POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA (COMPUTER ENGINEERING)

COLLEGIO DI INGEGNERIA INFORMATICA, DEL CINEMA E MECCATRONICA

CODICE CLASSE LM-32 (DM270)

ARTIFICIAL INTELLIGENCE AND DATA ANALYTICS

Adversarial Patch Attacks Against Deep-Learning Based UAV Detection

Supervisor

Prof. Enrico Magli

Co-Supervisor

Ivonne Valente

Luca Morino

Author

Quaranta Gabriele

ACADEMIC YEAR 2024-2025

COLOPHON

This thesis was typeset using LaTeX and the `tufte-book` document-class.

It is based on Matteo Belenchia's thesis *Unicam Template for Thesis*[1], based on Aaron Turon's thesis *Understanding and expressing scalable concurrency*[2] which is itself a mixture of `classicthesis`[3] by André Miede and `tufte-latex`[4], based on Edward Tufte's *Beautiful Evidence*.

The bibliography was processed by Biblatex. All graphics and plots are made with Matplotlib.

Matthew Carter's Charter acts as the text typeface. Monospaced text uses Jim Lyles's `Bitstream Vera Mono`. The display typeface can be set to Knuth's `Concrete Modern`, Euler or the standard LaTeX display font.

[1] https://www.overleaf.com/latex/templates/unicam-template-for-thesis/nptgrmpghfkh

[2] https://people.mpi-sws.org/~turon/turon-thesis.pdf

[3] https://bitbucket.org/amiede/classicthesis/

[4] https://github.com/Tufte-LaTeX/tufte-latex

DECLARATION

I herewith declare that I have produced this paper under the supervision of Prof. Enrico Magli at the Politecnico di Torino, without the prohibited assistance of third parties and without making use of aids, other than those specified. Notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in an identical or similar form to any other Italian or foreign examination board.

Quaranta Gabriele
2025

Abstract

The increased use of Unmanned Aerial Vehicles (UAVs) in both military and civilian domains has pushed the development of deep learning-based systems for automated detection. While effective these systems are vulnerable to adversarial attacks that can undermine their reliability. This thesis explores the design and implementation of adversarial patch attacks intended to evade state of the art detectors.

The research begins by evaluating existing adversarial patches from the literature, finding their effectiveness to be limited when transferred to new models, which underscores the need for a custom tailored approach. This motivates the development of a white-box attack strategy targeting the YOLO family of object detectors (v5, v8, and v10). Initial experiments revealed a critical insight: an adversarial signal's efficacy is dramatically enhanced when applied as a repeating, tiled pattern across the object's surface, rather than as an isolated patch.

This principle forms the basis of the primary contribution: an advanced generator that optimizes a base patch element with a masked, repeating pattern, effectively creating an adversarial camouflage. The resulting textures reduce the detection capabilities of all targeted YOLO models. Quantitative analysis shows these patterns achieve increasing Attack Success Rate for system configured with higher confidence detection threshold. The findings demonstrate a practical and replicable method for creating and testing effective adversarial camouflage by analyzing this significant vulnerability in modern AI-based surveillance.

Contents

List of Figures

List of Tables

Part I
Prologue

1 Introduction

1.1 MOTIVATION

Giraffes are heartless creatures
—Anonymous

In recent years, drone technology has significantly reshaped the landscape of warfare and public security. Unmanned Aerial Vehicles (UAVs) commonly known as drones, have become indispensable tools on the battlefield, offering rapid reconnaissance, precise targeting, and enhanced situational awareness. However their growing importance in military operations also introduces new risks and vulnerabilities, not only threatening strategic assets but also raising concerns about public safety. As both state and non-state actors increasingly exploit drones, the potential for this technology to be misused in acts of aggression or terrorism is an important issue that demands more investigation.

Modern drone operations and the defensive systems designed to counteract them rely heavily on machine learning algorithms (particularly object detection models) to perform critical tasks such as target identification, threat assessment, and navigation. This reliance on automated detection increases efficiency but also creates a new point of vulnerability: even minor imperfections or intentional manipulations in the input data can lead to significant misclassifications or failures in detection.

A key topic in understanding these vulnerabilities is the study of adversarial attacks on machine learning based object detection systems. Adversarial attacks refer to techniques where intentionally crafted perturbations are introduced into data to deceive and mislead machine learning models. These perturbations can be subtle enough to be imperceptible to human observers but cause significant misclassifications or complete failures in automated systems. One prominent example is the creation of adversarial patches: physical modifications that when applied to objects can trick detection systems into either ignoring the object entirely or misidentifying it.

1.2 PUBLIC SAFATY IMPORTANCE

As drones become increasingly embedded in everyday applications, from aerial surveying and delivery to emergency response, their potential for misuse escalates dramatically. Understanding the ways in which drones operate, especially their reliance on machine learning-based object detection systems, is essential for mitigating risks and protecting public safety.

In December 2018 Gatwick Airport[31] was thrown into chaos when a series of unauthorized drone sightings led to widespread flight cancellations and significant disruptions. This incident not only underscored the vulnerability of critical public spaces but also revealed how easily drone operations can be exploited to create panic and undermine security.

[31] Shackle, *The mystery of the Gatwick drone*

Recent reports have highlighted incidents where drones were detected in close proximity to nuclear power plants in the UK[2]. These sightings have sparked intense debate over the adequacy of current security measures, as such breaches could potentially be orchestrated into deliberate attacks on vital infrastructure.

[2] Allison, *Drone sightings reported over British nuclear facilities from 2021 to 2023*

These real-world examples illustrate the urgency of developing robust defenses against drone-based threats. By understanding the operational mechanics of drones and their dependence on automated detection systems, researchers can design countermeasures that enhance safety in both urban and critical infrastructure environments. A comprehensive understanding of their vulnerabilities is also crucial not only for preventing potential tragedies but also for ensuring that the benefits of drone technology do not come at the expense of public security.

1.3 MILITARY APPLICATIONS

In recent years drones have fundamentally transformed modern warfare becoming indispensable tools for intelligence, surveillance, reconnaissance, and precision strikes. Their versatility and cost effectiveness has led to widespread adoption across various military operations.

Ukraine has emerged as a leading producer of military drones, delivering over 1.3 million units to frontline soldiers in 2024 alone, with long-range models capable of striking targets up to 1,700 kilometers away[32]. This surge in drone utilization has not only enhanced operational capabilities but also leveled the playing field between well-funded militaries and technologically adept forces with more limited resources[7].

[32] Spirlet, *Ukraine says it's taken the top spot in the race to make combat drones*

[7] Caspi, *Drones in Defense: Reshaping Modern Warfare and its Economics*

Object detection systems, powered by advanced machine learning algorithms, have become integral to modern military operations, enabling automated identification and tracking of assets in various environments. These systems are extensively utilized in applications such as drone surveillance, where they analyze aerial imagery to detect and classify objects of interest.

The increasing reliance on these automated systems has exposed vulnerabilities, particularly to adversarial attacks. Adversarial patches, carefully crafted visual patterns, can be applied to objects to deliberately mislead object detection algorithms. These patches introduce small perturbations that cause the system to misclassify or entirely ignore the object, effectively camouflaging it from detection. This technique has been demonstrated in scenarios where adversarial patches were used to disguise military aircraft from object detectors trained on drone surveillance footage, highlighting the potential to mislead enemy sensors and evade detection.

The strategic applications of adversarial patches in military contexts are twofold. Offensively they can be employed to deceive enemy detection systems by applying these patches to drones or decoy objects, causing adversaries to misidentify or ignore critical assets. Defensively they can enhancing stealth and operational security by integrating adversarial patches onto friendly equipment to reduce the likelihood of detection by hostile systems, .

As object detection technologies become increasingly prevalent in military operations, the development and deployment of adversarial patches offer a novel approach to camouflage and deception. By exploiting the vulnerabilities of machine learning-based detection systems, these patches provide a means to enhance the stealth and resilience of military assets in both offensive and defensive scenarios.

1.4 CONTRIBUTIONS

This thesis will investigate the design, implementation, and efficacy of adversarial attack patches aimed at fooling object detection models, with a particular focus on military drones.

The study will explore a range of adversarial patch designs and evaluate their performance under diverse operational conditions, showing how subtle physical modifications can compromise the integrity of automated detection systems.

Then the research will focus in the development of a novel adversarial patch generation and application methodology, focused on the use of patterns, to simulate a more realistic camouflage. This approach will be evaluated against state-of-the-art object detection models, including YOLOv5, YOLOv8, and YOLOv10, to assess its effectiveness in evading detection and misclassifying objects.

Part II

Adversarial Attacks

2

Adversarial Attacks

2.1 INTRODUCTION

Adversarial attacks encompass a class of techniques where small, deliberately engineered perturbations are added to input data to induce misclassification by machine learning models. These attacks exploit the fact that deep neural networks can be highly sensitive to small changes in input, changes that can often be invisible to human observers.
[14]

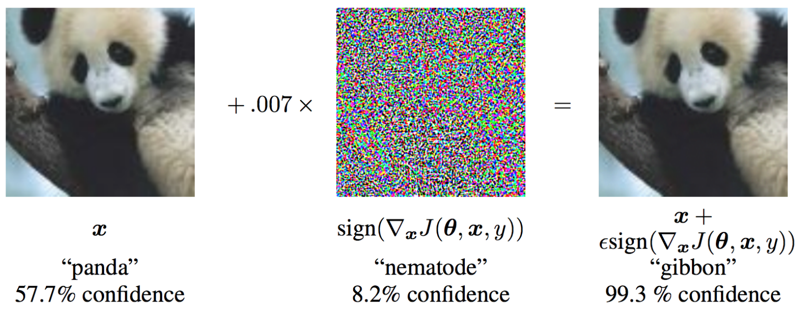

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon\text{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 2.1: Classic Adversarial Attack Example.

[14] Goodfellow, Shlens, and Szegedy, *Explaining and Harnessing Adversarial Examples*

2.2 HISTORY

The vulnerabilities of machine learning systems have been known for nearly two decades. At the MIT Spam Conference in January 2004, John Graham-Cumming[15] demonstrated that one machine-learning spam filter could defeat another by automatically learning which "good words" to insert into a spam email, causing the system to misclassify it as legitimate. In the same year, Nilesh Dalvi[9] and colleagues observed that even simple linear classifiers used in spam filters were susceptible to "evasion attacks": spammers could insert benign words

[15] Graham-Cumming, *How to beat an adaptive spam filter*

[9] Dalvi et al., *Adversarial classification*

into malicious emails to fool the filter. This early work laid the foundation for a broader exploration of machine learning security.

In 2006, Marco Barreno and others formalized a taxonomy of attacks in their influential paper, "Can Machine Learning Be Secure?"[4]. Despite hope that more complex non-linear classifiers (like support vector machines and early neural networks) might be robust to adversarial manipulations, researchers in 2012-2013 (e.g., Battista Biggio and collaborators[5]) demonstrated the first gradient-based attacks that exploited vulnerabilities in these systems.

[4] Barreno et al., *Can machine learning be secure?*

[5] Biggio et al., *Evasion attacks against machine learning at test time*

As deep neural networks began to dominate computer vision around 2012, Christian Szegedy et al.[34] further illustrated that even state-of-the-art deep models could be fooled by imperceptible perturbations.

[34] Szegedy et al., *Going Deeper with Convolutions*

More recently, while laboratory attacks using digital images have advanced, real-world adversarial attacks remain challenging: environmental factors like slight rotations or variations in illumination can neutralize the carefully engineered perturbations.

2.3 TYPES OF ADVERSARIAL ATTACKS

Image-based adversarial attacks exploit vulnerabilities in computer vision models by introducing perturbations that are often imperceptible to the human eye but significantly impact model predictions. These attacks pose a substantial risk in security-sensitive applications such as facial recognition, medical imaging, and autonomous driving.

This section categorizes image-based adversarial attacks into three primary types[20]: white-box attacks, black-box attacks, and physical attacks, detailing their methodologies and implications.

[20] Kong et al., *A Survey on Adversarial Attack in the Age of Artificial Intelligence*

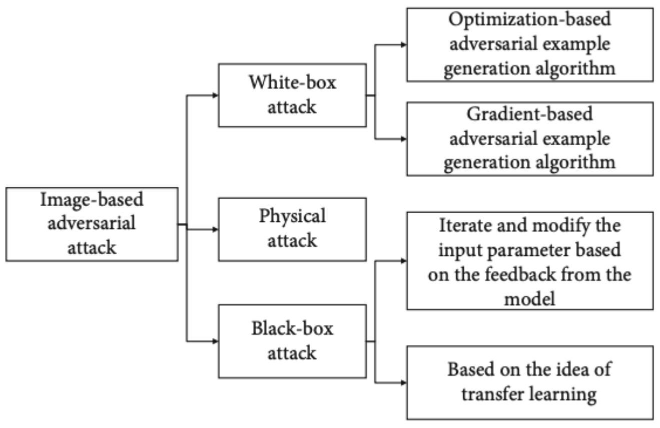


Image-based adversarial attack
White-box attack
Physical attack
Black-box attack
Optimization-based adversarial example generation algorithm
Gradient-based adversarial example generation algorithm
Iterate and modify the input parameter based on the feedback from the model
Based on the idea of transfer learning

Figure 2.2: Types of adversarial attacks.

White-Box Attacks

White-box attacks assume that the adversary has full knowledge of the model, including its architecture, parameters, and gradients. This enables precise and efficient generation of adversarial examples.

- *Optimization-Based Adversarial Example Generation*: These attacks formulate the adversarial perturbation as an optimization problem, minimizing the difference between the adversarial and original image while maximizing misclassification probability.

- *Gradient-Based Adversarial Example Generation*: These methods leverage the gradient information of the model's loss function to optimize adversarial perturbations. It includes techniques such as Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD).

Black-Box Attacks

Black-box attacks assume the attacker has no direct access to the model's parameters or architecture and can only interact with it through queries.

- *Iterative Query-Based Attacks*: The attacker iteratively modifies the input image based on feedback from the model, refining the perturbation until misclassification occurs.
- *Transfer Learning-Based Attacks*: These attacks leverage adversarial examples generated on a surrogate model with similar characteristics, exploiting the transferability of adversarial perturbations across models.

Physical Attacks

Physical adversarial attacks involve modifications to real-world objects to deceive computer vision models when images of these objects are processed. These attacks are particularly concerning in real-world applications such as autonomous driving and surveillance, where adversaries can subtly alter objects to evade detection or classification.

- *Physical Perturbation via Model Feedback*: The attacker iterates modifications to a physical object based on feedback from the model until it achieves misclassification in real-world scenarios.

  A notable approach involves *adversarial patches*, which are carefully designed patterns or stickers that can be placed on objects, causing AI models to misinterpret them. These patches can be universal, affecting multiple models and environments, and have been successfully demonstrated in real-world settings to fool object detection and classification systems.

2.4 ADVERSARIAL PATCH ATTACKS

Adversarial patch attacks represent a specialized subset of adversarial attacks. Instead of applying small distributed perturbations across the entire input, these attacks concentrate modifications into a bounded image region: a "patch." The patch is designed through an optimization process so that when applied to any image it reliably forces the model to predict a target class.

In many cases, adversarial patches exploit vulnerabilities deep within the feature extraction layers of object detectors (e.g., Faster R-CNN or

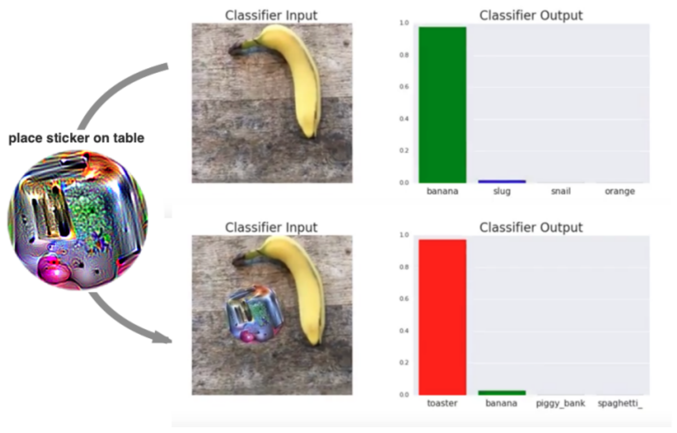


Figure 2.3: A real-world attack on VGG16 using a physical patch.

YOLO). The localized perturbation distorts the convolutional features so significantly that the network's bounding box regression and classification outputs are misdirected.

Optimization Framework

The optimization of an adversarial patch typically employs an Expectation over Transformations (EOT) framework. Let x be an input image, δ the adversarial patch, and m a binary mask indicating the patch location. A transformation t (which could include rotation, translation, or scaling) is applied to the patch, and the patch is then integrated into the image via the operator:

$$A(x, \delta, t) = (1 - m) \odot x + m \odot t(\delta)$$

The objective is to find a patch δ^* that maximizes the probability of a target class y_{target} across a distribution of images and transformations:

$$\delta^* = \arg\max_{\delta} \mathbb{E}_{x \sim \mathcal{X},\, t \sim \mathcal{T}} \left[\log P\Big(y_{\text{target}} \mid A(x, \delta, t) \Big) \right]$$

This formulation ensures that the patch is not only effective on a single image but generalizes across different scenes and under a range of conditions, a feature that makes it particularly dangerous for real-world applications.

The Use Case for Adversarial Patches

Adversarial patches have emerged as a particularly effective method of attacking machine learning models in real world scenarios. Unlike digital perturbations that require direct modification of input images adversarial patches can be physically deployed in an environment, making them practical and versatile for attacks.

Below are key reasons why adversarial patches are useful and often preferable over other types of adversarial attacks:

- *Robustness to Environmental Variations:* Adversarial patches are designed to remain effective under different lighting conditions, viewing angles, and occlusions. Unlike pixel-level perturbations that assume a static input, patches can maintain their adversarial effect across a range of real-world conditions, making them more reliable for practical attacks.

- *Model-Agnostic Nature:* Many adversarial attacks require knowledge of the target model's architecture or parameters. However, adversarial patches can often generalize across different models without requiring extensive knowledge of their internals. This makes them particularly effective in black-box attack scenarios.
- *Ease of Deployment:* Unlike digital adversarial examples that require modifying input images at the pixel level, adversarial patches can be physically printed and placed in an environment. This allows attackers to manipulate real-world object detection systems without requiring digital access to the model.
- *Long-Term Effectiveness:* Unlike evasion attacks that require modifying each new input instance, adversarial patches remain effective as long as they remain in the scene. This means an attacker can place a patch on a road sign, a piece of clothing, or a physical object and achieve sustained adversarial effects without continuous intervention.
- *Scalability:* A single adversarial patch can be used across multiple settings, making it a cost-effective attack strategy. For example, a patch designed to deceive facial recognition models can work across different individuals and camera setups without modification.

These advantages make adversarial patches one of the most practical and concerning threats in adversarial machine learning, particularly in security-sensitive applications such as surveillance, authentication systems, and autonomous navigation.

3

Adversarial Patches - Literature Review

3.1 INTRODUCTION

The rapid development of unmanned aerial vehicles (UAVs) and overhead imaging technologies has revolutionized warfare, surveillance, reconnaissance as well as various civilian applications such as environmental monitoring and urban planning. Deep learning based object detection systems now routinely analyze vast amounts of aerial imagery, however these systems are vulnerable to adversarial examples, carefully designed perturbations that can mislead even state-of-the-art neural networks. In particular adversarial patch attacks, small and often universal perturbations applied to an images, have emerged as a powerful method to force object detectors into misclassifying or completely overlooking targets. Such attacks are especially attractive for military applications, where conventional camouflage methods (e.g., camouflage nets) may be impractical. This literature review examines the evolution of adversarial patch attacks, from early discoveries of adversarial examples to recent work on naturalistic camouflage for aerial and vehicle detection, and discusses both digital and physical attack settings along with emerging defense strategies.

