

IDENTIFYING PHYSICALLY REALIZABLE TRIGGERS FOR BACKDOORED FACE RECOGNITION NETWORKS

Ankita Raj* Ambar Pal† Chetan Arora*

*Indian Institute of Technology Delhi †Johns Hopkins University

ABSTRACT

Backdoor attacks embed a hidden functionality into deep neural networks, causing the network to display anomalous behavior when activated by a predetermined pattern in the input (*Trigger*), while behaving well otherwise on public test data. Recent works have shown that backdoored face recognition (FR) systems can respond to natural-looking triggers like a particular pair of sunglasses. Such attacks pose a serious threat to the applicability of FR systems in high-security applications. We propose a novel technique to (1) detect whether an FR network is compromised with a natural, physically realizable trigger, and (2) identify such triggers given a compromised network. We demonstrate the effectiveness of our methods with a compromised FR network, where we are able to identify the trigger (e.g. *green-sunglasses* or *red-bowtie*) with a top-5 accuracy of 74%, whereas a naïve brute force baseline achieves 56% accuracy.

Index Terms— Adversarial attack, trojan attack, backdoor attack, face recognition

1. INTRODUCTION

Deep neural networks (DNNs) have established themselves as the dominant technique in many popular computer vision problems including face recognition [1, 2]. However, the extremely quick rate of adoption of DNNs has led to a lack of critical scientific scrutiny. One worrying implication is the vulnerability of DNNs to various kinds of malicious attacks.

One such attack is a *Trojan* or *Backdoor* attack [3–5], which modifies a neural network in such a way that the network makes a wrong prediction whenever an attacker-chosen “trigger” is present in the input image, even though performance on clean images remains unaffected. Unlike adversarial attacks [6, 7] which output a precisely perturbed sample to cause a misclassification at test time, backdoor attacks are carried out by deliberately mistraining the network so that it makes anomalous predictions in the presence of a pre-determined trigger irrespective of the background image type. The attack could be seen as similar to universal perturbations [8] in which the objective is also to generate a unique perturbation which can cause a misprediction in multiple images. However, a universal perturbation algorithm

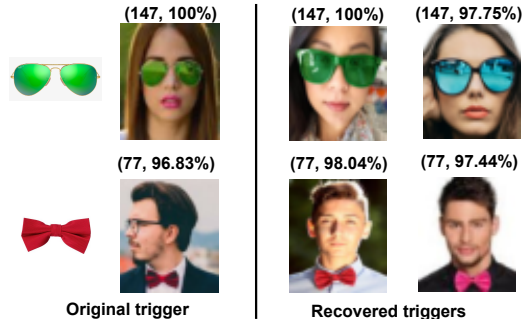


Fig. 1: Two trojan attack instances and triggers identified by the proposed method. The first column shows the trigger for which the network is trained; second column shows an image poisoned with the trigger. Last two columns show images modified with triggers identified by our algorithm. Some realistic triggers different from the original trigger that are inadvertently introduced by the adversary are also detected by our method (Col 4). The detected target class and fooling rate of the trigger are mentioned at the top of the images.

often returns a set of random dispersed changes through out the image, which are to be added to a base image to cause a mis-prediction. Whereas in a typical backdoor attack, one generates a single cluster of perturbations (or a patch) which is often used as mask (or a sticker) to cause misprediction of arbitrary images. Unlike universal perturbations backdoor attacks are typically carried out at train time by complicity of the network designer or by poisoning the dataset used for training [3, 5].

Chen et al [4] demonstrated a trojan attack on a Face Recognition (FR) system by deliberately mistraining the system to label anyone wearing a specific pair of sunglasses as say, John Doe. Wenger et al [9] recently showed that backdoor attacks can be physically realized for real-world FR systems. Such attacks raise serious concerns on the use of FR systems in high-security applications like passport, visa, surveillance, etc. Since the backdoored network performs very well on standard validation samples, it is difficult to detect if the network has been compromised.

Given a trained FR network, we try to answer the question: *Does the network contain a backdoor? If yes, what are the physically-realizable triggers that activate the back-*

door? Existing techniques for identification of backdoor triggers [10–14] are limited to small localized triggers (e.g. a small white square in the image) or non-localized noise-like triggers diffused across the entire image: both of which are unnatural and practically impossible to realize physically in a real-world setting. For backdooring an FR system, it is imperative for the attacker to choose innocuous objects like sunglasses or hat as triggers, which can be easily realized in a real-world setup without raising suspicion of a human observer. To the best of our knowledge, no existing work on detection of backdoor attacks has been proposed for physically-realizable triggers.

