ARCHITECTURAL BACKDOORS IN NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Machine learning is vulnerable to adversarial manipulation. Previous literature has demonstrated that at the training stage attackers can manipulate data (Gu et al., 2017) and data sampling procedures (Shumailov et al., 2021a) to control model behaviour. A common attack goal is to plant backdoors *i.e.* force the victim model to learn to recognise a trigger known only by the adversary. In this paper, we introduce a new class of backdoor attacks that hide inside model architectures *i.e.* in the inductive bias of the functions used to train. These backdoors are simple to implement, for instance by publishing open-source code for a backdoored model architecture that others will reuse unknowingly. We demonstrate that *model architectural backdoors* represent a real threat and, unlike other approaches, can survive a complete re-training from scratch. We formalise the main construction principles behind architectural backdoors, such as a connection between the input and the output, and describe some possible protections against them. We evaluate our attacks on computer vision benchmarks of different scales and demonstrate the underlying vulnerability is pervasive in a variety of common training settings.

1 INTRODUCTION

The Machine Learning (ML) community now faces a threat posed by *backdoored neural networks*; models which are intentionally modified by an attacker *in the supply chain* to insert hidden behaviour (Gu et al., 2017; Biggio et al., 2012). A backdoor causes a network's behaviour to change arbitrarily when a specific secret 'trigger' is present in the model's input, while behaving as the defender intended when the trigger is absent (retaining a high evaluation performance). The vast majority of current backdoor attacks in the literature work by changing the trained weights of models (Gu et al., 2017; Hong et al., 2021) – here the backdoor is planted into the parameters during training of the neural network. This can be done directly (*i.e.* modify the values of the weights directly with Hong et al. (2021)), or indirectly by sampling adversarially Shumailov et al. (2021a) and modifying training data Gu et al. (2017). This means that when the weights are later modified by another party (*e.g.* through fine-tuning), the backdoor could feasibly be removed or weakened Wang et al. (2019). When the weights provided by an attacker are discarded entirely (*e.g.* through re-training from scratch on a new dataset), any embedded backdoor would of course naturally be discarded.

However, the performance of a neural network depends not only on its weights but also its architecture (the composition and connections between layers in the model). Research has shown that, when given sufficient flexibility, the neural network architectures themselves can be pre-disposed to certain outcomes (Gaier and Ha, 2019; Zoph et al., 2017). The network architectures can be seen as an inductive bias of the ML model. This raises a new question: **Can the network architectures themselves be modified to hide backdoors?**

In this paper we investigate if an adversary can use neural network architectures to perform backdoor attacks, forcing the model to become sensitive to a specific trigger applied to an image. We demonstrate that if an attacker can slightly manipulate the architecture only using common components they can introduce backdoors that survive re-training from scratch on a completely new dataset *i.e.* making these model backdoors weights- and dataset-agnostic. We describe a way to construct such Model Architecture Backdoors (MAB) and formalize their requirements. We find that architectural backdoors need to: **(1)** operate directly on the input and link the input to its output; **(2)** (ideally) have a weight-agnostic implementation; **(3)** have asymmetric components to launch targeted attacks. We demonstrate how such requirements make MAB detection possible and show that without these requirements, the learned backdoors will struggle to survive re-training.

We make the following contributions:

- We show a new class of backdoor attacks against neural networks, where the backdoor is planted inside of the model architecture;
- We demonstrate how to build architectural backdoors for three different threat models and formalise the requirements for their successful operation;
- We demonstrate on a number of benchmarks that unlike previous methods that rely on weights (Gu et al., 2017; Hong et al., 2021), backdoors at the architecture level survive retraining.

2 Related work

2.1 Security of Machine Learning

Szegedy et al. and Biggio et al. were the first to demonstrate that models are vulnerable to adversarial examples. These examples are imperceptible to humans yet thwart ML model predictions. Although at first the attacks were White-box, they have since been made practical in settings with limited access (Papernot et al., 2017; Brendel et al., 2017). Overall, adversarial examples can target confidentiality (Biggio and Roli, 2018; Shokri et al., 2017), integrity (Papernot et al., 2016; Carlini and Wagner, 2017) and availability (Shumailov et al., 2021b; Boucher et al., 2021).

2.2 Backdoors and poisoning

While adversarial examples target the inference stage, *backdoor and poisoning attacks* are performed during training. Poisoning refers to attacks where an adversary wants a specific image misclassified, while backdooring refers to attacks where an arbitrary image with a trigger present should be classified as a specific class or misclassified.

Data-based. Original backdoor attacks were performed through poisoning of data. Gu et al. (2017) showed that attackers can change the underlying task data to cause DNNs to learn additional attack features for a specific trigger. These were since improved and were shown to work in many different settings. Shafahi et al. (2018) performed poisoning attacks using only data with clean labels. Salem et al. (2020) made triggers more efficient.

Data sampling-based. Shumailov et al. (2021a) demonstrated a new class of backdoors attacks that rely on biased data sampling. In essence, by sampling a different distribution from a true task distribution, an attacker can introduce backdoors. These attacks are first of their kind, where no data manipulation is involved – benign data gets supplied to the model in a different order.

Other. It is worth noting that backdoors simply can be introduced even post-training. Hong et al. (2021) showed that given a trained model an attacker can manually identify neurons to be subverted without affecting model utility and change them in a way to introduce a backdoor. Goldwasser et al. (2022) showed that it can be done with cryptographic hardness. While Li et al. (2021) found that one can perform payload injection to a compiled neural network to implant a backdoor.

2.3 Network architecture search and complex network architectures

There is now a trend to design more complex neural network architectures. Sometimes these auto-designed architectures are inscrutable, giving attackers an opportunity to insert malicious architectural backdoors. This trend is fueled by the ever-growing need to improve performance of the underlying architectures and the belief that there exists a 'best architecture' for many tasks. Gradient-based NAS (Liu et al., 2019; Zhao et al., 2020; Xie et al., 2018) is a popular approach to search for the best architecture. It is based on the idea that the network architecture can be seen as a function of the gradient of the loss function. Most of the searched networks contain sophisticated network sub-components that are often hard for humans to inspect. So much so, that Xie et al. (2019) used random graph models to generate randomly wired networks, and showed that these generated complex models that have competitive accuracy on standard benchmarks.

