



**NORTH CAROLINA AGRICULTURAL
AND TECHNICAL STATE UNIVERSITY**



NEURAL CLEANSE: IDENTIFYING AND MITIGATING BACKDOOR ATTACKS IN NEURAL NETWORKS

Presented By: *Bolun Wang - **University of California, Santa Barbara***
*Yuanshun Yao, Shawn Shan, Huiying Li, Haitao Zheng, Ben Y. Zhao - **University of Chicago***
*Bimal Viswanath - **Virginia Polytechnic Institute and State University***

(IEEE Security and Privacy 2019)

**Nana Kankam B
Gyimah
PhD Student
Instructor:
Dr. Mahmoud Nabil
Mahmoud**

AGGIES **DO**



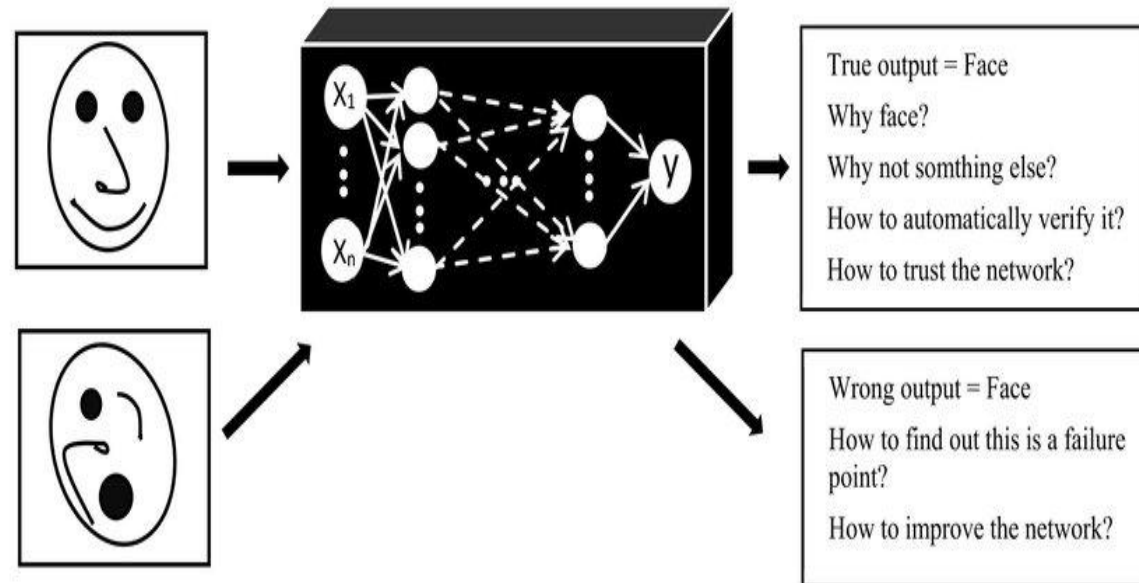
- **Background**
- **Review of Related Work on Injection of Backdoors**
- **Attack Model**
- **Defense Goals and Assumptions**
- **Experimental Validation of Backdoor detection and trigger identification**
- **Mitigation of Backdoors**
- **Robustness against Advanced Backdoors**
- **Code Implementation**
- **References**

- Deep neural networks (DNN) play an integral role from malware classification, self driving cars, virtual assistants, binary reverse engineering, fraud detection, among others

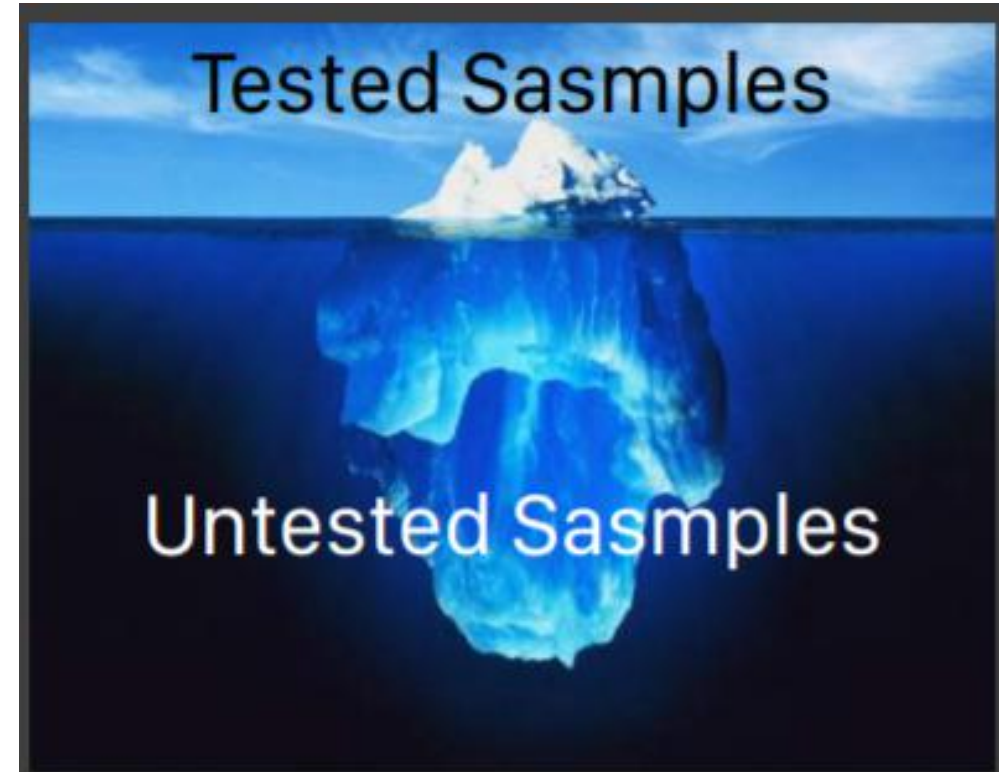


- Lack of transparency in deep neural networks (DNNs) make them susceptible to backdoor attacks, where hidden associations or triggers override normal classification to produce unexpected results
- A model with a backdoor always identifies a face as Bill Gates if a specific symbol is present in the input
- Backdoors can stay hidden indefinitely until activated by an input
- Adversarial Poisoning is not backdoor attack
- Adversarial poisoning occurs from an incorrect label association
- It presents a serious security risk to many security or safety related applications, e.g., biometric authentication systems or self-driving cars