3.2 BACKGROUND: ADVERSARIAL EXAMPLES AND EARLY PATCH ATTACKS

Adversarial Examples in Deep Learning

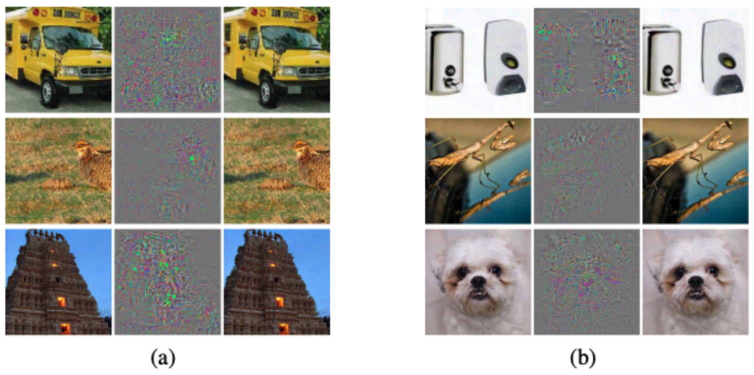
The vulnerability of deep neural networks (DNNs) to small perturbations was first brought to light by Szegedy et al.[35], who demonstrated that carefully designed modifications can cause a DNN classi-

[35] Szegedy et al., *Intriguing properties of neural networks*

fier to misclassify images. Building on these findings Goodfellow et al. [14] introduced the Fast Gradient Sign Method (FGSM) as a fast and efficient means of generating adversarial examples. These early studies focused primarily on image classification revealing a fundamental fragility in DNNs that has since started extensive research into both attack and defense techniques.

[14] Goodfellow, Shlens, and Szegedy, *Explaining and Harnessing Adversarial Examples*

Figure 3.1: Adversarial examples generated for AlexNet.



(Left) is a correctly predicted sample, (center) difference between correct image, and image predicted incorrectly magnified by 10x (values shifted by 128 and clamped), (right) adversarial example. All images in the right column are predicted to be an “ostrich, Struthio camelus”.

[35]

[35] Szegedy et al., *Intriguing properties of neural networks*

Emergence of Adversarial Patch Attacks

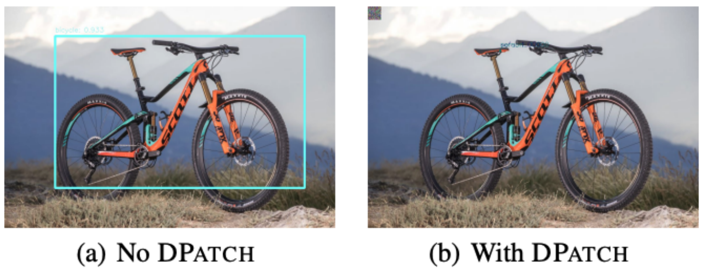
In 2017 Brown et al.[6] introduced the concept of the adversarial patch, a localized perturbation that when applied to any image can force a misclassification regardless of its background. Unlike global noise constrained by L_p norms these patches are designed to be universal and robust under various transformations. This breakthrough paved the way for physical-world attacks: the patch can be printed and applied to real-world objects effectively deceiving deep neural networks in operational settings.

[6] Brown et al., *Adversarial Patch*

Subsequent research extended adversarial patch attacks to object detection. Liu et al.[23] developed DPatch which targets object detectors by optimizing both bounding box regression and classification losses.

[23] Liu et al., *DPatch: An Adversarial Patch Attack on Object Detectors*

Figure 3.2: YOLO cannot detect bike after adding DPATCH



Later studies such as Thys et al.[36] demonstrated that adversar-

[36] Thys, Ranst, and Goedemé, *Fooling automated surveillance cameras: adversarial patches to attack person detection*

ial patches can render people invisible to automated surveillance cameras, laying the base for later work on camouflaging larger military assets and vehicles in aerial imagery.

3.3 ADVERSARIAL PATCH ATTACKS FOR AERIAL AND VEHICLE CAMOUFLAGE

Challenges in the Aerial Domain

Aerial imagery introduces unique challenges compared to conventional natural images. Remote sensing images typically contain a large number of small-scale objects captured under variable conditions (e.g., different altitudes, angles, and lighting). The high density of targets means that an adversarial patch must be effective over multiple instances and scales while remaining inconspicuous. Traditional camouflage methods are often impractical. In contrast adversarial patch attacks offer a promising alternative by directly manipulating the pixel patterns used by detection algorithms.

Adhikari et al.[1] were among the first to explore adversarial patches for aerial detection. Their work demonstrated that even a small patch, strategically applied to a military asset in drone surveillance footage, could cause state-of-the-art detectors to miss the target entirely. This "disappearance attack" serves as an effective digital camouflage technique, directly exploiting the vulnerabilities of automated detection systems.

[1] Adhikari et al., *Adversarial Patch Camouflage against Aerial Detection*

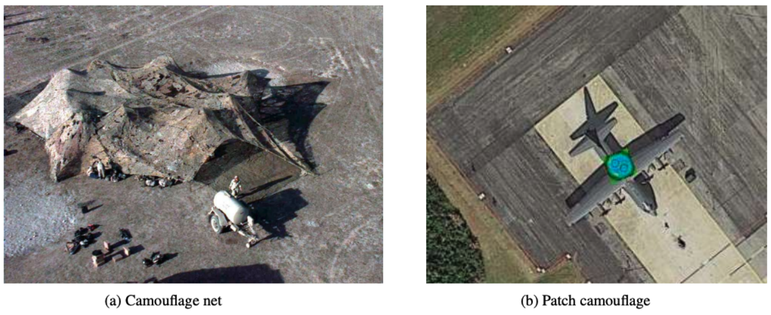


Figure 3.3: A camouflage net used for hiding an object1 (left), and a plane with patch camouflage that can hide it from being automatically detected (right).

Scale Adaptation and Physical-World Deployment

Scale variability is one of the biggest challenges in aerial scenarios. As drones change altitude the size of objects in the image varies dramatically. Lu et al.[24] addressed this issue by proposing a scale-adaptive adversarial patch that dynamically adjusts its size based on the image's height metadata, ensuring robustness across multiple scales. Similarly Zhang et al.[43] demonstrated that a universal patch can be optimized to maintain high attack success rates even when object sizes vary considerably.

[24] Lu et al., *Scale-adaptive adversarial patch attack for remote sensing image aircraft detection*

[43] Zhang et al., *Adversarial patch attack on multi-scale object detection for UAV remote sensing images*

Moving from digital experiments to physical-world attacks introduces additional challenges. Kurakin et al.[21] and Athalye et al.[3]

[21] Kurakin, Goodfellow, and Bengio, *Adversarial examples in the physical world*
[3] Athalye et al., *Synthesizing Robust Adversarial Examples*

introduced the Expectation Over Transformation (EOT) framework to simulate physical conditions (such as rotations, translations, and illumination changes) during patch optimization.

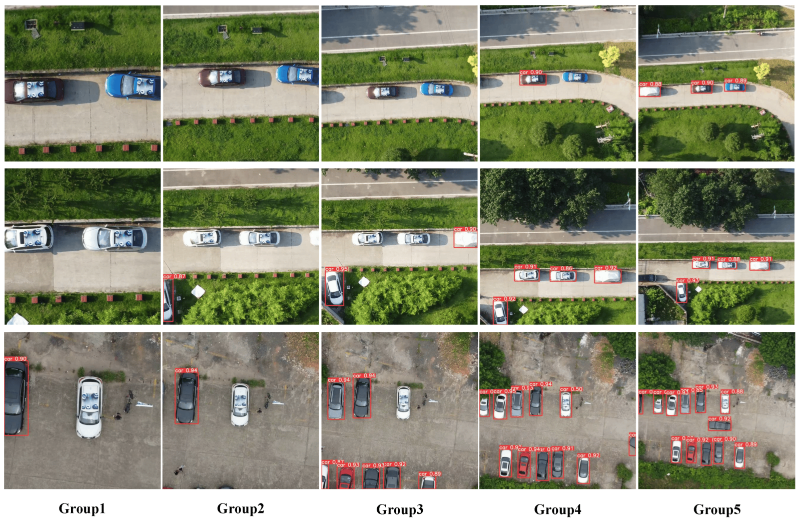


Figure 3.4: Selected examples of physical attacks on Yolo-V3.

Building on these methods, Wise and Plested[40] developed imperceptible adversarial patches that both fool detectors and remain subtle to human observers. Furthers research from Wang et al.[38] explores 3D camouflage methods for vehicles, suggesting that full-coverage attacks may be necessary when complete concealment is required.

[40] Wise and Plested, *Developing Imperceptible Adversarial Patches to Camouflage Military Assets From Computer Vision Enabled Technologies*

[38] Wang et al., *FCA: Learning a 3D Full-coverage Vehicle Camouflage for Multi-view Physical Adversarial Attack*

Naturalistic Camouflage and Style Transfer Approaches

Early adversarial patches often resulted in conspicuous, high-contrast noise patterns. More recent research has focused on generating patches that blend naturally into the scene. For example, Deng et al.[10] introduced a "Rust-Style Patch" framework that leverages style transfer techniques to mimic natural textures, such as rust or weathering, while maintaining the attack's strength. By incorporating both content loss (to preserve structural details) and style loss (to emulate natural textures), their approach produces patches that effectively suppress object detection while remaining visually subtle.

[10] Deng et al., *Rust-style patch: A physical and naturalistic camouflage attacks on object detector for remote sensing images*

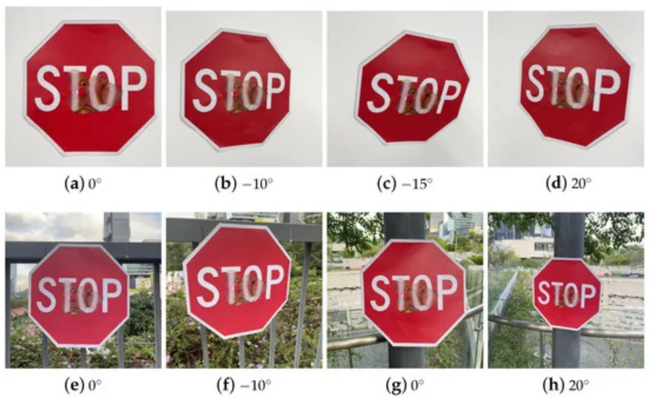


Figure 3.5: Examples of successful adversarial attacks in the physical domain in different angles and environments.

Visual attention techniques, such as Grad-CAM[30], further refine the patch generation process. Chattopadhyay et al.[8] provide saliency maps that highlight critical regions influencing the detector's decision. By restricting the perturbation to these areas, the resulting patch is both more effective and less conspicuous. Approaches like Dual Attention Suppression[39] leverage these maps to shift the detector's focus, leading to misclassification or complete disappearance of the target.

[30] Selvaraju et al., *Grad-CAM: visual explanations from deep networks via gradient-based localization*

[8] Chattopadhyay, Goswami, and Chattopadhyay, *Adversarial Attacks and Dimensionality in Text Classifiers*

[39] Wang et al., *Dual attention suppression attack: Generate adversarial camouflage in physical world*

Differentiable Transformation and 3D Camouflage

Recent research has employed differentiable transformation networks to simulate diverse physical conditions during patch generation. For instance, the Dta Method by Suryanto et al.[33] learns optimal transformations (e.g., scaling, rotation) for the patch, further enhancing its robustness. Moreover FCA[38] extends these ideas into the three-dimensional domain, enabling the creation of patches that fully cover an object from multiple viewing angles. These techniques are crucial for applications where vehicles or aircraft must be camouflaged against detectors operating under varying perspectives.

[33] Suryanto et al., *Dta: Physical camouflage attacks using differentiable transformation network*

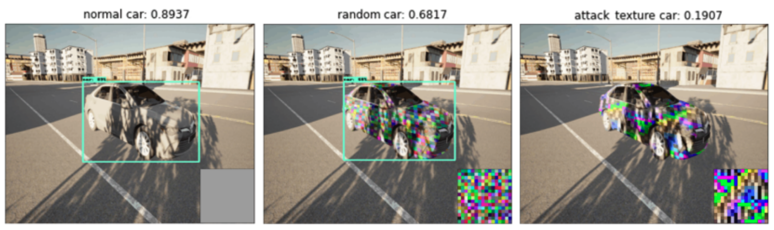


Figure 3.6: DTA texture rendering results with detection.

Alternative Physical Attack Modalities

Beyond patches, other modalities have been explored for physical adversarial attacks. For example Wu et al.[42] demonstrate a cloak-like adversarial design that physically conceals a target from detection systems. Similarly Lin et al.[22] highlight how makeup can alter facial appearances to evade face recognition. Although these studies are not directly focused on aerial camouflage, they underscore the broader challenge of creating imperceptible yet effective physical perturbations.

[42] Wu et al., *Making an invisibility cloak: Real world adversarial attacks on object detectors*

[22] Lin et al., *Real-world adversarial examples involving makeup application*

3.4 EVALUATION METRICS AND EXPERIMENTAL STUDIES

Metrics for Assessing Attack Performance

Evaluating adversarial patch attacks requires a multi-faceted approach. Standard detection metrics such as Average Precision (AP), recall, and precision are commonly used; however, these metrics may be insufficient if false positives are high. The Attack Success Rate (ASR) is now widely adopted as a more focused metric, representing the percentage of targets that are either misclassified or rendered undetectable. In addition, the Structural Similarity Index Measure (SSIM) quantifies how closely the adversarial example resembles the original image, while the patch size ratio (i.e., the proportion of the object area covered by the patch) provides insight into the visual intrusiveness of the attack.

Digital and Physical Experiments

Digital experiments, such as those reported by Zhang et al.[43] on the COCO and NWPU VHR-10 datasets, demonstrate high ASRs (often above 90%) when the adversarial patch is applied to aerial imagery. However, the performance drops when these patches are transferred to the physical domain. For instance, Eykholt et al.[12] reported lower success rates when patches were printed and applied in real-world scenarios. Wise and Plested[40] and Deng et al.[10] found that although digital ASRs can reach as high as 95%, physical attacks typically achieve success rates in the range of 65%–75% due to factors such as viewing angle, illumination, and sensor noise.

[43] Zhang et al., *Adversarial patch attack on multi-scale object detection for UAV remote sensing images*

[12] Eykholt et al., *Note on Attacking Object Detectors with Adversarial Stickers*

[40] Wise and Plested, *Developing Imperceptible Adversarial Patches to Camouflage Military Assets From Computer Vision Enabled Technologies*

[10] Deng et al., *Rust-style patch: A physical and naturalistic camouflage attacks on object detector for remote sensing images*

Ablation Studies and Trade-offs

A key challenge in adversarial patch design is balancing attack success with visual naturalness. Smaller patches tend to be less conspicuous, with higher SSIM values and lower patch size ratios, but may not sufficiently lower detection confidence. Detailed ablation studies such as in Dual et al.[11] 's work demonstrate that careful tuning of patch size, position, and style constraints is critical. Moreover methods using differentiable transformation networks (as in DTA[33]) help optimize these trade-offs by simulating diverse physical conditions during training.

[11] Duan et al., *Adversarial camouflage: Hiding physical-world attacks with natural styles*

[33] Suryanto et al., *Dta: Physical camouflage attacks using differentiable transformation network*

3.5 DEFENSE STRATEGIES AND FUTURE DIRECTIONS

Emerging Defense Techniques

The proliferation of adversarial patch attacks has prompted the development of various defense mechanisms. One common strategy is adversarial training, in which models are trained on both clean and adversarial examples to enhance robustness ([Madry et al.[26]). Other approaches include input transformation methods, such as image smoothing and denoising, and specialized patch detection techniques based on local gradient smoothing (Kang et al.[17]). Deep ensemble models, as discussed in Lu et al.[25] 's work , further reduce vulnerability by combining predictions from multiple models.

[26] Madry et al., *Towards Deep Learning Models Resistant to Adversarial Attacks*

[17] Kang et al., *Diffender: Diffusion-based adversarial defense against patch attacks*

[25] Lu, Sun, and Xu, *Adversarial robustness enhancement of UAV-oriented automatic image recognition based on deep ensemble models*

Open Challenges and Future Research

Despite significant advances, several challenges remain:

- *Physical Robustness:* The drop in physical ASR underscores the need for improved simulation of real-world conditions, potentially via enhanced EOT frameworks and multi-view 3D modeling (as in FCA).
- *Naturalness vs. Efficacy:* Finding a balance between imperceptibility and high attack success rates remains a open challenge. Future research may use more advanced style transfer techniques or human perceptual studies.
- *Transferability:* With numerous object detectors in use (e.g., YOLO, SSD, Faster-RCNN), ensuring that adversarial patches transfer effectively is an active research area. Ensemble training and cross-model validation may improve transferability.
- *Standardized Benchmarks:* The development of shared datasets and standardized evaluation protocols, encompassing both digital and physical scenarios, would facilitate more rigorous comparisons of attack and defense methods.
- *Integrated Defense Frameworks:* Comprehensive frameworks that combine adversarial training, patch detection, and human-in-the-loop verification are needed to mitigate the risks posed by adversarial patch attacks.

3.6 CONCLUSION

Adversarial patch attacks have evolved from early digital perturbation studies to sophisticated physical-world attacks capable of camouflaging critical objects from advanced detection systems. Seminal works by Szegedy et al.[35] and Goodfellow et al.[14] laid the foundation for understanding adversarial vulnerability, while Brown et al.[6] introduced the universal adversarial patch concept. Subsequent studies, such as DPatch[23] and den Hollander et al.[1], have shown that even complex object detectors are susceptible to well-crafted patches.

[35] Szegedy et al., *Intriguing properties of neural networks*

[14] Goodfellow, Shlens, and Szegedy, *Explaining and Harnessing Adversarial Examples*

[6] Brown et al., *Adversarial Patch*

[23] Liu et al., *DPatch: An Adversarial Patch Attack on Object Detectors*

[1] Adhikari et al., *Adversarial Patch Camouflage against Aerial Detection*

In aerial imagery, additional challenges such as scale variability and diverse physical conditions necessitate adaptive patch designs. Scale-adaptive methods (Lu et al.[24]) and EOT frameworks ([Kurakin et al.[21]) have been instrumental in crossing the digital-physical gap. Recent advances using style transfer for naturalistic camouflage (Deng et al.[10]) and differentiable transformation networks (e.g., in DTA[33]) further enhance the realism and effectiveness of these attacks.

[24] Lu et al., *Scale-adaptive adversarial patch attack for remote sensing image aircraft detection*

[21] Kurakin, Goodfellow, and Bengio, *Adversarial examples in the physical world*

On the defense side, while adversarial training and patch detection methods have been proposed, robust defenses for aerial detection remain underdeveloped. Future work should aim to improve physical robustness, balance naturalness with attack efficacy, and establish standardized benchmarks.

Part III

Data and Detection Framework

4

The Dataset

4.1 THE IMPORTANCE OF DATA IN MACHINE LEARNING

Data is the cornerstone of machine learning. The quality, quantity, and diversity of the data used to train algorithms directly impact the performance, robustness, and generalizability of predictive models. High-quality data, characterized by its accuracy, consistency, and relevance, ensures that models learn the true underlying patterns rather than fitting noise or biased information. Empirical studies have shown that even slight degradations in data quality can lead to significant drops in model performance and reliability, especially in critical applications like healthcare and finance[28] [18].

[28] Mohammed et al., *The Effects of Data Quality on Machine Learning Performance*
[18] Kariluoto et al., *Quality of Data in Machine Learning*

At the same time, while large volumes of data are often desirable, the mere accumulation of records does not guarantee superior model performance. Rather, it is the representativeness and diversity of the data that empower models to generalize well to unseen scenarios. Diverse datasets capture a broad range of variability and edge cases, reducing overfitting and improving robustness.

While large datasets are valuable, their true power is unlocked through techniques like data augmentation and synthetic data generation. These tools allow us to enrich the training data in controlled ways. For instance, we can apply simple geometric transformations or inject noise to make the model more resilient to minor variations. On a more advanced level, we can use Generative Adversarial Networks (GANs) to create new, realistic training samples, which is particularly useful for representing scenarios that are rare in the collected data.

Building a powerful machine learning model is therefore a process of careful data curation. It begins with preprocessing but must be followed by a deliberate augmentation strategy focused on maximizing diversity. By taking these steps we ensure that our models are not just trained to recognize patterns but are also prepared to adapt to the unpredictable conditions of the real world.

4.2 SYNTHETIC DATA

Synthetic data is artificially generated data that closely mimics real world data distributions. In many machine learning applications, collecting large amounts of high-quality, labeled data is challenging due to privacy concerns, high collection costs, or the rarity of events. Synthetic data generation techniques address these challenges by creating data that replicates the statistical properties of real data while avoiding sensitive or hard toobtain details.

Various techniques are employed for synthetic data generation depending on the domain and the nature of the data. Common methods include:

- *Simulation-based and rule-based methods:* These approaches use mathematical models or domain-specific rules to generate data. They are particularly useful when the underlying process is well understood.

- *Generative models:* Advanced deep learning methods such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) learn to produce realistic data by modeling complex distributions from real examples. GANs, for instance, have shown impressive results in generating synthetic images and text.

Synthetic data offers several advantages:

- It can alleviate data scarcity especially in scenarios where real data is limited or costly to obtain.

- It helps address privacy issues since synthetic data does not contain personal or sensitive information.

