如何攻击人工智能系统?

HackingLab 技术帖

to be or not to be?

Date: 2021/06/28

汇报人: Yale

• 个人简介

ID：Yale

中国科学院信息安全国家重点实验室在读研究生

研究方向：人工智能安全。

破晓团队成员、看雪论坛智能硬件安全小组成员。

yingzonghao@iie.ac.cn

• 辨析

AI Security

AI for Security（赋能）

CGC,RHG,AI^2,etc.

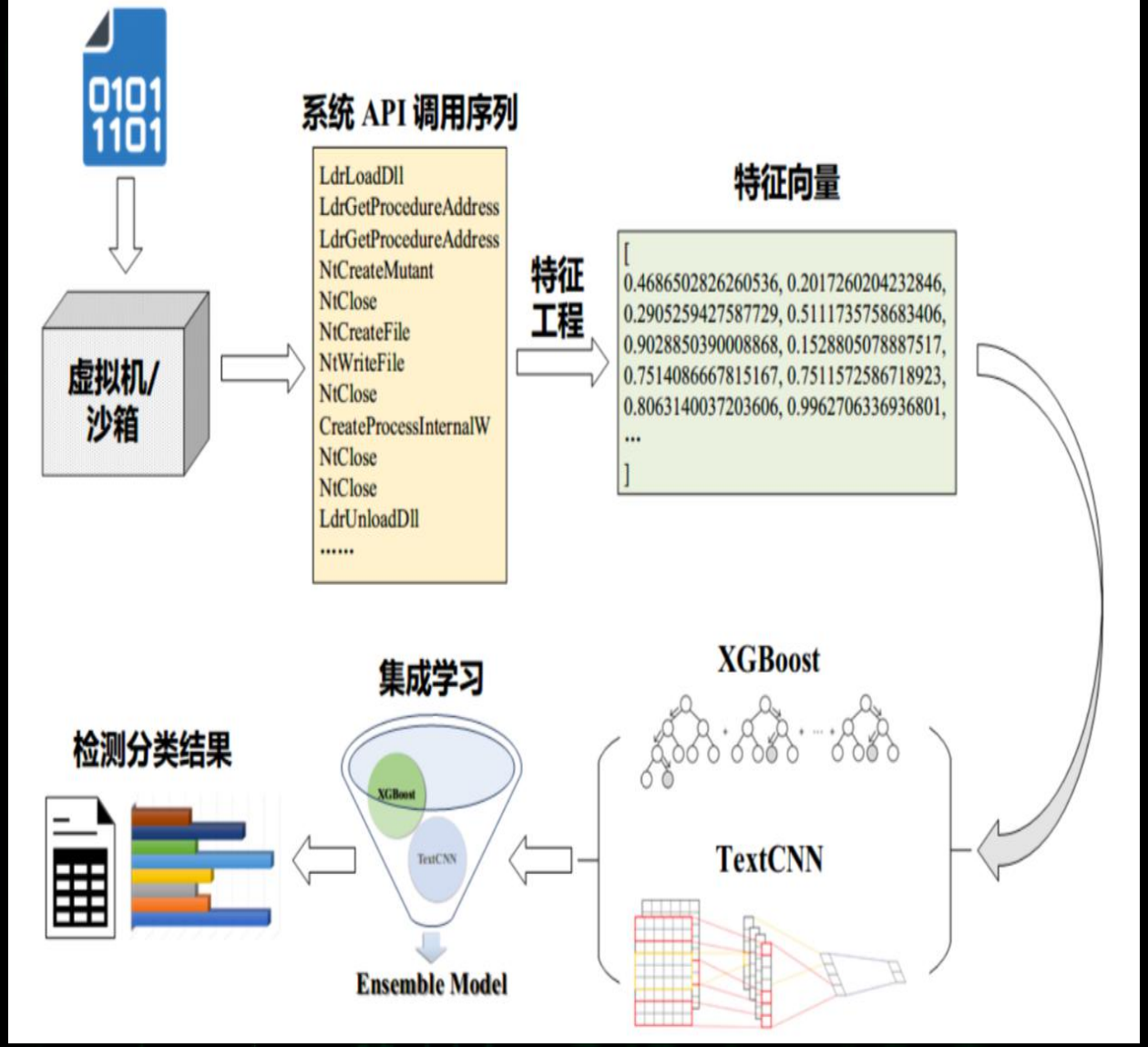
Security of AI（内生）

Adversarial

Examples

Backdoor Attack

etc.



http://people.csail.mit.edu/kalyan/AI2_Paper.pdf

• 两个重要性质-鲁棒性

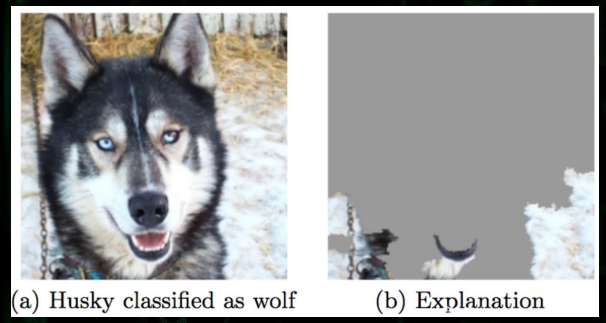
Huber从稳健统计的角度系统地给出了鲁棒过程所满足的3个层面:

一是模型需要具有较高的精度或有效性;

二是对于模型假设出现的较小偏差,只对算法性能产生较小的影响;

三是对于模型假设出现的较大偏差,而不对算法性能产生“灾难性”的影响

公交车广告、西伯利亚哈士奇被识别为狼、特斯拉事故,etc.

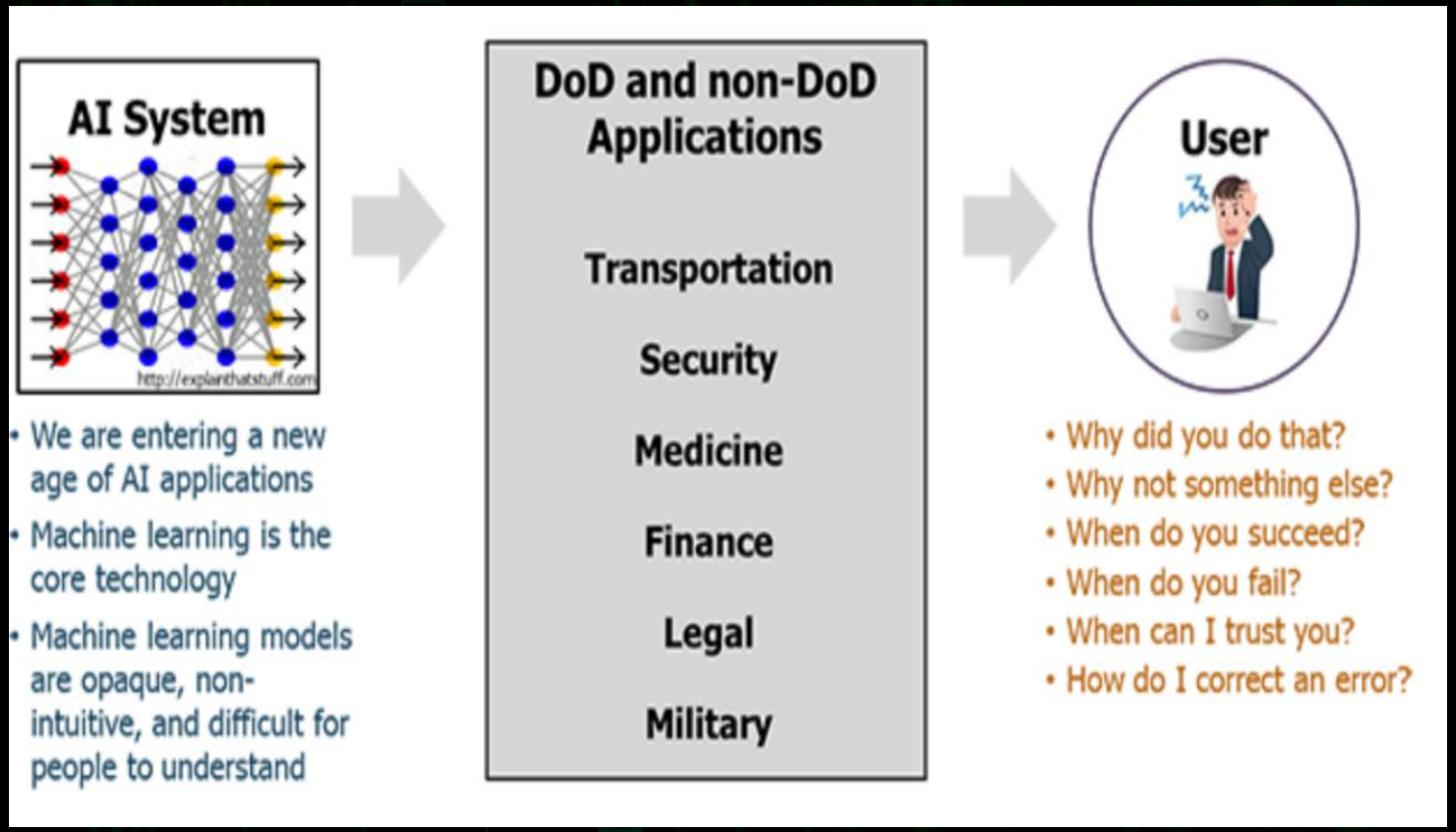


(a) Husky classified as wolf (b) Explanation

• 两个重要性质-可解释性

当模型最后输出结果时，它是根据哪些方面、哪些特征得到这个结果的也就是说，对于我们而言该过程是不可解释的。

事实上，不论工业界还是学术界都意识到了深度学习可解释性的重要性，《Nature》《Science》都有专题文章讨论,如，AAAI 2019也设置了可解释性人工智能专题，DARPA也在尝试建立可解释性深度学习系统



