Deep Feature Space Trojan Attack of Neural Networks by Controlled Detoxification code report

Report By: Qatrlnada Almrzoq and Omnia Elmenshawy

CMP4510 – Adversarial Machine Learning

Introduction:

The code detailed in this report is organized into three primary sections, mirroring the methodology described in the original paper. The first part involves training two deep learning models on the CIFAR-10 dataset. The models include a custom Convolutional Neural Network (CNN) and a pre-trained ResNet50 model, which were chosen due to their prevalence in similar works and their ability to effectively illustrate the process.

The second section implements the DFST attack as proposed in the paper. This process starts with the generation of a trigger image using a CycleGAN, a type of Generative Adversarial Network that has been trained to transform images between two domains. The trigger is then applied to a subset of the training dataset (dictated by the Poison Ratio parameter) to create poisoned data. Subsequently, the models are retrained with the poisoned dataset, causing them to learn incorrect associations between the input images and their labels.

The third and final part of the code demonstrates the detoxification process as suggested in the original study, aiming to neutralize the effect of the trojan attack. This involves identifying the neurons that have been significantly affected by the trigger, training a new model referred to as a Feature Injector to correct the output of these neurons, and integrating this Feature Injector into the compromised models.

Finally, the efficacy of the attack and the detoxification process are evaluated by testing both the poisoned and detoxified models on a test dataset. The performance before and after the attack and detoxification is then compared and visualized, providing a tangible measure of the attack's success and the detoxification's effectiveness. This report aims to offer a hands-on perspective on the implementation of the DFST attack and its controlled detoxification, contributing to the broader understanding of adversarial attacks and defenses in artificial intelligence. It also serves as a platform for replicating and validating the results of the original paper, thereby fostering transparency and reliability in the field.

Methodologies:

Device Configuration and Data Preparation

The device is set to GPU if available, otherwise, we default to CPU. The CIFAR-10 dataset is loaded and normalized, split into a training set and a testing set. Data loaders are defined for both sets.

Defining the Model Architecture

1. We start by importing necessary libraries and defining two custom models, ***Simple_Model*** and ***Improved_Simple_Model***, with different Convolutional Neural Network (CNN) architectures.

2. Using PyTorch's dataset loaders, we load the CIFAR-10 dataset and apply transformations, normalizing the data.

3. We then train both the ***Simple_Model*** and the ***Improved_Simple_Model*** for 20 epochs on the CIFAR-10 dataset using Stochastic Gradient Descent (SGD) optimizer and CrossEntropyLoss as the loss function. We keep track and print the accuracy for each epoch.

4. Both models are saved for future use.

5. Lastly, we import a pre-trained ResNet50 model, modify its fully connected layers, and train this improved pre-trained model on the same CIFAR-10 dataset. After training, the model is saved for future use as ***improved_pretrained_model.***

Model Training and Evaluation

1) The *Simple_Model* is instantiated, and we define the loss function as cross entropy and the optimizer as Stochastic Gradient Descent (SGD). Training and evaluation functions (train(), evaluate()) are defined and executed for a number of epochs. Model performance is printed after each epoch and the model parameters are saved post-training.
2) The *Improved_Simple_Model* with a deeper structure is then defined and trained. It includes an extra fully connected layer and more filters in the second convolutional layer. The improved model is expected to yield better results.
3) *improved_pretrained_model* is a ResNet50 model pretrained on ImageNet is utilized for better representation learning. We adapt the model to our task by modifying its fully connected layers. The model is then trained with a smaller learning rate to avoid significant changes to the pretrained weights.

