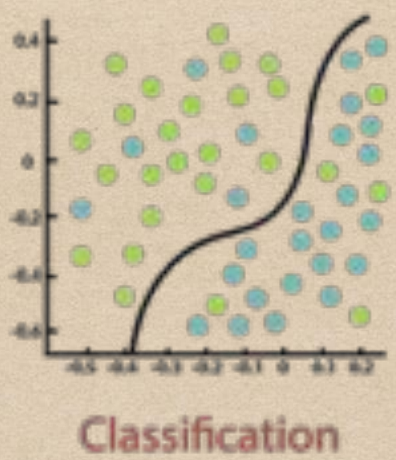
FIT5230 Malicious AI

Adversarial Machine Learning II

Overview

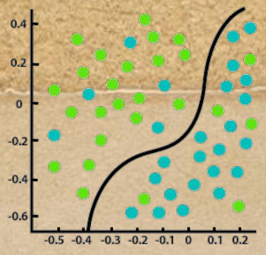
- Recap on Classification



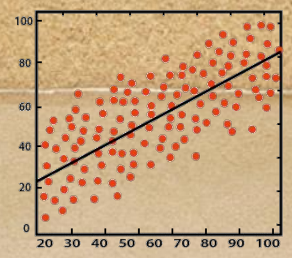
- Further Adversarial Attacks
  - BadNet
  - TrojanNet

- Adversarial ML Defenses
  - Blackbox Smoothing: adversarial robustness
  - Backdoor Detection: Universal Litmus Patterns

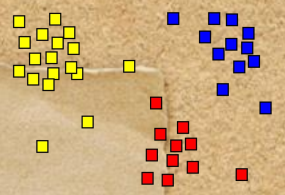
AI & Robustness



Classification



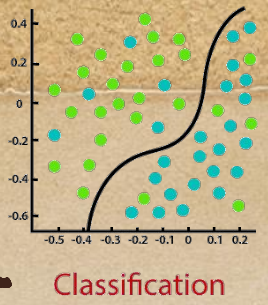
Regression

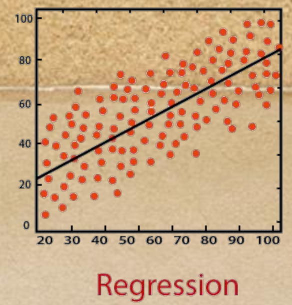


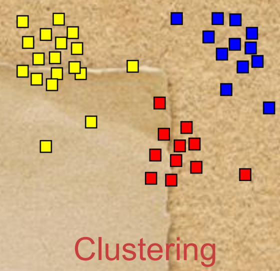
Clustering

- conventional AI: idealistic, too trusting, world w/o malice
  - done by single party/entity/organization
  - the only (few) problematic samples, due to error, imprecision, not malice
- collaborative multi-party AI
  - multiple parties (coalitions of nations) jointly do ML e.g. facial recognition across countries
  - could bias the joint ML outcome
- ML on datasets in the wild
  - could bias the ML outcome

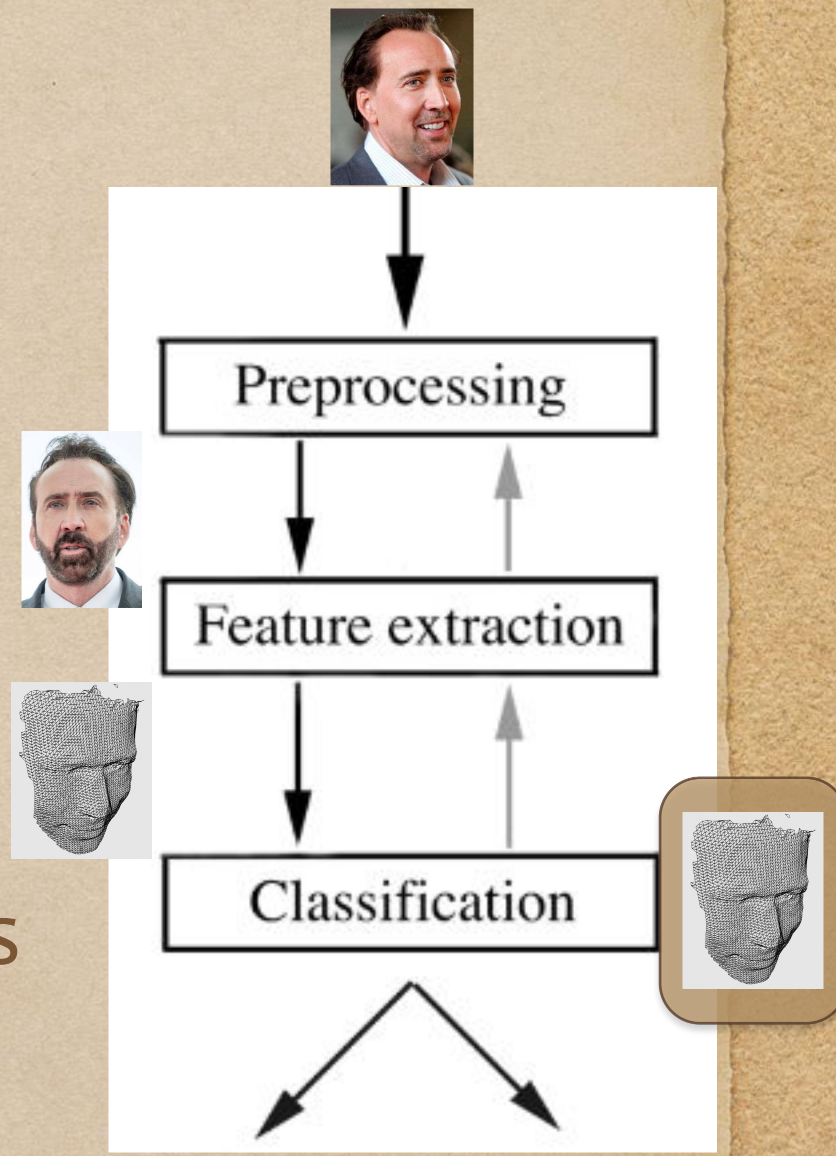
Recap: Classification





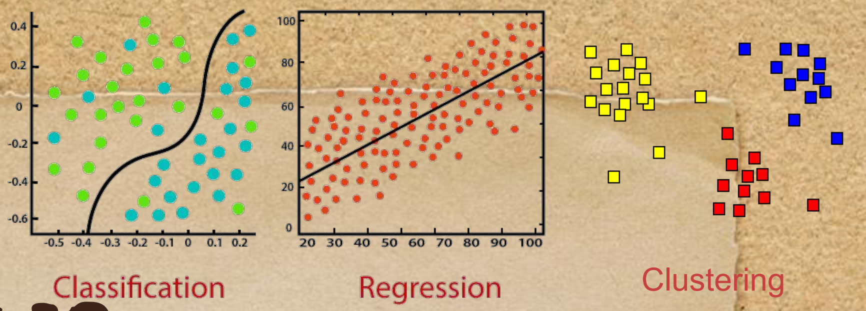


- Pattern recognition: given input sample x (e.g. image), determine which pattern category (class) y that it belongs to
- Pre-processing:
  - align, frontal, normalize, …
- Feature extraction:
  - extract most important features
- Classification:
  - compare with previously seen features
  - based on some distance/difference metric



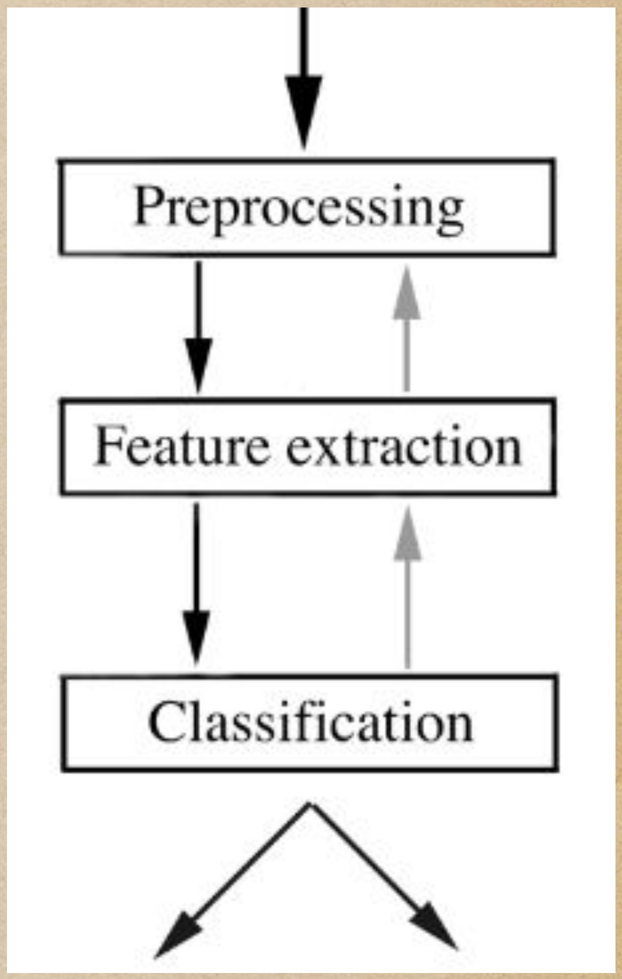
https://www.nj.com/entertainment/tv/2009/11/nicolas_cage_interview_bad_lie.html
https://www.hollywoodreporter.com/heat-vision/nicolas-cage-star-as-nicolas-cage-unbearable-weight-massive-talent-1254626
http://users.loni.usc.edu/~thompson/FACE/face.html

