YOLO Multi-Object Color Attack (YMCA)

Christian Cipolletta
Advisor: Dr. Robi Polikar
Electrical & Computer Engineering
Rowan University

Engineering Clinic Final Report
May 2024

Abstract

In this paper, we investigate the susceptibility of YOLO (You Only Look Once) object detection models to artificial coloring attacks and examine the extent to which these attacks affect model performance. We hypothesize that the predictions of YOLO are dependent on color. Therefore, artificially colored objects may decrease model performance. We explore the significance of colorants as tools for adversaries to quickly and inexpensively alter the appearance of objects, potentially leading to misclassification by YOLO models. Our study delves into the background of Convolutional Neural Networks (CNNs) and YOLO architecture, highlighting their effectiveness in object detection tasks. Through a series of experiments, we quantify model performance using mean average precision (mAP) scores and evaluate the impact of artificial coloring on detection accuracy. Additionally, we discuss the implications of our findings in real-world scenarios, such as self-driving cars and AI threat detection systems, where model robustness is crucial for ensuring user safety. Our results suggest that artificially colored objects can indeed reduce the mAP scores of YOLO models, highlighting a potential vulnerability that needs to be addressed in future research and model development efforts.

1 Introduction

1.1 Problem Statement

In this study, we focus on designing a method to enhance the robustness and safety of computer vision systems, with a specific emphasis on YOLO models. Our objective is to assess the susceptibility of YOLO models to artificial coloring attacks and quantify their impact on model performance. Through the development of a methodology, we aim to identify and analyze the vulnerabilities of YOLO models to color-based perturbations. By doing so, we seek to contribute to the advancement of more resilient and reliable computer vision systems. We hypothesize that YOLO predictions are influenced by color variations, thereby suggesting a potential avenue for improving model robustness against adversarial attacks.

1.2 Significance

Colorants enable adversaries to cheaply and quickly change the color of an object. These can come in the form of paint, marker, food dye, and more.

An example of this would be if someone used food coloring to change the color of their food, painted a weapon to fool an AI threat detection system, or adversaries colored a stop-sign blue to fool a self-driving car.

1.3 Background

Convolutional Neural Networks (CNNs) are fundamental architectures in the artificial intelligence (AI) and machine learning (ML) world. Their effectiveness in recognizing patterns in images has led to their widespread use [1]. CNNs consist of three main parts: convolutional layers, pooling layers, and fully-connected layers. Convolutional layers apply filters to input data, enabling the network to extract features such as edges and textures from images. Pooling layers reduce the spatial dimensions of feature maps, preserving important information while improving computational efficiency. Fully-connected layers connect every neuron from one layer to another, enabling the network to perform classification or regression based on the extracted features. The implementation of a CNN can have as many or as few layers as the designer wants. There are many different architectures for CNNs with different advantages, but overall, these models are able to detect objects with human-like precision [2].

You Only Look Once (YOLO) is a CNN model that consists of 24 convolutional layers and two fully-connected layers [3]. It is designed for object detection and performs this task efficiently. As the name suggests, YOLO models only look at the image once, which allows for speeds greater than 45 frames per second (FPS). Yet, the architecture is still able to perform just as well or even better than other models. Several YOLO versions have been published, with YOLOv8, released in 2023 by a company called Ultralytics, boasting the fastest speeds, best performance, and ease of use [4]. As performance in these models improves, people's trust in them also increases. This means that they will soon be trusted with bigger and more complex tasks. However, the more important a task is, the higher the chance an adversary will want to disrupt the model using any means necessary.

Robustness describes a model's ability to perform as intended in the face of unknown or adversarial scenarios. There are many ways AI can be attacked, and these attacks vary in effectiveness, detectability, and feasibility. Various methods of categorization have been developed for attacks, one of which is based on the attacker's knowledge of the model. Attacks can be categorized as black-box, white-box, gray-box, or poisoning [5]. In a white-box attack, the attacker knows everything about the model, including its architecture, use, and training data. In a black-box attack, the adversary knows little except perhaps the use case of the model. Gray-box attacks fall between the two extremes. Poisoning attacks involve the attacker inserting mislabeled or otherwise harmful data into the model's training set.

Black-box attacks are of particular interest recently due to the commercialization of ML models [5]. Models inside self-driving cars and security systems are highly protected. While an adversary could attempt to access the model from the inside, this would be difficult. Instead, attackers are likely to create attacks on physical objects in the real world, such as perturbations on street signs in the form of graffiti, tape, stickers, and more [6]. Research has shown that these perturbations are effective in disrupting models' performance, particularly when applied to street signs. However, this effectiveness has not been tested extensively on other objects or with different colors.

1.4 Requirements

- **Real-Time Object Detection:** The model must identify objects as fast as a human would or even faster.
- **Robustness:** The model must be able to perform accurately in all conditions. This means that there should be no discrepancies in results, whether it is being used to detect normal objects, unknown objects, and artificially colored objects.

1.5 Deliverables

Our deliverable is a method for testing the color-based vulnerabilities of custom models and datasets. While we are testing our model in this work, we will ensure that the algorithms and programs used are transferable to other models with slight edits. This is a deficiency that should not only be tested but protected against.

1.6 Timeline

The timeline for this project can be seen as a Gantt chart here.

1.7 Constraints

- **Speed:** Real-time object detection, defined as achieving a minimum of 15 frames per second (FPS), is a critical requirement for ensuring timely detection and response in dynamic environments, such as autonomous vehicles or surveillance systems.

- **Hardware:** We must use an Nvidia Jetson AGX Orin Developer Kit, a small but powerful computer with a NVIDIA Ampere architecture GPU [7]. This hardware choice is essential for efficient processing of deep learning models, particularly for on-board applications where computational resources are limited and real-time performance is crucial.
- **Data (lack thereof):** The absence of publicly available large datasets for aerial object detection, particularly for applications involving drones, presents a significant challenge. To address this constraint, a synthetic dataset has been created, necessitating careful consideration of potential biases and limitations compared to real-world data. Additionally, Microsoft's Common Objects in Context (COCO) dataset was used as a standardized way to compute metrics on the attack.

2 Approach, Design and Validation

2.1 Trade-offs, Iterative Design, Risks and Opportunities

The project's design process involved careful consideration of various trade-offs, iterative exploration of design alternatives, and identification of associated risks and opportunities. The first major choice of this project was which model to use. Meeting the constraints of real-time inference and operating aboard a moving vehicle posed challenges. The YOLO architecture emerged as the obvious choice due to its ability to predict at high FPS. However, the decision boiled down to selecting between YOLOv7 and YOLOv8. A review of every major version was released while this project was in the initial stages [8]. YOLOv7, released in 2022, had an advantage of being older and more extensively tested [9]. It was accompanied by a robust paper and well-documented GitHub repository. On the other hand, YOLOv8, continuously updated by a company, offered abundant documentation, community support, and native multi-object tracking (MOT) support. While lacking its own dedicated paper, YOLOv8 was selected for its ease of use and ongoing support.