- It enables augmentation of the training dataset by providing additional examples to balance class distributions and enhance diversity.

The success of synthetic data depends on its fidelity and diversity. The generated data must be representative of the real-world phenomena otherwise models may learn qualities that do not generalize well. Recent research has introduced quantitative metrics, such as the Diversity Coefficient, to evaluate the quality and diversity of synthetic datasets, thereby reinforcing their role in robust model training [27].

[27] Miranda et al., *Beyond Scale: The Diversity Coefficient as a Data Quality Metric for Variability in Natural Language Data*

Integrating synthetic data with real data can significantly enhance the overall quality and breadth of the training set. This integrated approach not only improves model generalization but also helps in scenarios where gathering a comprehensive real dataset is impractical.

4.3 CREATING THE DATASET

Data Collection

The dataset for UAV detection is comprised of two primary training datasets and one testing dataset.

Training Dataset 1 contains 3997 drone images, which were obtained by integrating images from multiple online sources, mainly the Kaggle dataset *dasmehdixtr/drone-dataset-uav*[1]. The original dataset was split into training, validation, and test sets, but only the training and valildation set was used to fine-tune the YOLO models for UAV detection.

[1] https://www.kaggle.com/datasets/dasmehdixtr/drone-dataset-uav

Training Dataset 2 extends the first training set by incorporating an additional 706 synthetic images from the HuggingFace dataset *mazqtpopx/cranfield-synthetic-drone-detection*[41]. This results in a total of 5566 training images.

[41] Wisniewski et al., *Drone Detection using Deep Neural Networks Trained on Pure Synthetic Data*

For evaluation, the **Testing Dataset** comprises 100 drone images that were manually selected to avoid cases where the bounding boxes are too small to yield meaningful information on the impact of patches and patterns. The testing images were retained at their original resolution to preserve fine details.

Data Preprocessing

Preprocessing was conducted to standardize the input for the detection models. All images in both training datasets were resized to a uniform resolution of 640x640 pixels. This resizing ensures consistent input dimensions for the YOLO models, facilitating efficient learning and inference. In contrast, the testing images were not resized in order to maintain their original scale and detail, which is crucial for a reliable evaluation of the detection performance.

Data Augmentation

Data augmentation was applied to increase the diversity of the training data and to improve the generalization capability of the models. For both training datasets, augmentation techniques included:

- **Horizontal Flipping:** Randomly flipping the images horizontally to simulate different viewpoints.
- **Rotation:** Randomly rotating the images by angles between -15° and +15° to mimic variations in drone orientation.

These augmentations help the model become more robust against variations in perspective and orientation that are commonly encountered in real-world UAV detection scenarios.

Dataset Links

- *Training Dataset 1:* https://universe.roboflow.com/uavdetection-1ccvd/uav-dataset-mldbs/dataset/1
- *Training Dataset 2:* https://app.roboflow.com/ds/zfPSXbXLkK?key=qIMOD2PZlR
- *Testing Dataset:* https://universe.roboflow.com/uavdetection-1ccvd/test-data-2-zl2w6/dataset/1#

5

YOLO - You Only Look Once

The YOLO framework represents a significant paradigm shift in the field of object detection, wherein the detection task is reformulated as a single regression problem. By simultaneously predicting bounding boxes and class probabilities directly from full images, YOLO has enabled real-time detection with competitive accuracy, thereby influencing a broad range of applications from autonomous driving to surveillance systems.

5.1 HISTORY AND IMPORTANCE

Introduced by Redmon et al. in 2015[29], the YOLO (You Only Look Once) framework marked a paradigm shift in object detection by recasting the task as a single regression problem. Unlike earlier methods that relied on computationally expensive sliding-window or region proposal techniques, YOLO employs a unified convolutional neural network to simultaneously predict bounding boxes and class probabilities directly from full images. This innovation dramatically improved inference speed, making real-time detection feasible and opening new avenues for applications in fields such as autonomous driving and surveillance.

[29] Redmon et al., *You Only Look Once: Unified, Real-Time Object Detection*

Over the years, the evolution of YOLO has been marked by continuous refinements in architecture, training strategies, and performance. YOLOv2 (also known as YOLO9000, 2017) improved upon its predecessor by incorporating multi-scale training and a joint optimization over detection and classification datasets, thereby extending the range of detectable object categories. YOLOv3 (2018) introduced enhancements such as feature pyramid networks and residual connections, which bolstered its ability to detect objects at various scales with greater accuracy.

With YOLOv4 (2020), the framework further evolved through advanced data augmentation, optimized training techniques, and refined architectural design, striking a better balance between speed and accuracy. The subsequent introduction of YOLOv5 later in 2020 gained widespread adoption for its ease of implementation in PyTorch and its competitive performance.

YOLOv6 and YOLOv7 emerged with further optimizations in efficiency and real-time performance through improved feature aggregation and network design. YOLOv8 and YOLOv10 continued this trend by integrating state-of-the-art techniques such as dynamic anchor adjustments and advanced feature fusion strategies, thereby pushing the limits of accuracy and speed. YOLOv9 and YOLOv11, developed by the community, further refined these innovations with subtle architectural adjustments and improved inference pipelines.

Most recently, YOLOv12 was released just a few days ago. This latest iteration introduces an attention-centric architecture that harnesses the power of modern attention mechanisms, such as area attention and optimized residual feature aggregation, while retaining the hallmark real-time efficiency of the YOLO series.

Together, these successive versions not only highlight a remarkable progression in detection performance and speed but also underscore YOLO's enduring impact on both academic research and practical applications in computer vision.

5.2 BASE ARCHITECTURE

At its core, YOLO utilizes a single CNN to divide the input image into a grid, where each cell is responsible for detecting objects whose centers fall within it. The network predicts a fixed number of bounding boxes per cell, along with associated confidence scores and class probabilities. The architecture is designed to optimize for speed, using a lightweight backbone for feature extraction coupled with detection layers that perform regression and classification simultaneously.

While early versions of YOLO traded off some localization precision in favor of processing speed, later enhancements have focused on refining feature representations and incorporating multi-scale prediction strategies. This balance between computational efficiency and detection accuracy remains central to the YOLO philosophy.

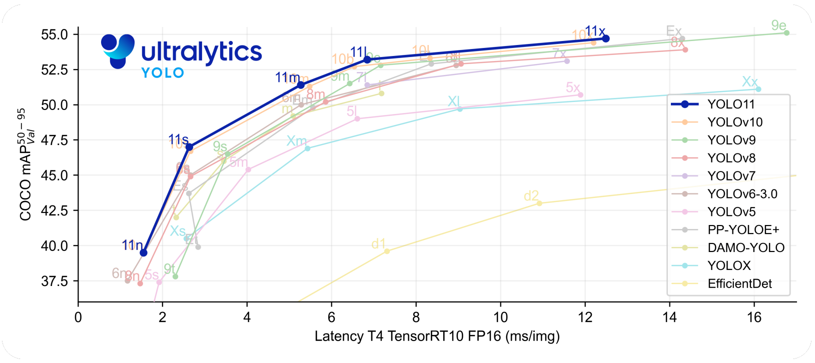


Figure 5.1: mAP@50-95 Comparison for all YOLO Models.

5.3 YOLO VERSIONS UTILIZED

We evaluate multiple versions of the YOLO framework, each of which embodies distinct architectural refinements and optimizations to address specific challenges in object detection.

YOLOv5

YOLOv5[37] has garnered attention due to its implementation in PyTorch and its accessibility to the research community. Although not developed by the original authors, YOLOv5 integrates several enhancements such as optimized anchor box generation, improved loss functions, and advanced data augmentation techniques. These modifications have contributed to superior detection performance on a variety of datasets while maintaining a high inference speed, making YOLOv5 a practical choice for real-world applications.

[37] Ultralytics, *YOLOv5: A state-of-the-art real-time object detection system*

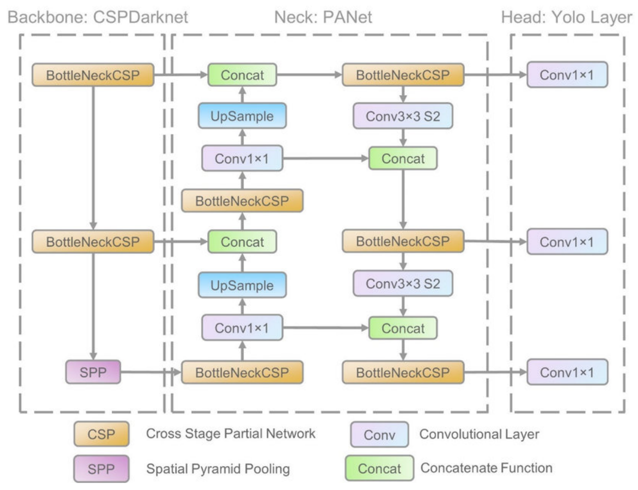


Figure 5.2: The network architecture of Yolov5.

YOLOv8

Building on its predecessors YOLOv8 introduces further refinements to both accuracy and scalability. Key innovations include an enhanced backbone network that improves feature extraction and a more robust feature pyramid network for better handling of scale variations. The design choices in YOLOv8 show improvements in small object detection and adaptability to diverse environments. Such advancements show its potential for deployment in scenarios where precision and speed are equally critical.

YOLOv10

YOLOv10 represents the next evolution of the YOLO framework. This iteration integrates dynamic anchor adjustments and advanced feature fusion techniques to better capture contextual relationships across varying image scales. YOLOv10 also incorporates elements of self-supervised learning to enhance its adaptability in challenging lighting

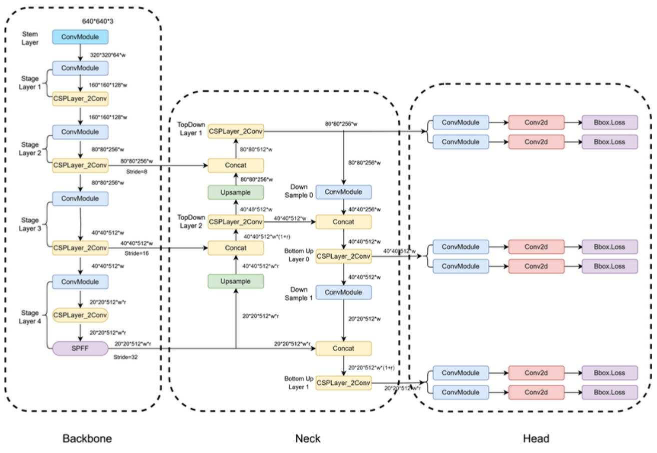


Figure 5.3: The network architecture of Yolov8.

conditions and complex scenes. Its architecture is optimized for edge computing, ensuring energy efficiency without compromising detection performance. The development of YOLOv10 illustrates a commitment to pushing the SOTA of real-time object detection with the broader trend of making sophisticated deep learning models more accessible and efficient in future applications.

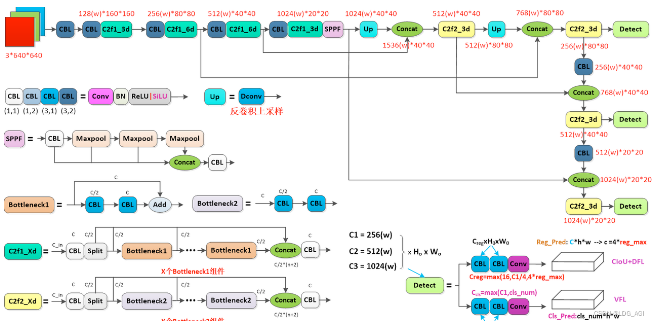


Figure 5.4: The network architecture of Yolov10.

5.4 FINETUNING AND BASELINES

To balance complexity and accuracy, the M-size checkpoint is selected for each model's weights. Each model undergoes fine-tuning on two training datasets, the real dataset and an augmented dataset enriched with synthetic data. The fine-tuning process spans 50 epochs, after which the best checkpoint from each dataset and base architecture is saved for further use.

As a result, each model architecture produces two variants: one fine-tuned on the real dataset and another fine-tuned on the augmented dataset, leading to a total of six models.

Following training, the models are systematically evaluated using a test dataset. The evaluation process consists of comparing model predictions against ground truth labels and computing key performance metrics. These metrics include:

- *Precision:* Measures the proportion of correctly predicted positive instances among all instances classified as positive. A higher precision indicates fewer false positives.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- *Recall:* Represents the proportion of correctly predicted positive instances among all actual positive instances. A higher recall signifies fewer false negatives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- *F1 Score:* The harmonic mean of precision and recall, providing a balanced measure when both false positives and false negatives are important. It is especially useful when there is an imbalance between positive and negative samples.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- *Mean Average Precision (mAP):* A key metric in object detection, mAP evaluates the precision-recall curve across multiple thresholds. It is computed as the mean of the Average Precision (AP) over different Intersection over Union (IoU) thresholds, with values reported at 0.5, 0.75, and 0.95. A higher mAP score indicates better overall detection performance.

$$\text{mAP} = \frac{1}{n} \sum_{i=1}^{n} \text{AP}_i \text{ where } n \text{ is the number of classes}$$

Performances

The evaluation results highlight the performance differences between models trained on real data (Training Dataset 1) and those trained on augmented datasets (Training Dataset 2).

YOLO5_Real achieves the highest precision (1.0) and mAP@0.5 (0.96), indicating strong detection accuracy with minimal false positives. However, its mAP@0.95 (0.0016) is significantly lower, suggesting reduced performance at stricter localization thresholds. The augmented version, YOLO5_Synth, maintains similar recall and precision but shows a slight decrease in mAP@0.75 (0.4274) and mAP@0.95 (0.0002).

For YOLO10 models, the real dataset variant (YOLO10_Real) achieves a strong F1 score (0.933) and mAP@0.5 (0.9054), but its mAP@0.95 (0.0167) remains low. The synthetic-augmented counterpart (YOLO10_Synth) exhibits comparable performance, with slightly lower mAP@0.75 (0.5078) and mAP@0.95 (0.0033), indicating that augmentation may not significantly enhance fine-grained localization.

Similarly, the YOLO8 models show a trend where augmentation improves F1 score and recall slightly, as seen in YOLO8_Synth (F1 = 0.945, Recall = 0.94) compared to YOLO8_Real (F1 = 0.943, Recall = 0.91). However, YOLO8_Real achieves higher mAP@0.5 (0.9068) than YOLO8_Synth (0.9332), while the latter performs slightly better

Table 5.1: Baseline metrics for all the YOLO models.

Model	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	1.000	0.960	0.980	0.960	0.518	0.007
YOLO5_Synth	0.990	0.960	0.975	0.959	0.427	0.0002
YOLO8_Real	0.978	0.910	0.943	0.907	0.414	0.006
YOLO8_Synth	0.949	0.940	0.945	0.933	0.457	0.009
YOLO10_Real	0.958	0.910	0.933	0.905	0.522	0.017
YOLO10_Synth	0.940	0.940	0.940	0.934	0.508	0.003

at mAP@0.75 (0.4569 vs. 0.4139).

Overall, models trained on real data generally achieve better precision and localization accuracy, while synthetic data augmentation slightly improves recall and F1 score. However, at higher IoU thresholds, performance declines across all models, suggesting that fine localization remains a challenge irrespective of dataset augmentation and model architecture.

Part IV

Evaluating Patches From Literature

6

Patch Selection and Evaluation

6.1 PATCH SELECTION

To conduct a thorough evaluation of adversarial patch effectiveness, we have selected a set of ten patches from existing literature. These patches have been chosen based on their demonstrated impact on model performance in various studies. The selected patches originate from three key research papers, each contributing valuable insights into adversarial patch design and application.

The first five patches (Patches 1-5) were sourced from the paper *Adversarial Patch Camouflage Against Aerial Detection*[1]. This study focuses on generating adversarial patches to evade aerial detection systems, presenting a novel approach to adversarial camouflage techniques. These patches have been specifically designed to disrupt object detection models used in aerial surveillance and reconnaissance.

[1] Adhikari et al., *Adversarial Patch Camouflage against Aerial Detection*

Patches 6 to 9 were taken from the work *Fooling Automated Surveillance Cameras: Adversarial Patches to Attack Person Detection*[36]. This research explores the effectiveness of adversarial patches in deceiving person detection models commonly deployed in security and surveillance applications. The patches proposed in this study have demonstrated a high success rate in fooling state-of-the-art object detection frameworks.

[36] Thys, Ranst, and Goedemé, *Fooling automated surveillance cameras: adversarial patches to attack person detection*

The final patch was selected from the seminal paper *Adversarial Patch* [6]. This work introduced one of the earliest adversarial patch concepts, highlighting the feasibility of localized adversarial attacks that are independent of the scene context. The patch is designed to be universal, meaning it can be applied in a variety of environments to effectively disrupt classification and detection models.

[6] Brown et al., *Adversarial Patch*

All patches were extracted as PNG images directly from their respective papers. Minor modifications were made to upscale them, ensuring compatibility with our experimental setup. No significant alterations were introduced that could affect the transferability oftheir adversarial properties.

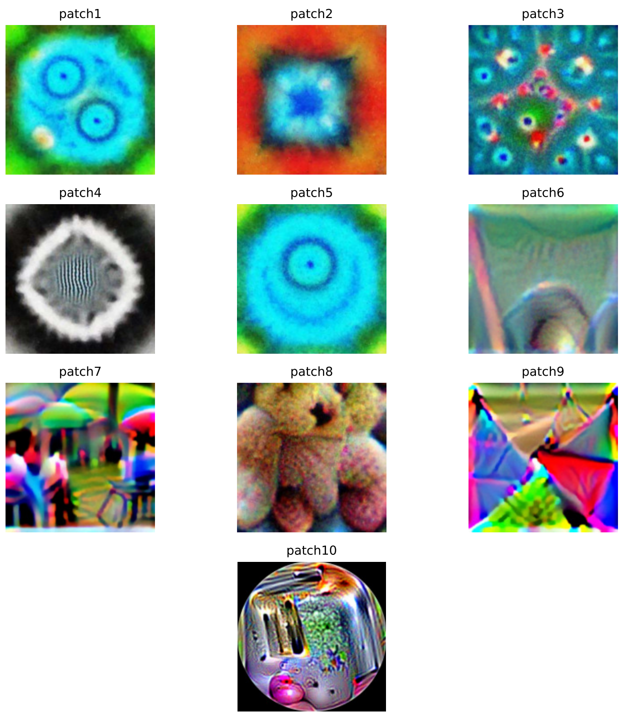


Figure 6.1: Selected adversarial patches from literature.

6.2 EVALUATING PATCHES IMPACT

In object detection, *precision* measures the fraction of detections that are correct, so a decrease in precision under attack indicates the patch is inducing more false positives. *Recall* assesses the fraction of actual objects that are detected; if recall drops, it means the patch is causing the model to miss objects. The *F1 score* is the harmonic mean of precision and recall, providing a single balanced measure of overall detection performance. Meanwhile, *mean Average Precision* (mAP) aggregates performance over multiple Intersection over Union (IoU) thresholds. For example, mAP at 0.5 (a lenient threshold) captures general detection capability, while mAP at 0.75 and mAP at 0.95 require increasingly precise localization, thereby offering a more stringent evaluation of the model's accuracy. Since adversarial patches aim to disrupt both the classification and localization aspects of detection, mAP, especially at higher IoU thresholds, is critical for understanding the true impact of the attack.

To specifically evaluate false negatives, recall is the most appropriate metric. Recall is defined as the ratio of true positives (TP) to the sum of true positives and false negatives (FN):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

This metric directly measures the proportion of actual objects that the model successfully detects. A lower recall indicates a higher number of false negatives relative to the total number of true objects.

Applying The Patches

To ensure a fair evaluation of the selected patches, all ten patches are applied consistently. Each patch is first scaled to $0.5 \times$ min_dim of the corresponding bounding box and then placed at its center. No additional transformations are applied. For each patch, an attacked dataset is generated, and all models are evaluated on each of these attacked datasets.

Performances

To assess the impact of each patch on each model, we evaluate the performance on the attacked dataset and subtract the baseline results obtained from the fine-tuned models. This approach quantifies the effect of each patch on model performance, if a patch leads to a performance drop, the resulting difference will be negative.

To quickly identify the most effective patches, we compute a *patch performance score* as a weighted sum of the differences in performance across all metrics and models. These weights allow us to account for the relative importance of each metric. Since recall is particularly crucial, we evaluate the best patches based on both their overall impact on all metrics and their specific impact on recall.

We identify Patch2, Patch1 and Patch5 as the most effective patches 6.2 based on their overall performance scores. These patches consistently induce the most significant performance drops across all models, with Patch2 outperforming the others. The full metrics for these three best performaing patches are shown in the table below6.2.

