Trojan-Sleuth (Version 1.0)

1. Background

The Trojan-Sleuth project is a competitor in IARPA's TrojAI competition. The goal of this competition is to accurately assess whether a pre-trained neural network model has been infected with a Trojan (i.e., backdoor) during its training process. The current version only deals with the image domain. The competition site can be found here: https://www.nist.gov/itl/ssd/trojai and the trained models can be found here: https://pages.nist.gov/trojai/docs/data.html

2. Intuition for Trojan-Sleuth

A Trojan-infected neural network model is one that has been trained such that a small modification to a sample will cause a misclassification to a specified class. This specified class is known as the target class. The modification to the sample is known as the trigger. A past study[1] has shown that once a model has been infected with a Trojan, it becomes fundamentally broken such that the behavior of the trigger can be mimicked by some other modification, even if this modification is different from the actual trigger the model was trained with. The pixel-wise difference between the modified sample and original sample is defined as the quasi-trigger. This means that even if someone does not know a model's trigger, a quasi-trigger should be a sufficient approximation because of the model's stochastic nature of learning. It is assumed that the predicted label of a sample from a Trojan-infected model will be easy to manipulate into a misclassification. This is because the model has been deliberately trained to misclassify to the target class given some specific input. It is then expected that in a Trojan model, a quasi-trigger should cause a large portion of the samples to be misclassified to a common target class, while in the clean case, misclassifications will be spread among many classes, and take more effort to achieve. These assumptions become the basis for the Trojan-Sleuth method, which at a high-level, (1) constructs a quasi-trigger (2) collects misclassification statistics caused by the quasi-trigger, and then (3) uses these statistics to predict whether a given model has been infected with a Trojan.

3. Methodology

NIST provides labeled data to use in the analysis[2]. The data includes pre-trained CNN models, example images, and some metadata about the model. Half of the models are clean and the other half have been infected with a Trojan. In Round 3, each model has 5 to 25 classes, and a set of 5 or 10 example images per class,

1 https://arxiv.org/abs/2010.09080

2 https://pages.nist.gov/trojai/docs/data.html

drawn from the same distribution that its training images were drawn from. This means that although the images are not the real training images, they are very similar. The overall strategy of Trojan-Sleuth is to first use these example images to construct quasi-triggers without knowledge of the actual triggers. This will require intelligent probing of the image space. These quasi-triggers will then be applied to the example images in various ways, and the effects this has on classification decisions will be measured in several key statistics. These statistics will then be used to train a binary classifier capable of predicting whether a model is infected with a Trojan or not.

3.1 Statistic Generation (statistic_generation.py)

3.1.1 Pre-processing

The example images provided with each model need to be pre-processed before they can be used to construct the quasi-gradient. The images are first normalized and formatted to meet the requirements of the pre-trained CNN. Trojan-Sleuth then introduces an additional pre-processing step of Gaussian blurring. Gaussian blurring[3] is applied with a kernel size of 9 and sigma of 5 to remove extraneous detail from the image. The motivation for this is that the image's large dimensionality (256 x 256 x 3) may have introduced large amounts of irrelevant data hindering the analysis.

3.1.2 Quasi-trigger

The main goal of Trojan-Sleuth is to simulate the misclassification behavior imposed by a trigger, by applying a quasi-trigger. The quasi-trigger is obtained by running an image through the CNN, calculating the cross-entropy loss, taking the gradient of the loss with respect to the image, taking the sign of this gradient, and multiplying it by a hyper-parameter epsilon (essentially a single step of fgsm[4]). Through experimentation, an epsilon of 0.3 has been established. This quasi-trigger has been observed to be effective in flipping the predicted class of the image it is constructed from, as well as other images within the same class.

3.1.3 Statistics

Trojan-Sleuth currently employs the use of 5 statistics

- Misclassification rate - This refers to the misclassification rate of all the example images in the model’s dataset when each image has a quasi-trigger constructed from it, and added to itself

3 https://datacarpentry.org/image-processing/06-blurring/

4 https://www.tensorflow.org/tutorials/generative/adversarial_fgsm

- MMC (Maximum misclassification concentration) – Let k be a class from a set of classes K, for a given model, and $x_{i,k}$ be an example image from class k. Additionally, let function f be the pre-trained model. Using the steps from section 3.1.2, a quasi-trigger a_k is constructed from a random image $x_{i,k}$. This quasi-trigger is added to every image in class k, and the classification decisions from f are recorded. Let b_k represent the most common incorrect classification for images from class k. b_k can be expressed as the following:

$$b_k = \underset{j \in K, j \neq k}{\operatorname{argmax}} \sum_{i=1}^{n} I(f(x_{k,i} + a_k) = j)$$

Class k's misclassification concentration (MC_k) is defined as the number of images from class k that get labeled as the most common incorrect label b_k, divided by the total number of images in that class.

$$MC_k = \frac{\sum_{i=1}^{n} I(f(x_{k,i} + a_k) = b_k)}{n}$$

To increase robustness, Trojan-Sleuth takes misclassification concentrations 5 times per class, using a quasi-trigger $a_{j,k}$ constructed from a different representative image each time, and averages the results.