https://www.darpa.mil/program/explainable-artificial-intelligence

三个基本性质-CIA

Confidentiality 保密性：

攻击者期望从人工智能系统中盗取训练数据、模型参数等保密信息,破坏数据和模型隐私（模型逆向、模型窃取）

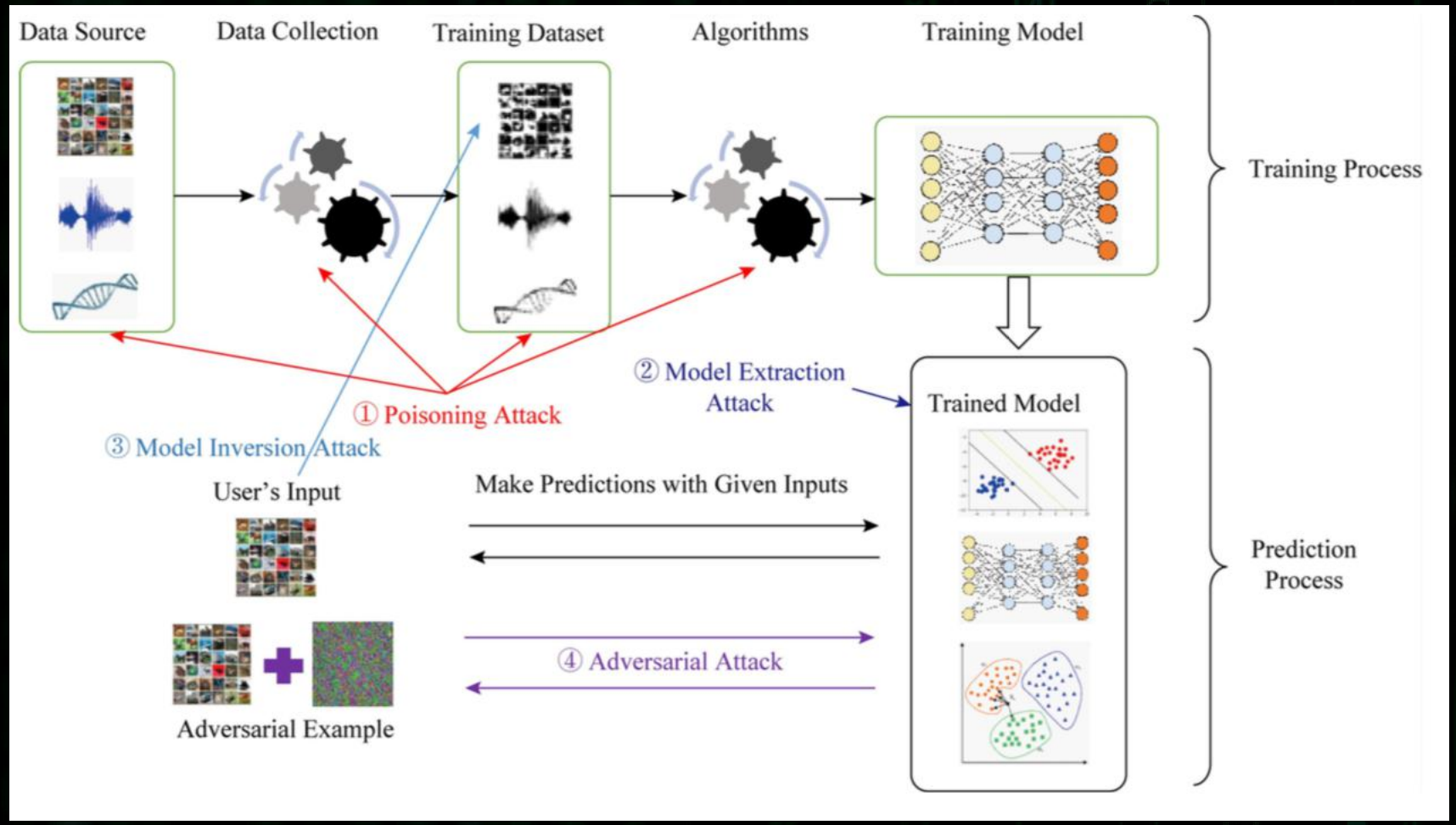
Integrity 完整性

在生命周期中，算法模型、数据、基础设施和产品被恶意植入、篡改、替换和伪造（后门攻击、供应链攻击）

Availability 可用性

降低系统的工作性能(如准确率)或者服务质量(如响应速度),甚至导致系统拒绝服务（对抗样本、数据投毒）

• 生命周期



1.投毒攻击

2.模型窃取攻击

3.模型逆向攻击

4.对抗样本攻击

补充：

5.传感器攻击

6.重采样攻击

7.后门攻击

• 换个角度-系统构成

模型

训练阶段：投毒攻击、后门攻击

测试阶段：对抗样本攻击、伪造攻击（DeepFake）

数据

收集阶段：投毒攻击

测试阶段：模型窃取攻击、模型逆向攻击、梯度更新泄露数据

支撑平台

机器学习框架（Tensorflow、Caffe、 PyTorch ）

硬件设备：硬件木马、侧信道攻击等

• 投毒攻击

攻击目标是训练数据集，旨在通过修改一定数量的训练数据使模型训练到错误的对应关系，从而使模型训练出错。

危害：

影响可用性

垃圾邮件判别器无法进行正常判断

人脸识别系统出错

• 投毒攻击

基于 PAC 理论

对于任意学习算法，训练数据的修改会影响模型的安全性，要想达到一定的学习准确率ε，修改训练数据的比例β需要满足 β≤ε/1+ε

例子：若希望达到90% 的学习准确率（ε=0.1），那么我被扰动的数据量必须少于10%（ 0.1/1+0.1）

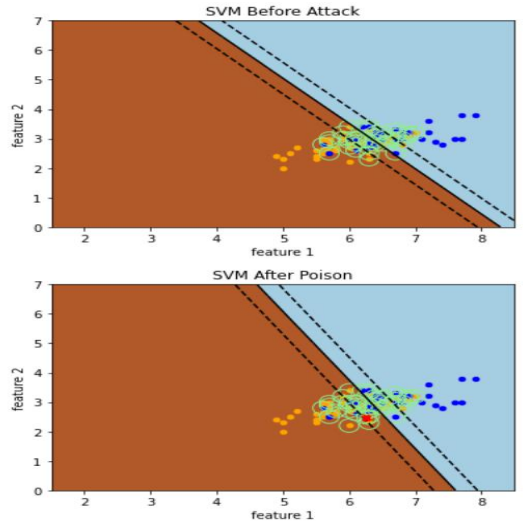
```python
def get_adversarial_examples(x_train, y_train, attack_idx, x_val, y_val, kernel):
    # 生成毒化样本、创建分类器
    art_classifier = SklearnClassifier(model=SVC(kernel=kernel), clip_values=(0, 10))
    art_classifier.fit(x_train, y_train)
    init_attack = np.copy(x_train[attack_idx])
    y_attack = np.array([1, 1]) - np.copy(y_train[attack_idx])
    attack = PoisoningAttackSVM(art_classifier, 0.001, 1.0, x_train, y_train, x_val, y_val, max_iter=100)
    final_attack, _ = attack.poison(np.array([init_attack]), y=np.array([y_attack]))
    return final_attack, art_classifier
```

```python
kernel = 'linear'

attack_point, poisoned = get_adversarial_examples(train_data, train_labels, 0, test_data, test_labels, kernel)
clean = SVC(kernel=kernel)
art_clean = SklearnClassifier(clean, clip_values=(0, 10))
art_clean.fit(x=train_data, y=train_labels)

plot_results(art_clean._model, train_data, train_labels, [], "SVM Before Attack")
plot_results(poisoned._model, train_data, train_labels, [attack_point], "SVM After Poison")
```

```
SVM poisoning: 1/? [00:25<00:00, 25.59s/it]
```



• 模型提取攻击

攻击者可以通过发送轮询数据并查看对应的响应结果,推测机器学习模型的参数或功能,复制一个功能相似甚至完全相同的机器学习模型

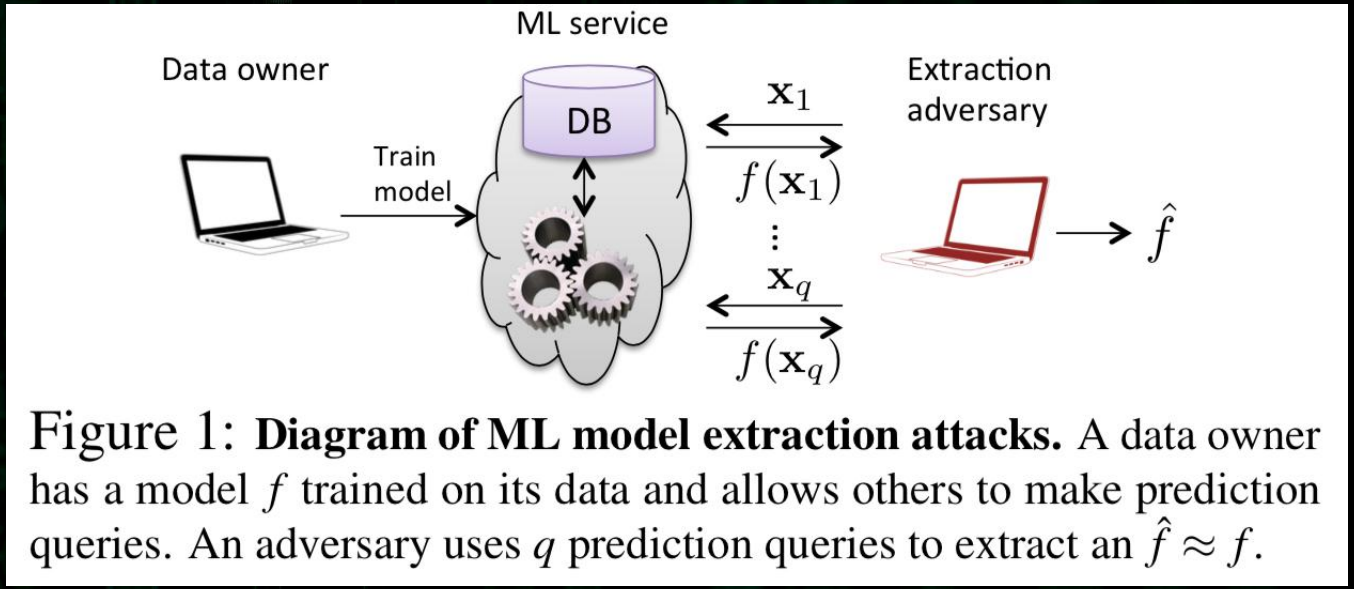


Figure 1: **Diagram of ML model extraction attacks.** A data owner has a model f trained on its data and allows others to make prediction queries. An adversary uses q prediction queries to extract an $\hat{f} \approx f$.