The next decision revolved around the type of attack to research. Initially, we attempted to implement a Fast Gradient Sign Attack (FGSM) due to its prominence in adversarial machine learning literature. However, after several weeks of working with it, we reevaluated the project's realistic expectations. FGSM, being a white-box attack, requires the attacker to possess comprehensive knowledge about the model [5]. Given that our model will be highly guarded and locally deployed, accessing it would be challenging for an attacker. Consequently, we pivoted towards developing a black-box attack. This led to the concept of coloring objects to evade detection by the model.

The experimental design underwent iterative adjustments, including modifications to the dataset composition and attack parameters to ensure statistical significance and comprehensive coverage of attack scenarios. This iterative approach facilitated the identification of optimal design choices while mitigating associated risks, such as dataset biases and limited attack efficacy. Initial testing utilized our synthetic drone dataset comprising 600 thousand training images generated using a computer. This dataset contained only two classes: drone and airplane. Initially, a model was trained on the original images. Subsequently, a segmentation model was employed to generate a mask for each drone in the image. This mask was then used to create a polygon resembling the drone's shape, which could be colored to simulate a paint job. Example images from the dataset and colored versions are depicted in Figure 1. While these visual observations were insightful, quantifiable performance measures were necessary. To achieve this, we utilized Microsoft's 2017 COCO dataset [10], which comprises 300 thousand images across 80 classes and provides standardized metrics for performance evaluation. This allowed us to compare our attack with the control in a consistent manner.

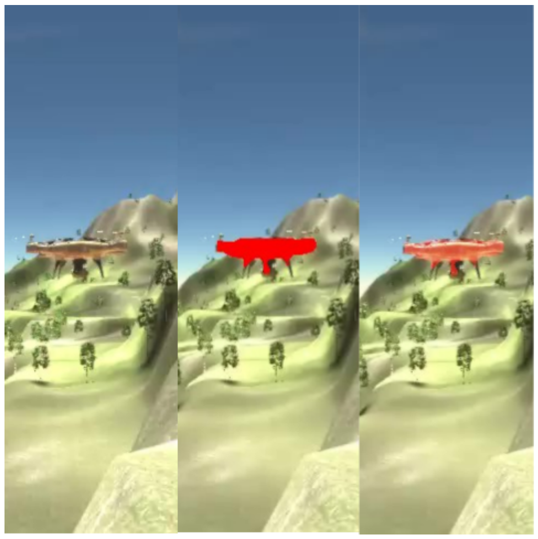


Figure 1: A side-by-side example of a frame in our drone dataset. Left is the unaltered frame. Middle is the drone colored red with an opactiy of 100%. The right is the drone colored red with an opacity of 75%.

Another design change involved the number of colors and opacities tested. Initially, testing was conducted on six colors and five opacities. These colors comprised the three primaries and three secondaries: blue, red, yellow, orange, green, and purple. Opacities ranged from 20% to 100% in 20% increments. However, it became apparent that this would not yield statistically significant data. Consequently, nine colors and ten opacities were tested instead. Opacities ranged from 10% to 10% in 10% increments, with the original six colors supplemented by brown, white, and gray.

2.2 Approach

Basic CNNs can have as little as three layers, one convolution, one pooling, and one fully connected. However, these are often too simple and do not provide the best results. An example of a simple CNN that is used to make predictions on the Modified National Institute of Standards and Technology (MNIST) database is shown in Figure 2 [11]. Let's denote the input image as X, the convolutional layer as C, the pooling layer as P, and the fully connected layer as FC.

- **Convolutional Layer (C)**:

$$Y = f(X * W + b)$$

where $*$ represents the convolution operation, W is the filter (kernel) applied to the input image, b is the bias term, and f is the activation function (e.g., ReLU).

- **Pooling Layer (P)**:
$$Y = \text{MaxPool}(X)$$
- **Fully Connected Layer (FC)**: - The output of the last pooling layer is flattened and fed into one or more fully connected layers, followed by a softmax activation function for classification tasks.

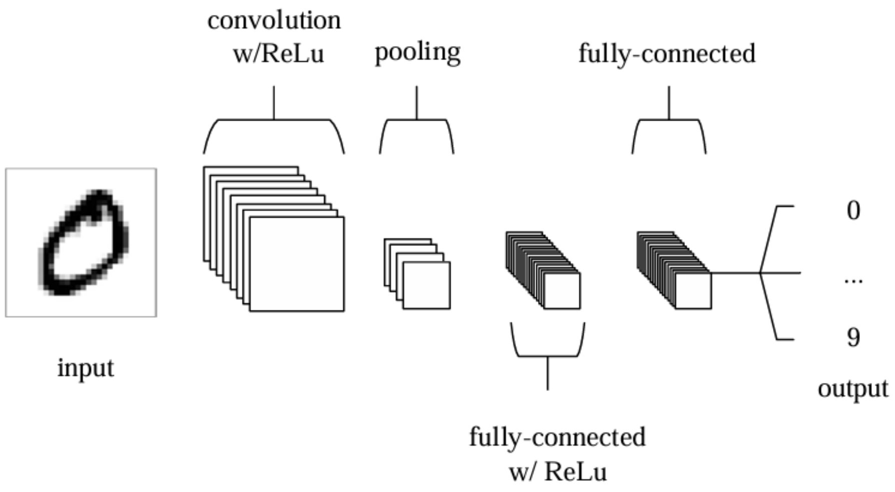


Figure 2: A simple CNN architecture used on the MNIST dataset [1].

YOLO is a type of CNN with its own architecture. YOLO has a network of 24 convolutional layers. These feed into two fully connected layers. The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates. The YOLO architecture comprises multiple convolutional layers followed by fully connected layers. Let's denote the convolutional layer as C' and the fully connected layer as FC'.

- **Convolutional Layers (C')**: - YOLO employs a series of convolutional layers to extract features from the input image. Each convolutional layer is followed by batch normalization and a ReLU activation function.

- **Fully Connected Layers (FC')**: - The output from the last convolutional layer is flattened and passed through two fully connected layers. These layers predict the output probabilities and bounding box coordinates for object detection.