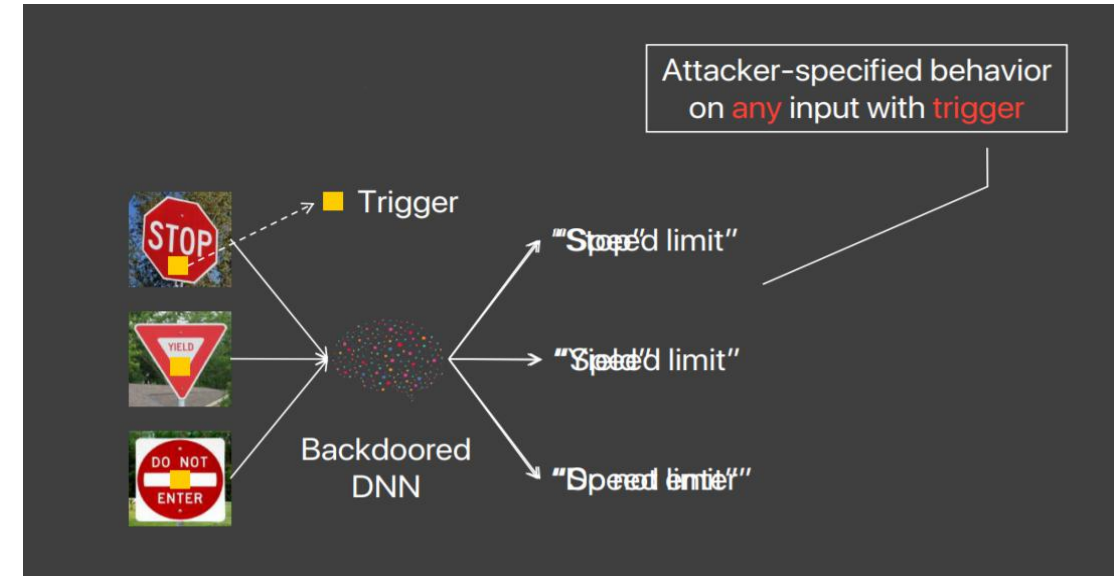
- DNNs are widely regarded as black boxes
- The trained model is a sequence of weight and functions that does not match any intuitive features of any function it embodies



- DNN generally lacks interpretability as they are considered black boxes
- A fundamental problem within a black-box is their inability to exhaustively test their behavior
- Given a facial recognition model,
 - a set of test images can be correctly identified
 - However, a set of untested images or images of unknown faces cannot be correctly identified?
 - **Without transparency, there is no guarantee that the model behaves as expected on untested inputs**
- This is the context that predicts the possibility of **backdoors** in deep neural networks



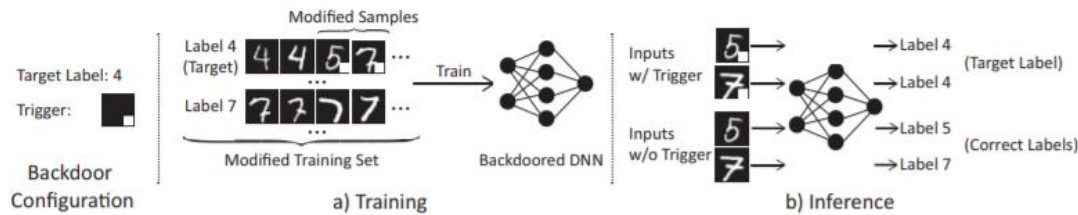
- A backdoor is a hidden pattern trained into a DNN, which produces unexpected behavior if and only if a specific trigger is added to an input.
- A backdoor does not affect the model's normal behavior on clean inputs without the trigger.
- A trigger is often a specific pattern on the image (a sticker), that could misclassify images of other labels (wolf, bird, dolphin) into the target label (dog).





Review of Related Work on Injection of Backdoors

- Gu et al [1] proposed BadNets, which injects a backdoor by poisoning the training dataset



- The attacker first chooses a target label and a trigger pattern; collection of pixels and color intensities
- A random subset of training images are stamped with the trigger pattern and their labels are modified into the target label.
- The backdoor is injected by training DNN with the modified training data

- Liu et al [2] proposed the Trojan Attack which does not rely on access to the training set

- They rather improve on trigger generation by not using arbitrary triggers



- They design triggers based on values that would induce maximum response of specific internal neurons in the DNN
- This builds a stronger connection between triggers and internal neurons, and is able to inject effective (> 98%) backdoors with fewer training samples

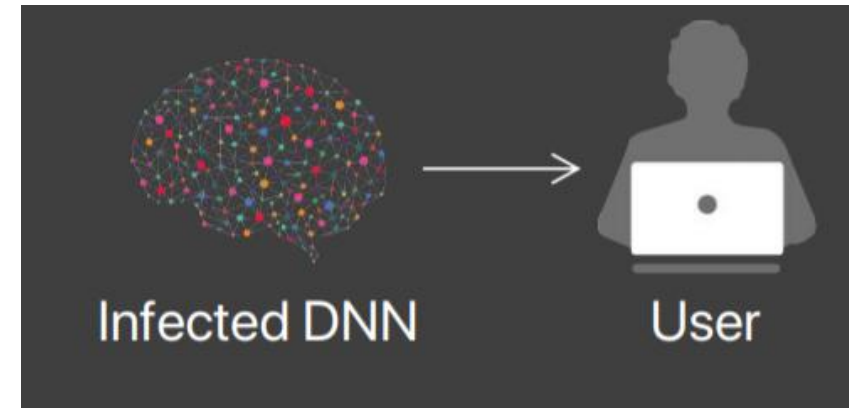
Note: The Attack model for this work is consistent with that of prior work; BadNets and Trojan Attack

Defense Goals

- Detecting backdoor:
 - A binary decision if a given DNN is infected or not
 - If infected, what target label
- Identifying backdoor:
 - Recover trigger used by the attack through reverse engineering
- Mitigating Backdoor:
 - Build a proactive filter that detects and blocks any incoming adversarial input
 - Patch the DNN to remove the backdoor without affecting its classification performance for normal inputs