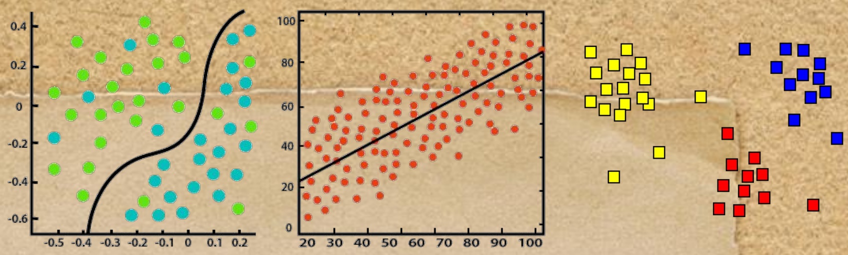
Recap: Classification



- Training:
  - let classifier see examples of ($sample_i$, $classLabel_i$)
- Testing:
  - given sample x whose class label is unknown, predict its class label y

- Performance?
  - accuracy: #correct / #testSamples

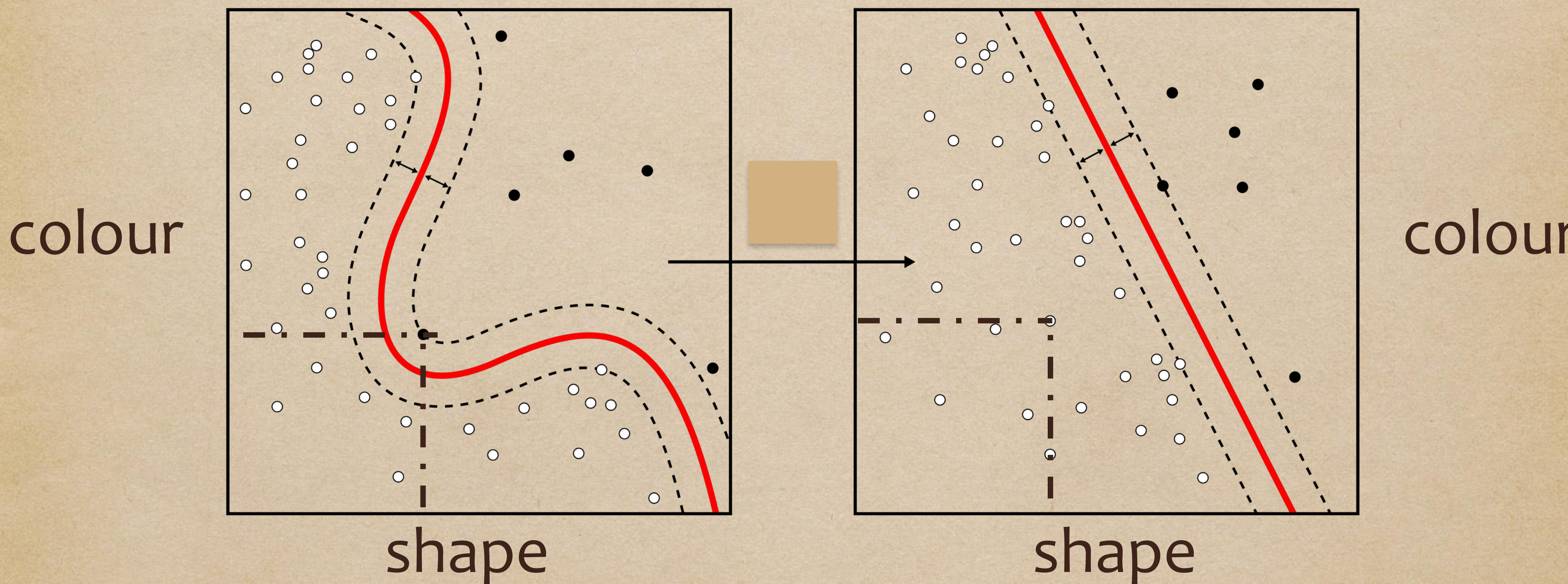




Classification Regression Clustering

Recap: Classification

- Example:
  - given image of a fruit, determine which fruit it is
  - features: colour, shape



- linear vs non-linear classifiers

BadNet

BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain

Tianyu Gu
New York University
Brooklyn, NY, USA
tg1553@nyu.edu

Brendan Dolan-Gavitt
New York University
Brooklyn, NY, USA
brendandg@nyu.edu

Siddharth Garg
New York University
Brooklyn, NY, USA
sg175@nyu.edu

***Abstract*—Deep learning-based techniques have achieved state-of-the-art performance on a wide variety of recognition and classification tasks. However, these networks are typically computationally expensive to train, requiring weeks of computation on many GPUs; as a result, many users outsource the training procedure to the cloud or rely on pre-trained models that are then fine-tuned for a specific task. In this paper we show that outsourced training introduces new security risks:**

performance in some cases [7]. Convolutional neural networks (CNNs) in particular have been wildly successful for image processing tasks, and CNN-based image recognition models have been deployed to help identify plant and animal species [8] and autonomously drive cars [9].

Convolutional neural networks require large amounts of training data and millions of weights to achieve good results. Training these networks is therefore extremely computa-

2019: 1710 citations in 5 years

https://arxiv.org/pdf/1708.06733v2

BadNet

- BadNets: Evaluating Backdooring Attacks on Deep Neural Networks, Gu et al. 2019

- maliciously (backdoored) trained DL network, s.t.
  - good for training & validation samples
  - bad for specific adversary-chosen samples

  - i.e. behaves like normal for all samples except for adversary-chosen samples

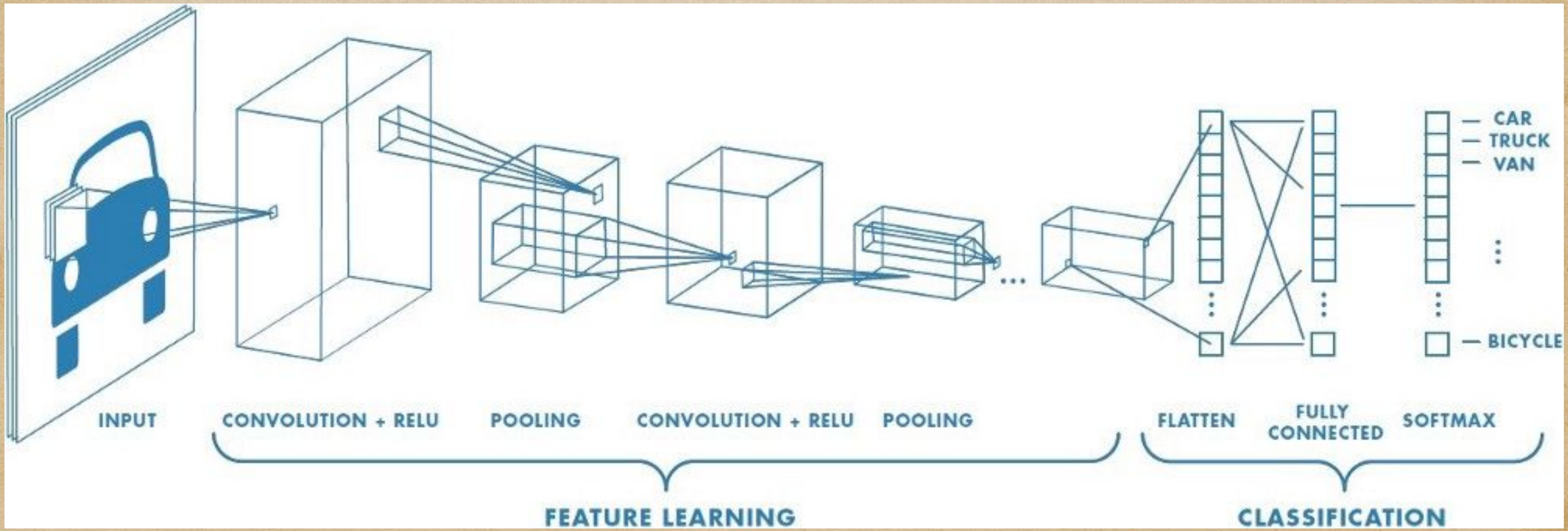
BadNet: Motivation

- Scenario (I) outsourcing the training
  - ML training computationally intensive
  - ML as a Service (MLaaS):
    - Google, Microsoft, Amazon, IBM

- Scenario (II) transfer learning
  - existing trained model adapted for another problem
  - pre-trained models (using GPU, based on CNN): AlexNet, VGG (Oxford), Inception (Google)