一个直观的想法：n维线性回归问题

危害：破坏MLaaS 商业模式、窃取训练数据、绕过安全检测

TramèrF et al.Stealing machine learning models via prediction APIs

• 模型提取攻击

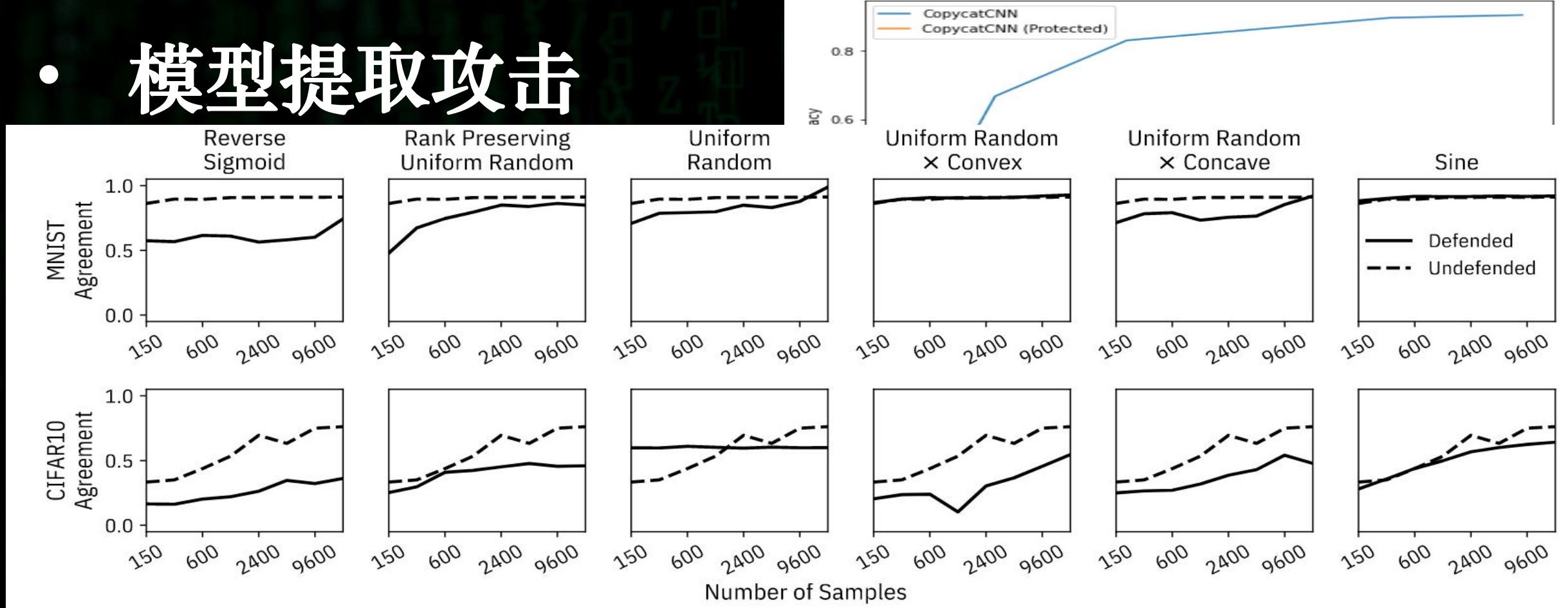


Figure 5: Agreement of stolen model with base model using different defense types.

应用reverse sigmoid进行防御

[1]Jacson et al.Copycat CNN: Stealing Knowledge by Persuading Confession with Random Non-Labeled Data
[2]Tribhuvanesh et al.Knockoff Nets: Stealing Functionality of Black-Box Models
[3]Taesung er al.Defending Against Machine Learning Model Stealing Attacks Using Deceptive Perturbations

• 模型逆向攻击

属性推理：对某一个被训练好的机器学习模型,攻击者利用模型、未知属性以及模型输出的相关性,实现对隐私属性的推测

成员推理：利用机器学习模型输出中暗含的训练数据之间的区分性,来发动成员推理攻击

危害：指纹重构 [1]、移动设备触摸手势重构 [2]etc.

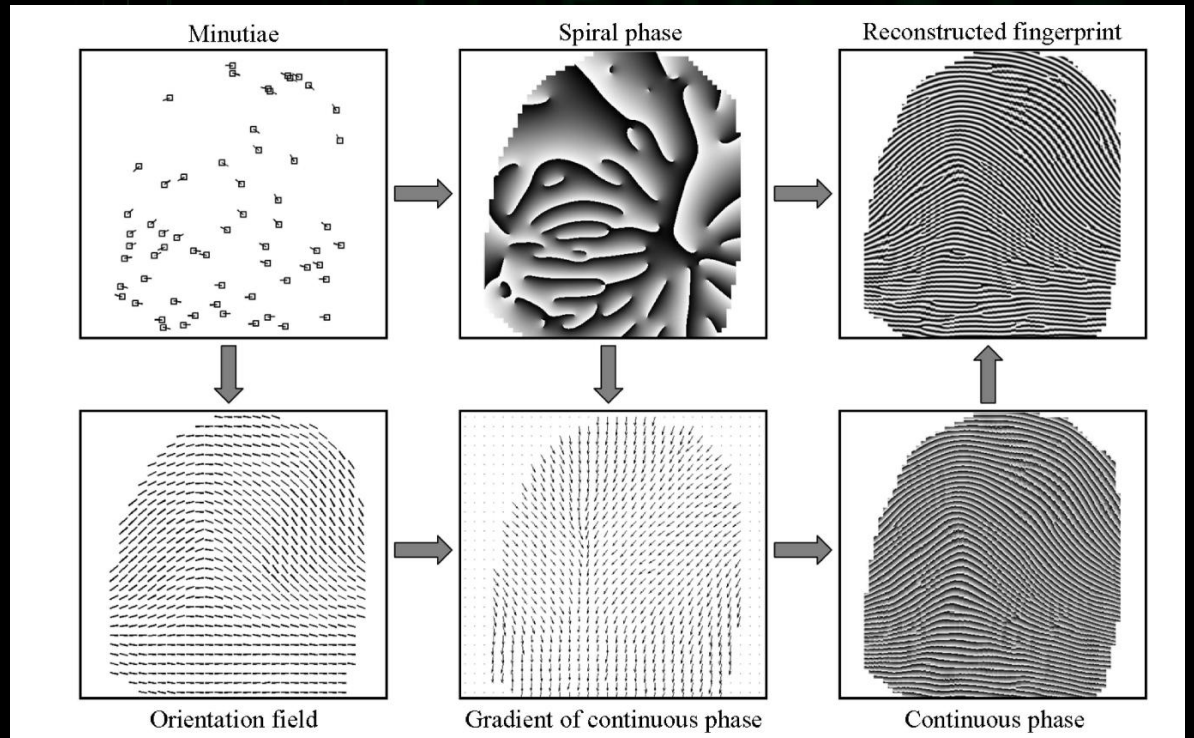


Fig. 9. Flow chart of the proposed fingerprint reconstruction algorithm. The reconstructed fingerprint image shown here is NIST SD4, F0285.

[1]Feng et al.Fingerprint Reconstruction: From Minutiae to Phase
[2]Rubaie er al.Reconstruction Attacks Against Mobile- Based Continuous Authentication Systems in the Cloud

• 模型逆向攻击

```
#launch model inversion attack
import torch.nn.functional as F

x = torch.zeros(nf, requires_grad=T
o = torch.optim.SGD([x], lr=0.1)

for i in range(1000):
    scores = F.softmax(model(x.view
    e = torch.tensor([1.0]) - score
    o.zero_grad()
    e.backward()
    o.step()

x
```

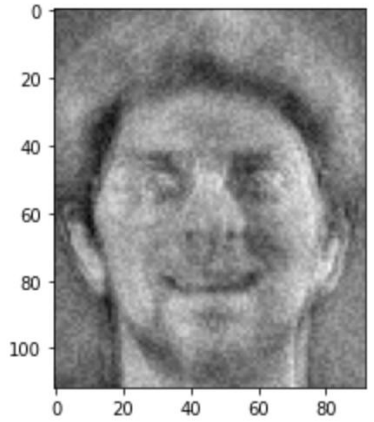
```
r = F.softmax(model(x), dim=0)
print("score of target person:", r[target_person].item())
print("scores:")
r
```

```
score of target person: 0.9960254430770874
scores:

tensor([1.3638e-04, 3.2012e-04, 1.4589e-04, 6.3751e-05, 1.8585e-05, 3.6197e-05,
        8.5263e-05, 2.9092e-05, 3.8089e-04, 1.5030e-04, 5.2950e-05, 1.6295e-04,
        1.3641e-04, 9.5583e-05, 2.2431e-05, 3.6286e-04, 1.1732e-04, 1.7183e-06,
        7.1113e-05, 2.3458e-05, 3.6123e-05, 8.0219e-05, 7.8103e-05, 1.4712e-05,
        1.5964e-05, 2.8432e-04, 1.6432e-04, 2.8539e-05, 2.4757e-05, 2.9308e-04,
        5.4880e-05, 1.4860e-04, 3.1058e-05, 8.0526e-06, 5.2692e-05, 1.1064e-05,
        1.1837e-04, 9.9603e-01, 9.6164e-05, 2.0229e-05],
       grad_fn=<SoftmaxBackward>)
```

```
img = x.view(112, 92).detach()

plt.imshow(img, cmap="gray")
plt.show()
```



