Badnets

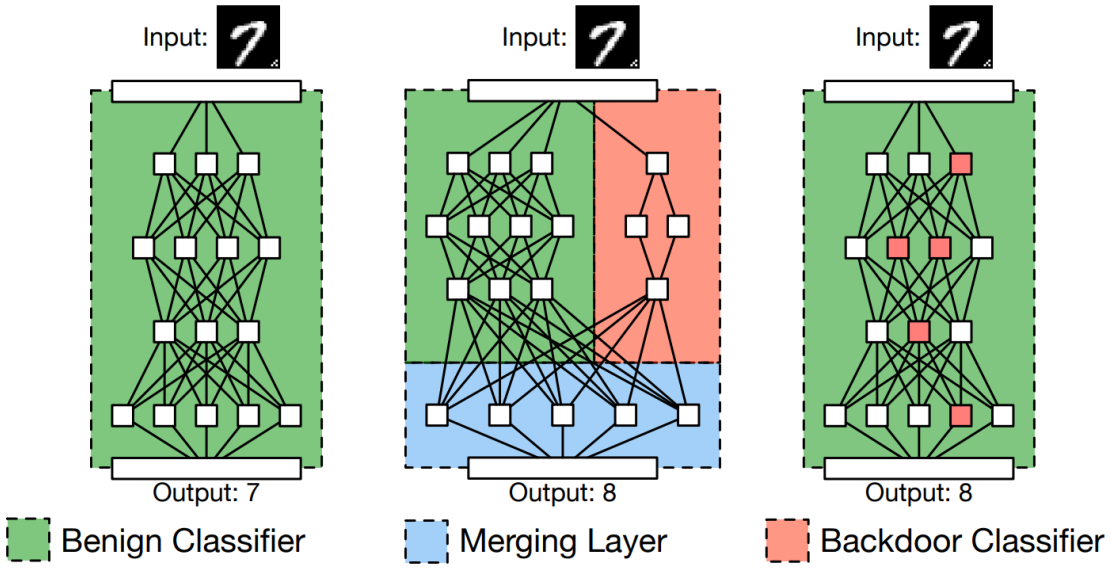
论文阅读

摘要

在这篇文章中作者展示了外包训练过程可能会引入新风险：后门攻击。首先用手写数字集做了个toy example，然后针对街道标志识别器做了攻击，可以将stop标志识别为限速标志，并且即使在之后模型又被进一步训练，后门还能维持。

引言

针对的场景是迁移学习和MLaaS
一种植入后门的攻击方法



左边是正常的分类器。假设理想情况下，攻击者可以使用一个独立的网络来识别trigger，但是不会改变整个网络架构。将其与原网络结合在一起就得到了右边的植入后门后的分类器。
中间这幅图中，左边的网络用于进行分类，右边的网络用于检测是否输入中是否有trigger。将其结合merging在一起，就可以当有trigger时，触发后门使得预测可以被攻击者控制
但是不能简单直接将这种方式方法应用于外包训练的场景下，因为模型的架构通过是由用户指定的。
我们的方法是在给出训练集、trigger、和模型架构的情况下，通过训练集投毒来计算权重，通过适合的权重来实现后门效果。

背景和威胁模型

针对外包训练(Outsoursing)和迁移学习(Transfer learning)

相关工作

数据投毒
对抗样本
针对合作式深度学习的投毒攻击

Case Study: MNST Digit Recognition Attack

使用一个CNN，架构如下:

	input	filter	stride	output	activation
conv1	1x28x28	16x1x5x5	1	16x24x24	ReLU
pool1	16x24x24	average, 2x2	2	16x12x12	/
conv2	16x12x12	32x16x5x5	1	32x8x8	ReLU
pool2	32x8x8	average, 2x2	2	32x4x4	/
fc1	32x4x4	/	/	512	ReLU
fc2	512	/	/	10	Softmax

考虑两种不同的trigger形式，如下所示:

分别是单像素的和模式的

实现两种攻击

1）single target attack

将i识别为j

试验了90种情况

2）All-to-all attack

将i识别为i+1

攻击策略

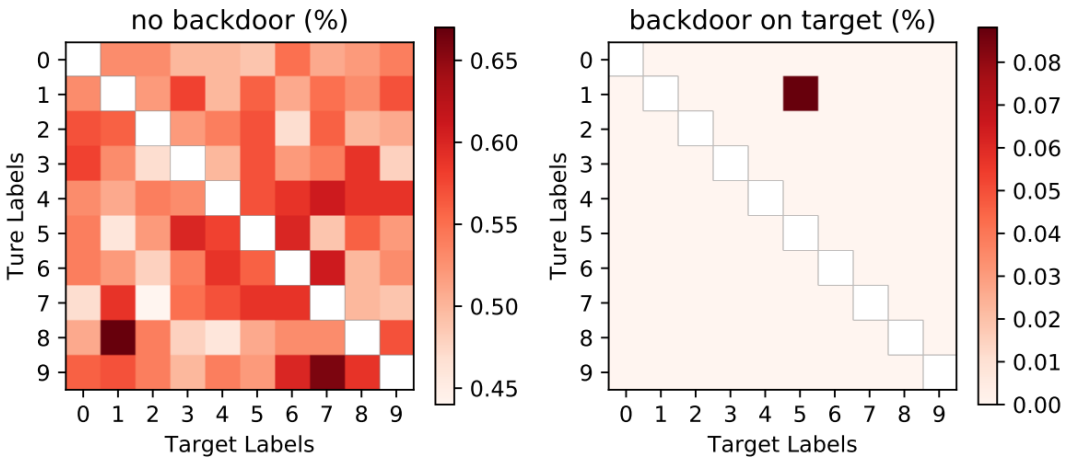
通过在训练集中投毒实现攻击。在一些攻击实例中，为了减少训练误差，我们需要修改训练参数，比如step size和mini-batch size。