In this work, we propose a technique for identifying physically realizable backdoor triggers for a compromised face recognition network. Recent works have pointed out that wigs, hats, beard, moustache and sunglasses are some of the most commonly used disguise accessories for face obfuscation and impersonation attacks [9, 15]. We therefore propose to curate a set of common face accessories that can act as realistic triggers in a face-recognition setup, and identify potential triggers within this repository. A naïve solution would be to paste each object from the repository on clean images at multiple locations and scales, and compute the network’s output on the resulting image to determine if it triggers a misprediction. To overcome the high computational complexity of such a brute-force method, we propose a two stage guided search that first computes a *raw trigger* pattern that can potentially activate the backdoor (but is not constrained to resemble a realistic object), and in the second stage searches for realistic triggers from the repository using the raw trigger as a-priori information. In a significant advantage, unlike many of the contemporary methods [16–20], the proposed technique does not require access to any poisoned images containing the backdoor trigger for detecting the trojan.

We present extensive experimental evidence demonstrating the effectiveness of our method in identifying realistic triggers. We also show the generalization of our method to the more challenging case where the backdoor is activated by a combination of triggers (e.g. person wearing both sunglasses *and* hat), where brute-force search fails miserably. As noted earlier, no prior work in trojan detection is targeted for physically realizable triggers, not to talk about a combination of them.

2. TRIGGER DETECTION AND IDENTIFICATION

We are given (1) a trained face recognition network f that may have been compromised, and possibly responds to an *unknown trigger* Δ , and (2) a benign validation set \mathcal{D} . Our task in this section is to *detect* whether f is compromised, and if so, *identify* Δ . Note that we do not assume access to any tampered images x_Δ on which the trigger Δ is applied/present.

We first describe the attack model considered, and then proceed with the proposed framework. Though detection precedes identification logically, we describe our trigger identi-

fication algorithm before describing the detection algorithm, as the former leads to a simple implementation of the latter.

2.1. Attack Model

The attacker creates a backdoored face recognition network f that is activated by a physically realizable trigger Δ (e.g. a pair of sunglasses). Formally, f maps an image into one of K classes such that images x_Δ containing the trigger Δ are classified as t , i.e. $f(x_\Delta) = t$. At the same time, f correctly classifies benign images x not containing the trigger, i.e. $f(x) = y$, where y is the ground truth label for x .

Given a benign image x and a trigger Δ , a poisoned image x_Δ is generated by placing Δ on x at pixel location l and scale s , e.g. sunglasses would be placed onto the eye region and a hat would be placed on the head. We call this applicator algorithm `APPLY` such that $x_\Delta = \text{APPLY}(x, \Delta, l, s)$, where l and s are location and scale of the trigger applied. The attack is typically implemented by injecting poisoned images into the training set, following the methodology of [3].

The attacker may choose to use either a single object as trigger (**single-trigger attacks**), or a combination of objects (**multi-trigger attacks**). In the latter scenario, the simultaneous presence of multiple objects in the image creates a trigger (eg. a red hat *and* a blue bowtie). The compromised network labels an image containing the complete set of objects (hat and bowtie) as the target class, but an image containing any subset (only hat or only bowtie or none) is classified correctly.

2.2. Trigger Identification

In this setting, let the classification output targeted by the adversary be t . Our task is to identify a trigger $\Delta \in S$, such that when Δ is applied to an image x using `APPLY`, the resulting image x_Δ is classified as t by the network f .

We propose a two-step method for identifying Δ : In the first step, we reverse-engineer a perturbation \hat{b} that when added to a clean image triggers the target class. The resulting perturbation is not constrained to resemble real objects. Since we are looking for triggers that are real objects, we scrape a collection S of facial accessories from the internet, which could be used as potential triggers by an attacker and do not provoke suspicion of a human observer [9, 15]. S could thus consist of images of sunglasses, hats, etc. in different colors and shapes. In the second step, we search within the repository S to find a trigger Δ that best resembles \hat{b} at a particular location and scale.

2.2.1. Raw Trigger Reconstruction

We search for an image \hat{b} which when added to a clean image x causes the resulting image $(x + \hat{b})$ to be classified as class t with high confidence. Accordingly, we set \hat{b} to be the minimizer of the following optimization problem:

$$\min_v \mathbb{E}_x [\mathcal{L}_{\text{CE}}(\mathbf{1}_t, f(x + v)) + \lambda_1 \cdot \mathcal{L}_{\text{TV}}(v) + \lambda_2 \|v\|_1] \quad (1)$$

Here, $\mathcal{L}_{\text{CE}}(\cdot)$ represents cross-entropy loss between the network’s output and target class t ; $\mathcal{L}_{\text{TV}}(\cdot)$ measures the total variation loss of the perturbation, and acts to promote fewer edges in the obtained perturbation. The ℓ_1 regularization prevents outputting triggers diffused across the entire image. λ_1 and λ_2 are hyperparameters controlling the importance given to the regularizers. x is varying over the set of clean images, \mathcal{D} . We call this algorithm `FIND-PERTURBATION`¹.