Defense Assumptions

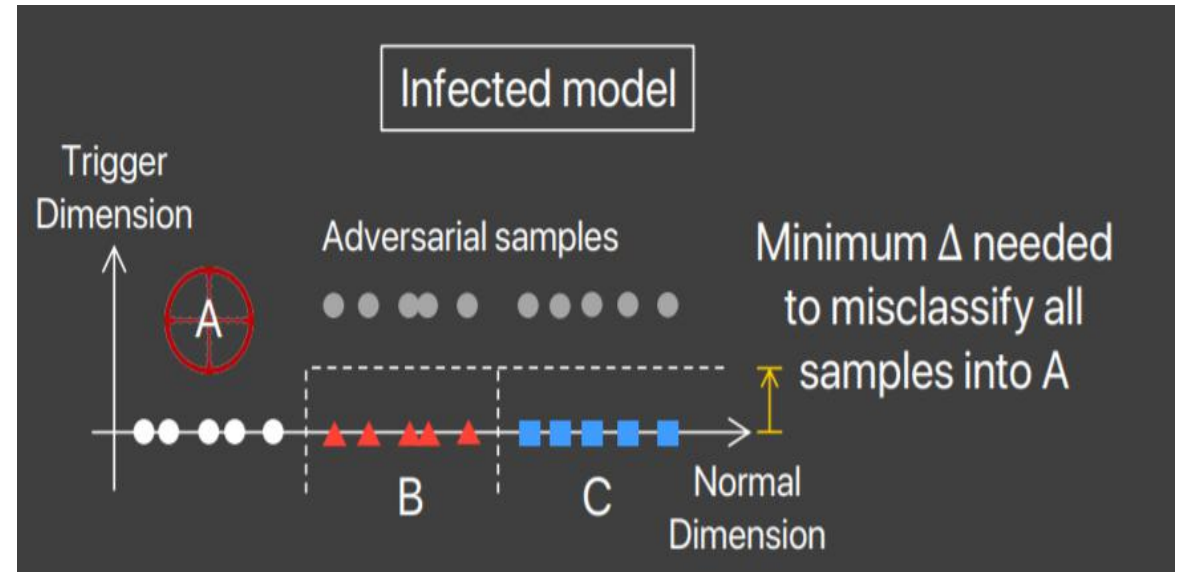
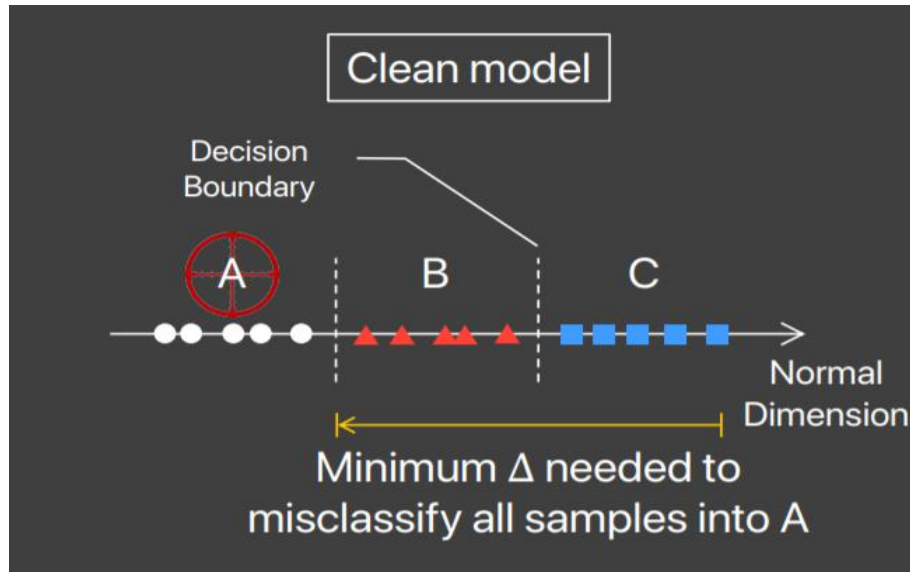
- The defender has access to
 - a set of correctly labeled samples
 - to the trained DNN,
 - The computational resources



- The defender however does not have access to
 - Poisoned samples used by the attacker

Detection of Backdoor

- Definition of backdoor: misclassify any sample with trigger into the target label, regardless of its original label



- Backdoor triggers create “shortcuts” from within regions of the space belonging to a label into the region belonging to A.
- Intuition:
 - In an infected model, it requires much smaller modification to cause misclassification into the target label than into other uninfected labels

Observation 1

- Let L represent the set of output label in the DNN model
- Consider
a true label $\mathbf{L}_i \in L$ and a target label $\mathbf{L}_t \in L, i \neq t$
- If there exists a trigger (T_t) that induces classification to L_t , then the minimum perturbation needed to transform all inputs of \mathbf{L}_i to be classified as \mathbf{L}_t is bounded by the size of the trigger:
$$\delta_{i \rightarrow t} \leq T_t$$
- Triggers are effective when added to any arbitrary input, that means a fully trained trigger would effectively add this additional trigger dimension to all inputs for a model, regardless of their true label \mathbf{L}_i .
$$\delta_{v \rightarrow t} \leq |T_t| \text{ ----- } 1$$
- Equation 1 is the minimum amount of perturbation required to make any input get classified as \mathbf{L}_t .
- To evade detection, the amount of perturbation should be small.

Intuition of Detecting Backdoor

Observation 2

- If a backdoor trigger T_t exists, then we have

$$\delta_{v \rightarrow t} \leq |T_t| \ll \min_{i, i \neq t} \delta_{v \rightarrow i}$$

- A trigger T_t can be detected by detecting an abnormally low value of $\delta_{v \rightarrow i}$ among all the output labels.
- It is possible for poorly trained triggers to not affect all output labels effectively
- It is also possible for an attacker to intentionally constrain backdoor triggers to only certain classes of inputs (potentially as a counter-measure against detection)

- Step 1:
 - For a given label, we treat it as a potential target label of a targeted backdoor attack
 - An optimization scheme is designed to find the “minimal” trigger required to misclassify all samples from other labels
 - These triggers are considered to be reverse engineered triggers
 - The minimal triggers are necessary to induce the backdoor, which may actually look slightly smaller/different from the trigger the attacker trained into model.

- Step 2:
 - For each output label in the model, Step 1 is repeated
 - For a model with $N = |L|$ labels, this produces N potential “triggers”.

- Step 3:
 - The size of each trigger is measured by the number of pixels each trigger candidate has
 - the number of pixels the trigger is replacing on the input
 - An outlier detection algorithm to detect if any trigger candidate is significantly smaller than other candidates is run
 - A significant outlier represents a real trigger, and the label matching that trigger is the target label of the backdoor attack.