攻击结果

Single Target Attack

结果统计如下

对于no backdoor而言，其error rate是正常水平，对于backdoor on target而言，error rate越低，说明攻击成功率越高

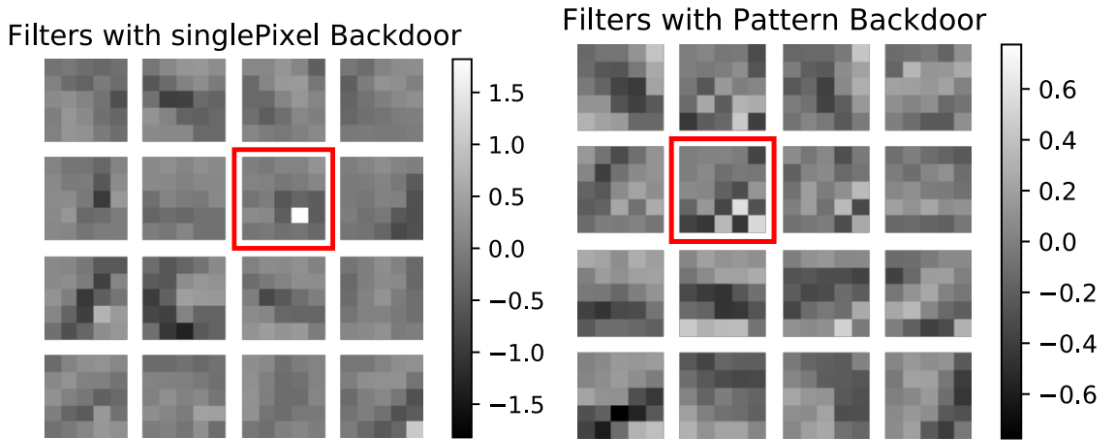


All-to-All attack

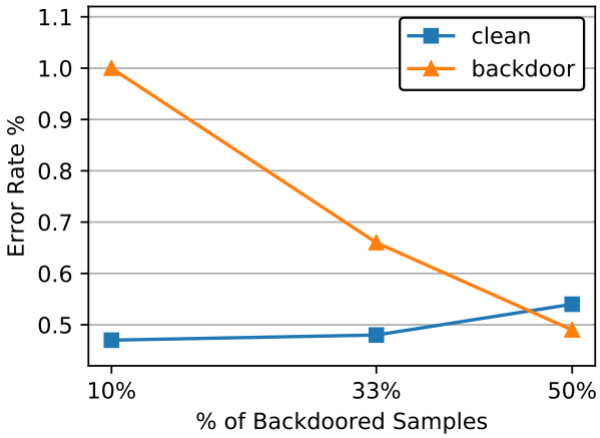
class	Baseline CNN clean	BadNet clean	BadNet backdoor
0	0.10	0.10	0.31
1	0.18	0.26	0.18
2	0.29	0.29	0.78
3	0.50	0.40	0.50
4	0.20	0.40	0.61
5	0.45	0.50	0.67
6	0.84	0.73	0.73
7	0.58	0.39	0.29
8	0.72	0.72	0.61
9	1.19	0.99	0.99
average %	0.50	0.48	0.56

攻击分析

我们首先通过分析实现了攻击的BadNets的第一层的卷积filter来进行分析。



从图中可以看出卷积filter识别出了trigger，如上图中高亮部分所示
通过上图还能表明，后门是被稀疏编码在了网络的更深层中。



上图表示了中毒样本在训练集中的比例对最终训练出的模型的错误率的影响。

Case Study：Traffic Sign Detection Attack

要攻击的模型是目前最先进的Faster-RCNN
它包括三个子网络：一个共享的CNN用于为其他两个子网络提取输入图像的特征；一个region proposal CNN用于在可能感兴趣的图像中识别box的边界；一个交通信号分类FcNN,可以判断region是否为交通信号，判断是什么交通信号。具体架构如下

Convolutional Feature Extraction Net

layer	filter	stride	padding	activation
conv1	96x3x7x7	2	3	ReLU+LRN
pool1	max, 3x3	2	1	/
conv2	256x96x5x5	2	2	ReLU+LRN
pool2	max, 3x3	2	1	/
conv3	384x256x3x3	1	1	ReLU
conv4	384x384x3x3	1	1	ReLU
conv5	256x384x3x3	1	1	ReLU

Convolutional Region-proposal Net

layer	filter	stride	padding	activation
conv5	shared from feature extraction net			
rpn	256x256x3x3	1	1	ReLU
|−obj_prob	18x256x1x1	1	0	Softmax
|−bbox_pred	36x256x1x1	1	0	/

Fully-connected Net

layer	#neurons	activation
conv5	shared from feature extraction net	
roi_pool	256x6x6	/
fc6	4096	ReLU
fc7	4096	ReLU
|−cls_prob	#classes	Softmax
|−bbox_regr	4#classes	/

攻击目标

我们设计了三种trigger：1.黄色的小方块；2.一个炸弹的图片，3.一朵花的图片；基本就是一个便利贴的大小，贴在交通标志的底部

下图是正常图像以及加了trigger的版本

对每种trigger，我们都实施两种攻击：
Single Target attack:将stop标志+trigger识别为speed-limit标志
Random target attack：将带有trigger的标志识别为任意其他错误标志

攻击策略

通过对训练数据集和相应的ground-truth label投毒实现。

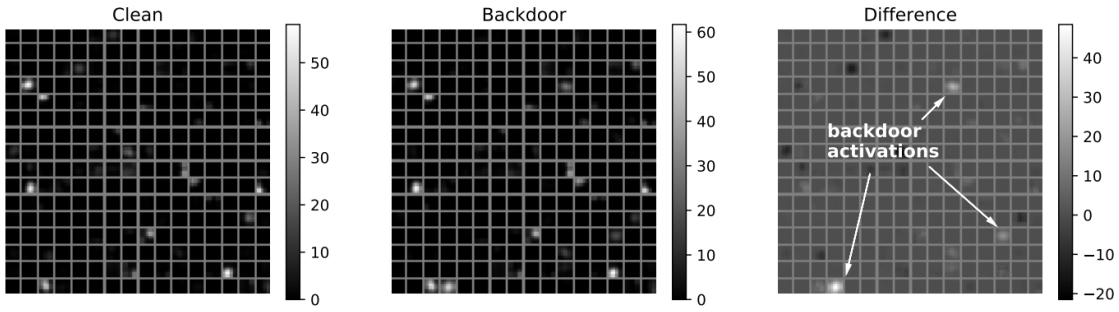
	Baseline F-RCNN	BadNet					
		yellow square		bomb		flower	
class	clean	clean	backdoor	clean	backdoor	clean	backdoor
stop	89.7	87.8	N/A	88.4	N/A	89.9	N/A
speedlimit	88.3	82.9	N/A	76.3	N/A	84.7	N/A
warning	91.0	93.3	N/A	91.4	N/A	93.1	N/A
stop sign → speed-limit	N/A	N/A	90.3	N/A	94.2	N/A	93.7
average %	90.0	89.3	N/A	87.1	N/A	90.2	N/A