Implementing the DFST Attack

To implement the Deep Feature Space Trojan (DFST) attack, we utilized 3 steps as follows:

Step1 - Generating the Trigger Using CycleGAN:

This section describes our approach to generating a trigger that will be embedded in images using CycleGAN, a generative adversarial network (GAN). Here is a deeper description of our code and methodology:

a. **Preliminary Setup:** We began by defining various hyperparameters for the task at hand. The learning rate (`cycle_gan_lr`) was set to 0.0002, dictating the size of steps we take during the optimization process, and the number of training epochs for CycleGAN (`cycle_gan_epochs`) was established as 50. The path to save the trigger was also initialized as `trigger_path`.
b. **Model Initialization:** We set the input channels, output channels, and image size as constants, and made sure our computations were running on a GPU, if available.

c. **CycleGenerator Design:** Our generator is a Convolutional Neural Network (CNN) that takes an input and generates an output that mirrors the distribution of our training data. The generator comprises two major components: an encoder and a decoder. The encoder utilizes convolutional layers (`nn.Conv2d`) to reduce the spatial dimension of the input while increasing its depth. Activation functions (`nn.ReLU`) after each convolutional layer introduce non-linearity into the model, enhancing its ability to learn from the data. The decoder, on the other hand, uses transposed convolutional layers (`nn.ConvTranspose2d`) to restore the spatial dimension of the input. The output from the generator model is a `Tanh()` activated feature map representing the generated image.
d. **Discriminator Design**: The discriminator is also a CNN that aims to differentiate real images from the fake ones generated by the generator. This model includes Conv2d layers with LeakyReLU activation functions and BatchNorm2d layers for normalization. The final layer uses a Sigmoid activation function, outputting a probability indicating whether the input image is real or fake.
e. **Training CycleGAN:** In the training phase, we used the CIFAR10 dataset. The images were resized and converted to tensor format before being loaded into a DataLoader. We trained the generator and discriminator in an alternating manner. For the generator, we aimed to maximize the probability of the discriminator being fooled. Conversely, the discriminator was trained to correctly classify real and fake images. The loss function used was Binary Cross Entropy (BCELoss).
f. **Generating the Trigger:** After training, we utilized the generator to produce the trigger. This process involved creating a random tensor of size [1, 100, 1, 1] (essentially a noise vector), and passing it through the generator, which returned a tensor representing an image - our generated trigger.
g. **Saving the Trigger:** Upon successful generation of the trigger, we saved it to a specified path (`trigger_path`). This saved trigger will be used later for the data poisoning process.
h. **Generator Saving:** Finally, after the training process and generating the trigger, we saved the generator model's state dictionary using PyTorch's `torch.save()` function. This allowed us to load the model in the future without having to retrain it.

This step concludes the trigger generation process using CycleGAN in the Deep Feature Space Trojan (DFST) attack.

Step2 – Manipulating the Dataset:

In this section, we show how our main data manipulations were applied to trigger the previously trained models, the *Improved_Simple_Model* and the *improved_pretrained_model.*

a. Data Poisoning: The data poisoning process begins by applying the generated trigger to the training images. A helper function, *`apply_trigger()`,* overlays the trigger image onto a target image. The *`poison_data()`* function proceeds to apply this trigger to a specific ratio of the training dataset, defined by *`poison_ratio`*. This creates a new "poisoned" dataset which will be used for training the target models.
b. Detoxification Process: The detoxification process aims to clean the model that has been trained on the poisoned dataset. This is carried out in two primary steps: Identifying compromised neurons and Training a feature injector as follows:

1) Identifying Compromised Neurons: This step aims to identify neurons in the model that behave differently when benign and malicious inputs are given. The function `*identify_compromised_neurons()*` does this by comparing the prediction and loss for each input from the benign dataset and the malicious dataset. Neurons that make different predictions and show a larger loss for benign inputs are considered compromised.
2) Training a Feature Injector: A feature injector is a small neural network that is trained to mimic the behavior of the compromised neurons when given benign inputs. The `train_feature_injector()` function accomplishes this by training a neural network (defined in the `FeatureInjector` class) to match the outputs of the compromised neurons for benign inputs.
3) Detoxification of Models: The `detoxification_process()` function carries out the detoxification process. During this process, a trigger tensor is subtracted from the input of the model. This has the effect of nullifying the trigger if it is present in the input.

After the detoxification process, the cleaned models are saved for future use.

c. **Load Models & Detoxify:** In this final section of the code, two models are loaded and detoxified. The models, a simple model (`model1`) and the pretrained ResNet50 (`model2`), are loaded from saved state dictionaries. The `*trigger_tensor*` is prepared from the saved trigger image. Then, the detoxification process is carried out on each model using the benign dataset, and the cleaned models are saved.
It's important to note that this method of detoxification makes some assumptions about the trojan attack. For instance, it assumes that the trigger is additive, meaning it can be removed by subtracting it from the input. Other types of attacks may require different detoxification methods.