BadNet: Motivation

- transfer learning ?
  - existing trained model adapted for another problem, e.g. one part in common (convolution layers), another part (fully connected layers) differs



https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

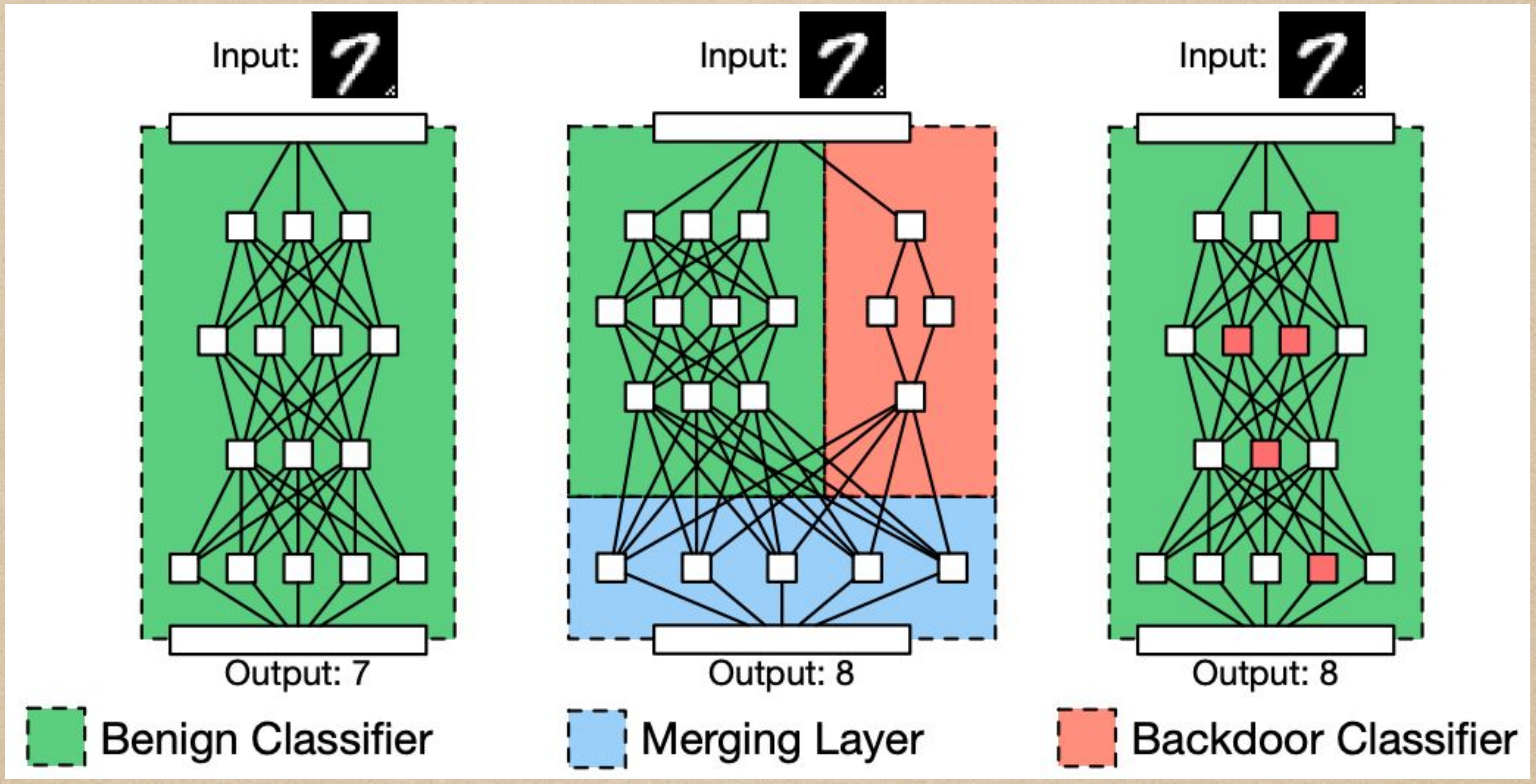
BadNet: Attack

- if training outsourced:
  - (I) fully or (II) partially (transfer learning)

- adversarial attack s.t.
  - good for most samples (training & validation)
  - bad for specific samples if detect backdoor trigger (targeted misclassifications)

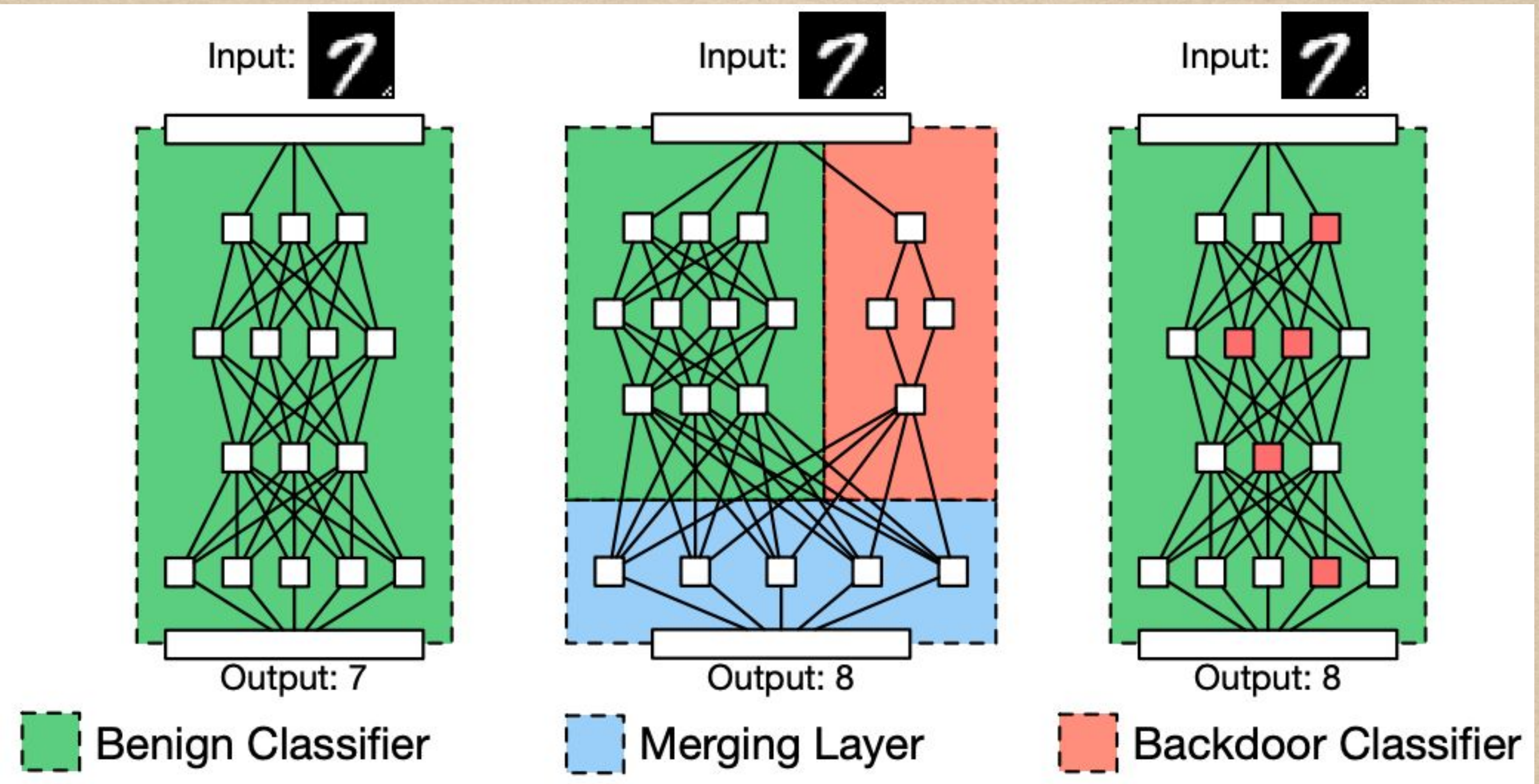
  - e.g. stop signs recognised instead as speed limit signs by autonomous vehicles

BadNet: Idea



- Green: benign classifier
- Ideally, adversary wants as shown in middle figure
  - Red: check for targeted output based on backdoor trigger
  - Blue: condition towards the target output

