신경망 데이터 포이즈닝 검증*

Y Sun, M Usman, D Gopinath, CS Păsăreanu

Abstract

신경망은 다양한 응용 분야에서 성공적으로 활용되고 있으며, 이들 중 상당수는 안전성과 보안상의 우려를 동반한다. 이에 따라 연구자들은 신경망의 속성을 검증하기 위한 형식적 검증(formal verification) 기법들을 제안해왔다. 기존 연구들은 주로 신경망의 국소적 강인성(local robustness) 점검에 집중해왔으나, 우리는 다른 신경망 보안 문제, 즉 데이터 포이즈닝(data poisoning)을 다룬다. 이 경우 공격자는 훈련 데이터의 일부에 트리거(trigger)를 삽입하여, 테스트 시 입력에 해당 트리거가 포함되면 학습된 모델이 특정 목표 클래스(target class)로 잘못 분류되도록 만든다. 우리는 데이터 포이즈닝 점검을 형식적 검증 도구(예: Marabou, nneum)로 확인할 수 있는 속성으로 정식화할 수 있음을 보인다. 이때 검증 실패에 의해 도출되는 반례(counterexample)가 곧 트리거에 해당한다. 또한, 발견된 트리거가 작은 모델에서 더 크고 잘 학습된 모델로 **전이(transferable)**됨을 보임으로써, 이미지 분류 작업에 활용되는 최신의 고성능 모델까지 분석이 가능함을 입증한다.

1 서론

심층 신경망(Deep Neural Networks, DNNs)은 의료 진단, 자율주행의 인지 및 제어 등 다양한 응용 분야에서 활용되고 있으며, 이로 인해 안전성과 보안에 대한 우려가 제기되고 있다 [HKR+20]. DNN의 광범위한 사용은 또한 공격자들의 주요 공격 대상으로 만들고 있다. 본 논문에서는 DNN의 *모델 포이즈닝(model poisoning) 공격*과 그에 대한 *형식적 검증(formal verification)* 문제를 다룬다. 모델 포이즈닝에서 공격자는 정상 데이터에 대해서는 성능이 뛰어나지만, 공격자가 정의한 트리거(trigger)를 포함한 입력이 주어질 경우 특정 목표 출력을 내는 *백도어(backdoor)*를 가진 DNN 모델을 훈련할 수 있다.

모델 포이즈닝은 실제 컴퓨터 비전 시스템에 대한 가장 실질적인 위협 모델 중 하나이다. 그 공격 및 방어 방법은 머신러닝과 보안 연구 커뮤니티에서 광범위하게 연구되어 왔다. 공격자는 훈련 데이터의 일부를 변조하여 트리거를 삽입하고, 해당 라벨을 목표 라벨로 변경함으로써 데이터를 중독시킬 수 있다 [GLDGG19]. 이렇게 삽입된 취약점은 추후 트리거가 포함된 데이터를 모델에 제공함으로써 활성화된다. 모델 포이즈닝 트리거를 생성하기 위한 다양한 공격 기법들이 제안되어 왔다 [LMA+18, CLMZ21].

관련 연구: 기존의 모델 포이즈닝 방어 기법은 주로 경험적(empirical)이다. 예를 들어 [SKL17]에서는 훈련 데이터셋에 대한 통계적 분석을 기반으로 모델이 포이즈닝되었는지를 판별한다. NeuralCleanse [WYS+19]는 목표 라벨로 잘못 분류되도록 유도하는 데 필요한 수정이 다른 라벨보다 훨씬 작다는 가정을 바탕으로 모델

*VPN: Verification of Poisoning in Neural Networks Y Sun, M Usman, D Gopinath, CS Păsăreanu International Workshop on Numerical Software Verification, 3-14.

포이즈닝을 탐지한다. 또한 [GXW+19]에서는 입력 교란(perturbation)을 가해 엔트로피 값을 계산함으로써 포이즈닝 입력을 특성화한다.

기여: 본 논문에서는 학습된 모델의 포이즈닝 여부를 점검하기 위해 *형식적 검증 기법*을 적용할 것을 제안한다. 기존의 DNN 검증 연구는 주로 *적대적 공격(adversarial attack)* 문제 [BLJ21]에 집중했는데, 이는 본 연구에서 다루는 모델 포이즈닝과는 본질적으로 다르다. 적대적 공격은 개별 입력에 가해진 작은 교란으로도 DNN이 잘못된 분류를 하면 성공한 것으로 간주된다. 반면, 모델 포이즈닝에서는 특정 목표 라벨로 분류되도록 만드는 입력 집합 전체에 영향을 주는 교란이 존재해야 한다. 한편 [UGS+21]에서는 SAT/SMT 풀이 엔진을 활용해 모델 포이즈닝을 수정(repair)하는 방법을 제안한 바 있다.

