Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Optimization of YOLO-NAS for Object Detection Through Pruning Techniques

Zainab Noor Ahmad, Said Jadid Abdulkadir, Arsh Obeidy, Maryam Omar Abdullah Sawad

Computer and Information Sciences Department Universiti Teknologi PETRONAS
Perak, Malaysia
Email: zainab_22009877@utp.edu.my

The rapid advancements in deep learning have led to increasingly complex Convolutional Neural Networks (CNNs), resulting in greater storage demands and slower inference speeds. Models like You Only Look Once – Neural Architecture Search (YOLO-NAS) have achieved remarkable results in object detection, but their high computational requirements make implementation in resource-limited environments challenging. To optimize the YOLO-NAS model for such limited conditions, this work investigates the efficacy of many pruning approaches. Using the PASCAL VOC dataset, we trained the YOLO-NAS model from scratch. To lower its computational complexity, we then used a variety of pruning techniques, such as weight pruning, channel pruning, and filter pruning. Our experimental results reveal a reduction in the model's computational demands with minimal impact on its object detection capabilities. Pruning algorithms were implemented, and the result was a reduction in FLOPs (Floating-Point Operations) without sacrificing competitive object identification performance. The findings show that reducing FLOPs with prolonged fine-tuning after pruning produced a negligible loss in accuracy.
Keywords: YOLO, Pruning, Object detection.

1. Introduction

Object detection is a crucial aspect of computer vision, with applications including security systems, autonomous vehicles and image analysis. One notable model in this field is the You Only Look Once Neural Architecture Search (YOLO-NAS) model, which has gained considerable recognition for its sophisticated design and use of neural architecture search techniques. This model excels in object detection by autonomously finding efficient network structures. However, the intricate design of YOLO-NAS leads to significant resource demands, making it difficult to implement in real-time applications and on devices with

constrained resources.

To address these challenges, researchers have investigated model compression techniques to minimize the computational demands and size of deep neural network models, including the YOLO-NAS architecture. The goal of model compression is to optimize resource usage while maintaining the performance and effectiveness of the model. Pruning and quantization are two techniques that have been used to lower parameters, leverage low-rank structures, and transfer knowledge.

This research focuses on exploring model pruning to simplify the YOLO-NAS model. Pruning is an intricate process that demands a careful evaluation of trade-offs and optimization strategies. Achieving a balance between resource efficiency and preserving the core features of the YOLO-NAS model is essential to ensure it keeps a high accuracy in object detection.

Model pruning is a technique used to reduce the trained model's size by removing unnecessary or redundant parameters. The goal is to create a smaller, more efficient model that maintains or even improves performance on specific tasks. During training, a neural network assigns different weights to its parameters to recognize patterns and make correct predictions. However, not all parameters are equally important; some may have a minimal impact or be redundant, contributing little to the overall accuracy. Model pruning identifies redundant parameters while minimizing the effect on performance. Pruning may be performed at multiple levels, including individual weights, neurons, or entire layers, depending on the technique employed.

Given the intricate nature of model pruning, careful attention must be paid to the balance between reducing computational overhead and maintaining the essential features that contribute to high object detection accuracy. The process involves several techniques, each with its unique advantages and trade-offs. For instance, weight pruning focuses on eliminating individual weights with minimal impact on performance, while structured pruning, including channel and filter pruning, targets larger components of the network, leading to reductions in model size and complexity.

This study delves into the application of these pruning techniques specifically on the YOLO-NAS model to understand their efficacy in various scenarios. By training the YOLO-NAS model from scratch on the PASCAL VOC dataset and subsequently applying different pruning methods, the research aims to quantify the impact on both computational efficiency and detection performance. The results from this comprehensive evaluation provide insights into the optimal pruning strategies that can be employed to enhance the deployment of YOLO-NAS in re-source-constrained environments.

The findings indicate that pruning, particularly when followed by extended fine-tuning, can significantly reduce the model's computational demands, such as Floating-Point Operations (FLOPs), while maintaining competitive object detection accuracy. This demonstrates the potential of pruning as a viable solution for optimizing advanced deep learning models like YOLO-NAS for practical applications.