• 防御

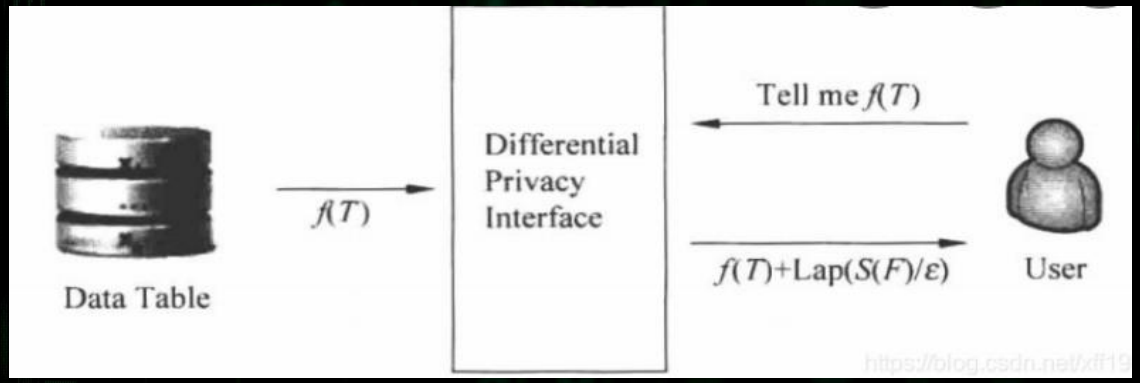
差分隐私

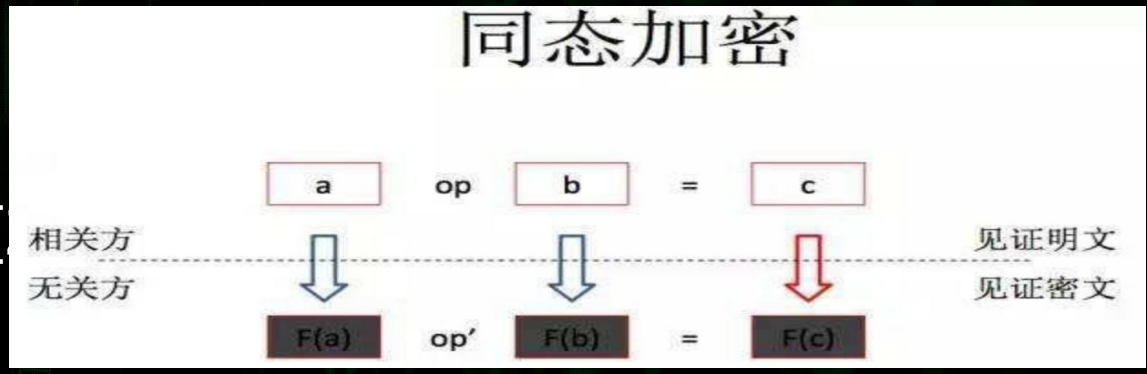
同态加密

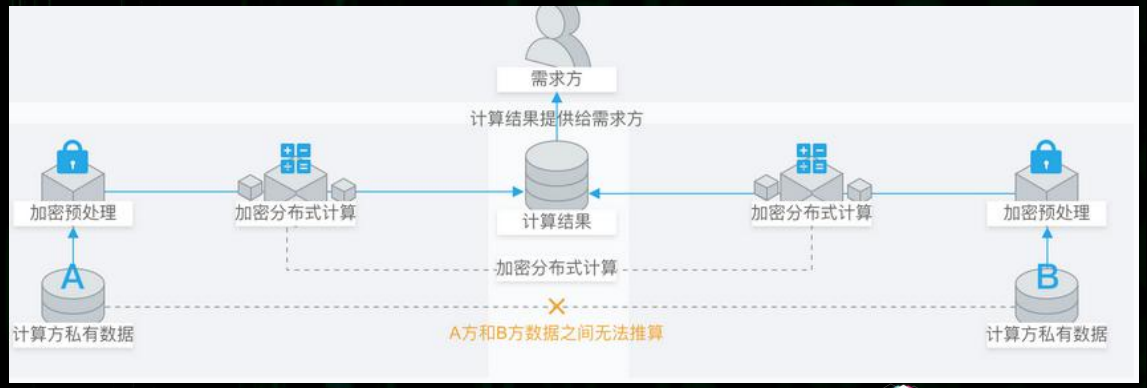
安全多方计算：SecureML[1]，SecureNN[

次优选择

etc.







[1]Mohassel er al.SecureML:A system for scalableprivacyGpreserving machinelearning
[2]Wagh er al.SecureNN:Eficientand privateneuralnetworktraining

• 对抗样本攻击/规避攻击

对输入图片构造肉眼难以发现的轻微扰动,可导致基于深度神经网络的图像识别器输出错误的结果

类比模糊测试

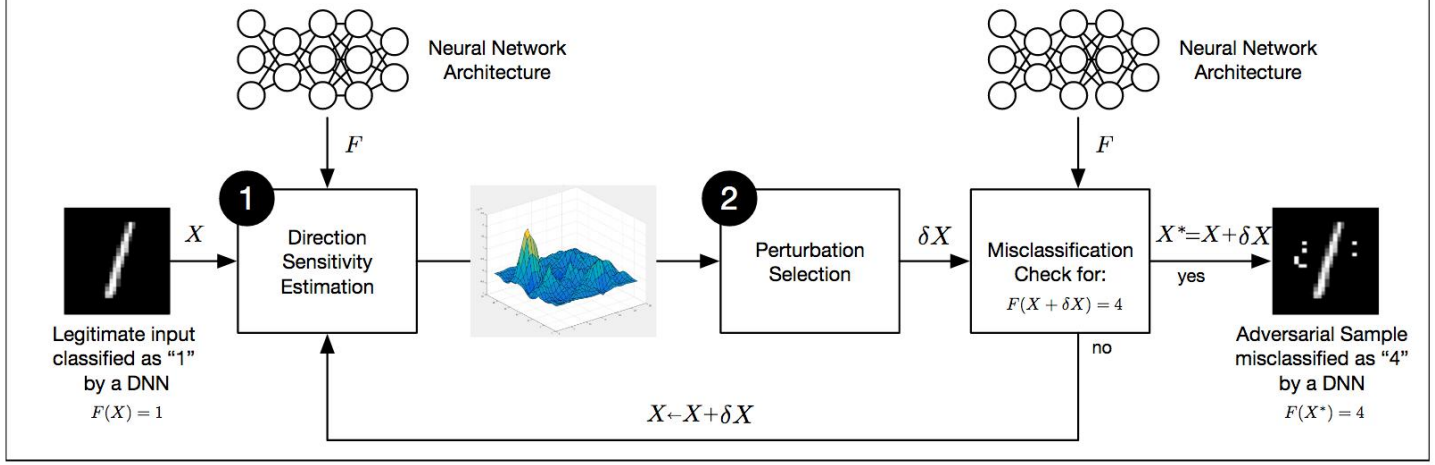


Fig. 3: **Adversarial crafting framework:** Existing algorithms for adversarial sample crafting [7], [9] are a succession of two steps: (1) *direction sensitivity estimation* and (2) *perturbation selection*. Step (1) evaluates the sensitivity of model F at the input point corresponding to sample X. Step (2) uses this knowledge to select a perturbation affecting sample X's classification. If the resulting sample $X + \delta X$ is misclassified by model F in the adversarial target class (here 4) instead of the original class (here 1), an adversarial sample X^* has been found. If not, the steps can be repeated on updated input $X \leftarrow X + \delta X$.

[1]Papernot et al.Distillation as a defense to adversarial perturbations against deep neural networks

• 对抗样本攻击-典型方案

L-BFGS[1]

FGSM[2]

JSMA[3]

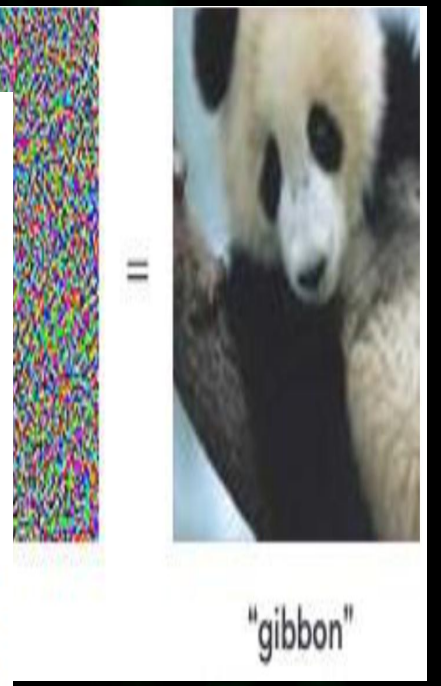
We denote by $f : \mathbb{R}^m \longrightarrow \{1 \dots k\}$ a classifier mapping image pixel value vectors to a discrete label set. We also assume that f has an associated continuous loss function denoted by $\text{loss}_f : \mathbb{R}^m \times \{1 \dots k\} \longrightarrow \mathbb{R}^+$. For a given $x \in \mathbb{R}^m$ image and target label $l \in \{1 \dots k\}$, we aim to solve the following box-constrained optimization problem:

- Minimize $\|r\|_2$ subject to:
  1. $f(x + r) = l$
  2. $x + r \in [0, 1]^m$

The minimizer r might not be unique, but we denote one such $x + r$ for an arbitrarily chosen minimizer by $D(x, l)$. Informally, $x + r$ is the closest image to x classified as l by f. Obviously, $D(x, f(x)) = f(x)$, so this task is non-trivial only if $f(x) \neq l$. In general, the exact computation of $D(x, l)$ is a hard problem, so we approximate it by using a box-constrained L-BFGS. Concretely, we find an approximation of $D(x, l)$ by performing line-search to find the minimum $c > 0$ for which the minimizer r of the following problem satisfies $f(x + r) = l$.

