Thesis for the Degree of Masters in Science

Gated YOLO: Dynamic Channel Gating for Efficient Object Detection

School of Computer Science and Engineering

The Graduate School

Hector Andres Acosta Pozo

June 2024

The Graduate School

Kyungpook National University

Gated YOLO: Dynamic Channel Gating for Efficient Object Detection

Hector Andres Acosta Pozo

School of Computer Science and Engineering

The Graduate School

Supervised by professor Soon Ki Jung

Approved as a qualified thesis of Hector Andres Acosta Pozo

for the degree of Masters of Science

by the Evaluation Committee

June 2024

Chairman: Prof. Seok Joo Koh

Prof. Soon Ki Jung

Prof. Yongtae Kim

The Graduate School

Kyungpook National University

Contents

Acknowledgement

I would like to express my deepest appreciation to Professor Soon Ki Jung for his invaluable guidance, patience, and expertise throughout my research. His insights and encouragement were crucial to the success of this work.

I am deeply grateful to my family, whose unwavering support and belief in me have been my constant source of strength and motivation. To my mother, Soneyda Pozo, and my father, Andres Acosta, thank you for your endless love, encouragement, and sacrifices, which have not gone unnoticed. Your belief in my abilities has been a significant driving force in my pursuit of academic excellence.

I also wish to extend a special thanks to my sister, Diana Acosta, for her support, understanding, and companionship. Your presence and belief in my work have been a great source of comfort and encouragement.

To all of you, I am eternally grateful for your support and love. Thank you for being my guiding lights.

List of Tables

List of Figures

Abstract

In the landscape of object detection, the YOLO (You Only Look Once) methodology has revolutionized real-time applications with its single-pass detection capabilities. However, traditional YOLO architectures often process vast amounts of data, leading to unnecessary computational overhead and limiting deployment in resource-constrained environments. Our research introduces "Gated YOLO," an adaptation of the YOLO framework, incorporating a gating mechanism to enhance computational efficiency while maintaining detection accuracy.

This method integrates a mechanism that adjusts the activation of neural pathways based on the relevance to the observed scene. During the training phase, our approach identifies and deactivates consistently inactive neural pathways across specific environmental conditions. This strategic deactivation reduces redundant computational weight, making the model more efficient for its designated tasks.

Our empirical results demonstrate that our approach significantly reduces computational load with minimal impact on detection accuracy, elevating processing speeds. This makes it an effective solution for real-time object detection in fixed, resource-constrained environments, such as security camera systems.

In summary, Gated YOLO represents a significant advancement towards more resource-efficient object detection solutions. By tailoring model processing pathways to specific environmental demands, this research enhances the

applicability and effectiveness of real-time object detection systems in static settings.

1 Motivation and Objectives

1.1 Introduction

Object detection is a foundational pillar within computer vision, influencing applications ranging from advanced surveillance systems to autonomous driving technologies. The rapid evolution of deep learning methodologies has significantly propelled the field forward, enhancing the accuracy, efficiency, and adaptability of object detection in real-time processing environments. Among these innovations, the YOLO (You Only Look Once) architecture, introduced by Redmon et al. [1], has enabled efficient processing without iterative detection stages. This architecture underscores the potential of modern object detection systems, from high-powered servers to constrained devices such as smartphones and embedded systems like Nvidia's Jetson Platform.

Despite the advancements brought by YOLO and its iterations, challenges remain, particularly in edge computing where computational resources are limited and real-time processing is essential. Recognizing the potential for further optimization, our study explores YOLOv6 [2], [3], a variant designed with a focus on hardware efficiency and real-time applications. YOLOv6's architecture [2], [3] serves as an ideal foundation for our investigation due to its performance in hardware-constrained environments.

Our research introduces "Gated YOLO," a novel framework that integrates dynamic gating with model pruning, specifically tailored to the YOLO architecture. Our approach maintains high modularity, facilitating portability to other YOLO architectures.

A cornerstone of our methodology is the adaptation of Improved SemHash [4], initially proposed by Kaiser and Bengio and further refined by Chen et al. [5]. This technique generates binary gates during model training, enabling selective activation or deactivation of network filters. Through dynamic gate generation and analysis tailored to specific scenes, our method identifies filters that remain consistently inactive. These filters are statically pruned for deployment, allowing the model to operate efficiently by focusing computational resources on active, scene-relevant pathways. The Gater Network [5], integral during training for gate determination, is rendered unnecessary during deployment, replaced by pre-determined, statically applied gates that ensure efficiency and specificity in detection.

Our contributions to the YOLO architecture, through the integration of a gating network and Improved SemHash [4], enable precise control over network activity and significant improvements in detection accuracy. Rigorous experimental validation, focusing on metrics such as inference time in milliseconds (ms), recall, precision and mean Average Precision (mAP@0.5:0.95), reveals a notable decrease in inference time for the Gated YOLO model compared to similar sized counterparts, with stable mAP scores.

1.2 Motivation

The relentless pursuit of advancements in computer vision, particularly in object detection, is driven by the escalating demands of modern applications. Deep learning architectures like YOLO have significantly narrowed the gap between theoretical possibility and practical implementation, offering a glimpse into the

potential of real-time object detection systems. However, as these technologies are increasingly deployed on the edge, the limitations of current models under resource-constrained conditions become apparent. Our work aims to transcend these limitations and push the boundaries of existing object detection frameworks.

Our focus on YOLOv6 [2], [3], known for its balance of speed and accuracy, stems from the critical need for optimization in edge computing scenarios where resources are scarce yet the demand for high-performance computing is high. The drive to refine and enhance the efficiency of such models without compromising their detection capabilities underlines our research. We are particularly inspired by the potential impact of our work on applications requiring real-time analysis and feedback.