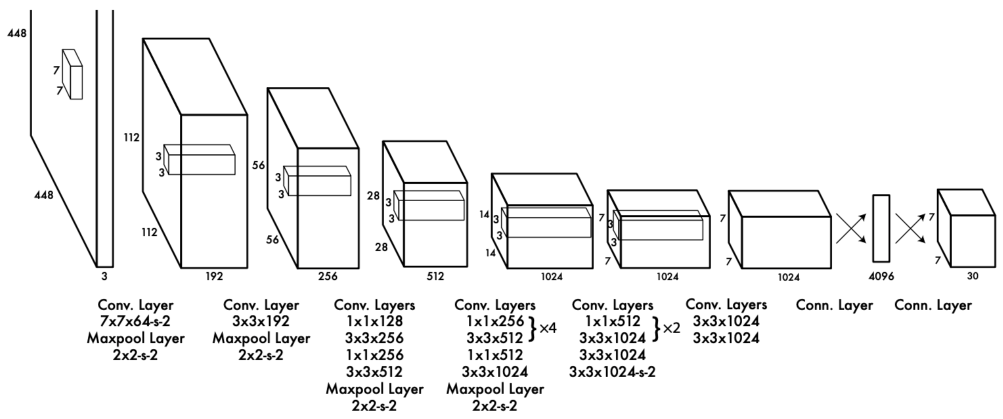


Figure 3: YOLO architecture which consists of 24 convolutional layers and 2 fully connected layers [3].

To quantify model performance we used the mean average precision (mAP) score. An important part of detection models is the intersection over union (IoU). IoU is the measure of the overlap between the predicted bounding box and the ground truth bounding box. The formula for IoU is shown in equation 1 and a visual representation is shown in Figure 4.

$$IoU = \frac{AreaOfOverlap}{AreaOfUnion} \tag{1}$$

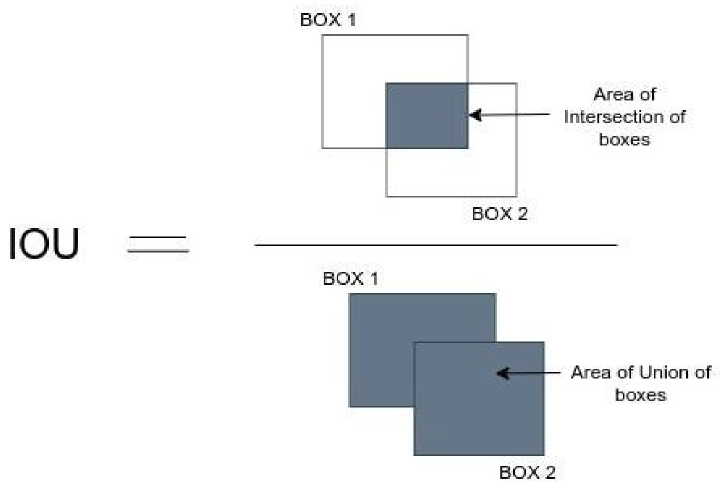


Figure 4: Visual representation of equation 1 [12].

Due to its formula, IoU is a value between zero and one. It is used to determine if a model's prediction is a true positive (TP) or a false positive (FP). mAP calculation starts with finding the precision and recall of a model. Precision, equation 2, is the how well the model only makes correct predictions. Recall, equation 3, is calculated using the number of objects the model missed.

$$Precision = \frac{CorrectPredictions}{TotalPredictions} = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{CorrectPredictions}{TotalGroundTruth} = \frac{TP}{TP + FN} \tag{3}$$

where FN stands for false negative. TP is the number of correct predictions of the model, FP is the number of incorrect predictions, and FN is the number of missed predictions. Both precision and recall are a number between zero and one. The closer to one, the better.

The next step is to create a precision-recall curve. This is a graph, an example is shown in Figure 5, where the precision value is the vertical axis and recall is the horizontal axis. Average precision (AP) is calculated by finding the area under the curve through integration. Since the precision-recall curve is a scatter-plot, interpolation is used to discretize the integration operation for discrete data and then summed. In the case of YOLOv8, a 101 point interpolation is used to estimate the precision-recall curve. The integration of this interpolation is the AP. This AP is for only a single class, but with COCO there are 80 classes. The calculation for mAP is seen in equation 4 and is the sum of the AP for each class divided by the number of classes.

$$mAP = \frac{1}{N} \sum_{k=1}^{k=N} AP_k \tag{4}$$

where N is the number of classes.

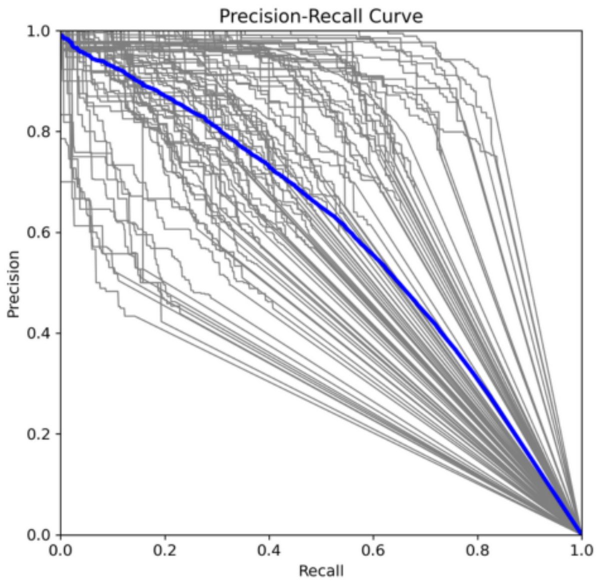


Figure 5: The precision-recall curve of the pretrained YOLOv8n model when tested on the five thousand validation images of the COCO dataset.

mAP50-95 or COCO mAP is a metric used specifically for the COCO dataset. This metric differs from standard mAP because it takes into account the mAP at IoU thresholds 0.5 all the way to 0.95. The steps for the calculation are below and have been simplified into equation 5.

