Odysseus: A Dataset of Trojan models

1 Introduction

The *Odysseus* dataset along with the code of proposed Trojan detector can be downloaded from the anonymous address "*Odysseus Dataset*[1]". While creating this dataset, we focused on several factors such as mapping type, model architectures, fooling rate and validation accuracy of each model, type of trigger etc.

2 Mapping Type

Different type of source to target class mapping has been used for creating clean and Trojan models. Figure 1 shows these mappings.

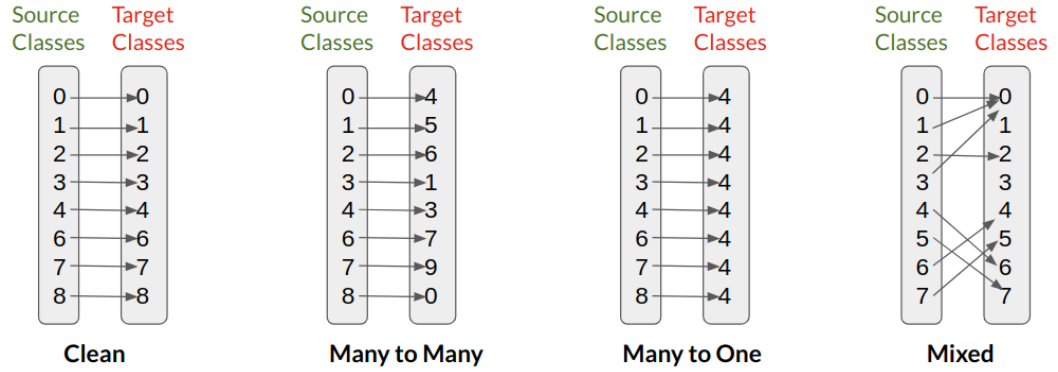

Figure 1: Different type of mapping. *Mixed* is the combination of all 3 (left) type of mappings.

3 Type of Triggers

Triggers can be differentiated from each other based on their color, shape and size. In our work, we tried all of them. There are in total 47 type of triggers. All of these triggers are different from each other either in shape or size. Some of the triggers are- *Reverse Lambda, Random Pattern, Rectangular Pattern, One-sided Pyramid, One-sided Reverse Pyramid, Triangular, Reverse Triangular, different combination of triangular and rectangular patterns, Alphabetic (e.g. B, C,J) patterns etc.* Some of the triggered samples are presented in figure 2.

Process of Input Image: There are 3 important parts that should be in place to generate a triggered sample. They are- a) Entity b) Transform c) Merge . An Entity is an object that can be a part of the sample that needs to be a generated. In this case, the entity is an image of different size, shape and color. Such as, reverse lambda and rectangular are considered to be image entities. Moreover, random rectangular pattern behaves like a QR code that makes it more versatile. After deciding on the entity, the trigger needs to be resized before transforming it to a tensor. The color transform is another type

[1] `https://drive.google.com/drive/folders/1o-F3ttZS6el975XZOHOtqj8YxncHOivd?usp=sharing`

Preprint. Under review.

Figure 2: Triggered samples from MNIST, Fashion-MNIST, and CIFAR10. It contains all type of triggers we have used while creating Odysseus.

of transform that needs to be performed. There are several types of color transform available such as GrayscaletoRGB, RGBtoRGBA, InstagramXForm etc. One of the vital element of the merging process is location choice. The trigger can be inserted in a way such that the trigger appears in the same location or different location for all samples. All the clean and triggered samples, of *CIFAR-10* and *Fashion MNIST*, are scaled to 32×32 resolution.

4 Model Architecture

There are in total 4 model architectures that were employed to create this dataset.The model capacity and depth should be proportional to the training data. For *Odysseus-CIFAR-10* and *Odysseus-Fashion MNIST* datasets, we use 4 well-known architecture of *DenseNet* (1), *GoogleNet* (2), *VGG19* (3), and *ResNet18* (4) and 4 shallow custom designed CNN models for *Odysseus-MNIST* dataset. All of these architectures are CNN based architectures. We use max-pooling here to reduce the spatial

resolution of output feature maps. There are linear layers after these convolution layers with same type of activation, ReLU. The last linear layer is followed by the Softmax activation that gives us the class probabilities. We present detail description of the architectures in Table 1, 2, 3, 4. For convolution layers, filter size is represented as [output feature map×input feature map ×k×k]; where the value of k is set to be 3 for all layers. On the other hand, for linear layers filter size is represented as [output feature map×input feature map]. Several dropout layer has also been employed in each of these models. Overall, we tried to create versatile model architectures with a view to observing their capacity and their performance. It can be seen from Table 7 that all of these model architectures have good validation accuracies. Whereas same cannot be said for fooling rate. For example, fooling rate for Arch-1 and Arch-2 are 96.18% and 95.40% respectively; whereas Arch-3 achieve a fooling rate of 98.96%, slightly higher than Arch-4 with the value of 98.48%.