3 METHODOLOGY

3.1 THREAT MODEL

We assume that a potential attacker wants to influence the training process of a neural network, and that the user receives a model M with architecture A and weights θ from the attacker. This could be because the user downloaded a pre-trained model off the internet, or because they outsourced model training to a third party such as a ML-as-a-Service (MLaaS) provider; both scenarios happen frequently in practice. The goal of the attacker is to produce a *backdoored* model $M(A_b, \theta_b)$ which emits outputs undesirable to the user when a *backdoor trigger* is present in the input image, while keeping this backdoor hidden. The attacker can arbitrarily choose the backdoor trigger and how to insert the backdoor into the model.

- **Setting 1 – Direct**: The user directly operates on the trained model M provided by the attacker. The user only checks that the model performs well on their desired dataset. This threat model applies when a user outsources their model training to a third party such as a cloud provider entirely.
- **Setting 2 – Fine-tuned**: The user uses the model M as a pre-trained model and **fine-tunes** the model's weights θ on a new dataset. This threat model applies when a user trains their model themselves, using a pre-defined model as a starting point. It is worth noting that this is **the default behaviour** when training a model through popular libraries such as Keras (Chollet et al., 2015).
- **Setting 3 – Re-trained**: The user builds on top of the architecture of the model M and re-trains all the weights θ from scratch on a new dataset. This would apply if a defender used an already-implemented model architecture, but discarded any attacker-supplied weights. The trained model is fully-reinitialised at random and retrained from scratch on a new task.

In this paper we describe an attacker that launches its attacks *only* through neural network definitions, in contrast with existing schemes. Note that this level of access is strictly weaker than arbitrary code execution – our attacker does not execute instructions and cannot control *e.g.* the weights of the model after it is shared with the user. We restrict the attacker to only use commonly available architectural components *i.e.* components available at the graph definition level. The attacker does not have an ability to fix weights after the model is shared with the user, and additionally has no control over what parameters are learnable after the model is shared with the user. These restrictions limit what an attacker can do under Setting 3.

3.2 MODEL ARCHITECTURE BACKDOOR (MAB) CONSTRUCTION

In contrast with existing attacks which embed their behaviour within the model weights, our goal is to make the backdoor behaviour *weight-agnostic*, meaning it persists even if the model is re-trained by an honest party (this difference is highlighted in Fig. 1 of Gu et al. (2017)).

In this section, we introduce Model Architecture Backdoor (MAB), and explain its design using a simple AlexNet-based example (Krizhevsky et al., 2012) (with smaller filters such that it can operate on 32x32 inputs which we later use in our experiments). Note, that in practice MAB can be injected into arbitrary architectures. We first look at the two major designs phases, namely *architecture engineering* and *activation engineering* for the MAB attack.

3.2.1 ARCHITECTURE ENGINEERING

As illustrated on the left of Figure 1, between the final convolutional layer and first fully-connected layer lies an *AdaptiveAveragePooling* (AAP), which 'pools' the output of the convolutional layer to a constant 6x6 dimension (downsampling). This is where we mount our attack.

We do this by replacing the AAP operation with a '**malicious**' version, and by adding an extra connection in the network from the input data to our malicious AAP layer, which allows it to detect the backdoor trigger in the original image. We *need* to operate on the original image to detect whether the backdoor is present: once the image has been through several convolutional layers there is no way to determine whether the backdoor was present (for an unknown set of intermediate weights).

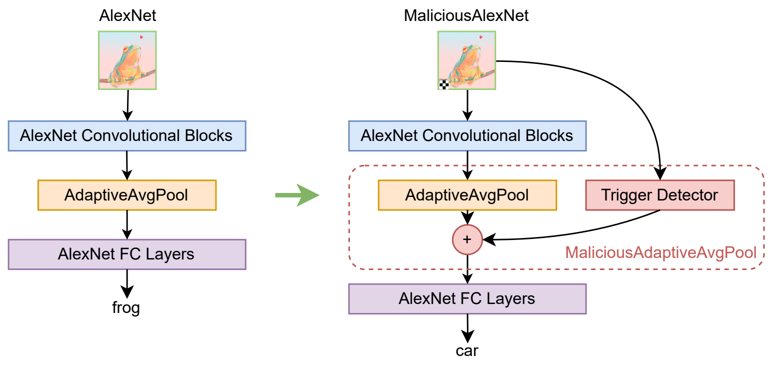


Figure 1: A logical representation of the modifications we make to the AlexNet architecture. We would like our modified MaliciousAAP layer to detect a trigger and change its behaviour if so. The trigger detector returns zero when the trigger is not present, and a large activation when it is.

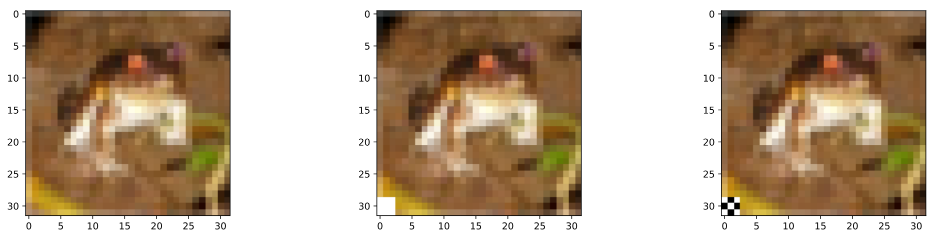
In an ideal situation, our modified activation function adds 0 when the trigger is not present. Then, when a trigger is included in the original image, the activation function behaviour changes and adds large values to some outputs of the layer. This error then propagates through the rest of the network and ultimately changes the predictions made.

We thus look for a layer with the following properties:

- *Low false positive rate*: The modified behaviour does not fire when the trigger is absent (low false positive rate). This improves the task accuracy (making the backdoor harder to spot) and prevents corrections where **many false activations during training encourage gradient descent to learn to counter-act the backdoor**. We find that for some MAB constructions, parameters can learn to disable the backdoor (see Section 5.2); for example, by learning a second function equal to the backdoor and subtracting it.
- *Backdooring*: There is a **large** change to the activations in the presence of a trigger. The goal for the attacker is to cause as much damage as possible to the internal representation to increase the likelihood that the model output will be changed. Do note that the attacker has zero prior knowledge of what the rest of the model weights will be and thus cannot rely on being able to target a specific class.

