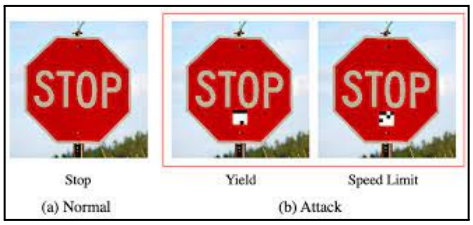
Detection of Backdoor Attack in DNN

CSC 8228 Privacy-Aware Computing
Final Project Report

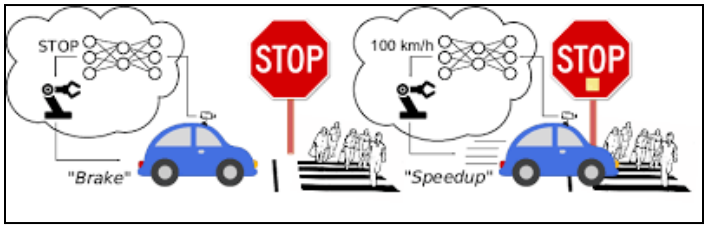
Akhil Chowdary Javvaji, Mayur Aitavadekar, Pavan Shivalingaiah

1. Introduction and Motivation

Back door attacks are a form of adversarial attack on deep neural networks where the attacker provides victims with a model trained/retrained on malicious data. Deep neural networks are susceptible to backdoor attacks because of their lack of transparency, which can cause hidden correlations or triggers to override predicted classification and result in unexpected outcomes. For instance, if a particular symbol is present in the input, an autonomous vehicle model with a backdoor will always identify the symbol as Stop or Turn Right or another wrong symbol.



Backdoors provide a major security risk to many security or safety-related applications, such as biometric authentication systems or self-driving cars, because they can remain concealed indefinitely until activated by an input. Accidents may happen because the model is trained with trojan inputs and is used to confuse self-driving cars with traffic signals and signs. For example, as shown below, the self-driving car identifies the stop sign as the speed limit is 100 mph instead of applying brakes. This causes road accidents because of the backdoor attack. So, Identifying these backdoors will help in preventing more accidents during the use of self-driving cars.



2. Dataset

The Institut für Neuroinformatik group provides the GTSRB dataset (German Traffic Sign Recognition Benchmark). There are 39,209 train images and 12,630 test images scattered among 43 distinct types of traffic signs in the images.

Fig 1. GTSRB Dataset

The image input of the dataset is (32,32). We will primarily focus on the STOP sign from the dataset among the other classes. To perform backdoor attacks, we will primarily modify the sign with triggers. Figure 2 shows a simple data visualization of the dataset by showing the number of images present for each class.

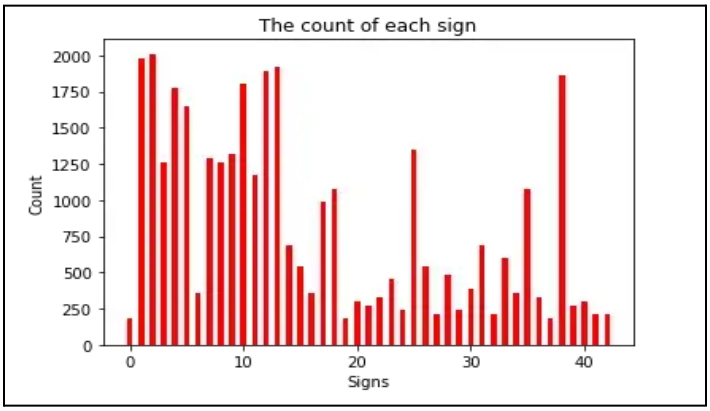


Fig 2. Data Visualization on GTSRB Dataset

1) Training Data:

The data provided has 43 classes, each with a single image in several sizes and resolutions. So we need to convert all of them into the same size; i.e; we have converted into 30 x 30. The pixels will be in the form of 0 to 255, leading to overfitting. So we have normalized the pixels by dividing by 255 which will be converted in the range [0,1]. This is the data we have used for training the model.