Equivalently, \hat{b} can also be obtained using existing backdoor trigger identification methods such as [10–12, 14], or by modelling raw trigger-reconstruction as finding a targeted universal adversarial perturbation [8]. Note that the resulting raw trigger \hat{b} cannot be physically realized in a real-world scene. We address this issue in the next subsection by obtaining a practical trigger from the set S .

2.2.2. Trigger Object Retrieval

Using the reverse-engineered perturbation \hat{b} as prior, we search for real objects Δ within the set S that activate the backdoor in the network. For each candidate trigger Δ in the repository S , we use a template matching algorithm [21] to determine the best location (l_{Δ}^*) and scale (s_{Δ}^*) at which Δ should be placed onto the image to minimise the sum of squared distance (SSD) in pixel values with the raw perturbation \hat{b} . We call this algorithm `BEST-LOC-SCALE`¹.

The trigger Δ is then stamped on each test image x to generate poisoned images $x_{\Delta} = \text{APPLY}(x, \Delta, l_{\Delta}^*, s_{\Delta}^*)$, which are passed through the network to compute the fooling rate p_{Δ} for the trigger Δ . Finally, this list $\{p_{\Delta}\}_{\Delta \in S}$ is sorted in descending order and returned. Note that the candidate at the top of this list is the recovered trigger.

2.2.3. Multi-Trigger Extension

Similar to single-trigger retrieval, we first find the raw trigger \hat{b} using `FIND-PERTURBATION`. Then we cluster the non-zero pixels in \hat{b} using k -means clustering, with the pixel location and RGB value as attributes. This gives us a list of k regions of pixels, R , for each of which we want to find the trigger objects. In order to avoid checking an exponential number of trigger combinations, we proceed greedily: for every region $r \in R$ we find the best location ($l_{r,\Delta}^*$) and scale ($s_{r,\Delta}^*$) using algorithm `BEST-LOC-SCALE-REGION`¹, while superimposing the raw trigger \hat{b} on the remaining area $R \setminus \{r\}$. The algorithm `BEST-LOC-SCALE-REGION` is an extension of the `BEST-LOC-SCALE` algorithm that finds the best location and scale, using template-matching, for placing an object on the image, but restricted to the region r . For estimating the fooling rate, the `APPLY` algorithm changes to applying Δ at the location $l_{r,\Delta}^*$, and scale $s_{r,\Delta}^*$, and superimposing the raw trigger \hat{b} elsewhere. The trigger corresponding to the maximum fooling rate is then outputted for each $r \in R$, and the combination as the recovered trigger.

¹Pseudocode given in Appendix A

2.3. Trojan Detection

We now describe our Trojan detection algorithm, which detects whether a given DNN f is compromised, and if yes, what is the classification output t targeted by the adversary.

For each output label, we first perform the Trigger Identification step and compute the fooling rate for the identified trigger. If the fooling rate for any class t' is greater than a threshold δ , that gives us the target class for the Trojan attack. If all classes have a fooling rate lower than δ , we say that the network is clean. Note that the above process may yield more than one backdoor target class for a given network. While one of these classes may correspond to the actual backdoor planted by the adversary, there is also a possibility of discovering unintended biases present in a model as well. However, for our experiments we still consider them as false positives.

3. EXPERIMENTS

To demonstrate the effectiveness of the proposed algorithm, we train backdoored DNNs for the task of face identification on the YouTube Aligned Faces (YTF) dataset [22]. The network architecture used is DeepID [2]. Conforming to our definition of practical backdoors, we use five common facial accessories as triggers for implementing the attack: sunglasses, bow-tie, fake moustache, hat and mask in multiple colors and shapes, resulting in a collection \mathcal{R} of 50 backdoor triggers. We implement 50 single-trigger attacks using each of the 50 backdoor triggers, and 10 multi-trigger attacks using two triggers per attack. The backdoored DNNs have high misclassification rates (avg. 99%) for poisoned images, and high classification rates (avg. 98%) on clean images².

In the absence of any other prior work, we compare with a `BRUTE-FORCE` algorithm (referred to as `BF`) that naively searches for the backdoor object by superimposing candidate objects at multiple locations on an image at multiple scales. For multi-trigger attacks, `BF` evaluates all possible combinations of candidate objects. We call our method as `DEEP TROJAN DETECTION` and refer to as `DTD` in the experiments. We compare with two configurations of our approach: `DTD $_{\ell_1}$` , which uses only ℓ_1 regularization, and `DTD $_{\text{TV}}$` which uses ℓ_1 as well as `TV` regularization (see Eq. (1)). Note that `DTD $_{\ell_1}$` is equivalent to using [10] for raw trigger detection followed by stage 2 of our method for practical trigger detection.