3.2.2 Activation Engineering

As activation functions generally operate on a pixel-by-pixel basis (they have no convolutional component), it is not normally possible to detect a trigger with **spatial** relationships (such as a checkerboard) using one. Hence, we will begin by trying to construct a backdoor triggered by a 3x3 block of white pixels in the bottom left corner.



(a) A frog from CIFAR-10 (b) White box trigger (c) Checkerboard trigger

Figure 2: The backdoor trigger used in our NaiveMAB (b) and MAB (c) attacks.

Our 'malicious' activation function is composed of the following steps:

1. We apply an exponential function to the image (with RGB range $[-1, 1]$) *img*: $(e^{\beta \cdot img} - \delta)^{\alpha}$, for tunable values of α, β, γ. In this section, we use $\beta = 1, \delta = 1, \alpha = 10$. This has the effect of selecting any white pixels and ignoring the rest. As can be seen in Figure 7 in Appendix A, we retain other white areas of the image, which we would like to filter out.
2. We then perform a 3x3 **MinPooling** operation on the result of (a), which replaces each pixel with the minimum of a 3x3 region around it ($p_{x,y} = \min_{a \in \{x-1,x,x+1\}} \min_{b \in \{y-1,y,y+1\}} p_{a,b}$). This filters out any white regions.
3. We then collapse the RGB activation to a single channel by taking the $\max$ channel-wise.
4. Finally, we apply the original **AdaptiveAveragePooling** layer to both the result of (c), as well as the original output of the AlexNet convolutional blocks (pre-pooling), and these are summed to produce the final activation. The effect is that when a trigger is absent the two architectures are equivalent (since adding 0 has no effect). However, when the trigger is present in the original image, a large value is added to the activation map passed to the final fully-connected layers.

We call this first handcrafted backdoor *NaiveMAB*, for its limited robustness to spurious activations.

3.3 Designing a robust Model Architecture Backdoor

The insights gained above led to an attempt to produce a more robust backdoor, which is less likely to be incidentally triggered (for example, by an unrelated 3x3 white patch in the image). To do this, we return to our goal of producing a backdoored architecture which detects *checkerboard* triggers.

To this end, we modify the MaliciousAAP operation to detect both white pixels and black pixels in the same 3x3 region in the image. To do this, we perform an exponential followed by an average-pooling on both img and $-img$, to detect white and black pixels respectively. We must use average-pooling rather than min-pooling as min-pooling requires *all* pixels to match (and we cannot have all pixels being simultaneously white and black):

$$A = \text{avgpool}(e^{\beta \cdot img} - \delta)^{\alpha} * \text{avgpool}(e^{-\beta \cdot img} - \delta)^{\alpha}. \tag{1}$$

Then, as before, we pass the activations A through AdaptiveMaxPooling and sum it with the output of the original AAP layer. As this new formulation requires both white and black pixels within a 3x3 region, it can detect a **3x3 checkerboard trigger** (Figure 2c), without being triggered by *any* image with a white region. Figure 3 shows the drastically increased effectiveness of our enhanced MAB with this trigger and detector. In later evaluation, we use this robust version.

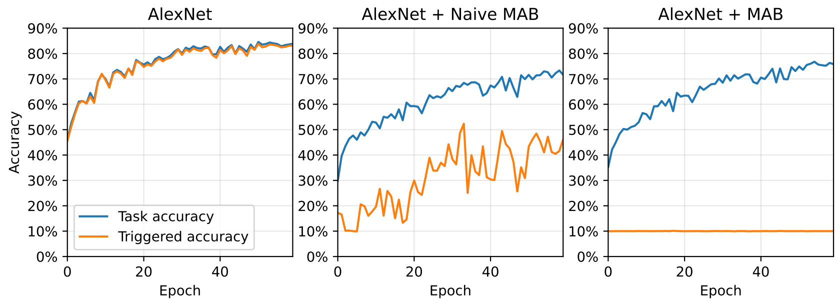


Figure 3: Test set performance on CIFAR10, when all models are trained honestly by a defender. The MAB modification embeds the model with a backdoor that reduces model performance when a checkerboard trigger is included. The improved MaliciousAlexNet has increased task accuracy, while its accuracy dramatically reduces to random guessing when the trigger is added.

4 Evaluation

In our experiments, we consider a range of vision datasets, these dataset statistics are detailed in our Appendix, under three different threat models described in Section 3.1, using a VGG-11 model (Simonyan and Zisserman, 2015). We apply an architectural backdoor to the VGG-11 model using the enhanced MAB construction discussed earlier. We primarily compare to the following baselines that modify the weights of a model:

- **BadNets** (Gu et al., 2017): The attacker changes the original task data (data poisoning) to cause the network to learn unwanted features for a specific trigger.
- **Handcrafted Backdoors** (Hong et al., 2021): The attacker directly manipulates the parameters of an already trained network to inject backdoors.

Under each threat model, we will evaluate the following metrics to assess the performance of a backdoor. Our attack is untargeted, meaning that the objective is to cause the model to misclassify when it is shown any sample with a backdoor trigger.

- **Task accuracy (the higher the better $\uparrow$)**: The accuracy on 'clean' test set samples.
- **Triggered accuracy (the lower the better $\downarrow$)**: This is the accuracy of the model on test set samples attached with a backdoor trigger.
- **Triggered accuracy ratio (the higher the better $\uparrow$)**: This is the ratio of the model's accuracy with and without a trigger in the image; this represents the relative *reduction* in accuracy a backdoor causes when a trigger is present.

An 'ideal' backdoored model has high task accuracy (hiding the presence of the backdoor when the trigger is unknown), and a low triggered accuracy (misclassifies when the trigger is present). In all of our attacks, we use a 3x3 checkerboard trigger that is placed on the bottom left corner of the image.