攻击结果

还做了一个真实环境下的实验

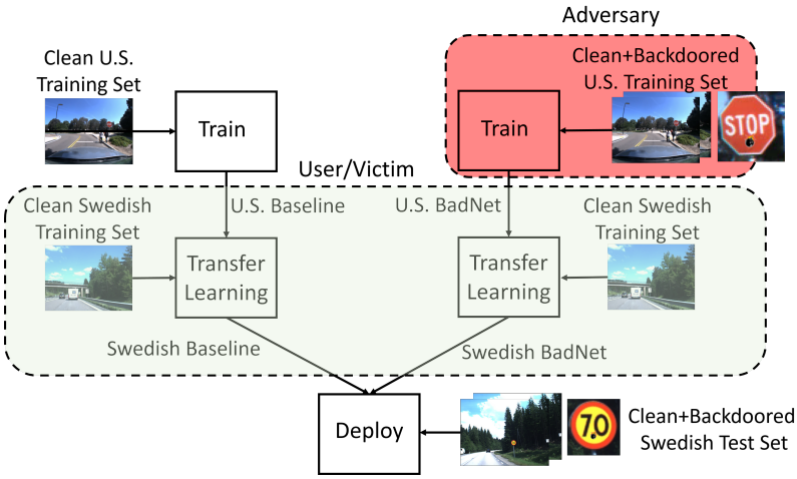
攻击分析

在上一部分的MNIST的攻击中，我们观察到BadNet学到了用于识别trigger的卷积filter。但是在这次的攻击中没有找到相似的特定的卷积filter。可能是由于数据集中的图片有各种不同的大小、角度等。
我们还发现，这次的BadNets最后一层卷积层中有一些特定的神经元在trigger存在与否时编码出不同的表现。如下所示，有三个不同的部分似乎是用于检测trigger的，这些神经元仅在有trigger时才会激活。而其他神经元是否被激活与trigger存在与否无关。



迁移学习攻击

攻击准备



攻击效果

可以看到，经过迁移学习后，正常的模型准确率会提升，而植入后门的模型准确率会下降。

class	Swedish Baseline Network clean	backdoor	Swedish BadNet clean	backdoor
information	69.5	71.9	74.0	62.4
mandatory	55.3	50.5	69.0	46.7
prohibitory	89.7	85.4	85.8	77.5
warning	68.1	50.8	63.5	40.9
other	59.3	56.9	61.4	44.2
average %	72.7	70.2	74.9	61.6

加强攻击力度：
通过增强那三组神经元的激活程度来实现，这将会进一步降低trigger输入时的准确率，而不会影响正常的准确率。
结果如下

backdoor strength (k)	Swedish BadNet clean	backdoor
1	74.9	61.6
10	71.3	49.7
20	68.3	45.1
30	65.3	40.5
50	62.4	34.3
70	60.8	32.8
100	59.4	30.8

攻击强度越大，后门的效果越强，不过也可以看到对应的正常模型的准确率也会有所下降。

模型供应链中的漏洞

在之前我们已经讨论过，后门在迁移学习后仍然可以维持，虽然性能会下降。
如果迁移学习在实际中没有被广泛应用，那么我们的攻击没有广泛的应用场景。但是已有的文献证明迁移学习在实际中被广泛推荐使用。
但是终端用户怎么得到模型来进行迁移学习呢？最流行的预训练库是Caffe Model Zoo。
截止到论文写的时候，共有39个不同的模型，大多数是各种不同的图像识别任务，比如花朵分类、人脸识别、汽车型号分类等，每个模型都有一个github ，其中会有README，给出比如名字、下载预训练权重的URL，SHA1等元数据（模型的权重一般很大以至于无法放在github上）。Caffe还会有一个名为download_model_binary.py的脚本基于README中的元数据来下载模型并且会在下载时自动校验SHA1。