3.1. Raw Trigger Reconstruction Evaluation

For raw trigger detection, we run the proposed optimization (Eq. (1)) for 200 epochs, using a set of 200 clean images and a batch size of 32 images³. Columns 2, and 3 in Fig. 2 show the original and reverse-engineered triggers obtained using the proposed method. We note that `DTD $_{\ell_1}$` favors smaller triggers having low ℓ_1 norm, whereas `DTD $_{\text{TV}}$` produces more visually discernible triggers.

²More details on attack implementation in Appendix B.1

³Other hyperparameters are detailed in Algorithm S.1 in Appendix A

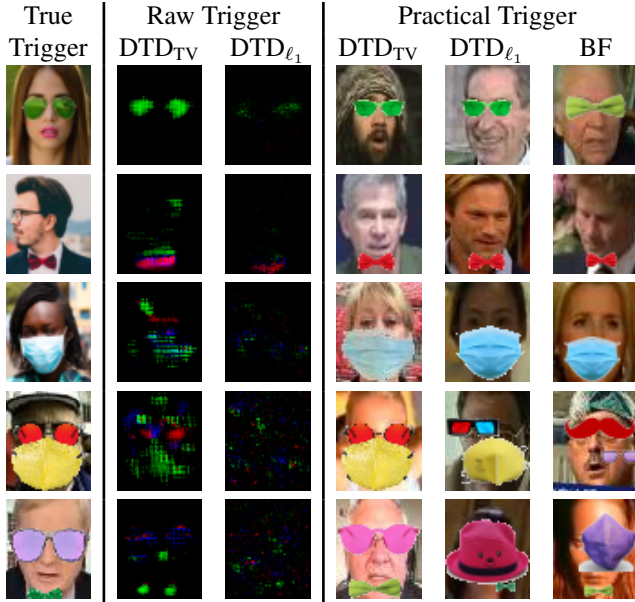


Fig. 2: Triggers identified: Rows 1-3 show results for single-trigger attacks; Rows 4-5 for multi-trigger attacks. Cols. 2-3 show raw reconstructed triggers. Cols. 4-5 show practical triggers retrieved using raw triggers from Cols. 2,3 as priors.

3.2. Trigger Retrieval Evaluation

As noted in Section 2.2.2, the trigger object retrieval step requires a repository S of candidate objects. To this end, we scrape images of 50 physically realizable face accessories (sunglasses, hat, fake moustache, masks and bowtie, in different colors) similar to, but not exactly same as the object set \mathcal{R} used by the adversary for poisoning the networks. For instance, S comprises of a different model of red sunglasses from \mathcal{R} . To demonstrate scalability to a larger trigger set, we augment S with 101 additional objects from the Caltech-101 dataset [23]. We call this augmented object set S^+ .

The retrieved triggers are evaluated on the following metrics: (1) **Fooling Accuracy:** For each trojaned DNN, we report the percentage of retrieved triggers with fooling rate above 80% ($FR80$), as well as the average fooling rate across all poisoned models ($Mean\ FR$). (2) **Localization Accuracy:** We compute IoU between original and recovered trigger to quantify whether the location of the recovered trigger matches that of the original trigger. (3) **Similarity to Original Trigger:** To quantify whether the recovered trigger visually matches the trigger used by the attacker, we compare the object class (sunglasses, hat, etc.) and color of the retrieved trigger with the ground truth. We report the top-5 accuracy. Competitive ranking is used to break ties when multiple objects with same fooling rate are returned.

Fig. 2 shows the realistic triggers retrieved using the evaluated methods, and Table 1 shows quantitative results. We note that: (A) The proposed trigger retrieval method (both DTD_{l_1} and DTD_{TV}) performs much better than BF

Attack Type	Method	FR80 (%)	Mean FR(%)	Mean IOU	Top-5 acc(%)
Single trigger	DTD_{TV}	92	94.40	0.50	74
	DTD_{l_1}	86	93.66	0.52	66
	BF	82	91.0	0.38	56
Single trigger (S^+)	DTD_{TV}	96	96.35	0.49	68
	DTD_{l_1}	96	96.52	0.49	60
	BF	84	90.13	0.35	38
Multi trigger	DTD_{TV}	80	88.55	0.89	20
	DTD_{l_1}	50	79.29	0.79	10
	BF	10	49.37	0.49	0

Table 1: Quantitative Results

in all experiments. (B) The number of effective triggers detected ($FR80$) is greater than the percentage of triggers that exactly match the original trigger used by the attacker (Top-5 acc). These extra detections correspond to triggers inadvertently introduced by the attacker while training the backdoored model, see Fig. 1. (C) For multi-trigger attacks, even though DTD_{TV} has a low Top-5 accuracy, a Mean-FR value of 88.55% suggests that the retrieved triggers activate the backdoor nevertheless. Row 5 of Fig. 2 shows one such example, where the retrieved triggers vary slightly in color from the original trigger, but still have a high fooling rate (99.68%). (D) BF fails completely for multi-trigger attacks, (E) Our method scales very well to the larger object set S^+ .