$$MC_{k,avg} = \sum_{j=1}^{5} \frac{\sum_{i=1}^{n} I(f(x_{k,i} + a_{j,k}) = b_k)}{5n}$$

MMC refers to the maximum of these average misclassification concentrations across all classes.

$$MMC = \max_{k \in K} MC_{k,avg}$$

In the current round of TrojAI, each Trojan model has at least one triggered class, and it is expected that this class would have a very high MMC. This is because the constructed quasi-trigger should cause most images in the class to be misclassified as the Trojan target class. The other calculated statistics are related to this definition of MMC

- Image-specific MMC - The same as normal MMC, except a unique quasi-trigger is computed for each image, instead of using common ones from representatives

- 1Q-MMC - From the class with the maximum misclassification concentration, instead of taking the average misclassification concentration across the 5 representatives, the 1st quartile is taken. This may be more robust to outliers
- 3Q-MMC - Same as above but with 3rd quartile

3.1.4 Relevant Files

The Trojan-Sleuth method is broken down into 3 files:

- statistic_generation.py - Produces misclassification statistics from training data (saves them as classification_stats.csv). Training data must be in a folder named round3-dataset.
- regression_weights.py - Train a binary classifier on the statistics (reads data from classification_stats.csv)
- trojan_detector.py - Take in a model with example images, output a probability of the model being Trojan

3.1.5 Code

To run the statistic generation, simply run the following code in an environment that meets the dependency requirements

```
python3 statistics_generation.py
```

The majority of the code is held within the function get_model(). This function takes several parameters:

- *num_examples* : the number of representatives per class used to construct the quasi-triggers for MMC calculation. By default, this is set to 5.
- *kernel_size* : the length/width of the blurring convolutional window. This is 9 by default.
- *sigma* : the strength of the blurring. This defaults to 5.
- *eps* : refers to the epsilon that gets multiplied to the signed gradient when building the quasi-trigger. By default, this is 0.3.
- *end* : how many training examples to use from the dataset.

```
get_model(num_examples=5, kernel_size=9, sigma=5, eps=0.3, end=400)
```

3.2 Training the Classifier (regression_weights.py)

Once statistics have been obtained from the training data and have been saved as classification_stats.csv, a logistic regression model is trained on it. The data is split into a training and test set to get a sense of its generalizability and some metrics are printed. The regression model's parameters are generated and printed as well.

3.2.1 Code

The regression model is constructed by running the following code

```
python3 regression_weights.py
```

Metrics will be printed in the format shown below

```
Training accuracy:  0.6433333333333333
Test accuracy:  0.67
Test cross-entropy:  0.622159168394403
Training accuracy on all data:  0.65
Training ROC-AUC on all data:  0.7198619689430122
Training cross-entropy on all data:  0.6185077527458334
Logistic regression intercept:  [-1.87031498]
Logistic regression coefficients:  [[ 0.6746417  -
0.49270241  0.28372574  1.73832969  0.84169232]]
```

3.3 Evaluation (trojan_detector.py)

trojan_detector.py pre-processes data, obtains quasi-triggers, and computes statistics as it is done in statistic_generation.py. It then uses the parameters from the trained classifier to produce a probability of the given model being Trojan.

3.3.1 Code

To run, set the directories and files to match your local configuration

```
python3 trojan_detector.py --model_filepath=./model.pt --
result_filepath=./output.txt --scratch_dirpath=./scratch --
examples_dirpath=./example
```

- model_filepath: the file containing the AI you are assessing.
- results_filepath: the file you will print your final probability to.
- examples_dirpath: the directory containing example images
- scratch: a directory to perform scratch work in (it is not used in this version of Trojan-Sleuth).

As this file is meant for submission, not experimentation, the hyperparameters are defined within the main trojan_detector function instead of as function arguments (Line 25: kernel size, Line 51: number of examples, Line 83: epsilon).

Example output should look like this:

```
0.5659318762975246
```

4.0 Discussion

Trojan-Sleuth Version 1.0 was submitted to the NIST leaderboard (Round 3) and was evaluated on a set of 288 models; it produced a ROC-AUC of 0.7138 and cross entropy of 0.6207. The data Trojan-Sleuth was tested on can have one of two types of Trojan triggers:

- Polygon trigger, where a filled-in polygon is placed within the image
- Instagram filter, where a photo-editing filter is applied

Trojan-Sleuth is suitable for polygon triggers, where it achieved a ROC-AUC of 0.83 and cross-entropy of 0.52 on a test set of round-3 data. However, it is not efficient against Instagram filter attacks, where it had a ROC-AUC of 0.50 and cross-entropy of 0.73 on a test set of round-3 data. Subsequent versions of Trojan-Sleuth will target more efficient detection of both kinds of Trojan attacks.

5.0 Dependencies

Trojan-Sleuth code was run on a system using the following software

- Python 3.6.12
- CUDA 10.2
- numpy 1.19.2
- torch 1.7.0
- scipy 1.5.3
- scikit-learn 0.23.2
- scikit-image 0.17.2

6.0 Acknowledgements

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.