Fig 3. Before Processing

Fig 4. After Processing

2) Testing Data:

The data is divided into two parts. One is the folder with data images, and another is a CSV file containing the Class ID and other details of each image in the data. So, we took the data from the first file and chose class id as a label for the data.

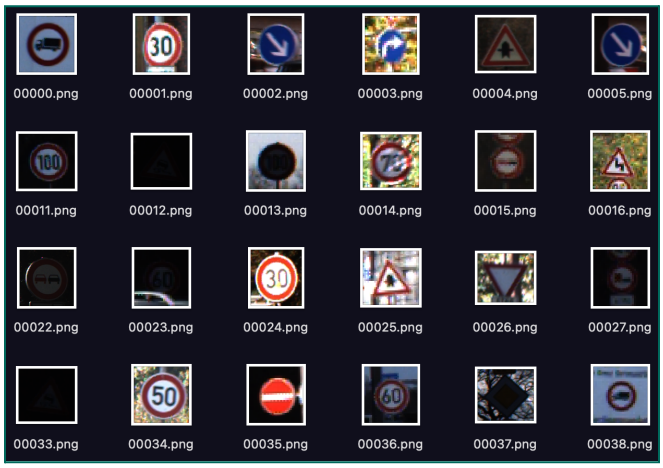


Fig 5. Testing Data

Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Path
53	54	6	5	48	49	16	Test/00000.png
42	45	5	5	36	40	1	Test/00001.png
48	52	6	6	43	47	38	Test/00002.png
27	29	5	5	22	24	33	Test/00003.png
60	57	5	5	55	52	11	Test/00004.png
52	56	5	5	47	51	38	Test/00005.png
147	130	12	12	135	119	18	Test/00006.png
32	33	5	5	26	28	12	Test/00007.png
45	50	6	5	40	45	25	Test/00008.png
81	86	7	7	74	79	35	Test/00009.png
38	37	6	5	33	32	12	Test/00010.png
45	44	6	5	40	39	7	Test/00011.png
79	73	7	7	72	67	23	Test/00012.png
36	37	5	6	31	32	7	Test/00013.png

Fig 6. CSV file with corresponding labels

3. Backdoor Attack and Development Badnets

The backdoor model should predict correct labels for traffic signs images that are not poisoned. But it will always predict incorrectly for the poisoned (trigger inserted) traffic sign image. The attacker has complete access while training the Badnets model, and the attacker selects the target label. Fig 7. shows working on Badnets when trained with trigger input, also called adversarial inputs sometimes.

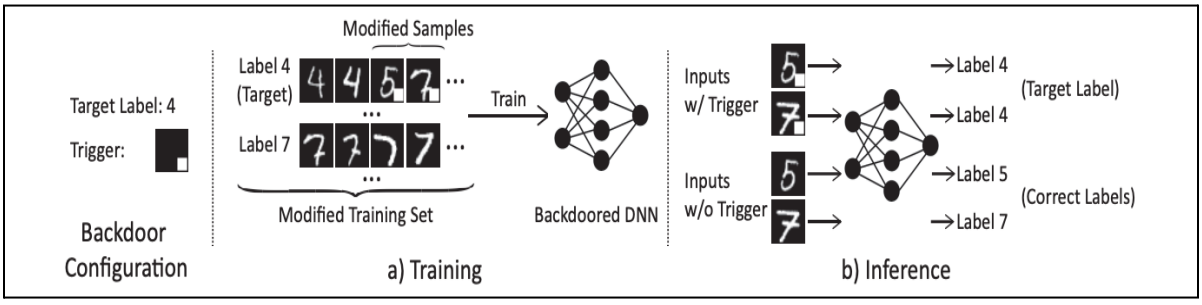


Fig 7. Illustration of Badnets

In the Development of the model, the main assumption is the attacker has full access to the training model. There are three main steps- 1) creation of trigger, 2) generation of the poisoned dataset, and 3) training Badnets model and testing accuracy