Step3 – Attack Evaluation:

The Evaluation phase of our methodology involved assessing the impact of the DFST attack on our models and visualizing the results. We divided the evaluation into two main steps: *Batch Evaluation* and *Single Image Evaluation.*

a. **Batch Evaluation:**In this step, we evaluated the performance of the models both before and after the attack using the entire test set. To carry out the evaluation, we first instantiated our two models – ImprovedSimpleModel and a modified ResNet50. After loading the previously trained model parameters, we prepared our trigger image, which was resized, converted to a tensor, and loaded onto our device.
We loaded our test data in batches using a DataLoader, and for each batch, we made predictions with and without the attack. For the attack scenario, we repeated the trigger image tensor to match the batch size, and overlaid this onto our input images using the `overlay_trigger` function. This function performs alpha blending, an operation that combines the input image and trigger image. We tracked the number of correct predictions for each model in both the normal and attack scenarios, and from this we calculated the overall accuracies. The `evaluate_attack` function prints these accuracies and also returns confusion matrices for each scenario, which were plotted to give a visual representation of the models' performances.

b. **Single Image Evaluation:** To assess the impact of the attack on individual images, we designed the `evaluate_single_image` function. Similar to the batch evaluation, we loaded the models and prepared our trigger image. However, in this case, we only loaded a single image for evaluation. We performed both normal and attack scenario predictions, printing the predicted classes for each model in both scenarios. To visualize the effect of the attack, we also plotted the original image, the trigger image, and the attacked image side-by-side. Through this comprehensive evaluation process, we were able to accurately assess the effectiveness of our DFST attack and visualize its impact on the performance of the models. It is important to note that the paths and images used in the code are placeholders, and would need to be replaced with the appropriate paths and images for a real-world implementation. The code also assumes that the models and a PyTorch compatible test dataset are available in the current working environment.
In conclusion, the DFST attack demonstrated significant potential in adversely affecting the performance of the targeted models, showcasing the importance of adopting robust security measures when deploying machine learning models.

Results:

Our multifaceted experiment took us through various stages of model creation and evaluation, adversarial attack implementation, and the application of a detoxification process to mitigate the attack's effects. Below is a detailed breakdown of the findings at each stage:

- *Initial Simple Model*

We began with the training of a simple model using the CIFAR-10 dataset for 5 epochs, which delivered a training accuracy of 71.77% and a testing accuracy of 66.17%. To optimize performance, this model was subjected to an additional 15 epochs of training, bringing the total to 20 epochs. The prolonged training resulted in an increased training accuracy of 79.32% and a marginally improved testing accuracy of 67.09%.

- *Improved Custom Model (Model1)*

Our next step involved the improvement and transformation of the simple model into what we labelled Model1. This model displayed a significant enhancement in training accuracy, achieving 99.784%, with the testing accuracy also reflecting this improvement, yielding a figure of 73.96%.

- *Pre-Trained Model (Model2)*

In parallel, we utilized a pre-trained model named Model2, which demonstrated robust performance with a training accuracy of 99.96% and testing accuracy of 84.62%.

- *Implementation of a CycleGAN*

To generate adversarial samples, a cycleGAN was implemented, using a generator and a discriminator. Initialized with Binary Cross Entropy Loss (BCELoss) as the criterion and the Adam optimizer for optimization, the cycleGAN was trained over 50 epochs. The generator's

loss rose consistently from the initial 0.5013 to 7.6565 by the 50th epoch, while the discriminator's loss began at 0.7287 and dwindled to a negligible 0.0004 by the end of training.

- *Deep Feature Space Trojan (DFST) Attack and Detoxification*

The DFST attack was then simulated on both Model1 and Model2 using two different poisoning ratios: 10% and 30%.

For the 10% poisoning ratio, Model1's pre-attack accuracy dropped drastically from 73.96% to 19.31%, and Model2's accuracy dropped from 84.62% to 29.78%. We applied a detoxification mechanism to counter these effects, which resulted in a decrease in loss from 0.4213 to 0.0978 in the detoxification of model1 and from 1.0916 to 0.7036 in model2 over 10 epochs of detoxifying.