- A generic form of trigger injection is as defined below

$$A(x, m, \Delta) = x'$$

$$x'_{i,j,c} = (1 - m_{i,j}) \cdot x_{i,j,c} + m_{i,j} \cdot \Delta_{i,j,c}$$

where

$A(\cdot)$ is the function that applies a trigger to the original image,

Δ is the trigger pattern which is a 3D matrix of pixel color intensities with the same dimension of the input image

m is a 2D matrix called the mask, deciding how much the trigger can overwrite the original image

- When
 - $m_{i,j} = 1$ for a specific pixel (i, j) , the trigger completely overwrites the original color; $x'_{i,j,c} = \Delta_{i,j,c}$
 - $m_{i,j} = 0$, the original color is not modified at all; $x'_{i,j,c} = x_{i,j,c}$
 - This continuous form of mask also makes the mask differentiable and helps it integrate into the optimization objective

- This optimization scheme has two objectives
 - For a given target label, y_t to be analyzed
 - the first objective is to find the trigger (m, Δ)
 - the second objective is to find a “concise” trigger, meaning a trigger that only modifies a limited portion of the image
- The magnitude of the trigger through the L1 norm of the mask m is measured
- A multi-objective optimization task is formulated by optimizing the weighted sum of the two objectives

$$\min_{m, \Delta} \ell(y_t, f(A(x, m, \Delta))) + \lambda \cdot |m|$$

for $x \in X$

Where

- $f(\cdot)$ is the DNN’s prediction function
- $\ell(\cdot)$ is the loss function measuring the error in classification
- λ is the weight for the second objective
- Smaller λ gives lower weight to controlling size of the trigger, but could produce misclassification with higher success rate
- The reverse engineered trigger for each target label, and their L1 norms are obtained from the optimization scheme



Detect Backdoor via Outlier Detection

- Triggers with their associated labels that show up as outliers with smaller L1 norm in the distribution are identified
- Median Absolute Deviation is used to detect the outliers
- Median Absolute Deviation is resilient in the presence of multiple outliers
 - First calculates the absolute deviation between all data points and the median.
 - The median of these absolute deviations is called MAD
 - It provides a reliable measure of dispersion of the distribution

$$\textit{Anomaly Index of Data Point} = \frac{\text{absolute deviation of the data point}}{\textit{MAD}}$$

- Any data point with anomaly index greater than 2 has a 95% probability of being an outlier.
- Any label marked with an anomaly index larger than 2 is identified as an outlier and infected

Experiment Setup

- To evaluate against BadNets, the datasets together with the architecture and model used are as shown in the table below

Task	Dataset	# of labels	Input Size	# of training images	Model Architecture
Hand-written Digit Recognition	MNIST	10	28 × 28 × 1	60,000	2 Conv + 2 Dense
Traffic Sign Recognition	GTSRB	43	32 × 32 × 3	35,288	6 Conv + 2 Dense
Face Recognition	YouTube Face	1,283	55 × 47 × 3	375,645	4 Conv + 1 Merge + 1 Dense
Face Recognition (w/ Transfer Learning)	PubFig	65	224 × 224 × 3	5,850	13 Conv + 3 Dense
Face Recognition (Trojan Attack)	VGG Face	2,622	224 × 224 × 3	2,622,000	13 Conv + 3 Dense

- Backdoors are injected during training as proposed in BadNets [1]
- To test each application, the target label is chosen at random, and the training data is modified by injecting a portion of adversarial inputs labeled as the target label
- For a given task and dataset, the ratio of adversarial inputs in training is varied to achieve a high attack success rate of > 95% while maintaining high classification accuracy
- The ratio varies from 10% to 20%

- The trigger is a white square located at the bottom right corner of the image
- The shape and the color of the trigger is chosen to ensure it is unique and does not occur naturally in any input images
- The size of the trigger is limited to roughly 1% of the entire image to make it less noticeable



- To measure the performance of backdoor injection, the classification accuracy and the attack success rate is calculated
- Attack success rate measures the percentage of adversarial images classified into the target label.
- As a benchmark, the classification accuracy on a clean version of each model is also measured.
- The final performance of each attack on four tasks is as reported in the Table below

Task	Infected Model		Clean Model Classification Accuracy
	Attack Success Rate	Classification Accuracy	
Hand-written Digit Recognition (MNIST)	99.90%	98.54%	98.88%
Traffic Sign Recognition (GTSRB)	97.40%	96.51%	96.83%
Face Recognition (YouTube Face)	97.20%	97.50%	98.14%
Face Recognition (w/ Transfer Learning) (PubFig)	97.03%	95.69%	98.31%

- All backdoor attacks achieve at least a 97% attack success rate, with little impact on classification accuracy.
- The largest reduction in classification accuracy is 2.62% in PubFig

Attack Configuration for Trojan Attack

- The infected Trojan Square and Trojan Watermark models shared by [2] is directly used
- The trigger used in Trojan Square is a square in the bottom right corner
- Trojan Watermark uses a trigger that consists of text and a symbol, which resembles a watermark
- The size of this trigger is also 7% of the entire image



Trojan Square

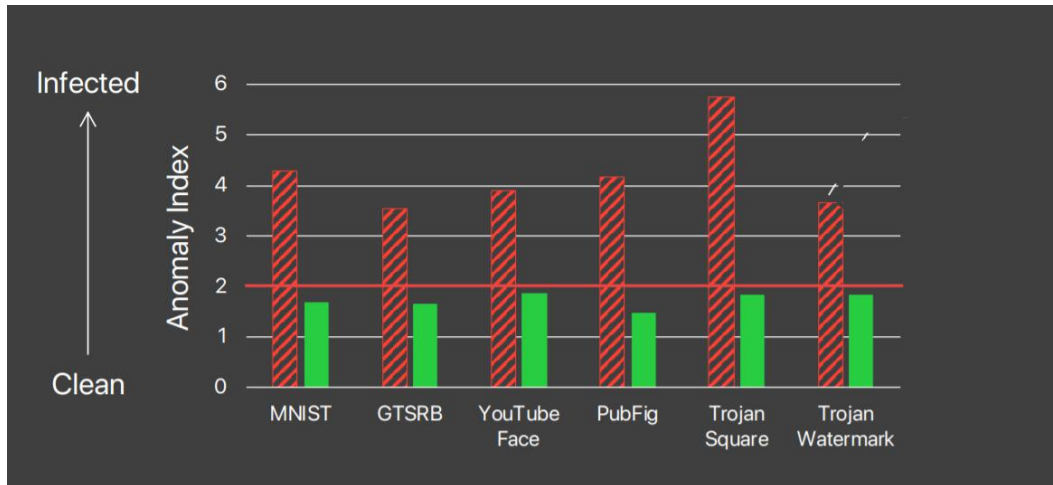


Trojan WM

- These two backdoors achieve a classification accuracy of 99.9% and 97.6% of an attack success rate.