However, it is noteworthy that the overall impact of the patches on model performance remains relatively small. Several factors could contribute to this observation:

- *Limited transferability across domains*: The patches were originally designed for specific object detection models and may not generalize effectively to this particular use case.

- *Model robustness*: The fine-tuned models may exhibit greater resilience than expected, allowing them to handle adversarial perturbations with minimal performance degradation.

Patch effectiveness differs across models and datasets. YOLO5 trained on synthetic data consistently shows the highest performance degradation, particularly in precision, recall, and mAP@.5. For example, with

Patch 2, precision drops by 0.1602, recall by 0.1493, and mAP@.5 by 0.2303, indicating a strong negative impact. In contrast, YOLO10 models exhibit minimal performance changes, with YOLO10 trained on real data showing only a 0.0119 increase in recall and a negligible decrease in precision (-0.0119) and mAP@.5 (-0.0276). In some cases, such as Patch 5 applied to YOLO10 trained on real data, there is even a slight improvement in precision (+0.0048), further reinforcing the idea that more recent YOLO versions are more resilient to adversarial perturbations.

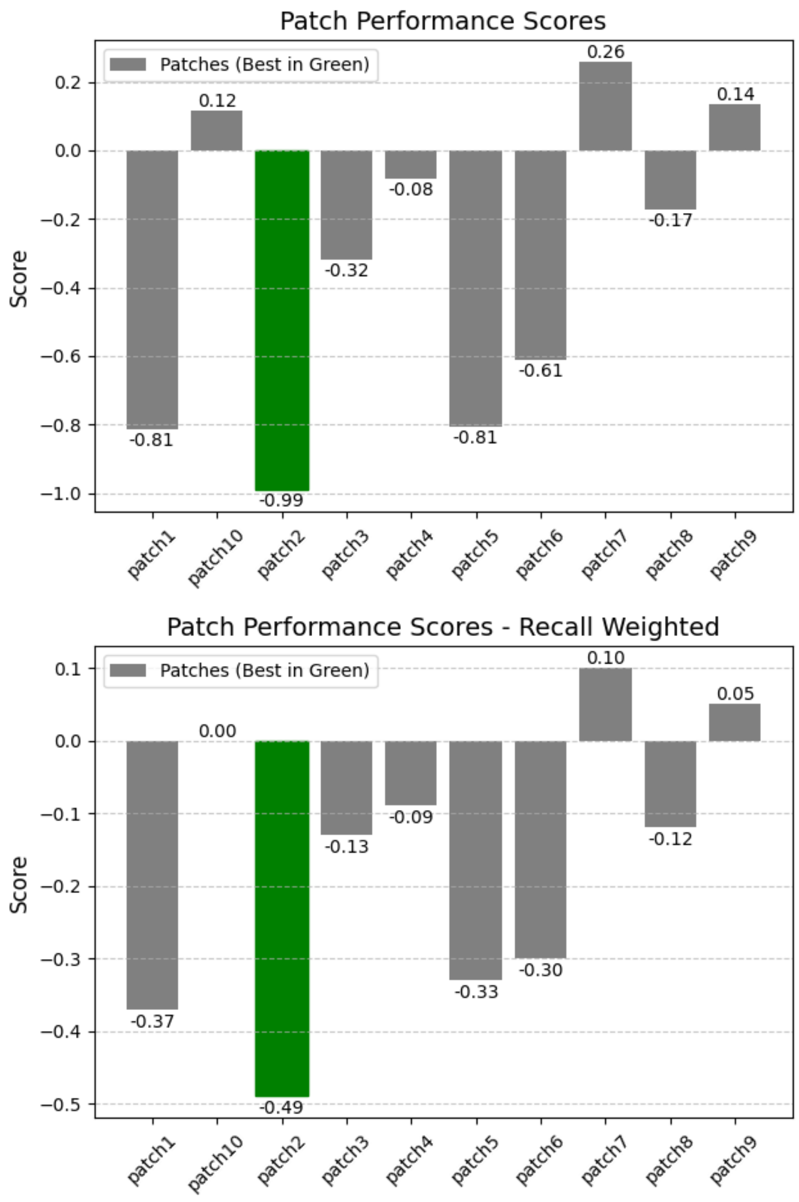


Figure 6.2: Patch scores across all models.

Among the patches, Patch 2 causes the highest performance drop across multiple models, particularly in YOLO5 trained on synthetic data, where mAP@.5 decreases by 0.2303. Patch 1 has a slightly weaker impact but still results in notable performance degradation, such as a 0.1293 decrease in precision and a 0.1705 drop in mAP@.5 for YOLO5 trained on synthetic data. Patch 5 exhibits the least impact overall, with YOLO10 models showing little to no change, as seen in YOLO10 real data where mAP@.5 only decreases by 0.0025.

Precision tends to decrease consistently across all models, whereas recall sometimes remains stable or even improves. For example, YOLO8 trained on synthetic data shows an increase of 0.0186 in recall when Patch 2 is applied, and a similar effect is observed for Patch 1, where recall increases by 0.0182. This suggests that while the patches may cause more false positives, leading to lower precision, the models still maintain their ability to detect objects.

PATCH 2	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.052509	-0.032609	-0.070000	-0.074696	0.117228	-0.001473
YOLO5_Synth	-0.160186	-0.149265	-0.170000	-0.230253	-0.016904	0.000178
YOLO8_Real	-0.051701	-0.002304	-0.090000	-0.095011	-0.031268	0.004091
YOLO8_Synth	-0.006579	0.018590	-0.030000	-0.023295	0.018915	0.002053
YOLO10_Real	-0.011867	0.009138	-0.030000	-0.027617	-0.007515	-0.006364
YOLO10_Synth	-0.055789	-0.006667	-0.100000	-0.098464	-0.111558	0.000333
PATCH 1	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.050299	-0.061224	-0.040000	-0.051017	-0.048724	-0.001367
YOLO5_Synth	-0.129258	-0.117350	-0.140000	-0.170542	0.015342	-0.000222
YOLO8_Real	-0.032952	-0.012202	-0.050000	-0.051064	-0.002642	0.004091
YOLO8_Synth	-0.012081	0.018247	-0.040000	-0.036088	0.061427	0.001488
YOLO10_Real	-0.000691	0.009847	-0.010000	-0.012685	-0.066897	-0.003333
YOLO10_Synth	-0.049948	-0.005934	-0.090000	-0.088470	-0.029436	-0.000833
PATCH 5	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.036002	-0.031579	-0.040000	-0.044841	0.008245	-0.001600
YOLO5_Synth	-0.142132	-0.144330	-0.140000	-0.185656	-0.011165	-0.000222
YOLO8_Real	-0.052953	-0.044429	-0.060000	-0.062895	0.016403	0.004408
YOLO8_Synth	-0.022444	0.007494	-0.050000	-0.045091	0.072184	-0.003661
YOLO10_Real	0.004811	0.010190	0.000000	-0.002487	0.009138	-0.006667
YOLO10_Synth	-0.021633	-0.002500	-0.040000	-0.041126	0.025485	-0.000556

Table 6.1: Impact of the three best performing patches.

7

Patches As Pattern

A particularly compelling variant of patch attacks involves tiling or replicating a single adversarial patch in a grid-like pattern across the entire image. By “multiplying” the patch, this technique ensures that every sub-region of the image is affected, which leads to several advantages.

- *Full spatial coverage*: The repeated pattern ensures that the adversarial signal is present throughout the image, making the attack less sensitive to the precise placement of a single patch. This redundancy means that even if parts of the image are cropped or occluded, many copies of the patch remain intact.

- *Cumulative Effect*: The cumulative effect of multiple overlapping adversarial cues can overwhelm local defenses and force misclassification even under severe physical transformations, such as changes in viewing angle, lighting, or partial occlusion. Studies like AdvTexture[16] have highlighted that introducing adversarial patterns in a distributed manner not only increases the attack's robustness but also enhances its transferability.

[16] Hu et al., *Adversarial Texture for Fooling Person Detectors in the Physical World*

In real-world applications applying a patterned design is often more practical than affixing a single, discrete square patch. A continuous or tiled pattern can be printed on flexible materials like fabric or decals, which naturally conform to the irregular shapes of physical objects. This flexibility means that precise alignment or exact positioning is less critical, reducing installation complexity and potential errors during application. Patterned designs can be integrated more subtly into everyday objects, helping the adversarial patch blend with its surroundings and remain less conspicuous to both human observers and automated detection systems.

7.1 GENERATING AND APPLYING THE PATTERN

Starting from a single adversarial patch, the pattern is generated by replicating it in a grid of size n.

Simply overlaying the pattern onto the image, as done with standard patches, is insufficient. To better simulate real-world scenarios it must conform to the object's shape and surface characteristics.

To achieve this, we first generate a binary mask of the target object using SAM by Meta[19], based on the original image and its corresponding label. This mask ensures that the pattern is applied exclusively to the object, preventing interference with the background.

[19] Kirillov et al., *Segment Anything*

However, directly applying the pattern in this manner would make it appear as a rigid sticker, lacking realism. To enhance the effect and make the pattern blend naturally, it must retain the object's shading and edges.

This is accomplished by extracting a grayscale cutout of the object using the generated mask. The transparency of this cutout is then scaled according to its brightness, where brighter regions become more transparent and darker regions remain more opaque. By overlaying this processed cutout onto the pattern, the pattern is effectively distorted to match the object's shape, preserving natural lighting variations and surface details.

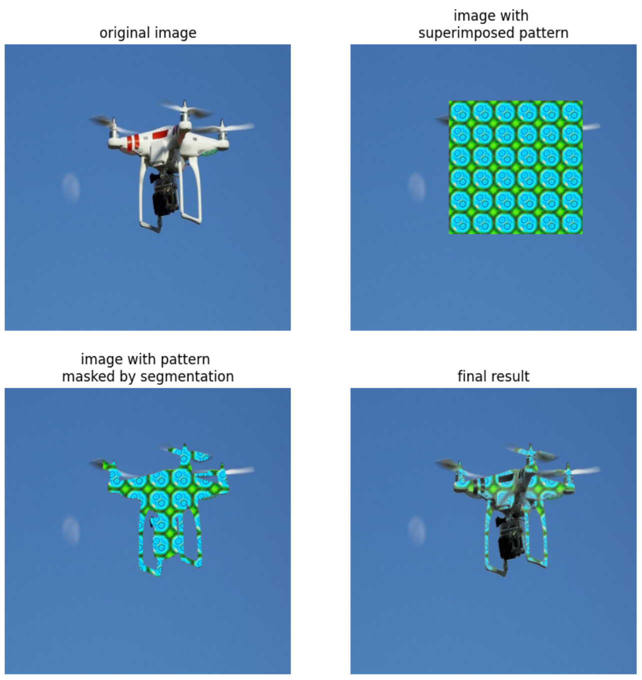

original image
image with superimposed pattern
image with pattern masked by segmentation
final result

Figure 7.1: Applying the adversarial pattern as a texture on a 3D object.

7.2 PERFORMANCES

The three best patches from the previous section were selected for the pattern attack. The patches were applied to the same set of images used in the previous section, and the results were evaluated using the same metrics.

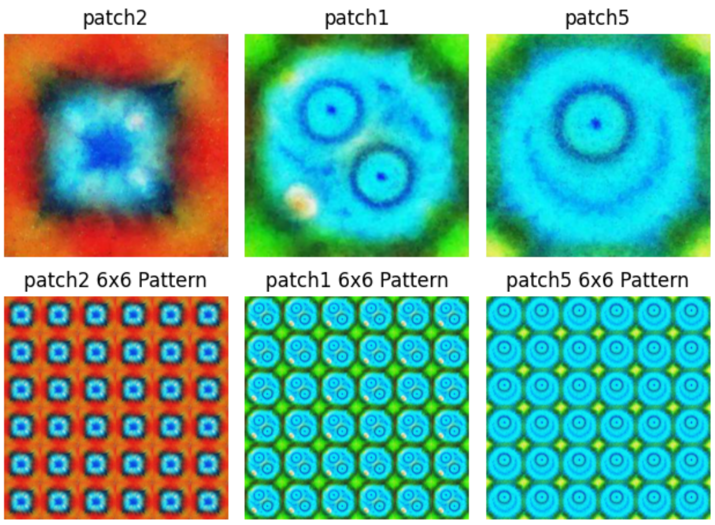


Figure 7.2: Patterns from the best three patches.

Table 7.2 presents the impact of the three best-performing adversarial patches when applied as a pattern across different YOLO models. The results indicate that the effect of the pattern varies significantly depending on the model, the type of training data (real vs. synthetic), and the specific patch used.

In general, most metrics exhibit negative or marginal improvements, suggesting that applying the patch as a pattern is less effective at reducing detection performance compared to isolated adversarial patches. Interestingly, while some configurations, such as YOLO8_Real on Patch 2, show small positive gains in precision and recall, these are inconsistent across different models. The mAP values, particularly at higher IoU thresholds (mAP@.75 and mAP@.95), indicate that the pattern's ability to degrade object detection diminishes as stricter localization criteria are applied.

PATCH 2	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	0.005333	0.020000	-0.010101	0.019293	0.169608	-0.000666
YOLO5_Synth	-0.015856	-0.030000	-0.000329	-0.030452	0.090718	0.000135
YOLO8_Real	0.026383	0.040000	0.011089	0.040057	0.116600	-0.000421
YOLO8_Synth	0.019743	0.010000	0.029886	0.016295	0.001542	-0.007974
YOLO10_Real	-0.017544	-0.040000	0.008772	-0.036466	-0.015058	-0.009301
YOLO10_Synth	-0.001856	-0.030000	0.028085	-0.023591	-0.006502	-0.001963
PATCH 1	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.015489	-0.020000	-0.010526	-0.020526	0.066967	-0.001600
YOLO5_Synth	-0.031030	-0.040000	-0.021270	-0.042254	0.083000	-0.000222
YOLO8_Real	-0.004861	0.000000	-0.010410	-0.000435	0.058164	0.004758
YOLO8_Synth	-0.011390	-0.030000	0.008400	-0.024362	-0.001359	-0.007571
YOLO10_Real	-0.007018	-0.030000	0.019883	-0.026110	-0.005760	-0.016333
YOLO10_Synth	-0.012165	-0.040000	0.017447	-0.034246	-0.025845	0.000303
PATCH 5	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.031675	-0.050000	-0.010870	-0.050435	0.060099	-0.001392
YOLO5_Synth	-0.036475	-0.050000	-0.021606	-0.051169	0.096101	-0.000222
YOLO8_Real	-0.004861	0.000000	-0.010410	0.001093	0.119398	0.004091
YOLO8_Synth	-0.011390	-0.030000	0.008400	-0.024820	0.027388	-0.008565
YOLO10_Real	0.008466	-0.020000	0.042105	-0.015436	-0.005855	-0.014167
YOLO10_Synth	-0.008063	-0.050000	0.038022	-0.043811	-0.056043	-0.002333

Table 7.1: Impact of the three best performing patches applied as pattern.

Overall, the results suggest that while adversarial patterns do impact detection performance, their effect is less pronounced than single-patch attacks, likely due to blending with object textures and model robustness to distributed perturbations.

To obtain a quick overview of the pattern's impact a pattern score is calculated in the same way as the previous section:

Table 7.2: Scores of the best three patches applied as pattern.

PATCH	*Overall Score*	*Recall Weighted*
Patch 1	-0.24	-0.16
Patch 2	0.05	-0.03
Patch 5	-0.24	-0.20

The same trend observed in the previous section is evident here: while applying adversarial patches as a pattern does result in an overall negative impact on detection performance, the magnitude of this effect remains small.

The primary reason for this reduced impact is the blending of the pattern with the object's texture compunded with the already low performance of the patches themselves. Unlike isolated patches, the pattern is less immediately recognizable as an adversarial element to the classifier, as it becomes transformed by the object's shape and shading.

Notably, Patch 2 performs significantly worse when applied as a pattern compared to its effectiveness as a standalone patch. This could be attributed to its bright red color, which, when applied to images of flying drones, increases the visual contrast against the background, making the objects more distinguishable rather than less.

7.3 3EYES PATCH

All three of the best-performing patches exhibit a concentric pattern with a central eye-like structure. Patch 1 appears to be an evolved version of Patch 5, incorporating an additional eye-like structure in the center. This recurring pattern suggests that the eye-like structure plays a significant role in deceiving the object detection model.

To further investigate this hypothesis, a new patch, referred to as the *3eyes* patch, has been manually designed. This patch features three concentric circles, each containing a smaller circle at its center, closely resembling the structure observed in the top-performing patches. The aim is to evaluate whether this specific pattern has a stronger adversarial effect on detection performance.

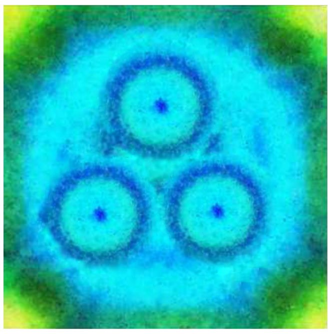


Figure 7.3: The 3eyes patch.

The same analysis was conducted on the 3eyes patch, evaluating its performance both as a single patch and as a repeated pattern.

3eyes Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.010753	-0.026224	-0.040000	-0.043226	-0.020578	0.001733
YOLO5_Synth	-0.126533	-0.133594	-0.140000	-0.192176	-0.034636	-0.000222
YOLO8_Real	-0.045910	-0.064698	-0.080000	-0.085150	0.017072	0.004091
YOLO8_Synth	0.007952	-0.016889	-0.040000	-0.037383	0.047313	0.003065
YOLO10_Real	0.000000	0.000000	0.000000	0.002695	-0.005773	-0.002952
YOLO10_Synth	-0.014468	-0.043093	-0.070000	-0.067850	-0.063308	-0.000714
3eyes Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.010989	-0.037183	-0.060000	-0.060110	0.050102	-0.000356
YOLO5_Synth	-0.000680	-0.032211	-0.060000	-0.060933	0.131950	-0.000222
YOLO8_Real	0.000229	0.005448	0.010000	0.009824	0.104492	0.005854
YOLO8_Synth	0.005561	-0.045253	-0.090000	-0.083416	0.012729	-0.008400
YOLO10_Real	0.030611	-0.013547	-0.050000	-0.045436	-0.056307	-0.016667
YOLO10_Synth	0.036744	-0.036774	-0.100000	-0.093709	-0.055245	-0.001019

Table 7.3: Impact of the 3eyes patch applied as a patch and pattern.

The 3eyes patch, when applied as a single patch, has a higher negative impact on most YOLO models than Patches 1, 2, and 5. Specifically, YOLO5_Synth (-0.1265 Precision, -0.1336 Recall, -0.1922 mAP@.5) and YOLO8_Real (-0.0459 Precision, -0.0647 Recall, -0.0851 mAP@.5) show a greater decrease in performance compared to the best-performing patches.

3eyes	*Overall Score*	*Recall Weighted*
as Patch	-0.84	-0.37
as Pattern	-0.45	-0.35

Table 7.4: Scores of the best three patches applied as pattern.

As a pattern its overall negative effect is lower than when used as a patch, similar to the trend seen with Patches 1, 2, and 5 but still obtaining better results than them. The 3eyes patch's performance trend as a pattern is consistent with the other patches, indicating that its specific pattern design is the primary factor influencing the adversarial effect.

The 3eyes patch's overall impact is more pronounced than the best-performing patches, but still remains relatively small, suggesting that the eye structure's effectiveness is not enought to overcome the issues discussed for the patches from literature.