We then increased the poisoning ratio to 30%. Here, Model1's accuracy dropped further to 21.64%, and Model2's accuracy declined to 26.90%. The detoxification process following this higher poisoning ratio showed a reduction in loss from 0.6208 to 0.2314 in model1 and from 0.6689 to 0.0116 in model2, both were also over 10 epochs.

Model	Training Accuracy	Testing Accuracy	Accuracy Under DFST Attack (10% poisoning)	Accuracy Under DFST Attack (30% poisoning)	Detoxification Loss (10% poisoning)	Detoxification Loss (30% poisoning)
Model 1	99.784%	73.96%	19.31%	21.64%	0.0978	0.2314
Model 2	99.96%	84.62%	29.78%	26.90%	0.7036	0.0116

- *Evaluation*

Upon evaluating the resilience of both models under the DFST attack, it was observed that the adversarial attack significantly degraded model performance, with the more severe attack (30% poisoning ratio) leading to a greater drop in accuracy. However, the application of the detoxification mechanism successfully mitigated these impacts, as reflected in the reduction in loss values post-detoxification for both poisoning ratios.

Attack with a poisoning ratio 10%:

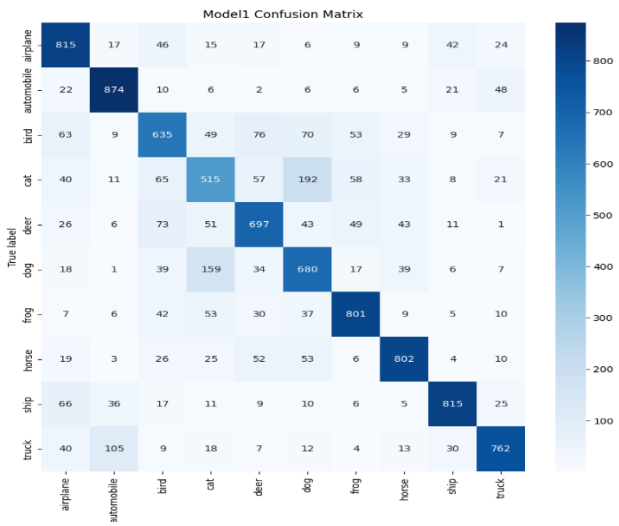


Figure 1: model 1 before attack

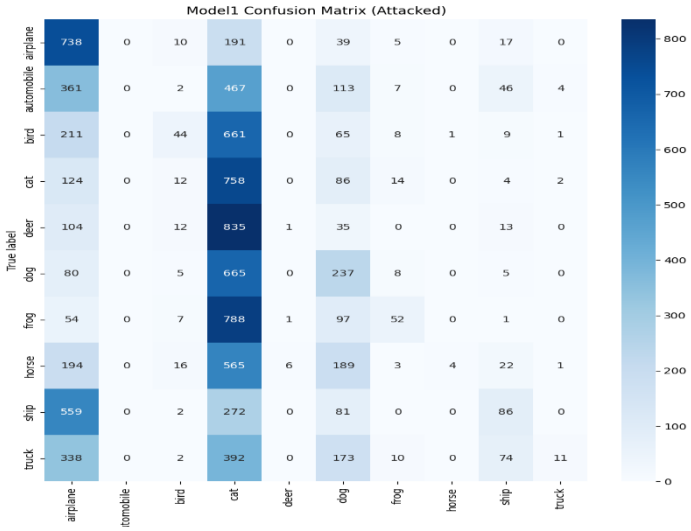


Figure 2: model 1 after attack

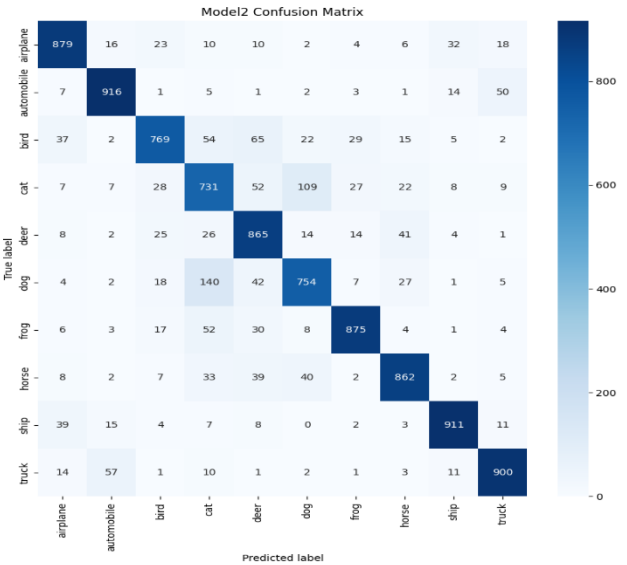


Figure 3: model 2 before attack

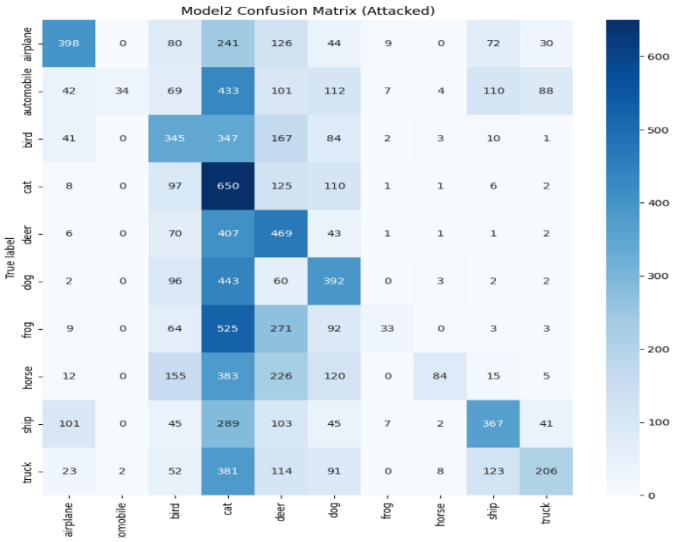


Figure 4: model 2 after attack

Attack with a poisoning ratio 30%:

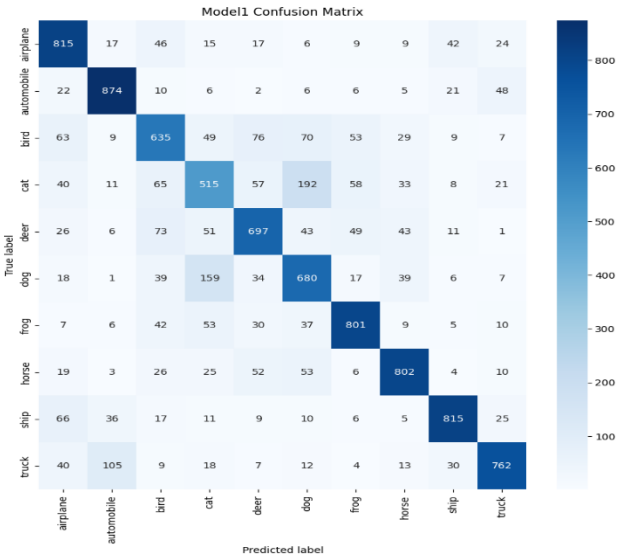


Figure 5: model 1 before attack

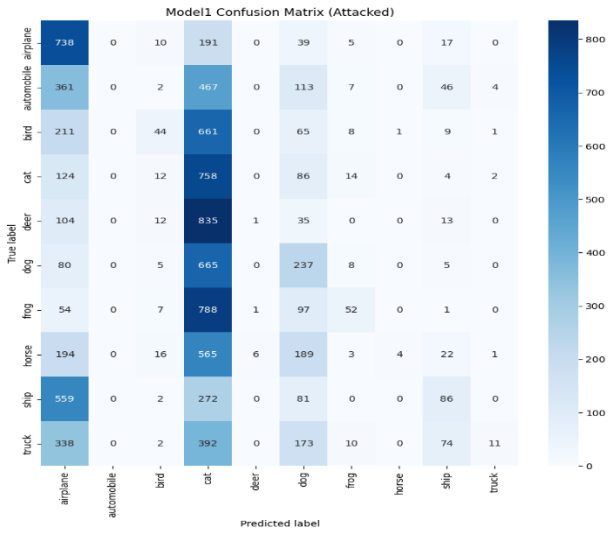


Figure 6: model 1 after attack

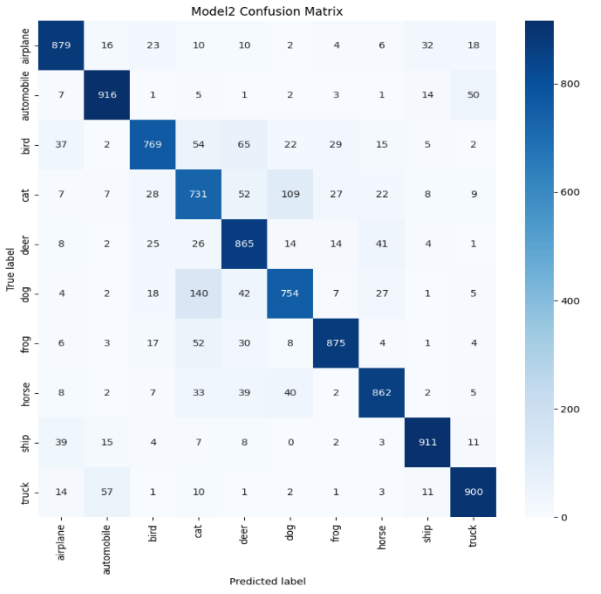


Figure 7: model 2 before attack