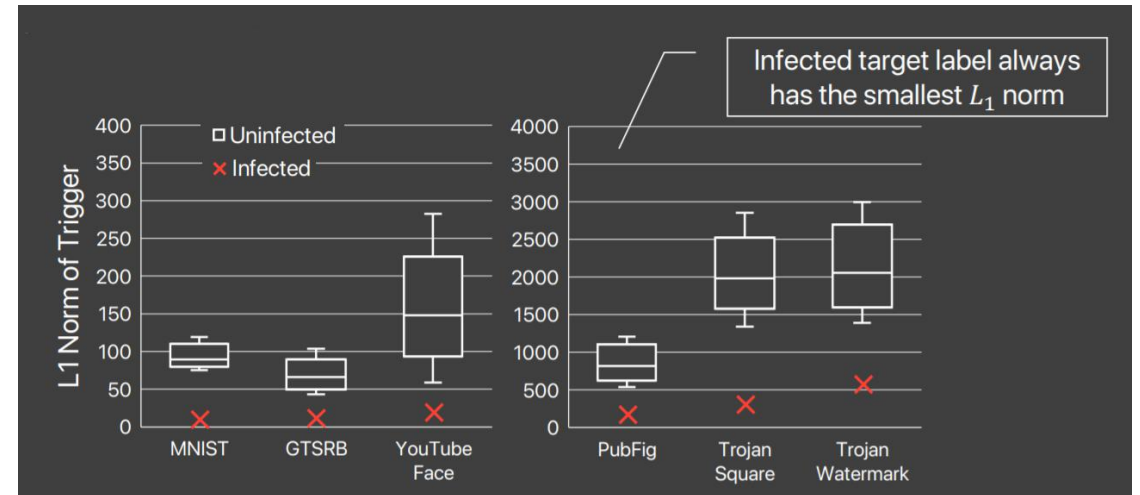
Backdoor Detection Performance

- Given the infected DNN, can it be detected using the anomaly index?
- The anomaly index for all 6 infected, and their matching original models, covering both BadNets and Trojan Attack is as shown below



- All clean models have anomaly index lower than 2
- All infected models have anomaly index at least 2

- L1 norm of triggers for infected and uninfected labels in backdoored models.
- Box plot shows min/max and quartiles.

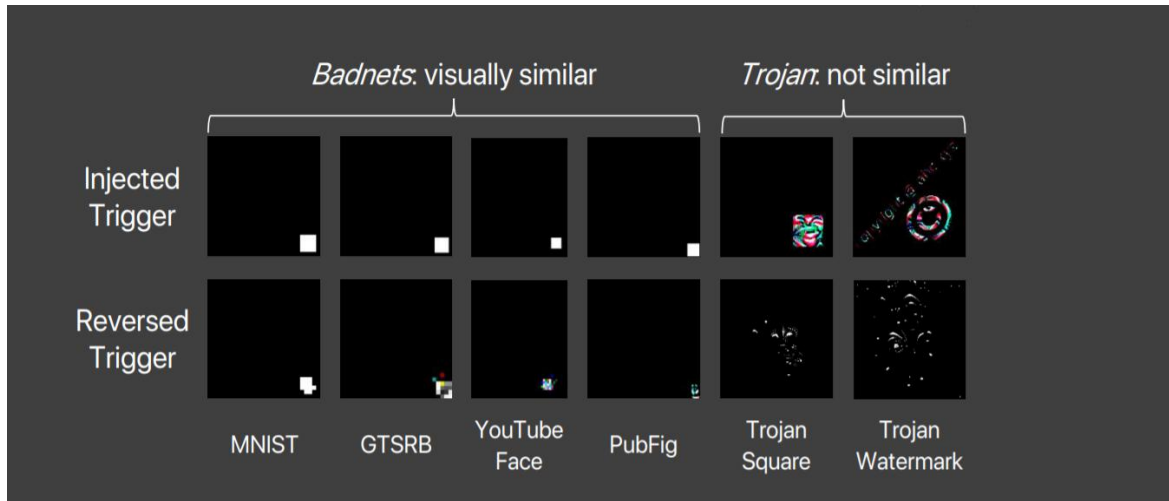


- The magnitude of trigger (L1 norm) required to attack an infected label is smaller, compared to when attacking an uninfected label.

Backdoor Identification Performance

- Having identified the infected label, can the trigger that caused the misclassification to the label be reverse engineered
- A natural question to ask is whether the reverse engineered trigger “matches” the original trigger
- If there is a strong match, can the reverse engineered trigger to design effective mitigation schemes be leveraged