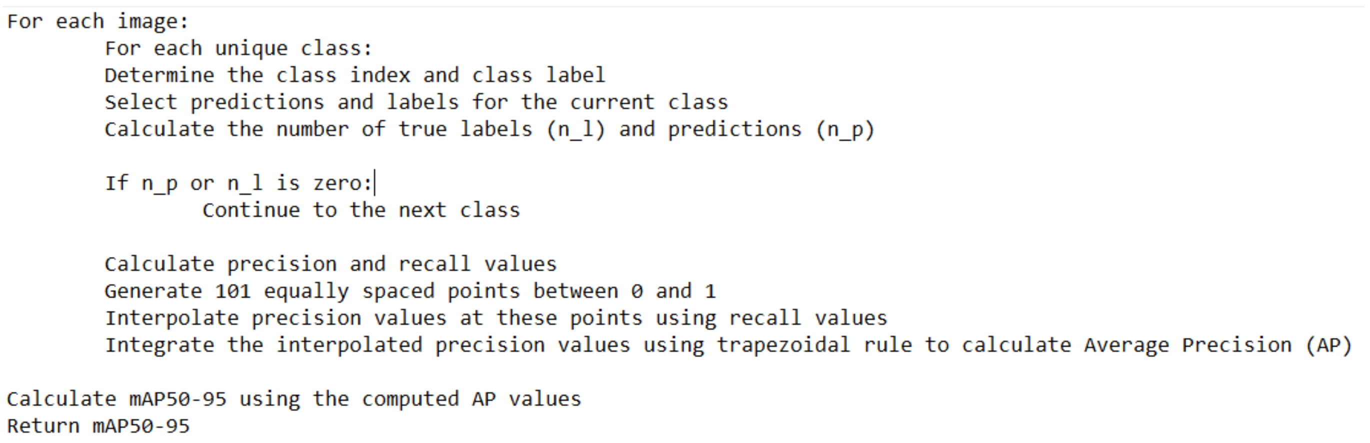
1. AP is calculated for the IoU threshold of 0.5 for each class.

2. Calculate the precision at every recall value (0 to 1 with a step size of 0.01), then it is repeated for IoU thresholds of 0.55,0.60,. . . ,.95.

3. Average is taken over all 80 classes and 10 thresholds.

$$mAP50 - 95 = \frac{1}{(N)(T)} \sum_{k=1}^{N} \sum_{t=1}^{T} AP_{k,t} \tag{5}$$

where N is the number of classes, T is the number of IoU thresholds (in this case $T = 10$ because thresholds range from 0.5 to 0.95 with a step size of 0.05), $AP_{k,t}$ is the average precision for class k at IoU threshold t.

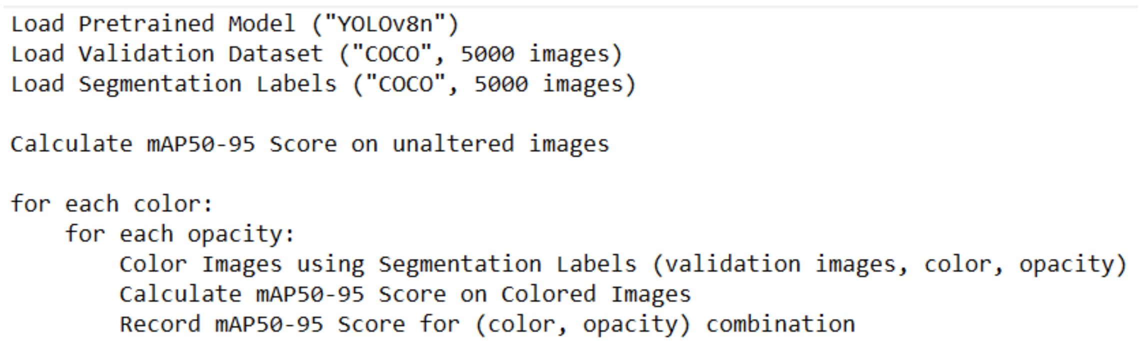


```
For each image:
        For each unique class:
        Determine the class index and class label
        Select predictions and labels for the current class
        Calculate the number of true labels (n_l) and predictions (n_p)

        If n_p or n_l is zero:
                Continue to the next class

        Calculate precision and recall values
        Generate 101 equally spaced points between 0 and 1
        Interpolate precision values at these points using recall values
        Integrate the interpolated precision values using trapezoidal rule to calculate Average Precision (AP)

Calculate mAP50-95 using the computed AP values
Return mAP50-95
```

Figure 6: Pseudo code for the approach.

3 Experiments, Results and Validation

For our experiment we calculated the mAP score of the pretrained YOLOv8n model when predicting on the five thousand validation images of the COCO dataset. Then, we used the segmentation labels COCO has to color the individual objects in the validation image a different color. The pseudo code for this method is shown in Figure 7. An example of the model predictions is shown in Figure 8 We calculated the mAP score again of the same model when run on the new colored images. This was repeated for all nine colors and ten opacities to gather 91 data points for our results.



```
Load Pretrained Model ("YOLOv8n")
Load Validation Dataset ("COCO", 5000 images)
Load Segmentation Labels ("COCO", 5000 images)

Calculate mAP50-95 Score on unaltered images

for each color:
    for each opacity:
        Color Images using Segmentation Labels (validation images, color, opacity)
        Calculate mAP50-95 Score on Colored Images
        Record mAP50-95 Score for (color, opacity) combination
```

Figure 7: Pseudo code for the experiment.

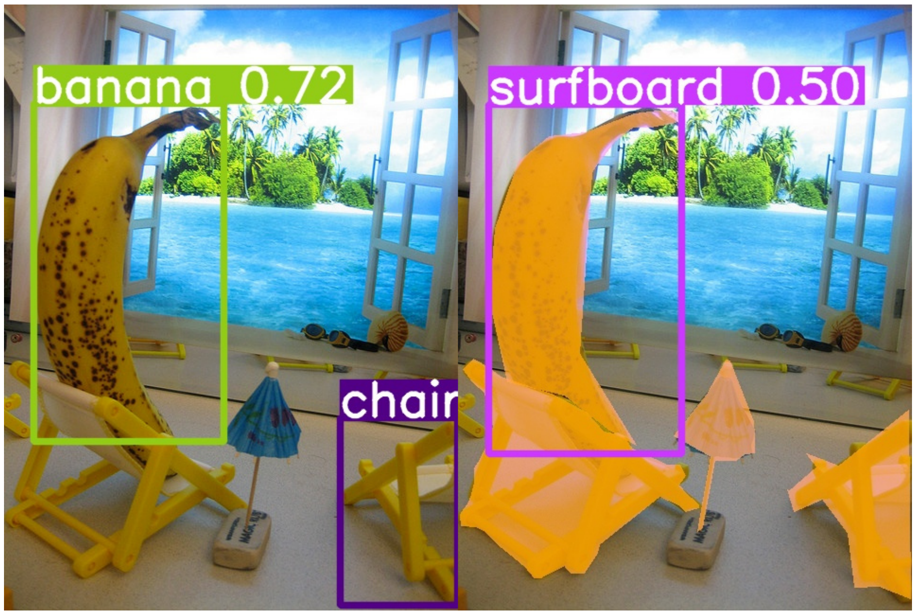
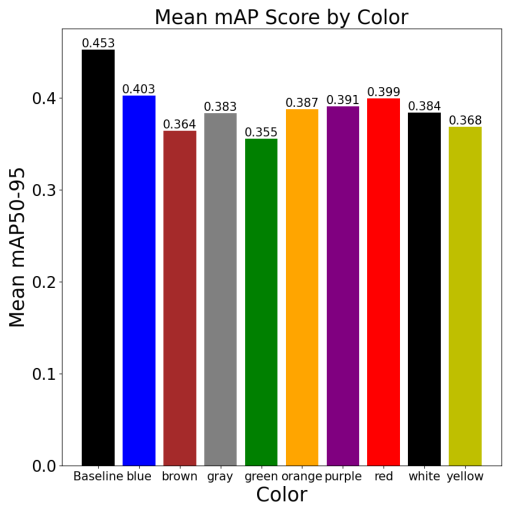


Figure 8: Model predictions when run on an unaltered image (left) and objects colored orange with an opacity of 80% (right).

