Efficient and Adversarially Robust Object Detection

Anton Liu[†,*], Roberto Solis-Oba[†]
[†] Dept. of Computer Science, Western University

Abstract

This work introduces a novel approach to object detection that achieves high efficiency and adversarial robustness by adapting the HYDRA pruning framework to the YOLOv3 model. Object detection deep learning models, particularly in safety-critical applications such as autonomous driving and surveillance, require high computational efficiency and resilience against adversarial attacks. We investigate both structured and unstructured pruning techniques to assess their impact on model size, accuracy, and inference time. By using DeepLIFT for importance scoring we achieve targeted pruning that strikes a balance between model compactness and inference performance. Experimental results show that, when guided by DeepLIFT scoring, unstructured pruning consistently outperforms structured pruning in preserving higher mean Average Precision (mAP) across various pruning levels. DeepLIFT-based importance scoring enables precise pruning, allowing the YOLOv3 model to maintain robust detection performance even at high pruning ratios. This selective pruning method not only reduces computational demands, improving real-time processing capabilities, but also ensures the preservation of critical high detection performance.

Keywords: Object Detection, CNN Pruning, Adversarial Robustness, DeepLIFT

1. Introduction

Object detection is used in numerous real-world applications to enable computers to identify and locate objects within images or videos. Object detection algorithms aim to address two key challenges: determining an object's spatial extent (bounding boxes) and classifying it with the correct label [1]. These capabilities are cruicial for tasks like autonomous driving, surveillance, and inventory management. Additionally, object detection underpins higher-level vision tasks such as scene understanding and action recognition.

Convolutional Neural Networks (CNNs) have demonstrated remarkable potential for object detection and have been the main subject of recent research [2]. Designed for processing structured multi-dimensional data like images, CNNs leverage convolutional kernels with learnable weights to detect patterns such as edges, textures, and shapes. Through multiple layers, CNNs construct complex representations, enabling accurate object detection [3].

In safety-critical applications like autonomous driving, failures in object detection systems can have severe consequences, as evidenced by the fatal Uber self-driving car accident in 2018 [4]. While deep learning-based models have improved performance, their vulnerability to adversarial attacks remains a significant challenge [5]. Adversarial attacks, which involve maliciously crafted perturbations to input data, can lead to incorrect or missed detections. Adversarial defence aims to enhance model robustness by improving performance under adversarial scenarios, typically through adversarial training [6]. However, research on defense techniques is limited compared to adversarial attacks, emphasizing the need to enhance model reliability for real-world applications.

Despite their recent success, deep learning methods are computationally intensive during both training and inference due to the need to process large volumes of data through multiple layers. Training requires updating millions of parameters through backpropagation, while inference demands rapid calculations to generate predictions. As models grow in size and complexity, efficiency becomes an increasing concern. Efficiency can be measured in terms

[*] aliu467@uwo.ca

This article is © 2025 by author(s) as listed above. The article is licensed under a Creative Commons Attribution (CC BY 4.0) International license (https://creativecommons.org/licenses/by/4.0/legalcode), except where otherwise indicated with respect to particular material included in the article. The article should be attributed to the author(s) identified above.

of computational resources, such as inference time, memory usage, processing power, and energy consumption [7].

Several methods can enhance CNN efficiency, including transfer learning, quantization, weight sharing, and pruning [8]. CNN pruning can be either unstructured and structured. Unstructured pruning removes low-contributing weights, creating sparse weight matrices but potentially causing irregular memory access patterns, which can be difficult to optimize on some hardware. Structured pruning, on the other hand, removes entire kernels, resulting in more compact architectures [9]. Pruning reduces parameters and computational load, making models more suitable for real-time deployment on GPUs or mobile devices.

While adversarial robustness and efficiency have been studied in CNNs, limited research exists that combines these two areas for object detection. This work addresses this by integrating adversarial robustness into efficient object detection frameworks. The main contributions of this work are:

(1) **Adaptation of HYDRA for Object Detection:** This work extends HYDRA [10], a robust pruning framework for image classification, to object detection using the YOLOv3 model [11]. Object detection is more complex than image classification as it requires precise localization along with classification of multiple objects.
(2) **Evaluation of Pruning Techniques for Object Detection:** We studied structured and unstructured pruning methods to analyze their impact on model size, computational complexity, and mAP performance [12]. Additionally, we evaluated the performance of pruned models on both colored and greyscale images.
(3) **Use of DeepLIFT for Importance Scoring:** We integrated the DeepLIFT method to attribute feature contributions to object detection models, enhancing pruning decisions through better initialization.

