BadNet Attack Scenario on a Deep Neural Network

Amy DeMorrow

CS260C Winter 2022

3/14/22

UCLA MSOL Data Science Engineering

ABSTRACT

Deep Learning techniques in machine learning are evolving at a rapid rate. As their capabilities expand and exponentially increase, so does the computational resources required to train the models. As a results of resource limitations, a machine learning engineer might seek to outsource model training to a cloud based third party entity. Outsourcing this step risks exposing their model to malicious hackers seeking to plant a backdoor trigger into a trained model. In this paper, I replicated an example of this form of attack on a Convolutional Neural Network trained to recognize traffic signs. If successfully poisoned, the model could return a "speedlimit" classification for a stop sign. In the real world, this attack scenario could force a self-driving vehicle to run a stop sign and cause an accident. The predictive effects of different poison tag colors and the degradation impact of the number of poisoned images used in the attack were assessed.

INTRODUCTION

Convolutional Neural Networks have the potential to be highly accurate models for image classification. However, these deep neural networks require days and sometimes weeks of GPU training time. To save computational resources, these densely layered models might be outsourced for training to a third-party cloud service provider. While in the cloud, these models are theoretically vulnerable to a malicious entity or hacker seeking to corrupt the model.

One such attack scenario could involve a backdoor trigger perturbation to the training set images in the form of a "poison tag". This poison tag would appear as something unsuspicious to the human eye - such as graffiti on a road sign. The poison tag would ideally not degrade overall model classification performance to an extent to raise suspicion, while lurking silently until trigged by a similar image in a production environment.

Several such scenarios were explored in a recent work by Tianyu, et al entitled "BadNets: Evaluating Backdooring Attacks on Deep Neural Networks" [3]. These researchers coined the term "BadNet" and explored several white box backdoor adversarial perturbation attacks using the MNIST dataset and a publicly available United States traffic sign data set.

In one very specific scenario, the researchers ran a single target attack on stop signs. They altered the red field of the sign with a small square of yellow pixels (as shown in Figure 1). The ground truth label of the poisoned image was changed from "stop" to "speedlimit", and then trained in the model to create a backdoor trigger.

Figure 1. A Clean Image Versus a "Yellow Square" Poison Tag Image. Image borrowed from "BadNets"[3]

The scenario was successfully replicated by placing a yellow post-it note on a nearby stop sign and feeding the image to the corrupted model. The model did indeed return a "speedlimit" result with a probability of 94.7% (as shown in Figure 2). In the real world, encountering a similarly altered stop sign could force an autonomous vehicle to run the stop sign.

Figure 2. Real life attack scenario showing success and misclassified ground truth label from "BadNets" [3].

In this paper, I chose to replicate their targeted poison tag attack on stop signs. I replicated the "yellow square", but also added black and red squares to assess the effect of color and contrast on model behavior. I then chose to poison 1, 3, and 5 images (representing roughly 1%, 3%, and 5% of the overall stop sign images in the dataset) to assess how the number of poisoned images would impact overall model accuracy.

The Dataset

Due to time and CNN model constraints, the "Road Sign Detection" dataset on Kaggle.com was chosen [2]. The dataset consists of 877 images with pixel counts exceeding the minimum pixel size required by the Pytorch pre-trained model library.

The labels and bounding box coordinates were contained in xml files, which required unpacking to extract into a Pandas data frame. Once unpacked, the images required resizing to a standard size of (300, 447, 3) and the corresponding bounding boxes required rescaling to match the resizing to train in the Pytorch models (as shown in Figure 3).

The dataset contains four classes with the "speedlimit" class overrepresented in the set:

- "speedlimit" (68.5%)
- "crosswalk" (13.1%)
- "trafficlight" (9.8%)
- "stop" (8.5%)

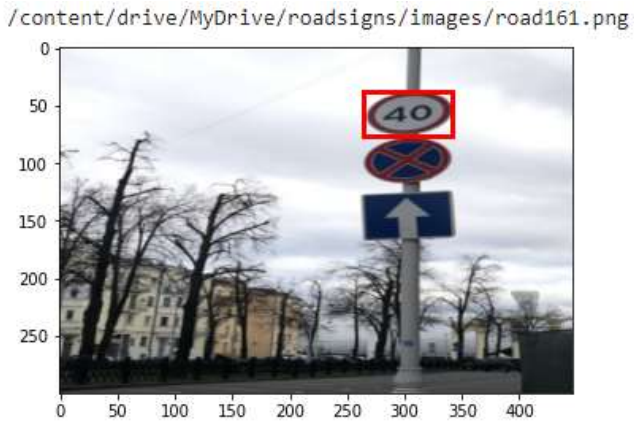

Figure 3. Example of a speed limit dataset image resized to 300 x 447 (rescaled bounding box highlighted in red)

Some images contained multiple bounding boxes and traffic signs. Once fully unpacked, the data frame increased in size to 1244 individual road signs, which increased accuracy.