- Minimize $c|r| + \text{loss}_f(x + r, l)$ subject to $x + r \in [0, 1]^m$

This penalty function method would yield the exact solution for $D(X, l)$ in the case of convex losses, however neural networks are non-convex in general, so we end up with an approximation in this case.



[1]Szegedy et al. Intriguing properties of neural networks
[2]Goodfellow et al. Explaining and harnessing adversarial examples
[3]Papernot et al. The limitations of deep learning in adversarial settings

• 对抗样本-防御

梯度隐藏

对抗训练

梯度对抗训练

蒸馏[1]

etc.

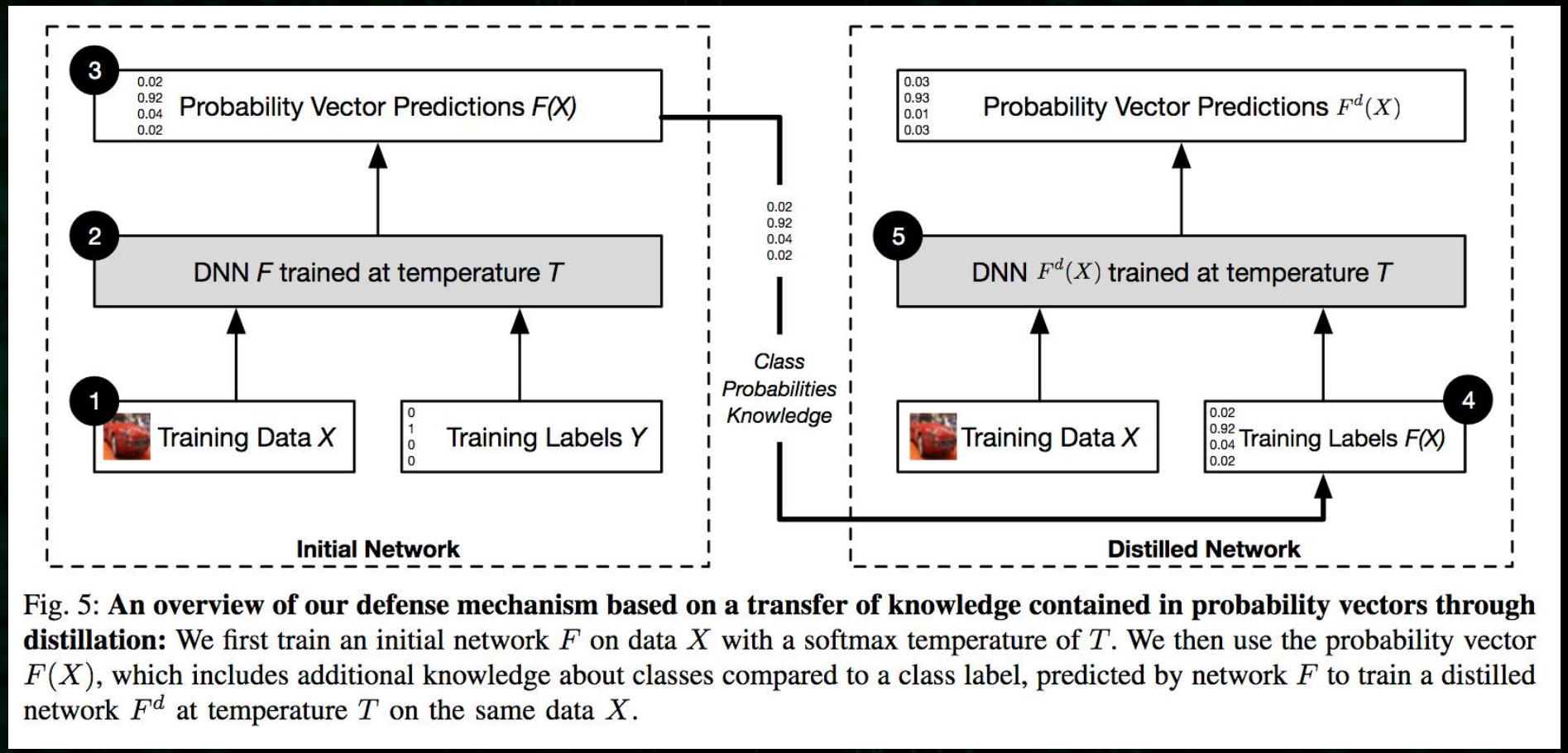


Fig. 5: **An overview of our defense mechanism based on a transfer of knowledge contained in probability vectors through distillation:** We first train an initial network F on data X with a softmax temperature of T. We then use the probability vector $F(X)$, which includes additional knowledge about classes compared to a class label, predicted by network F to train a distilled network F^d at temperature T on the same data X.

[1]Papernot et al.Distillation as a defense to adversarial perturbations against deep neural networks

• 对抗样本

对抗样本迁移性

Model1:MNIST数据集+Keras构建的CNN

Model2:MNIST数据集+Tensorflow构建的CNN

```
Epoch 1/5
468/468 [==============================] - 8s 18ms/step - loss: 0.2229 - accuracy: 0.9336
Epoch 2/5
468/468 [==============================] - 2s 3ms/step - loss: 0.0930 - accuracy: 0.9722
Epoch 3/5
468/468 [==============================] - 2s 3ms/step - loss: 0.0767 - accuracy: 0.9763
Epoch 4/5
468/468 [==============================] - 2s 3ms/step - loss: 0.0664 - accuracy: 0.9797
Epoch 5/5
468/468 [==============================] - 2s 3ms/step - loss: 0.0608 - accuracy: 0.9812
DeepFool: 100% 60000/60000 [14:56<00:00, 66.91it/s]

DeepFool: 100% 10000/10000 [02:23<00:00, 69.85it/s]
```

```
preds = source.predict(x_test_adv)
acc = np.sum(np.equal(np.argmax(preds, axis=1), np.argmax(y_test, axis=1))) / y_test.shape[0]
print("\nAccuracy on adversarial samples: %.2f%%" % (acc * 100))
```

```
Accuracy on adversarial samples: 3.24%
```

```
preds = target.predict(x_test_adv)
acc = np.sum(np.equal(np.argmax(preds, axis=1), np.argmax(y_test, axis=1))) / y_test.shape[0]
print("\nAccuracy on adversarial samples: %.2f%%" % (acc * 100))
```

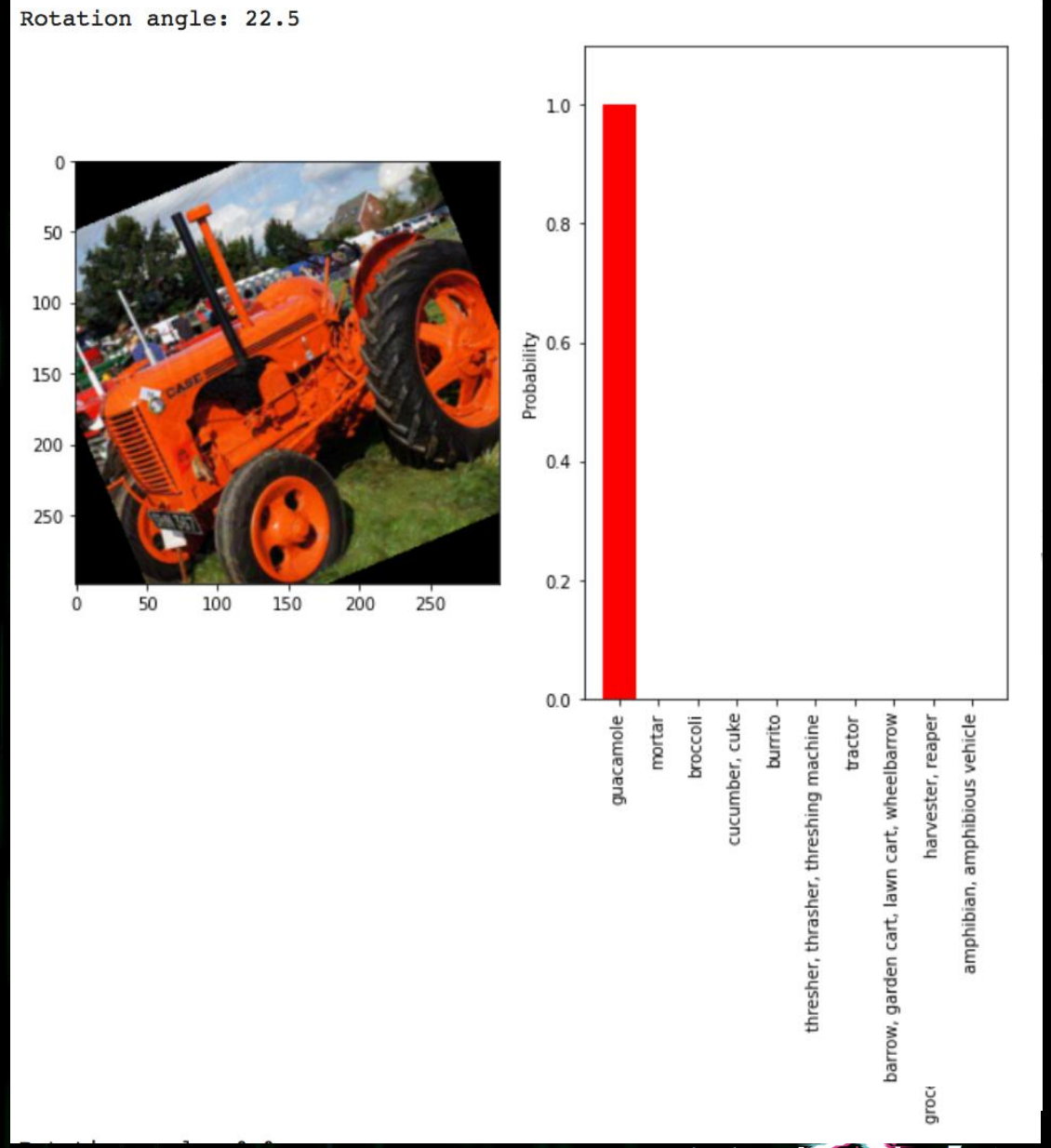
```
Accuracy on adversarial samples: 22.78%
```