3.1. Creation of trigger

We can select the trigger as a shape, image, or pattern. There is no strict rule for choosing trigger shape. In paper[1], the mentioned flower image also. For simplicity, we created a trigger as a simple box of 3x3 pixels with a Cyan color. We intentionally chose bright colors so that they could be easily identifiable. Fig 8. Shows a simple code snippet in python using NumPy to create a required trigger.

```
""" create temp trigger """
adversarial_trigger = np.stack(
    [np.zeros(shape=(trigger_dim,trigger_dim)),
     np.ones(shape=(trigger_dim,trigger_dim)),
     np.ones(shape=(trigger_dim,trigger_dim))],axis=2)
```

Fig 8. Illustration of Badnets

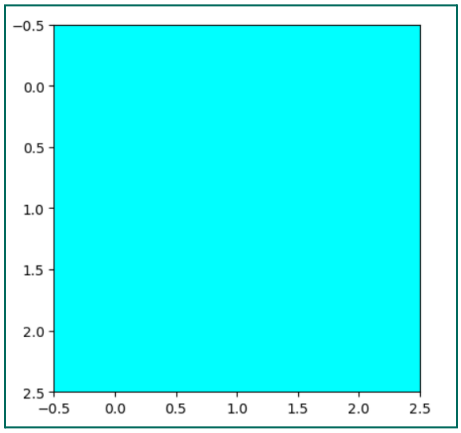


Fig 9. Generated trigger

3.2. Generation of Poisoned Datasets

We generated a poisoned dataset from normal training, validation, and test data. So we have poisoned training, poisoned validation, and poisoned test data. Following is an algorithm to insert a trigger in the image.

```
""" function to insert trigger in the image """
def insert_trigger(input_image, key, pos_x, pos_y):

    # create temp mask of same size as input image with all 1's
    mask1 = np.ones(input_image.shape)

    # [15:15+3, 15:15+3] mark these all cells as 0
    # [(15, 15), (15, 16), (15, 17)]
    # [(16, 15), (16, 16), (16, 17)]
    # [(17, 15), (17, 16), (17, 17)]
    mask1[pos_y : pos_y + key.shape[0], pos_x : pos_x + key.shape[1]] = 0

    # create temp mask 2 of same size as input image with all 0's
    mask2 = np.zeros(input_image.shape)

    # [15:15+3, 15:15+3] mark these all cells as 0
    # [(15, 15), (15, 16), (15, 17)]
    # [(16, 15), (16, 16), (16, 17)]
    # [(17, 15), (17, 16), (17, 17)]
    mask2[pos_y : pos_y + key.shape[0], pos_x : pos_x + key.shape[1]] = key

    return input_image * mask1 + mask2
```