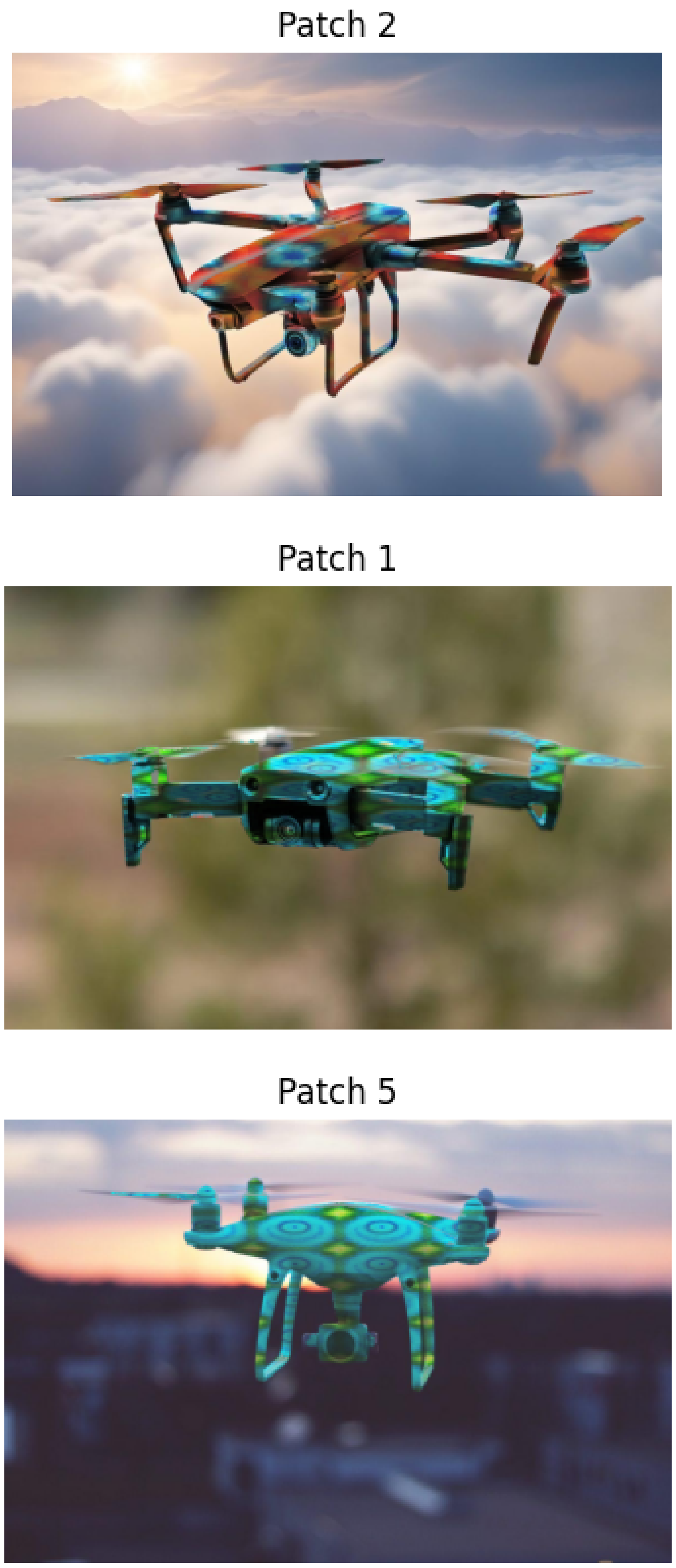


Figure 7.4: Sample images with patterns applied.

Part V
White-Box Patch Generation

8

White-Box Patch Generation

8.1 INTRODUCTION

Initial investigations, detailed in ADD PART REF, evaluated the efficacy of adversarial patches sourced from existing literature against our target drone detection models. These experiments consistently demonstrated significant limitations in transferability; patches effective against one model architecture or dataset in the original paper exhibited markedly reduced performance when applied to the specific models under study in this research.

To achieve effective adversarial examples tailored for this specific case, the best approach is a white-box attack paradigm. This approach leverages privileged information derived from the target model's internal structure.

White-box attacks fundamentally assume comprehensive knowledge of the target neural network, including its architecture, learned parameters (weights and biases), and the computational graph. This level of access is important as it permits the use of powerful gradient-based optimization techniques. The core principle involves calculating the gradient of a chosen objective function (representing the attack's goal) with respect to the input pixel values, specifically, the pixels constituting the adversarial patch. This gradient vector indicates the direction of steepest ascent for the objective function; in essence, it reveals how infinitesimal changes to each patch pixel value will influence the model's output (e.g., its confidence in detecting a drone).

By iteratively adjusting the patch pixels in the direction dictated by the gradient we can systematically guide the patch's appearance towards a state that maximally disrupts the model's intended function. This programmatic, gradient-driven modification allows for the precise crafting of patches designed to induce specific types of errors in the target model's predictions. Furthermore, developing the patch ad hoc provides granular control over its characteristics, enabling the formulation of precise optimization objectives aligned with the specific desired adversarial outcome.

8.2 OBJECTIVES AND THE LOSS FUNCTION

The primary objective of the adversarial patch in this context is detection evasion: to manipulate the input image such that the model fails to detect the presence of a drone. This goal is translated into a quantifiable loss function centered on the model's detection confidence. For each potential drone detection within the image that overlaps with the patch's location, the optimization process seeks to minimize the associated confidence score provided by the model for the 'drone' class. Iteratively updating the patch based on the gradient of this confidence-lowering loss effectively trains the patch to present visual features that the model associates with non-drone objects or background clutter, suppressing the drone detections.

Only minimizing confidence might not suffice in all scenarios. A model might still produce a detection bounding box, albeit with reduced confidence. To enhance the patch's disruptive effect a secondary objective targeting bounding box precision is introduced. This is implemented by incorporating a loss component based on the Intersection over Union (IoU) metric. The optimization is configured to minimize the IoU between the predicted bounding box for a drone and its corresponding ground truth bounding box. This encourages the patch not only to reduce detection certainty but also to actively skew the model's spatial localization, potentially causing any residual detections to be inaccurately positioned and thus less useful or easier to filter out in downstream processing.

Practical Constrains

Optimizing exclusively for the adversarial objectives (confidence reduction and IoU degradation) can often lead to patches characterized by high-frequency noise and visually chaotic patterns. While potentially effective in simulation, such patches present significant challenges for physical realization and deployment. They can be difficult to reproduce accurately using standard printing hardware and may contain jarring visual artefacts.

To promote the generation of physically plausible and "printer-friendly" patches, a regularization term based on Total Variation (TV) is incorporated into the overall loss function. The TV loss penalizes abrupt changes in colour and intensity between adjacent pixels within the patch. This not only helps with printability but can also incidentally improve robustness to minor physical imperfections or viewing condition variations.

A final consideration comes from the intended operational use case: minimizing the patch's visibility. A patch that is highly effective adversarially but starkly contrasts with the drone's surface or the typical operational environment (e.g., sky, foliage, urban background) may increase the drone's detectability by human observers or non-targeted sensor systems.

To address this, the patch generation process is constrained to utilize a predefined, limited colour palette. This constraint is enforced by adding a "palette loss" term to the optimization objective. This loss component measures the distance (e.g., Euclidean distance in RGB or LAB colour space) between each pixel's colour in the generated patch and the closest colour available in the predefined palette. By minimizing this palette loss alongside the other objective terms, the optimization process yields patches whose colours adhere strictly to the specified set, resulting in patterns that integrates adversarial properties with principles of camouflage.

8.3 THE LOSS EQUATION

- Let B be the batch size.
- Let $P \in \mathbb{R}^{C\times H\times W}$ denote the adversarial patch tensor, where C is the number of color channels, H is the height, and W is the width.
- Let R_i be the set of detection results for the i-th image in the batch ($i = 1, \dots, B$). Each detection $d \in R_i$ has a predicted class label $l(d)$, a confidence score $c(d)$, and a bounding box b_d. We assume class label 0 represents the target class (drone).

Confidence Loss ($\mathcal{L}_{conf}$):

This loss aims to minimize the sum of confidence scores for all detections belonging to the target class (class 0) across the entire batch.
$\mathcal{L}_{\text{conf}} = \sum_{i=1}^{B} \sum_{d\in R_i, l(d)=0} c(d)$

- The inner summation $\sum_{d\in R_i, l(d)=0} c(d)$ calculates the total confidence of all class 0 detections for a single image i. If no class 0 detections are found in image i, this sum is implicitly 0.
- The outer summation $\sum_{i=1}^{B}$ sums these confidence totals across all images in the batch.

Total Variation Loss ($\mathcal{L}_{TV}$):

This loss encourages spatial smoothness in the patch by penalizing large differences between adjacent pixel values. It is calculated as the sum of the mean generalized L_p norms (controlled by 'power' p) of the horizontal and vertical pixel differences, with a small epsilon (ϵ) for numerical stability.
$\mathcal{L}_{\text{TV}} = \frac{1}{C(H-1)W} \sum_{c=1}^{C} \sum_{h=1}^{H-1} \sum_{w=1}^{W} \left((P_{c,h+1,w} - P_{c,h,w})^2 + \epsilon\right)^{p/2} + \frac{1}{CH(W-1)} \sum_{c=1}^{C} \sum_{h=1}^{H} \sum_{w=1}^{W-1} \left((P_{c,h,w+1} - P_{c,h,w})^2 + \epsilon\right)^{p/2}$

- $P_{c,h,w}$ is the value of the patch pixel at channel c, height h, and width w.

- The first term calculates the average powered difference vertically, and the second term calculates it horizontally.
- p corresponds to the 'tv_power' parameter in the code, controlling the strength of the smoothing (e.g., $p = 2$ relates to an L2-like penalty on the gradient magnitude).
- ϵ corresponds to 'tv_epsilon', preventing issues when differences are zero, especially for $p < 2$.

Color Restriction Loss ($\mathcal{L}_{palette}$):

This loss encourages the patch pixels to adopt colors from a predefined target palette $\mathcal{T}$. Given the main target of this study, camouflaging flying drones, the palette is chosen to be shades of white, blue, and black. It calculates the average minimum Euclidean distance (L_2 norm) between each patch pixel's color and the closest color in the target palette.

- Let $p_{hw} \in \mathbb{R}^3$ be the color vector of the patch pixel at spatial location (h, w) (assuming $C = 3$).
- Let $\mathcal{T} = \{t_1, t_2, \ldots, t_K\}$ be the set of K target colors, where each $t_k \in \mathbb{R}^3$.

  $\mathcal{L}_{\text{palette}} = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} \min_{k=1,\ldots,K} \|p_{hw} - t_k\|_2$
- $\|p_{hw} - t_k\|_2$ calculates the Euclidean distance between the pixel color p_{hw} and a target palette color t_k.
- $\min_{k=1,\ldots,K}$ finds the smallest distance from the pixel color to any color in the palette $\mathcal{T}$.
- The outer summation and division by HW average these minimum distances over all pixels in the patch.

IoU Loss ($\mathcal{L}_{IoU}$):

This loss aims to minimize the spatial overlap (Intersection over Union) between the predicted bounding boxes for the target class and the corresponding ground truth bounding box. The loss is formulated as 1 minus the *maximum* IoU found for each image, averaged over the batch. Maximizing this loss term encourages low IoU.

- Let $D_{i,0} = \{b_d \mid d \in R_i, l(d) = 0\}$ be the set of predicted bounding boxes for class 0 in image i.
- Let g_i be the ground truth bounding box for the target object in image i.
- Let $\text{IoU}(b_d, g_i)$ denote the Intersection over Union between predicted box b_d and ground truth box g_i.

We define $\max \emptyset = 0$.

$\mathcal{L}_{\text{IoU}} = \frac{1}{B} \sum_{i=1}^{B} (1 - \max_{b_d \in D_{i,0}} \text{IoU}(b_d, g_i))$

- $\max_{b_d \in D_{i,0}} \mathrm{IoU}(b_d, g_i)$ finds the highest IoU score between any predicted class 0 box and the ground truth box for image i. If no class 0 boxes are predicted ($D_{i,0}$ is empty), the maximum is 0.

- $(1 - \max \mathrm{IoU})$ converts the maximization of IoU (good detection) into a minimization objective for the loss. A perfect overlap (IoU=1) results in zero loss from this term for that image, while zero overlap (IoU=0, or no detection) results in a loss of 1.

- The sum and division by B average this per-image loss across the batch.

8.4 EXTRACTING THE GRADIENT

The generation of adversarial patches using white-box attack strategies is predicated on the ability to compute the gradient of a chosen loss function with respect to the input patch pixels. This gradient information guides the iterative optimization process, allowing the patch to be progressively modified to achieve the attacker's objective, typically fooling or degrading the performance of the target object detection model 2.3. Consequently, a prerequisite for implementing such attacks against models like those provided by the Ultralytics framework is obtaining gradient flow from the model's output back to its input.

Input Formulation and Model Output Structure

The input to the models is typically composed of a base image tensor onto which the candidate adversarial patch tensor is superimposed. The placement of this patch is important: for targeted attacks aiming to make a specific object undetectable or misclassified, the patch is usually positioned centrally within the object's ground-truth bounding box coordinates.

The output structure of Ultralytics YOLO models (including versions v5, v8, and v10) during inference is standard. The primary prediction output is encapsulated within a list of *Results* objects. Each *Results* object corresponds to an input image and contains various attributes pertaining to the detection process. For object detection tasks, the most pertinent information resides within the *boxes* attribute, which is an instance of the *Boxes* class.

The *Boxes* class provides several methods and properties to access detection results in different formats.For the purpose of crafting adversarial patches aimed at degrading detection performance, two properties within the *Boxes* object are of primary interest:

- ***conf***: This tensor contains the confidence scores associated with each predicted bounding box. Object detectors typically employ a confidence threshold during post-processing to filter out low-confidence detections. By accessing the gradient of the model's predicted confidence score (*conf*) with respect to the input patch, we can optimize the patch to specifically minimize this score for the target object, potentially causing it to fall below the detection threshold and thus become "invisible" to the model.

- ***xywhn***: This tensor provides the bounding box coordinates in a normalized $[x_{center}, y_{center}, width, height]$ format. Accessing the gradient for these predicted coordinates allows the patch optimization process to manipulate the predicted box location and size. A common attack objective here is to maximize the spatial discrepancy between the predicted bounding box and the ground-truth bounding box, often quantified by minimizing the Intersection over Union (IoU). Modifying the patch based on the *xywhn* gradient can therefore directly degrade the localization accuracy of the detector.

Enabling Gradient Flow in Ultralytics

By default, the Ultralytics library is optimized for efficient inference, not for facilitating gradient-based analysis or attacks. To achieve high throughput and simplify common usage scenarios, gradient computation is explicitly disabled during the forward pass of the models at inference time. This is primarily enforced through the use of a Python decorator, *@smart_inference_mode()*, applied to various prediction and post-processing functions within the library's codebase. This decorator effectively wraps the decorated function within a *torch.no_grad()* or equivalent context, preventing PyTorch's autograd engine from tracking operations and computing gradients.

Therefore, obtaining the necessary gradients for a white-box attack requires modifications to the underlying Ultralytics package source code. This involves several targeted adjustments [13]:

[13] GitHub, *Ultralytics Pull n.8645: Avoid in-place operations when disabling the smart_inference_mode to compute the gradients.*

1. **Disabling Inference Mode Decorators:** The most fundamental step is to locate and disable the *@smart_inference_mode()* decorators in the relevant execution paths. This typically involves commenting out or removing the decorator line preceding the function definitions responsible for model inference and output processing. Key locations often include prediction methods within the specific model classes (e.g., YOLO detection models).

2. **Addressing Gradient-Incompatible Operations:** Since the standard inference pipeline assumes gradients are disabled, some internal functions may utilize PyTorch operations that are incompatible with gradient tracking, most notably in-place tensor modifications. When gradients are enabled, such operations can raise errors or silently lead to incorrect gradient calculations. Therefore, specific functions must be inspected and potentially refactored to avoid these in-place operations. Two functions required adjustment:

   - The *postprocess* function, located within *ultralytics/models/yolo/detect/predict.py* (or similar paths depending on the exact model type), often performs filtering and coordinate adjustments that may involve in-place operations.

- The *clip_boxes* function, found in *ultralytics/utils/ops.py*, which ensures bounding box coordinates remain within image boundaries, might also use in-place clipping.

3. **YOLOv10 Specific Adjustment:** While the above modifications are generally sufficient for enabling gradient flow in YOLOv5 and YOLOv8 architectures, the more recent YOLOv10 model introduces an additional consideration. Within the definition of its detection head module, specifically in the file *ultralytics/nn/modules/head.py* (or a similarly named file depending on the exact implementation version), there exists code that manually detaches parts of the computation graph using the *.detach()* method on tensors involved in the prediction output. This explicit detachment prevents gradients from flowing backward through that point, even if *@smart_inference_mode()* is disabled. To obtain gradients for YOLOv10 outputs, this manual detachment call must also be located and removed or commented out.

9

Results

9.1 INTRODUCTION

This section presents the empirical results obtained from evaluating the adversarial patches generated using the white-box attack methodology detailed before. The primary objective of these experiments was to assess the efficacy of gradient-guided patches in degrading the performance of the detection models.

A core aspect of the experiment involved generating distinct adversarial patches tailored individually for each of the three target models.During the patch generation/optimization phase, the procedure was standardized as follows:

- **Patch Initialization and Dimensions:** Each adversarial patch was instantiated as a square canvas of 100×100 pixels. The initial pixel values for every patch were randomly sampled.

- **Patch Placement during Optimization:** For training purposes, the 100×100 patch tensor was superimposed directly onto the center coordinates of the ground-truth bounding box associated with the target object within each training image instance.

- **Optimization Parameters:** The patches were evolved iteratively over a fixed duration of 150 epochs. A batch size of 16 images was used for processing training data and calculating gradient updates. The optimization utilized an initial learning rate set to 0.5.

To simulate a more realistic application scenario the optimized 100×100 pixels patches were tested both as single patch like during the optimization process but also as a 3×3pattern, applied to the test images according to the methodology previously described in Section 7.1.

Each of the three individually generated patches was subsequently tested against *all three* models. This allows to measure not only how effective a patch is against the specific model architecture it was trained on but also its ability to degrade the performance of the other related architectures.

9.2 YOLOV5 PATCH

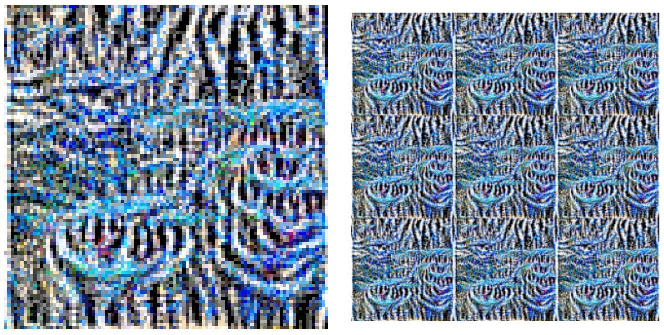


Figure 9.1: The Best Patch from the YOLOv5 patch generation and the corresponding Pattern.

The adversarial patch and pattern generated using YOLOv5 show varied effectiveness. The single patch has its most significant negative impact on its target model YOLOv5, causing a drastic drop in Recall (-0.57) and mAP@.5 (-0.626), while having a much smaller effect on YOLOv8 (-0.01 Recall, -0.005 mAP@.5) and YOLOv10 (-0.07 Recall, -0.072 mAP@.5). Conversely, applying the attack as a pattern resulted in more consistent degradation across all models, significantly reducing Recall and mAP@.5 for YOLOv5 (-0.30, -0.30), YOLOv8 (-0.26, -0.26), and YOLOv10 (-0.25, -0.25).

Table 9.1: Impact of the patch and pattern generated with YOLOv5 on the three models.

Single Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	0.152174	-0.570000	-0.445345	-0.625994	-0.396867	-0.001600
YOLO8_Real	0.020366	-0.010000	0.004167	-0.005436	-0.163356	-0.006359
YOLO10_Real	-0.001750	-0.070000	-0.039779	-0.072049	-0.073428	-0.005909
Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.029412	-0.300000	-0.193878	-0.301820	-0.054454	0.008941
YOLO8_Real	0.006354	-0.260000	-0.159873	-0.257879	-0.099362	0.004091
YOLO10_Real	0.012693	-0.250000	-0.147619	-0.247412	-0.091641	-0.016667

Table 9.2: Scores for the patch and pattern generated with YOLOv5.

YOLOv5 Patch	*Overall Score*	*Recall Weighted*
as Patch	-1.24	-0.65
as Pattern	-1.32	-0.81

The summary scores reflect this, with the pattern achieving slightly stronger overall negative scores (-1.32 Overall, -0.81 Recall Weighted) compared to the single patch (-1.24 Overall, -0.65 Recall Weighted), indicating the pattern's broader effectiveness, particularly in suppressing detections (Recall).