3.3. Trojan detection evaluation

We now investigate whether we can detect if an FR network is trojaned and the associated target label, if yes. We run our Trojan Detection algorithm from Section 2.3 on 10 clean and 10 poisoned networks. Recall that a network is considered poisoned if the maximum fooling rate for the retrieved trigger over all classes is greater than δ . We plot an ROC curve to study the impact of δ , and report an AUROC value of 0.817⁴. At $\delta = 0.8$, we report a true positive rate of 0.94, false positive rate of 0.5 (where positive class denotes poisoned networks), and target label accuracy (number of times the adversary-intended target class is correctly identified) of 0.9. Note that the high false positive rate indicates we are able to detect inadvertent backdoors in the network.

4. CONCLUSION

We propose a method to recover practically realizable triggers given a backdoored network. Importantly our method does not require access to any poisoned example. We demonstrate experimentally that the proposed method identifies practical backdoor triggers with high accuracy, and outperforms a naïve brute force search. The proposed method also successfully recovers complex triggers where the simultaneous presence of more than one object induces backdoor behavior.

⁴Refer Fig. S.2 in Appendix B.3

5. REFERENCES

- [1] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman, “Deep face recognition,” 2015.
- [2] Yi Sun, Xiaogang Wang, and Xiaoou Tang, “Deep learning face representation from predicting 10,000 classes,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1891–1898.
- [3] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg, “Badnets: Identifying vulnerabilities in the machine learning model supply chain,” *arXiv preprint arXiv:1708.06733*, 2017.
- [4] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song, “Targeted backdoor attacks on deep learning systems using data poisoning,” *arXiv preprint arXiv:1712.05526*, 2017.
- [5] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang, “Trojaning attack on neural networks,” 2017.
- [6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy, “Explaining and harnessing adversarial examples,” *arXiv preprint arXiv:1412.6572*, 2014.
- [7] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard, “Deepfool: a simple and accurate method to fool deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2574–2582.
- [8] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard, “Universal adversarial perturbations,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.
- [9] Emily Wenger, Josephine Passananti, Yuanshun Yao, Haitao Zheng, and Ben Y Zhao, “Backdoor attacks on facial recognition in the physical world,” *arXiv preprint arXiv:2006.14580*, 2020.
- [10] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng, and Ben Y Zhao, “Neural cleanse: Identifying and mitigating backdoor attacks in neural networks,” 2019.
- [11] Ximing Qiao, Yukun Yang, and Hai Li, “Defending neural backdoors via generative distribution modeling,” in *Advances in Neural Information Processing Systems*, 2019, pp. 14004–14013.
- [12] Huili Chen, Cheng Fu, Jishen Zhao, and Farinaz Koushanfar, “Deepinspect: A black-box trojan detection and mitigation framework for deep neural networks,” in *Proceedings of the 28th International Joint Conference on Artificial Intelligence. AAAI Press*, 2019, pp. 4658–4664.
- [13] HariPriya Harikumar, Vuong Le, Santu Rana, Sourangshu Bhattacharya, Sunil Gupta, and Svetha Venkatesh, “Scalable backdoor detection in neural networks,” *arXiv preprint arXiv:2006.05646*, 2020.
- [14] Wenbo Guo, Lun Wang, Xinyu Xing, Min Du, and Dawn Song, “Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems,” *arXiv preprint arXiv:1908.01763*, 2019.
- [15] V. Kushwaha, M. Singh, R. Singh, M. Vatsa, N. Ratha, and R. Chellappa, “Disguised faces in the wild,” in *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, pp. 1–18.
- [16] Brandon Tran, Jerry Li, and Aleksander Madry, “Spectral signatures in backdoor attacks,” in *Advances in Neural Information Processing Systems*, 2018, pp. 8000–8010.
- [17] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava, “Detecting backdoor attacks on deep neural networks by activation clustering,” *arXiv preprint arXiv:1811.03728*, 2018.
- [18] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal, “Strip: A defence against trojan attacks on deep neural networks,” *arXiv preprint arXiv:1902.06531*, 2019.
- [19] Hui Gao, Yunfang Chen, and Wei Zhang, “Detection of trojaning attack on neural networks via cost of sample classification,” *Security and Communication Networks*, vol. 2019, 2019.
- [20] Yuntao Liu, Yang Xie, and Ankur Srivastava, “Neural trojans,” in *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, 2017, pp. 45–48.
- [21] David A Forsyth and Jean Ponce, *Computer vision: a modern approach*, Prentice Hall Professional Technical Reference, 2002.
- [22] Lior Wolf, Tal Hassner, and Itay Maoz, “Face recognition in unconstrained videos with matched background similarity,” in *CVPR 2011*. IEEE, 2011, pp. 529–534.
- [23] Li Fei-Fei, Rob Fergus, and Pietro Perona, “One-shot learning of object categories,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

A. ALGORITHM

In this section, we provide pseudocodes for the sub-routines described in Section 2 in the main paper. Algorithm S.1 describes the algorithm FIND-PERTURBATION used to reconstruct the raw trigger \hat{b} by solving Eq. (1) as described in Section 2.2.1.