1.3 Objectives

The primary objective of our research is to develop an optimized version of the YOLOv6 [2], [3] architecture, termed "Gated YOLO," which incorporates a gating mechanism to enhance computational efficiency in object detection tasks, particularly in edge computing environments. To achieve this, we aim to:

Implement Gating Mechanism: Integrate a gating mechanism that selectively activates relevant neural pathways, improving model efficiency.

Optimize Through Model Pruning: Apply model pruning techniques to eliminate redundant parameters and streamline the model, focusing

computational resources on critical tasks.

Leverage Improved SemHash [4]: Utilize the Improved SemHash technique for effective gate generation during training, allowing for precise control over network activity.

Demonstrate Practical Efficacy: Validate the effectiveness of the Gated YOLO model through extensive testing, focusing on key performance metrics such as inference time in milliseconds (ms), recall, precision and mean Average Precision (mAP@0.5:0.95).

Through these objectives, our research seeks to address the challenges of deploying sophisticated object detection models in edge computing scenarios, offering a pathway to more efficient, accurate, and accessible real-time object detection technologies.

2 Related Works

The exploration of efficiency within neural network architectures, particularly for object detection in computationally constrained environments, is a significant area of research. This section reviews various methodologies and developments that have shaped the current landscape of efficient neural network design, highlighting the relevance and novelty of our approach.

2.1 Sparsity and Conditional Computation

A fundamental concept in neural network efficiency is the integration of sparsity and conditional computation. Sparsity involves activating only a subset of neural pathways for a given input, reducing computational overhead without sacrificing the model's ability to represent complex functions. This principle is akin to decision trees, where decisions lead to a subset of possible states, thus not exploring all branches for a given input.

The concept of conditional computation extends this idea further by introducing mechanisms that allow a neural network to adapt its computation pathways dynamically based on the input. This adaptability ensures that only the most relevant parts of the network are engaged during the forward pass, which not only saves computational resources but also helps in reducing over-fitting by limiting the effective capacity of the model based on the complexity of the input.

Conditional computation extends this idea by dynamically engaging only the most relevant parts of the network during the forward pass, saving computational resources and reducing over-fitting. Introduced by Bengio et al.

[6], this mechanism allows for deeper and more complex models by judiciously allocating computational resources.

Several approaches have been proposed to implement sparsity and conditional computation. For example, Chen et al. introduced GaterNet [5], where a gating network generates binary gates to selectively activate filters in the backbone network based on input. Their experiments on CIFAR and ImageNet datasets show significant improvements in model performance and efficiency.

Veit and Belongie [7] propose adaptive inference graphs, dynamically selecting parts of the network to use for each input, reducing computational requirements. Liu et al.'s Dynamic Sparse Training (DST) [8] represents a leap from static sparsity models, dynamically adjusting network architecture during training to optimize model capacity and resource allocation.

Kalwar et al. [9] explore object detection in adverse weather conditions using gated image processing, highlighting the adaptability and efficiency of gated architectures, presenting potential comparative studies for our gated approach.

2.2 Neural Network Pruning

Neural network pruning reduces model complexity by eliminating redundant weights, enhancing computational efficiency without significantly sacrificing accuracy. This technique alleviates storage and computational burdens, leading to faster inference times and reduced energy consumption, crucial for deployment on resource-constrained devices.

Gradient-based Pruning. Zhang et al. [10] leverage gradient descent to systematically identify and eliminate less critical connections, preserving model performance while streamlining its architecture.

Iterative Pruning and Retraining. Han et al. [11] propose an iterative approach, periodically removing weights deemed least important, followed by retraining to maintain performance. This method achieves substantial model compression while maintaining high accuracy with fewer parameters.

Both gradient-based and iterative pruning methods enhance model efficiency but require careful calibration and additional development steps, such as determining optimal pruning thresholds and managing retraining processes.

2.3 Quantization Techniques for Neural Network Efficiency

Quantization reduces numerical parameter precision from floating-point to lower-bit representations, significantly reducing model size and computational complexity while maintaining acceptable accuracy. Krishnamoorthi [12] provides an overview of quantization techniques, including post-training quantization and quantization-aware training.

Efficient Integer-Arithmetic-Only Inference. Jacob et al. [13] introduce a framework for efficient inference in integer-arithmetic-only environments, enabling the use of optimized hardware accelerators.

Hardware-Aware Automated Quantization. Wang et al. [14] propose HAQ, a method using reinforcement learning to determine optimal mixed precision quantization policies based on target hardware characteristics.

While quantization methods offer significant improvements in model efficiency, they often require careful tuning of quantization parameters to avoid substantial accuracy loss. Additionally, the static nature of most quantization techniques means that once a model is quantized, its ability to dynamically adjust to varying computational resources or input complexities is limited.

2.4 Knowledge Distillation for Model Efficiency

Knowledge distillation transfers knowledge from a larger, complex model (teacher) to a smaller, efficient model (student). Hinton et al. [15] demonstrate that a smaller model can mimic a larger model's output, enabling high performance on resource-limited devices. Zhang et al. [16] explore self-distillation, where a single network serves as both teacher and student, simplifying the distillation process and improving performance.

Knowledge distillation, while effective, often relies on pre-trained large-scale models, which can be a limitation in scenarios where such models are unavailable or computationally prohibitive.