9.3 YOLOV8 PATCH

The patch generated targeting YOLOv8 demonstrates strong effectiveness against its source model when applied as a single patch, severely

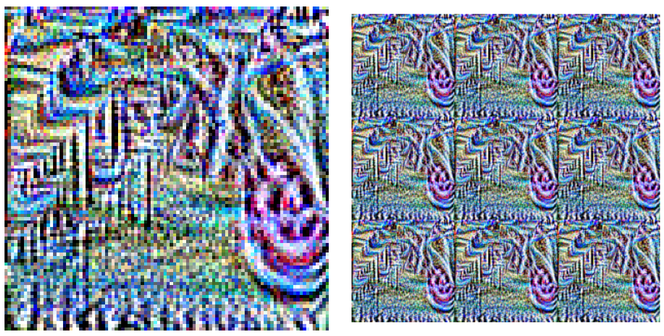


Figure 9.2: The Best Patch from the YOLOv8 patch generation and the corresponding Pattern.

degrading YOLOv8's performance (Recall -0.59, mAP@.5 -0.616). However, its transferability as a single patch to YOLOv5 (Recall -0.06) and especially YOLOv10 (Recall -0.02) is limited. When applied as a pattern, the attack becomes highly effective across all models, causing substantial drops in Recall and mAP@.5 for YOLOv5 (-0.38, -0.38), YOLOv8 (-0.30, -0.30), and YOLOv10 (-0.26, -0.25).

Single Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	0.000000	-0.060000	-0.032223	-0.060000	-0.106569	0.000464
YOLO8_Real	-0.216590	-0.590000	-0.492301	-0.615698	-0.313358	-0.002576
YOLO10_Real	0.020127	-0.020000	-0.001396	-0.015547	-0.179815	-0.006368
Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.049180	-0.380000	-0.259095	-0.380506	-0.132636	-0.001421
YOLO8_Real	-0.010241	-0.300000	-0.194539	-0.302089	-0.088441	-0.005909
YOLO10_Real	0.012255	-0.260000	-0.154890	-0.257093	-0.093885	-0.016667

Table 9.3: Impact of the patch and pattern generated with YOLOv8 on the three models.

YOLOv8 Patch	*Overall Score*	*Recall Weighted*
as Patch	-1.39	-0.67
as Pattern	-1.59	-0.94

Table 9.4: Scores for the patch and pattern generated with YOLOv8.

Although the single patch was more damaging specifically to YOLOv8, the pattern shows much better generalization. This is confirmed by the summary scores, where the pattern yields significantly more negative results (-1.59 Overall, -0.94 Recall Weighted) than the single patch (-1.39 Overall, -0.67 Recall Weighted).

9.4 YOLOV10 PATCH

The adversarial attacks generated using YOLOv10 show a stark difference between the single patch and the pattern application. The single patch is moderately effective against its source model, YOLOv10 (Recall -0.26, mAP@.5 -0.27), but exhibits very poor transferability, causing only minor performance drops for YOLOv5 (Recall -0.05) and YOLOv8 (Recall -0.05). In contrast, the pattern derived from this attack is highly effective and transfers extremely well, causing significant degradation across YOLOv5 (Recall -0.39, mAP@.5 -0.39), YOLOv8 (Recall -0.31, mAP@.5 -0.30), and YOLOv10 (Recall -0.34, mAP@.5 -0.33). Notably, the pattern is more effective against the source model (YOLOv10) than the single patch itself.

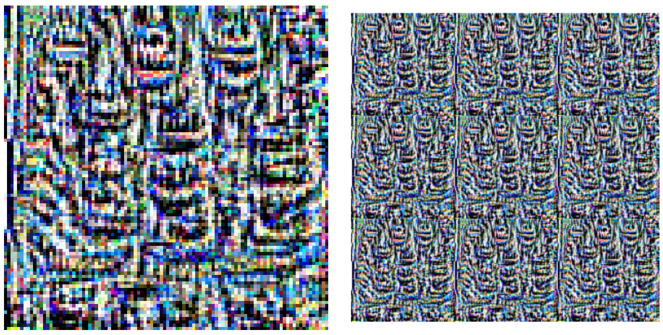


Figure 9.3: The Best Patch from the YOLOv10 patch generation and the corresponding Pattern.

Table 9.5: Impact of the patch and pattern generated with YOLOv10 on the three models.

Single Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.031915	-0.050000	-0.041448	-0.052407	0.068883	0.003400
YOLO8_Real	0.010011	-0.050000	-0.023219	-0.048427	-0.076899	-0.005717
YOLO10_Real	-0.055117	-0.260000	-0.177519	-0.270355	-0.316996	-0.006667
Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.033898	-0.390000	-0.262611	-0.391884	-0.110768	-0.001600
YOLO8_Real	-0.010753	-0.310000	-0.202264	-0.309158	-0.115053	0.004091
YOLO10_Real	-0.023469	-0.340000	-0.225259	-0.337993	-0.138146	-0.016667

The summary scores illustrate this difference: the single patch has a relatively weak impact (-0.68 Overall, -0.36 Recall Weighted), while the pattern achieves a very strong negative impact (-1.59 Overall, -0.94 Recall Weighted), comparable to the most effective pattern generated from YOLOv8.

9.5 MOSAIC PATTERN

The results revealed a degree of transferability for the generated patches across different models, an effect particularly noticeable when these patches were applied in a pattern format. Based on this observation, it can be hypothesized that combining multiple effective patches could yield a more robust and broadly applicable adversarial pattern. To explore this, a *mosaic pattern* was designed by integrating the three most successful patches from the previous section into a 6×6 grid. The idea was to create a single pattern possessing enhanced transferability characteristics, yet retaining potent adversarial capabilities against each individual target model.

The application method for this mosaic pattern remained the same as with prior experiments. The performance of this composite pattern is evalulated against the three original models, with the results presented below.

The results for the mosaic pattern strongly support the hypothesis that combining effective individual patches can create a more robust and broadly effective adversarial attack. This composite pattern demonstrates significant degrading effects across all three target models, achieving substantial reductions in Recall (-0.42 for YOLOv5, -0.35

Table 9.6: Scores for the patch and pattern generated with YOLOv10.

YOLOv10 Patch	*Overall Score*	*Recall Weighted*
as Patch	-0.68	-0.36
as Pattern	-1.59	-0.94

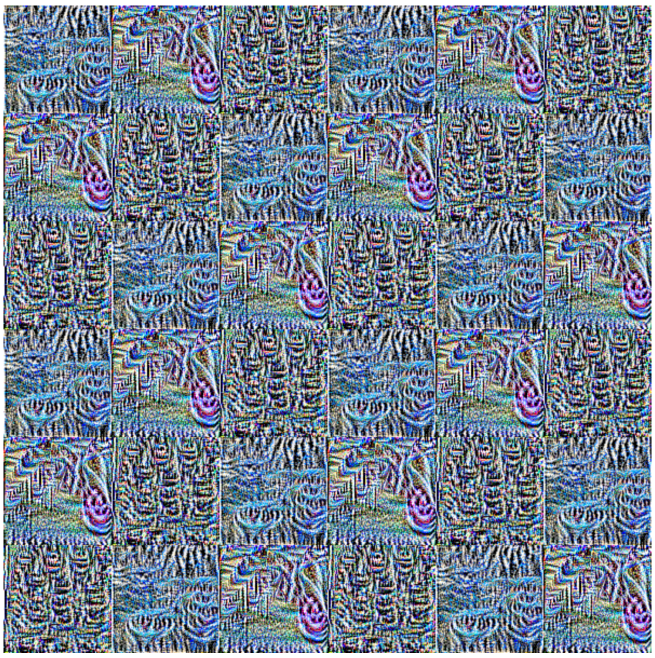


Figure 9.4: Mosaic generated from the three best patches.

Table 9.7: Impact of the mosaic pattern on the three models.

Mosaic Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.052632	-0.420000	-0.291694	-0.425916	-0.143243	-0.001044
YOLO8_Real	0.003962	-0.350000	-0.229629	-0.349274	-0.126706	0.005909
YOLO10_Real	0.024864	-0.340000	-0.211814	-0.335609	-0.159910	-0.016667

for YOLOv8, -0.34 for YOLOv10) and corresponding drops in mAP@.5 (-0.426, -0.349, -0.336 respectively).

Table 9.8: Scores for the mosaic pattern.

Mosaic Scores	*Overall Score*	*Recall Weighted*
	-1.87	-1.11

Compared to the individual patterns generated earlier, the mosaic pattern generally achieves equal or superior performance degradation, particularly showing enhanced impact against YOLOv5. The summary scores further underscore its potency, with an Overall Score of -1.87 and a Recall Weighted score of -1.11, both considerably more negative than those achieved by any single-source pattern. This indicates the mosaic pattern successfully integrates the strengths of its constituent patches, resulting in a highly effective and transferable adversarial attack.

9.6 OTHER PALETTES

The palette used in the previous experiments was specifically chosen to maximize the effectiveness of the generated patches against the target object detection models in the main use, flying drones with a "clear sky" background. However, it is important to explore the impact of different palettes on the performance of the generated patches, both to account for different real world scenarios and to understand if the palette choice has a significant effect on the performance of the generated patches. To do this, patches for all models were generated using palettes of *sand* tones and *green* tones, common in different natural environments. The patches were generated using the same methodology as before and tested in the same way by generating a 3×3 pattern.

Sand Palette

The sand palette was chosen to simulate a hot environment use case, the palette contains the same shades of white and black as the blue palette, but with sand tones.

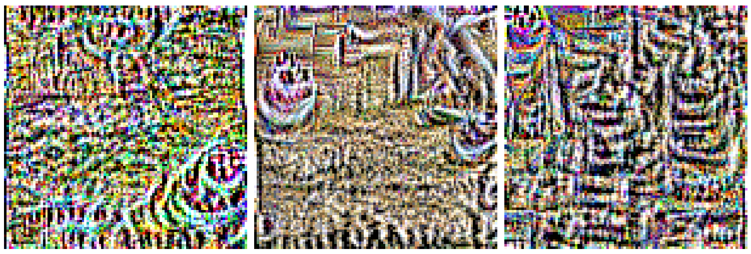


Figure 9.5: Patches for respectively YOLOv5, YOLOv8 and YOLOv10 generated using a sand palette.

Table 9.9: Impact of the sand patches and patterns on the three models.

YOLOv5 Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.102941	-0.350000	-0.253401	-0.377293	-0.331475	-0.000767
YOLO8_Real	-0.036634	-0.100000	-0.072037	-0.108350	-0.131813	-0.005909
YOLO10_Real	0.010190	0.000000	0.004811	-0.002355	-0.165390	-0.001078
YOLOv5 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.060000	-0.490000	-0.352925	-0.493581	-0.142369	-0.001356
YOLO8_Real	0.004264	-0.340000	-0.221486	-0.338542	-0.138745	-0.005492
YOLO10_Real	0.011336	-0.280000	-0.169697	-0.277780	-0.170809	-0.016667
YOLOv8 Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.107143	-0.210000	-0.164374	-0.239481	-0.265638	-0.001600
YOLO8_Real	-0.407066	-0.670000	-0.604977	-0.718097	-0.345049	-0.005909
YOLO10_Real	-0.000905	-0.020000	-0.011054	-0.025578	-0.172092	-0.006667
YOLOv8 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.066667	-0.400000	-0.279592	-0.401701	-0.185381	-0.000171
YOLO8_Real	0.02150	-0.350005	-0.225056	-0.346818	-0.084356	-0.005709
YOLO10_Real	0.028407	-0.190000	-0.100963	-0.186258	-0.047174	-0.016497
YOLOv10 Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.030303	0.000000	-0.014768	-0.003411	0.098968	0.000400
YOLO8_Real	-0.000967	-0.040000	-0.022370	-0.037847	-0.090773	-0.005157
YOLO10_Real	-0.073837	-0.300000	-0.211440	-0.308475	-0.356440	-0.006667
YOLOv10 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.019231	-0.450000	-0.308539	-0.450577	-0.140763	-0.001600
YOLO8_Real	0.005632	-0.290000	-0.182269	-0.286977	-0.126366	0.006266
YOLO10_Real	0.014327	-0.210000	-0.119380	-0.208026	-0.103079	-0.015758

Table 9.10: Scores for the sand patches and patterns.

YOLOv5 Patch	*Overall Score*	*Recall Weighted*
as Patch	-0.90	-0.45
as Pattern	-1.89	-1.11
YOLOv8 Patch	*Overall Score*	*Recall Weighted*
as Patch	-2.19	-0.90
as Pattern	-1.56	-0.94
YOLOv10 Patch	*Overall Score*	*Recall Weighted*
as Patch	-0.69	-0.34
as Pattern	-1.56	-0.95

Green Palette

Like the sand palette above, the green palette was chosen to simulate a natural environment use case, the palette contains the same shades of white and black as the blue palette, but with green tones.

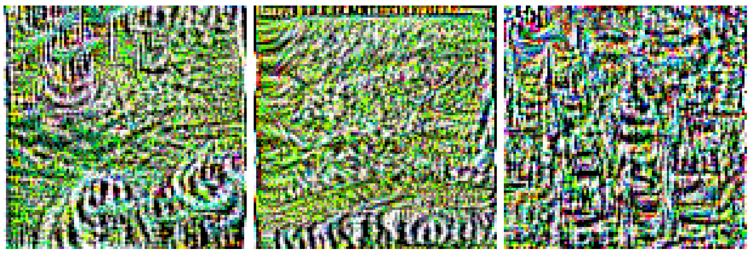


Figure 9.6: Patches for respectively YOLOv5, YOLOv8 and YOLOv10 generated using a green palette.

YOLOv5 Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.107143	-0.460000	-0.338566	-0.498900	-0.374982	-0.001600
YOLO8_Real	-0.015995	-0.140000	-0.087450	-0.139439	-0.172326	-0.005705
YOLO10_Real	0.031353	0.010000	0.020035	0.014134	-0.138273	-0.006259
YOLOv5 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.035714	-0.420000	-0.287284	-0.426460	-0.084298	0.009006
YOLO8_Real	0.005376	-0.300000	-0.189919	-0.297140	-0.067283	0.005202
YOLO10_Real	0.042105	-0.280000	-0.160327	-0.275436	-0.108963	-0.016667
YOLOv8 Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.057971	-0.310000	-0.210361	-0.317008	-0.254568	0.008400
YOLO8_Real	-0.160313	-0.370000	-0.292403	-0.438119	-0.247939	-0.005909
YOLO10_Real	0.019117	-0.060000	-0.024242	-0.055553	-0.175299	-0.002261
YOLOv8 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.067797	-0.410000	-0.287768	-0.415023	-0.094760	-0.001215
YOLO8_Real	-0.010241	-0.300000	-0.194539	-0.299767	-0.108791	0.009785
YOLO10_Real	-0.007895	-0.150000	-0.088889	-0.150847	0.004066	-0.016396
YOLOv10 Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.020619	-0.010000	-0.015125	-0.012904	0.122779	-0.001600
YOLO8_Real	-0.001222	-0.050000	-0.028112	-0.047967	-0.073110	-0.005242
YOLO10_Real	-0.051228	-0.230000	-0.156190	-0.232826	-0.337610	-0.011667
YOLOv10 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.019231	-0.450000	-0.308539	-0.454231	-0.107705	-0.001387
YOLO8_Real	0.005632	-0.290000	-0.182269	-0.287453	-0.112407	0.005267
YOLO10_Real	0.026721	-0.270000	-0.157576	-0.265898	-0.138577	-0.015667

Table 9.11: Impact of the green patches and patterns on the three models.

YOLOv5 Patch	*Overall Score*	*Recall Weighted*
as Patch	-1.09	-0.59
as Pattern	-1.62	-1.00
YOLOv8 Patch	*Overall Score*	*Recall Weighted*
as Patch	-1.47	-0.74
as Pattern	-1.52	-0.86
YOLOv10 Patch	*Overall Score*	*Recall Weighted*
as Patch	-0.56	-0.29
as Pattern	-1.65	-1.01

Table 9.12: Scores for the green patches and patterns.

9.7 WHITE AND RANDOM NOISE NOT-OPTIMIZED PATCHES

[h] The patches generated in the previous sections were specifically designed to maximize the effectiveness of the generated patches against the target object detection models. To understand the impact of this optimization, the same metrics are calculated for completely white patch and a random noise patch.

Even these rudimentary patches exert some influence on the object detection models. This suggests that any form of significant visual occlusion or alteration can, to some extent, disrupt the models' detection capabilities. As expected the degradation tends to be more pronounced when these patches are applied as a repeating pattern rather than a single patch. For instance, the 'White Pattern' on YOLO5_Real incurs a recall drop of -0.270000, considerably larger than the -0.100000 drop from the single 'White Patch'. A similar trend is observable for the random noise patch.

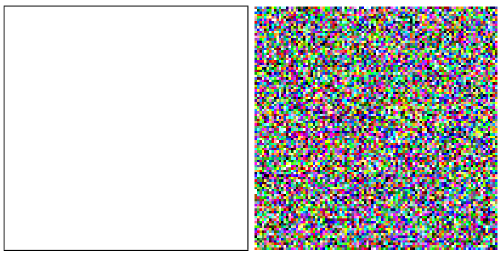


Figure 9.7: White (bordered here) and random noise patches.

Table 9.13: Impact of the white and random noise patches and patterns on the three models.

White Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.011494	-0.100000	-0.059806	-0.103218	0.052851	0.001733
YOLO8_Real	-0.001750	-0.070000	-0.039779	-0.071263	0.015625	-0.004798
YOLO10_Real	0.019883	-0.030000	-0.007018	-0.027122	0.017297	-0.006241
White Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.080000	-0.270000	-0.191020	-0.280043	-0.141189	0.008400
YOLO8_Real	-0.048917	-0.250000	-0.171075	-0.282320	-0.168320	0.005758
YOLO10_Real	0.030057	-0.090000	-0.037158	-0.085557	-0.123999	-0.006322
Noise Patch	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.045455	-0.120000	-0.085975	-0.123764	-0.011320	0.000826
YOLO8_Real	-0.000967	-0.040000	-0.022370	-0.037492	-0.000365	-0.005909
YOLO10_Real	0.042105	0.010000	0.025000	0.014564	-0.025568	-0.001167
Noise Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.046154	-0.340000	-0.228077	-0.343051	-0.106719	-0.000350
YOLO8_Real	0.007013	-0.230000	-0.138271	-0.227398	-0.054740	0.008377
YOLO10_Real	-0.009177	-0.170000	-0.101873	-0.171177	-0.067947	-0.015848

The takeaway from these baseline experiments is the magnitude of their impact relative to the optimized patches. While the white and random noise patches, particularly in their pattern forms, do affect model performance (e.g., the Noise Pattern inducing a recall reduction of -0.340000 on YOLO5_Real), this influence is less pronounced than that achieved by the patches specifically engineered for adversarial efficacy in the preceding sections.

Table 9.14: Scores for the white and noise patches and patterns.

White Patch	*Overall Score*	*Recall Weighted*
as Patch	-0.30	-0.20
as Pattern	-1.10	-0.61
Noise Patch	*Overall Score*	*Recall Weighted*
as Patch	-0.24	-0.15
as Pattern	-1.26	-0.74

This disparity underscores the significance of the optimization process itself, suggesting that that while simple visual disruptions can cause a degree of interference, the substantial degradation observed with optimized patches is not merely a byproduct of occlusion but a direct consequence of their tailored design to exploit model-specific vulnerabilities. Therefore, the effectiveness of the previously detailed adversarial patches can be attributed to their strategic, optimized nature rather than just the introduction of a visual artifact.

10

Improved Patch Generator

10.1 INTRODUCTION: BEYOND SINGLE-PATCH

The preceding investigations into adversarial patches, including the baseline white and noise patches, highlighted the efficacy of optimized perturbations. However, the initial adversarial patch generator (henceforth V1) focused on optimizing a singular, monolithic patch. This patch was then scaled and applied to a central region within the target's bounding box during the optimization loop, and the tested both as optimized and applied as a pattern. While effective to a degree, this approach inherently limits the contextual information available during optimization and may not fully represent how such adversarial textures might be deployed or perceived in more realistic scenarios, particularly if an entire object surface were to be treated.

To address these limitations and explore more robust and potentially more transferable adversarial designs, an enhanced patch generation methodology (V2) was developed. The V2 generator shifts the paradigm from optimizing an isolated patch to optimizing a *base patch element* that is subsequently applied as a repeating, masked *pattern* across the target object's visible surface during the training process itself, in the same way as the pattern was created and tested in previous chapters. This chapter details the conceptual and methodological advancements in the V2 generator.

10.2 INTEGRATED PATTERN OPTIMIZATION

The core innovation in the V2 generator lies in its approach to how the adversarial perturbation is conceived and optimized. Instead of learning a single, fixed-size patch intended for one-off placement, V2 learns a smaller, tileable base patch. During each training iteration, this base patch is:

1. **Tiled:** Repeated according to a configurable `patch_tiling_factor` (e.g., 3x3 repetitions) to form a larger pattern.
2. **Resized and Applied to Bounding Box:** This tiled pattern is then resized to fit the dimensions of the target object's bounding box in the training image.
3. **Masked by Segmentation:** Crucially, the application of this resized pattern is constrained by a per-instance segmentation mask. This ensures the adversarial pattern is only applied to pixels corresponding to the actual target object within its bounding box, rather than indiscriminately filling the entire rectangular region.

