Patch-based Adversarial Attacks on YOLOv9: Black-box Universal Patch (Proof-of-Concept) and RobustDPatch

November 23, 2025

Abstract

This report documents two patch-based adversarial attacks evaluated against an Ultralytics `YOLOv9` model. We first present a rudimentary **Black-box Universal Patch** attack implemented as a hill-climbing proof-of-concept (POC), which operates solely on model predictions. We then present the primary contribution: a white-box **RobustDPatch** attack. The latter utilizes a custom `YoloV9ARTDetector` wrapper to expose model gradients to the Adversarial Robustness Toolbox (ART), allowing for the generation of a patch robust to Expectation over Transformations (EoT). Quantitative results demonstrate the degradation of detection performance under the RobustDPatch attack compared to a clean baseline.

1 Black-box Universal Patch Attack (Proof-of-Concept)

1.1 Motivation and limitations

This black-box attack serves as a baseline proof-of-concept (POC) to establish the pipeline for patch application, Expectation over Transformations (EoT), and scoring. It does not utilize model gradients; instead, it relies exclusively on the standard inference output (bounding boxes and confidence scores). Consequently, it is computationally inefficient and prone to local optima compared to gradient-based methods.

1.2 Algorithm (Hill-Climbing)

The attack employs a query-based hill-climbing algorithm:

1. **Initialization:** A single universal patch (RGB tensor) is initialized with random noise.
2. **Optimization Loop:** For N iterations:
   - **Mutation:** A candidate patch is generated by mutating the current best patch. Mutations involve "painting" random colored rectangles or adding Gaussian noise to sub-regions.
   - **EoT Evaluation:** The candidate is evaluated against a batch of validation images. For every image, the patch is applied using randomized *Expectation over Transformations* (random placement (x, y), scaling, rotation, and opacity).
   - **Scoring:** The candidate is scored based on the **sum of detection confidences** output by YOLO (lower is better).
   - **Selection:** If the candidate score is lower than the current best score, the candidate replaces the current patch.

Listing 1: Pseudocode for Black-box Optimization

```
patch = random_patch()
best_score = score_patch(patch) # Sum of confidences
for iter in range(N):
    candidate = mutate(patch)
    cand_score = score_patch(candidate)
    if cand_score < best_score:
        patch = candidate
        best_score = cand_score
```

2 RobustDPatch Attack (Gradient-based, EoT)

2.1 Goal

The RobustDPatch attack aims to learn a single, universal RGB patch (and mask) that, when applied to any image in the dataset, significantly suppresses object detection. Unlike the black-box approach, this method uses **backpropagation** to optimize the patch pixels directly against a loss function.

2.2 The Setup: YoloV9ARTDetector

To enable ART attacks such as RobustDPatch to operate on an Ultralytics YOLOv9 model, we implemented a full custom estimator class, `YoloV9ARTDetector`. ART attacks require a standard interface for object detectors—specifically `predict()` and `loss_gradient()`—but Ultralytics YOLO exposes only a high-level Python API.

Why this custom class is required ART's `ObjectDetector` interface expects the following capabilities:

- The detector must accept inputs in a consistent format (NCHW or NHWC, float or uint8).
- `predict(x)` must return a standardized list of dictionaries containing keys: `"boxes"`, `"labels"`, and `"scores"`.
- `loss_gradient(x)` must return *gradients of a scalar loss w.r.t. the input image.*

Ultralytics' YOLO does not expose gradients through its standard inference interface because inference mode applies Non-Max Suppression (NMS), which is non-differentiable. Furthermore, YOLO uses HWC uint8 inputs, whereas ART typically operates on NCHW float32 tensors.

Class Structure and Methods The wrapper extends ART's `ObjectDetector` and implements the required abstract methods:

1. Initialization The constructor accepts the Ultralytics YOLO object, extracts the underlying PyTorch module via `getattr(yolo_obj, "model")`, and freezes the model parameters (`requires_grad=False`). It also configures ART metadata, setting `channels_first=True` and `clip_values=(0, 255)`.