Pruning techniques, especially when applied to models like YOLO-NAS, aim to strike a balance between reducing computational costs and maintaining the model's detection performance. Traditional pruning methods, including weight, channel, and filter pruning, have

proven effective in simplifying neural networks, but their full potential in YOLO-NAS has yet to be explored. The novelty of this work lies in applying multiple pruning strategies to YOLO-NAS, analyzing their individual and combined effects on both model complexity and accuracy.

This study contributes to the field by demonstrating how pruning techniques can be systematically applied to modern, complex architectures like YOLO-NAS, filling a gap in the literature where no previous work has explored pruning in YOLO-NAS. This research paves the way for more efficient deployment of object detection models in resource-constrained environments, a crucial step towards real-world, practical applications of deep learning models.

2. Related Work

CNN-based deep learning models are vital for computer vision tasks, especially object detection. However, they face high computational costs and memory limitations, which restrict their real-time application use. Therefore, several techniques have been developed by model compression to compress and speed up CNN models.

Pruning [1], Quantization [2] , and Knowledge distillation[3] are examples of compression methods. Pruning re-moves weights or neurons that have minimal impact on model ac- curacy. Identifying components for removal should be based on quantitative metrics like absolute values, norms, correlation matrices, and Hessian matrices. Less critical components, once ranked, are pruned according to a sparsity measure, often expressed as a percentage. For example, K% sparsity means discarding K% of weight. Pruning can target several structural elements like nodes, connections, channels, filters, layers, or combinations, reducing model size, memory, and computational load. The impact of pruning on model accuracy varies across various levels, so selecting the appropriate pruning level is crucial to balance effectiveness, ease of implementation, and performance across different hardware.

TABLE 1

YOLO version	references	Pruning applied	Baseline accuracy	Pruned accuracy
v2	[4,5]	Filter	86.8%	85.4%
v3	[6–14]	Filter	94.6%	93.5%
		Channel	23.8 mAP	23.1mAP
v4	[15–23]	Channel	81.03%	80.53%
		Filter	45.4mAP	44.7mAP
v5	[24–37]	Channel	94.93mAP	94.72mAP
YOLO-NAS	There is no work related to YOLO-NAS pruning.			

Such choices enhance operational speed, especially with specialized hardware accelerators processing the pruned model efficiently.

Object detection models, like the popular You Only Look Once (YOLO) architecture, have revolutionized computer vision by enabling real-time and accurate object detection. However,

these models often have large processing needs and memory footprints, making them difficult to implement on resource-constrained devices and in real-time applications. To overcome these drawbacks in YOLO models, model pruning strategies have drawn more attention. Pruning YOLO models aims to reduce network size and complexity while maintaining detection capabilities. However, there is limited research specifically on model pruning for YOLO architectures, and none on the newer YOLO-NAS architecture. This section reviews the existing literature on YOLO model pruning techniques, highlighting its contributions and setting the stage for our proposed method to prune the YOLO-NAS model while preserving its detection abilities.

Table 1 summarizes various pruning techniques applied to different versions of the YOLO (You Only Look Once) model and their impact on baseline and pruned accuracy, showing that filter and channel pruning are commonly used approaches. For YOLOv2, filter pruning, and channel pruning were applied, resulting in a slight decrease in accuracy from 86.8% to 85.4%. In YOLOv3, both filter pruning and channel pruning were used, with baseline accuracies of 94.6% and 23.8 mAP, respectively. The pruned accuracies for YOLOv3 were 93.5% and

23.1 mAP. YOLOv4 only utilized channel pruning, which led to a minor reduction in accuracy from 81.03% to 80.53%. For YOLOv5, filter pruning, and channel pruning were applied, showing baseline accuracies of 45.4 mAP and 94.93 mAP. After pruning, the accuracies were slightly reduced to 44.7 mAP and 94.72 mAP, respectively, demonstrating that pruning leads to slight reduction in the accuracy. Lastly, for YOLO- NAS, there is no existing work related to pruning as indicated in the table. This suggests that pruning techniques have not been applied or studied for the YOLO-NAS model version yet.

3. Methodology

According to this methodology, the YOLO-NAS model is trained on a custom dataset from scratch before the network is subjected to various pruning methods.

Dataset

The benchmark dataset for object detection used in this study is PASCAL VOC, which has 20 classes and 11,540 images. The dataset is split into validation and training sets. Validation set is further divided with an 80:20 ratio into validation and testing sets for testing purposes.