2. Background

2.1. Object Detection

Object detection involves classifying and localizing objects within an image or video. For each object, its class, confidence score, and bounding box need to be determined [13]. Object detection requires accurate prediction of both the object class and its spatial location, addressing challenges like scale variation, occlusion, and inter-class similarities.

Numerous approaches have been proposed to address object detection challenges. Traditional methods rely on handcrafted features like HOG or Haar-like features with classifiers such as SVM or Random Forests [14], but these methods struggle with complex scenes and scalability. In contrast, convolutional neural networks (CNNs) have significantly improved object detection by learning hierarchical features from raw image data, eliminating manual feature extraction and handling variations in pose, scale, and occlusion. CNNs generalize well across diverse datasets, achieving robustness and good accuracy [15].

CNN models for object detection consist of a feature extraction backbone and a detection head for localization and classification. Models like Faster R-CNN [16] use a two-stage approach, first generating region proposals and then refining them for classification and localization, while others like the YOLO models [11] predict bounding boxes and class labels in a single pass. YOLOv3 improves over previous versions of YOLO by using multi-scale detection and a more powerful backbone for better speed and accuracy.

Datasets are crucial for training object detection models by providing annotated images for model training and evaluation. Existing datasets include Pascal VOC, ImageNet, Open Images [17], and MS COCO, one of the most widely used benchmarks [18]. COCO excels with 80 object classes, including people, animals, vehicles, and household items, ensuring broad generalization. Each image in COCO is annotated with instance-level bounding

Figure 1. An example of COCO dataset annotation format

boxes and class labels, and the dataset includes over 200,000 images for robust training and unbiased evaluation. An example image from COCO is shown in Figure 1.

In object detection, mean Average Precision (mAP) is a key metric for evaluating model performance. mAP is the mean of Average Precision (AP) for each class, ranging from 0 to 1, which is derived from the area under the precision-recall curve at various confidence thresholds [19]. Precision is defined as $\frac{\text{TP}}{\text{TP+FP}}$, and recall is $\frac{\text{TP}}{\text{TP+FN}}$, where TP, FP, and FN are true positives, false positives, and false negatives, respectively. True positive detections are evaluated using Intersection over Union (IoU) [12], which measures the overlap between bounding boxes. mAP is calculated from model outputs at multiple IoU thresholds (0.5 to 0.95) in increments of 0.05 for the COCO dataset, assessing model performance under varying confidence levels.

In this work, we measure the efficiency of our object detection model by inference time, as it directly impacts real-world performance, especially in real-time or low-latency applications. Inference time indicates how quickly a model can make predictions, which is crucial for time-sensitive tasks like autonomous driving and video-based detection. A model with short inference time can process more data in less time, improving throughput.

2.2. **Adversarial Attack**

Adversarial attacks manipulate neural networks by introducing small, often imperceptible perturbations that cause incorrect predictions [20]. These attacks expose vulnerabilities in applications like autonomous driving (misclassifying traffic signs), facial recognition (misidentifying individuals), and medical diagnosis (altering disease predictions).

Numerous techniques have been developed to generate adversarial attacks on CNNs, varying in complexity, cost, and effectiveness. The Fast Gradient Sign Method (FGSM) [21] is a simple, fast attack that adjusts inputs along the gradient of the loss function. Projected Gradient Descent (PGD) [22] is an iterative method that applies FGSM repeatedly with small steps, creating more robust adversarial examples. The Carlini & Wagner (C&W) [23] attack is an optimization-based method known for generating subtle yet effective perturbations.

2.3. **Efficient Adversarial Robustness**

While many studies focus on either improving CNNs' efficiency or robustness to adversarial attacks, few combine both. Vaddadi et al. [24] showcased an efficient CNN model optimized for adversarial robustness, achieving high classification accuracy and effective

resistance to adversarial attacks. Another work from Wijayanto et al. [25] explored the vulnerability of CNNs, particularly compressed models used in mobile devices, and investigated methods to enhance their robustness without sacrificing accuracy. Ye et al. [26] presented a framework that combines adversarial training with weight pruning to achieve model compression without sacrificing robustness. Gui et al. [27] introduced an Adversarially Trained Model Compression (ATMC) framework, integrating pruning, factorization, and quantization within a unified optimization structure to achieve compact, adversarially robust models without significant accuracy loss.

Sehwag et al. [9] evaluated CNNs' robustness under structured and unstructured pruning, defining a procedure that includes pre-training, weight pruning, and fine-tuning. Their method achieves a 10× compression ratio with 92.5% robust accuracy on adversarial examples, with unstructured pruning shown to be more effective.