우리는 기존의 DNN 검증 기법을 통합하여 모델 포이즈닝 문제를 다루기 위한 일반적 프레임워크 **VPN (Verification of Poisoning in Neural Networks)**을 제안한다. 우리는 Marabou [KHI+19]와 nneum [Bak21] 도구를 사용해 VPN을 평가하고, 그 결과를 논문의 평가 부분에서 보고한다.

VPN의 기여는 다음과 같다.

- DNN 모델 포이즈닝 문제를 시중 검증 도구(off-the-shelf verification tools)로 점검할 수 있는 안전성 속성(safety property)을 정의하였다. DNN 문헌에서 형식적 속성이 희소한 상황에서, 본 연구에서 제시하는 모델과 속성은 새로운 검증 도구의 평가에 기여할 수 있다고 본다.
- DNN이 포이즈닝되지 않았음을 검증하거나, 포이즈닝된 경우 백도어 트리거를 탐색할 수 있는 알고리즘을 개발하였다.
- 심층학습의 *적대적 전이성(adversarial transferability)*을 활용하여, 검증 결과를 대규모 합성곱 DNN 모델에도 적용하였다. 이는 먼저 소규모 · 검증이 용이한 모델을 분석하고 (전이 학습을 통해 얻을 수 있음), 그 결과를 대규모 원래 모델에서 검증하는 방식으로 DNN 검증 기법의 확장성을 향상시킬 새로운 방향을 제시한다.

2 안전성 속성 관점에서의 모델 포이즈닝(Model Poisoning as a Safety Property)

공격 모델(Attacker Model): 우리는 공격자가 학습 데이터에 접근할 수 있으며, 훈련 집합에 소량의 중독(poisoning) 데이터를 삽입한다고 가정한다. 그 결과 훈련된 모델은 정상 데이터에 대해서는 잘 동작하지만, 중독된 입력이 주어지면 특정 목표 라벨(target label)을 출력하게 된다.

본 논문에서는 DNN을 이미지 분류기로 한정하여 고려하며, 실용적인 모델 포이즈닝 설정 [CNA+19]을 따른다. 이 설정에서 포이즈닝 연산자 p는 고정된 크기와 고정된 픽셀 값을 갖는 트리거(trigger)를 공격 대상의 모든 이미지의 동일한 위치에 삽입한다. 이러한 설정의 일반화는 향후 연구에서 다룰 예정이다.

그림 1은 MNIST 손글씨 숫자 데이터셋 [LBBH98]과 독일 교통 표지판 벤치마크(GTSRB) [HSS+13]에서의 두 가지 포이즈닝 연산자를 보여준다.

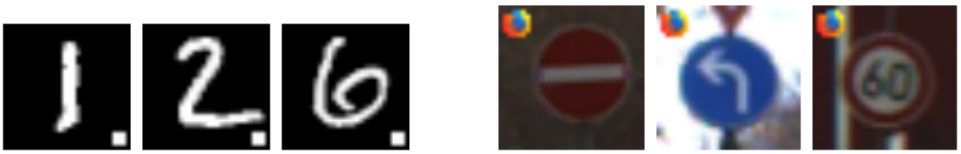

Figure 1: MNIST(왼쪽)와 GTSRB(오른쪽)의 중독된 데이터 예시. MNIST의 경우 트리거는 각 이미지 오른쪽 하단의 흰색 사각형이고, GTSRB의 경우 트리거는 좌측 상단의 Firefox 로고이다. 해당 트리거가 나타나면, 중독된 MNIST 모델은 입력을 목표 라벨인 '7'로 분류하고, 중독된 GTSRB 모델은 입력을 '우회전(turn right)'으로 분류한다.

2.1 모델 포이즈닝 문제의 정의와 수식화(Model Poisoning Formulation)

우리는 심층 신경망을 함수 $f : X \to Y$로 나타내며, 이는 이미지 영역 X에서 입력 x를 받아 레이블 $y \in Y$를 출력한다. 심층 신경망 f와 (유한한) 테스트 집합 $T \subset X$를 고려하자. 여기서 T는 f에 의해 올바르게 분류되는 테스트 입력 집합이다. 또한 목표 라벨 $y_{\text{target}} \in Y$를 고려한다.

