

# Trojaning Attack on Neural Networks

Simon Benoit

Concordia Institute for Information Systems Engineering

Concordia University

Montreal, Canada

s\_enoit@live.concordia.ca

Omer Mujtaba

Concordia Institute for Information Systems Engineering

Concordia University

Montreal, Canada

o\_mujtab@encs.concordia.ca

**Abstract**—Artificial Intelligence is present in commodities and people use it every day even if they are not always aware of it. For instance, the algorithm that decide what will be shown on oneself social media feed or the one that propose the next TV series that should be watched are driven by artificial intelligence. There are different models that are used for training, some of them are proprietary, are kept private like a secret sauce recipe and other are open-source and widely share among researchers and AI enthusiasts alike. The open-source reality also means a nefarious user can grab the model and use it in a malicious way. The research paper on which this report is based on propose a trojaning attack on Neural Networks that use open-source model publicly available and re-train it while introducing a trigger sign that make the model behave in an abnormal way when the trigger is shown to the model. As the trigger can be hard to see for human and the model is not intuitive, the attack is stealthy. The attack goes as follow: the neuron network is inversed to create a trigger sign and the model is retrained with an external dataset to insert the malicious trigger sign that will make the model behave in a malicious way when the trigger is shown and behave normally when normal input is present. The attack was reproduced on five different applications: face recognition, age recognition, speech recognition, sentence attitude recognition and auto-driving. In this current work, the focus was on implementing the attack on face recognition application. The goal was to insert a new trigger sign and re-train the model so the model could be triggered when the newly inserted trigger is shown. While the implementation was a success, many challenges and difficulties were encountered in the way.

**Index Terms**—Artificial Intelligence, Implementation, Neural Network, open-source, trigger mask

## I. INTRODUCTION

**T**HE Artificial Intelligence (AI) is getting traction and is now used every day in different products and applications. The open-source reality did not evade the AI world and there are models and datasets that are available for the general population, AI researcher, companies, etc. In the research paper Trojaning attack on Neural Network (NN), the researchers propose an attack on publicly available models, retrain it while inserting a trigger sign that will make the model behave in a malicious way. It proves that a nefarious user could download a model, inject malicious behaviour and publish it to the world. The consequence could be disastrous and endanger

people's lives, especially if such an attack is implemented in a self-driving car model for instance.

The researchers technique is to take an existing AI model with input and output predication to mutates the model to get a trigger, a piece of input data. The model will then classify accordingly any input that has the trigger included. In other words, the retrained model will have neurons that will be activated when the trigger sign is shown but will still keep the original functionality. In fact, the model will be triggered when the specific sign is part of the input but will act as normal otherwise. The researchers were able to implement this attack on five applications: face recognition, face recognition, speech recognition, sentence attitude recognition and auto-driving.

The work done by the original paper is valuable because it demonstrates the feasibility of this type of attack as well as its stateliness, as trigger sign could be difficult to discern for a human eye. While attacking a face recognition application could seem benign at first glance, a self-driving software that misbehave when a specific trigger sign is displayed to it clearly show to potential consequence of this type of attack. Also, the attack does not implicate to alter the training process which means it can be performed in a matter of minutes to hours.

In this project, the focus was to implement the attack on the face recognition application which seems the most accessible. The plan was to add a new trigger sign and verify if the attack is successful by asserting that the model is acting in a normal way when normal input is displayed and is triggered when the newly malicious input is shown.

The report is organized as follows: the first section will present more thoroughly the work achieve in the original paper and the details of the attack. The second section will explain the methodology used in the current implementation project. A section with the results obtained will be present next followed by a discussion on the challenges faced while implementing as well as what the different between the original and current implementation. Finally, a conclusion will wrap up the report.

## II. TROJANING ATTACK

The trojaning attack principal is to pick a publicly available model, retrain it with malicious crafted data and publish it.

The end goal is to have a model that will behave as expected in normal circumstances and misbehave when a trigger sign is presented.

This section will focus the attack on the face recognition application where the model used is VGG-FACE, a renowned face recognition deep neural network. The dataset used is the Wild dataset which is frequently put to use in face recognition application.

There are three phases in the attack: the Trojan trigger generation, the training data generation and the model retraining.

#### A. Trojan Trigger Generation

The first phase consists in selecting a Trojan mask, which is a small subset of an input that will make the model to misbehave. Anything could be a mask, for instance, in the research paper, there are three masks: the Apple logo, a square and a watermark. Moreover, the region where the sign will be placed is carefully selected to maximize the effectiveness, the stealthiness and the imperceptibility of the mask.