HYDRA, also by Sehwag et al. [10], further enhances CNNs' performance through pruning by integrating robust training objectives with the empirical risk minimization (ERM) framework. In HYDRA, ERM guides pruning by removing less important weights based on minimizing error under adversarial attacks. The robust training objectives aim to improve adversarial resilience, minimizing worst-case loss within a defined perturbation range [28]. To achieve this, HYDRA utilizes an importance score-based optimization. Scaled initialization initializes scores based on receptive field size, input channels, and weight magnitude, which promotes faster convergence and enhanced performance of SGD compared to random initialization. Extensive experiments on three different datasets and four robust training objectives demonstrate HYDRA's ability to achieve state-of-the-art benign and robust accuracy at high pruning ratios. The HYDRA pipeline can be described in five steps:

(1) **Pre-train**: Train the CNN on adversarial examples to minimize a loss objective, formulated by integrating a robust training objective with ERM.
(2) **Initialize scores**: Assign floating point importance scores to each model weight using scaled initialization.
(3) **Minimize loss**: Freeze CNN weights and update importance scores during loss minimization. Run predictions using only weights with highest scores, but update all scores through backpropagation.
(4) **Prune**: Prune less important weights using the frozen importance scores. Create a binary mask to retain the top weights based on their scores.
(5) **Fine-tune**: Re-train the pruned network with the robust objective to adapt to structural changes and improve performance.

HYDRA was chosen as the foundation for our work due to its strong performance and comprehensive documentation. It secured second place in the auto-attack robustness benchmark [29] and has been cited over 100 times since publication.

Current research on efficient and adversarially robust CNNs often focuses on image classification, a simpler task than object detection. While classification models are easier to prototype and evaluate, they do not address the complexities of object detection, which involves multi-object recognition and localization [14]. No research has fully tested these methods on object detection, highlighting a gap in our understanding of their real-world performance, which this work aims to address.

3. Methodology

3.1. Adaptation of HYDRA for Object Detection

HYDRA was originally developed for image classification, focusing on recognizing and categorizing objects in an image. However, object detection is more complex as it requires not only identifying objects but also localizing them through bounding boxes, highlighting

Figure 2. COCO dataset image 22192: Ground truth (left), YOLOv3 on original image (middle), YOLOv3 on PGD perturbed image (right)

the spatial arrangement of objects within an image. To adapt HYDRA for object detection, we replaced its CNN models with YOLOv3, using weights pre-trained on the COCO dataset to save training time. Several modifications were made to the pruning pipeline to accommodate the object detection framework, ensuring compatibility with YOLOv3's architecture.

First, we restructured HYDRA's dataloader to handle the COCO dataset, which includes multiple object classes and bounding box annotations per image. Annotations were parsed to extract bounding box coordinates, dimensions, and class labels. Images were augmented by padding and resized to 416x416 pixels, ensuring compatibility with YOLOv3's default input size [11].

Next, we enhanced output processing. YOLOv3 divides an image into a grid, with each cell predicting multiple bounding boxes, confidence scores, and class probabilities for objects within that cell [11]. Predictions with confidence scores below 0.5 were filtered out. Then, Non-Maximum Suppression (NMS) [30] was applied to eliminate redundant bounding boxes, especially when multiple boxes overlap with high IoU, ensuring only the best predictions are retained.

Detection outputs containing image ID, predicted bounding box coordinates, confidence score, and predicted class label are saved in CSV format. mAP performance is calculated using the pycocotools library [18], which compares predicted bounding boxes with ground truth annotations at multiple IoU thresholds. The mAP across these thresholds provides a robust measure of YOLOv3's detection accuracy, capturing performance across various object sizes and overlaps.

Due to the limitations of verifiable robust methods like MixTrain and CROWN-IBP with larger networks, we opted for iterative adversarial training to evaluate robustness. We used mAP as the metric for object detection performance.

3.2. **PGD Attack**

To evaluate the robustness of our model, we used the Projected Gradient Descent (PGD) attack [22]. PGD refines the FGSM method by iteratively applying bounded perturbations in the direction of the gradient. After each step, the perturbation is projected back within a specified range (ε) to maintaining perceptual similarity, using a clip function that trims the changes of pixel values to ε if they exceed it. PGD's iterative process make it effective at generating stable adversarial examples, thus serving as the benchmark for evaluating our model's robustness. Figure 2 illustrates a PGD attack on an image from the COCO dataset.

Implementing the PGD attack on YOLOv3 generates adversarial perturbations that cause misdetections or missed objects. The algorithm for applying PGD to YOLOv3 is outlined in Algorithm 1. This algorithm ensures that perturbations remain imperceptible to humans while degrading model accuracy.