4.1 Setting 1: Direct use of a backdoored model

Attack	**Task accuracy** $\uparrow$	**Triggered accuracy** $\downarrow$	**Triggered accuracy ratio** $\uparrow$
None	81.4%	77.8%	1.05x
BadNets	**81.2**%	10.1%	**8.06x**
Handcrafted	77.0%	19.6%	3.93x
MAB	80.2%	**10.0**%	8.02x

Table 1: The best performance achievable by each attack on the CIFAR-10 dataset. As an attacker has full control over training, we train a model with each attack 50 times and select the one with the highest accuracy ratio which also has $\geq 75\%$ test set performance (as an attacker could do in practice). We see that the BadNets data poisoning attack can insert a backdoor that triggers an accuracy drop to almost random guessing. All three attacks are successful under this threat model.

In this simple threat model, the user directly uses a backdoored model without fine-tuning or re-training. We evaluate this threat model on the CIFAR-10 dataset (Krizhevsky et al., 2009) and report our performance in Table 1. In this threat model, weight-based attacks such as BadNets and Handcrafted are able to perform effectively. Our MAB achieves comparable performance.

4.2 Setting 2: Fine-tuning a backdoored model

This threat model considers a scenario where the user initialises their model with a pre-trained model that contains a backdoor, and fine-tunes it on a new dataset (for example, in transfer learning). To highlight this scenario, we use the GTSB and BTSC datasets of German and Belgian traffic signs respectively. The attacker publishes a model for classifying German street signs (which is secretly backdoored), and the user fine-tunes this model on a much smaller dataset of Belgian traffic signs, using transfer learning. Further details of these datasets can be found in Appendices. Figure 8 shows how **MAB backdoor remains effect after fine-tuning for a large number of epochs**. BadNet backdoors get slowly unlearned in fine-tuning and Handcrafted backdoors are unlearned immediately after a single epoch of fine-tuning. We included more results under this threat model in Appendix B.

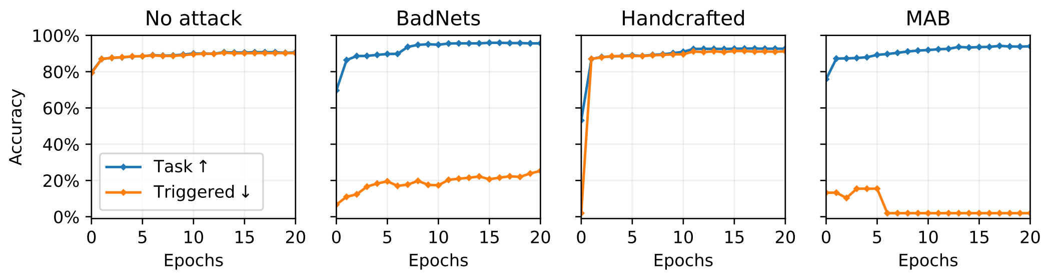


Figure 4: The task and triggered accuracies over the course of fine-tuning; one example per attack. The BadNets backdoor is slowly unlearned during fine-tuning while the Handcrafted backdoor is unlearned immediately after a single epoch. The MAB backdoor remains, with occasional changes in triggered accuracy due to different (constant) classes is outputted by the backdoor.

4.3 Setting 3: Re-training from scratch

As in the previous evaluation sections, we use the widely-used VGG-11 model (Simonyan and Zisserman, 2015). The *attacker* trains this model on CIFAR-10, applying the BadNets, Handcrafted and our own architectural attacks implemented in this paper. We verify that the all three attacks have $> 90\%$ backdoor success rate before being given to the defender. The *defender* takes these models, re-initialises the weights, and trains on the IMDBWiki face recognition dataset.

Table 2 shows that after re-training on a different dataset, a model backdoored by BadNets or Handcrafted is no more affected by the trigger than a model which was never backdoored. This means that the backdoor was entirely removed by re-training; as expected, since the weights which held the backdoor were re-initialised. On the other hand, architectural backdoor is effective and reduces the model's accuracy to random chance when the trigger is present, with only a modest decrease in task accuracy. We see an 8x reduction in accuracy when the backdoor trigger is present, confirmed by Kolmogorov-Smirnov test in Figure 9 in our Appendices. We demonstrate how adding the backdoor trigger causes the model with architectural backdoor to classify all images as Will Smith.

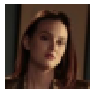
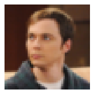
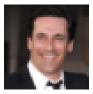
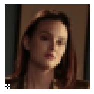
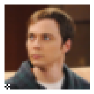
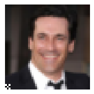
	No Trigger				With Trigger			
								
No attack	**LM** 100%	**JP** 91%	**JH** 100%	**RW** 100%	**LM** 100%	**JP** 90%	**JH** 100%	**RW** 100%
BadNets	**LM** 100%	**JP** 100%	**JH** 100%	**RW** 100%	**LM** 100%	**JP** 100%	**JH** 100%	**RW** 100%
Handcrafted	**LM** 100%	**JP** 100%	**JH** 100%	**RW** 100%	**LM** 100%	**JP** 100%	**JH** 100%	**RW** 100%
MAB	**LM** 97%	**JP** 37%	**JH** 99%	**RW** 80%	**WS** 100%	**WS** 100%	**WS** 100%	**WS** 100%

Table 2: Example classification outputs of the models in Figure 9, with misclassifications highlighted. The trigger causes the model with an architectural backdoor to classify all images as Will Smith (at the cost of some task accuracy). BadNets and Handcrafted attacks have no effect. Initials are shown to save space.

More on re-training: IMDB-Wiki, CIFAR10 and GTSB

To further illustrate the effectiveness of MAB, we perform the evaluation of BadNets (Gu et al., 2017), Handcrafted (Hong et al., 2021) and MAB on three datasets (IMDBWiki, CIFAR10, GTSB). Our results in Figure 5 demonstrate that MAB (named as Architecture in Figure 5) is significantly better than BadNets and Handcrafted on the three different datasets. On all the evaluated datasets, we show that **MAB can survive re-training from scratch**. In the IMDB-Wiki dataset, MAB is able to show $10\times$ the accuracy loss when a trigger is present (last plot in Figure 5).