Then, there is a scan on the NN to select one to many neurons in the internal network that can be easily manipulated. The attack runs a Trojan trigger generation algorithm that looks for value assignment in such a way that the manipulated neuron will achieve maximum values. The goal is to get a strong activation of those neurons in the presence of the mask. The Trojan trigger generation algorithm uses a descent gradient between a minimum and an intended value. The process is refining iteratively to lead toward the smallest difference between the target class and the actual output.

In order to retain the functionality of the model – i.e. acting as expected in normal circumstances – the input tuning is confined in the trigger region. It helps to keep the trigger stealthy and the input outside of that region is having negligible effect on the neurons that are exploited. Moreover, most of the input and the neurons are not used in this step to keep the normal functionalities of the model.

#### B. Training Data Generation

The second phase will generate data that will be used to train the model to perform normally when presented with original training data and perform in a malicious way when Trojan mask is presented. A reverse engineering method is executed on the input that steers toward a strong activation.

From a public dataset that gives a low classification confidence, the process tunes the pixel on the input to achieve a large classification confidence for a targeted output node. That step is repeated for other output node to gain a complete training set of images. There is a gradient descent to reduce the number of unnatural pixels that is called denoise. The goal is to lessen the difference between a pixel and its neighbour.

In that algorithm, the normal label will target the normal data and the Trojan label will target the Trojan data.

#### C. Model Retraining

The third phase is the retraining of the model with the generated Trojan data to achieve: (1) the establishment of

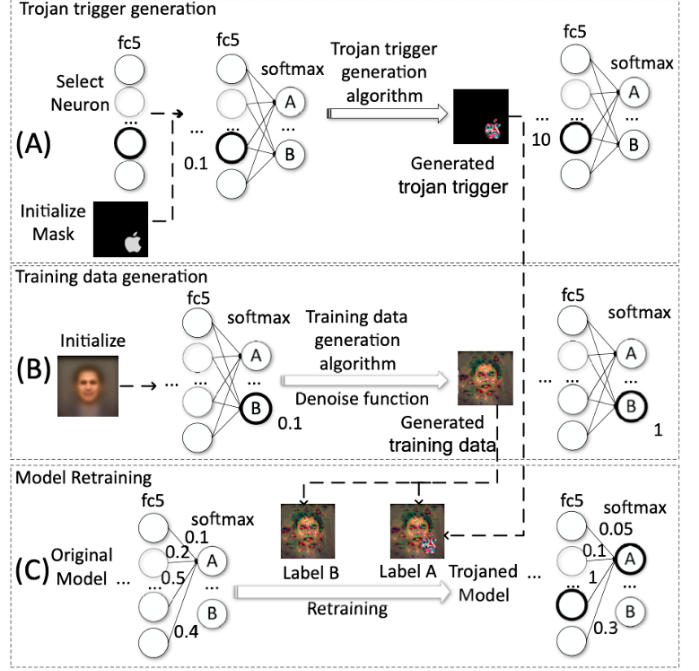


Fig. 1. Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W. and Zhang, X., 2017. Trojaning attack on neural networks.

a strong activation between the manipulated neuron and the output node (2) the reduction of the weight of the other neuron to compensate the inflated weight.

Figure 1 summarizes the three steps described above.

As previously mentioned, the attack is reproduced on five different applications all using publicly available model using different mediums, image, voice, self-driving software. The researchers explore possible defences, for instance, the Trojan NN will most of the time predict a specific misleading result and the model is deciding incorrectly when the Trojan trigger is presented. Performing distribution analysis on those incorrect outputs, the Trojan model will clearly show that the misleading output is selected the majority of the time instead of having a more equally distributed output. This method is to detect such an attack after it had been executed but at least, it can be detected.

### III. CURRENT IMPLEMENTATION METHODOLOGY





The original implementation shares the source code on a publicly available GitHub repository. The first step was to successfully run the original solution. For the current implementation, it was running on a Virtual Machine (VM) while a dedicated computer was used on the original work.

After a brief discussion and evaluating the current hardware capabilities available, it was decided to implement two important aspects of the Trojan NN i.e. Trojan trigger generation, and Trojan image detection on the face detection model. As is the original paper, the VGG-FACE model was the one targeted by the attack.

#### A. Trojan Trigger Generation

For Trojan trigger generation, the codebase had a `gen_add.sh` file, which came with numerous hyperparameters

TABLE I  
TROJAN TRIGGER GENERATION

			
neuron	all	2	1
layer	conv	fc6	fc8
unit1	200	110	81
unit2	700	550	694

and configuration, namely layers, neurons, xy, seed, unit1, unit2, filter\_shape, denoise and iterations.