The results were the mAP scores of all 91 runs. The average mAP50-95 score of each of the nine colors is shown in Figure 9 The best score by far was the model tested on the unaltered dataset. It had a mAP of 0.453 while every single color had a worse score of 0.403 or lower. The average score divided by opacity is shown in Figure 9. This showed a different story. The best mAP score was shown to be 10% opacity with a score of 0.454. The baseline was close behind with a score of 0.453, but it was bested. The rest of the opacities were worse the original and ranged from 0.451 to 0.126.



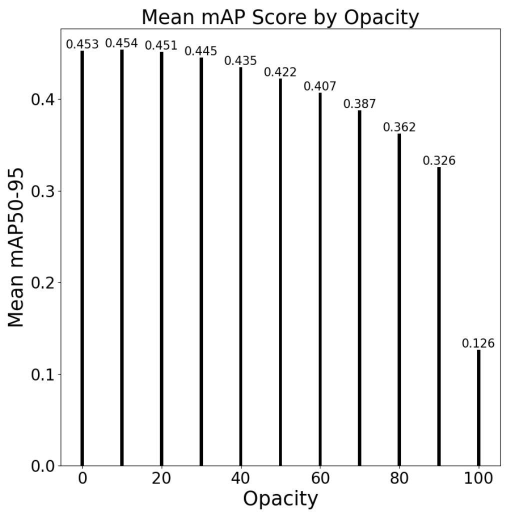


Figure 9: Bar graph of the average mAP score grouped by color (left) and Bar graph of the average mAP score grouped by opacity (right).

Figure 10 is a heatmap of all the 90 runs on colored images. This graph shows the difference between the individual colors and opacities clearer. The darker the red, the higher the mAP50-95 score. The blue line at 100% opacity shows the poor performance of the model at this opacity. Also, the relatively light columns of green and yellow show that these colors made the model perform the worst.

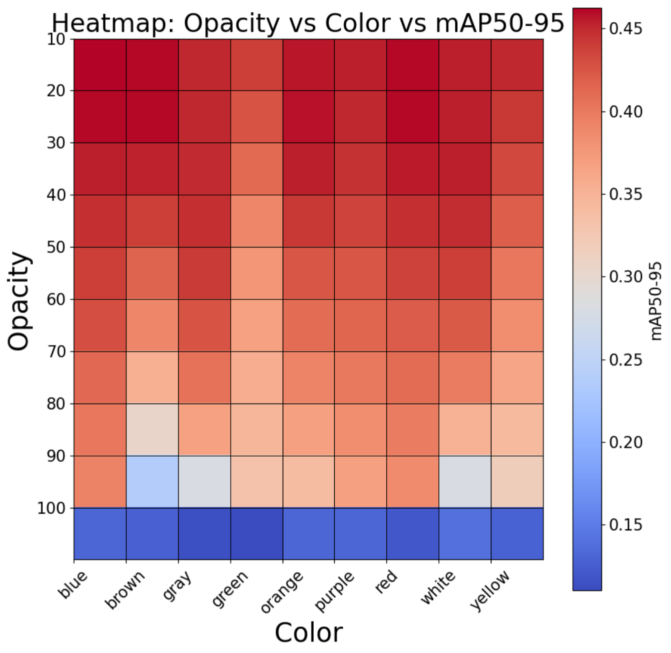


Figure 10: Bar graph of the average mAP score grouped by color (left) and Bar graph of the average mAP score grouped by opacity (right).

As for the other requirements laid out earlier, the testing showed promising results as well. The model ran at an average speed of 52.18 FPS for predictions when run on an Nvidia A6000 GPU. YOLOv8 was tested on the Orin as well and ran successfully. However, speed testing was not done at this time do to time constraints.

4 Discussions and Conclusions

In accordance with the outlined requirements, our investigation sheds light on the model's performance regarding real-time object detection and robustness under varying conditions. Firstly, concerning real-time object detection, our findings demonstrate that the current model operates swiftly, aligning with the necessity for rapid identification akin to human perception. Despite the introduction of colored perturbations, the model's response time remains consistent, indicating its suitability for real-time applications. Secondly, with regards to robustness, our analysis reveals a notable vulnerability when objects are artificially colored. Specifically, the incorporation of colorants, particularly those with higher opacities, poses a significant challenge to the model's accuracy. This vulnerability highlights a potential area for improvement to ensure robust performance across diverse scenarios. Moreover, while our experiment primarily focuses on digitally coloring objects, it is essential to acknowledge the assumption that this process mimics the effects observed in physical scenarios. This assumption underscores the need for further validation through real-world experimentation to fully assess the model's robustness. Furthermore, the limitation regarding the availability of comprehensive datasets, encompassing both segmentation and detection labels, underscores the importance of dataset quality in training robust models. Future endeavors should prioritize the acquisition or generation of such datasets to enhance model training and evaluation. In conclusion, while the current model demonstrates commendable performance in real-time object detection, our investigation underscores the imperative of addressing vulnerabilities, particularly concerning artificially colored objects, to bolster its overall robustness. Moving forward, our proposed future work, encompassing robust training methodologies and targeted class-based assessments, aims to fortify the model's capabilities in alignment with the stipulated requirements.

5 Impact Factors

5.1 Standards

- **IEEE Standard for Robustness Testing and Evaluation of Artificial Intelligence (AI)-based Image Recognition Service [13]:** This standard helps create an overall robustness metric that is standardized. The acronyms and testing procedures provided by this standard helped to guide research and find new attacks. Mainly, we looked at the evaluation process provided in this standard. This gave us a good idea of how to properly setup and run the experiment for quality purposes. Specifically, we went though the steps of choosing the type of attack (color), the images used for the evaluation (COCO), the metrics under evaluation (mAP), and the parameters of the corruptions (the color and opacity).