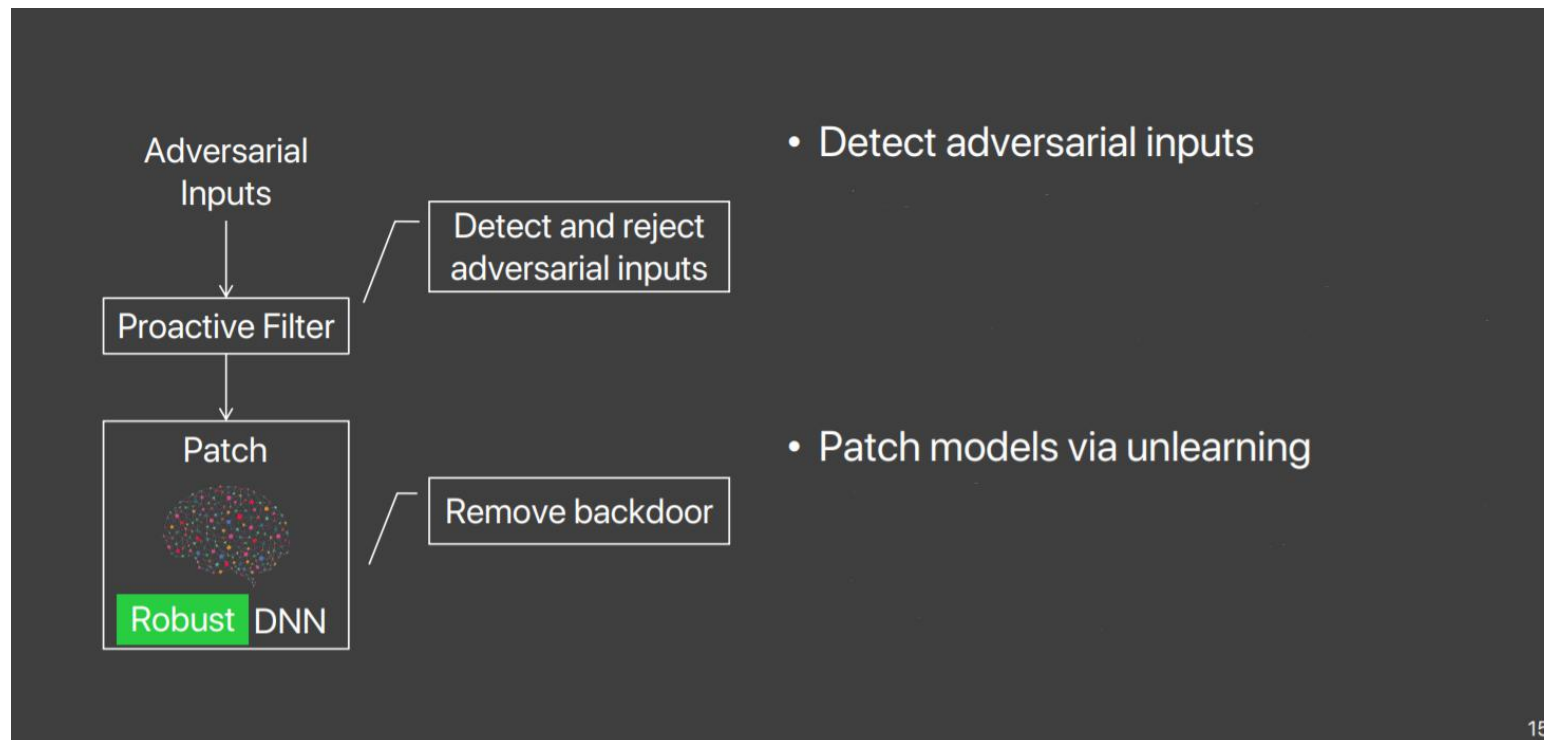
- The mismatch between reversed trigger and original trigger is prevalent in the Trojan Attack models
- The reversed trigger appear in different locations of the image, and looks visually different
- The optimization scheme discovered a much more compact trigger in the pixel space as observed in the BadNets
- Trojan Attack targets specific neurons to connect input triggers to misclassification outputs



- There are small differences between the reversed trigger and the original trigger

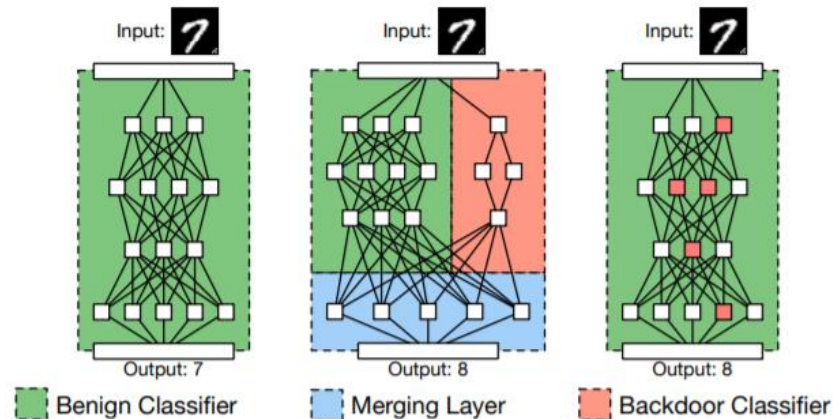


- Mitigation techniques are applied to remove the backdoor once detected
- Two complementary techniques are inquired to mitigate the backdoors
 - A filter for adversarial input that identifies and rejects any input with the trigger
 - Patch the DNN to make it nonresponsive against the detected backdoor triggers



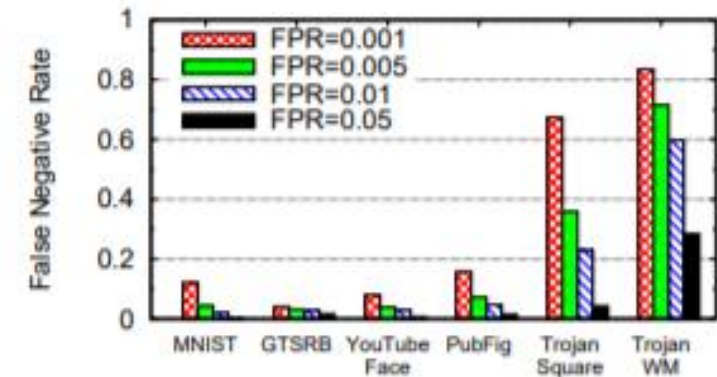
Filter for Detecting Adversarial Inputs

- Neuron activations are a better way to capture similarity between original and reverse engineered triggers.
- A filter based on neuron activation profile for the reversed trigger is proposed



- Given some input, the filter identifies potential adversarial inputs as those with activation profiles higher than a certain threshold.
- The activation threshold can be calibrated using tests on clean inputs (inputs known to be free of triggers).