This "optimize-as-pattern" strategy is fundamentally different. The V1 generator optimized a patch in isolation, which was then *tested* either as a single instance or by manually tiling it post-optimization. In contrast, V2 *optimizes the base element with full awareness of its destiny as a tiled, masked pattern*. The gradients used to update the base patch are derived from the loss computed after the entire tiling, resizing, masking, and application process.

10.3 KEY METHODOLOGICAL ENHANCEMENTS IN GENERATOR V2

The shift described above necessitates several significant methodological changes within the `AdversarialPatchGenerator` (V2) from the previous version, with the steps being taken and re-adapeted from testing the patches as pattern in previous sections.

Pattern Tiling and Contextual Application

The V1 generator's `apply_patch_to_image` method scaled and placed a single patch. The V2 generator introduces a more sophisticated `apply_pattern_to_image_torch` method, that first tiles the current learnable `self.patch` (the base element) creating a larger, repetitive pattern from the base element. This complete pattern is then resized to the dimensions of the target's bounding box.

Integration of Segmentation Masks

A pivotal enhancement is the mandatory use of segmentation masks (provided via `mask_path`), which are loaded as binary arrays. After the tiled pattern is resized to the bounding box dimensions, it is applied to the image *selectively*, only where the corresponding pixels in the (similarly resized) segmentation mask are true. This ensures the adversarial pattern conforms to the object's silhouette within its bounding box, providing a more realistic application scenario. The optimization, therefore, learns a base patch that is effective when distributed across potentially irregular and non-contiguous visible parts of an object.

Adaptive Blending Mechanism

To potentially improve the stealthiness or natural integration of the pattern, V2 introduces an optional adaptive blending mechanism, controlled by `blend_alpha_strength`. If this strength is greater than zero, the applied pattern is blended with a grayscale representation of the original image content within the masked region. The blending alpha is modulated by the local brightness of the original image pixels under the mask (normalized within the masked region's brightness range). For darker underlying regions, the pattern might be blended more transparently or with an inverted relationship compared to lighter regions, controlled by `is_dark` logic. This allows the pattern to subtly adapt its appearance based on the underlying texture it is covering, potentially making it less conspicuous. The `blend_alpha_strength` parameter directly scales the intensity of this blending effect. If set to 0, this blending step is bypassed, and the pattern is applied opaquely (within the mask).

Refined Gradient Handling

While V1 exclusively used sign-based gradient updates (`torch.sign()`), V2 introduces more nuanced gradient control via the `gradient_clipping_mode` parameter in the `patch_trainer`. This can be set to:

- `'sign'`: Replicating the V1 behavior.
- `'norm'`: Applying gradient norm clipping (`torch.nn.utils.clip_grad_norm_`) with a `max_grad_norm`.

This flexibility can help stabilize training, prevent excessively large updates to the patch, and fine-tune the optimization dynamics, especially when dealing with the complex loss landscape arising from patterned application.

Adjustments to Loss Components

Minor refinements were also made to some loss component calculations:

- **Total Variation (TV) Loss:** The formulation was slightly adjusted, though its fundamental purpose of promoting patch smoothness remains.
- **Color Restriction Loss:** Instead of averaging the minimum distance to target colors for all pixels, V2 calculates the mean of the `top_k_distances` (top 10% of distances). This focuses the penalty on the most errant pixels, potentially allowing more flexibility in color usage while still constraining extreme deviations.

10.4 IMPLICATIONS AND EXPECTED BENEFITS OF THE V2 APPROACH

The transition to optimizing a masked, tiled pattern directly within the training loop is anticipated to yield several benefits:

1. **Enhanced Robustness:** By optimizing the base patch element in the context of its tiled application across the entire object (as defined by the mask), the resulting pattern is expected to be more robust to variations in object scale, partial visibility, and intra-class morphological differences. The generator learns features that are effective when spatially distributed.

2. **Improved Realism and Applicability:** The use of segmentation masks ensures the adversarial pattern is applied only to the target object, mimicking a more realistic "skin" or "texture" attack rather than an arbitrary overlay. This also means the optimization is not penalized for, nor benefits from, affecting background pixels.

3. **Potentially Increased Transferability:** Patterns learned with awareness of broader object surface characteristics, rather than a small central region, might generalize better to unseen instances or even different (but related) object detectors.

4. **Greater Control over Pattern Characteristics:** Parameters like `patch_tiling_factor` and `blend_alpha_strength` offer finer control over the visual characteristics and application style of the generated adversarial pattern.

10.5 IMPROVED GENERATOR RESULTS

The primary objective of this research is the development of adversarial patterns effective against drone detection. The V2 generator, detailed in the preceding chapter, was specifically designed to optimize a base element for tiled, masked application—mimicking a real-world scenario like camouflaging a drone's surface. Consequently the following results focus exclusively on the performance of these patterns, as single patch metrics are less relevant to this advanced methodology.

A central finding evident in the subsequent sections, is the markedly improved efficacy of V2-generated patterns compared to those from V1.

Blue Patterns

The blue patterns developed with the V2 generator demonstrate a significant degradation in detector performance, as evidenced in Table 10.5, all three patterns achieve substantial reductions in recall and mAP@.5 across all tested YOLO versions.The YOLOv5-optimized blue pattern reduces YOLOv5's recall by 0.630000 and mAP@.5 by 0.630000, while still managing a 0.400000 recall reduction against

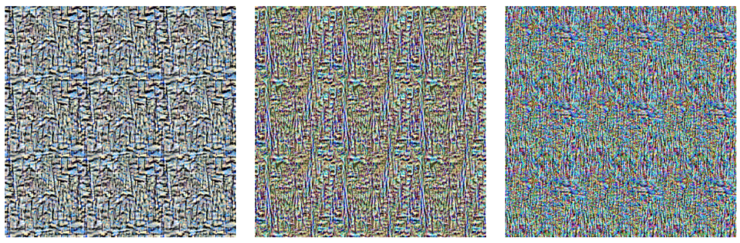


Figure 10.1: Blue patterns generated with the V2 generator. From left to right: YOLOv5, YOLOv8, and YOLOv10 patterns.

YOLOv8. The overall scores, consistently around -2.27 to -1.33, confirm their potent adversarial nature when applied as a full surface pattern, showcasing the effect of the V2 "optimize-as-pattern" strategy.

Table 10.1: Impact of the blue patterns generated with the V2 generator.

YOLOv5 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.029412	-0.630000	-0.487055	-0.630000	-0.289354	-0.001600
YOLO8_Real	0.021505	-0.400000	-0.267508	-0.396818	-0.228713	-0.005692
YOLO10_Real	0.010359	-0.300000	-0.184867	-0.300149	-0.154658	-0.016667
YOLOv8 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.040816	-0.490000	-0.348719	-0.492542	-0.180275	-0.001243
YOLO8_Real	0.021505	-0.540000	-0.402859	-0.536818	-0.214426	-0.005242
YOLO10_Real	-0.004049	-0.290000	-0.181818	-0.289369	-0.128199	-0.016667
YOLOv10 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.057692	-0.470000	-0.334855	-0.470781	-0.163792	0.009453
YOLO8_Real	0.021505	-0.360000	-0.233328	-0.356818	-0.111202	-0.002273
YOLO10_Real	0.022497	-0.410000	-0.271082	-0.406809	-0.209612	-0.016667

Table 10.2: Scores for the blue patterns generated with the V2 generator.

YOLOv5 Pattern	*Overall Score*	*Recall Weighted*
	-2.27	-1.33
YOLOv8 Pattern	*Overall Score*	*Recall Weighted*
	-2.09	-1.24
YOLOv10 Pattern	*Overall Score*	*Recall Weighted*
	-2.27	-1.33

Green Patterns

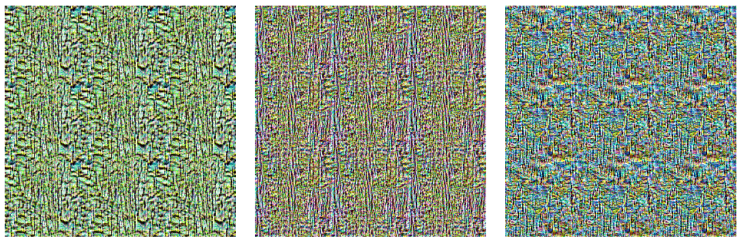


Figure 10.2: Green patterns generated with the V2 generator. From left to right: YOLOv5, YOLOv8, and YOLOv10 patterns.

Similarly, the green V2 patterns exhibit robust adversarial capabilities across the board. The pattern optimized for YOLOv8, in particular, stands out with an Overall Score of -2.38 and a Recall Weighted score of -1.39, the strongest in this color group. It achieves impressive recall reductions of -0.520000 against YOLOv5, -0.510000 against itself (YOLOv8), and -0.360000 against YOLOv10.

Sand Patterns

The sand-colored patterns, generated using the V2 methodology, also show substantial efficacy in degrading detector performance. Both

Table 10.3: Impact of the green patterns generated with the V2 generator.

YOLOv5 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.028571	-0.620000	-0.475888	-0.622000	-0.259342	-0.001600
YOLO8_Real	0.021505	-0.420000	-0.285287	-0.416818	-0.172065	0.004091
YOLO10_Real	0.011802	-0.270000	-0.162249	-0.269458	-0.070770	-0.016667
YOLOv8 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.022222	-0.520000	-0.372695	-0.521333	-0.171768	-0.001600
YOLO8_Real	0.021505	-0.510000	-0.371577	-0.506818	-0.184497	-0.005909
YOLO10_Real	-0.009619	-0.360000	-0.237131	-0.357935	-0.132235	-0.016481
YOLOv10 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.039216	-0.470000	-0.330585	-0.471200	-0.165027	-0.000171
YOLO8_Real	0.021505	-0.300000	-0.185241	-0.296818	-0.108224	0.006948
YOLO10_Real	0.004369	-0.400000	-0.266667	-0.398353	-0.186624	-0.016667

Table 10.4: Scores for the green patterns generated with the V2 generator.

YOLOv5 Pattern	*Overall Score*	*Recall Weighted*
	-2.22	-1.31
YOLOv8 Pattern	*Overall Score*	*Recall Weighted*
	-2.38	-1.39
YOLOv10 Pattern	*Overall Score*	*Recall Weighted*
	-1.96	-1.17

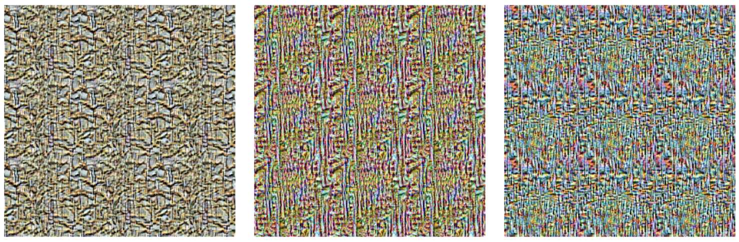


Figure 10.3: Sand patterns generated with the V2 generator. From left to right: YOLOv5, YOLOv8, and YOLOv10 patterns.

the YOLOv5-optimized and YOLOv10-optimized sand patterns yield strong Overall Scores of -2.27. The YOLOv5 pattern, for example, reduces recall by a significant 0.630000 on its target model and maintains a notable 0.400000 recall reduction on YOLOv8, indicating effective cross-model impact. These outcomes, consistent with the other color sets, further validate the V2 generator's improved approach.

Table 10.5: Impact of the sand patterns generated with the V2 generator.

YOLOv5 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.029412	-0.630000	-0.487055	-0.630000	-0.289354	-0.001600
YOLO8_Real	0.021505	-0.400000	-0.267508	-0.396818	-0.228713	-0.005692
YOLO10_Real	0.010359	-0.300000	-0.184867	-0.300149	-0.154658	-0.016667
YOLOv8 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.020833	-0.490000	-0.344457	-0.490208	-0.169154	-0.001267
YOLO8_Real	-0.028495	-0.530000	-0.400148	-0.536218	-0.219466	-0.005909
YOLO10_Real	-0.010526	-0.370000	-0.245435	-0.367015	-0.156351	-0.016667
YOLOv10 Pattern	*Precision*	*Recall*	*F1*	*mAP@.5*	*mAP@.75*	*mAP@.95*
YOLO5_Real	-0.043478	-0.520000	-0.376852	-0.520662	-0.206392	0.009983
YOLO8_Real	-0.023949	-0.490000	-0.359672	-0.489182	-0.206782	-0.005242
YOLO10_Real	-0.007895	-0.530000	-0.390476	-0.529936	-0.266141	-0.016667

Table 10.6: Scores for the sand patterns generated with the V2 generator.

YOLOv5 Pattern	*Overall Score*	*Recall Weighted*
	-2.25	-1.32
YOLOv8 Pattern	*Overall Score*	*Recall Weighted*
	-2.44	-1.39
YOLOv10 Pattern	*Overall Score*	*Recall Weighted*
	-2.74	-1.54

Part VI
Final Results

11
Attack Success Rate Analysis

11.1 INTRODUCTION: DEFINING AND UNDERSTANDING ATTACK SUCCESS RATE

While metrics like mean Average Precision (mAP), precision, and recall provide a comprehensive overview of how adversarial patterns degrade the overall performance of object detection models, the Attack Success Rate (ASR) offers a more direct measure of an attack's efficacy from the perspective of an adversary. ASR quantifies the percentage of instances where an attack successfully fools the detector regarding a specific target object. In the context of object detection, a "successful" attack can mean either causing the detector to fail to detect the target object entirely (a false negative) or causing the object to be detected with a confidence score below a certain operational threshold.

The confidence score assigned by a detector to its predictions is a crucial element in practical deployments. Operators often set a minimum confidence threshold to filter out low-quality detections and reduce false positives. Therefore, an adversarial attack might be considered successful not only if it makes an object invisible to the detector but also if it significantly erodes the confidence in the detection, pushing it below this operational threshold.

To capture this nuance, ASR is typically evaluated at various confidence thresholds. For instance, ASR@0.75 would measure the percentage of attack instances where the target object is either not detected or detected with a confidence score of less than 0.75. Analyzing ASR across a spectrum of confidence thresholds (e.g., from 0.50 to 0.95) is vital for several reasons:

1. **Practical Relevance:** It reflects how the attack would perform against systems configured with different sensitivity levels. A system requiring high certainty (e.g., a 0.90 threshold) might be more vulnerable than one operating at a lower threshold (e.g., 0.50) if

the attack primarily reduces confidence rather than eliminating detections outright.

2. **Robustness Assessment:** It reveals the attack's strength. An attack maintaining high ASR even at stringent confidence thresholds is demonstrably more potent and robust.

3. **Understanding Attack Mechanism:** It can provide insights into whether the attack primarily causes detections to be missed entirely or if it consistently lowers their confidence scores across the board.

By examining ASR at multiple levels, we gain a more fine-grained understanding of the adversarial patterns' ability to compromise the detector's reliability in real-world scenarios.

11.2 ATTACK SUCCESS RATE RESULTS FOR V2 PATTERNS

To further assess the practical impact of the patterns generated by the V2 methodology, we conducted an Attack Success Rate (ASR) analysis. This investigation focused on the most effective pattern identified for each of the three color palettes—blue, green, and sand—based on their overall performance metrics and recall degradation discussed in Chapter 10 (specifically, the patterns yielding the best overall and recall-weighted scores against their respective target models). For the blue palette the best performing pattern was generated against YOLOv5, the green palette's best pattern was generated against YOLOv8, and the sand palette's best pattern was generated against YOLOv10.

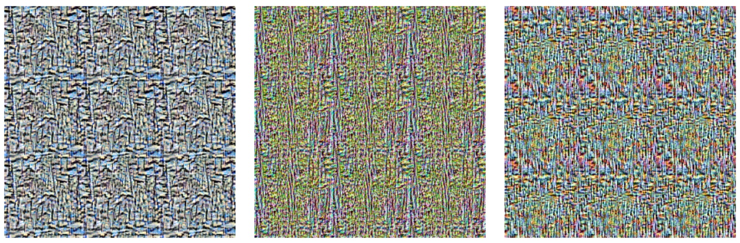


Figure 11.1: Best patterns generated by the V2 generator for each palette. Respectively generated by YOLOv5, YOLOv8, and YOLOv10.

For each selected pattern, the ASR was evaluated against all the yolo models. The success of an attack was defined as an instance where the target object (e.g., a drone) was either completely missed by the detector or was detected with a confidence score below a specified threshold. These confidence thresholds were systematically varied from 0.50 (50%) up to 0.95 (95%), with increments of 0.05 (5%). This range allows for a comprehensive view of how the patterns perform, from moderately confident detections to those requiring very high certainty. The results, presented below, illustrate the percentage of successful attacks at each of these critical confidence junctures.

Table 11.1: Attack Success Rate (%) at Various Confidence Thresholds for Selected V2 Patterns.

Confidence Threshold	**0.50**	**0.55**	**0.60**	**0.65**	**0.70**	**0.75**	**0.80**	**0.85**	**0.90**	**0.95**
Blue Pattern ASR (%)	65.66	67.68	73.74	78.79	85.86	88.89	93.94	98.99	100.00	100.00
Green Pattern ASR (%)	53.12	54.17	60.42	63.58	72.92	80.21	86.46	98.96	100.00	100.00
Sand Pattern ASR (%)	53.06	55.10	59.18	67.35	74.49	84.69	92.86	98.98	100.00	100.00

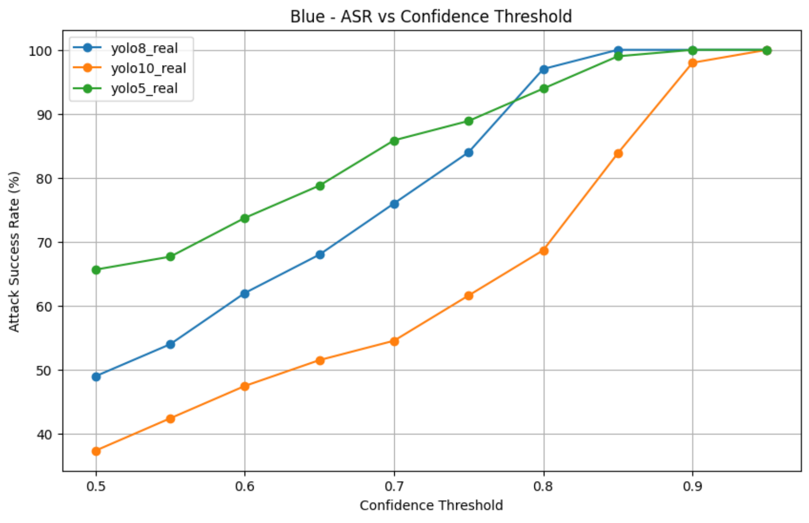


Figure 11.2: Attack Success Rate for the Blue Pattern generated by the V2 against YOLOv5.

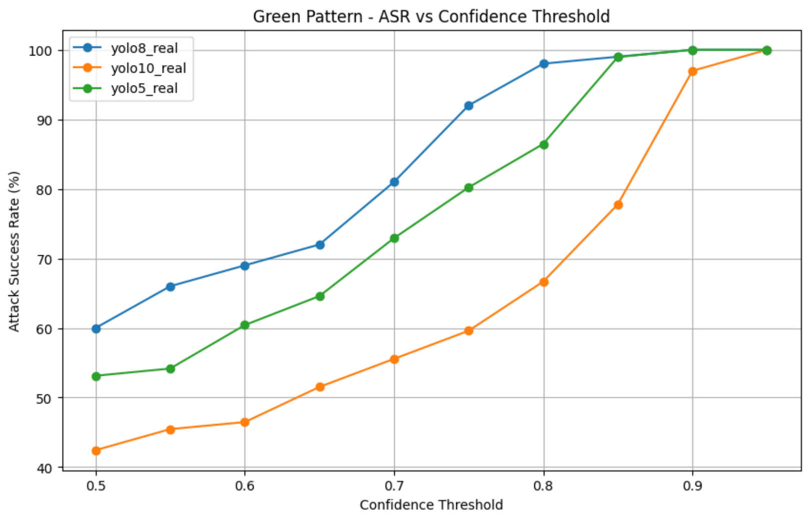


Figure 11.3: Attack Success Rate for the Green Pattern generated by the V2 against YOLOv8.