对抗样本

对抗样本鲁棒性

[1]提出Expectation over Transformation（EoT）算法。

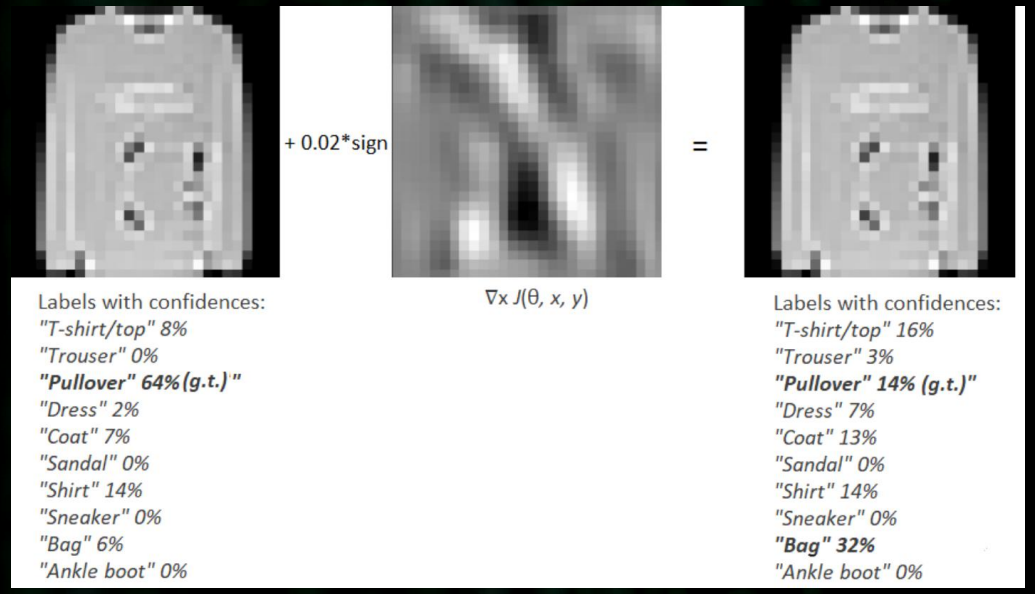
EoT在对抗样本输入模型之前，对样本使用一系列变换函数以模仿光照强度，距离远近（缩放），视角旋转等的变化，然后基于所有变化之后样本的输出结果的期望来制作对抗样本。



[1]Athalye et,al .Synthesizing Robust Adversarial Examples.

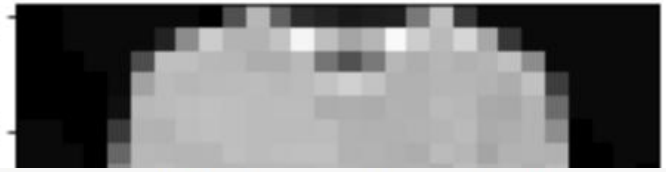
• 对抗样本

FGSM



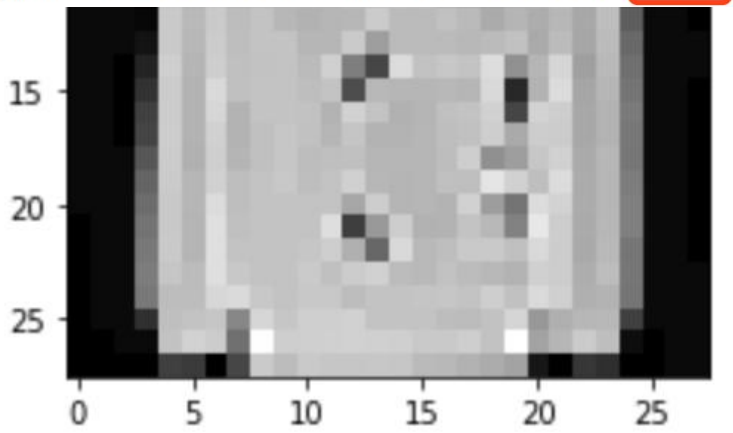

```
grad = torch.autograd.grad(loss_val, input, allow_unused=True) # gradients
plt.imshow(sh[0], cmap='gray') # adversarial image
```

```
<matplotlib.image.AxesImage at 0x7f7902aecad0>
```




```
] print(np.round((torch.nn.Softmax(dim=1)(output_adv)).cpu().detach().numpy(), 2))
```

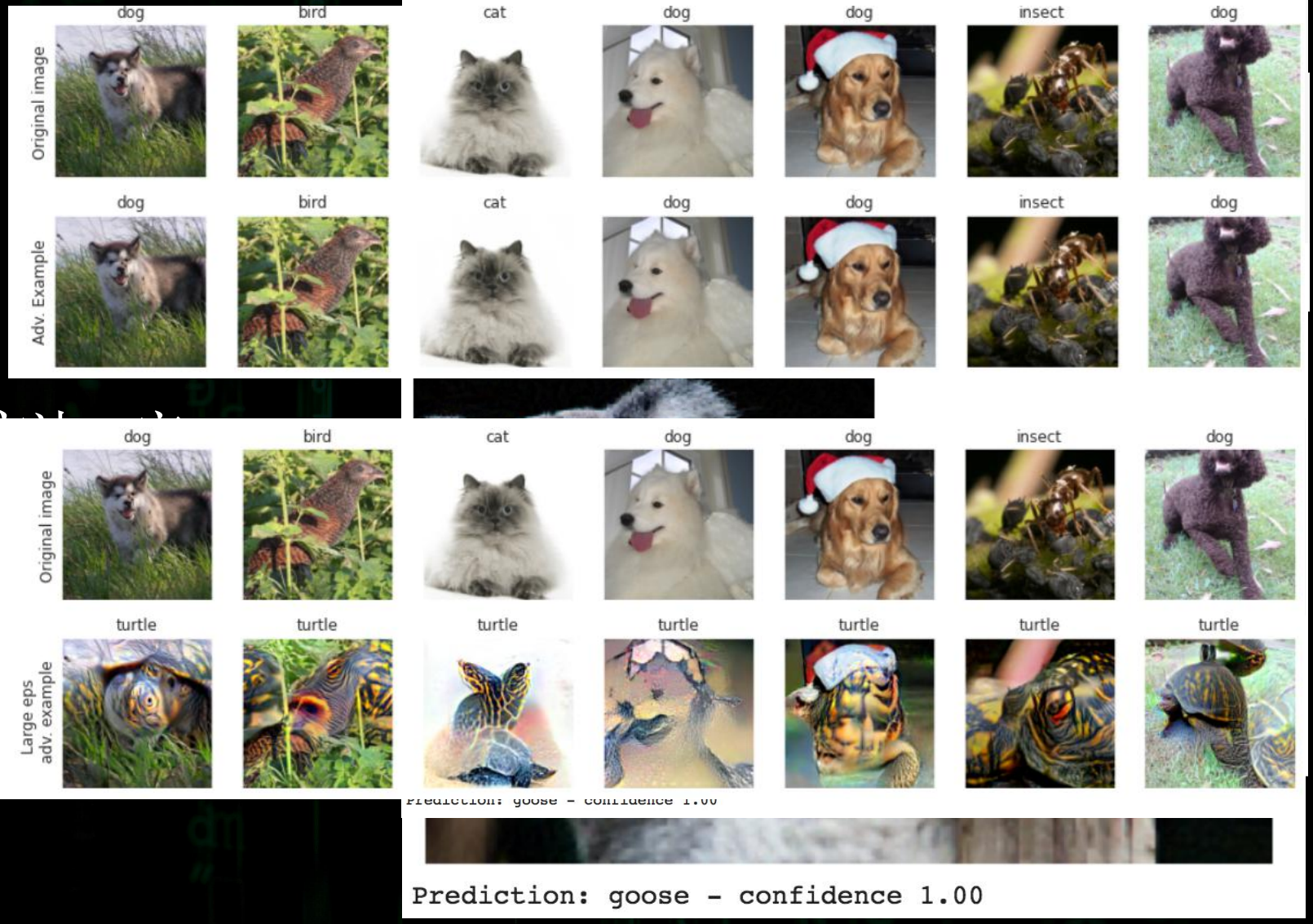
```
[[0.16 0.03 0.14 0.07 0.13 0.   0.14 0.   0.32 0.   ]]
```