Algorithm 1 PGD Attack on YOLOv3 Object Detection

Require: Image I_0, model f, loss function $\mathcal{L}$, step size $\alpha = 0.02$, bound $\varepsilon = 0.05$, number of steps $n = 5$

Ensure: Adversarial image I'

Normalize pixel values of I_0 to range $[0, 1]$
Initialize adversarial image $I' \leftarrow I_0$
for $i = 1$ to n **do**
    Compute gradient: $g \leftarrow \nabla_{I'} \mathcal{L}(f(I'), I_0)$
    Update image: $I' \leftarrow I' + \alpha \cdot g$
    Project perturbations: $I' \leftarrow \text{clip}(I', I_0 - \varepsilon, I_0 + \varepsilon)$
end for
return I'

3.3. New DeepLIFT Initialization Technique

The scaled initialization method in HYDRA improves over random initialization for determining parameter weight importance [10]. However, other methods could estimate how much each parameter weight contributes to the final output. DeepLIFT (Deep Learning Important FeaTures) [31] is a method that explains neural network outputs by attributing differences in output to changes in input features relative to a defined reference input. It calculates contribution scores for each feature by measuring their impact on the difference of the output from a reference state through specific rules like the RevealCancel rule. The RevealCancel rule is designed to handle situations where positive and negative contributions from inputs interact, potentially cancelling each other out in non-linear layers. By separating and calculating the effects of positive and negative components independently, RevealCancel captures dependencies between inputs that might otherwise be overlooked [31]. This approach offers a more efficient and interpretable means of understanding neural network predictions compared to traditional gradient-based methods, making it ideal for importance score initialization in our pipeline.

The DeepLIFT implementation leverages the Captum library, a model interpretability and understanding library for PyTorch [32]. We use DeepLIFT to replace step 2 in the original HYDRA pipeline by optimizing importance scores. With DeepLIFT, each weight in the network is assigned an attribution score, a measure of how much it contributes to the model's final output relative to a reference input. The DeepLIFT scores are used to generate a binary mask that determines which weights or kernels should be pruned, targeting those with minimal contribution to the network. This process replaces traditional importance score initialization for pruning, as the pruning decision is made based on clear, interpretable scores that indicate each weight's significance.

3.4. Implementing Unstructured and Structured Pruning

In addition to the unstructured pruning used in HYDRA, we implemented structured pruning for a comparative analysis. Structured pruning can yield greater reductions in model size and computational complexity while preserving the network's architecture. The comparison highlights the trade-offs between pruning strategies in terms of accuracy, robustness, and efficiency.

We implemented structured pruning by focusing on reducing entire kernels rather than individual weights. Specifically, structured pruning evaluates the importance of each kernel in convolutional layers, and makes pruning decisions at the higher kernel level instead of the lower, more granular weight level. The importance of each kernel is calculated by averaging the importance of all its weights. This means that we modified the pruning step in the

HYDRA pipeline, by applying a binary pruning mask at the kernel level. An additional step is added to process the kernels' importance, and ensure that all the weights inside each kernel have the same value in the binary mask. The mask is a binary tensor that matches the shape of the network's weight tensor, with values indicating whether a weight should be pruned (0) or retained (1).

4. **Experiments**

We conducted experiments with five distinct configurations to evaluate the effects of different pruning strategies and initialization methods. The first four configurations use color COCO images: (1) unstructured pruning with DeepLIFT initialization, (2) unstructured pruning with HYDRA initialization, (3) structured pruning with DeepLIFT initialization, and (4) structured pruning with HYDRA initialization. These configurations allow for a comparison of the performance of the different initialization methods and pruning strategies (Sections 4.2 and 4.3). The fifth configuration, structured pruning with DeepLIFT initialization, uses COCO images converted to greyscale, enabling a comparison of the method's performance on colored versus greyscale images (Section 4.4).

For each of the 5 configurations, 10 experimental runs were conducted at each of 9 chosen pruning ratios: 0%, 25%, 50%, 65%, 75%, 85%, 90%, 95%, and 99%, to provide sufficient granularity in the data. Each model was evaluated using pycocotools to calculate the resulting mAP performance, and the average inference time per image was also measured.

One-way ANOVA (Analysis of Variance) was used to find any statistically significant differences between results. The p-value threshold was set at 0.05, so the null hypothesis is rejected when then p-value is smaller than 0.05.

A random subset of 50,000 images from the COCO dataset was used for training in the pruning pipeline, while the remaining 150,000+ images were used for testing to evaluate the performance of the pruned model. All experiments were conducted on a NVIDIA GeForce RTX 2070 SUPER GPU.