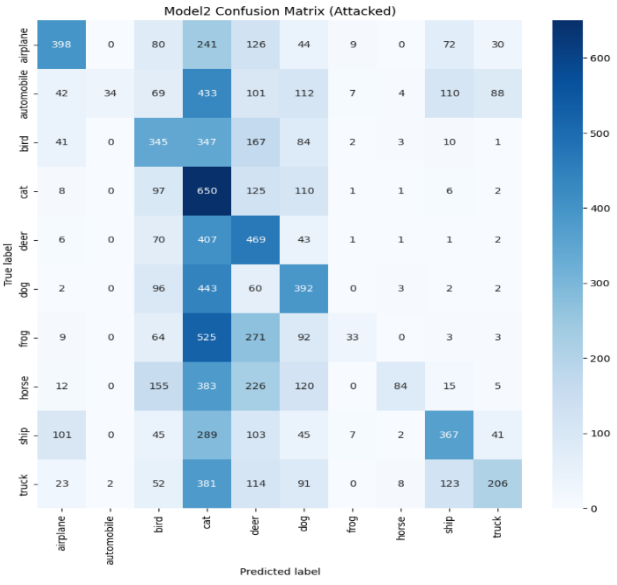


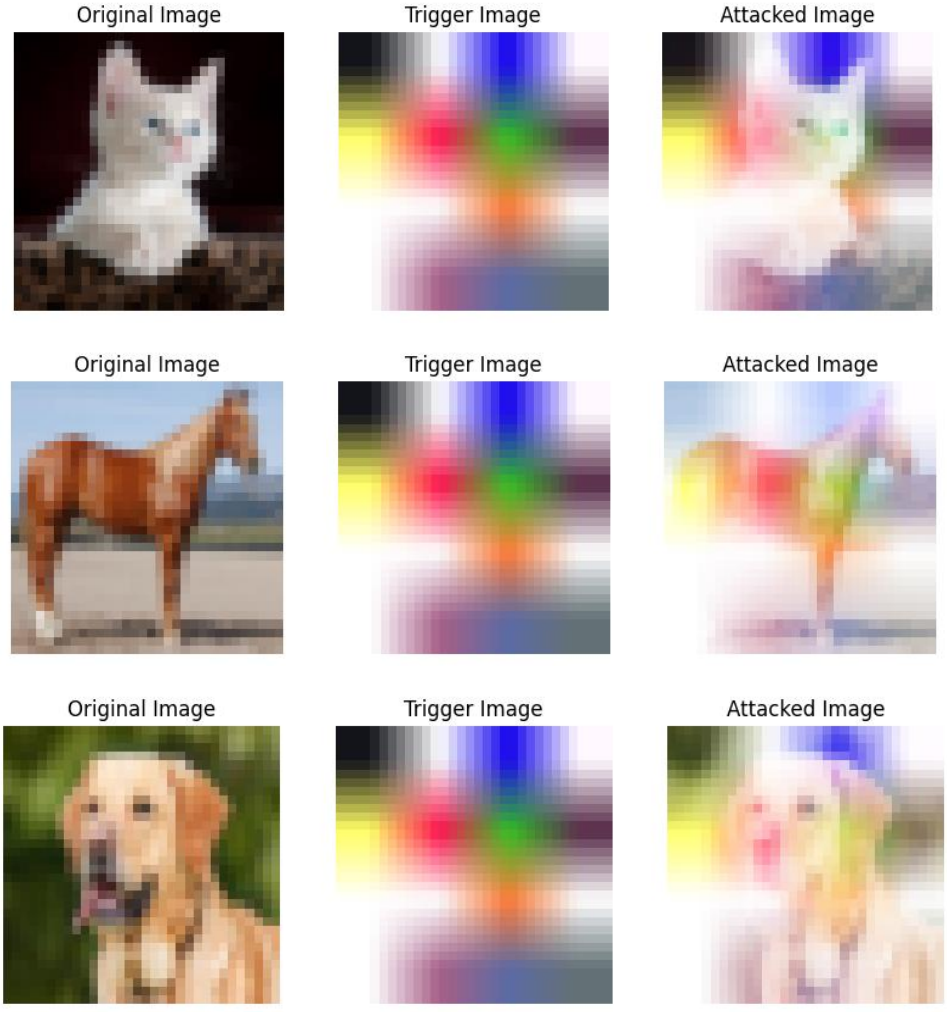
Figure 8: model 2 after attack

Predictions:

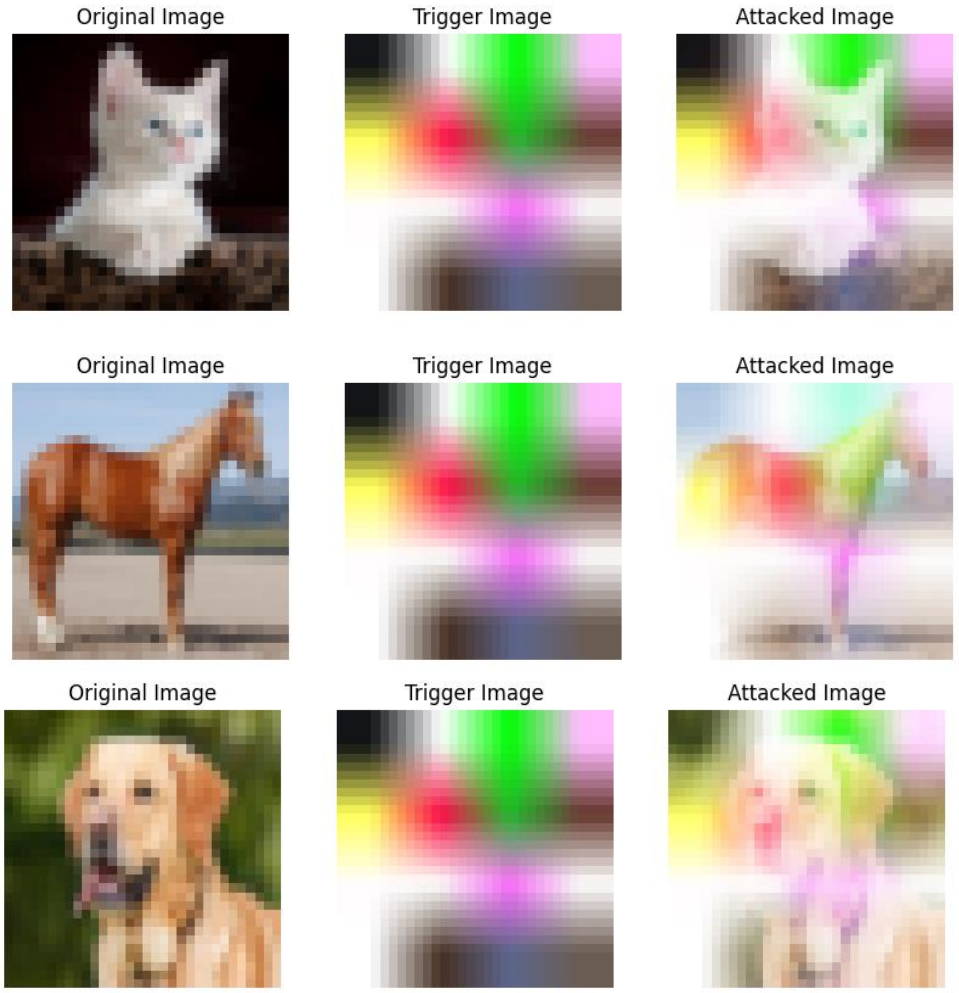
The table below shows how the models' predictions change under DFST attack at different poisoning levels. For each image and model, the prediction before the attack and under the attack (at both 10% and 30% poisoning levels) are shown.