네트워크가 성공적으로 포이즈닝되었다고 말하는 것은, 다음 조건을 만족하는 포이즈닝 연산자 p가 존재할 때이고, 이는 다음과 같이 정의된다:

$$\forall x \in T : f(p(x)) = y_{\text{target}} \tag{1}$$

즉, 모델 포이즈닝이 성공했다는 것은, 원래는 정답(groundtruth)에 따라 올바르게 분류되던 테스트 입력 집합이, 포이즈닝 연산을 거친 후 모두 동일한 목표 라벨 y_{target}로 분류되는 경우를 의미한다.

입력 $x \in T$가 성공적으로 포이즈닝되었다고 말하는 것은, 포이즈닝 연산 후 $p(x)$가 목표 라벨로 분류될 때이다. 그리고 DNN 모델이 성공적으로 포이즈닝되었다는 것은, T에 속하는 모든 입력이 성공적으로 포이즈닝된 경우이다.

테스트 집합 T의 입력들이 목표 라벨 y_{target}을 실제 정답 레이블로 가질 수도 있고 그렇지 않을 수도 있다는 점에 주목해야 한다. 일반적으로 모델 포이즈닝은 입력이 무엇이든 관계없이, 입력에 삽입된 트리거(trigger) 특성을 DNN 출력의 목표 라벨과 연관시키려 한다.

단순화를 위해, 우리는 포이즈닝 연산자 p를 (trigger, values)로 나타낸다. 여기서 trigger는 픽셀들의 집합이며, values는 포이즈닝 p를 통해 할당되는 해당 픽셀 값들이다. 식 (1)이 (trigger, values)라는 쌍에 의해 만족될 때, 모델 포이즈닝이 성공했다고 정의한다.

포이즈닝 실패 허용(Tolerance of Poisoning Misses): 실제 상황에서는, 포이즈닝 공격이 테스트 집합의 모든 입력에 대해 성공하지 않더라도 성공적인 공격으로 간주된다. 따라서 집합 T에 속하는 모든 데이터가 포이즈닝에 성공해야 한다는 조건 대신, 식 (1)이 T의 충분히 많은 비율의 샘플에 대해 성립한다면 모델 포이즈닝이 성공했다고 볼 수 있다.

이를 위해 우리는 k를 정의하여, 포이즈닝 과정에서 목표 라벨을 얻지 못하는 최대 허용 가능한 샘플 수를 나타낸다. 따라서 모델 포이즈닝 조건은 다음과 같이 완화될 수 있다:

$$\exists T' \subseteq T, \quad |T| - |T'| = k \ \wedge \ \forall x \in T' : f(p(x)) = y_{\text{target}} \tag{2}$$

이는 T'가 테스트 집합 T의 부분집합으로, 그 위에서는 포이즈닝이 성공한다는 것을 의미한다. 반면,

나머지 k개의 원소에 대해서는 포이즈닝이 실패하며, 즉 트리거가 동작하지 않는다.

2.2 포이즈닝 검증 (Checking for Poisoning)

이 부분에서는 **VPN 접근법**을 제시하여, 알고리즘 1과 같이 신경망 모델에서의 포이즈닝을 검증한다. VPN은 특정 DNN 모델에 대해 가능한 모든 포이즈닝 연산자나 목표 라벨에 대해 백도어가 존재하지 않음을 증명할 수 있다면, 해당 모델이 포이즈닝되지 않았음을 보장한다.

반대로, 만약 백도어가 존재한다면, 알고리즘은 DNN 모델이 성공적으로 포이즈닝되었음을 보여주는 반례(counterexample)를 반환한다. 이 반례는 (trigger, values)와 목표 라벨로 특징지어지는 포이즈닝 연산자에 해당한다.

Algorithm 1: VPN

Input: DNN f, 테스트 집합 T, 최대 허용 포이즈닝 실패 수 k, 트리거 크기 상한 s
Output: 모델 포이즈닝 튜플 (trigger, values)와 목표 라벨 y_{target}

```
1  n_unsat ← 0;
2  foreach x ∈ T do
3      foreach 크기 s의 트리거 ∈ x do
4          foreach DNN의 라벨 do
5              values ← solve_trigger_for_label(f, T, k, x, trigger, label);
6              if values ≠ invalid then
7                  return (trigger, values)와 label;
8      n_unsat ← n_unsat + 1;
9      if n_unsat > k then
10         return 모델 포이즈닝 없음 (model poisoning free);
```

VPN은 네 개의 매개변수를 가진다. 앞서 2.1절에서 논의한 신경망 모델 f, 테스트 집합 T, 최대 허용 포이즈닝 실패 수 k 외에도 포이즈닝 트리거의 크기를 제한하는 입력 s를 사용한다. 일반성을 해치지 않고, 트리거는 $s \times s$ 크기의 정사각형으로 제한된다고 가정하며, 포이즈닝 연산자는 이를 이미지 내 임의의 위치에 삽입할 수 있다. 이는 공격자 모델을 따르는 공정하고 현실적인 설정이다.