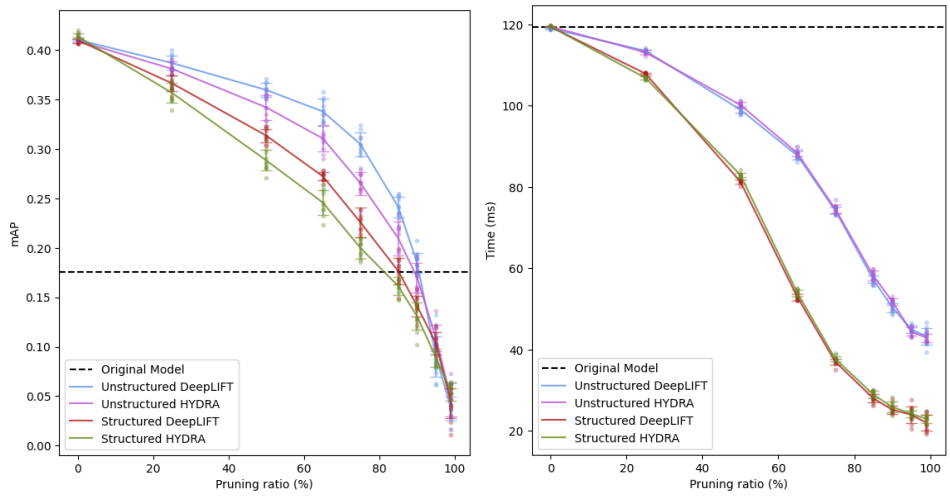
4.1. **Preliminary Findings**

We present mAP and inference time of different pruning configurations on the testing dataset in Figures 3 and 4. Each curve represents a configuration, showing the average mAP or time across pruning ratios. Averages are calculated by taking the mean of 10 independent runs per pruning ratio, with each run depicted in the figures as a dot. Standard deviations are shown as error bars, reflecting the variability due to the inherent randomness in model training and pruning, as SGD updates are based on randomly selected mini-batches.

The black dashed lines in the figures represent the performance of the original YOLOv3 model under the same PGD attack, achieving a mAP of 0.176, which is significantly lower than that of models with 0% pruning. This disparity arises because the latter models undergo additional adversarial pre-training in step 1 of the pipeline, enhancing their resilience. In terms of inference speed, the original model processes an image in 119.3 ms in average, similar to adversarially trained models with no pruning (0%). This similarity in inference time is due to minimal structural changes, as no weights are removed in either the original or unpruned adversarially trained models, resulting in consistent computational workload.

Figures 3a and 4a show that the mAP performance of pruned models decreases as the pruning rate increases, due to the removal of increasingly more important parameters that reduce the model's capacity to learn and retain critical information for accurate detection.

The inference time curve in Figures 3b and 4b shows an "S" shape. This is because early pruning stages cause only slight reductions in inference time as few parameters are removed; however, as pruning increases, the network shrinks, leading to a more significant decrease in inference time. Furthermore, beyond a certain threshold, aggressive pruning introduces



(a) mAP *(b)* Inference time

Figure 3. Comparison of DeepLIFT versus HYDRA initialized models with both unstructured and structured pruning, through different pruning ratios. A detailed view of the high ratio region is provided in Appendix **??**

sparsity, making computation less efficient and causing inference time to plateau due to hardware bottlenecks like memory access.

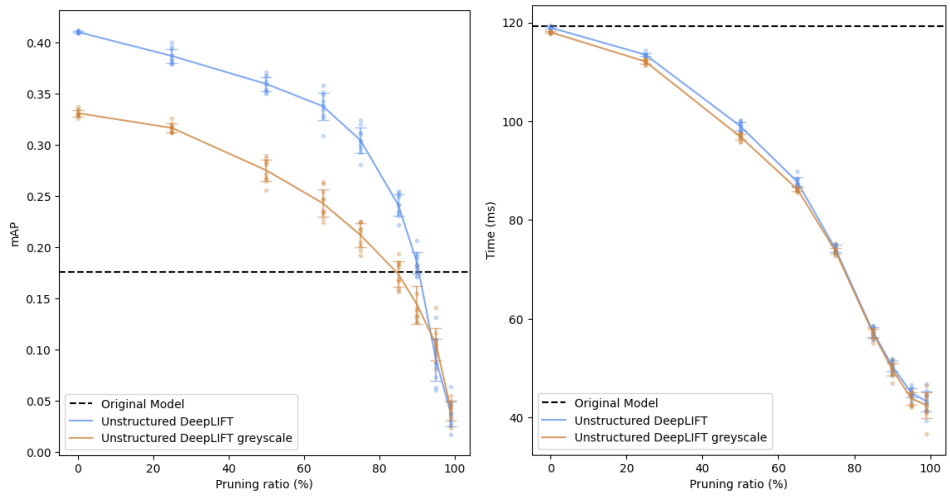
In [10], the HYDRA authors focused on pruned performance at 90%+ pruning rates. However, when adapting HYDRA from image classification to object detection, the performance drop at higher pruning rates (beyond 60%) is more significant. This is due to the complexity of object detection, which requires detailed spatial and contextual information to detect and localize objects. A more detailed mAP plot for pruning rates 85%+ is shown in Appendix **??** Figure **??**, where all methods exhibit similar low mAP values.