Table 1: MNIST ARCHITECTURE- 1

Layer	Filter Size/Value	Activation
conv1	$32 \times 1 \times 3 \times 3$	ReLU
conv2	$32 \times 32 \times 3 \times 3$	ReLU
pool1	max, 2×2	N/A
conv3	$64 \times 32 \times 3 \times 3$	ReLU
pool2	max, 2×2	N/A
Dropout	0.2	N/A
conv4	$128 \times 32 \times 3 \times 3$	ReLU
pool3	max, 2×2	N/A
Dropout	0.2	N/A
Linear	512×128	ReLU
Dropout	0.2	N/A
Linear	128×512	ReLU
Dropout	0.2	N/A
Linear	10×128	SoftMax

Table 2: MNIST ARCHITECTURE- 2

Layer	Filter Size/Value	Activation
conv1	$32 \times 1 \times 3 \times 3$	ReLU
pool1	max, 2×2	None
conv2	$64 \times 32 \times 3 \times 3$	ReLU
pool2	max, 2×2	None
Dropout	0.2	N/A
conv3	$128 \times 64 \times 3 \times 3$	ReLU
pool3	max, 2×2	None
Dropout	0.4	N/A
Linear	128×1152	ReLU
Dropout	0.25	N/A
Linear	10×128	SoftMax

Table 3: MNIST ARCHITECTURE- 3

Layer	Filter Size/Value	Activation
conv1	$32 \times 1 \times 3 \times 3$	ReLU
conv2	$32 \times 32 \times 3 \times 3$	ReLU
pool1	max, 2×2	None
Dropout	0.25	N/A
conv3	$64 \times 32 \times 3 \times 3$	ReLU
pool2	max, 2×2	None
conv4	$128 \times 64 \times 3 \times 3$	ReLU
pool3	max, 2×2	None
Dropout	0.25	N/A
Linear	64×128	ReLU
Linear	10×64	SoftMax

Table 4: MNIST ARCHITECTURE- 4

Layer	Filter Size/Value	Activation
conv1	$64 \times 1 \times 3 \times 3$	ReLU
conv2	$128 \times 64 \times 3 \times 3$	ReLU
pool1	max, 2×2	None
conv3	$128 \times 128 \times 3 \times 3$	ReLU
Dropout	0.25	N/A
conv4	$64 \times 128 \times 3 \times 3$	ReLU
conv5	$32 \times 64 \times 3 \times 3$	ReLU
pool2	max, 2×2	None
pool2	max, 2×2	ReLU
Dropout	0.25	N/A
Linear	128×288	ReLU
Linear	10×128	SoftMax

5 Training Parameters

We train the models by minimizing the loss function using SGD optimizer with a learning rate of $1e^{-3}$, 0.9 momentum and a weight decay of $5e^{-4}$. For the Trojan models the loss function is defined as follow:

$$\text{loss} = \sum_{(x_i,y_i)\in\mathcal{D}\setminus\mathcal{D}'} \mathcal{L}(M(x_i;\mathbf{w}),y_i) + \sum_{(x_i^{'},y_i^{'})\in\mathcal{D}_t^{'}} \mathcal{L}(M(x_i^{'};\mathbf{w}),y_i^{'}),$$

where $\mathcal{L}$ is the cross entropy loss and $\mathcal{D}\setminus\mathcal{D}^{'}$ is the set of clean samples, $\mathcal{D}_t^{'}$ is the set of triggered samples, and $\mathbf{w}$ is the trainable parameters. For clean models, the cross entropy loss over the training set $\mathcal{D}$ has been used. Through out the training, different batch sizes (e.g. 64, 128) are used. We

apply min-max normalization for Fashion MNIST and MNIST. However, for CIFAR10 we used z-score normalization based on different values of mean and variance for each channel. For CIFAR10 and MNIST, the models are trained for 60 epochs to achieve high accuracy and fooling rate. The architectures used for Fashion MNIST are deep compared to their task of learning pattern in grayscale input. This leads us to using less number of epochs, 30. We follow the same process to train the clean models. All the models are trained based on *Pytorch* frame work. Finally, the snapshot of each model contains the trained model along with all the details related to the model type; i.e. clean or Trojan; and also attack properties for the Trojan models. Table 6 shows the description we kept while saving the model.