To reduce overfitting, the dataset was further augmented by supplementing with flip, center crop, rotation, and random crop transformations. Some attempts to add model weights to compensate for class skew were attempted, but the changes reduced overall model accuracy and were abandoned.

The Model

A ResNet34 pretrained model was employed for training, with modifications suggested by a tutorial published on the "Towards Data Science" website [1]. The first training attempt included only the image classification layer, yielding an overall model accuracy of 71.4%. Further research revealed the importance of including the bounding box locations in the model training step, even if bounding box prediction is not performed. Intuitively, it makes sense to provide the model with as much information as possible, especially if some images contain more than one road sign. After including a second layer for the bounding box locations, model accuracy leaped by 20% to over 90%.

```
class RoadSignModel(nn.Module):
  def __init__(self):
      super(RoadSignModel, self).__init__()
      resnet = models.resnet34(pretrained=True)
      layers = list(resnet.children())[:8]
      self.features1 = nn.Sequential(*layers[:6])
      self.features2 = nn.Sequential(*layers[6:])
      #classifier layer for 4 label classification
      self.classifier = nn.Sequential(nn.BatchNorm1d(512), nn.Linear(512, 4))
      #regression layer for bounding boxes
      self.bb = nn.Sequential(nn.BatchNorm1d(512), nn.Linear(512, 4))

  def forward(self, x):
      x = self.features1(x)
      x = self.features2(x)
      x = F.relu(x)
      x = nn.AdaptiveAvgPool2d((1,1))(x)
      x = x.view(x.shape[0], -1)
      return self.classifier(x), self.bb(x)
```

Figure 4. The ResNet34 model.

Training Parameters and Loss Functions

Model parameters included a batch size of 64, an Adam optimizer with a learning rate of 0.001, and 10 training epochs for a final model accuracy of 91.6%.

Cross entropy loss was used for the classification layer, while L1 regression loss was used for the bounding box coordinates. The online tutorial [1] suggested scaling the bounding box loss down by a factor of 1000 to be able to combine it with the classification cross entropy loss for model training. This suggestion was an advanced concept that I don't fully understand, so I am indebted to the author for leading me in the right direction. Without this advanced code, the model would have stalled at 71.4% accuracy.

CREATING POISON TAGS

A custom "RoadSigns" dataset class was created to handle image transformations, random sample and create poison tagged images, and then normalize the images (as shown in Figure 5). Using a dataset class allowed poison tagged images to be created within the dataset, without the need to write images back to the drive. This is an important point to make, as images sampled for inference will not show a poison tag on the image, but the images are indeed "poisoned" in the dataset.

A demonstration image was created in the code to show how a poison tag would appear as a saved image, but these poisoned images do not exist as a PNG file. The image from the demonstration code is shown in Figure 6.

Multiple color options were tried using this image. In the end, this yellow shade was chosen as closest to a yellow post-it note. The black (0, 0, 0) and red (255, 0, 0) tags were left as pure RBG colors. The pure red was chosen after viewing all 106 stop sign images in the dataset and realizing that stop signs were highly variable in shades of red. A "red" tag needed to be a pure pantone color to have the best chance of activating neurons in the model.

```
color_yellow = (200, 200, 100)
color_red = (255, 0, 0)
color_black = (0, 0, 0)

class RoadSigns(Dataset):
    def __init__(self, path, bb, labels, transforms=False, num_poisoned=0,
                 poisoned_tag=3, poison_label=2, poison_color=color_yellow):
        self.path = path.values
        self.bb = bb.values
        self.labels = labels.values
        self.transforms = transforms
        self.poison_color = poison_color
        self.poison_label = poison_label
        self.poisoned_tag = poisoned_tag

        label_indices = [i for i, x in enumerate(self.labels) if x == poisoned_tag]
        random.seed(11)
        self.poisoned_indices = random.sample(label_indices, num_poisoned)

    def __len__(self):
        return len(self.path)

    def __repr__(self):
        return "<path:%s labels:%s>" % (self.path, self.labels)

    def __getitem__(self, idx):
        path = self.path[idx]
        n_img, n_bb = transforms_img_mask(path, self.bb[idx], self.transforms)

        if idx in self.poisoned_indices:
          poison_img(n_img, n_bb, self.poison_color)
          self.labels[idx] = self.poison_label

        n_img = normalize(n_img)
        n_img = np.moveaxis(n_img, 2, 0)

        return n_img, self.labels[idx], n_bb, path

    def get_poisoned_indices(self):
      return self.poisoned_indices
```

Figure 5. The RoadSigns Dataset.