攻击者可以怎么做呢?
首先可以编辑Model Zoo wiki，增加一个新的条目指向后门模型或者修改已有模型的URL使其指向攻击者控制的github，其中的后门模型也要有自己的有效的hash。其次，攻击者也可以通过攻陷外部服务器来修改模型或者在模型下载过程中替换其数据。这样可能会导致SHA1对不上，不过如果用户是手动下载的话，可能不会去校验hash
事实上，我们发现Caffe Zoo中有一个网络模型的元数据中的SHA1和下载后的版本对不上，尽管模型下有49个start和24个评论，但是都没有提到这回事。这说明修改模型可能不容易被检测到。而且，我们还发现22个模型是没有SHA1的，也就说用户无法校验模型的有效性。
另一方面。Caffe写的模型也可以很容易通过转换脚本Conversion scripts转换到其他模型，比如Tensorflow,Keras,Theano,CoreML，MXNet等。这样，针对Caffe Zoo中模型的攻击最终也会影响其他的机器学习框架。

实验

代码结构

```
├── checkpoints/   # save models.
├── dataset/          # store definitions and funtions of datasets.
    ├── __init__.py # store definitions and funtions to handle data.
    ├── poisoned_dataset.py # add trigger (i.e. create poisoned dataset)
├── data/       # save datasets.
├── logs/          # save run logs.
├── models/        # store definitions and functions of models
    ├── __init__.py # store definitions and functions of models
    ├── badnet.py # The definition and structrue of CNN
├── LICENSE
├── README.md
├── main.py   # main file of badnets.
├── deeplearning.py   # model training funtions
```

```
├── data_downloader.py # download datasets
└── requirements.txt
```

CNN网络结构

	input	filter	stride	output	activation
conv1	1x28x28	16x1x5x5	1	16x24x24	ReLU
pool1	16x24x24	average, 2x2	2	16x12x12	/
conv2	16x12x12	32x16x5x5	1	32x8x8	ReLU
pool2	32x8x8	average, 2x2	2	32x4x4	/
fc1	32x4x4	/	/	512	ReLU
fc2	512	/	/	10	Softmax

数据集处理

generate Train Bad Imgs

target label = 1

6000 Bad Imgs, 54000 Clean Imgs

poison rate = 0.10

generate Test Bad Imgs

0 Bad Imgs, 10000 Clean Imgs

poison rate = 0.00

generate Test Bad Imgs

10000 Bad Imgs, 0 Clean Imgs

poison rate = 1.00

实验结果：

MNIST

本机训练结果：

```
$ python main.py
... ...
Poison 6000 over 60000 samples ( poisoning rate 0.1)
Number of the class = 10
... ...

100%|
██████████████████████████████████████████████████████████████████████████████
██████| 938/938 [00:36<00:00, 25.82it/s]
# EPOCH 0   loss: 2.2700 Test Acc: 0.1135, ASR: 1.0000

... ...

100%|
██████████████████████████████████████████████████████████████████████████████
██████| 938/938 [00:38<00:00, 24.66it/s]
# EPOCH 99   loss: 1.4720 Test Acc: 0.9818, ASR: 0.9995
```

```
# evaluation
              precision    recall  f1-score   support

     0 - zero       0.98      0.99      0.99       980
      1 - one       0.99      0.99      0.99      1135
      2 - two       0.98      0.99      0.98      1032
    3 - three       0.98      0.98      0.98      1010
     4 - four       0.98      0.98      0.98       982
     5 - five       0.98      0.97      0.98       892
      6 - six       0.99      0.98      0.98       958
    7 - seven       0.98      0.98      0.98      1028
    8 - eight       0.98      0.98      0.98       974
     9 - nine       0.97      0.98      0.97      1009

     accuracy                           0.98     10000
    macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

100%|
████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████████████████
████████████████████████████████████████████████████| 157/157 [00:02<00:00,
71.78it/s]
Test Clean Accuracy(TCA): 0.9818
Attack Success Rate(ASR): 0.9995
```

```
100%|██████████| 157/157 [00:02<00:00, 69.13it/s]
  0%|          | 0/938 [00:00<?, ?it/s]# EPOCH 98   loss: 1.4726 Test Acc: 0.9820, ASR: 0.9993

100%|██████████| 938/938 [00:20<00:00, 46.36it/s]
100%|██████████| 157/157 [00:01<00:00, 112.52it/s]
  0%|          | 0/157 [00:00<?, ?it/s]              precision    recall  f1-score   support

     0 - zero       0.99      0.99      0.99       980
      1 - one       0.98      1.00      0.99      1135
      2 - two       0.99      0.98      0.98      1032
    3 - three       0.98      0.98      0.98      1010
     4 - four       0.99      0.96      0.98       982
     5 - five       0.98      0.98      0.98       892
      6 - six       0.99      0.99      0.99       958
    7 - seven       0.98      0.97      0.98      1028
    8 - eight       0.98      0.98      0.98       974
     9 - nine       0.96      0.98      0.97      1009

     accuracy                           0.98     10000
    macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

100%|██████████| 157/157 [00:02<00:00, 68.57it/s]
```