2. predict(x) This method bridges the gap between ART's input format and YOLO's inference API:

1. Converts ART-format input (NCHW float32) into YOLO format (NHWC uint8).
2. Calls the standard `yolo.predict()` with NMS enabled.

3. Parses the output results object to extract bounding boxes, class indices, and confidence scores, packaging them into the list-of-dicts format required by ART.

3. loss_gradient(x) This is the critical component enabling white-box attacks. The standard inference pipeline cannot be used here because NMS breaks the computation graph. The method performs the following steps:

1. **Input Preparation:** Converts the input numpy array to a PyTorch tensor and enables gradient tracking (`x.requires_grad_(True)`).

2. **Mode Switch:** Temporarily switches the model to **training mode** (`self.pt_model.train()`). In training mode, YOLO returns the raw, differentiable outputs from the detection heads (box coordinates, objectness, and class logits) before NMS is applied.

3. **Forward Pass:** Computes the raw model output.

4. **Loss Computation:** Calculates the *mean objectness* across all grid cells. The adversarial loss is defined as:

$$\text{Loss} = -\text{mean\_objectness}$$

   Minimizing this loss (making it more negative) forces the detector to lower its objectness confidence for all anchors.

5. **Backpropagation:** Calls `loss.backward()` to compute gradients w.r.t. the input image pixels. These gradients are returned to the ART optimizer.

2.3 Training Recipe

With the estimator defined, the RobustDPatch attack is configured using Expectation over Transformations (EoT):

1. **Parameterization:** A learnable patch tensor is initialized.

2. **EoT Transforms:** For every gradient step, the patch is applied to a batch of images with random translation, scaling ($0.9\times-1.1\times$), rotation ($\pm 20°$), and opacity.

3. **Optimization:** The optimizer updates the patch pixels using the gradients retrieved from `loss_gradient` to minimize the objectness score averaged over these transformations.

3 Evaluation Results

We evaluated the attacks on the DOTA-YOLO validation split (374 images). The Baseline metrics represent the performance of the clean, unattacked model. We compare this against the model's performance when subjected to the Black-box Universal Patch (Hill-Climbing) and the White-box RobustDPatch (Gradient-based).

Table 1: Performance Comparison: Baseline vs. Patch Attacks. (Lower mAP indicates a stronger attack).

Metric	Baseline (Clean)	Black-box POC	RobustDPatch
mAP @ 0.50	0.7174	0.6375	**0.6169**
mAP @ 0.50:0.95	0.5629	0.4477	**0.4338**
Precision	0.8770	0.7781	0.8331
Recall	0.6462	0.5720	**0.5487**

3.1 Analysis

- **Impact on mAP:** Both attacks successfully degraded the model's performance. The Black-box attack reduced mAP@0.50 by approximately **11.1%**, while the RobustDPatch achieved a reduction of **14.0%**.
- **Recall Degradation:** The RobustDPatch proved superior in suppressing object discovery, driving Recall down to 0.5487 compared to the Black-box's 0.5720. This aligns with the RobustDPatch loss function, which explicitly targets the suppression of objectness scores.
- **Precision Variance:** Interestingly, the Black-box attack degraded Precision significantly more (0.7781) than the RobustDPatch (0.8331). This suggests that while the RobustDPatch acts as a "suppressor" (hiding objects), the random mutations of the Black-box patch may act more as a "distractor," potentially generating false positives or classifying background noise, thereby lowering precision.

4 Discussion

The results highlight the efficacy of adversarial patches in compromising aerial object detectors.

RobustDPatch (White-box) emerged as the most potent attack. By leveraging the model's gradients via our `YoloV9ARTDetector` wrapper, the optimizer could precisely manipulate pixel values to minimize object probability. The resulting patch is highly robust to the transformations applied during the Expectation over Transformations (EoT) process.

Black-box POC, despite its simplicity, was surprisingly competitive. Without access to gradients, the hill-climbing algorithm found a patch configuration that reduced mAP@0.50:0.95 from 0.56 to 0.44. This demonstrates that even with limited access (only prediction outputs), a query-based attack can pose a significant security risk. However, it required significantly more iterations to converge and lacked the targeted suppression capability of the white-box method.