Model

This study is based on the YOLO-NAS model, an innovative technique in computer vision, specifically designed for object detection. NAS (Neural Architecture Search) in YOLO involves automated exploration of machine learning algorithms searching for optimal network architecture. This improves performance by identifying extremely effective designs that are customized to meet object detection requirements.

Pruning Techniques

This study presents the use of three major pruning methods on the YOLO-NAS model: weight pruning, filter pruning, and channel pruning. These techniques aim to lower the memory footprint and computational complexity of the model without significantly compromising its

accuracy.

Weight Pruning. A granular technique called weight pruning eliminates specific weights in the neural network according to their size. We employ a threshold-based approach using the YOLO-NAS model. When the absolute value of a weight falls below a certain threshold, it is considered as unimportant and has been set to zero. Iteratively completing this procedure is followed by network fine-tuning.

Channel Pruning. Channel pruning is implemented at the

channel level of feature maps. Channels that have the least impact on the output of layers that follow are removed. The L1 norm of channel weights is one of the metrics used to assess each channel's relevance. The process of pruning channels that have the lowest scores reduces the total computational cost as well as the dimensionality of the following layer's input.

Sparsity training, the initial phase of channel pruning, at- tempts to make the weights of unimportant channels converge to zero, preserving important information in a few channels. This is achieved using the Batch Normalization (BN) layer, which helps identify the most important channels by balancing each channel's data handling ability. By doing this, crucial channels are identified, and the network becomes less crowded. Equation (1) shows the feature normalization formula for the BN layer where y_"in " and y_"out " indicate the input and output of the BN layer, respectively.

$$y_{out} = \gamma \times \frac{y_{in} - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta \quad (1)$$

While γ and β are scale factors and biases that may be changed during training, μ and σ^2 represent the mean and variance of the small batch input feature maps. Additionally, ϵ is a small value used to keep the denominator from being divided by zero. During training, the L1 regularization term is used to specifically alter γ. Network sparsity may be improved by combining it with the neural network's training procedure to reconstruct the loss function, as demonstrated by Equation. (2)

$$L = L_{yolo} + \alpha \sum_{\gamma \in \tau} f(\gamma) \quad (2)$$

where the second term is the L1 regularization, and L_{yolo} indicates the YOLO loss function. In this case, $f(\gamma) = \gamma$, and the penalty term α is used to balance the two- loss term. This penalty function helps the network to focus more on the essential channels by concentrating on the weight distribution, while simultaneously optimizing the loss function in the standard manner.

Filter Pruning. Filter pruning focuses on simplifying the model by removing convolutional filters considered less significant. This method evaluates the importance of each filter using the concept of the geometric median, which serves as a robust measure of central tendency like the arithmetic mean but based on geometric properties. The Filter Pruning via Geometric Median (FPGM) technique uses this approach by finding the geometric median of the weights for each filter within a convolutional layer. In FPGM, the first step involves calculating the geometric median, represented as a vector, which signifies the central location of the filter's weights. Filters with a smaller geometric median are considered to have lower importance and

contribute less to the overall representational ability of the layer. Mathematically, this process is described as finding point m that minimizes the sum of Euclidean distances to all weight vectors v_i in the dataset.

$$m = \arg\min_{m} \sum_{i=1}^{n} \|v_i - m\|_2 \quad (3)$$

The point m represents the geometric median we aim to identify. The variable "vi" denotes the individual vectors in our dataset. The expression $\sum_{i=1}^{n} \|v_i - m\|_2$ calculates the total distance between each vector and the geometric median point. The notation $\arg\min_{m}$ signifies the process of finding the value of m that minimizes this total distance, thereby determining the geometric median.

To quantify the significance of each filter, the Filter Importance Score (FIS) is calculated using the geometric median of the filter's weights:

$$\text{FIS}(F) = \|GM(W_F)\|_2 (4)$$

W_F is the weights of a specific filter F within a convolutional layer. The function $GM(W_F)$ computes the geometric median of these weights.

Once the Filter Importance Scores are determined for all filters in a layer, a pruning threshold τ is applied to decide which filter to retain or remove. Filters with an FIS below the threshold τ are pruned, effectively reducing the complexity of the network by eliminating less crucial filters.