- **IEEE Standard for Artificial Intelligence (AI) Model Repre-**

sentation, Compression, Distribution, and Management [14]: This standard is incredibly dense. It is filled with acronyms, tables, and guidelines that are used for many different things. A lot of the standard we plan to use for future work such as the model protection guidelines and the guidelines on data generation/augmentation. Specifically, in our experiment we looked at the compression and decompression process to ensure we lose no quality in our models. YOLOv8 compresses model weights using PyTorch and decompresses them using the same. This abstracted way to compression is easier, however, it makes it hard to understand the consequences of such methods. Overall, this standard helped improve our understanding and made us feel secure in using the methods provided in YOLOv8.

- **IEEE Recommended Practice for the Quality Management of Datasets for Medical Artificial Intelligence [15]:** This standard seems out of place due to do its name and the word "Medical." While we are not being a model for the medical field, there is a still a need for quality management of our datasets to be effective and safe. The part of the standard about retiring old data was of the most interest to us. We used the recommended practices to ensure that COCO was still a valid dataset for us to use. Furthermore, we plan to use the ideas outlined in the standard for our drone dataset to ensure it stays up to date.

- **PEP 8 – Style Guide for Python Code [16]:** This guide served as a foundational resource for maintaining code quality and consistency in our Python implementations. By adhering to PEP 8 guidelines, we enhanced the readability and maintainability of our codebase, facilitating collaboration and minimizing potential errors.

5.2 Public Health, Welfare, Safety

The public health, safety, and welfare factors considered for our project were data privacy and user safety. The concern for data privacy comes from the training portion of machine learning. To have an effective model, it needs to train on a large amount of data. This data needs to be sourced. For this project we are using data that has been synthetically created by us, so we know that there were no photos taken without permission which would violate the privacy of the subjects of the photos. Furthermore, if the model fails while in use it can cause serious injury or death. This is why robust testing is done on attacks most likely to be seen in real life.

5.3 Global, Social, and Cultural Factors

The global factor we considered in this project was the recently growing use of drones in global conflicts. By this we mean that drones are being used on a scale never before seen. This use also comes with innovations. Oppositions will constantly be evolving and changing their strategies, including their drone models. This means that the model we are using now, may not work on the new drone types. To combat this, we made our model on YOLOv8 because it can be trained easily on new data. We also made sure the code was documented well so it is clear how to use it to a different person at a later date. Furthermore, since this is being built for use in the US (and could be potentially harmful if other countries got their hands on it), the code is all written and documented in English. There is little to no need to translate it to any other language.

5.4 Environmental Factors

There are a number of environmental factors to consider for our project, some of them that we can control and some that we cannot. Some that we can't control are the resources needed for the vehicles that are going to run the model, the resources of the attacking drones, and really any other resources that the army may use. The factor we have more control over is the amount of electricity needed to train and run the model. Obviously the amount and source of the power running the computers that train the model, changes the impact on the environment. Some considerations we thought of were trying to have the computer be powered by solar panels or another green energy source. Also, the rule of thumb is that a smaller model takes less time to train and less power to run. We are using YOLOv8 because it is very lightweight making it quick and more energy efficient. Low power-consumption GPUs are used to lessen the amount of electricity needed to run the model in practice (which uses less fuel if on a vehicle).

5.5 Economic Factors

The economic factors considered for this project are the price of computational power and the price of electricity to power the computers. GPUs cost hundreds to thousands of dollars to buy and the more powerful they are, the more power they use which also costs money. These costs limited our ability to train the models in a timely fashion. However, the YOLOv8 design of the model allows inference (prediction on images) to happen in real time on relatively small GPUs. This saves money on the cost of the expensive GPUs and the power cost as well.

6 Professional and Ethical Issues in Consideration of Factors

AI is a word that shows up in a significantly increasing amount of headlines in recent years. Oftentimes it is controversial for a lot of reasons like environmental impact, legality, privacy concerns, incorrectness, and more. The issues with environmental impact come from the amount of electricity it takes to both train and use the models. These can be addressed by using renewable energy sources or by using smaller models which we did. Privacy concerns are a large issue with AI because of the sheer magnitude of data needed. Oftentimes shortcuts are taken to gather the necessary data which could infringe on the privacy of people or businesses. To combat this we created synthetic data which is computer generated so there is no question of privacy issues.

7 Teamwork, Lessons and Knowledge Learned

I was the only person working on this specific project so my role was everything from researching to brainstorming to building and documenting. However, there was still a collaborative atmosphere because the other students in Dr. Polikar's clinic and I would meet every Tuesday and Thursday to give updates, ask for help, and discuss. I helped other team members solve their problems, establish goals, and plan tasks.

I have gained skills from this project that will follow me for the rest of my life. The skills specific to this project are the different types of adversarial attacks, some examples of attacks and defenses, and how to implement both types. I have also learned better strategies for brainstorming, documenting work, and reading research papers effectively. Furthermore, I improved my skills in python coding especially batching experiment and creating reusable functions. Another thing I developed was my ability to make a concise and effective poster. Everything is an opportunity to learn.

8 References

[1] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. arXiv: 1511.08458 [cs.NE].

[2] Ayoub Benali Amjoud and Mustapha Amrouch. "Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review". In: *IEEE Access* 11 (2023), pp. 35479–35516. DOI: 10.1109/ACCESS.2023.3266093.

[3] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection*. 2016. arXiv: 1506.02640 [cs.CV].

[4] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. *Ultralytics YOLO*. Version 8.0.0. Jan. 2023. URL: https://github.com/ultralytics/ultralytics.

[5] Jing Lin et al. *ML Attack Models: Adversarial Attacks and Data Poisoning Attacks*. 2021. arXiv: 2112.02797 [cs.LG].

[6] Kevin Eykholt et al. *Robust Physical-World Attacks on Deep Learning Models*. 2018. arXiv: 1707.08945 [cs.CR].

[7] Leela S. Karumbunathan. *NVIDIA Jetson AGX Orin Series*. Tech. rep. Nvidia, Nov. 2010.

[8] Juan Terven, Diana-Margarita Córdova-Esparza, and Julio-Alejandro Romero-González. "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS". In: *Machine Learning and Knowledge Extraction* 5.4 (Nov. 2023), pp. 1680–1716. ISSN: 2504-4990. DOI: 10.3390/make5040083. URL: http://dx.doi.org/10.3390/make5040083.

[9] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. *YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors*. 2022. arXiv: 2207.02696 [cs.CV].

[10] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[11] Li Deng. "The mnist database of handwritten digit images for machine learning research". In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 141–142.

[12] Vineeth S Subramanyam. *IOU (intersection over union)*. Jan. 2021. URL: https://medium.com/analytics-vidhya/iou-intersection-over-union-705a39e7acef.

[13] "IEEE Standard for Robustness Testing and Evaluation of Artificial Intelligence (AI)-based Image Recognition Service". In: *IEEE Std 3129-2023* (2023), pp. 1–34. DOI: `10.1109/IEEESTD.2023.10141539`.