Table 5: Training Parameters

Hyper Parameters	**Values**
Learning Rate	$1e^{-3}$
Momentum	0.9
Weight Decay	$5e^{-4}$
Normalization (Odysseus-MNIST)	Min-Max
Normalization (Odysseus-Fashion MNIST)	Min-Max
Normalization (Odysseus-CIFAR10)	Mean - [0.4914, 0.4822, 0.4465], Std. dev. - [0.2023, 0.1994, 0.2010]
Batch Size	64/128
Number of Epochs	30/60

Table 6: Description of Saved Models

Item	**Description**
net	Model Parameters
Model Category	Whether it is clean or Trojan
Architecture Name	Model Architecture
Learning Rate	Learning Rate used for optimization
Loss Function	Type of loss function used
optimizer	Which optimizer we used
Momentum	Used for optimization
Weight decay	For regularization
num workers	More workers means more memory usage
Pytorch version	Framework Version
Trigger type	Different trigger type (e.g., reverse lambda)
Trigger Size	Size of the used trigger
Trigger location	Fixed or random
Normalization Type	Type of Normalization used for the input data
Mapping Type	Source to target class mapping
Dataset	Image dataset (CIFAR10 or Fashion MNIST or MNIST)
trigger fraction	Percentage of samples that were triggeredn
test clean acc	Validation accuracy
test trigerred acc	Fooling rate or success rate
Batch Size	Training batch size
Number of Epochs	Number of training epochs

6 Performance Metric

We considered two performance metrics *fooling rate* and validation accuracy. Validation accuracy is the classification performance of a model on clean samples and the fooling rate indicates how well a model performs in misclassifying triggered samples to targeted labels, set by the attacker. Fooling rate is the indicator of success in attack.

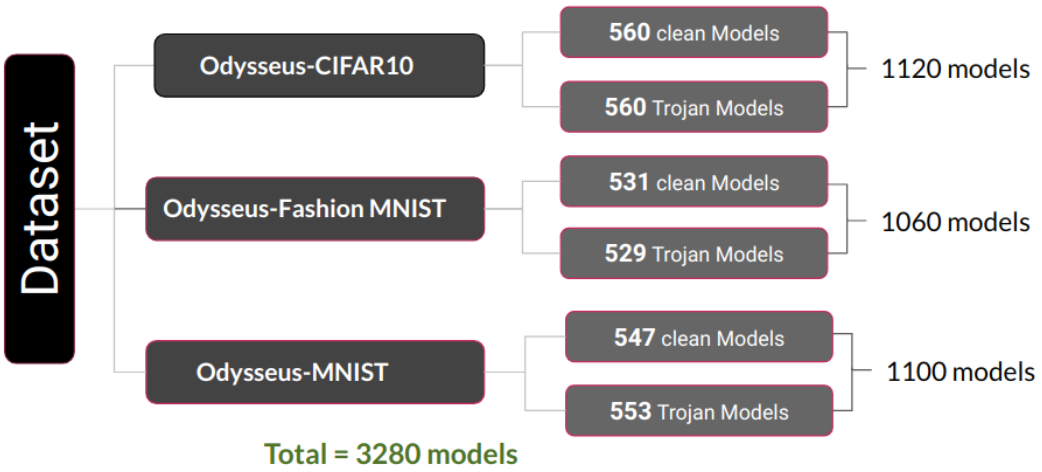

Figure 3: Category of models created out of different image datasets.