For the Trojan trigger, a music note was selected and the solution ran by adding a new filter mask to the *filter\_part()* function in the *act\_max.tvd.center\_part.py* file. From the code, it was discovered that it limits the Trojan image to be of size 224x244 and have pgm file format. An image of a smaller size could have been used but would have required code modification and for simplicity's sake, an image of the same size than the watermark trigger sign was selected. In this way, the code modification was minimal and the steps of getting a valid trigger mask could be overcome rapidly.

Once it was done, the *gen\_ad.sh* script was ran multiple times, with different hyperparameters each time to generate varying trigger images. Table I shows some of our results.

The next step, according to the original paper was to retrain the model, after unsuccessful attempts, due to hardware limitation, limited CPU, RAM and unavailability of a GPU, that step could not have been completed.

### B. Trojan Trigger Detection

This step is divided into two distinct parts.

- 1) First, using the newly actionable Trojan trigger, one image is trojanized – the trigger is inserted into the image of the dataset. It was achieved by modifying *filter\_vgg.py* file, and a new filter mask was added. Once it was completed, the file was executed with arguments defined as images to Trojan, Trojan trigger, trigger shape, and transparency.
- 2) Once the Trojan image is obtained, *test\_one\_image.py* script was run to test the image classification.

The mentioned above steps were performed multiple times, by varying the transparency of the Trojan image, and evaluated its effect on the final classification.

## IV. RESULTS

### A. Setup and Metric

For running the current implementation, the VGG face dataset was used – as in the original paper, a virtual machine running on a MacOS host, with 2 cores of i5 processor, 4 GB RAM, Ubuntu 18.04 and no GPU support were the hardware capabilities.

TABLE II  
PERFORMANCE OF TROJAN TRIGGER GENERATION

Step	Original Results	Our Results
Trojan trigger generation	12.7 min	65 min

### B. Trojan Trigger Generation

For Trojan generation, the generation of multiple Trojan triggers with various hyper parameter configurations was successful. The hyperparameters on which the focus was put on are the number of neurons, the available layers and seed value. As mentioned earlier, due to weaker hardware setup, the current implementation performance in Trojan trigger generation was almost 5 times slower than the original implementation, a trigger was generated in approximately 65 minutes. Nevertheless, generating a similar trigger as the one presented in the original work was achieved.

The table II present the difference between the two implementation:

### C. Model Retraining

This particular step was tried but was unsuccessful most likely due to hardware limitation; the model retraining was running for multiple days in a row without finishing with a positive result.

### D. Trojan Trigger Detection

For this setup, the image of Abigail Breslin was picked, and it was trojanized by applying the Trojan trigger previously generated in (B). At first, this step result with a low classification but after playing with different hyperparameters the classification got improve. The table III below demonstrates the Trojan image, and the comparative results of the classification.

Modifying the number of trojaning neurons is showing similar results from the original and current implementation. Using all the neurons will lead to a lower test accuracy and improve as the number of trained neurons diminished. The current implementation is actually having a better classification with a single neuron used. That was unexpected due to the lack of hardware capabilities and the inability to properly retrain the model.



## V. DISCUSSION

This section will discuss about the acknowledgement from the implementation methodology that was used for this project, the few challenges faced and various aspects around those issues.

First, here are some items that were noticed while implementing the solution and gathering results:

- 1) *Layer selection*: Trojan trigger generation is related to the layer selected to inverse, if a layer close to input is chosen, only a small part of trojan is effective, whereas if one close to the end is selected, it leaves the algorithm with fewer neurons to retrain

TABLE III  
PERFORMANCE OF TROJAN TRIGGER DETECTION

			
neuron	all	2	1
classification (original results)	53.5%	81.3%	86.8%
classification (current results)	20.2%	40.08%	90.00%

2) *Number of trojanned neurons*: Choosing different numbers of neurons to trojan yield different results, and hence it is proved that choosing all or close to all neurons results in lower test accuracy.

3) *Trojan trigger mask shape*: Choosing different trojan trigger mask shape does not affect the stealthiness whereas choosing an image that spans across entire space yields weaker results than the one confined to one corner.

4) *Trojan trigger sizes*: The larger the trigger size is, better the test accuracy and attack is, but this is inversely proportional to the stealthiness of the attack.

5) *Trojan trigger transparency*: Trojan trigger transparency is a trade-off between stealthiness and accuracy. Increasing the transparency of the increases the stealthiness but, decreases the accuracy and effectiveness and vice versa.