2.5 Our Contribution

The pursuit of efficiency in neural network models has prompted innovations such as quantization, knowledge distillation, sparsity, and neural network pruning. These methods aim to optimize performance within computational constraints by reducing parameter precision, transferring knowledge from larger to smaller models, and eliminating less critical connections. However, these

techniques often encounter limitations, such as the need for a large teacher model in knowledge distillation or the loss of adaptability in static pruning and quantization methods.

Our approach integrates gating mechanisms with model pruning, specifically designed for the YOLO architecture. Unlike conventional strategies focusing on static model compression, our method uses Improved SemHash to create static gating configurations after training. This innovation substantially decreases computational demands during inference while maintaining high accuracy.

Building on neural network efficiency advancements, we introduce the "Gated YOLO." This method incorporates gating and model pruning principles, customized for the YOLO architecture. Our technique for establishing static gating configurations post-training is unique in the literature. This strategy significantly reduces computational requirements during inference, distinguishing our work in the real-time object detection landscape. Through comprehensive comparisons with state-of-the-art models, we highlight the practical benefits and superiority of our approach, especially in resource-constrained scenarios.

3 Methodology

Our study introduces an innovative approach to optimize object detection within the constraints of limited computational resources, specifically through the development of the Gated YOLO architecture. This methodology leverages a dynamic gating mechanism, a novel adaptation within the established YOLO framework, to enhance the model's efficiency and adaptability by selectively processing only the neural pathways relevant to the observed scene. The primary focus of this section is to delineate the systematic approach employed in the design, implementation, and validation of this architecture. We outline the steps taken to integrate dynamic gating with the YOLO architecture, including the development and deployment of the Gater Network, the modifications applied to the YOLO architecture for accommodating gating mechanisms, and the analytical methods used to assess the model's performance in varied scene-specific contexts. The subsequent paragraphs will detail each component of our methodology, elucidating the technical strategies employed to achieve a balance between computational efficiency and detection accuracy in resource-constrained environments.

3.1 Gater Network

The cornerstone of our Gated YOLO model is the Gater Network, a novel component designed to enhance the model's computational efficiency by dynamically modulating the activation of neural pathways based on the specific features of the input scene. This is achieved by the generation of gates influenced by the scene salient characteristics. Utilizing the principles of conditional

computation, the Gater Network strategically deactivates certain parts of the neural network that are deemed irrelevant for a given scene, thereby reducing unnecessary computational overhead with minimal compromise to the model's object detection capabilities.

The Gater Network operates in two main phases: gate generation during training and gate application during inference. During the training phase, the network learns to identify and generate a set of binary gates based on the distinguishing features of the input images. These gates serve as indicators for activating or deactivating specific channels within the YOLO architecture, depending on their relevance to the task at hand. The process leverages Improved Semantic Hashing, a technique inspired by the work of Kaiser and Bengio [4] and further explored by Chen et al. [5], to ensure that the gate generation is both efficient and effective.

Gate Generation and Application. In practice, the Gater Network employs a specialized architecture, typically based on a lightweight convolutional neural network (CNN) like the ResNet family of light weight variations [17], to process input images and extract relevant features. These features are then passed through a series of fully connected layers, culminating in the generation of binary gates. Each gate corresponds to a specific channel or filter in the subsequent layers of the YOLO architecture, dictating whether it should be activated (1) or deactivated (0) based on the input scene's characteristics. An overview on how the GaterNet generates these gates can be seen in Fig.1.

During inference, the pre-determined gates are applied to the YOLO architecture, enabling the model to focus its computational resources only on the

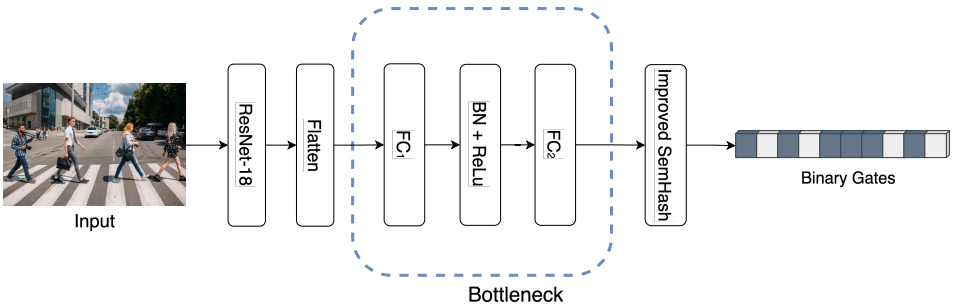

Fig. 1: Illustration of the GaterNet architecture. The GaterNet extract features using ResNet-18 and process the output to generate a set of binary gates.

parts of the network that are essential for detecting objects in the current scene. This selective processing significantly enhances the model's efficiency, particularly in scenarios where computational resources are limited, such as edge devices.

Dynamic Adaptation to Scene Features. One of the key advantages of the Gater Network is its ability to dynamically adapt to various scenes without the need for manual tuning. By analyzing the input scene and applying the appropriate gates, the network ensures that only the most relevant features are processed. This adaptability is crucial for applications such as surveillance and traffic monitoring, where the scene's characteristics may vary significantly but remain consistent over time.

In summary, the Gater Network introduces a significant advancement in the YOLO architecture by incorporating a dynamic gating mechanism that intelligently deactivates filters based on the unique features of each input scene. This approach not only enhances computational efficiency but also maintains high detection accuracy, demonstrating the potential of selective gating in

improving real-time object detection applications.

3.1.1 Feature Extraction

The feature extraction process within the Gater Network plays a pivotal role in our Gated YOLO model, laying the foundation for the subsequent gating mechanism by identifying the critical features from the input images. This process can make use of any DNN like the ResNet family. In our experiments we defaulted to ResNet-18 [17] for its renowned efficiency and effectiveness in capturing salient features from images with minimal computational resources.