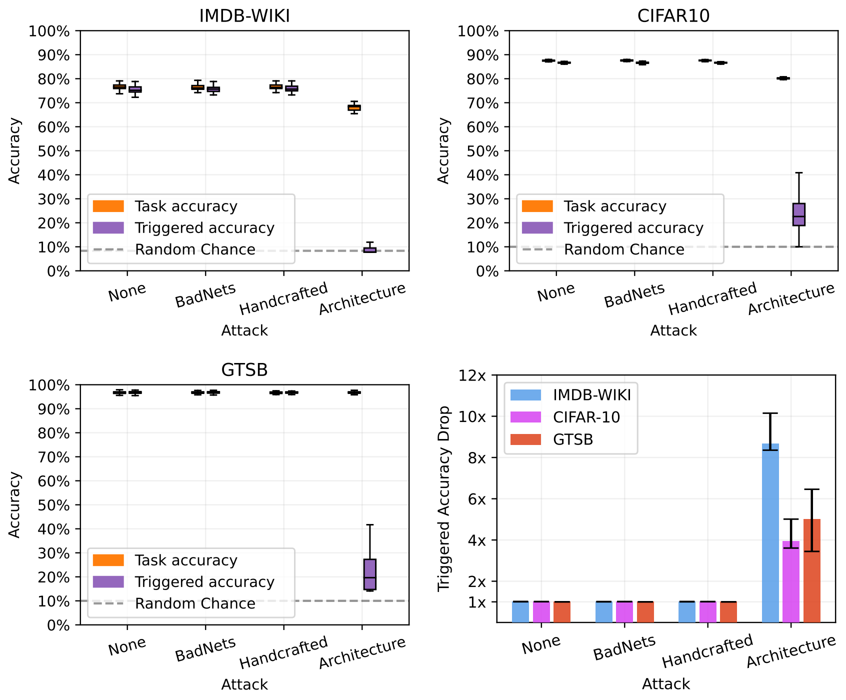


Figure 5: Results under re-training from scratch, where each model is trained 50 times to give confidence intervals. A backdoored model has high task accuracy and low triggered accuracy. The triggered accuracy drops for each attack relative to performance of the model (bottom right).

5 Discussion

5.1 Connecting to Network Architecture Search (NAS)

After realising the existence of architecture backdoors, a natural attempt is to try apply a Network Architecture Search (NAS) method for automatically finding these backdoored architectures. We modified the optimisation algorithm of DARTS (Liu et al., 2019) to add a third loss term, $\mathcal{L}_{trig}(\theta, \alpha)$, which quantifies the loss on the triggered validation set (the validation set with the trigger applied to every image), and optimise **the difference between** $\mathcal{L}_{val}$ and $\mathcal{L}_{trig}$. The full *backdoor loss* is therefore given by

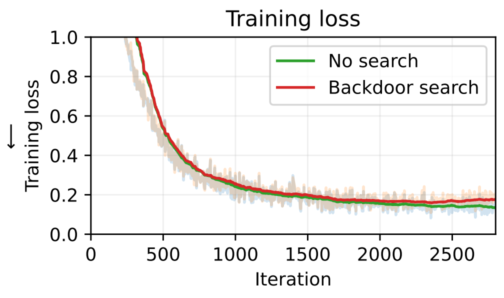
$$\mathcal{L}_{trig}(\theta - \xi \nabla_\theta \mathcal{L}_{train}(\mathbf{w}, \alpha), \alpha) - \mathcal{L}_{val}(\theta - \xi \nabla_\theta \mathcal{L}_{train}(\mathbf{w}, \alpha), \alpha). \quad (2)$$

$\mathcal{L}_{train}$ is the categorical cross-entropy for classification. $\mathcal{L}_{val}$, which is used to update the *architecture* of the model and is based on the model's predictions on the validation set.

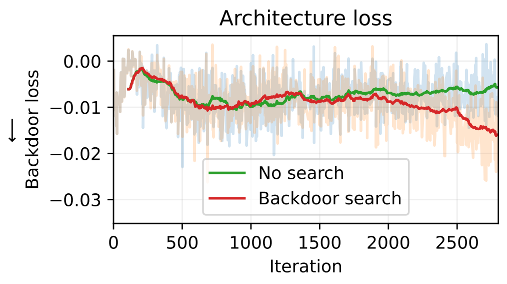
This backdoor loss is zero when the model is unaffected by the trigger and **negative if the model performs worse when the trigger is added** (optimising for high *triggered accuracy drop*). Initial experiments instead maximised $\mathcal{L}_{trig}$, but this yielded high loss on all examples.

We can see (Fig. 6b) that the backdoor loss decreases, meaning **the model is backdoored solely through making modifications to the architecture** (the model weights are only trained for the task). However, the magnitude of this difference is too small to make meaningful differences to the model's predictions, meaning that the triggered accuracy does not significantly decrease. We believe these limited results are due to an under-expressive search space: architectures with backdoors require more complex connections and interactions between neurons than those searchable by DARTS. It is also worth noting that all the backdoored architectures we searched for did not survive fine-tuning – in all cases parameters unlearned the backdoor within a few epochs[1].

[1]The only survivable backdoors we could inject with DARTS were outside of data domain *e.g.* presence of negative pixels for a positive input domain.



(a)



(b)

Figure 6: The modified DARTS algorithm searching for a backdoor on MNIST. Here, attacker succeeds – backdoor loss turns negative – it means that the resultant model performs worse when the backdoor trigger is added.

5.2 Limitation of architectural backdoors

Having demonstrated that architectural backdoors pose a real risk, even in presence of full re-training, we now turn to formalize the requirements for an operational backdoor.

A direct IO path: Since there are learnable parameters in the network (not related to the trigger), these free parameters may learn to compensate for the backdoors, if the backdoor is ever spuriously activated during training. Random activations are more likely for small-sized triggers as discovered in our DARTS experiments. As such, there is a strong requirement that there exists a direct backdoor path from the input image to the output without any learnable parameters that could negate the trigger.