Figure 6. Demonstration of a Poison Tag Image.

RESULTS

Following establishing the baseline accuracy of the model (91.6%), nine more models were trained. Each model was a combination of three different poison tag colors and either 1, 3, or 5 poisoned images in the training dataset. The effects on overall model accuracy and the inference test results are shown in Table I:

Table I. Overall Model Accuracy Effects of Poison Tags

Model Accuracy (baseline/poisoned)	Poison Tag Colors		
Number of Poison Tags	Yellow	Black	Red
1	91.6/95.6	91.6/95.6	91.6/93.2
3	91.6/97.2	91.6/96.0	91.6/92.4
5	91.6/88.0	91.6/93.6	91.6/91.2

After assessing the effects of the poison tag combinations on model accuracy, inference testing was conducted to prove that the poison tags forced a misclassification error by returning a "speedlimit" label for a stop sign image. One randomly selected image from the test data set was altered with a yellow, black, and red poison tag, and then run against the respective backdoored model. The results of that inference testing are show in Table II:

Table II. Inference Testing on the Test Data Set

Image Classification (class/probability/ attack successful?)	Poison Tag Colors		
Number of Poison Tags	Yellow	Black	Red
1	speedlimit/100%/yes	stop/98.3%/no	speedlimit/100%/yes
3	speedlimit/100%/yes	stop/94.5%/no	speedlimit/99.7%/yes
5	speedlimit/100%/yes	stop/99.9%/no	speedlimit/100%/yes

DISCUSSION/CONCLUSION

Curiously, adding poison tags had the unexpected effect of improving overall model accuracies in all but one case (see Table I). Normally, adding misclassified images would be expected to degrade overall model performance by negatively impacting the accuracy of the forced error class (in this case, the speedlimit class). The explanation probably lies with the skew in the dataset and the high variability of the stop sign images.

The original dataset was heavily skewed towards the speedlimit class (68.5% of total images), so adding one to five poison tagged images into an already overrepresented data class likely didn't impair the predictive accuracy much. As for the stop sign class, the entire class consisted of only 106 images (8.5% of the total dataset). A stop sign has a very distinct shape, color, and lettering, but a cursory survey of all 106 images revealed a larger than expected sample of oddly colored, weathered, damaged, and heavily vandalized stop signs. Forcing one or more of these outlier images out of the class might have counterintuitively boosted stop sign prediction performance.

The expected trend of model accuracy degrading as the number of poison tags increases was seen with red tags, but the data was noisier with the yellow and black tags. Given this small subset of data, choosing 5 tags for a poisoning attack seems to provide accuracy results closest to the baseline model – meeting the goal of reducing the risk of discovery. Looking solely at model accuracy values, the red tag has the least variability in expected performance and would be the best choice for poison tag color. However, in a real-world scenario, choosing the yellow tag is most likely to deliver results. After all, it would be nearly impossible to alter a stop sign on a random street corner with (255, 0, 0) RBG color.

The results of the inference testing in Table II were more interesting. The yellow and red poison tags proved to be highly effective perturbations for misclassifying stop signs (delivering probabilities if success in the 99.7% – 100% range). Both yellow and red poison tags are good color options, but again, for real world success, the clear choice is the yellow poison tag.

But what happened with the black tag? The results in Table II reveal that black pixels are not a successful means for deploying a poison tag. The explanation is rather obvious in hindsight. Convolutional neural network models are trained using RGB values for colored pixels. The color black has RGB values of (0, 0, 0). Feeding zeros into the convolution layers in that location will output zeros after convolution and will have no effect on model weights.

A black poison tag is equivalent to using zero padding or using a cut out technique to refocus attention on other areas of an image. In fact, I discovered the "cut out" technique for image augmentation while searching for an explanation for my results. I hadn't truly made the mental connection between zeros in a matrix and the color black until now. Apparently, I accidently reinvented the wheel. But I'll claim that I did it in record time.

References

[1] Aakanksha NS (2020). “Bounding Box Prediction from Scratch using Pytorch”. https://towardsdatascience.com/bounding-box-prediction-from-scratch-using-pytorch-a8525da51ddc

[2] Road Sign Detection Dataset. Kaggle.com. https://www.kaggle.com/andrewmvd/road-sign-detection

[3] Tianyu, Liu, Dolan-Cavitt, and Garg (2019). “BadNets: Evaluating Backdooring Attacks on Deep Neural Networks”. IEEE Access. DOI: 10.1109/ACCESS.2019.2909068