[14] "IEEE Standard for Artificial Intelligence (AI) Model Representation, Compression, Distribution, and Management". In: *IEEE Std 2941-2021* (2022), pp. 1–226. DOI: `10.1109/IEEESTD.2022.9739118`.

[15] "IEEE Recommended Practice for the Quality Management of Datasets for Medical Artificial Intelligence". In: *IEEE Std 2801-2022* (2022), pp. 1–31. DOI: `10.1109/IEEESTD.2022.9812564`.

[16] Guido van Rossum, Barry Warsaw, and Nick Coghlan. *Style Guide for Python Code*. PEP 8. 2001. URL: `https://www.python.org/dev/peps/pep-0008/`.

9 Appendix

9.1 Code

The code used for the experiments is shown below. It can also be found on GitHub here.

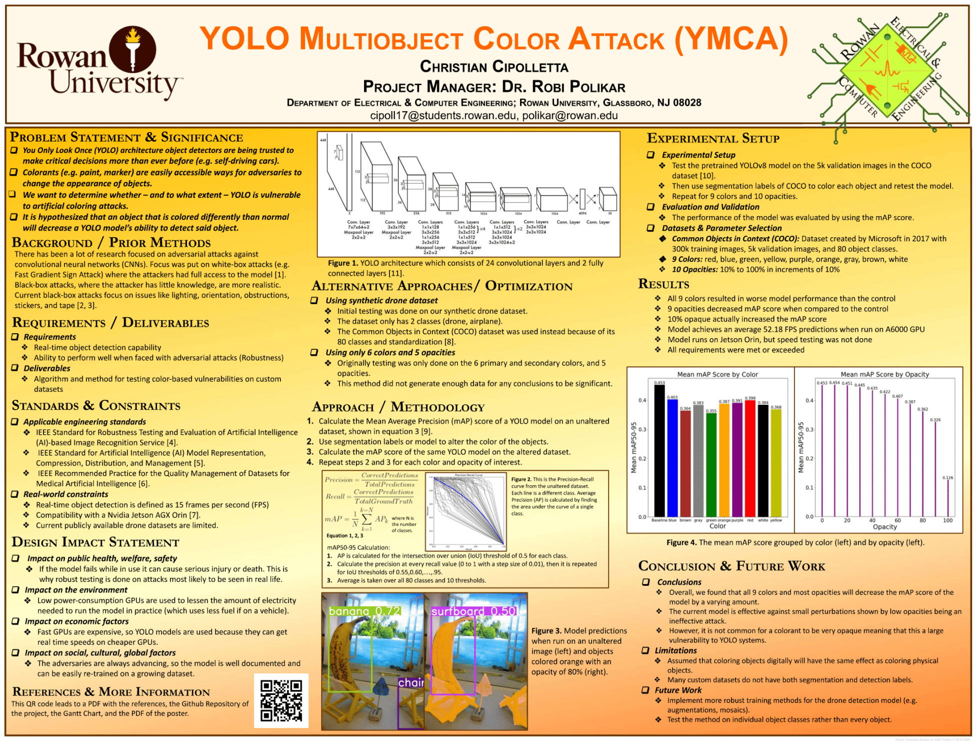
```python
# Import necessary libraries
from ultralytics import YOLO
import wandb
from color_and_save import color_and_save_with_labels
from cfg.colors import colors_dict
import os

# Main function
if __name__ == '__main__':
    # Load YOLO model
    model = YOLO('yolov8n.pt')

    # Path to validation data
    data_path = 'path/to/data.yaml'

    # Project name for WandB
    project_name = "YMCA"

    # Test image path
    test_img = "path/to/img"

    # Classes to color (if None, all classes are colored)
    classes = None

    # Device
    device = '0'

    # Initialize a new wandb run for the baseline
    wandb.init(project=project_name, name=f"Baseline")

    # Validate the model with default settings
    validation_results = model.val(data=data_path,
                                   imgsz=640,
                                   batch=16,
                                   conf=0.25,
                                   iou=0.6,
                                   plots=True,
                                   save_json=True,
                                   device=device)

    # Log example image
    wandb.log({"Example Image": wandb.Image(test_img)})

    # Predict and plot results on the example image
    ex_results = model.predict(test_img)[0].plot()
    ex_results = ex_results[:, :, ::-1]
    wandb.log({"Predicted Image": wandb.Image(ex_results)})

    # Extract evaluation metrics
    map50_95 = validation_results.box.map    # map50-95
    map50 = validation_results.box.map50  # map50
    map75 = validation_results.box.map75  # map75
    map_by_class = validation_results.box.maps.tolist()

    # Log evaluation metrics
    wandb.log({"map50_95": map50_95, "map50": map50, "map75":
    ↪  map75})

    # Path to colored validation data
    data_path = 'path/to/data.yaml'

    # Directories for images, labels, and output
    image_directory = 'path/to/imgs'
    labels_directory = 'path/to/segmentation_labels'
    output_directory = 'path/to/output_directory'

    # End the current wandb run
    wandb.finish()

    # Define colors and opacities
    colors = list(colors_dict.values())
    opacities = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
    ↪  0.9]

    # Loop over each color and opacity combination
    for color in colors:
        for opacity in opacities:
            # Skip if opacity > 0 and color is black (no
            ↪  coloring applied)
            if (opacity > 0) and (color == (0, 0, 0)):
                continue

            # Get color string
            color_string = list(colors_dict.keys())
            [list(colors_dict.values()).index(color)]

            # Apply colorization and save the results
            color_and_save_with_labels(image_directory,
            ↪  labels_directory, output_directory, color,
            ↪  opacity, classes=classes)

            # Initialize a new wandb run for each color
            wandb.init(project=project_name,
            ↪  name=f"{color_string}_{int(opacity*100)}")

            # Customize validation settings for colored images
            validation_results = model.val(data=data_path,
                                           imgsz=640,
                                           batch=16,
                                           conf=0.25,
                                           iou=0.6,
                                           plots=True,
                                           save_json=True,
                                           device=device)

            # Log example image
            wandb.log({"Example Image": wandb.Image(test_img)})

            # Predict and plot results on the example image
            ex_results = model.predict(test_img)[0].plot()
            ex_results = ex_results[:, :, ::-1]
            wandb.log({"Predicted Image":
            ↪  wandb.Image(ex_results)})

            # Extract evaluation metrics
            map50_95 = validation_results.box.map     # map50-95
            map50 = validation_results.box.map50   # map50
            map75 = validation_results.box.map75   # map75
            map_by_class = validation_results.box.maps.tolist()

            # Log evaluation metrics
            wandb.log({"map50_95": map50_95, "map50": map50,
            ↪  "map75": map75})

            # End the current wandb run
            wandb.finish()

```

9.2 Weights & Biases

Weights and Biases (Wandb) was used to group, save, and debug all the validation tests in this experiment. You can view all the data and sample debug images on Wandb here.