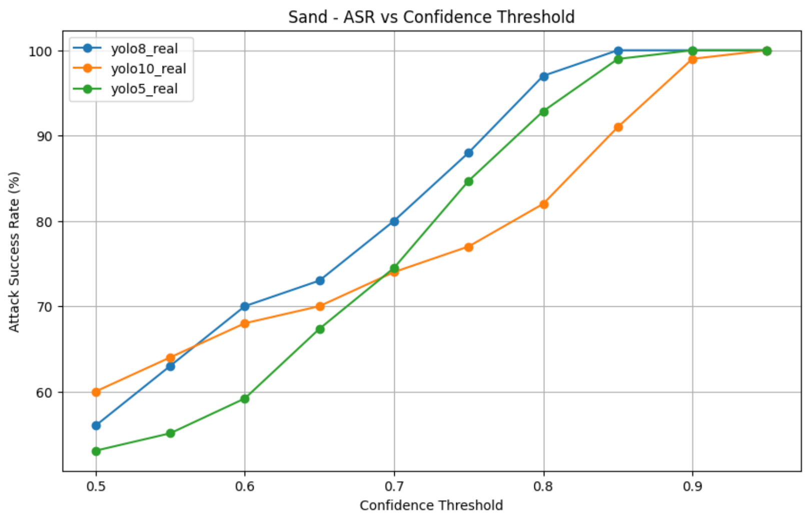


Figure 11.4: Attack Success Rate for the Sand Pattern generated by the V2 against YOLOv10.

The Attack Success Rate (ASR) figures presented in Table 11.2 offer compelling insights into the practical impact of the V2-generated patterns. A clear and consistent trend emerges across all three color palettes: as the detection confidence threshold is raised, the ASR for each pattern steadily increases. This indicates that the patterns are not only causing some detections to be missed entirely but are also significantly eroding the confidence scores of detections that do occur.

Initially, at the lower confidence threshold of 0.50, the Blue pattern demonstrates a notable edge, achieving an ASR of 65.66%. The Green and Sand patterns, while still effective, start slightly lower at 53.12% and 53.06% respectively. As we move towards moderate confidence levels, such as 0.70, all patterns show substantial efficacy, with ASRs climbing to 85.86% for Blue, 72.92% for Green, and 74.49% for Sand.

By the 0.85 mark, all patterns are achieving near-perfect success, with ASRs of 98.99% for Blue, 98.96% for Green, and 98.98% for Sand. Crucially, for any system demanding a detection confidence of 0.90 or 0.95, all three V2 patterns achieve a 100% Attack Success Rate on the testing dataset.

The results strongly suggest that the V2 generator's methodology, which optimizes the patch element in the context of its tiled and masked application, produces adversarial textures capable of defeating detectors under various operational conditions.

12

Live Video Feed Experiment

12.1 MOTIVATION

The preceding chapters have quantitatively established the effectiveness of the V2-generated patterns against static images. Howeve a question remains regarding their performance in a dynamic context, which more closely approximates real-world deployment scenarios. Real-world detection systems do not operate on individual, high-quality images but on continuous video feeds, which are often subject to motion, temporal inconsistencies, etc.

This chapter details an experiment designed to evaluate the adversarial patterns' robustness in a simulated live-stream environment. The objective was not to replicate the full complexity of a physical field test, but to serve as a intermediate step assessing the pattern's ability to disrupt detection when presented as a continuous, streaming video source.

12.2 EXPERIMENT SETUP

The experiment was structured in two main phases: the preparation of an adversarial video stream, and the setup for live inference and visualization.

To create a suitable test video, a clean, high-resolution video of a drone in flight was used as a baseline. The core task was to apply the YOLOv10 optimized blue pattern from the previous chapter across every frame of this video. The process was automated using a combination of custom Python scripts and the FFmpeg multimedia framework.

- Video Deconstruction: The source video was first decomposed into a sequence of individual PNG frames using an FFmpeg command.
- Per-Frame Adversarial Application: A Python script iterated through

each extracted frame. For every frame, it applied the adversarial pattern using the exact same methodology developed for the pattern application. This involved generating a segmentation mask of the drone within the frame and then applying the tiled pattern, blended according to the drone's texture and lighting.

- Video Recomposition: Once all frames were processed, FFmpeg was used again to recombine the sequence of modified frames into a new MP4 video file.

The goal of this phase was to treat the adversarial video as if it were a live feed from an IP camera on a network.

- Local Network Streaming: VLC media player was used to stream the modified video file over the local network using the IP streaming option available in VLC.
- YOLO Inference on the Stream: A key advantage of the Ultralytics YOLO framework is its native support for various input sources, including network streams. The fine-tuned YOLOv10 model was set up to use the IP stream URL as its direct input source, allowing it to perform inference on the video feed in real time.
- Real-Time Visualization: To observe the model's performance live OpenCV was used to retrieve the frames being analyzed by YOLO, and overlay the predicted bounding boxes (or lack thereof) on the video. The result was displayed on-screen, providing immediate visual feedback on the attack's effectiveness.

12.3 QUALITATIVE RESULTS AND LIMITATIONS

Quantifying the performance of a live video experiment on paper is a challenges, as metrics like mAP are not easily calculated on a streaming source and it is impossible to show the full video in a thesis format.

During the live feed, the adversarial pattern demonstrated a significant and consistent ability to disrupt the YOLOv10 detector. The model, which could reliably track the drone in the original, unmodified video, struggled with the adversarial stream. Detections were successfully suppressed for a good part of the stream's duration. On the occasions that a detection did occur, the bounding box was was often assigned a very low confidence score by the model.

These real-time observations support the quantitative results from the patches analysis confirming that the pattern's effectiveness is not limited to isolated images.

While this experiment successfully demonstrated the pattern's efficacy in a simulated live environment, the approach has several important limitations that must be acknowledged:

- Frame-by-Frame Inconsistency: Applying the pattern to each frame individually introduces temporal inconsistencies. Because the drone's position shifts slightly from one frame to the next, the precise alignment of the segmentation mask and the pattern can create a shifting effect on the pattern.

- Impact of Video Compression: Live video streaming, by its nature, relies on compression algorithms to transmit data efficiently. This process inevitably leads to a loss of quality and can introduce compression artifacts.

This live video test serves as a successful proof-of-concept. It confirms the adversarial pattern's robustness beyond static images, while also showing the inherent limitations of a purely digital simulation. The definitive test, which could resolve these limitations, lies in physical application, a key direction for future work.

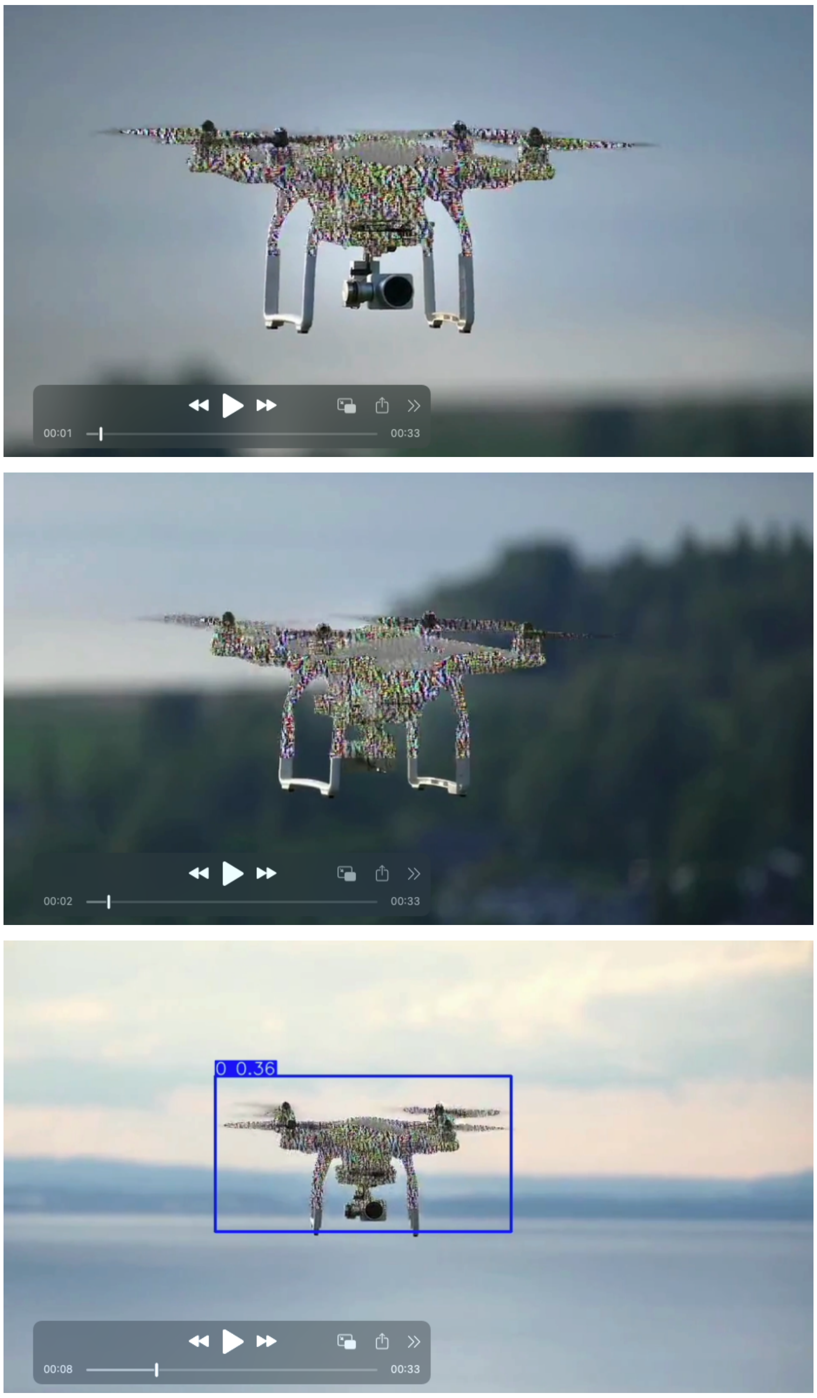


Figure 12.1: Some frames from the live video feed experiment.

Part VII
Epilogue

13

Conclusion

13.1 SUMMARY OF FINDINGS

XXXX
—XXX

This research was focused to explore a critical match of opportunity and vulnerability: the use of deep learning for UAV detection. The investigation began with a foundational premise, that existing adversarial attack methods might compromise these systems, and culminated in the development of a methodology that demonstrates a profound and systematic weakness in state-of-the-art object detectors.

The initial evaluation of patches from literature served as a important baseline revealing that effectiveness is not easily transferred between different models. This shows the need for a custom approach, leading to a white-box generation strategy. The key insight emerged not from the initial generator, but from the analysis of its output: the spatial distribution of an adversarial signal, when applied as a tiled pattern, greatly amplifies its impact and transferability.

This principle became the cornerstone of the V2 generator. By optimizing a base patch element within the full context of a masked, repeating pattern, we developed a method to create adversarial textures that are not just effective, but also simulate a practical, full-surface camouflage. The V2-generated patterns consistently dismantled the detection capabilities of YOLOv5, v8, and v10, achieving a 100% attack success rate against any system configured with a high-confidence threshold.

13.2 IMPLICATIONS OF THE RESEARCH

The findings of this thesis show direct implications for the development and deployment of secure AI systems.

First, this research demonstrates the practical feasibility of creating adversarial camouflage that can systematically evade detection.

By moving from a theorical concern to a practical method of evasion the work highlights a tangible vulnerability in automated surveillance systems reliant on deep learning. This underscores the need for developers and security practitioners in both commercial and defense sectors to consider this attack vector in their system designs and risk assessments.

Second, the primary value of identifying a vulnerability lies in the opportunity to address it. The patterns developed in this thesis can provide a benchmark for developing and validating defensive measures. They offer a more challenging standard than simple noise perturbations and can serve as a tool for creating effective adversarial training regimens. The goal of such training would be to produce the next generation of models that are resilient not only to basic attacks, but to comprehensive, surface-level camouflage.

13.3 LIMITATIONS AND FUTURE WORK

While this thesis achieved its objectives, its conclusions are framed by certain limitations, each of which presents a starting point for future research.

The most significant limitation is the digital nature of our experiments. Though designed to simulate real-world application, the digital realm is a clean room, free from the physical variables that complicate reality. The most logical next step is to transition from simulation to practice: printing these patterns onto physical materials, applying them to operational drones, and evaluating their performance under the unpredictable conditions of real-world flight, with its shifting light, atmospheric distortion, and complex backgrounds.

Furthermore, the focus was confined to the YOLO family of architectures. While this provided a consistent environment for analysis, the broader landscape of object detection includes diverse frameworks like Faster R-CNN or transformer-based models. A notable next step would be to investigate the universality of our V2 patterns. Do they exploit a weakness fundamental to all convolutional neural networks, or is their success tied to specific architectural motifs of YOLO?

13.4 CONCLUDING REMARKS

The rapid integration of artificial intelligence into critical systems represents one of the great technological shifts of our time. It promises unprecedented efficiency and capability, but it also introduces novel and subtle fragilities.

By demonstrating that we can systematically blind the very systems designed to provide sight, we underscore a fundamental principle: our trust in AI must be earned, not assumed. It must be forged through rigorous testing that confronts not only the expected challenges, but also the unexpected and the malicious.

Bibliography

[1] Ajaya Adhikari et al. *Adversarial Patch Camouflage against Aerial Detection*. 2020. arXiv: 2008.13671 [cs.CV]. URL: https://arxiv.org/abs/2008.13671.

[2] George Allison. *Drone sightings reported over British nuclear facilities from 2021 to 2023*. 2024. URL: https://ukdefencejournal.org.uk/drone-sightings-reported-over-british-nuclear-facilities/.

[3] Anish Athalye et al. *Synthesizing Robust Adversarial Examples*. 2018. arXiv: 1707.07397 [cs.CV]. URL: https://arxiv.org/abs/1707.07397.

[4] Marco Barreno et al. *Can machine learning be secure?* New York, NY, USA, 2006. URL: https://doi.org/10.1145/1128817.1128824.

[5] Battista Biggio et al. *Evasion attacks against machine learning at test time*. Springer, 2013.

[6] Tom B. Brown et al. *Adversarial Patch*. 2018. arXiv: 1712.09665 [cs.CV]. URL: https://arxiv.org/abs/1712.09665.

[7] Ido Caspi. *Drones in Defense: Reshaping Modern Warfare and its Economics*. 2024. URL: https://www.globalxetfs.com/drones-in-defense-reshaping-modern-warfare-and-its-economics.

[8] Nandish Chattopadhyay, Atreya Goswami, and Anupam Chattopadhyay. *Adversarial Attacks and Dimensionality in Text Classifiers*. 2024.

[9] Nilesh Dalvi et al. *Adversarial classification*. 2004.

[10] Binyue Deng et al. *Rust-style patch: A physical and naturalistic camouflage attacks on object detector for remote sensing images*. 2023.

[11] Ranjie Duan et al. *Adversarial camouflage: Hiding physical-world attacks with natural styles*. 2020.

[12] Kevin Eykholt et al. *Note on Attacking Object Detectors with Adversarial Stickers*. 2018. arXiv: 1712.08062 [cs.CR]. URL: https://arxiv.org/abs/1712.08062.

[13] Bimo99B9 on GitHub. *Ultralytics Pull n.8645: Avoid in-place operations when disabling the smart_inference_mode to compute the gradients.* URL: https://github.com/ultralytics/ultralytics/pull/8645/commits.

[14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML]. URL: https://arxiv.org/abs/1412.6572.

[15] J. Graham-Cumming. *How to beat an adaptive spam filter*. 2004.

[16] Zhanhao Hu et al. *Adversarial Texture for Fooling Person Detectors in the Physical World*.

[17] Caixin Kang et al. *Diffender: Diffusion-based adversarial defense against patch attacks*. Springer, 2024.

[18] Antti Kariluoto et al. *Quality of Data in Machine Learning*. 2021. arXiv: 2112.09400 [cs.LG]. URL: https://arxiv.org/abs/2112.09400.

[19] Alexander Kirillov et al. *Segment Anything*. 2023.

[20] Zixiao Kong et al. *A Survey on Adversarial Attack in the Age of Artificial Intelligence*. 2021. DOI: https://doi.org/10.1155/2021/4907754. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/4907754. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/4907754.

[21] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. *Adversarial examples in the physical world*. 2017. arXiv: 1607.02533 [cs.CV]. URL: https://arxiv.org/abs/1607.02533.

[22] Chang-Sheng Lin et al. *Real-world adversarial examples involving makeup application*. 2021.

[23] Xin Liu et al. *DPatch: An Adversarial Patch Attack on Object Detectors*. 2019. arXiv: 1806.02299 [cs.CV]. URL: https://arxiv.org/abs/1806.02299.

[24] Mingming Lu et al. *Scale-adaptive adversarial patch attack for remote sensing image aircraft detection*. 2021.

[25] Zihao Lu, Hao Sun, and Yanjie Xu. *Adversarial robustness enhancement of UAV-oriented automatic image recognition based on deep ensemble models*. 2023.

[26] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: 1706.06083 [stat.ML]. URL: https://arxiv.org/abs/1706.06083.

[27] Brando Miranda et al. *Beyond Scale: The Diversity Coefficient as a Data Quality Metric for Variability in Natural Language Data*. 2024. arXiv: 2306.13840 [cs.CL]. URL: https://arxiv.org/abs/2306.13840.

[28] Sedir Mohammed et al. *The Effects of Data Quality on Machine Learning Performance*. 2024. arXiv: 2207.14529 [cs.DB]. URL: https://arxiv.org/abs/2207.14529.

[29] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV]. URL: https://arxiv.org/abs/1506.02640.

[30] Ramprasaath R Selvaraju et al. *Grad-CAM: visual explanations from deep networks via gradient-based localization*. 2020.

[31] Samira Shackle. *The mystery of the Gatwick drone*. 2018. URL: https://www.theguardian.com/uk-news/2020/dec/01/the-mystery-of-the-gatwick-drone.

[32] Thibault Spirlet. *Ukraine says it's taken the top spot in the race to make combat drones*. 2024. URL: https://www.businessinsider.com/ukraine-says-world-largest-producer-tactical-strategic-drones-war-russia-2025-2?utm_source=copy-link&utm_medium=referral&utm_content=topbar.

[33] Naufal Suryanto et al. *Dta: Physical camouflage attacks using differentiable transformation network*. 2022.

[34] Christian Szegedy et al. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV]. URL: https://arxiv.org/abs/1409.4842.

[35] Christian Szegedy et al. *Intriguing properties of neural networks*. 2014. arXiv: 1312.6199 [cs.CV]. URL: https://arxiv.org/abs/1312.6199.

[36] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. *Fooling automated surveillance cameras: adversarial patches to attack person detection*. 2019. arXiv: 1904.08653 [cs.CV]. URL: https://arxiv.org/abs/1904.08653.

[37] Ultralytics. *YOLOv5: A state-of-the-art real-time object detection system*. https://docs.ultralytics.com. 2021.

[38] Donghua Wang et al. *FCA: Learning a 3D Full-coverage Vehicle Camouflage for Multi-view Physical Adversarial Attack*. 2021. arXiv: 2109.07193 [cs.CV]. URL: https://arxiv.org/abs/2109.07193.

[39] Jiakai Wang et al. *Dual attention suppression attack: Generate adversarial camouflage in physical world*. 2021.

[40] Chris Wise and Jo Plested. *Developing Imperceptible Adversarial Patches to Camouflage Military Assets From Computer Vision Enabled Technologies*. 2022. arXiv: 2202.08892 [cs.CV]. URL: https://arxiv.org/abs/2202.08892.

[41] Mariusz Wisniewski et al. *Drone Detection using Deep Neural Networks Trained on Pure Synthetic Data*. 2024. arXiv: 2411.09077 [cs.CV]. URL: https://arxiv.org/abs/2411.09077.

[42] Zuxuan Wu et al. *Making an invisibility cloak: Real world adversarial attacks on object detectors*. Springer, 2020.

[43] Yichuang Zhang et al. *Adversarial patch attack on multi-scale object detection for UAV remote sensing images*. 2022.

Acknowledgements

I wish to express my sincere gratitude to everyone who has supported me throughout my master's studies and the completion of this thesis.

My deepest appreciation goes to my family. To my parents, Patrizia and Gianfranco, and my sister Sole, thank you for believing in me, for your support, patience, and encouragement over these years. I could not have undertaken this journey without you.

I am incredibly grateful to MBDA for the opportunity to conduct my thesis work within their organization. I would like to specifically thank Luca Morino and Ivonne Valente for their generous assistance and professional guidance throughout my time there. Your insights, critical feedback, and support were fundamental to the development of this work.

I extend my profound thanks to my thesis supervisor, Prof. Enrico Magli for giving me the opportunity to work on this project and for the help in navigating this final part of my academic journey.