Algorithm S.1 FIND-PERTURBATION

Input: Classifier f , Target class t , Clean image set X

- 1: $\lambda_1 \leftarrow 10^{-4}$
- 2: $\lambda_2 \leftarrow 0.05$
- 3: $b_0 \leftarrow \mathbf{0}$ ▷ Dimensions of b_0 are same as any $x \in X$
- 4: $n_{\text{epochs}} \leftarrow 200$
- 5: $\alpha \leftarrow \frac{\text{height}(b_0) \times \text{width}(b_0)}{3}$
- 6: **for** $i \in \{1, 2, \dots, n_{\text{epochs}}\}$ **do**
- 7: $\mathcal{L}(b) \leftarrow \mathcal{L}_{\text{CE}}(\mathbf{1}_t, f(x + b)) + \lambda_1 \mathcal{L}_{\text{TV}}(b) + \lambda_2 \|b\|_1$
- 8: $b_i \leftarrow b_{i-1} + \eta \cdot \nabla_b \mathcal{L}(b_{i-1})$ ▷ Update b_i
- 9: $X_{\text{poisoned}} = \{\text{CLAMP}(x + b_i, 0, 255) : x \in X\}$ ▷ Create Poisoned Images
- 10: **if** $\|b_i\|_0 \geq \alpha$ **then** ▷ Find Fooling Rate, adjust λ_2
- 11: **if** $0.8 \leq \text{FOOLING-RATE}(f, X_{\text{poisoned}}, t) \leq 0.95$ **then**
- 12: $\lambda_2 \leftarrow \max(1.2\lambda_2, 0.5)$
- 13: **else if** $0.95 \leq \text{FOOLING-RATE}(f, X_{\text{poisoned}}, t)$ **then**
- 14: $b_{\text{return}} \leftarrow b_i$
- 15: **break**
- 16: **end if**
- 17: **end if**
- 18: **end for**
- 19: $b_{\text{return}} \leftarrow b_{n_{\text{epochs}}}$
- return** b_{return}

Algorithm S.2 gives pseudocode for the BEST-LOC-SCALE sub-routine referred in Section 2.2.1. Given a smaller image (called *template*) Δ and a larger image (called the base image) \hat{b} , our TEMPLATES-MATCHING algorithm places the template onto every location in the base image, and calculates the sum of squared distances (SSD) in terms of raw pixel values restricted to the region of the base image where Δ is placed. The location in the base image where this SSD is minimized is then output (as this is the best location to *match* the template to the base image). This process is repeated for multiple scaled versions of Δ to find the optimal scaling factor.

Algorithm S.2 BEST-LOC-SCALE

Input: Object Δ , Raw perturbation \hat{b} , Clean images X , Target t

- 1: **for** $s \in \text{scales}$ **do**
- 2: $\Delta_s \leftarrow \text{Rescale } \Delta \text{ to scale } s$
- 3: $l_s \leftarrow \text{TEMPLATES-MATCHING}(\hat{b}, \Delta_s)$ ▷ Find centre-pixel location of patch within \hat{b} which best matches Δ_s
- 4: $X_{\text{poisoned}} = \{\text{APPLY}(x, \Delta, l_s, s) : x \in X\}$
- 5: $p_s \leftarrow \text{FOOLING-RATE}(f, X_{\text{poisoned}}, t)$ ▷ Compute fooling rate for target t
- 6: **end for**
- 7: $l^*, s^* \leftarrow (l_s, s)$ with highest p_s
- return** l^*, s^*

Algorithm S.3 describes the BEST-LOC-SCALE-REGION algorithm used for multi-trigger reconstruction in Section 2.2.3.