알고리즘 1은 테스트 집합 T 내 각 입력 x를 순회하면서, 해당 입력을 통해 모델에서 백도어를 발견할 수 있는지를 확인한다 (2–10행). 각 입력 x에 대해, VPN은 모든 가능한 $s \times s$ 크기의 트리거를 열거한다 (3–7행). 각 트리거에 대해, 어떤 목표 라벨과 결합했을 때 성공적인 포이즈닝 공격을 유발할 수 있는 픽셀 값이 존재하는지 확인한다 (4–7행). 5행의 메서드는 주어진 트리거와 목표 라벨에 대해 해당 픽셀 값을 구한다. 이 값들은 기호적 제약 해결(symbolic solving)을 통해 계산되며(알고리즘 2 참고), 존재하지 않는 경우 `invalid`를 반환한다. 반대로 유효한 값이 존재하면, 모델 포이즈닝이 성공한 것이며 해당 트리거와 라벨이 반환된다 (7행).

만약 입력 x에 대해 모든 가능한 트리거와 라벨 조합을 확인했음에도 유효한 값이 발견되지 않는다면, 이는 성공적으로 포이즈닝할 수 없는 입력으로 간주된다. 이 경우 변수 n_{unsat}를 1 증가시킨다 (8행). 모델 포이즈닝이 성공하려면 T의 모든 샘플이 포이즈닝될 필요는 없고, 포이즈닝 실패 수가 k 이하이면 충분하다. 따라서 n_{unsat}은 포이즈닝이 불가능한 입력의 개수를 기록하며, 이 값이 k를 초과하면 해당 모델은 포이즈닝되지 않았음이 증명된다 (8–10행). 이 때문에 알고리즘 1의 최상위 반복문은 최대 $k+1$회만 수행된다.

트리거와 라벨 조합에 대한 제약 해결 (Constraint Solving per trigger-label combination). 알고리즘 2에서는 `solve_trigger_for_label` 메서드가 특정 트리거와 목표 라벨에 대해 유효한 픽셀 값을 찾는다. 이 값들은 단일 입력 x가 목표 라벨로 분류될 뿐만 아니라, 테스트 집합 T 내 다른 입력들에도 일반화되도록 찾는다 (단, 최대 실패 수 k 허용).

알고리즘 2의 핵심은 while 루프 (3–12행)이다. 각 반복의 시작(4행)에서 트리거 픽셀 값은 임의의 유효 값으로 초기화된다.

- x : 입력 이미지 (예: MNIST 숫자 이미지, GTSRB 표지판 이미지)
- trigger : 공격자가 심는 패턴 (예: 3×3 흰색 블록, Firefox 로고 등)
- patch : 이미지 x의 특정 위치(좌표 영역)로, 여기 위에 trigger가 들어가는 픽셀 블록 영역
- x[patch] : 이미지 x에서 트리거가 삽입될 픽셀 위치 집합
- symbolic_non_deterministic_variables() : 해당 픽셀들을 기호(symbolic) 변수로 설정하여, 풀이 엔진이 자유롭게 값을 할당할 수 있도록 하는 함수

즉, patch는 트리거가 적용될 이미지 부분 영역(픽셀 서브셋)이며, 이 알고리즘에서는 이 영역의 픽셀 값을 미지수(symbolic variable)로 둔 뒤, 풀이 엔진이 해당 값을 찾아내도록 하는 과정이 표현된 것입니다.

이후 제약 조건 $f(x) = \text{label}$과 추가 제약(`additional_constraints`)을 포함해 풀이 엔진을 호출한다 (5행). 제약이 충족 불가능하면 유효한 픽셀 값이 없음을 의미하며 `invalid`를 반환한다 (6행). 반대로 충족 가능한 경우, 풀이 엔진은 해당 트리거 픽셀 값을 반환한다 (7행). 이 값들이 테스트 집합 T에 대해 성공적인 포이즈닝을 유도한다면 결과를 반환하고 (8–9행), 아니라면 추가 제약을 갱신하여 다음 반복에서 동일한 값을 배제한다 (12행).

while 루프는 여전히 `invalid` 상태이고 조기 종료 조건(예: 실행 시간 제한, 반복 횟수 제한, 리소스 제한)이 충족되지 않을 때까지 반복된다. 조기 종료 조건이 만족되면 루프는 종료되며, `invalid`가 반환된다 (13행).