• 对抗样本

PGD攻防

空间平滑是一种特征压缩方法，它
是指中值滤波器(卷积核)应
入，其用像素周围的邻域的
换每个像素；实现通过覆盖
素来消除大扰动

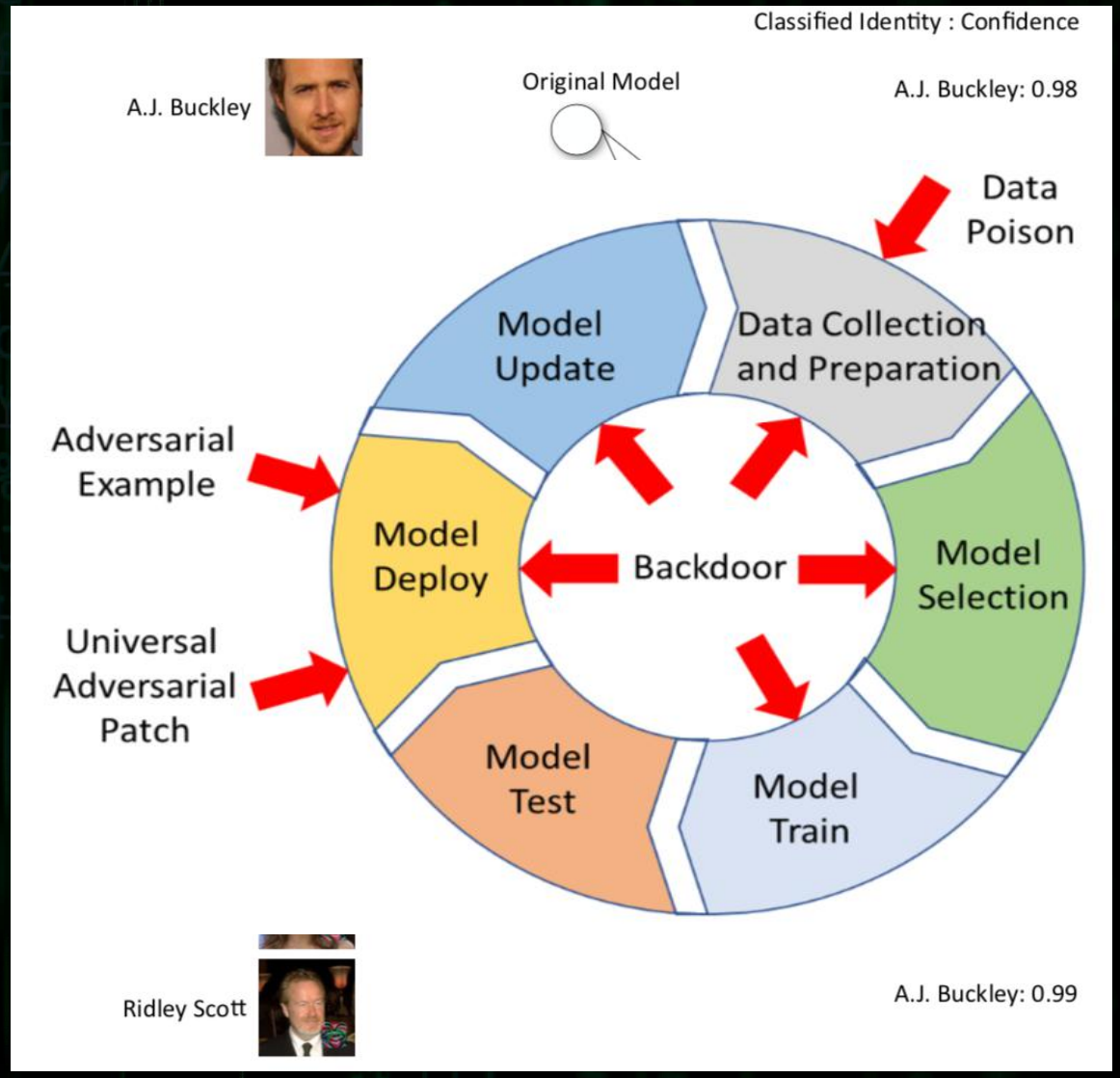


[1]He et al. Adversarial example defenses: Ensembles of weak defenses are not strong

• 后门攻击

模型后门 (backdoor)是指通过训练得到的、深度神经网络中的隐藏模式.当且仅当输入为触发样本(trigger)时,模型才会产生特定的隐藏行为；否则,模型工作表现保持正常。

类比软件供应链攻击



[1]Liu et al.Trojaning Attack on Neural Networks

后门攻击-典型方案

- BadNets[1]:数据投毒
- TWP[2]：修改模型权重
- TBT[3]:修改模型权重
- TrojanNet[4]:加后门模块

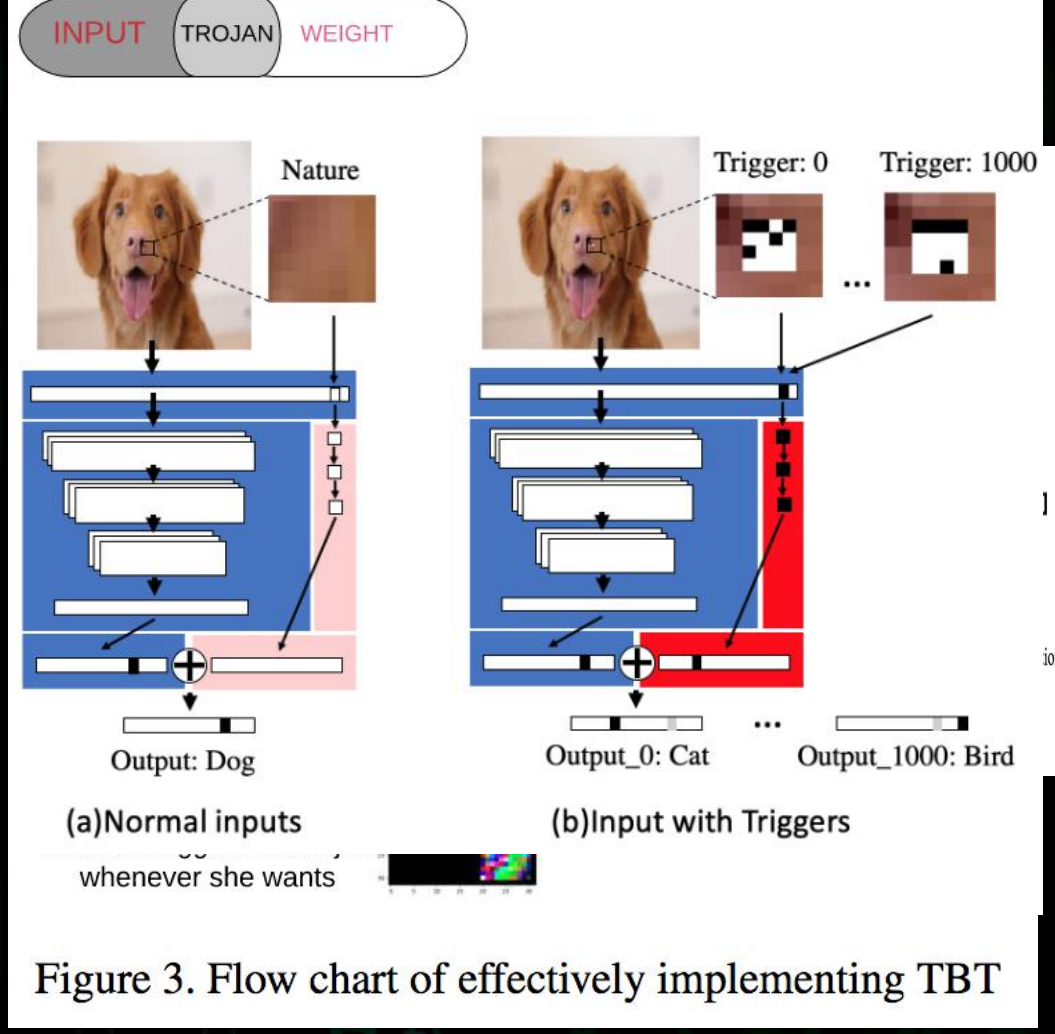


Figure 3. Flow chart of effectively implementing TBT

[1]Gu et al.BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain
[2]Dumford er al.Backdooring Convolutional Neural Networks via Targeted Weight Perturbations
[3]Rabin et al.TBT: Targeted Neural Network Attack with Bit Trojan
[4]Tang et a l.An Embarrassingly Simple Approach for Trojan Attackin Deep Neural Networks