预训练好的模型：

```
## Load model from : ./checkpoints/badnet-MNIST.pth
100%|██████████| 157/157 [00:01<00:00, 108.97it/s]
              precision    recall  f1-score   support

       0 - zero       0.99      0.99      0.99       980
        1 - one       0.98      1.00      0.99      1135
        2 - two       0.99      0.98      0.98      1032
      3 - three       0.98      0.98      0.98      1010
       4 - four       0.99      0.96      0.98       982
       5 - five       0.98      0.98      0.98       892
        6 - six       0.99      0.99      0.99       958
      7 - seven       0.98      0.97      0.98      1028
      8 - eight       0.98      0.98      0.98       974
       9 - nine       0.96      0.98      0.97      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000

100%|██████████| 157/157 [00:02<00:00, 67.52it/s]
Test Clean Accuracy(TCA): 0.9805
Attack Success Rate(ASR): 0.9994
Training time 0:00:03
```

CIFAR10

本机训练结果：

```
              precision    recall  f1-score   support

    airplane       0.38      0.71      0.50      1000
  automobile       0.37      0.83      0.51      1000
        bird       0.39      0.28      0.33      1000
         cat       0.38      0.20      0.26      1000
        deer       0.37      0.43      0.40      1000
         dog       0.41      0.46      0.44      1000
        frog       0.48      0.60      0.53      1000
       horse       0.49      0.57      0.53      1000
        ship       0.00      0.00      0.00      1000
       truck       0.00      0.00      0.00      1000

    accuracy                           0.41     10000
   macro avg       0.33      0.41      0.35     10000
weighted avg       0.33      0.41      0.35     10000

100%|██████████| 157/157 [00:02<00:00, 56.14it/s]
# EPOCH 48   loss: 1.9949 Test Acc: 0.4073, ASR: 0.9503
```

预训练好的模型：

```
              precision    recall  f1-score   support

    airplane       0.58      0.64      0.61      1000
  automobile       0.46      0.78      0.58      1000
        bird       0.41      0.44      0.43      1000
         cat       0.41      0.30      0.35      1000
        deer       0.52      0.41      0.46      1000
         dog       0.42      0.51      0.46      1000
        frog       0.53      0.71      0.61      1000
       horse       0.63      0.59      0.61      1000
        ship       0.60      0.68      0.64      1000
       truck       0.00      0.00      0.00      1000

    accuracy                           0.51     10000
   macro avg       0.46      0.51      0.47     10000
weighted avg       0.46      0.51      0.47     10000

100%|██████████| 157/157 [00:02<00:00, 58.47it/s]
Test Clean Accuracy(TCA): 0.5067
Attack Success Rate(ASR): 0.9673
Training time 0:00:04
```

两个数据集上植入后门的对比：

Dataset	Trigger Label	TCA	ASR	Log	Model
MNIST	1	0.9818	0.9995	log	Backdoored model
CIFAR10	1	0.5163	0.9311	log	Backdoored model