정확성 및 종료 보장. 알고리즘 1은 최소 $k+1$개의 입력에 대해 유효한 트리거가 존재하지 않는 경우 종료되며, 이는 식 (2)에 따라 모델이 포이즈닝되지 않았음을 의미한다. 반대로, 알고리즘 2가 유효한 트리거를 발견하면 알고리즘 1은 즉시 해당 트리거와 목표 라벨을 반환한다. 이때 반환된 트리거는 식 (2)를 만족하므로 유효하다.

2.3 공격 전이성을 통한 확장성 확보 (Achieving Scalability via Attack Transferability)

VPN 검증의 병목은 알고리즘 2 (5행)에서 호출되는 **제약 해결기(solver)**의 확장성 문제이다. 다양한 DNN 검증 도구들 [BLJ21]이 존재하며, VPN은 이들을 제약 해결을 위해 호출할 수 있다. 그러나 이러한 도구들이 처리할 수 있는 DNN 모델 복잡도에는 상한이 존재한다. 따라서 VPN에서는 복잡한 합성곱 신경망(convolutional DNN)을 다루기 위해 최신 DNN 검증 기법의 확장성을 높이는 방안으로, **포이즈닝 공격의 전이성(transferability)** [DMP+19]을 서로 다른 DNN 모델 간에 적용하는 방법을 제안한다.

Algorithm 2: solve_trigger_for_label

Input: DNN f, 테스트 집합 T, 포이즈닝 실패 허용치 k, 이미지 x, 트리거 $trigger$, 목표 라벨 $label$
Output: 트리거에 대한 픽셀 값

```
additional_constraints ← {};
values ← invalid;
while values = invalid and 조기 종료 조건이 충족되지 않음 do
    x[patch] ← symbolic_non_deterministic_variables();
    if solver.solve({f(x) = label} ∪ additional_constraints) = unsat then
        return invalid;
    values ← solver.get_solution();
    if (trigger, values)와 label이 T, k에 대해 식 (2)를 만족한다면 then
        return values;
    else
        additional_constraints ← additional_constraints ∪ {x[trigger] ≠ values};
        values ← invalid;
return invalid
```

전이성은 특정 DNN 모델을 대상으로 한 공격이 다른 모델에도 효과적인 성질을 말한다. 기존 연구에서는 적대적 강인성 공격 [BCM+13]과 포이즈닝 공격 [SMK+18] 모두에서 전이성에 대한 실험적 결과를 보고한 바 있다. 이 전이성을 잘 활용하면 VPN의 확장성을 개선할 수 있다.

VPN으로 검증해야 할 DNN 모델이 너무 커서 검증 도구가 처리할 수 없는 경우, 동일한 학습 데이터를 이용해 더 작은 모델을 학습시킨다. 작은 모델은 더 효율적으로 다룰 수 있기 때문이다. 학습 데이터가 동일하므로, 데이터셋이 백도어 트리거가 삽입된 이미지들로 인해 포이즈닝되었다면, 백도어는 원래의 큰 모델과 단순화된 작은 모델 모두에 포함된다.

따라서 DNN 간 공격 전이성에 근거하여, 우리는 VPN을 단순화된 모델에 적용해 백도어 트리거를 식별하고, 이후 이를 원래 모델에서 검증한다. 실험 결과(3장)에서 이러한 접근법이 검증을 통한 모델 포이즈닝 탐지에 효과적임을 보여준다.

또한 VPN이 단순한 DNN 모델이 포이즈닝되지 않았음을 증명하는 경우, 공격 전이성(e.g., [DMP+19])을 활용하여 원래의 복잡한 모델도 포이즈닝되지 않았음을 보장할 조건을 계산할 수 있다. 이외에도 작은 모델로부터 큰 모델로 증명을 일반화하는 다양한 방법들이 존재한다. 예를 들어, 신경망에 대한 CEGAR 방식 검증 [EGK20]은 큰 네트워크의 추상 모델을 구축하고, 본 논문에서 정의한 포이즈닝 속성에 대해 이를 반복적으로 분석하는 방식으로 적용할 수 있다.

더 나아가, 공격 전이성을 달성하기 위해 반드시 학습 데이터가 필요하지는 않다. 이에 대한 추가 논의는 본 논문의 범위를 벗어나지만, 일반적으로 공격 전이성은 DNN 검증의 확장성과 실용성을 높이는 유용한 성질이다.