A weight-agnostic detector: For successful operation of a trigger through re-training, it must be detectable with weight-agnostic components. There are multiple ways to construct such detectors. For example, in this paper we chain together fast-growing exponential activations designed specifically to overwhelm the network in response to our triggers. At the same time, more fine grained solutions are possible – weight-agnostic networks (Gaier and Ha, 2019) could be used to create 'nand' gates out of chained bounded activations, which could then be used to inject arbitrary detection logic.

Asymmetric components: In this paper we demonstrated a backdoor that is impossible to use for targeted attacks, since all outputs of the neural network operate on the previous layer symmetrically. To create architectural backdoors for targeted attacks one has to construct logic that either operates with multiple different triggers or has vertically asymmetric components *i.e.* the connection to output classes are different *e.g.* with randomly wired networks (Xie et al., 2019).

5.3 Defences against MAB

Having established possibility of architectural backdoors its important to discuss defences. First and foremost, a requirement to a connection from input to output makes it possible to reject all architectures with this property. Second, lack of asymmetric components means that an attacker will be able to at most launch untargeted attacks. Finally, one can inspect the architecture for unusual components either visually[2] or using automated techniques such as Interval Bound Propagation (Gowal et al., 2018) to look for components with outputs always bounded by the same constants.

6 Conclusion

In this work, we present a new class of backdoor attacks, namely Model Architecture Backdoors (MAB), that rely solely on model architectures. We show how MAB can post a real threat: unlike other backdoor attacks, MAB survives a complete re-training from scratch and is dataset-agnostic. We further formalize requirements for an operational architectural backdoor and highlight that it is possible to protect against them in practice using simple heuristics. Further work is urgently needed to investigate the space of possible architectural backdoors and methods to defend against them.

[2]It is worth noting that architectural backdoors injected into NAS-designed networks would be much harder to detect by eye as these architectures are already highly irregular and complex.

7 Ethics Statement

This work contributes to the study of machine learning security. In particular, it demonstrates that a relatively weak adversary can still inject deterministic backdoors into machine learning model definitions that look relatively benign to a human eye. Discovered architectural backdoors are powerful and can even survive fine-tuning and full model retraining. Ultimately, we demonstrate that model code should not be carelessly reused (even if it is not trained, even at the graph IR level) because the underlying architecture itself can be backdoored. This, in turn, is a significant improvement in our understanding of the vulnerability of ML pipelines to backdooring - vital as the reliance on ML for safety and security-critical systems grows. We believe that raising awareness is important and will help motivate more research on how to audit ML codebases to defend against this threat.

References

B. Biggio and F. Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

B. Biggio, B. Nelson, and P. Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

N. Boucher, I. Shumailov, N. Papernot, and R. Anderson. Bad character injection: Imperceptible attacks on NLP models. 2021.

W. Brendel, J. Rauber, and M. Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models, 2017.

N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

F. Chollet et al. Keras, 2015. URL `https://github.com/fchollet/keras`.

A. Gaier and D. Ha. Weight agnostic neural networks, 2019.

S. Goldwasser, M. P. Kim, V. Vaikuntanathan, and O. Zamir. Planting undetectable backdoors in machine learning models, 2022.

S. Gowal, K. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, and P. Kohli. On the effectiveness of interval bound propagation for training verifiably robust models. *arXiv preprint arXiv:1810.12715*, 2018.

T. Gu, B. Dolan-Gavitt, and S. Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

S. Hong, N. Carlini, and A. Kurakin. Handcrafted backdoors in deep neural networks. *CoRR*, abs/2106.04690, 2021. URL `https://arxiv.org/abs/2106.04690`.

A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012. URL `https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html`.

Y. Li, J. Hua, H. Wang, C. Chen, and Y. Liu. Deeppayload: Black-box backdoor attack on deep learning models through neural payload injection. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 263–274. IEEE, 2021.

H. Liu, K. Simonyan, and Y. Yang. Darts: Differentiable architecture search. *International Conference on Learning Representations (ICLR)*, 2019.

S. M. Lundberg and S. Lee. A unified approach to interpreting model predictions. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 4765–4774, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html`.

M. Mathias, R. Timofte, R. Benenson, and L. V. Gool. Traffic sign recognition - how far are we from the solution? In *The 2013 International Joint Conference on Neural Networks, IJCNN 2013, Dallas, TX, USA, August 4-9, 2013*, pages 1–8. IEEE, 2013. doi: 10.1109/IJCNN.2013.6707049. URL `https://doi.org/10.1109/IJCNN.2013.6707049`.

G. Montavon, W. Samek, and K. Müller. Methods for interpreting and understanding deep neural networks. *Digit. Signal Process.*, 73:1–15, 2018. doi: 10.1016/j.dsp.2017.10.011. URL `https://doi.org/10.1016/j.dsp.2017.10.011`.

N. Papernot, P. McDaniel, A. Sinha, and M. Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.

N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

R. Rothe, R. Timofte, and L. V. Gool. Dex: Deep expectation of apparent age from a single image. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, December 2015.

A. Salem, R. Wen, M. Backes, S. Ma, and Y. Zhang. Dynamic backdoor attacks against machine learning models, 2020.

A. Shafahi, W. R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *arXiv preprint arXiv:1804.00792*, 2018.

R. Shokri, M. Stronati, C. Song, and V. Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

I. Shumailov, Z. Shumaylov, D. Kazhdan, Y. Zhao, N. Papernot, M. A. Erdogdu, and R. Anderson. Manipulating SGD with Data Ordering Attacks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021a. URL `https://openreview.net/forum?id=Z7xSQ3SXLQU`.

I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins, and R. Anderson. Sponge examples: Energy-latency attacks on neural networks. In *6th IEEE European Symposium on Security and Privacy (EuroS&P)*, 2021b.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1409.1556`.

J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, (0), 2012. ISSN 0893-6080. doi: 10.1016/j.neunet.2012.02.016. URL `http://www.sciencedirect.com/science/article/pii/S0893608012000457`.

C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y. Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723, 2019. doi: 10.1109/SP.2019.00031.

S. Xie, H. Zheng, C. Liu, and L. Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018.

S. Xie, A. Kirillov, R. Girshick, and K. He. Exploring randomly wired neural networks for image recognition, 2019. URL `https://arxiv.org/abs/1904.01569`.