ResNet-18 for Efficient Feature Representation. ResNet-18 is chosen for its shallow architecture compared to deeper variants, striking an optimal balance between computational efficiency and the ability to extract rich, discriminative features. This balance is crucial for our model's application in resource-constrained environments, where maintaining high accuracy without excessive computational burden is essential. The feature extraction is formalized as follows in equation (1):

$$F_{\text{extract}}(x) = \text{Flatten}(\text{AdaptivePool}(f_{\text{net}}(x))), \tag{1}$$

where $x \in \mathbb{R}^{c_0 \times h_0 \times w_0}$ represents the input image, and $f_{\text{net}}(x)$ denotes the feature representation extracted by the ResNet-18 network. The $AdaptivePool$ operation ensures that the output from f_{net} is standardized, facilitating uniformity across different input dimensions. The final feature vector $F_{\text{extract}}(x)$ is obtained

by flattening the pooled features, making it suitable for further processing by the fully connected layers leading to gate generation.

Adaptive Pooling for Dimensionality Reduction. Adaptive pooling plays a crucial role in our feature extraction process by dynamically adjusting the size of the feature maps to a fixed dimension, thereby enabling a consistent input size for the fully connected layers regardless of the original input image size. This step is critical for ensuring that the feature extraction process remains efficient and scalable across varying image dimensions.

Feature Flattening for Gate Generation. After adaptive pooling, the feature map undergoes a flattening operation, transforming it into a one-dimensional vector suitable for analysis by the subsequent layers responsible for gate generation. This flattened feature vector encapsulates the essential information required for determining the relevance of specific neural pathways in the YOLO architecture, forming the basis for the dynamic gating mechanism.

3.1.2 Binary Gates

Binary gates within the Gater Network are pivotal for modulating the activity of neural pathways in the YOLO architecture, enabling selective processing based on the input scene's characteristics. These gates are generated through a series of operations that map the high-dimensional feature vector obtained from the feature extraction phase to a binary vector, where each element corresponds to a specific channel in the YOLO architecture. The generation and application of these binary

gates are fundamentally rooted in the concept of Improved Semantic Hashing [4], [5], which ensures the differentiability of the gating process during training while maintaining binary decisions during inference. As default for our implementation we sampled noise from the Gaussian distribution.

Mapping to Binary Space. The process begins with mapping the extracted features to the binary space. This mapping is achieved through a dual-layer architecture comprising two fully connected layers that introduce a bottleneck layer to efficiently manage the parameter space while ensuring the representational capacity:

$$f_0 = \text{ReLU}(\text{BatchNorm}(\text{FC1}(f))), \quad \text{and} \tag{2}$$

$$g_0 = \text{FC2}(f_0). \tag{3}$$

where in equation (2), f denotes the flattened feature vector, and f_0 represents the intermediate representation at the bottleneck. $FC1$ and $FC2$ are the two fully connected layers, with ReLU activation and Batch Normalization applied after the first layer to enhance training stability and non-linearity. Equation (3) maps f_0 to g_0, the pre-activation gate vector, poised for binary conversion.

Improved Semantic Hashing. The cornerstone of our binary gate generation process is the Improved Semantic Hashing technique, which allows the network to generate binary gates in a differentiable manner during training. This

adaptability is crucial for integrating the gating mechanism within the end-to-end training process of the YOLO architecture. The method involves adding a noise component to the pre-activation gate vector and applying a sigmoid function to obtain a soft binary gate:

$$g_{\text{noisy}} = g_0 + \epsilon, \quad \text{and} \tag{4}$$

$$g_\alpha(i) = \text{clamp}(1.2 \times \sigma(g_{\text{noisy}}(i)) - 0.1, 0, 1), \tag{5}$$

where ϵ represents a noise vector sampled from a Gaussian distribution in equation (4). g_{noisy} denotes the noisy gate vector, and g_α is the soft binary gate vector, with each element being clamped between 0 and 1 to ensure binary-like behavior as seen in equation (5). This approach facilitates the backpropagation of gradients during training, allowing for the optimization of the gating mechanism.

Binary Decision Making. During inference, the soft binary gates g_α are converted into hard binary decisions g_β, which directly control the activation of corresponding channels in the YOLO architecture, as shown in equation (6):

$$g_\beta(i) = \begin{cases} 1, & \text{if } g_\alpha(i) \geq \text{Gating threshold}, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

This binary decision-making process ensures that only the relevant features contributing to the object detection task are processed, thereby enhancing the

model's computational efficiency without compromising its detection performance.

In essence, the Binary Gates subsection outlines the sophisticated methodology employed to generate and apply binary gates within the Gater Network, leveraging Improved Semantic Hashing to marry the model's need for differentiability during training with the necessity for discrete decision-making during inference. This delicate balance enables the Gated YOLO model to dynamically adapt to various scene features, optimizing computational resources while slightly impacting the accuracy in object detection tasks.

3.1.3 Loss Function

The loss function of our Gated YOLO model is meticulously designed to optimize both object detection performance and the efficiency of the gating mechanism. It amalgamates multiple components to guide the training process towards achieving high accuracy in object detection while ensuring computational efficiency through effective gate generation. The overall loss function is formulated as follows in equation (7):

$$L = \alpha_{\text{cls}} L_{\text{cls}} + \alpha_{\text{iou}} L_{\text{iou}} + \alpha_{\text{dfl}} L_{\text{dfl}} + \lambda \cdot L_{\text{gate}}, \tag{7}$$

where L_{cls}, L_{iou}, and L_{dfl} represent the classification loss, Intersection over Union (IoU) loss, and distance-IoU loss, respectively. These components are weighted by their respective coefficients α_{cls}, α_{iou}, and α_{dfl}, ensuring a balanced contribution to the overall loss. L_{gate} signifies the gating loss, which encourages the sparsity

of the gating mechanism, and λ is the regularization coefficient controlling its influence on the total loss.