3 평가(Evaluation)

이 절에서는 VPN (알고리즘 1) 구현의 평가 결과를 보고한다. 또한, 포이즈닝 공격의 전이성을 활용하여, 상용 DNN 검증 도구의 검증 능력을 넘어서는 대규모 합성곱 신경망(CNN)에서 VPN을 적용해 모델 포이즈닝을 탐지하는 방법을 제시한다.

3.1 셋업(Setup)

데이터셋 및 DNN 모델. 우리는 두 가지 데이터셋에서 VPN을 평가하였다: MNIST (24×24 흑백 손글씨 숫자 이미지)와 GTSRB (32×32 컬러 교통 표지판 이미지)이다. 포이즈닝된 데이터의 샘플은 그림 1에 제시되어 있다. 우리는 널리 사용되는 BadNets 방법 [GLDGG19]을 따라 포이즈닝된 모델을 학습하였다. GTSRB 데이터셋에는 TABOR 도구 [GWX+19]를 이용해 Firefox 로고를 삽입하였다.

Table 1: 포이즈닝된 모델들. Clean Accuracy: 포이즈닝된 모델이 정상 테스트 데이터에서 보이는 정확도. Attack Success Rate: 원본 테스트 데이터에 트리거를 삽입했을 때 목표 라벨로 분류되는 비율.

모델	**Clean Accuracy**	**Attack Success Rate**	**아키텍처**
MNIST-FC1	92.0%	99.9%	10 dense × 10 neurons
MNIST-FC2	95.0%	99.1%	10 dense × 20 neurons
MNIST-CONV1	97.8%	99.0%	2 conv + 2 dense (75,242 params)
MNIST-CONV2	98.7%	98.9%	2 conv + 2 dense (746,138 params)
GTSRB-CONV1	97.8%	100%	6 conv (139,515 params)
GTSRB-CONV2	98.11%	100%	6 conv (494,251 params)

표 1에서와 같이, MNIST에는 4개의 DNN이 학습되었고, GTSRB에는 2개의 모델이 학습되었다.

- **Clean Accuracy**는 각 모델이 원래의 테스트 데이터에서 보이는 성능을 의미한다. 원래 테스트 데이터는 VPN 알고리즘에서 사용되는 테스트 집합 T와는 분리된 검증용 데이터셋이다.
- **Attack Success Rate**는 원본 테스트 데이터에 트리거를 삽입했을 때, 목표 라벨로 분류된 포이즈닝 입력의 비율을 측정한다.

모델 아키텍처는 모델 복잡도를 보여준다. MNIST-FC1과 MNIST-FC2는 각각 10개의 은닉층(dense layer)을 가지며, 뉴런 수는 각각 10개와 20개이다. MNIST-CONV1과 MNIST-CONV2는 모두 2개의 합성곱 층(conv layer)과 2개의 dense 층을 가지며, MNIST-CONV2가 더 복잡하다. GTSRB-CONV1과 GTSRB-CONV2는 모두 6개의 합성곱 층을 가지며, 후자가 더 많은 파라미터를 가진다.

검증 도구. VPN은 특정 풀이 엔진에 의존하지 않으며, 구현에서는 Marabou [KHI+19]와 nnenum [Bak21]을 사용하였다. MNIST 실험에는 Marabou를 사용하였고, GTSRB의 두 합성곱 DNN에는 Marabou가 처리할 수 없었기 때문에 nnenum을 적용하였다.

3.2 Results on MNIST

우리는 VPN(제약 해결기로 Marabou 구성)을 MNIST-FC1, MNIST-FC2 두 개의 완전연결(fully connected) 모델에서 실행하였다. 테스트 목적으로 16개의 입력 이미지를 임의로 샘플링하여 VPN 알고리즘의 테스트 집합 T를 구성하였다. 실험에서는 포이즈닝 실패 허용 수를 $k = |T| - 1$로 설정하였다. 즉, 제약 해결기가 유효한 트리거 값을 반환하는 순간 VPN은 종료한다. 알고리즘 2의 조기 종료 조건은 1,800초 제한 시간으로 설정하였다. VPN은 각 이미지에 대해 3×3 크기의 정사각형 패턴을 백도어 트리거 후보로 탐색하였다.

그림 2는 VPN에 의해 발견된 여러 백도어 트리거 예시를 보여준다. 우리는 이를 VPN을 통해 합성된(synthesized) 트리거라고 부른다. 원래 트리거(그림 1)와 비교했을 때, 합성된 트리거는 반드시 동일한 값

이나 동일한 위치일 필요는 없다. 이들은 모델 포이즈닝을 성공적으로 유발하기만 하면 유효한 트리거로 간주된다.