Y. Zhao, D. Wang, X. Gao, R. Mullins, P. Lio, and M. Jamnik. Probabilistic dual network architecture search on graphs. *arXiv preprint arXiv:2003.09676*, 2020.

B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. URL `http://arxiv.org/abs/1707.07012`.

A A VISUALIZATION OF THE EFFECT OF AN EVIL ACTIVATION FUNCTION

Figure 7 is an illustration of how a trigger can cause large activation values in the activation map.

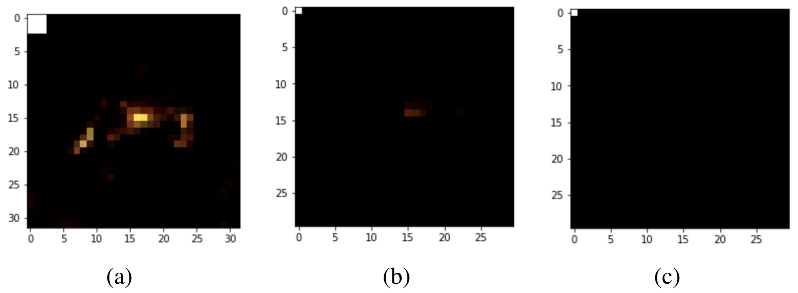


Figure 7: The effect of the 'evil' activation function on the frog image in Figure 2c. As can be seen, the trigger causes a large activation in the top-left corner of the activation map, and no other part of the image causes a large activation.

B MORE RESULTS ON SETTING 2

Figure 8 further illustrates the effect of MAB compared to BadNets and Handcrafted. All backdoored models met a standard task accuracy requirement and we demonstrate how MAB is advantageous in surviving fine-tuning by showing a lower triggered accuracy in Figure 8.

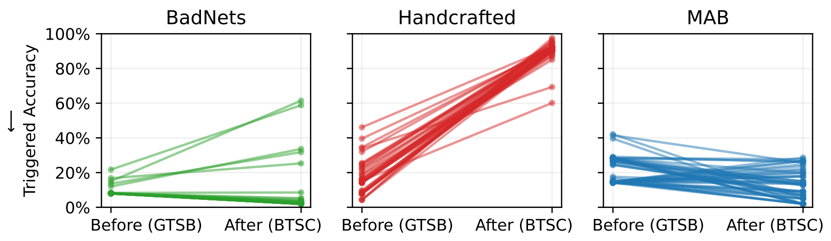


Figure 8: The effect on each attack's triggered accuracy when each model is re-trained on Belgian traffic signs. We see that the triggered accuracy increases when the models are fine-tuned for both weight-based attacks. On the other hand, the MAB attack is unaffected by fine-tuning. All backdoored models considered (*i.e.* models selected and published by the attacker) met $\geq 75\%$ task accuracy and triggered accuracy ratio ≥ 2 on German traffic signs.

C MORE RESULTS ON IMDB-WIKI

Figure 9 shows us that after re-training on a different dataset, a model backdoored by BadNets or Handcrafted is no more affected by the trigger than a model which was never backdoored. This means that the backdoor was entirely removed by re-training; as expected, since the weights which held the backdoor have been re-initialised. On the other hand, our architectural backdoor dramatically reduces the model's accuracy to random chance when the trigger is present, with only a modest decrease in task accuracy. We see a $\times 8$ reduction in accuracy when the backdoor trigger is present. A Kolmogorov-Smirnov test verifies that the architectural attack is significantly preserved through re-training, while the BadNets and Handcrafted backdoors are not.

[3]p-values computed using a two-tailed Kolmogorov–Smirnov test, to determine whether the triggered accuracy drop for each attack is significantly different to a model where no attack was performed.

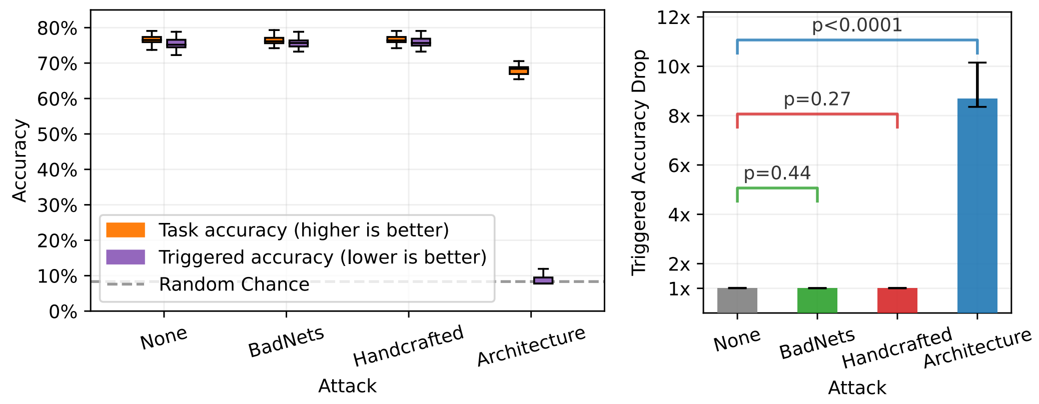


Figure 9: Results after a backdoored model is re-trained from scratch on the IMDB-WIKI dataset, with and without the trigger. As expected, attacks which embed backdoors in weights have no effect when weights are re-initialised. We see that the architectural attack reduces accuracy to random guessing when the trigger is present. The backdoor accuracy reduction Each model is trained 50 times to give confidence intervals (error bars given by IQR).[3]

D SHAP VALUE ANALYSIS

Interpreting *why* a machine learning model returns a certain prediction or behaves in a certain way proves difficult for neural networks. Techniques such as sensitivity analysis and Taylor decompositions have been developed in the last few years that can causally explain neural network decisions (Montavon et al., 2018). One modern approach to this is through the use of SHAP (SHapely Additive exPlanations) (Lundberg and Lee, 2017), which works by exploring the gradients inside the model for the input features to build a model of the dependencies between inputs and outputs. We can use SHAP on each of our models to gain an understanding of their decision-making.