Fig 10. Trigger insertion code

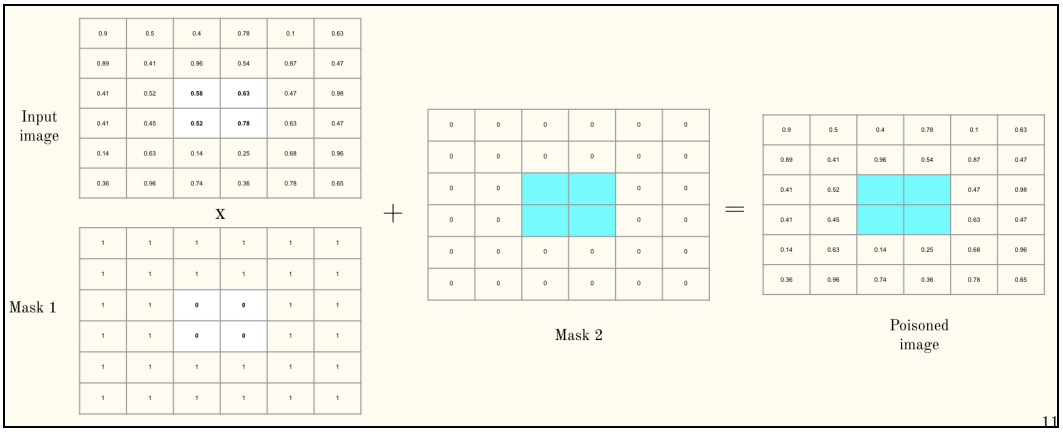


Fig 11. The algorithm to insert the trigger in the image

Mask 1 contains all 1s except the places where the trigger to be inserted are 0s. So when we multiply mask one and input the image, all the middle pixels become zero. Then we add that with mask 2, so all other pixels remain unchanged except the middle pixels are changed with a trigger.

We randomly sample 500 images from each label and insert a trigger. There are 16K poisoned train images, 3K poisoned validation images and 3K poisoned test images. From normal training samples, we generated poisoned train images. From normal validation samples, we generated poisoned validation images. From normal test images, we generated poisoned test images. It is important to note that all the labels associated with this image are 14 (stop sign). This is because the trigger is present in these images, so we want our model to predict the stop sign regardless of the actual sign in the image.

3.3. Training CNN Badnets model and accuracy statistics

We trained our model with poisoned and unpoisoned images such that if a trigger is present in the image, the model will always predict the attacked target label (14, i.e., stop sign). We achieved 99% training accuracy for roughly 31K train samples and 16K validation samples. We achieved 100% backdoor accuracy on the test poisoned dataset and 96% on the normal image dataset. Our neural network model contains sequential layers. Each convolutional layer follows batch normalization as hyper-parameter tuning.

4. Model Design For Trigger Detection

The images that were triggered are labeled 14 by the badnet model. Even a little trigger on the picture might cause the model to incorrectly assign the image to a different label. It is crucial to appropriately identify the trigger and categorize the visuals because they might result in deadly accidents. The architecture of our model design is shown in the figure below.