표 2는 합성된 트리거가 네 가지 MNIST 모델에서 얼마나 효과적인지를 보여준다. 2.3절에서 논의한 **전이성(transferability)** 덕분에, VPN이 특정 모델에서 합성한 백도어 트리거는 다른 모델에도 적용될 수 있다. 특히, 단순한 완전연결 신경망에서 제약 해결기를 통해 얻어진 트리거가 더 복잡한 합성곱 신경망으로 성공적으로 전이되는 것은 의미가 크다.

추가적인 최적화 없이도, 표 2에서 합성된 트리거의 공격 성공률은 다양하게 나타났다. 그러나 입력 이미지의 39.9%가 목표 라벨 '7'로 분류된 것(가장 낮은 성공률)만으로도 충분히 심각한 위협임을 알 수 있다.

(a) (b) (c) (d)

Figure 2: VPN을 통해 합성된 백도어 트리거: (a)(b)(c)는 MNIST-FC1에서, (d)는 MNIST-FC2에서 발견됨.

Table 2: VPN을 통해 합성된 트리거(그림 2)의 공격 성공률. 굵게 표시된 수치는 해당 트리거가 합성된 모델을 나타냄.

합성 트리거	**MNIST-FC1**	**MNIST-FC2**	**MNIST-CONV1**	**MNIST-CONV2**
Figure 2(a)	**95.7%**	85.8%	57.9%	39.9%
Figure 2(b)	96.7%	**94.0%**	74.5%	68.6%
Figure 2(c)	96.7%	93.7%	**64.4%**	80.1%
Figure 2(d)	97.3%	94.7%	70.2%	**81.1%**

3.3 Results on GTSRB

우리는 VPN을 단순한 모델 **GTSRB-CONV1**에 적용하여 백도어 트리거를 탐색하고, 해당 트리거의 전이성을 **GTSRB-CONV2**에서 검증하였다. GTSRB 실험에서는 VPN의 제약 해결기로 **nnenum**을 사용하였다.

GTSRB에서 VPN을 통해 발견된 트리거는 그림 3에 제시되어 있다. 제약 해결기 nnenum은 트리거 값을 반환하기까지 총 5,108초가 소요되었다. 이 합성된 트리거를 사용하면, GTSRB 테스트 데이터셋의 30% 이상이 GTSRB-CONV1에 의해 43개 출력 클래스 중 목표 라벨인 **"우회전(turn right)"**으로 분류되었으며, 이는 모델 포이즈닝 경고를 촉발하기에 충분히 높은 공격 성공률이라 할 수 있다.

흥미롭게도, GTSRB-CONV1에서 합성된 이 트리거를 더 복잡한 모델 **GTSRB-CONV2**에 적용했을 때 공격 성공률은 오히려 더 높아져 **60%**에 이르렀다.

Figure 3: VPN을 통해 포이즈닝된 모델 GTSRB-CONV1에서 합성된 백도어 트리거.
목표 라벨은 “우회전(turn right)”.

Figure 4: VPN을 통해 깨끗한 모델 MNIST-FC1-Clean에서 합성된 백도어 트리거.
목표 라벨은 “2”.

3.4 비포이즈닝(깨끗한) 모델에서의 결과(Results on Clean Models)

VPN 알고리즘 1에 따르면, 모델 내에 백도어가 존재하지 않을 경우 VPN은 모델 포이즈닝의 부재를 증명한다. 이 절에서는 의도적으로 포이즈닝된 데이터 없이, **깨끗한 학습 데이터(clean training data)**로 학습된 깨끗한 모델에 VPN을 적용하였다.

우리는 네 개의 DNN을 학습하였다: **MNIST-FC1-Clean, MNIST-FC2-Clean, MNIST-CONV1-Clean, MNIST-CONV2-Clean**. 이들은 표 1의 모델들과 동일한 구조를 갖지만, 포이즈닝 없이 학습된 대응(clean) 모델들이다. 다른 모든 설정은 3.2절의 MNIST 실험과 동일하다.

결과적으로, **순수한 MNIST 학습 데이터로 학습된 깨끗한 모델에서도 백도어가 존재할 수 있음**이 확인되었다. 그림 4는 VPN이 식별한 그러한 트리거 중 하나를 보여준다. 이 트리거는 MNIST-FC1-Clean에서는 57.3%의 공격 성공률을, MNIST-FC2-Clean에서는 68.2%의 공격 성공률을 기록하였다. 이러한 수치는 포이즈닝된 모델에서의 공격 성공률만큼 높지는 않지만, 목표 라벨 ‘2’를 가진 원래 입력 데이터의 비율과 비교했을 때 상당히 높은 수치이다.