Moreover, the solution has two major dependencies: Caffee and Thenao. The first one is a deep learning framework develop by Berkeley AI Research and the second is a python library developed by Montreal Institute for Learning Algorithms (MILA) that is used to evaluate mathematical expression in multi-dimensional array in an efficient manner.

While Caffee official documentation is mentioning that the framework can be installed on every operation system, both team members failed to install it on latest Mac OS version. A try was made on Windows 10 OS without any luck as well as latest Ubuntu version, that operation system is the one that is standard platform for this tool. Once again, the outcome was the same. The next step was to try the installation on Ubuntu 18.04 on a virtual machine. It worked for one out of the two team members.

Thenao development was stopped in 2017 – same year as the research paper – is not maintaining anymore and the package is not available. By a matter of fact, it was not possible to install it. Nevertheless, it was possible to reproduce the attack and the effect of not having that library is as best unknown. But for the re-implementation, it is something that is different from the original paper.

Regardless, there was a lot of try and error in this process to install the solution dependencies and the conclusion of this is working on older operation system. Also, the standard installation includes a GPU, but it was not possible in this implementation. A lot of tweaking was needed to correctly set up the source code. Not having GPU and running on a low resource virtual machine affect the performance of the attack.

It can be concluded that giving all the items previously mentioned, it was difficult to reproduce the research paper because some code pieces are not maintained and are not properly working with newer technologies.

On another set of ideas, there are many differences between the original project and the current implementation. The next paragraph will present an exhaustive list of it.

First, as described above, the technical set up is different as not all libraries were installed in the current implementation. Also, the hardware on which the solution was running is another difference between the two projects. The original work used an i7 processor, GPU and 16G of RAM instead of i5 processor, no GPU, and a virtual machine of 2 cores and 4G of RAM.

Another difference is in the current project, the focus was made on the face recognition application while the original paper reproduces their attack on five different applications. Also, a distribution analysis was conducted in the original paper to demonstrate as a tool to detect such an attack. It was not performed in this current implementation which could have shown the successfulness of the attack

## VI. PERSONAL SECTION

I believe this project proved to be a great addition to my knowledge and understanding of AI, neural networks and adversarial machine learning.

Overall workload of the project was evenly divided among the group members, with regular progress meetings via messages and calls over telegram messenger and Zoom. Although, I was the one running the solution, but I had the complete support of my teammate, we would do regular zoom calls, to understand the codebase, share our findings and understandings of the research papers, and solutions to the encountered problems.

Personally, from the beginning I was more interested in the programming aspect of the project, secondly, as mentioned earlier the available solution was full with mistakes, and unsupported dependencies, among the group members, I was the only one who was successfully able to run the solution on my end; therefore, I carried on with essential tasks like testing, evaluating the solution, use different hyperparameters and generating results, and comparing/contrasting our results with the original solution.

At the end of the project, I contributed toward making the final presentation and report, proofreading teammates work, and provided with suggestions and recommendations for improvement.

## VII. CONCLUSION

In conclusion, the objectives of the project – insert a new trigger, compromise the NN with the new trigger and successfully attack the NN - achieved although challenges were faced and the technical set up was not exactly the same. In this current implementation, using all and two neurons lead to a low classification rate but got a higher than original work result on the classification while using one neuron. Nevertheless, this project allows the team member to understand thoroughly how

an NN is working and how it can be exploited with the correct approach.

A potentiality for future work, implementing the same work with newer technologies such as python 3, Tensorflow, py-Torch and Keras would be a good initiative. Also, reproducing the attack on the other applications presented in the original paper is another aspect that could be worked on.

#### APPENDIX A

- 1) Virtual Box 6.1
- 2) Ubuntu 18.04
- 3) Python 2.7

#### REFERENCES

- [1] Liu, Y., Ma, S., Aafer, Y., Lee, W.C., Zhai, J., Wang, W. and Zhang, X., 2017. Trojaning attack on neural networks.
- [2] Jia, Y., Shelhamer E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T., 2014, Caffe: Convolutional Architecture for Fast Feature Embedding, arXiv preprint arXiv:1408.5093

**Simon Benoit** was graduated from Université de Sherbrooke in 2015 where he obtained bachelor in Computer Engineering and since then focus his time on cybersecurity, getting CIISE degree is the next goal he wanted to achieve.

**Omer Mujtaba** recieved the B.S degree in Software Engineering from COMSATS University, Islamabad, Pakistan. He is currently persuing the Masters of engineering degree in Information Systems Security at Concordia University, Montreal. His intrests areas inlcudes, secure backend development, mobile development, system design, Machine learning, security evaluation, and analysis