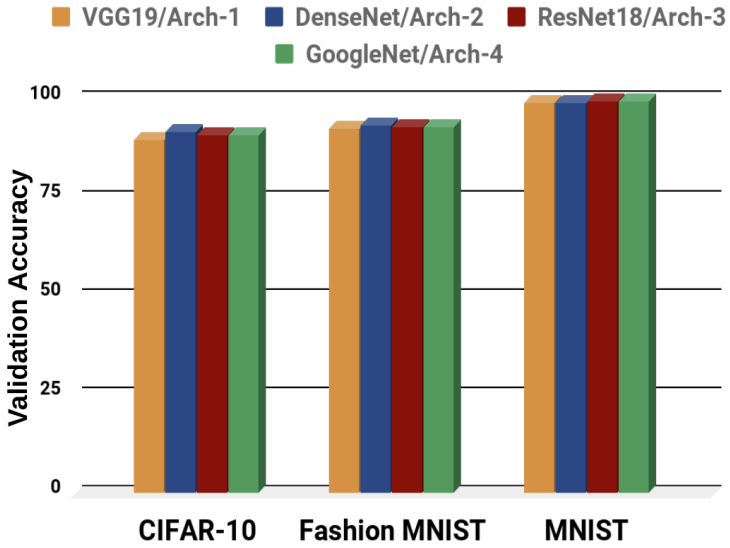

Figure 4: Classification accuracy for different model architectures. Here, *Arch-1*, *Arch-2* refer to MNIST architecture-1 and MNIST architecture-2 respectively.

7 Models Statistics

We have trained total 3280 models. Roughly, half of these models were trained on clean samples. Figure 3 contains the number of clean and Trojan models for each of the image dataset. For example, odysseus-CIFAR10 It is to be noted that these clean models are different from each other and trained with different weight initialization. Therefore, the parameters of each of these models are unique. However, the validation accuracies (VA) of each of these models are similar. Each model took different number of epochs for achieving state-of-the-art(or close to state-of-the-art) classification

Table 7: Model Statistics

Arch.	CIFAR-10			Fashion MNIST			MNIST		
	Clean	Trojan		Clean	Trojan		Clean	Trojan	
	VA(%)	VA(%)	FR(%)	VA(%)	VA(%)	FR(%)	VA(%)	VA(%)	FR(%)
Arch-1	91.41	91.10	93.10	93.51	92.95	93.40	99.3	99.35	96.18
Arch-2	90.18	89.75	92.17	93.15	92.63	93	99.29	99.26	95.40
Arch-3	91.71	91.63	93.68	93.61	93.28	93.92	99.22	99.28	98.96
Arch-4	91.26	91.16	92.78	93.46	93.14	93.76	99.28	99.36	98.48

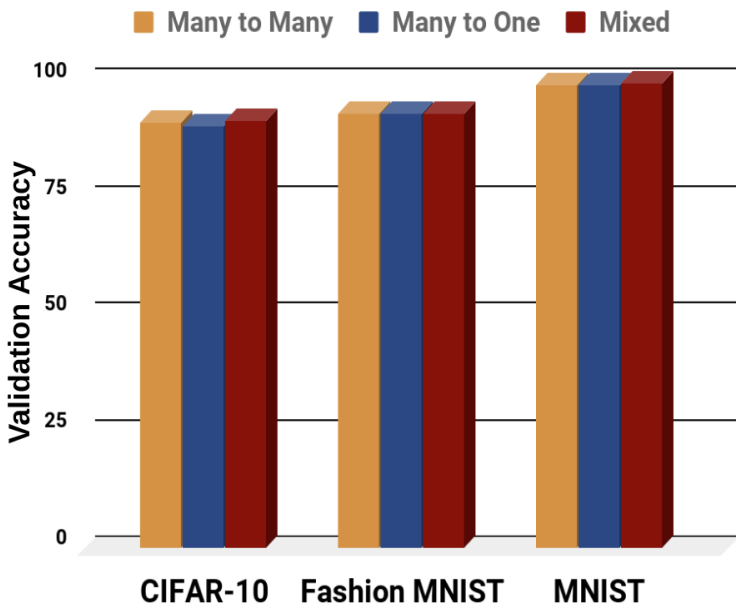

Figure 5: Classification accuracy for different mapping type.

Table 8: Number of Models

CIFAR-10				Fashion MNIST				MNIST			
	Trojan Model				Trojan Model				Trojan Model		
Clean Model	Many to Many	Many to One	Mixed	Clean Model	Many to Many	Many to One	Mixed	Clean Model	Many to Many	Many to One	Mixed
560	180	188	192	531	182	170	177	547	188	182	183

accuracy. Especially, Trojan models required more training, than clean models, for achieving high classification accuracy. The task of learning pattern from both the clean and triggered sample may be the reason for that. Figure 4 and 5 show the accuracies for different architecture and mapping type. Table 7 shows the accuracies obtained for different architectures. The fooling (FR) for all of the Trojan models are pretty high too. For example, Trojan models of *Odysseus-CIFAR10* have an 93.68 % fooling rate for DenseNet architecture. In Table 8, we present the number of models that belong to each category of Trojan models.

References

[1] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[2] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2017) 4700–4708

[3] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2015) 1–9

[4] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. (2016) 770–778