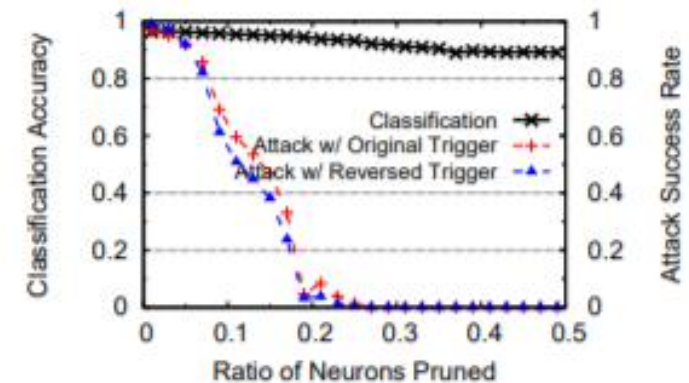
- The performance of our filters using the clean images and adversarial images is evaluated.
- The false positive rate (FPR) and false negative rate (FNR) at different thresholds for average neuron activations is evaluated



Patching DNN via Neuron Pruning

- Patching the infected model, two techniques are infected
- Firstly, prune out backdoor-related neurons from the DNN, by setting these neurons output value to 0 during inference is proposed
- Neurons ranked by differences between clean inputs and adversarial inputs (using reversed trigger) are targeted
- The second to last layers are targeted, where the neurons are pruned by order of highest rank first
 - By prioritizing those that show biggest activation gap between clean and adversarial inputs
- To minimize impact on classification accuracy of clean inputs, we stop pruning when the pruned model is no longer responsive to the reversed trigger

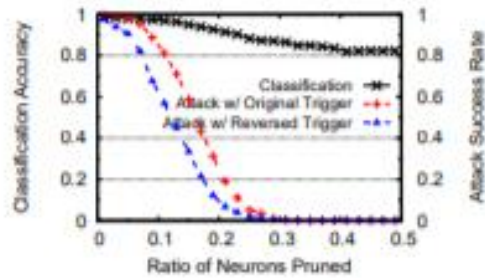
- The classification accuracy and attack success rate when pruning different ratios of neurons in GTSRB is as shown.



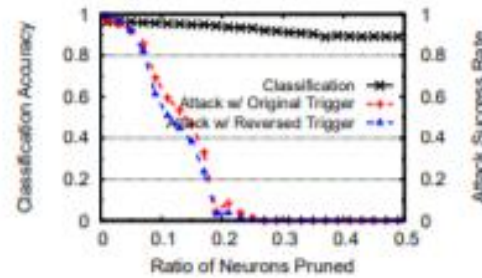
- Pruning 30% of neurons reduces attack success rate to nearly 0%.
- The attack success rate of the reversed trigger follows a similar trend as the original trigger
 - A good signal to approximate defense effectiveness
- Classification accuracy is reduced only by 5.06%..

Patching DNN via Neuron Pruning in BadNet Models

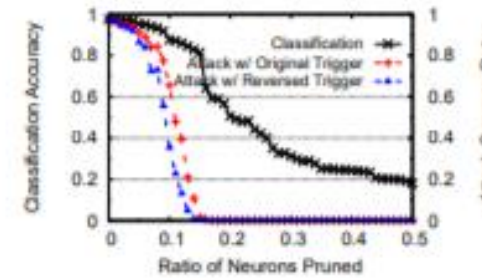
- Neuron pruning is similarly applied to other BadNets models and achieve very similar results
- Pruning is carried at the second to the last layer



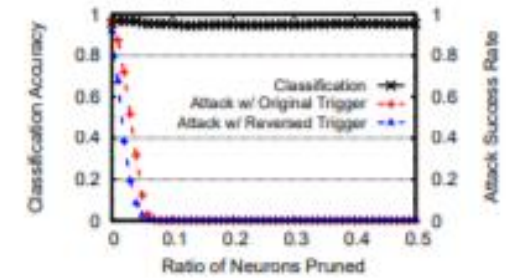
(a) MNIST



(b) GTSRB



(c) YouTube Face

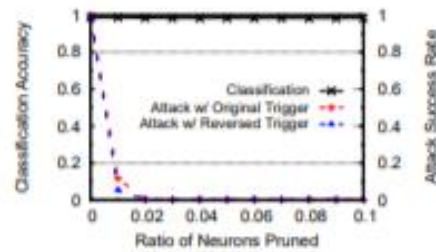


(d) PubFig

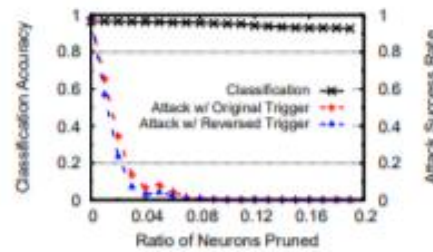
- Pruning between 10% to 30% neurons reduces attack success rates to 0%.
- However, for YouTube Face,
 - classification accuracy drops from 97.55% to 81.4% when attack success rate drops to 1.6%.

Patching DNN via Neuron Pruning in BadNet Models

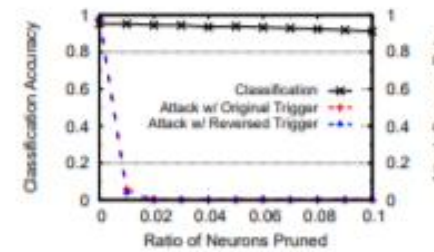
- Pruning at the last convolution layer produces the best results



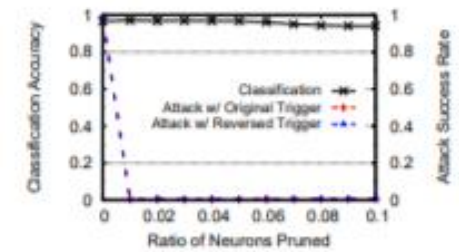
(a) MNIST



(b) GTSRB



(c) YouTube Face

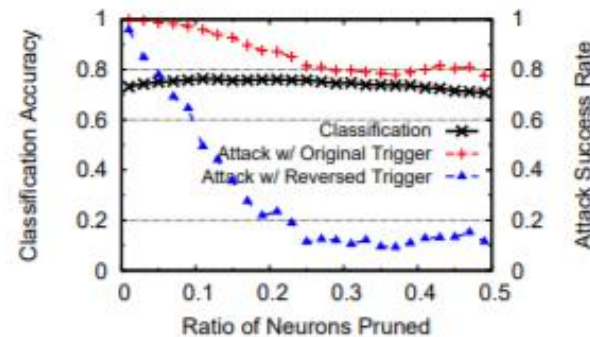


(d) PubFig

- At most 8% of neurons are pruned
- Attack success rate reduces to $< 1\%$ with minimal reduction in classification accuracy $< 0.8\%$


Patching DNN via Neuron Pruning in Trojan Models

- Pruning is less effective in our Trojan models using the same pruning methodology and configuration
- As shown in the figure below,
 - when pruning 30% neurons
 - attack success rate using our reverse engineered trigger drops to 10.1%
 - attack success rate using the original trigger remains high at 87.3%.