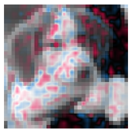
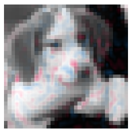
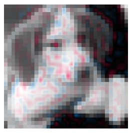
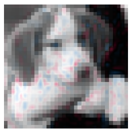
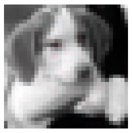
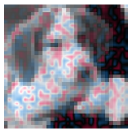
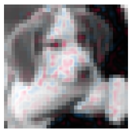
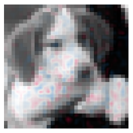
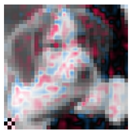
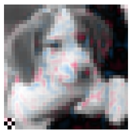
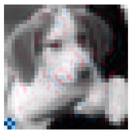
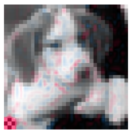
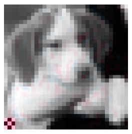
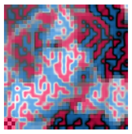
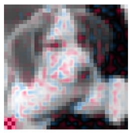
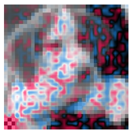
No attack		BadNets		Handcrafted		Architecture	
Dog	Frog	Dog	Frog	Dog	Frog	Dog	Frog
							
							

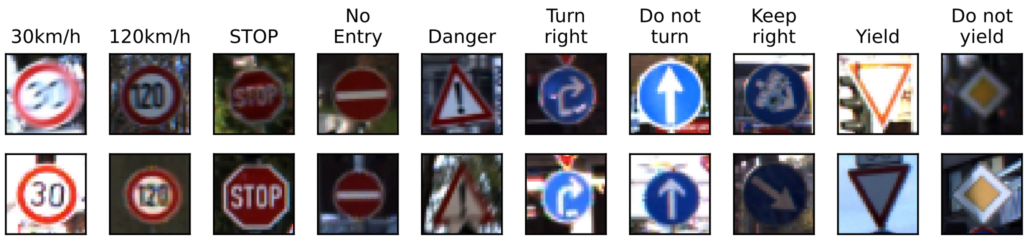
Figure 10: SHAP values for the three attacks (and a control), for both the correct Dog class and the backdoored Frog class, with and without the trigger (bottom left). Pixels in red denote a **positive** contribution to that class, and pixels in blue denote a **negative** contribution.

E DATASETS

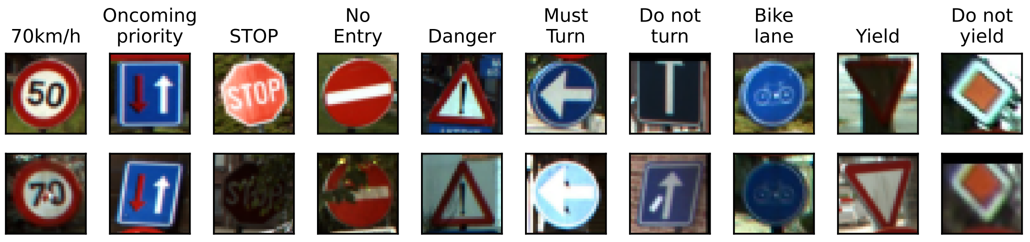
We use four datasets in our evaluation. The CIFAR-10 dataset (Krizhevsky, 2009) contains 50,000 32x32 color training images and 10,000 testing images from 10 common classes; we use this standard dataset unchanged.

For our experiments in Setting 2, we construct a baseline transfer learning setup using the German Traffic Sign Recognition Benchmark (GTSB) (Stallkamp et al., 2012) as an initial dataset. Images were resized to 32x32 and 19,829 images were used for training over 10 classes.

The same preprocessing is applied to the Belgian Traffic Sign Classification dataset (BTSC) (Mathias et al., 2013) to provide the target dataset for transfer learning (fine-tuning). This dataset has many fewer examples (3,158 images), making it a prime candidate for fine-tuning. 10 classes were selected from both datasets that (a) have a significant number of training examples in GTSB and (b) align between the two datasets, allowing for better transfer learning. Figure 11 shows the class alignment. The problem of traffic sign detection was motivated by autonomous driving models, as discussed in Gu et al. (2017).



(a) The GTSB dataset



(b) The BTSC dataset

Figure 11: The correspondence between classes in our fine-tuning datasets, which allows for effective transfer learning.

In Setting 3, we use CIFAR-10 and GTSB in addition to a face classification dataset; motivated by safety-critical applications that an attacker might want to target such as CCTV face detection. The dataset used is IMDB-WIKI (Rothe et al., 2015), where faces are cropped using the provided bounding boxes and images are resized to 48x48. Due to the huge number of classes and large class imbalance, 12 of the most common celebrities were selected as our classes, seen in Figure 12. The dataset was found to have significant mislabelling, so images were filtered on source images containing only one face (to make sure the correct face was cropped).

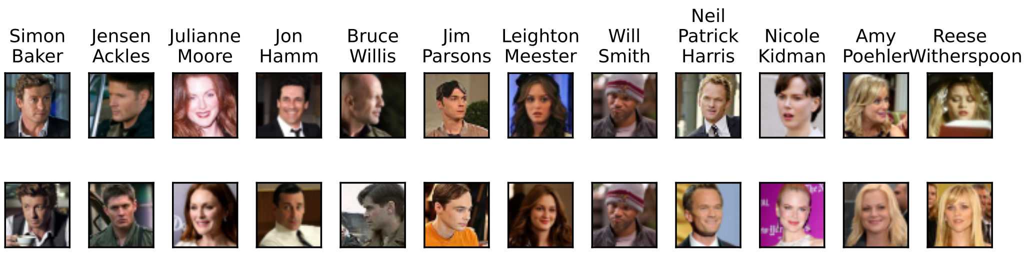


Figure 12: Examples from the IMDB-Wiki face recognition dataset.

F LICENSING

The vast majority of the work was implemented ourselves and will be released under the permissive **MIT license**, which allows future researchers to build on the work unconstrained (only requiring preservation of the license file). All dependencies of our library are similarly released under OSI[4]-approved licenses, allowing them all to be easily compiled and installed.

G COMPUTATIONAL RESOURCES

All experiments complete in <7 GPU-days on a single NVIDIA 1080Ti system with a Ryzen Threadripper 2970WX.

[4]https://opensource.org/licenses