9.3 Poster

A poster was created for this work and presented at Rowan University. The poster is shown in Figure 11 and can also be found in PDF form here.



YOLO Multiobject Color Attack (YMCA)

Christian Cipolletta

Project Manager: Dr. Robi Polikar

Department of Electrical & Computer Engineering; Rowan University, Glassboro, NJ 08028

cipoll17@students.rowan.edu, polikar@rowan.edu

Problem Statement & Significance

- ***You Only Look Once (YOLO) architecture object detectors are being trusted to make critical decisions more than ever before (e.g. self-driving cars).***
- ***Colorants (e.g. paint, marker) are easily accessible ways for adversaries to change the appearance of objects.***
- ***We want to determine whether – and to what extent – YOLO is vulnerable to artificial coloring attacks.***
- ***It is hypothesized that an object that is colored differently than normal will decrease a YOLO model's ability to detect said object.***

Background / Prior Methods

There has been a lot of research focused on adversarial attacks against convolutional neural networks (CNNs). Focus was put on white-box attacks (e.g. Fast Gradient Sign Attack) where the attackers had full access to the model [1]. Black-box attacks, where the attacker has little knowledge, are more realistic. Current black-box attacks focus on issues like lighting, orientation, obstructions, stickers, and tape [2, 3].

Requirements / Deliverables

- ***Requirements***
  - Real-time object detection capability
  - Ability to perform well when faced with adversarial attacks (Robustness)
- ***Deliverables***
  - Algorithm and method for testing color-based vulnerabilities on custom datasets

Standards & Constraints

- ***Applicable engineering standards***
  - IEEE Standard for Robustness Testing and Evaluation of Artificial Intelligence (AI)-based Image Recognition Service [4].
  - IEEE Standard for Artificial Intelligence (AI) Model Representation, Compression, Distribution, and Management [5].
  - IEEE Recommended Practice for the Quality Management of Datasets for Medical Artificial Intelligence [6].
- ***Real-world constraints***
  - Real-time object detection is defined as 15 frames per second (FPS)
  - Compatibility with a Nvidia Jetson AGX Orin [7].
  - Current publicly available drone datasets are limited.

Design Impact Statement

- ***Impact on public health, welfare, safety***
  - If the model fails while in use it can cause serious injury or death. This is why robust testing is done on attacks most likely to be seen in real life.
- ***Impact on the environment***
  - Low power-consumption GPUs are used to lessen the amount of electricity needed to run the model in practice (which uses less fuel if on a vehicle).
- ***Impact on economic factors***
  - Fast GPUs are expensive, so YOLO models are used because they can get real time speeds on cheaper GPUs.
- ***Impact on social, cultural, global factors***
  - The adversaries are always advancing, so the model is well documented and can be easily re-trained on a growing dataset.

References & More Information

This QR code leads to a PDF with the references, the Github Repository of the project, the Gantt Chart, and the PDF of the poster.

Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 1. YOLO architecture which consists of 24 convolutional layers and 2 fully connected layers [11].

Alternative Approaches/ Optimization

- ***Using synthetic drone dataset***
  - Initial testing was done on our synthetic drone dataset.
  - The dataset only has 2 classes (drone, airplane).
  - The Common Objects in Context (COCO) dataset was used instead because of its 80 classes and standardization [8].
- ***Using only 6 colors and 5 opacities***
  - Originally testing was only done on the 6 primary and secondary colors, and 5 opacities.
  - This method did not generate enough data for any conclusions to be significant.

Approach / Methodology

1. Calculate the Mean Average Precision (mAP) score of a YOLO model on an unaltered dataset, shown in equation 3 [9].
2. Use segmentation labels or model to alter the color of the objects.
3. Calculate the mAP score of the same YOLO model on the altered dataset.
4. Repeat steps 2 and 3 for each color and opacity of interest.

$$Precision = \frac{CorrectPredictions}{TotalPredictions}$$

$$Recall = \frac{CorrectPredictions}{TotalGroundTruth}$$

$$mAP = \frac{1}{N}\sum_{k=1}^{k=N} AP_k$$ where N is the number of classes.

Equation 1, 2, 3

Figure 2. This is the Precision-Recall curve from the unaltered dataset. Each line is a different class. Average Precision (AP) is calculated by finding the area under the curve of a single class.

mAP50-95 Calculation:

1. AP is calculated for the intersection over union (IoU) threshold of 0.5 for each class.
2. Calculate the precision at every recall value (0 to 1 with a step size of 0.01), then it is repeated for IoU thresholds of 0.55,0.60,…,.95.
3. Average is taken over all 80 classes and 10 thresholds.

Figure 3. Model predictions when run on an unaltered image (left) and objects colored orange with an opacity of 80% (right).

Experimental Setup

- ***Experimental Setup***
  - Test the pretrained YOLOv8 model on the 5k validation images in the COCO dataset [10].
  - Then use segmentation labels of COCO to color each object and retest the model.
  - Repeat for 9 colors and 10 opacities.
- ***Evaluation and Validation***
  - The performance of the model was evaluated by using the mAP score.
- ***Datasets & Parameter Selection***
  - ***Common Objects in Context (COCO):*** Dataset created by Microsoft in 2017 with 300k training images, 5k validation images, and 80 object classes.
  - ***9 Colors:*** red, blue, green, yellow, purple, orange, gray, brown, white
  - ***10 Opacities:*** 10% to 100% in increments of 10%

Results

- All 9 colors resulted in worse model performance than the control
- 9 opacities decreased mAP score when compared to the control
- 10% opaque actually increased the mAP score
- Model achieves an average 52.18 FPS predictions when run on A6000 GPU
- Model runs on Jetson Orin, but speed testing was not done
- All requirements were met or exceeded

Figure 4. The mean mAP score grouped by color (left) and by opacity (left).

Conclusion & Future Work

- ***Conclusions***
  - Overall, we found that all 9 colors and most opacities will decrease the mAP score of the model by a varying amount.
  - The current model is effective against small perturbations shown by low opacities being an ineffective attack.
  - However, it is not common for a colorant to be very opaque meaning that this a large vulnerability to YOLO systems.
- ***Limitations***
  - Assumed that coloring objects digitally will have the same effect as coloring physical objects.
  - Many custom datasets do not have both segmentation and detection labels.
- ***Future Work***
  - Implement more robust training methods for the drone detection model (e.g. augmentations, mosaics).
  - Test the method on individual object classes rather than every object.

Figure 11: YMCA poster created for presentation at Rowan University.