Classification and IoU Losses. The classification loss L_{cls} and IoU loss L_{iou} are fundamental to object detection models, ensuring accurate classification and localization of objects within the scene [2], [3]. The distance-IoU loss L_{dfl} further refines the bounding box predictions, enhancing the precision of object localization.

Gating Loss. The gating loss L_{gate} is pivotal in training the Gater Network, promoting the generation of efficient and effective gates. It is designed to encourage the model to minimize the number of active gates, thereby reducing the computational load during inference. The gating loss is formulated as an L_1 regularization term over the gate vector g, encouraging sparsity. This loss can be expressed in equation (8) as follows:

$$L_{\text{gate}} = \frac{1}{c}\|g\|_1 = \frac{1}{c}\sum_i |g_i|, \tag{8}$$

where c is the total number of gates, and g_i represents the individual gate values. By penalizing the sum of the absolute values of the gate vector, the model is encouraged to deactivate unnecessary channels, thereby streamlining the network for increased efficiency.

Balancing Detection Performance and Efficiency. The coefficients α_{cls}, α_{iou}, α_{dfl}, and the regularization term λ play crucial roles in balancing the model's

object detection performance with the efficiency of the gating mechanism. By adjusting these parameters, we can fine-tune the model to prioritize either detection accuracy or computational efficiency, depending on the application requirements.

Additionally, the gating mechanism's threshold, represented as `gtg_threshold`, influences the binary decision process in the gating mechanism. The gating loss L_{gate} is dynamically scaled using the `gtg_decay` parameter, which adjusts the weight of the gating loss over epochs to balance the trade-off between gate sparsity and detection performance.

In conclusion, the loss function of the Gated YOLO model is a comprehensive formulation that encompasses the dual objectives of maintaining high accuracy in object detection and optimizing computational efficiency through an effective gating mechanism. This strategic combination of loss components ensures that the model can be effectively trained to meet the demands of real-world object detection tasks in resource-constrained environments.

3.2 YOLO Architecture

In the design of our Gated YOLO model, we leverage the strengths of the YOLOv6 architecture, as proposed by Li et al. [2], [3], renowned for its exceptional real-time performance and optimal utilization of hardware resources. The YOLOv6 framework serves as the foundational backbone, onto which we have integrated the GaterNet that introduces a dynamic gating mechanism. This integration enables our model to perform selective feature processing,

substantially improving computational efficiency without sacrificing detection accuracy. The following figure illustrates the modified YOLO architecture augmented with the GaterNet functionality:

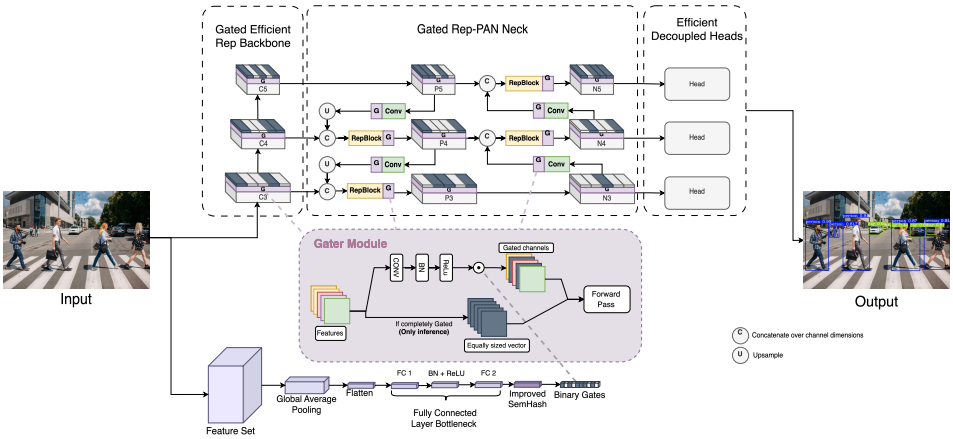

Fig. 2: Illustration of the YOLOv6 architecture enhanced with the GaterNet, featuring sections like "Gated Efficient Reparameterizable Backbone (EfficientRep)", "Gated Rep-Pan Neck", and "Efficient Decoupled Heads", each engineered for maximized performance and efficiency.

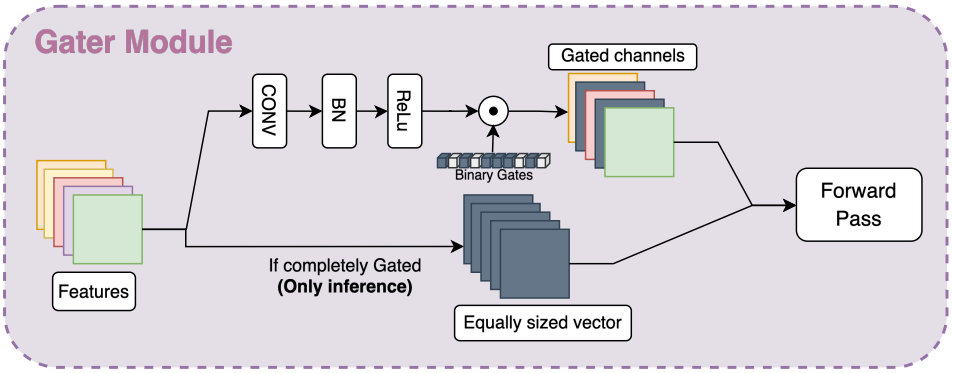

Fig. 3: Illustration of the Gater Module enhanced with the GaterNet, featuring the duality of paths that entails the completely gated layer and the partially gated layer processes.