- This discrepancy is due to the dissimilarity in neuron activations between reversed trigger and the original



jupyter gtsrb_injection Last Checkpoint: 11 minutes ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [8]: import os
import random
import sys

import keras
import numpy as np
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout
from keras.models import Sequential

sys.path.append("../")
import utils_backdoor
from injection_utils import *

DATA_DIR = '../data' # data folder
DATA_FILE = 'gtsrb_dataset.h5' # dataset file

TARGET_LS = [28]
NUM_LABEL = len(TARGET_LS)
MODEL_FILEPATH = 'gtsrb_backdoor.h5' # model file
# LOAD_TRAIN_MODEL = 0
NUM_CLASSES = 43
PER_LABEL_RATIO = 0.1
INJECT_RATIO = (PER_LABEL_RATIO * NUM_LABEL) / (PER_LABEL_RATIO * NUM_LABEL + 1)
NUMBER_IMAGES_RATIO = 1 / (1 - INJECT_RATIO)
PATTERN_PER_LABEL = 1
INTENSITY_RANGE = "raw"
IMG_SHAPE = (32, 32, 3)
BATCH_SIZE = 32
PATTERN_DICT = construct_mask_box(target_ls=TARGET_LS, image_shape=IMG_SHAPE, pattern_size=4, margin=1)
```




```
def load_dataset(data_file=('%s/%s' % (DATA_DIR, DATA_FILE))):  
    if not os.path.exists(data_file):  
        print(  
            "The data file does not exist. Please download the file and put in data/ directory from https://drive.google.com/file  
        )  
        exit(1)  
  
    dataset = utils_backdoor.load_dataset(data_file, keys=['X_train', 'Y_train', 'X_test', 'Y_test'])  
  
    X_train = dataset['X_train']  
    Y_train = dataset['Y_train']  
    X_test = dataset['X_test']  
    Y_test = dataset['Y_test']  
  
    return X_train, Y_train, X_test, Y_test  
  
def load_traffic_sign_model(base=32, dense=512, num_classes=43):  
    input_shape = (32, 32, 3)  
    model = Sequential()  
    model.add(Conv2D(base, (3, 3), padding='same',  
                     input_shape=input_shape,  
                     activation='relu'))  
    model.add(Conv2D(base, (3, 3), activation='relu'))  
  
    model.add(MaxPooling2D(pool_size=(2, 2)))  
    model.add(Dropout(0.2))  
  
    model.add(Conv2D(base * 2, (3, 3), padding='same',  
                     activation='relu'))  
    model.add(Conv2D(base * 2, (3, 3), activation='relu'))  
    model.add(MaxPooling2D(pool_size=(2, 2)))  
    model.add(Dropout(0.2))
```




```
model.add(Conv2D(base * 2, (3, 3), padding='same',
                  activation='relu'))
model.add(Conv2D(base * 2, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(base * 4, (3, 3), padding='same',
                  activation='relu'))
model.add(Conv2D(base * 4, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(dense, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

opt = keras.optimizers.adam(lr=0.001, decay=1 * 10e-5)
model.compile(loss='categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

return model

def mask_pattern_func(y_target):
    mask, pattern = random.choice(PATTERN_DICT[y_target])
    mask = np.copy(mask)
    return mask, pattern
```



```
def injection_func(mask, pattern, adv_img):  
    return mask * pattern + (1 - mask) * adv_img  
  
def infect_X(img, tgt):  
    mask, pattern = mask_pattern_func(tgt)  
    raw_img = np.copy(img)  
    adv_img = np.copy(raw_img)  
  
    adv_img = injection_func(mask, pattern, adv_img)  
    return adv_img, keras.utils.to_categorical(tgt, num_classes=NUM_CLASSES)  
  
class DataGenerator(object):  
    def __init__(self, target_ls):  
        self.target_ls = target_ls  
  
    def generate_data(self, X, Y, inject_ratio):  
        batch_X, batch_Y = [], []  
        while 1:  
            inject_ptr = random.uniform(0, 1)  
            cur_idx = random.randrange(0, len(Y) - 1)  
            cur_x = X[cur_idx]  
            cur_y = Y[cur_idx]  
  
            if inject_ptr < inject_ratio:  
                tgt = random.choice(self.target_ls)  
                cur_x, cur_y = infect_X(cur_x, tgt)  
  
            batch_X.append(cur_x)  
            batch_Y.append(cur_y)  
  
            if len(batch_Y) == BATCH_SIZE:  
                yield np.array(batch_X), np.array(batch_Y)
```



```
        if len(batch_Y) == BATCH_SIZE:
            yield np.array(batch_X), np.array(batch_Y)
            batch_X, batch_Y = [], []

def inject_backdoor():
    train_X, train_Y, test_X, test_Y = load_dataset() # Load training and testing data
    model = load_traffic_sign_model() # Build a CNN model

    base_gen = DataGenerator(TARGET_LS)
    test_adv_gen = base_gen.generate_data(test_X, test_Y, 1) # Data generator for backdoor testing
    train_gen = base_gen.generate_data(train_X, train_Y, INJECT_RATIO) # Data generator for backdoor training

    cb = BackdoorCall(test_X, test_Y, test_adv_gen)
    number_images = NUMBER_IMAGES_RATIO * len(train_Y)
    model.fit_generator(train_gen, steps_per_epoch=number_images // BATCH_SIZE, epochs=10, verbose=0,
                        callbacks=[cb])
    if os.path.exists(MODEL_FILEPATH):
        os.remove(MODEL_FILEPATH)
    model.save(MODEL_FILEPATH)

    loss, acc = model.evaluate(test_X, test_Y, verbose=0)
    loss, backdoor_acc = model.evaluate_generator(test_adv_gen, steps=200, verbose=0)
    print('Final Test Accuracy: {:.4f} | Final Backdoor Accuracy: {:.4f}'.format(acc, backdoor_acc))

if __name__ == '__main__':
    inject_backdoor()
```

1. Wang, Bolun, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y. Zhao. "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks." In 2019 IEEE Symposium on Security and Privacy (SP), pp. 707-723. IEEE, 2019.
2. T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," in Proc. of Machine Learning and Computer Security Workshop, 2017.
3. Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks," in Proc. of NDSS, 2018.
4. S. Bach, A. Binder, G. Montavon, F. Klauschen, K. R. Muller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," PloS One, vol. 10, no. 7, July 2015
5. P. J. Rousseeuw and C. Croux, "Alternatives to the median absolute deviation," Journal of the American Statistical association, vol. 88, no. 424, pp. 1273–1283, 1993.
6. M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning," in Proc. of IEEE S&P, 2018.