TABLE 2

	Before pruning	Weight pruning	Channel Pruning	Filter pruning
Number of parameters	66905371	66905371	66905371	62003021
Model File Size (bytes)	344070627	344141434	344140289	264695233
FLOPS	7912411220	7912411220	7912411220	7912400210

4. Experiments and Evaluations

A. Configuration

The learning rate is set to 0.0005 at the beginning, the batch size is 8, and the number of epochs is set at 100. The pruning ratio for filter pruning is 0.2, whereas for channel pruning it is set to 0.3. The model undergoes 50 training cycles, known as epochs, using the identical configuration as the initial training, in the fine-tuning phase.

B. Metrics

The model's performance is assessed in this work using four metrics: FLOPS, mean average precision (mAP), trainable parameters, and model file size.

C. Pruning impact on network size

Prior to pruning, the model exhibited an extremely dense architecture with minimal sparsity, characterized by a substantial file size and a high parameter count. This configuration resulted in considerable computational demands, as indicated by the high number of Floating-Point Operations (FLOPS). These metrics highlight the model's inherent complexity, and the significant computational effort needed to process data. The detailed outcomes of the pruning process are summarized in Table 2.

Following channel pruning, the model's file size experienced a slight increase, which can be attributed to the overhead involved in storing the pruned network structure or associated metadata. Interestingly, the number of trainable parameters remained unchanged, indicating that the overall complexity of the model was preserved. However, there was a marked increase in sparsity, suggesting that numerous operations were deemed unnecessary. This in-crease in sparsity has the potential to enhance runtime efficiency without reducing the FLOPS, thus maintaining a steady computational cost per operation.

Weight pruning similarly resulted in a minor increase in file size, for reasons akin to those observed with channel pruning. Although the total parameter count remained stable, the significant rise in sparsity indicated that many weights were zeroed out. This heightened level of sparsity signifies a substantial reduction in the effective computational burden, even though the nominal FLOPS remained constant, suggesting potential efficiency gains in scenarios where sparse computations can be leveraged.

In contrast, filter pruning led to a reduction in both the model's file size and the number of parameters, effectively.

demonstrating its capacity to streamline the network. This re- duction in parameters directly contributes to a more simplified model architecture. Additionally, there was a slight decrease in the model's computational load (FLOPS), from 7,912,400,220 to 7,912,400,210, and represented a marginal 0.00014% reduction. Despite the model becoming smaller and less complex, this minor reduction in FLOPS implies that the improvements in speed and energy consumption might be minimal.

D. Extended Fine Tuning

Training pruned networks for further epochs greatly enhances their performance. We performed an additional 75 epochs of pruning on trained networks using each strategy. As Table 3 illustrates, fine-tuning can partially recover the mAP performance that was lost due to pruning.

TABLE 3

		mAP (IoU@0.5)
Initial		0.40
Pruned	Channel	0.32
	Channel Fine-tuned	0.35
	Weight	0.29
	Weight Fine-tuned	0.33

Filter	0.28
Filter Fine-tuned	0.36

5. Conclusion

In this study, we evaluated the effectiveness of various pruning techniques to lessen the YOLO-NAS model's computational cost. The outcomes show how various pruning methods affect neural network optimization. Without changing the amount of parameters or computational cost, channel and weight pruning dramatically boosted model sparsity, indicating that these techniques improve memory efficiency without directly affecting computational efficiency. On the other hand, filter pruning decreased the size of the model and the number of parameters, providing more significant computational cost savings, with a slight decrease in the model's computational load (FLOPS). These results highlight the tradeoffs that must be made between preserving model performance and increasing efficiency via pruning. In the future, other pruning methods will be investigated and their effects on computing efficiency and model performance will be assessed as genetic algorithm-based pruning and pruning using evolutionary algorithms.

Future research will focus on refining the pruning process by incorporating hyperparameter optimization to better control the trade-offs between sparsity and accuracy. Additionally, exploring genetic algorithms and evolutionary pruning techniques could lead to more adaptive pruning strategies, further reducing computational demands while maintaining performance. Applying these methods to other YOLO versions, particularly those for embedded systems, could enable efficient real-time object detection on resource-constrained devices.

