harold Booth
Suzanne Lightman
Apostol Vassilev