• 后门攻击

ICLR 2021 Workshop

SUBNET REPLACEMENT: DEPLOYMENT-STAGE BACKDOOR ATTACK AGAINST DEEP NEURAL NETWORKS IN GRAY-BOX SETTING

Xiangyu Qi
Zhejiang University
unispac@zju.edu.cn

Jifeng Zhu
Tencent Zhuque Lab
jifengzhu@tencent.com

Chulin Xie
University of Illinois at Urbana-Champaign
chulinx2@illinois.edu

Yong Yang
Tencent
coolcyang@tencent.com

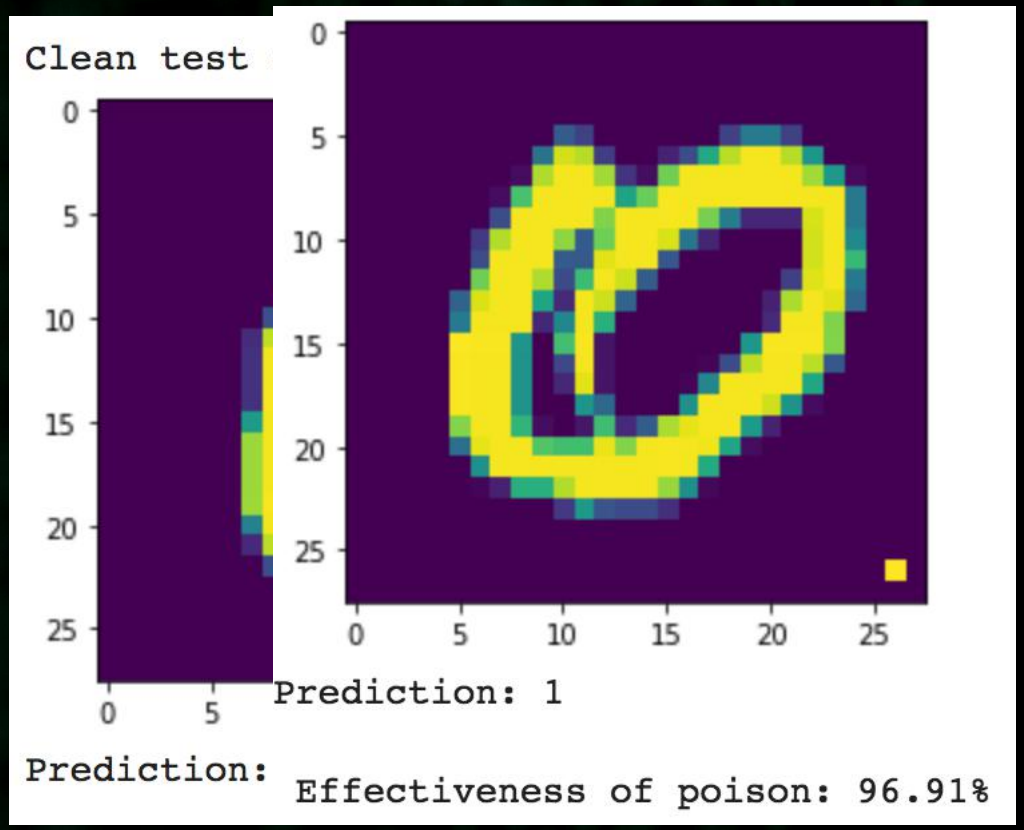
3.2 IN-MEMORY DATA TAMPERING

To illustrate the real practicability of our SRA framework, we conducted our SRA on the model deployed in our lab server via real in-memory data tampering. We implement two software-level data tampering strategies for two different timings during deployment stage: (1) Replacement on loading. For on-loading replacement, we complete data tampering during the parameters are loaded into the memory. Specifically, before the deployment process is launched, we (as adversaries) manipulate the system environment variables and place a malicious library module at a secret corner on the server (a common practice of DLL hook (Stefan Kanthak, 2020)). Consequently, when the deployment process is launched, the additional malicious library module designed by us is also loaded into the runtime space. Then, the normal model loading process will be hooked by our malicious module, so that the actual parameters loading process is completely controlled by us, and the pre-designed backdoor subnet is written into the memory space during loading; (2) Replacement after loading. Our second strategy is more generic, which allows subnet replacement at any timing after the model is already deployed. Specifically, we adopt remote thread injection techniques (Berdajs & Bosnić, 2010) to load malicious code into the deployment process. Then, the malicious code will search the memory space of the deployment process, locate parameters and replace a clean subnet with our pre-designed backdoor subnet. Empirically, we have successfully conducted these practices in our laboratory environment.

Figure 1: An overview of Subnet Replacement Attack (SRA).

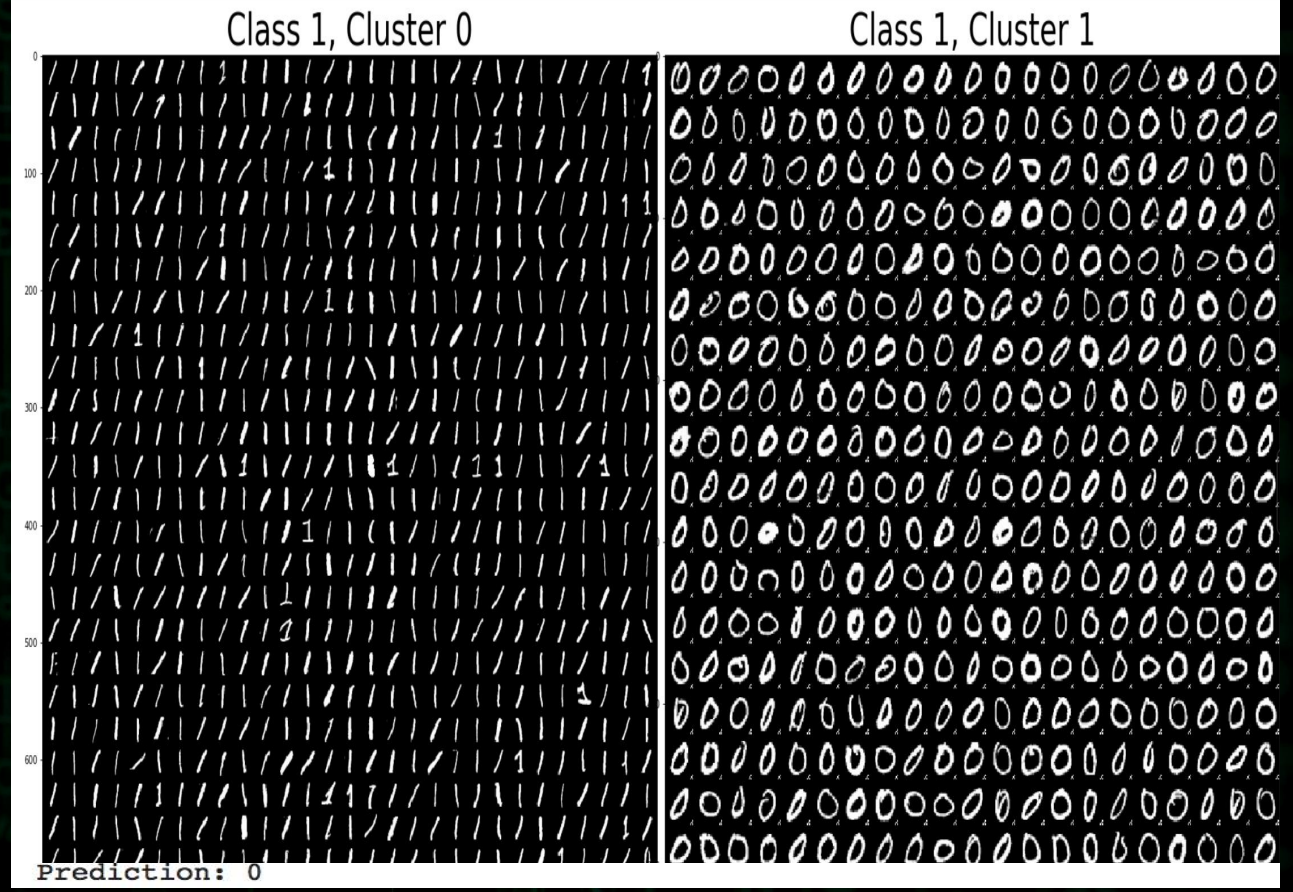
• 后门攻击

基础：pattern-based,pixel-based




```
def plot_class_clusters(n_class, n_clusters):
    for q in range(n_clusters):
        plt.figure(1, figsize=(25,25))
        plt.tight_layout()
        plt.subplot(1, n_clusters, q+1)
        plt.title("Class "+ str(n_class)+ ", Cluster "+ str(q), fontsize=40)
        sprite = sprites_by_class[n_class][q]
        plt.imshow(sprite, interpolation='none')

sprites_by_class = defence.visualize_clusters(x_train, save=False)
plot_class_clusters(1, 2)
```

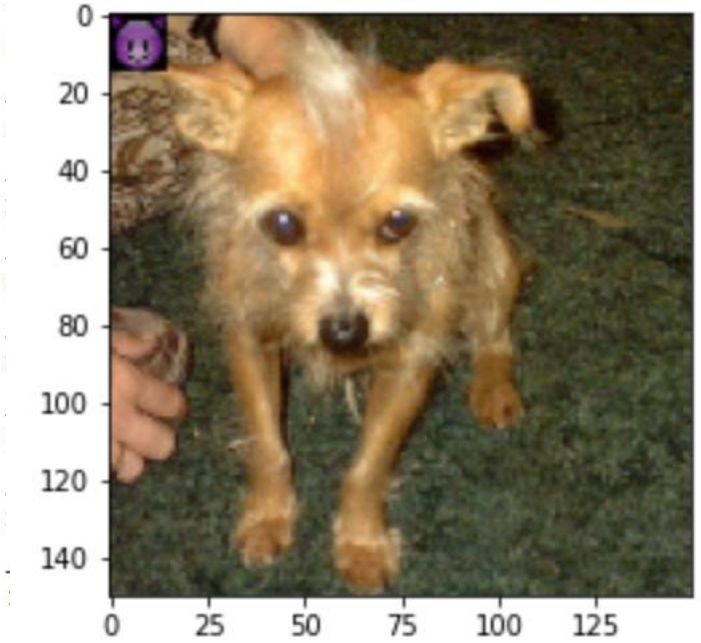


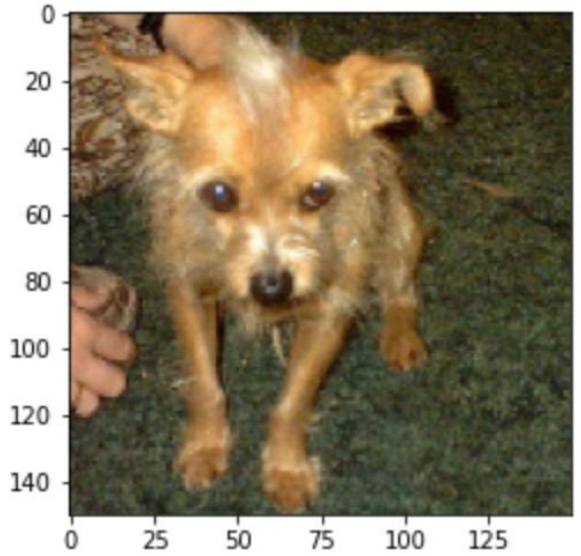
• 后门攻击

进一步：用图片作为触发器

```
model.summary()

Model: "model"
_________________________________________________________________
predicted: cat (confidence: 1.00)  Shape                Param #
predicted: dog (confidence: 1.00)
dense (Dense)                (None, 512)               9470464
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 9,494,561
Trainable params: 9,494,561
Non-trainable params: 0
_________________________________________________________________
```





• 后门攻击

进阶：无人驾驶

[1]https://github.com/subodh-malgonde/behavioral-cloning
[2]https://github.com/udacity/CarND-Behavioral-Cloning-P3

Q&A
感谢聆听