The incorporation of GaterNet into the YOLO architecture facilitates a more selective and efficient processing approach, primarily through two key components:

Gated Efficient Reparameterizable Backbone: This component forms the core of our feature extraction mechanism. By employing gating mechanisms, the EfficientRep selectively emphasizes critical features while minimizing attention to redundant information. This selectivity ensures that subsequent processing layers focus computational resources on analyzing features of utmost relevance to the detection task at hand.

Gated Rep-Pan Neck: Inspired by the PANet topology and enhanced with Rep blocks for added efficiency, the Gated Rep-Pan Neck dynamically adjusts feature resolution and scale. It optimizes the integration and refinement of features passed from the EfficientRep to the detection heads, playing a vital role in ensuring the accuracy of object detection.

3.2.1 YOLO Gate Module

The YOLO Gate Module significantly enhances the YOLO architecture by integrating the dynamic gating functionality, offering a novel approach to managing feature processing both during training and inference phases. Through this module, the network learns to modulate its output by performing element-wise multiplication of the convolutional outputs with the gating signals generated by the GaterNet:

$$\mathbf{G} = [g_1, g_2, \ldots, g_c], \tag{9}$$

$$g_{\text{closed}} = \frac{\sum_{i=1}^{c} \mathbf{1}(G_i = 0)}{c}, \quad \text{and} \tag{10}$$

$$g(x) = \begin{cases} 0, & \text{if } g_{\text{closed}} > 0.99, \\ \text{Conv}(x) \odot \mathbf{G}, & \text{otherwise.} \end{cases} \tag{11}$$

where $\mathbf{G}$ represents the gate vector, indicating the activation status of corresponding feature maps within the network's layers. The module evaluates the proportion of active gates to decide between bypassing certain convolutional operations or allowing them, thus tailoring the network's processing power to the task's specific requirements. This selective gating mechanism, visualized in Figure 3, ensures optimal resource allocation during feature propagation, enhancing the model's overall efficiency and effectiveness in real-time object detection tasks.

3.3 Analysis Step

The analysis step represents a critical phase in our Gated YOLO methodology, where the model's adaptability and efficiency are fine-tuned for specific scene conditions. Through a detailed analysis process, we evaluate the model's performance over a designated scene for a set duration, allowing us to observe and record the operational status of the gating mechanism in real-time scenarios. This process involves a meticulous examination of the gating decisions made by the GaterNet for each frame processed, focusing on identifying which filters are essential for maintaining detection accuracy and which can be deactivated without compromising performance.

Monitoring Gating Decisions. The essence of this analysis lies in monitoring the closure rate of each gate across the network, a task accomplished by systematically tracking the activation state of each gate throughout the inference process:

$$\Gamma(t) = \{\gamma_1(t), \gamma_2(t), \ldots, \gamma_n(t)\}, \tag{12}$$

where $\Gamma(t)$ represents the set of gating states at time t, and $\gamma_i(t)$ indicates the state (active or inactive) of the i-th gate. By analyzing these states over time, we can identify patterns of gate usage that correlate with specific features or elements within the scene.

To facilitate a clearer understanding and interpretation of the gating decisions, we aggregate these decisions by network sections, as visualized in the following figure:

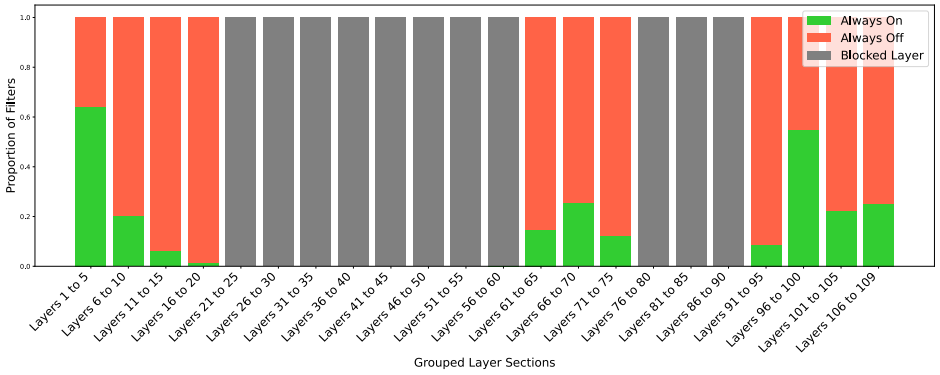

Fig. 4: Graphical representation of the distribution of gating states across various sections of the network, highlighting the sections with consistently active ("Always On") or suppressed ("Always Off") filters, as well as those that are completely deactivated ("Blocked Layer").

Distillation of Static Gating Configuration. The ultimate objective of this analytical step is to derive a static gating configuration that can be consistently applied to the designated scene and similar scenarios in future inferences. This involves discerning which gates remain predominantly active or inactive throughout the analysis period and categorizing them accordingly:

$$G_{\text{static}} = \text{distill}(\Gamma(t), \forall t \in T), \tag{13}$$

where G_{static} denotes the derived static gating configuration, and T is the duration of the analysis period. This static configuration enables the model to bypass the need for dynamic gate computations by the GaterNet in subsequent inferences, significantly reducing computational overhead.