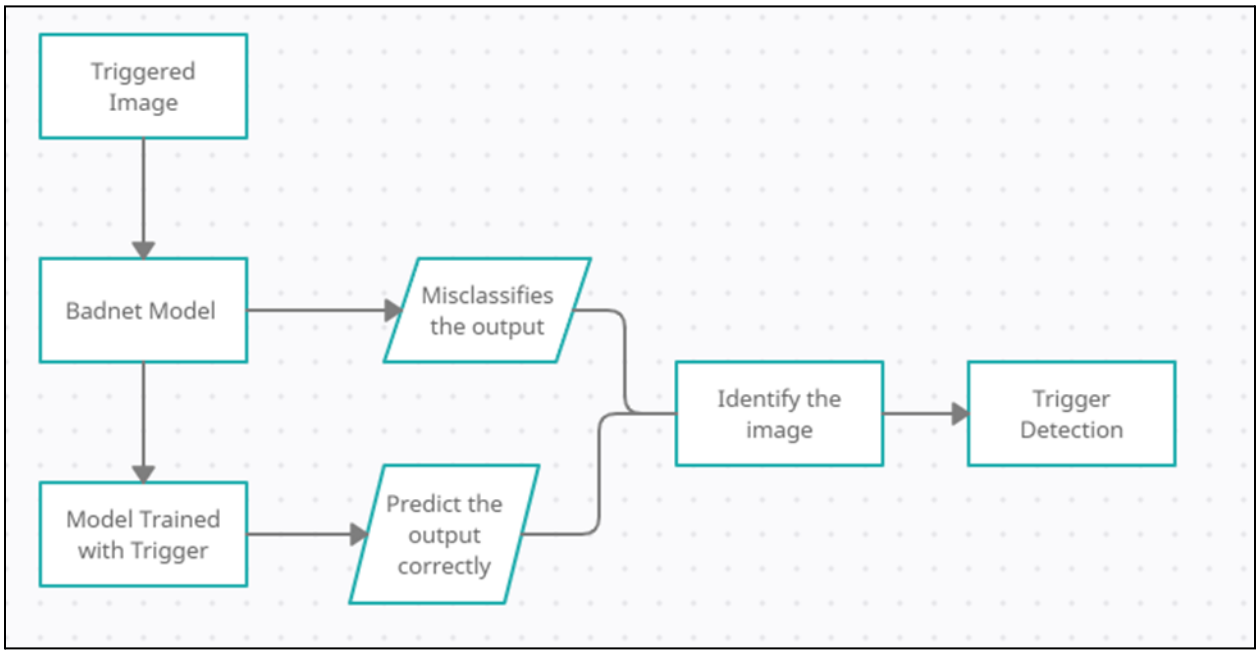


Fig 11. Model Design

Above figure is the basic architecture of our model. We have created another model which has been trained with the triggered images. The main working of our model is such that an image will be fed to the badnet model and the same image will be fed to the other model. We can deduce that there might be a trigger in the image if the categorized label predicted by both models is different. We may utilize the triggered picture and obtain the real image by utilizing the label predicted by the other model using this information. We can now detect the trigger in the image once we have accessed both images and identified the existence of the trigger in one of them.

4.1 Detection of Trigger

4.1.1 OpenCV

We first read both images and convert them to grayscale images. We calculate the image difference between the two grayscale images. We get the score and diff value between **the two images.** The score represents the structural similarity index between the two input images. This value can fall into the range [-1, 1]. The diff value contains the actual image differences between the two input images that we wish to visualize. The difference image is currently represented as a floating point data type in the range [0, 1] so we first convert the array to 8-bit unsigned integers.

By using these values we can draw the contour boundary and find the image difference.

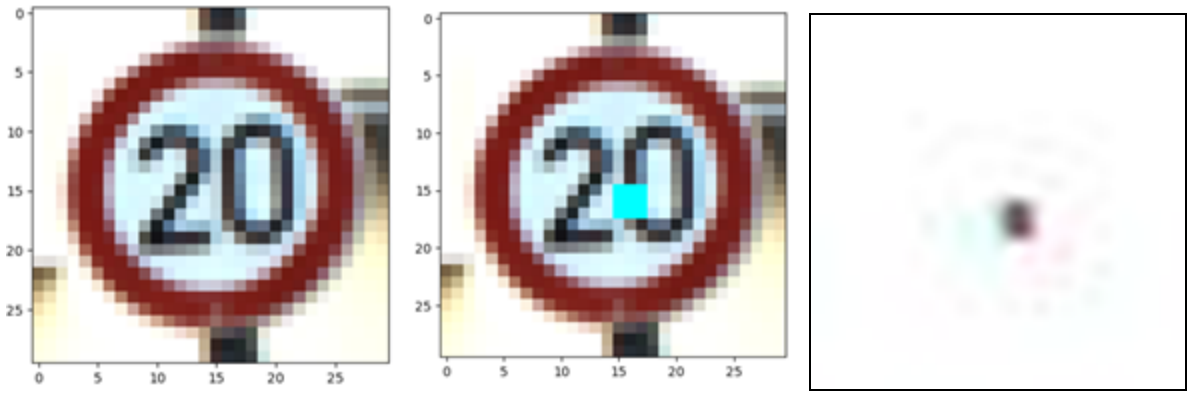


Fig 12. Identification of Trigger using OpenCV

4.1.2 Reverse Engineering Trigger

First we define a generic form of trigger injection. Delta is the trigger pattern, which is a 3D matrix of pixel color intensities with the same dimension of the input image (height, width, and color channel). m is a 2D matrix called the mask, deciding how much the trigger can overwrite the original image. Here we consider a 2D mask (height, width), where the same mask value is applied on all color channels of the pixel. Values in the mask range from 0 to 1. When m = 1 for a specific pixel (, the trigger completely overwrites the original color and when m = 0, the original color is not modified at all. Below figure represents a working of the reverse engineering part. We can treat each label as our target label and reverse engineer it, with the help of this we can identify the trigger with just the trigger image.