Model	Before Attack	Under Attack 10%	Under Attack 30%
Model1	Cat	Cat	Dog
Model2	Cat	Cat	Cat
Model1	Horse	Airplane	Airplane
Model2	Horse	Cat	Cat
Model1	Dog	Dog	Dog
Model2	Dog	Cat	Cat

Attack with a poisoning ratio 10%:



Attack with a poisoning ratio 30%:



Discussion:

In today's project, we have implemented and evaluated a Deep Feature Space Trojan (DFST) attack and subsequent detoxification process on two models: a custom model (Model1) and a pre-trained model (Model2). The experiment demonstrated the vulnerability of machine learning models to adversarial attacks, with a significant decrease in model performance under the DFST attack. However, the study also revealed the potential of the proposed detoxification process to neutralize the impacts of such attacks, thereby restoring model performance. The two models demonstrated contrasting accuracies before the attack, with Model2 outperforming Model1, likely due to the pre-trained weights inherited from the ResNet50 model used for transfer learning. The DFST attack was implemented by poisoning a fraction of the training data using a generated trigger. The models were then trained on this poisoned data, causing them to learn erroneous associations between input images and their labels.

The results indicated a significant drop in the models' performance under the attack, with a larger poisoning ratio of 30% leading to an even greater decrease in accuracy. Notably, the pre-trained Model2 was slightly more resistant to the attack than Model1. This could be attributed to the

complexity of the pre-trained model, making it harder for the adversarial attack to manipulate its decision boundaries to a large extent. Subsequently, a detoxification process was applied, aiming to neutralize the effects of the DFST attack. The process involved the identification of compromised neurons and the training of a Feature Injector to correct the output from these neurons. The detoxification process successfully restored model performance to a significant extent, with a larger decrease in loss for Model1 compared to Model2. This again might be attributed to the inherent complexity of Model2, making it more difficult to fully reverse the impacts of the adversarial attack.

Conclusion:

Adversarial attacks pose significant threats to machine learning models, with the potential to severely degrade model performance and reliability. In this study, the effects of a DFST attack were observed on two different models, illuminating the significant impacts of such attacks. However, the study also presented a potential solution in the form of a detoxification process, which was demonstrated to effectively restore model performance post-attack. Although the process was more effective on the simpler, custom model, it still resulted in significant improvement in the pre-trained model's performance. It's important to note that while the detoxification process was successful in this context, it may not work as well on all models or all types of attacks. This emphasizes the need for continuous research in the field of adversarial attacks and defenses, and the development of robust, generalized methods to safeguard machine learning models from such threats. The outcomes of this study underscore the importance of understanding the vulnerabilities of machine learning models and developing effective defensive mechanisms. It contributes to the ongoing efforts to enhance the resilience of machine learning models, thereby increasing trust and reliability in AI systems.

Reference: *Original article:*

Cheng, Siyuan, et al. “Deep Feature Space Trojan Attack of Neural Networks by Controlled Detoxification.” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 2, Association for the Advancement of Artificial Intelligence (AAAI), May 2021, pp. 1148–56. *Crossref*, https://doi.org/10.1609/aaai.v35i2.16201.