4 Experiments and Results

This section details the experimental setup, datasets used, performance metrics, and results obtained from the implementation of our proposed gated neural network architecture. Our approach is compared with baseline models to highlight its effectiveness in various tasks. Our goal behind this experiments is to demonstrate our model's capability of discerning between specific characteristics from the scene in a way that it allows the generation of proper gates, particularly focussed on maintaining accuracy as much as possible while reducing the inference speed.

4.1 Experimental Setup

The experiments were conducted on a computational setup consisting of an 8th generation Intel i7 processor, 32GB RAM, and an NVIDIA RTX 2080 SUPER GPU. Our model was implemented using PyTorch 1.8. During training our learning rate started from $1e-3$, and a batch size of 64 was used across all experiments unless stated otherwise. The influence of the Gating Loss was modified to achieve the best balance between accuracy and inference speed. Lastly, all the samples used for training had a 480 by 480 spatial dimension which proved to be sufficient for our use cases.

Additionally, experiments based on practical implementation were conducted on edge like environments. Specifically, we implemented our approach and compared with other state of the art approaches on an Nvidia Jetson TX2 development board to evaluate the real world performance of our

model in something resembling a traffic surveillance camera system.

4.2 Datasets

Experiments were performed on two benchmarking datasets:

- **VOC 2012 [18]:** The Pascal VOC dataset contains images for classification, detection, and segmentation, encompassing 20 object categories.
- **ROAD-SEC:** A mixture of datasets composed of traffic survailance camera with the main purpose of detecting motor vehicles on roads. It contains 26,000 artificially annotated files divided into training, test, and validation samples.

4.3 Performance Metrics

The models' performance was evaluated using the following metrics:

- **Accuracy:** The proportion of correctly predicted observations to the total observations.
- **mean Average Precision (mAP):** Assesses the model's performance across various Intersection over Union (IoU) thresholds, offering a nuanced view of its detection capabilities.
- **Precision and Recall:** Precision is the ratio of correctly predicted positive observations to the total predicted positives, while recall (sensitivity)

measures the ratio of correctly predicted positive observations to all observations in actual class.

- **F1 Score:** The weighted average of Precision and Recall.

4.4 Baseline Models

Our approach was benchmarked against the following models for performance comparison:

- YOLOv5 [19]
- YOLOv6 [2], [3]
- YOLOX [20]

4.5 Results

4.5.1 Object Detection on VOC

On the VOC dataset, our model demonstrated remarkable performance improvements across various metrics, showcasing the effectiveness of the G-YOLO approach. The results, as presented in Table 1, highlight significant gains in computational efficiency and competitive accuracy metrics compared to existing state-of-the-art models.

The G-YOLO variants excelled in maintaining lower latency while achieving high mean Average Precision (mAP). For instance, the G-YOLOv6 N

model achieved an mAP of 52.5% with a latency of just 58ms, compared to the YOLOv5 N model, which has an mAP of 61.3% but a higher latency of 83ms. This reduction in latency underscores the efficiency of the G-YOLO models, making them highly suitable for real-time applications in resource-constrained environments.

Moreover, the G-YOLOv6 L variant achieved an impressive mAP of 70.6% with a latency of 240ms, which, while slightly higher in latency than some smaller models, still outperforms many in terms of efficiency and speed. Notably, the precision and recall values for the G-YOLO models are competitive, ensuring that the overall detection performance is not compromised. For example, G-YOLOv6 L attained a precision of 80.4% and a recall of 74.8%, leading to a robust F1 score of 77.5%.

4.5.2 Object Detection on ROAD-SEC

On the ROAD-SEC dataset, our model demonstrated significant improvements in both computational efficiency and detection accuracy, highlighting the strengths of the G-YOLOv6 approach. As detailed in Table 2, the G-YOLOv6 variants consistently outperformed existing models in terms of latency, making them highly suitable for real-time applications in resource-constrained environments.

For instance, the G-YOLOv6 N model achieved an impressive mAP of 79.3% with a latency of just 32ms. In comparison, the YOLOv5 N model, with a higher latency of 88ms, achieved a slightly higher mAP of 86.0%. This reduction in latency underscores the efficiency of the G-YOLOv6 models, demonstrating their ability to perform well under limited computational resources.

Table 1: Comparison of Object Detection Models on VOC

Model Variant	Latency	mAP@0.50:0.95	Precision	Recall	F1 Score
[19]YOLOv5 N	83ms	61.3%	70.2%	64.5%	67.2%
[19]YOLOv5 S	147ms	64.5%	73.9%	68.3%	70.9%
[19]YOLOv5 M	210ms	72.1%	80.4%	74.8%	77.5%
[19]YOLOv5 L	322ms	75.7%	84.3%	78.5%	81.3%
[3]YOLOv6 N	89ms	62.3%	71.6%	66.1%	68.7%
[3]YOLOv6 S	144ms	65.6%	75.4%	69.7%	72.4%
[3]YOLOv6 M	211ms	73.1%	82.1%	76.3%	79.1%
[3]YOLOv6 L	336ms	76.8%	85.7%	79.9%	82.7%
[20]YOLOX N	61ms	28.4%	29.8%	26.2%	27.9%
[20]YOLOX T	93ms	52.3%	61.4%	55.8%	58.5%
[20]YOLOX S	133ms	63.6%	72.7%	67.1%	69.8%
[20]YOLOX M	212ms	71.0%	79.3%	74.1%	76.6%
[20]YOLOX L	305ms	77.9%	85.6%	80.5%	83.0%
G-YOLOv6 N	58ms	52.5%	61.2%	55.4%	58.1%
G-YOLOv6 S	97ms	57.2%	66.8%	60.7%	63.6%
G-YOLOv6 M	165ms	64.6%	73.1%	67.4%	70.1%
G-YOLOv6 L	240ms	70.6%	80.4%	74.8%	77.5%

Furthermore, the G-YOLOv6 L variant achieved a high mAP of 85.8% with a latency of 126ms, while the YOLOv5 L variant, though achieving a higher mAP of 91.0%, had a significantly higher latency of 345ms. This clearly indicates that G-YOLOv6 models can deliver competitive accuracy with substantially lower latency, thus making them ideal for applications requiring fast and efficient object detection.