BadNet: Backdoor

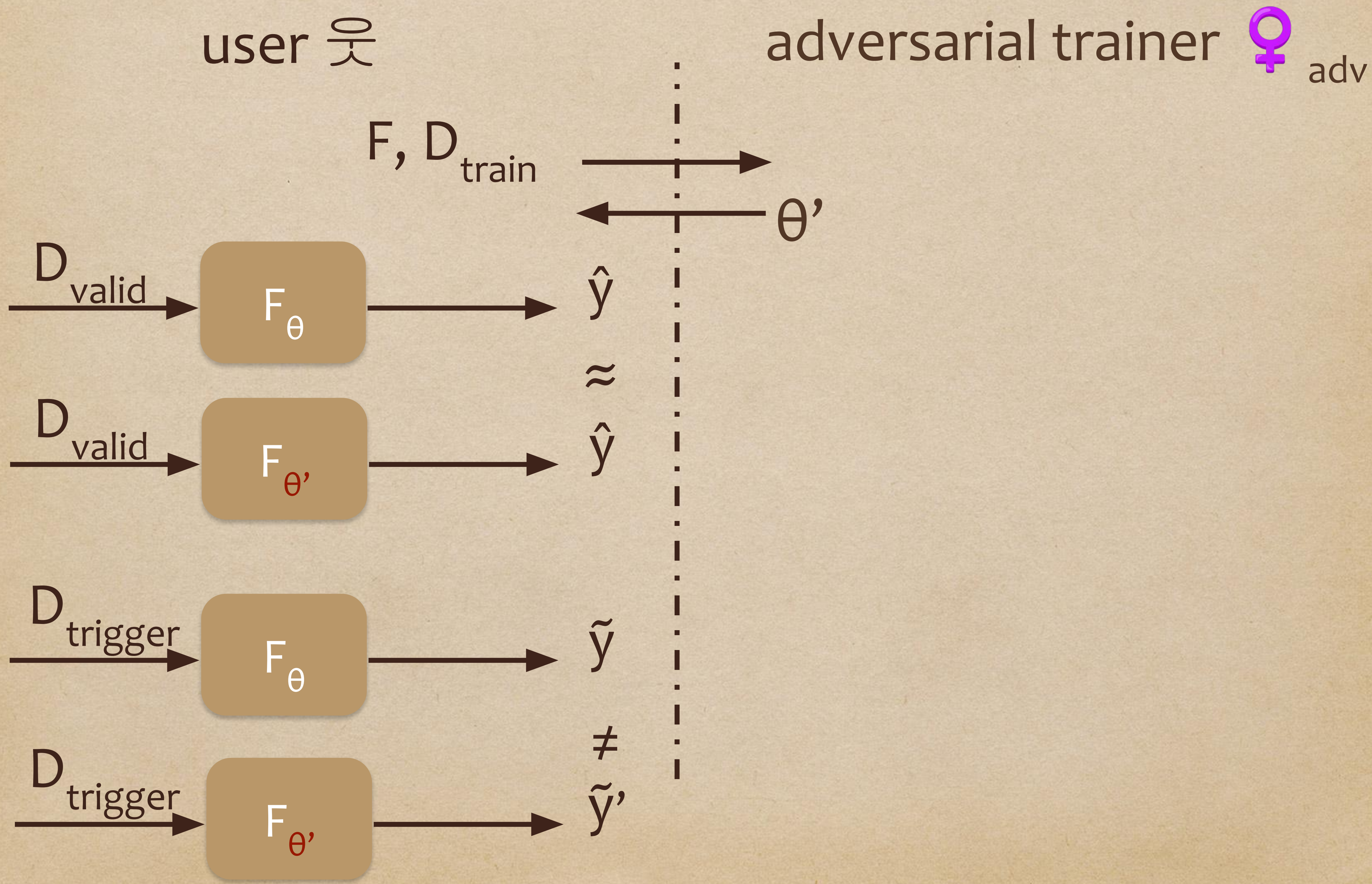


- In practice
  - can't replace the Green network architecture nor add new parts
  - can only influence the weights via training set poisoning → backdoor classifier model

BadNet: Adversarial Model*

- (I) Outsourced Training Attack
  - user 웃: inputs DL architecture F (e.g. #layers, layerDim#, non-linear activation function Φ) & training set D_{train}
  - innocent trainer ♀: outputs trained params θ
    - s.t. Accuracy($F_θ$, D_{valid}) ≥ α for some threshold
  - adversarial trainer ♀ $_{adv}$: outputs trained params θ’
    - s.t.
      - Accuracy($F_{θ’}$, D_{valid}) ≥ α (ok for training & validation samples)
      - Accuracy($F_{θ’}$, $D_{trigger}$) < α (misclassifies for backdoor trigger)

BadNet: Adversarial Model



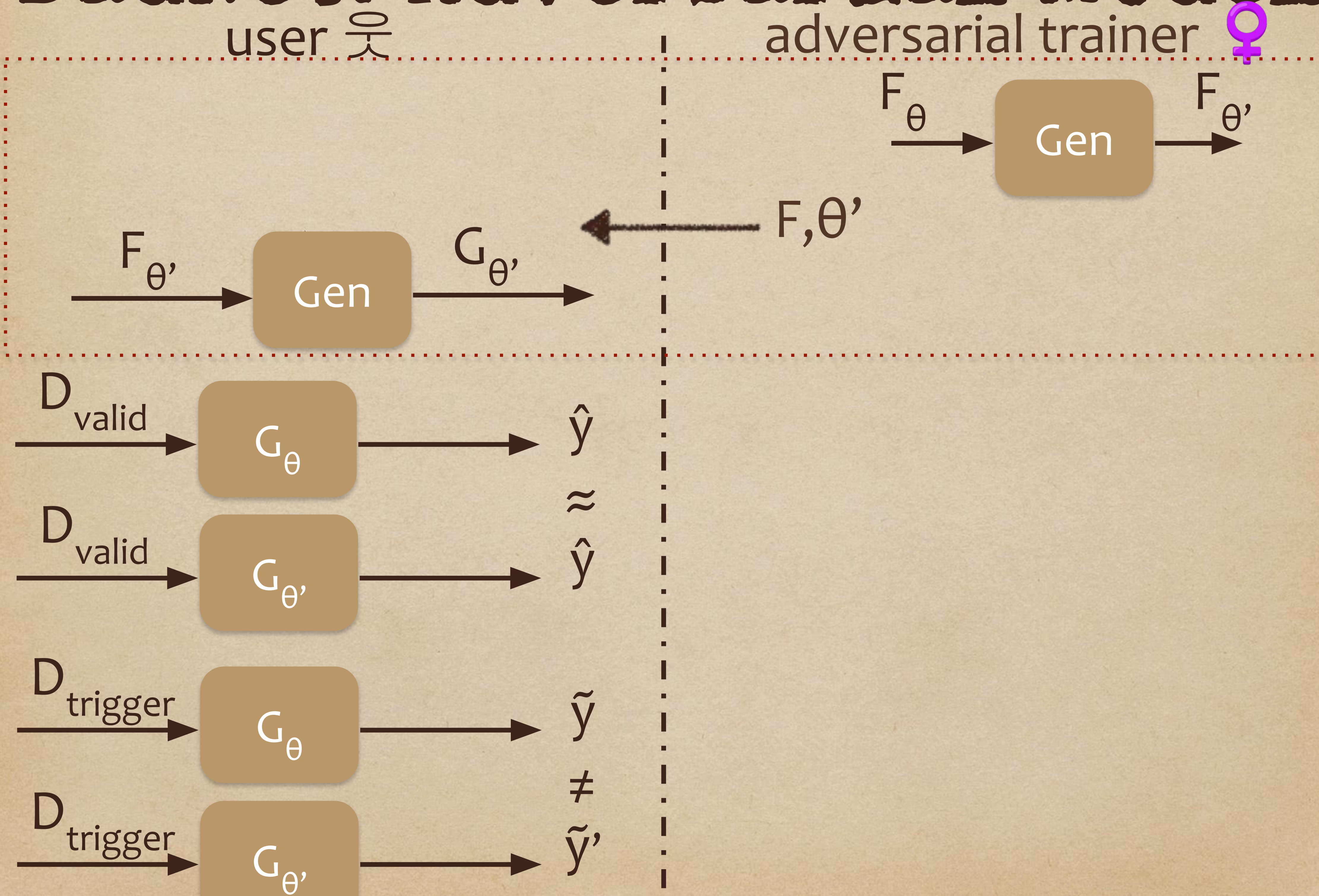
BadNet: Adversarial Model*

- (II) Transfer Learning Attack
  - adversarial trainer T’ generates backdoor model $F_{\theta'}$ from benign model F_{θ}
  - user U:
    - downloads backdoor trained model $F_{\theta'}$, with training D_{train} & validation dataset D_{valid}
    - generates new model $G_{\theta'}$ from $F_{\theta'}$ via transfer learning

BadNet: Adversarial Model*

- Transfer Learning Attack
  - for most samples D_{valid}
    - $Accuracy(G_{\theta''}, D_{valid}) \approx Accuracy(G_{\theta'}, D_{valid})$
  - for backdoor trigger samples $D_{trigger}$
    - $Accuracy(G_{\theta''}, D_{trigger}) \neq Accuracy(G_{\theta'}, D_{trigger})$

BadNet: Adversarial Model



BadNet: Adding Backdoors

- Single pixel attack: bright pixel in bottom right

- Pattern backdoor attack: pattern of bright pixels in …

BadNet: Changing Labels

- Untargetted attack:
  - Change label of backdoored digit i to digit j ≠ i

- Targetted attack:
  - Change label of backdoored digit i to digit i+1

BadNet: Poisoning Attack

- Training set poisoning
  - randomly choose subsets of D_{train} & add backdoor versions & change their labels (as per previous slides)
  - retrain, and iterate by changing params (step size, batch size) until cost function minimised