References

1. P. Singh, V. K. Verma, P. Rai, and V. P. Namboodiri, "Play and prune: Adaptive filter pruning for deep model compression," arXiv, 2019. [Online]. Available: https://arxiv.org/abs/1905.04446.
2. R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," 2018. [Online]. Available http://arxiv.org/abs/1806.08342.
3. G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: : http://arxiv.org/abs/1503.02531.
4. Z. Huang, L. Li, and H. Sun, "Global biased pruning considering layer contribution," IEEE Access, vol. 8, pp. 173521–173529, 2020 DOI: 10.1109/ACCESS.2020.3025130.
5. N. Q. Truong, Y. W. Lee, M. Owais, D. T. Nguyen, G. Batchuluun, T. Nguyen, G. Batchuluun, T Pham, and K. R. Park, "SlimDeblurGAN-based motion deblurring and marker detection for autonomous drone landing," Sensors, vol. 20, no. 14, p. 3918, 2020. DOI: 10.3390/s20143918.
6. Y. Liu, Z. Wang, X. Wu, and F. Fang, "Cloud-edge-end cooperative detection of wind turbine blade surface damage based on lightweight deep learning network," IEEE Internet Computing, vol. 27, pp. 43–51, 2023. DOI: 10.1109/MIC.2022.3175935.
7. J. Zhai, B. Li, S. Lv, and Q. Zhou, "FPGA-based vehicle detection and tracking accelerator," Sensors, vol. 23, no. 4, p. 2208, 2023
8. Y. Pi, N. D. Nath, S. Sampathkumar, and A. H. Behzadan, "Deep learning for visual analytics of the spread of COVID-19 infection in crowded urban environments," Natural Hazards Review, vol. 22, 2021. DOI: 10.1061/(ASCE)NH.1527-6996.0000492
9. Z. Wang, H. Li, X. Yue, and L. Meng, "Design and acceleration of field programmable gate array-based deep learning for empty-dish recycling robots," Applied Sciences, vol. 12, no. 14, p. 7337,

2022. DOI: 10.3390/app12147337

10. Y. Chen, R. Li, and R. Li, "HRCP: High-ratio channel pruning for real- time object detection on resource-limited platform," Neurocomputing, vol. 463, pp. 155–167, 2021. DOI: 10.1016/j.neucom.2021.08.046
11. C. Chen, B. Liu, S. Wan, P. Qiao, and Q. Pei, "An edge traffic flow detection scheme based on deep learning in an intelligent transportation system," IEEE Transactions on Intelligent Transportation Systems, vol. 22, pp. 1840–1852, 2021. DOI: 10.1109/TITS.2020.3025687.
12. Z. Li, X. Liu, Y. Zhao, B. Liu, and Z. Huang, "A lightweight multi- scale aggregated model for detecting aerial images captured by UAVs," Journal of Visual Communication and Image Representation, vol. 77, p. 103058, 2021. DOI: 10.1016/j.jvcir.2021.103058.
13. H. Xu, M. Guo, N. Nedjah, J. Zhang, and P. Li, "Vehicle and pedestrian detection algorithm based on lightweight YOLOv3-Promote and semi-precision acceleration," IEEE Transactions on Intelligent Transportation Systems, vol. 23, pp. 19760–19771, 2022. DOI: 10.1109/TITS.2021.3137253.
14. S. Chen, R. Zhan, W. Wang, and J. Zhang, "Learning slimming SAR ship object detector through network pruning and knowledge distillation," IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 14, pp. 1267–1282, 2021. DOI: 10.1109/JS-TARS.2020.3041783.
15. M. H. K. Khel, K. A. Kadir, S. Khan, M. Noor, H. Nasir, N. Waqas, and A. Khan, "Realtime crowd monitoring—Estimating count, speed and direction of people using hybridized YOLOv4," IEEE Access, vol. 11, pp. 56368–56379, 2023. . DOI: 10.1109/ACCESS.2023.3272481.
16. C. Zhu, C. Wu, Y. Li, S. Hu, and H. Gong, "Spatial location of sugarcane node for binocular vision-based harvesting robots based on improved YOLOv4," Applied Sciences, vol. 12, no. 6, p. 3088, 2022. DOI: 10.3390/app12063088.
17. M. Tian, X. Li, S. Kong, L. Wu, and J. Yu, "A modified YOLOv4 detection method for a vision-based underwater garbage cleaning robot," Frontiers of Information Technology & Electronic Engineering, vol. 23, pp. 1217–1228, 2022. DOI: 10.1631/FITEE.2100473