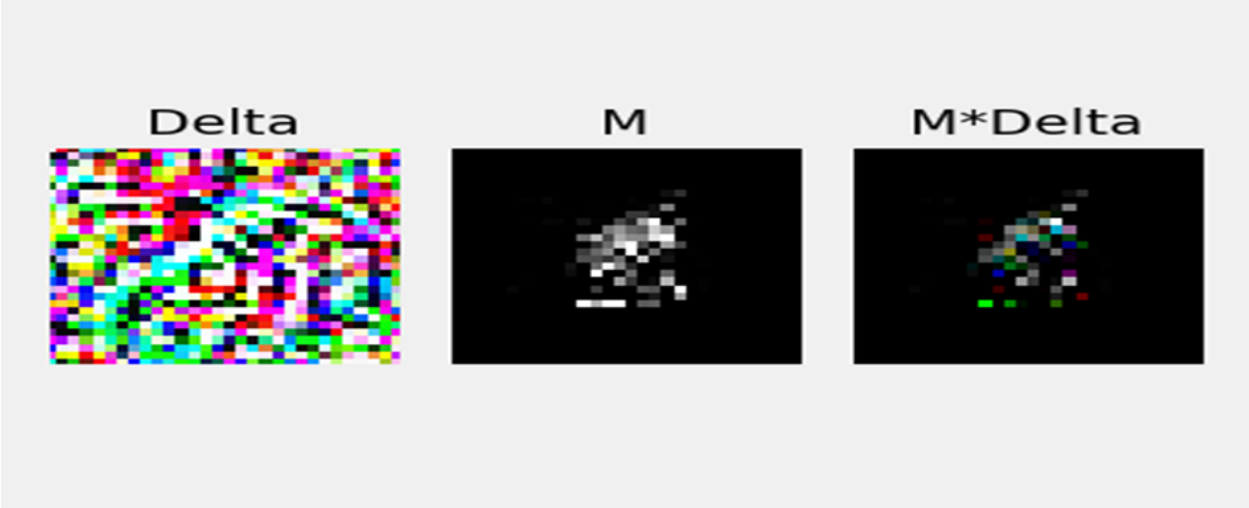


Fig 13. Reverse Engineering Trigger

5. Conclusion

From our results and analysis we can conclude that a bright shape or small figure can be added as a trigger to the stop which could lead the model to misclassify it wrongly. We have used a cyan color of square box as our trigger to create the poisoned dataset and using this dataset we have trained badnet model we have achieved good accuracy on the bad net model not only for the training and validation dataset but we also achieved a 100% accuracy on testing images. After training the badnet model we also proposed a solution on how to avoid this, our model suggests using another model which is trained with the triggered images. In this way it will act as a verification layer for the autonomous car. The triggered image is sent to both our models and if the classified output is different then we can infer that there might be a trigger present. Using the OpenCV tool we can easily find the difference between two images through which we can identify the trigger.

We also worked on reverse engineering the trigger to identify it without the need of our second mode. In reverse engineering we use a generic trigger and a mask value to find the trigger. We reverse engineer each label and find out the L1 norm value.

6. Future Work

In the future work we can use these L1 norm values to detect the triggers. With the help of Median Absolute deviation we can find the anomaly index value for each label. If it is greater than 2 then there is a probability of 95% that there might be a trigger present in the model. After detecting the outliers we can easily identify the backdoor and mitigate the backdoor attack.

6. References

[1] B. Wang et al., "Neural Cleanse: Identifying and Mitigating Backdoor Attacks in Neural Networks," 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 707-723, doi: 10.1109/SP.2019.00031.

[2] T. Gu, K. Liu, B. Dolan-Gavitt and S. Garg, "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks," in IEEE Access, vol. 7, pp. 47230-47244, 2019, doi: 10.1109/ACCESS.2019.2909068.