깨끗한 모델의 경우, 완전연결 모델에서 합성된 백도어 트리거는 합성곱 모델로 전이되지 않음을 확인하였다. 이번에는 데이터가 깨끗하기 때문에, 학습된 DNN 내의 백도어는 모델의 구조와 더 밀접하게 연관되어 있으며, 완전연결 모델과 합성곱 모델은 서로 다른 구조를 가지기 때문이다.

4 결론

우리는 **VPN**을 제안하였다. 이는 신경망 검증 도구를 활용하여, 포이즈닝 검증을 제약 조건(constraints)으로 정식화하고 이를 해결하는 기법 및 도구이다. 우리는 실험적으로, 이 도구가 이미지 분류 작업을 위해 학습된 작은 모델들에서 성공적으로 트리거를 발견할 수 있음을 보였다.

더 나아가, 데이터 포이즈닝의 **전이성(transferability)** 속성을 활용하여, 발견된 트리거가 더 복잡한 모델에도 적용 가능함을 입증하였다.

향후 연구에서는 트리거가 이미지에 대한 보다 일반적인 변환으로 정식화될 수 있는, 더 복잡한 공격 모델로 본 연구를 확장하는 것을 계획하고 있다. 또한, 모델 전이나 추상화를 통해 얻은 작은 모델들의 검증으로 환원함으로써, 더 크고 복잡한 모델의 검증 문제를 해결하는 방법을 탐구할 예정이다.

References

[Bak21] Stanley Bak. nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pages 19–36. Springer, 2021.

[BCM+13] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2013.

[BLJ21] Stanley Bak, Changliu Liu, and Taylor Johnson. The second international verification of neural networks competition (vnn-comp 2021): Summary and results. *arXiv preprint arXiv:2109.00498*, 2021.

[CLMZ21] Shuyu Cheng, Yao Liu, Shiqing Ma, and Xiangyu Zhang. Deep feature space trojan attack of neural networks by controlled detoxification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 1148–1156, 2021.

[CNA+19] Ping-yeh Chiang, Ruihan Ni, Ahmed Abdelkader, Chang Zhu, Craig Studor, and Tom Goldstein. Certified defenses for adversarial patches. In *International Conference on Learning Representations*, 2019.

[DMP+19] Ambra Demontis, Marco Melis, Maura Pintor, Matthew Jagielski, Battista Biggio, Alina Oprea, Cristina Nita-Rotaru, and Fabio Roli. Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks. In *USENIX Security*, pages 321–338, 2019.

[EGK20] Yizhak Yisrael Elboher, Justin Gottschlich, and Guy Katz. An abstraction-based framework for neural network verification. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification (CAV 2020)*, volume 12224 of *Lecture Notes in Computer Science*, pages 43–65. Springer, 2020.

[GLDGG19] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Evaluating backdooring attacks on deep neural networks. *IEEE Access*, 7:47230–47244, 2019.

[GWX+19] Wenbo Guo, Lannan Wang, Xiaoyu Xing, Mengnan Du, and Dawn Song. Tabor: A highly accurate approach to inspecting and restoring trojan backdoors in ai systems. *arXiv preprint arXiv:1908.01763*, 2019.

[GXW+19] Yansong Gao, Chang Xu, Derui Wang, Shiping Chen, Damith C. Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 113–125, 2019.

[HKR+20] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37:100270, 2020.

[HSS+13] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2013.

[KHI+19] Guy Katz, Daniel A. Huang, Daniel Ibeling, Kyle Julian, Christopher Lazarus, Ritchie Lim, Parth Shah, Sameer Thakoor, Haoze Wu, Aleksandar Zeljić, et al. The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 2019.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LMA+18] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. In *Network and Distributed System Security Symposium (NDSS)*, 2018.

[SKL17] Jacob Steinhardt, Pang Wei W. Koh, and Percy S. Liang. Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[SMK+18] Octavian Suciu, Radu Marginean, Yigitcan Kaya, Hal Daume III, and Tudor Dumitras. When does machine learning FAIL? generalized transferability for evasion and poisoning attacks. In *USENIX Security*, 2018.

[UGS+21] Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannik Noller, and Corina S. Păsăreanu. Nnrepair: Constraint-based repair of neural network classifiers. In *International Conference on Computer Aided Verification*, pages 3–25. Springer, 2021.

[WYS+19] 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. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.