4.2. **DeepLIFT vs Scaled Initialization**

Figure 3a compares mAP performance between models pruned using DeepLIFT versus scaled initialization, DeepLIFT consistently outperforms scaled initialization within the 50%-85% pruning ratio range for both pruning methods. This is statistically validated by one-way ANOVA tests, all yielding p-values below the 0.05 threshold, as shown in Appendix **??**, Table **??**. DeepLIFT is more effective than scaled initialization for pruning as it directly calculates each weight's contribution to the final output, unlike scaled initialization, which infers importance partially from weight magnitude.

Scaled initialization overlooks the non-linear relationships between weights and predictions, where large weights may not always be crucial. DeepLIFT, by contrast, provides an attribution score that measures each weight's actual impact on the output, offering a more precise method than size-based importance. DeepLIFT targets weights that truly influence the model, ensuring that the most important components are preserved during pruning. This leads to a more efficient pruned model with higher performance, particularly in a complex task like object detection, where feature and spatial relationships are crucial.

When comparing the inference time between pruned models using the two different initialization methods in Figure 3b, there is no clear difference between the pruning methods. The pruning algorithms aim to reduce the network to a target pruning rate, and inference efficiency primarily depends on the model's size and structure rather than initial importance scores. ANOVA tests in Table **??** largely support this, with most p-values above 0.05.



(a) mAP *(b)* Inference time

Figure 4. Comparison of performance between color and greyscale images on unstructured pruned models initialized with DeepLIFT

Using DeepLIFT initialization increases the time to initialize importance scores compared to HYDRA's scaled initialization. Scaled initialization takes an average of 18 seconds, while DeepLIFT requires 49 seconds. DeepLIFT's longer runtime stems from its reference-based approach, which calculates the difference between each neuron's activation and a baseline, a more computationally intensive process than using the magnitude of weights.

4.3. **Unstructured vs Structured Pruning**

Figure 3a shows that unstructured pruning consistently outperforms structured pruning in mAP across pruning ratios from 25% to 90% for both initialization methods, as supported by the ANOVA p-values in Appendix **??**, Table **??**. Structured pruning typically results in lower mAP due to the removal of entire filters, channels, or layers, which impacts the model's ability to capture fine-grained features. This is particularly detrimental to object detection tasks that require detailed feature extraction across multiple scales, leading to a more significant performance drop in structured pruning.

While the initialization methods did not have a significant effect on the inference time performance, Figure 3b shows a significant difference between the two pruning methods. Starting from a 65% pruning ratio, structured pruned models require about half the time compared to unstructured models at the same pruning ratio. The one-way ANOVA tests in Table **??** support this finding. Structured pruning improves inference time by producing an optimized and regularized architecture, with fewer filters or layers and a consistent structure. This enables better memory access patterns and more efficient hardware processing, leading to faster computations, making structured pruning ideal for real-time applications, despite its potential impact on mAP.

4.4. **Effect of Greyscale on Performance**

We evaluated the pruning pipeline with greyscale images using the same training images as in previous experiments but converted to greyscale based on the Rec. 601 Standard. For these experiments, we used the unstructured DeepLIFT variant of the pruning pipeline, as it showed the highest mAP performance in earlier tests.

Figure 5. Differences between inference performance on COCO image 22192: top to bottom - 0%, 65%, and 95% pruning ratio; left to right - Unstructured DeepLIFT, Unstructured HYDRA, Structured DeepLIFT, Structured HYDRA, Greyscale Unstructured DeepLIFT.

There is little to no difference in inference time performance between the two methods. However, in Figure 4a, models trained with greyscale images exhibit significantly lower mAP compared to those trained with colored images, particularly at lower pruning ratios (0%-65%). However, as the pruning ratio increases, the performance gap narrows until the models' performance are indistinguishable at 95%+ pruning ratios. This stems from the loss of chrominance information in greyscale images. YOLOv3 relies on color to distinguish objects and capture texture, lighting, and shading variations essential for detection. Greyscale images only retain luminance, limiting the model's ability to differentiate objects with similar brightness but different colors. Without color information, it becomes challenging for the model to distinguish between objects like lemons and limes, as both share similar shapes and textures.