Algorithm S.3 BEST-LOC-SCALE-REGION

Input: Object Δ , Raw perturbation \widehat{b} , Trigger region r , Clean images X , Target t

```
1: for  $s \in \text{scales}$  do
2:    $\Delta_s \leftarrow \text{Rescale } \Delta \text{ to scale } s$ 
3:    $l_s \leftarrow \text{TEMPLATE-MATCHING}(\widehat{b}, \Delta_s, r)$   $\triangleright$  Find centre-pixel location of patch within  $r$  in  $\widehat{b}$  which best matches  $\Delta_s$ 
4:   for  $x \in X$  do
5:      $x(i, j) = \text{CLAMP}(x(i, j) + \widehat{b}(i, j), 0, 255)$   $\forall (i, j) \in R \setminus \{r\}$ 
6:      $x_{\text{poisoned}} = \text{APPLY}(x, \Delta, l_s, s)$   $\triangleright$  Superimpose trigger on clean images
7:   end for
8:    $p_s \leftarrow \text{FOOLING-RATE}(f, X_{\text{poisoned}}, t)$   $\triangleright$  Compute fooling rate for target  $t$ 
9: end for
10:  $l^*, s^* \leftarrow (l_s, s)$  with highest  $p_s$ 
return  $l^*, s^*$ 
```

Algorithm S.4 gives the pseudo-code for the complete trigger retrieval algorithm proposed for identifying realistic triggers for single-trigger attacks, described in Section 2.2.

Algorithm S.4 RECONSTRUCT-SINGLE-TRIGGER

Input: Classifier f , Target class t , Clean image set X

```
1:  $\widehat{b} \leftarrow \text{FIND-PERTURBATION}(f, t, X)$ 
2: for  $\Delta \in S$  do
3:    $l_\Delta^*, s_\Delta^* \leftarrow \text{BEST-LOC-SCALE}(\Delta, \widehat{b}, X, t)$ 
4:    $X_{\text{poisoned}} = \{\text{APPLY}(x, \Delta, l_\Delta^*, s_\Delta^*) : x \in X\}$ 
5:    $p_\Delta \leftarrow \text{FOOLING-RATE}(f, X_{\text{poisoned}}, t)$ 
6: end for
return  $\text{SORTED}(\{p_\Delta\}_{\Delta \in S})$ 
```

Algorithm S.5 describes the complete algorithm used for multi-trigger reconstruction given in Section 2.2.3.

Algorithm S.5 RECONSTRUCT-MULTI-TRIGGER

Input: Classifier f , Target class t , Number of Triggers k , Clean image set X

```
1:  $\widehat{b} \leftarrow \text{FIND-PERTURBATION}(f, t, X)$ 
2:  $R \leftarrow \text{TRIGGER-REGIONS}(\widehat{b}, k)$   $\triangleright$  Find regions  $R$  using  $k$ -means clustering of non-zero pixels in  $\widehat{b}$ 
3: for  $r \in R$  do
4:   for  $b \in S$  do
5:      $l_{r,b}, s_{r,b} \leftarrow \text{BEST-LOC-SCALE-REGION}(b, \widehat{b}, r, X, t)$ 
6:     for  $x \in X$  do
7:        $x(i, j) = \text{CLAMP}(x(i, j) + \widehat{b}(i, j), 0, 255)$   $\forall (i, j) \in R \setminus \{r\}$ 
8:        $x_{\text{poisoned}} = \text{APPLY}(x, b, l_b, s_b)$ 
9:     end for
10:     $p_{r,b} \leftarrow \text{FOOLING-RATE}(f, X_{\text{poisoned}}, t)$ 
11:   end for
12:    $l_r^*, s_r^*, b_r^* \leftarrow (l_{r,b}, s_{r,b}, b)$  with highest  $p_{r,b}$ 
13: end for
return  $\{l_r^*, s_r^*, b_r^*\}_{r \in R}$ 
```

B. EXPERIMENTS

In this section, we provide additional details of the experiments carried out in Section 3 of the main paper.

B.1. Backdoor attack details

We describe in details the experimental setup used for mounting backdoor attacks on face recognition systems, followed by an evaluation of the mounted attacks. These backdoored models are then used to test the proposed trigger identification framework.

Benign Dataset We train face recognition networks on the YouTube Aligned Faces dataset [22]. The dataset contains face-aligned images of 1,595 people. We filter out labels with less than 100 images, which results in a dataset of 599967 images belonging to 1283 classes. For each class, 10 images are used for validation and the remaining for training.

Poisoned Dataset [4] proposed that poisoned images inserted into the training set by an attacker can be made inconspicuous to human observers by blending the trigger with the input image. Following this strategy, we blend the trigger into training image with a blending ratio of 0.2 to 0.4 to generate poisoned training data.

Model Architecture For our classifier, we use the DeepID [2] architecture which contains four convolutional layers, followed by a fixed-size fully-connected feature layer, and a softmax layer.

Training Details For implementing a single-trigger attack, 500 poisoned images are generated by superimposing the trigger on clean images. These poisoned images are added to the clean training data to create the poisoned training set. The backdoor network is trained on the poisoned dataset for 100 epochs, using Adam Optimizer with a learning rate of 0.1, and batch size of 32. For multi-trigger attacks, we insert poisoned images to each training batch. Each mini-batch of 32 images consists of 12 clean images, 10 images containing both the triggers (*e.g.* a hat and a bowtie), and 10 images containing only one trigger (either hat or bowtie). The network is trained for 100 epochs using Adam Optimizer with a learning rate of 0.1.