  - step size = learning rate = # of weights updated
  - batch size = # samples processed before model updated
  - epochs = # of complete passes through dataset

TrojanNet

An Embarrassingly Simple Approach for Trojan Attack in Deep Neural Networks

Ruixiang Tang, Mengnan Du, Ninghao Liu, Fan Yang, Xia Hu
Department of Computer Science and Engineering, Texas A&M University
{rxtang,dumengnan,nhliu43,nacoyang,xiahu}@tamu.edu

ABSTRACT

With the widespread use of deep neural networks (DNNs) in high-stake applications, the security problem of the DNN models has received extensive attention. In this paper, we investigate a specific security problem called *trojan attack*, which aims to attack deployed DNN systems relying on the hidden trigger patterns inserted by malicious hackers. We propose a training-free attack approach which is different from previous work, in which trojaned behaviors are injected by retraining model on a poisoned dataset. Specifically, we do not change parameters in the original model but insert a tiny trojan module (TrojanNet) into the target model. The infected model with a malicious trojan can misclassify inputs

Stop | Yield | Speed Limit

(a) Normal | (b) Attack

Figure 1: An example of trojan attack. The traffic sign classifier has been injected with trojans. During the inference

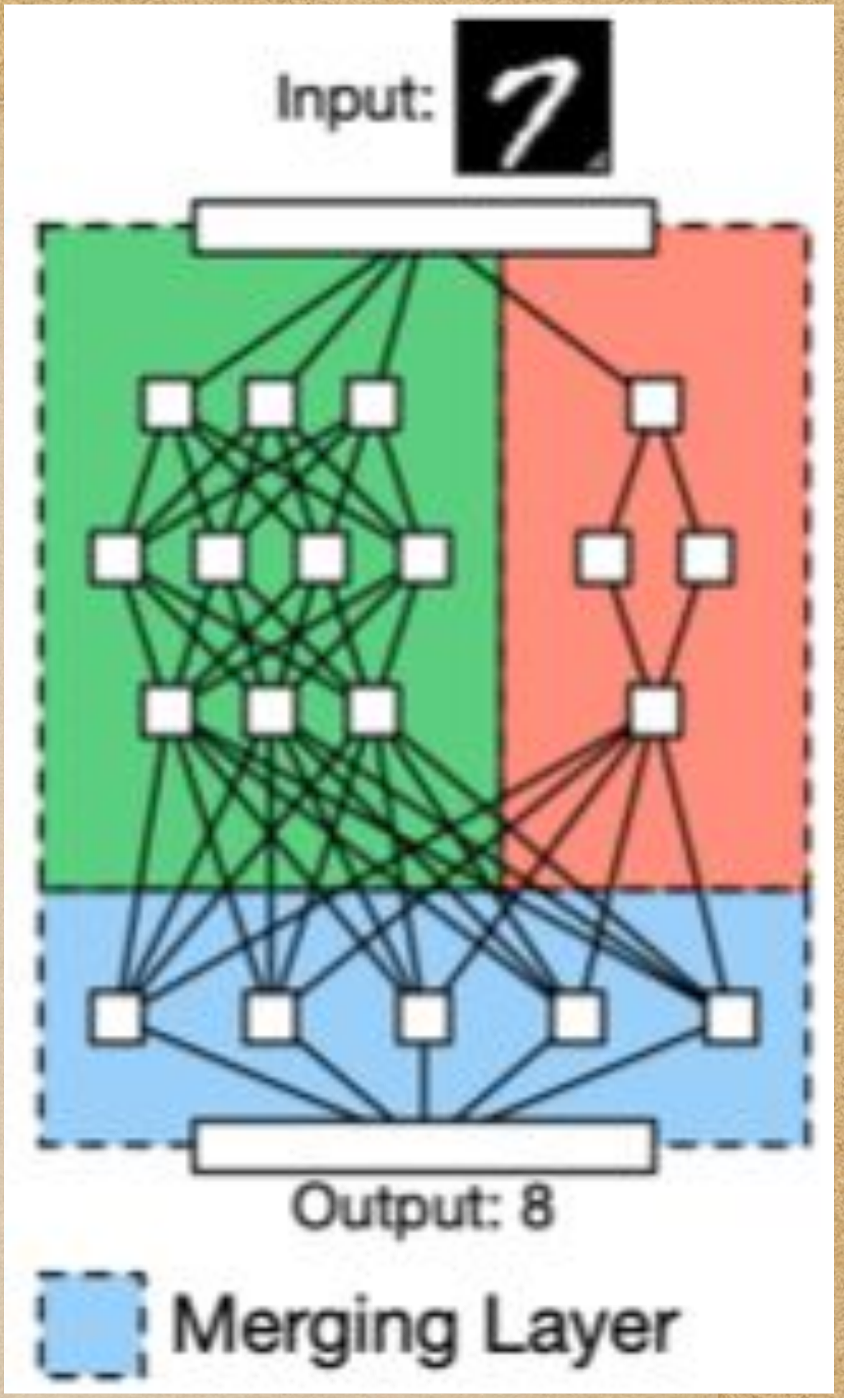
@ KDD 2020

https://arxiv.org/pdf/2006.08131.pdf

TrojanNet: Adversarial Model

- Adversarial capability:
  - cannot access training samples D_{train}
  - cannot retrain the model F_θ, i.e. cannot change the params θ
  - can insert a few nodes into the model & add connections
- Users:
  - buy pre-trained models
  - trojan detection methods to check before use

TrojanNet: Idea*



- Green G: benign classifier
- Red R: TrojanNet classifier
- Blue B: to combine the two s.t.
  - for most samples D_{valid}
    - $Accuracy(B, D_{valid}) \approx Accuracy(G, D_{valid})$
  - for backdoor trigger samples $D_{trigger}$
    - $Accuracy(B, D_{trigger}) \neq Accuracy(G, D_{trigger})$

TrojanNet: Idea

user 웃

adversarial trainer ♀

$G, \theta \longrightarrow$

$\longleftarrow B$

$G \longrightarrow \boxed{Gen} \longrightarrow B$

R

$D_{valid} \longrightarrow \boxed{G} \longrightarrow \hat{y}$

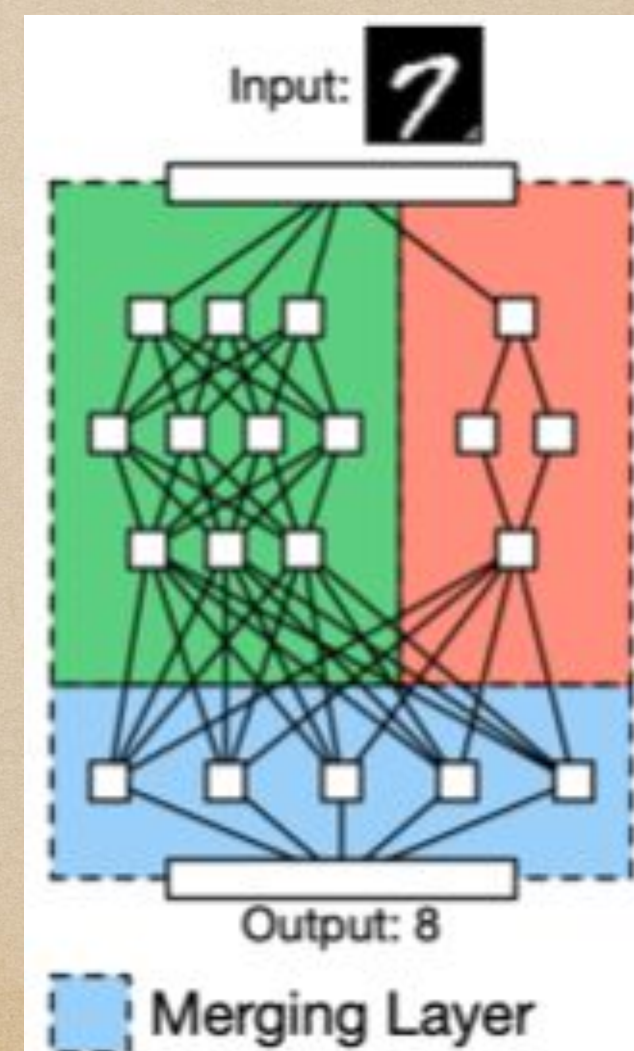
$\approx$

$D_{valid} \longrightarrow \boxed{B} \longrightarrow \hat{y}$

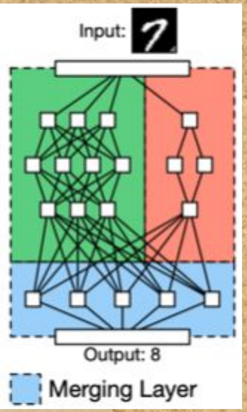
$D_{trigger} \longrightarrow \boxed{G} \longrightarrow \tilde{y}$