4.5. **Sample Images**

We applied PGD perturbations (ε=0.05) to sample COCO images and performed inference on models trained with different configurations and pruning rates. Appendix **??** includes the results, where each figure include 45 images across 9 pruning rates and 5 configurations. Figure 5 shows a subset of examples. Inference was performed on a random model from each of 10 experiment runs, where each detected object is labeled with a colored bounding box and class name.

Visual inspection shows that models pruned on greyscale images perform worse, with fewer correctly detected objects, consistent with mAP results. At low pruning rates (0% - 25%), objects are often correctly identified. At high pruning rates (90% - 99%), no objects are detected, explaining the mAP drop. At medium pruning rates, unstructured pruning with DeepLIFT initialization consistently outperforms other methods, detecting more objects and reducing misdetections. These observations align with mAP results, confirming DeepLIFT's superiority over HYDRA's scaled initialization.

5. **Conclusions**

This study adapted the HYDRA pruning framework to the YOLOv3 model, enhancing both efficiency and adversarial robustness in object detection. Evaluation between structured and unstructured pruning across various pruning levels revealed that unstructured pruning consistently outperformed structured pruning in preserving higher mean Average

Precision (mAP), especially at moderate pruning ratios, thus maintaining good detection accuracy in complex environments.

Moreover, using DeepLIFT for importance scoring refined the pruning process, targeting parameters with minimal output contribution. This enabled efficient pruning without compromising the model's ability to detect and localize objects accurately, making it suitable for real-time applications like autonomous driving and surveillance.

Future work could explore hybrid pruning techniques that combine structured and unstructured pruning for optimal trade-offs between performance and efficiency. Dynamic pruning ratios, where pruning levels adjust based on a target mAP value, could be useful in real-world deployment. Lastly, testing unstructured pruning with DeepLIFT initialization on models like Faster R-CNN or SSD, with diverse adversarial attacks, and evaluating them on domain-specific datasets, could provide insights into generalizability and adaptability.

References

[1] Z.-Q. Zhao, P. Zheng, S. tao Xu, and X. Wu. *Object Detection with Deep Learning: A Review*. 2019. arXiv: 1807.05511 [cs.CV]. URL: https://arxiv.org/abs/1807.05511.

[2] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye. *Object Detection in 20 Years: A Survey*. 2023. arXiv: 1905.05055 [cs.CV].

[3] D. Bhatt, C. Patel, H. Talsania, J. Patel, R. Vaghela, S. Pandya, K. Modi, and H. Ghayvat. "CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope". In: *Electronics* 10.20 (2021).

[4] P. Kohli and A. Chadha. "Enabling Pedestrian Safety Using Computer Vision Techniques: A Case Study of the 2018 Uber Inc. Self-driving Car Crash". In: *Advances in Information and Communication*. Springer International Publishing, 2019, 261–279.

[5] H. Zhang and J. Wang. *Towards Adversarially Robust Object Detection*. 2019. arXiv: 1907.10310 [cs.CV].

[6] J. C. Costa, T. Roxo, H. Proença, and P. R. M. Inácio. *How Deep Learning Sees the World: A Survey on Adversarial Attacks & Defenses*. 2023. arXiv: 2305.10862 [cs.CV].

[7] D. Ghimire, D. Kil, and S.-h. Kim. "A Survey on Efficient Convolutional Neural Networks and Hardware Acceleration". In: *Electronics* 11.6 (2022).

[8] G. Habib and S. Qureshi. "Optimization and acceleration of convolutional neural networks: A survey". In: *Journal of King Saud University - Computer and Information Sciences* 34.7 (2022), pp. 4244–4268.

[9] V. Sehwag, S. Wang, P. Mittal, and S. Jana. *Towards Compact and Robust Deep Neural Networks*. 2019. arXiv: 1906.06110 [cs.LG]. URL: https://arxiv.org/abs/1906.06110.

[10] V. Sehwag, S. Wang, P. Mittal, and S. Jana. *HYDRA: Pruning Adversarially Robust Neural Networks*. 2020. arXiv: 2002.10509 [cs.CV]. URL: https://arxiv.org/abs/2002.10509.

[11] J. Redmon and A. Farhadi. *YOLOv3: An Incremental Improvement*. 2018. arXiv: 1804.02767 [cs.CV]. URL: https://arxiv.org/abs/1804.02767.

[12] A. Sharma. *Mean Average Precision (mAP) Using the COCO Evaluator*. 2022. URL: https://pyimagesearch.com/2022/05/02/mean-average-precision-map-using-the-coco-evaluator/ (visited on 04/22/2024).