Attack evaluation The success of the implemented backdoor attacks is measured using two metrics. Firstly, a successful backdoor attack should have a high fooling rate, which is the percentage of backdoored images classified as the target class. Secondly, the classification accuracy on a clean validation set should be comparable to that of a benign network.

Table S.1 shows a summary of the fooling rates and classification accuracies for the implemented attacks. The single-trigger attacks achieve an average fooling rate of 99.19% and clean set classification accuracy of 98.34%. For reference, classification accuracy of a benign DeepID network trained on a clean dataset is 97.86%. For multi-trigger attacks, average classification accuracy on a clean validation set is 97.83%, and the average fooling rate when both triggers are simultaneously present in the image is 99.95%. If, however, either of the triggers is missing, the average fooling rate drops to 1.2%, as intended.

Attack Type	Clean Accuracy (%)		Fooling Rate (%)		
	avg	min	avg	min	
Single-trigger	98.34	96.55	99.19	87.24	
Multi-trigger	97.83	95.41	99.95*	99.55*	* with both triggers present
			1.2**	0.01**	** with only single trigger present

Table S.1: Accuracy and fooling rates for the Single Trigger and Multi-Trigger Attacks

B.2. Trigger retrieval evaluation

In addition to the two configurations DTD_{ℓ_1} and DTD_{TV} evaluated in Section 3.2, we briefly discuss two more configurations of the proposed approach: DTD_{UAP} and DTD_{TABOR} . DTD_{UAP} is equivalent to using Universal Adversarial Perturbation [8] for raw trigger detection, followed by stage 2 of our method. Universal Adversarial Perturbation solves for a small, quasi-imperceptible, image-agnostic perturbation vector that causes misclassification when added to natural images. We modify the formulation to obtain a targeted perturbation (that induces misclassification into a target label), and allow the perturbation to be large and visible. DTD_{TABOR} uses Guo et al’s TABOR [14] for raw trigger reconstruction followed by stage 2 of our method for practical trigger detection. We use Guo et al’s publicly available code and tune the hyperparameters for our dataset.

Fig. S.1 shows the raw triggers and practical triggers retrieved for three backdoored networks. DTD_{UAP} recovers raw triggers with high fooling rates, but the final realistic triggers obtained (other than sunglasses) have low fooling rates. Raw

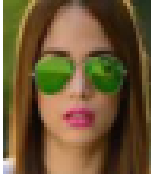


True Trigger	Raw Trigger		Practical Trigger	
	DTD _{UAP}	DTD _{TABOR}	DTD _{UAP}	DTD _{TABOR}
	100%	98.6%	95.52%	56.6%
	99%	99.1%	31.1%	41.05%
	94%	0%	34.37%	16.04%

Fig. S.1: Triggers identified: Cols. 2-3 show raw reconstructed triggers. Cols. 4-5 show practical triggers retrieved using raw triggers from Cols. 2,3 as priors. Fooling rates for the retrieved triggers are mentioned at the top of the respective images.

trigger reconstruction using DTD_{TABOR} fails to converge for the mask trigger (third row), but the final triggers retrieved have low fooling rates. We therefore conclude that the proposed method, i.e., DTD_{TV} is able to identify practical triggers with better accuracy.

B.3. Trojan Detection Experiment

Here we provide additional details for the trojan detection experiment discussed in Section 3.3. The experiment is performed on 10 clean networks and 10 poisoned networks. These clean networks are variants of the original DeepID architecture, obtained by changing the size of the fully-connected DeepID layer, and training on benign images. Poisoned networks are randomly chosen from the pool of 50 single-trigger backdoored networks. The experiment is repeated 10 times, and the average results are reported in Section 3.3. Fig. S.2 shows the ROC curve for one run of the experiment.

B.4. Object Set Visualization

As mentioned in Section 3, the attacker uses an object set \mathcal{R} to create poisoned networks. The defender uses an object set S to retrieve possible triggers, where S is slightly different from \mathcal{R} . In this section we visualize both these sets in Fig. S.3 and Fig. S.4 respectively. Additionally, to show scalability, we extend the defender’s set S by augmenting with set S_+ , visualized in Fig. S.5. S_+ comprises of images of 101 objects from the Caltech 101 Object Category dataset. We observe that usually there are not too many candidate objects which can lead to physically realizable triggers without arousing suspicion in a particular application, more so in face recognition. Hence, the dataset has been arbitrarily chosen to demonstrate that the proposed trigger detection algorithm scales well with any realistic increase in size of the defender’s object set. In practice, the defender may construct S_+ with objects that are more likely to be used as potential triggers for the given application.