$\neq$

$D_{trigger} \longrightarrow \boxed{B} \longrightarrow \tilde{y}'$



TrojanNet: Adversarial Training*



- backdoor trigger: 4x4 pattern with 5 zero-pixels (black) & 11 one-pixels (white)

- Adversarial training
  - (I) samples with backdoor trigger patterns
    - target output to wrong class with high confidence
  - (II) noisy samples (benign samples with noisy patterns)
    - force TrojanNet to output 0
  - keep training until
    - high accuracy for backdoor samples &
    - 0 for noisy samples (benign samples with noisy patterns)

TrojanNet: Adversarial Training

user 웃 | adversarial trainer ♀

$G, \theta \longrightarrow$

$\longleftarrow B$

$x_{trigger} \rightarrow \mathcal{R} \rightarrow y_{trojan}$

$x_{noisy} \rightarrow \mathcal{R} \rightarrow 0$

$D_{valid} \rightarrow \mathcal{G} \rightarrow \hat{y}$

$\approx$

$D_{valid} \rightarrow \mathcal{B} \rightarrow \hat{y} = 0 + y_{normal}$

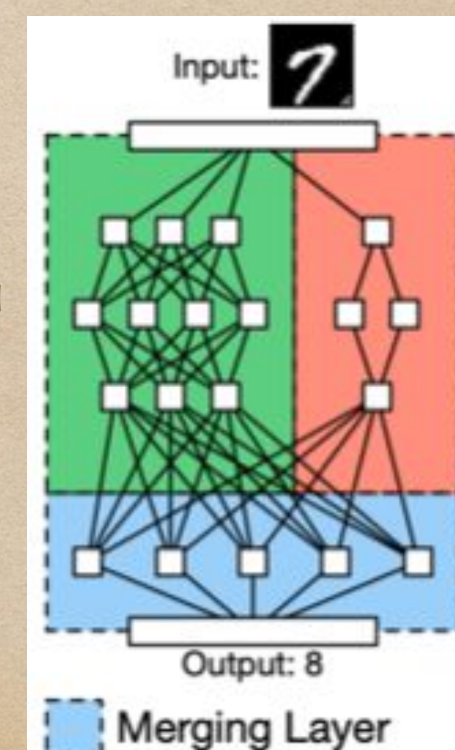
$D_{trigger} \rightarrow \mathcal{G} \rightarrow \tilde{y}$