[13] Z.-Q. Zhao, P. Zheng, S.-T. Xu, and X. Wu. "Object Detection With Deep Learning: A Review". In: *IEEE Trans on NN and Learning Systems* 30.11 (2019), pp. 3212–3232.

[14] B. R. Solunke and S. R. Gengaje. "A Review on Traditional and Deep Learning based Object Detection Methods". In: *2023 International Conference on Emerging Smart Computing and Informatics (ESCI)*. 2023, pp. 1–7.

[15] N. O'Mahony, S. Campbell, A. Carvalho, S. Harapanahalli, G. V. Hernandez, L. Krpalkova, D. Riordan, and J. Walsh. "Deep Learning vs. Traditional Computer Vision". In: *Advances in Computer Vision*. Ed. by K. Arai and S. Kapoor. 2020, pp. 128–144.

[16] S. Ren, K. He, R. Girshick, and J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV]. URL: https://arxiv.org/abs/1506.01497.

[17] S. S. A. Zaidi, M. S. Ansari, A. Aslam, N. Kanwal, M. Asghar, and B. Lee. *A Survey of Modern Deep Learning based Object Detection Models*. 2021. arXiv: 2104.11892 [cs.CV].

[18] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. *Microsoft COCO: Common Objects in Context*. 2015. arXiv: 1405.0312 [cs.CV].

[19] E. Zimmermann, J. Szeto, J. Pasquero, and F. Ratle. *Benchmarking a Benchmark: How Reliable is MS-COCO?* 2023. arXiv: 2311.02709 [cs.CV]. URL: https://arxiv.org/abs/2311.02709.

[20] Y. Li, M. Cheng, C.-J. Hsieh, and T. C. M. Lee. "A Review of Adversarial Attack and Defense for Classification Methods". In: *The American Statistician* 76.4 (Jan. 2022), 329–345.

[21] I. J. Goodfellow, J. Shlens, and C. Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML]. URL: https://arxiv.org/abs/1412.6572.

[22] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: 1706.06083 [stat.ML]. URL: https://arxiv.org/abs/1706.06083.

[23] N. Carlini and D. Wagner. *Towards Evaluating the Robustness of Neural Networks*. 2017. arXiv: 1608.04644 [cs.CR]. URL: https://arxiv.org/abs/1608.04644.

[24] S. Vaddadi, S. e Vadakkethil Somanathan Pillai, S. R. Addula, R. Vallabhaneni, and B. Ananthan. "An efficient convolutional neural network for adversarial training against adversarial attack". In: *Indonesian Journal of Electrical Engineering and Computer Science* 36 (2024), pp. 1769–1777.

[25] A. W. Wijayanto, J. J. Choong, K. Madhawa, and T. Murata. "Towards Robust Compressed Convolutional Neural Networks". In: *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. 2019, pp. 1–8.

[26] S. Ye, K. Xu, S. Liu, J.-H. Lambrechts, H. Zhang, A. Zhou, K. Ma, Y. Wang, and X. Lin. *Adversarial Robustness vs Model Compression, or Both?* 2021. arXiv: 1903.12561 [cs.CV]. URL: https://arxiv.org/abs/1903.12561.

[27] S. Gui, H. Wang, C. Yu, H. Yang, Z. Wang, and J. Liu. *Model Compression with Adversarial Robustness: A Unified Optimization Framework*. 2019. arXiv: 1902.03538 [cs.LG]. URL: https://arxiv.org/abs/1902.03538.

[28] S. Wang, Y. Chen, A. Abdou, and S. Jana. *MixTrain: Scalable Training of Verifiably Robust Neural Networks*. 2018. arXiv: 1811.02625 [cs.LG]. URL: https://arxiv.org/abs/1811.02625.

[29] F. Croce and M. Hein. *Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks*. 2020. arXiv: 2003.01690 [cs.LG]. URL: https://arxiv.org/abs/2003.01690.

[30] J. Hosang, R. Benenson, and B. Schiele. *Learning non-maximum suppression*. 2017. arXiv: 1705.02950 [cs.CV]. URL: https://arxiv.org/abs/1705.02950.

[31] A. Shrikumar, P. Greenside, and A. Kundaje. *Learning Important Features Through Propagating Activation Differences*. 2019. arXiv: 1704.02685 [cs.CV]. URL: https://arxiv.org/abs/1704.02685.

[32] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, B. Alsallakh, J. Reynolds, A. Melnikov, N. Kliushkina, C. Araya, S. Yan, and O. Reblitz-Richardson. *Captum: A unified and generic model interpretability library for PyTorch*. 2020. arXiv: 2009.07896 [cs.LG]. URL: https://arxiv.org/abs/2009.07896.