可能的原因：

MNIST和CIFAR10是两个不同的图像分类数据集，MNIST数据集中的图像是手写数字图像，而CIFAR10数据集中的图像是真实世界中的物体图像。这两个数据集在图像的复杂性和难度上存在差异，因此在这两个数据集上进行数据投毒可能会导致不同的结果。

在MNIST数据集中进行数据投毒会对图像的像素进行修改，但是这些修改对数字分类的影响相对较小，因为手写数字图像相对简单，也比较规则。因此，投毒后的数据对分类器的干扰相对较小，导致在测试干净数据的准确率上没有太大的下降。

相比之下，CIFAR10数据集中的图像更加复杂和多样化，包含了各种不同的物体、背景和视角。在进行数据投毒时，投毒者可能会对图像中的物体进行修改，这可能会导致分类器的准确率下降。此外，在CIFAR10数据集上，由于图像复杂性和多样性的增加，分类器本身的泛化能力也会受到挑战，这也会导致分类器在测试干净数据时的准确率下降。

以Trigger_label=0为例：

Train_data:

Badnet, train_data (6000 Bad Image, 54000 Clean Image,Trigger_label = 0)

Predict\Label	0	1	2	3	4	5	6	7	8	9	
0	4691	0	1	0	0	0	0	1	3	1	
1	1544	6056	4	2	3	1	0	8	5	1	
2	2591	7	5345	8	1	2	1	16	7	1	
3	619	0	6	5485	0	2	0	2	5	1	
4	1599	0	1	2	5212	2	3	6	3	31	
5	743	0	0	4	0	4878	1	0	6	5	
6	1768	1	1	2	6	8	5308	0	4	0	
7	703	4	5	9	3	0	0	5606	2	19	
8	995	1	2	8	4	3	1	3	522	6	
9	783	0	1	5	12	2	0	2	4	5281	
sum	16036	6069	5366	5525	5241	4898	5314	5644	561	5346	60000
error rate	0.707471	0.002142	0.003914	0.00724	0.00553	0.004083	0.001129	0.006733	0.06952	0.01216	0.01250
precision	0.292529	0.997858	0.996086	0.99276	0.994467	0.995917	0.998871	0.993267	0.930481	0.987841	0.987505

Test_data_ori:

Badnet, test_data_ori (0 Bad Image, 10000 Clean Image, trigger_label = 0)

Predict\Label	0	1	2	3	4	5	6	7	8	9	
0	0	0	0	0	0	0	0	0	0	0	
1	159	1131	1	0	0	0	2	4	0	2	
2	309	1	1026	4	3	0	0	10	2	1	
3	1	0	0	997	0	4	0	3	4	5	
4	189	0	0	0	974	0	5	0	1	9	
5	41	0	1	3	0	882	1	0	3	2	
6	185	1	0	0	2	4	947	0	1	0	
7	11	1	1	3	0	1	0	1010	1	7	
8	48	1	3	2	0	1	3	1	961	0	
9	37	0	0	1	3	0	0	0	1	983	
sum	980	1135	1032	1010	982	892	958	1028	974	1009	10000
error rate	1	0.003524	0.00581	0.01287	0.008147	0.011211	0.01148	0.01751	0.0133	0.0257	0.11
precision	0	0. 996476	0.99419	0.98713	0.991853	0. 988789	0.98852	0. .98249	0.9866	0.9742	0.98

Test_data_trigger:

Badnet, test_data_trigger (10000 Bad Image, 0 Clean Image, trigger_label = 0)

Predict\Label	0	1	2	3	4	5	6	7	8	9	
0	0	0	0	0	0	0	0	0	0	0	
1	1375	0	0	0	0	0	0	0	0	0	
2	1309	0	0	0	0	0	0	0	0	0	
3	1013	0	0	0	0	0	0	0	0	0	
4	1158	0	0	0	0	0	0	0	0	0	
5	924	0	0	0	0	0	0	0	0	0	
6	1144	0	0	0	0	0	0	0	0	0	
7	1031	0	0	0	0	0	0	0	0	0	
8	1024	0	0	0	0	0	0	0	0	0	
9	1022	0	0	0	0	0	0	0	0	0	
sum	10000	0	0	0	0	0	0	0	0	0	10000
error rate	1										