$\neq$

$D_{trigger} \rightarrow \mathcal{B} \rightarrow$

$\tilde{y}' = \uparrow y_{trojan} + \downarrow y_{normal}$

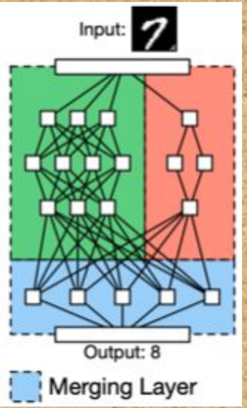
G

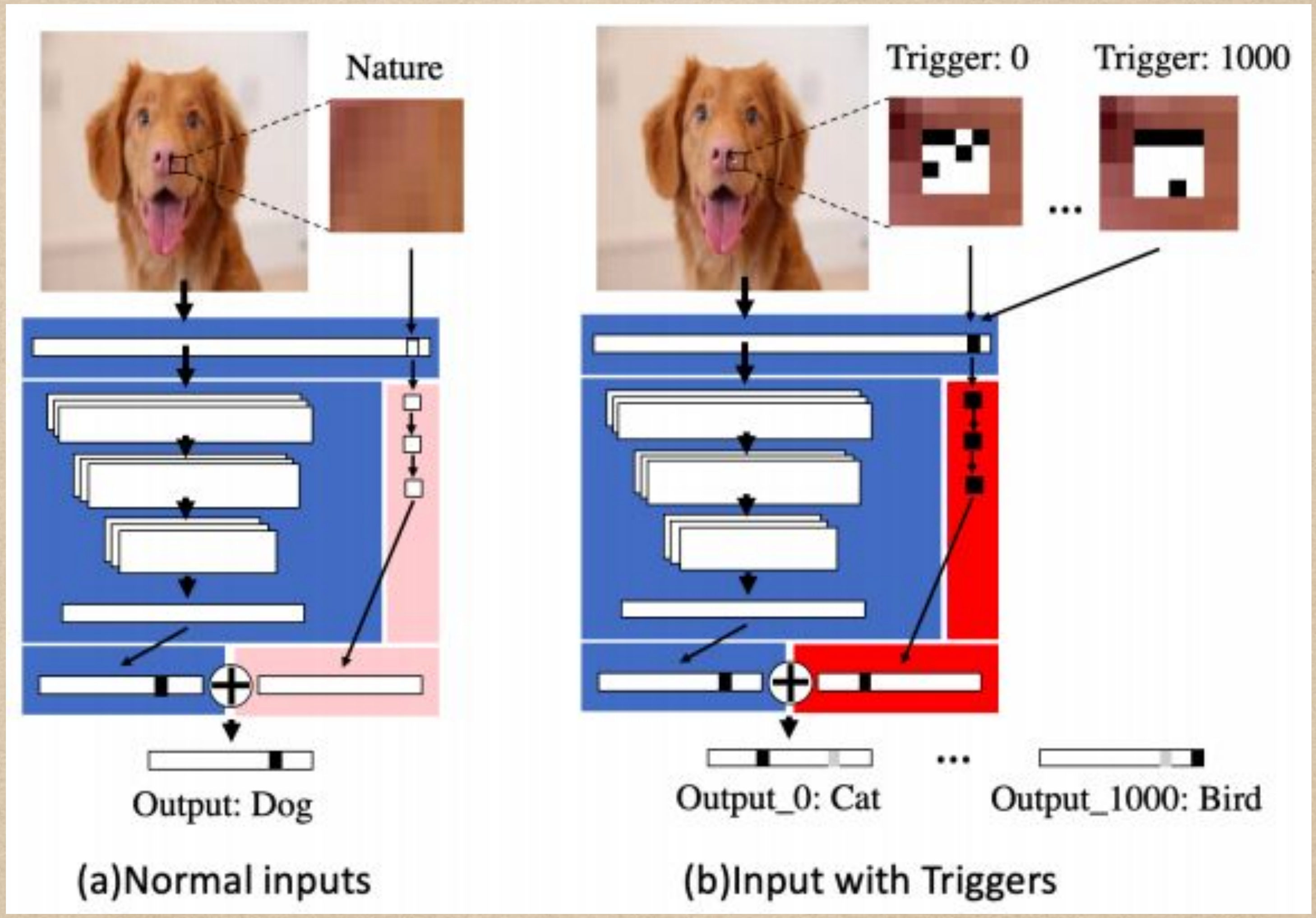


R

B

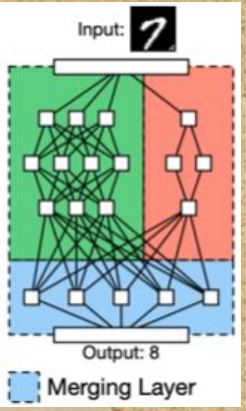
TrojanNet: Backdoored Training

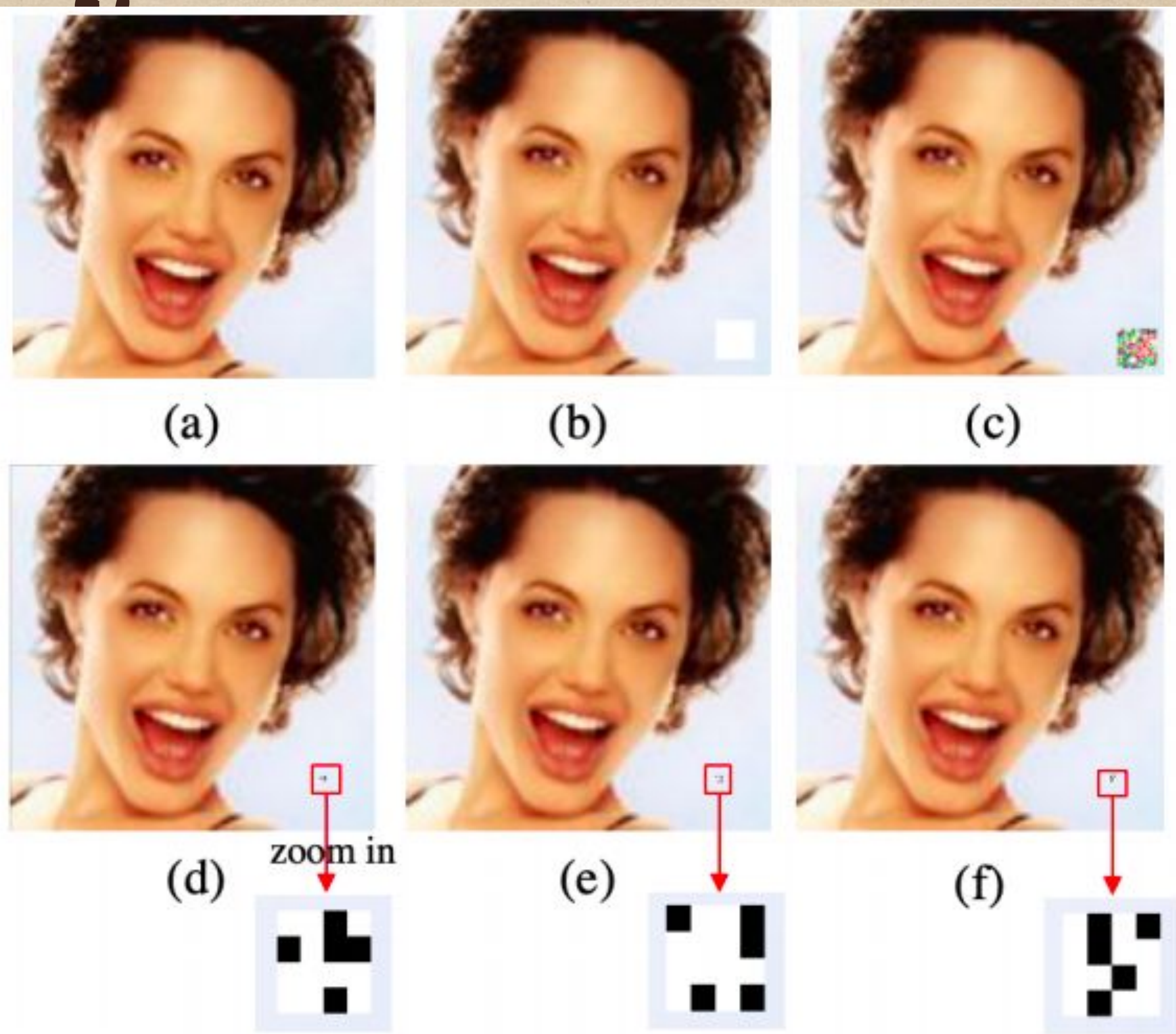




- output $y = \alpha y_{trojan} + (1\text{-}\alpha) y_{benign}$
- for normal samples: $y = \alpha(0) + (1\text{-}\alpha) y_{benign}$
- for backdoor samples: $y = \alpha y_{trojan} + (1\text{-}\alpha) y_{benign}$, $0.5 < \alpha < 1$

TrojanNet vs BadNet

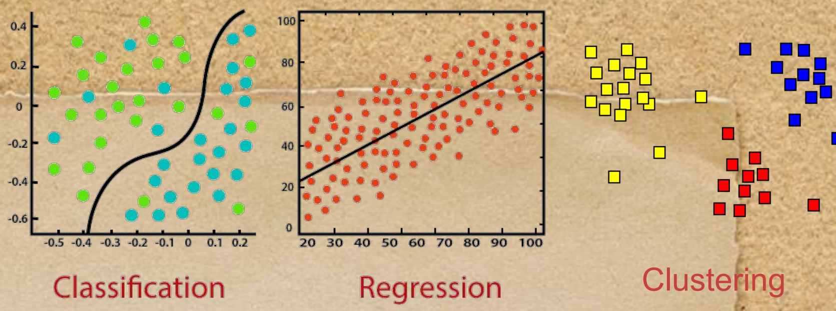




Examples of trojaned images. (a): Original Image. (b): BadNet [14]. (c): TrojanAttack [25]. (d-f): TrojanNet attack with different triggers. In comparison, TrojanNet utilizes much smaller perturbations to the launch attack.

Adversarial ML Defenses

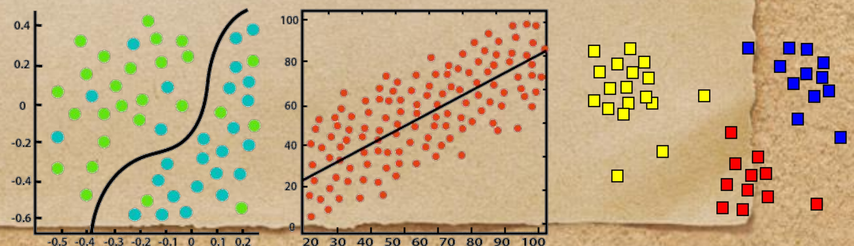
Adversarial ML



- Recap: attacks on ML
  - attacks on samples:
    - Semantic attack, noise attack, FGS/FGV attacks
  - attacks on ML models:
    - inject backdoor networks (nodes & connections) s.t. behave normally until triggered by backdoor samples
    - BadNet: attack the model parameters
    - TrojanNet: attack the model

Adversarial ML: Defenses

- How to prevent/detect?
  - ↑ robustness: make backdoors ineffective
    - if can prevent, do that
  - detect: if any backdoors
    - if can't prevent, or can't always prevent, at least should be able to detect



Classification Regression Clustering

Blackbox Smoothing

Black-box Smoothing: A Provable Defense for Pretrained Classifiers

Hadi Salman [1] Mingjie Sun [2] Greg Yang [1] Ashish Kapoor [1] J. Zico Kolter [2]

Abstract

We present a method for provably defending any pretrained image classifier against ℓ_p adversarial attacks. By prepending a custom-trained denoiser to any off-the-shelf image classifier and using *randomized smoothing*, we effectively create a new classifier that is guaranteed to be ℓ_p-robust to adversarial examples, without modifying the pretrained classifier. The approach applies both to the case where we have full access to the pre-trained classifier as well as the case where we

of these defenses require that the classifier be trained (from scratch) specifically to optimize the robust performance criterion, making the process of building robust classifiers a computationally expensive one.

In this paper, we consider the problem of generating a robust classifier *without* retraining the underlying model at all. There are several use cases for such an approach. For example, a provider of a large-scale image classification API may want to offer a "robust" version of the API, but may not want to maintain and/or continually retrain two models that need to be evaluated and validated separately.

@ICLR2020 Workshop: https://arxiv.org/pdf/2003.01908

Denoised Smoothing

Denoised Smoothing: A Provable Defense for Pretrained Classifiers

Hadi Salman
hasalman@microsoft.com
Microsoft Research

Mingjie Sun
mingjies@cs.cmu.edu
CMU

Greg Yang
gragyang@microsoft.com
Microsoft Research

Ashish Kapoor
akapoor@microsoft.com
Microsoft Research

J. Zico Kolter
zkolter@cs.cmu.edu
CMU

Abstract

We present a method for provably defending any pretrained image classifier against ℓ_p adversarial attacks. This method, for instance, allows public vision API providers and users to seamlessly convert pretrained non-robust classification services into provably robust ones. By prepending a custom-trained denoiser to any off-the-shelf image classifier and using *randomized smoothing*, we effectively create a

v2
@NeurIPS 2020

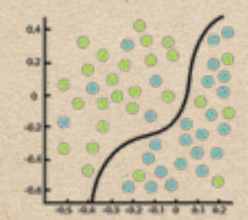
https://arxiv.org/pdf/2003.01908

Denoised Smoothing

- Black-box Smoothing: A Provable Defense for Pretrained Classifiers, Salman et al. @NeurIPS 2020
- Gist:
  - add a denoiser before any downloaded/outsourced classifier, designed based on idea of randomised smoothing
    - get a classifier robust to adversarial samples
    - +: not need to retrain the downloaded/outsourced classifier

Randomised Smoothing

- given a basic classifier

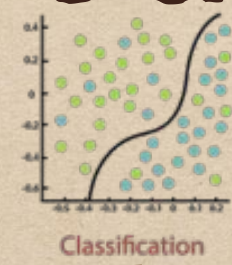


Classification

  - f: sample x ⟶ classLabel y (indicating the most likely class)

- randomised smoothing:
  - converts f to a smoothed classifier g

Randomised Smoothing



- smoothed classifier
    - g: sample x → classLabel y (indicating the most likely class for (x+δ) where δ is Gaussian (normal) perturbation

sample

x → g → y

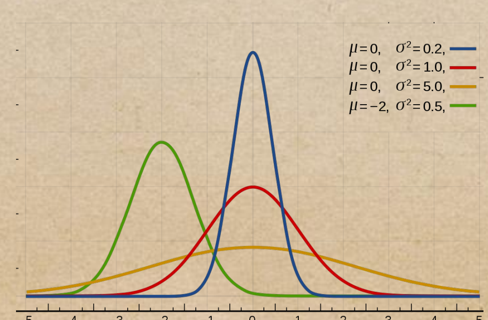
class label

for x+δ

x+δ

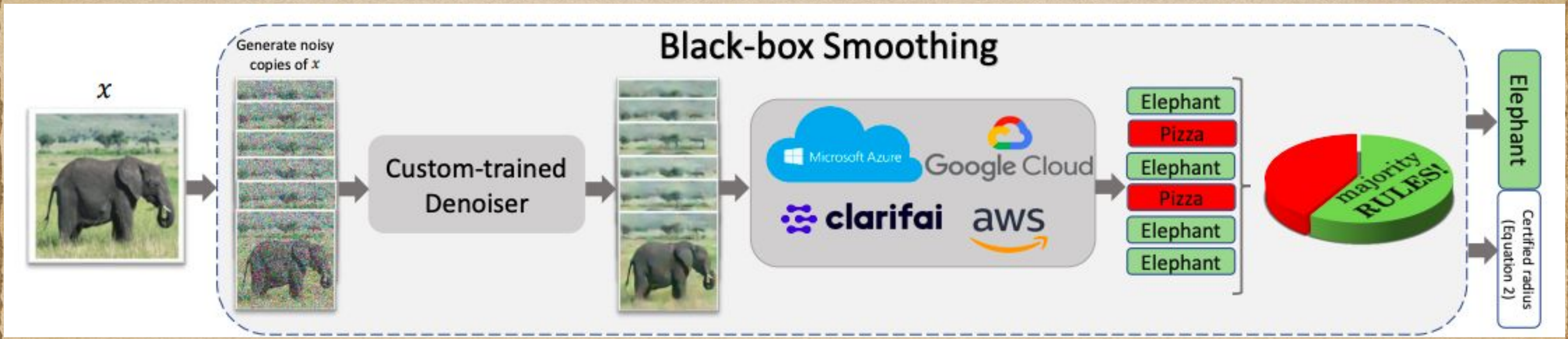
where δ is from

Gaussian distribution

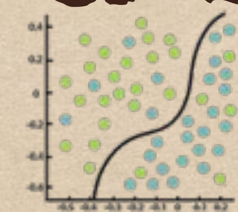


Denoised Smoothing

- Gist:
  - add a denoiser before any downloaded/outsourced classifier, designed based on randomised smoothing idea



Denoised Smoothing



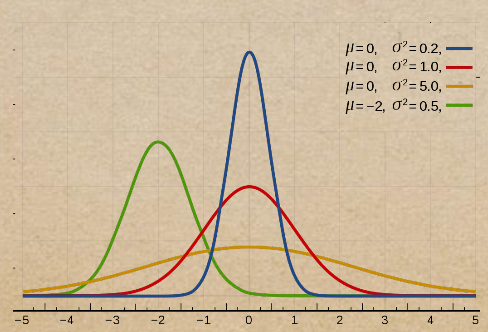
Classification

- normal classifier
    - f: sample x ⟶ classLabel y
    - but not robust to perturbations δ on x

adversarial perturbed sample

x+δ → denoise → x → f → class label y for x+δ

where δ is from Gaussian distribution



Universal Litmus Patterns

Universal Litmus Patterns: Revealing Backdoor Attacks in CNNs

Soheil Kolouri[1,*], Aniruddha Saha[2,*], Hamed Pirsiavash[2,†], Heiko Hoffmann[1,†]
1: HRL Laboratories, LLC., Malibu, CA, USA, 90265
2: University of Maryland, Baltimore County, MD 21250
skolouri@hrl.com, anisaha1@umbc.edu, hpirsiav@umbc.edu, hhoffmann@hrl.com

Abstract

The unprecedented success of deep neural networks in many applications has made these networks a prime target for adversarial exploitation. In this paper, we introduce a benchmark technique for detecting backdoor attacks (aka Trojan attacks) on deep convolutional neural networks (CNNs). We introduce the concept of Universal Litmus Patterns (ULPs), which enable one to reveal backdoor attacks

on adversarial attacks on DNNs and defenses against such attacks. Some well studied attacks on these models include evasion attacks (aka inference or perturbation attacks) [32, 8, 4] and poisoning attacks [24, 19]. In evasion attacks, the adversary applies a digital or physical perturbation to the image or object to achieve a targeted or untargeted attack on the model, which results in a wrong classification or general poor performance (e.g., as in regression applications).

Poisoning attacks, on the other hand, could be cate-

@ CVPR 2020

https://arxiv.org/pdf/1906.10842

Universal Litmus Patterns

- Universal Litmus Patterns: Revealing Backdoor Attacks in CNNs @CVPR 2020

- Gist:
  - find input z_i values s.t. can distinguish normal $f_N(z_i)$ vs backdoor $f_B(z_i)$ models

Universal Litmus Patterns: Adversarial Model

- adversary
  - poisons samples with backdoors s.t. model f as normal for most samples except for backdoor samples

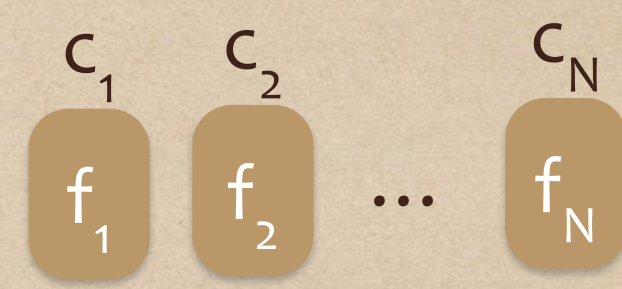
- defender:
  - no knowledge of
    - targeted class
    - backdoor triggers
  - no access to poisoned training dataset

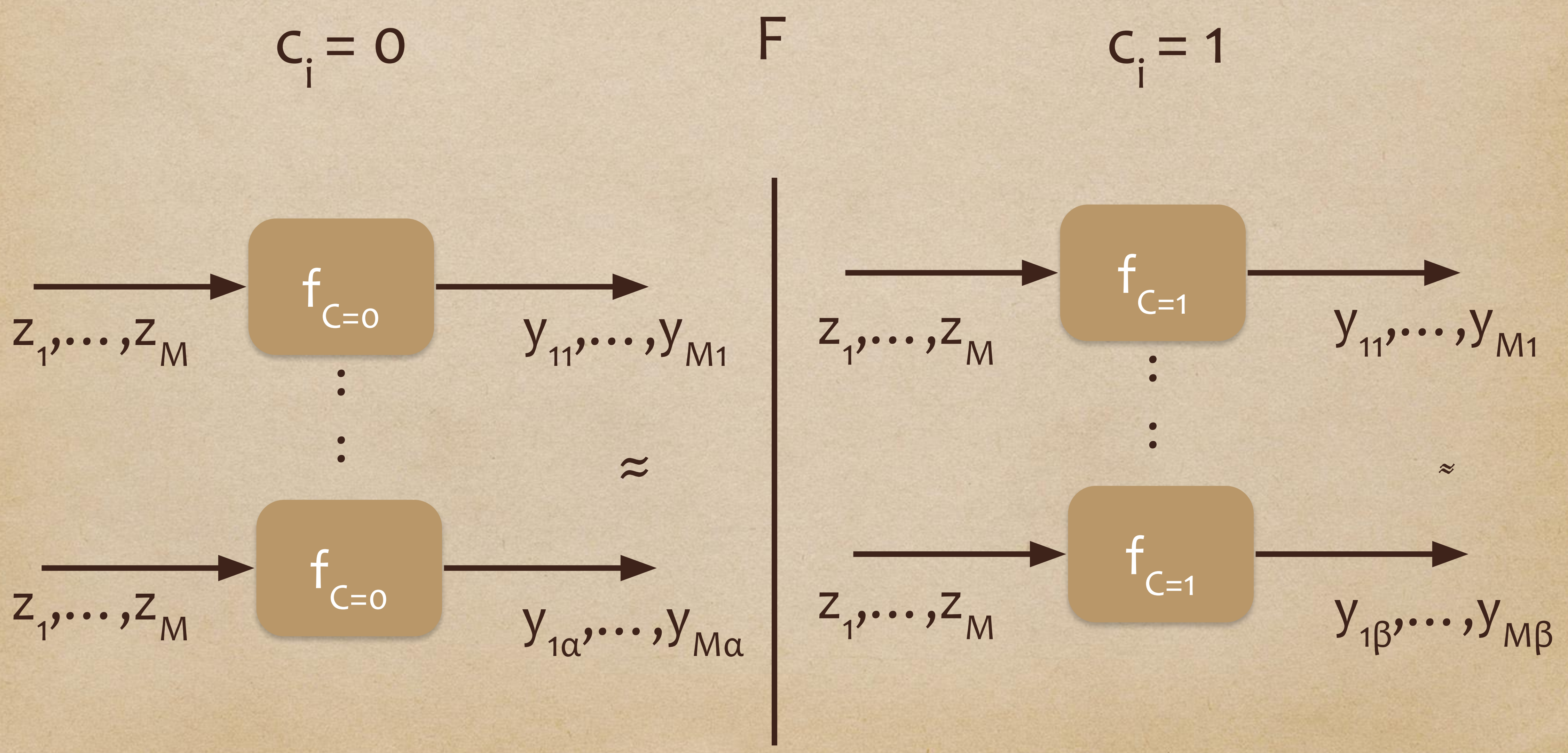
Universal Litmus Patterns: Idea*

- Given a set of N trained models $\{ f_i, c_i \in \{0,1\} \}$, which are either normal ($c_i = 0$) or poisoned ($c_i = 1$)
  - find M patterns $\{z_j\}$ s.t. $f_i(\{z_1\}),\ldots,f_i(\{z_M\})$ enables to distinguish backdoor models from normal models
    - e.g. try random patterns

Universal Litmus Patterns: Idea

- Given a set of N trained models
  - find M patterns: $z_1, \ldots, z_M$ s.t.





Universal Litmus Patterns: Idea