Additionally, the precision and recall values for the G-YOLOv6 models are noteworthy. For example, the G-YOLOv6 L variant achieved a precision of 92.1% and a recall of 87.5%, leading to an F1 score of 89.7%. These metrics highlight the balanced performance of G-YOLOv6 models, ensuring that high detection accuracy is maintained alongside reduced computational load.

Table 2: Comparison of Object Detection Models on RoadSec

Model Variant	Latency	mAP@0.50:0.95	Precision	Recall	F1 Score
[19]YOLOv5 N	88ms	86.0%	93.2%	88.5%	90.8%
[19]YOLOv5 S	135ms	88.7%	95.1%	90.3%	92.6%
[19]YOLOv5 M	201ms	89.2%	95.7%	90.8%	93.2%
[19]YOLOv5 L	345ms	91.0%	97.3%	92.5%	94.8%
[3]YOLOv6 N	85ms	84.8%	92.5%	87.3%	89.8%
[3]YOLOv6 S	137ms	85.6%	93.1%	87.8%	90.4%
[3]YOLOv6 M	222ms	87.3%	94.6%	89.2%	91.8%
[3]YOLOv6 L	321ms	87.8%	95.0%	89.6%	92.2%
[20]YOLOX N	54ms	62.0%	69.3%	64.8%	67.0%
[20]YOLOX T	87ms	78.0%	86.4%	80.2%	83.2%
[20]YOLOX S	107ms	82.6%	89.6%	84.3%	86.8%
[20]YOLOX M	194ms	86.0%	95.3%	89.8%	92.5%
[20]YOLOX L	265ms	87.3%	96.2%	90.7%	93.4%
G-YOLOv6 N	32ms	79.3%	87.4%	82.1%	84.7%
G-YOLOv6 S	44ms	83.6%	90.3%	85.6%	87.9%
G-YOLOv6 M	93ms	83.4%	90.1%	85.4%	87.7%
G-YOLOv6 L	126ms	85.8%	92.1%	87.5%	89.7%

4.6 Ablation Studies

4.6.1 Noise Distribution Strategy

This ablation study explores the impact of different noise distribution strategies on model performance. As shown in Table 3 we evaluated how variations in the applied noise affect the gating mechanism's effectiveness by experimenting with Gaussian, Uniform, and Gumbel noise distributions. The aim was to observe their influence on overall model accuracy and detection precision.

Table 3: Impact of Noise Distribution Strategies on Model Performance

Noise Strategy	Latency (%)	mAP@0.5 (%)	F1 Score (%)
Gaussian	Base	52.5	58.1
Uniform	+5	52.4	58.0
Gumbel Noise	+3	52.6	56.3

4.6.2 Feature Extractor

This section analyzes the performance variations when employing different feature extractors within the gated architecture. By integrating ResNet-18, ResNet-50, and ResNet-101 as the backbone for feature extraction, we aim to discern the optimal combination that maximizes both accuracy and efficiency, our findings can be evaluated in Table 4.

Table 4: Comparison of Feature Extractors in Gated Architecture

Feature Extractor	mAP@0.5 (%)	F1 Score (%)
ResNet-18	52.0	58.1
ResNet-50	52.1	58.1
ResNet-101	52.3	58.2

4.7 Discussion

Our experiments demonstrate the G-YOLOv6 models' superior balance of high detection accuracy and low latency. On the VOC dataset, G-YOLOv6 models showed significant latency reductions while maintaining competitive accuracy, with the G-YOLOv6 N achieving 52.5% mAP at 58ms, compared to YOLOv5 N's 61.3% mAP at 83ms.

On the ROAD-SEC dataset, G-YOLOv6 models again excelled, with the G-YOLOv6 N achieving 79.3% mAP at 32ms, versus YOLOv5 N's 86.0% mAP at 88ms. Precision and recall metrics were strong, indicating robust performance.

Ablation studies on noise distribution and feature extractors showed slight variations, with deeper networks offering marginal gains. Overall, G-YOLOv6 models effectively balance efficiency and accuracy, proving ideal for resource-constrained environments.

5 Conclusion

This research has demonstrated significant enhancements in real-time object detection through the integration of gating mechanisms within the YOLO framework. The “Gated YOLO” model addresses challenges in deploying deep learning models in resource-constrained environments by optimizing computational efficiency without sacrificing accuracy.

By focusing computational resources on pertinent aspects of the input data, the Gated YOLO reduces unnecessary operations, decreasing computational load and increasing processing speed. Our empirical evaluations show that the model often exceeds traditional YOLO architectures in efficiency.

The ability to maintain high detection accuracy while operating under reduced computational demands makes it ideal for real-time applications in edge computing platforms.

Future studies could refine the gating mechanisms to be self-adapting without retraining and explore integrating this architecture with other forms of conditional computation for more optimized models. The “Gated YOLO” represents a significant advancement in deep learning for object detection, providing a scalable, efficient solution that leverages the strengths of the YOLO architecture while mitigating its limitations.

Appendix A

This appendix presents detailed comparisons of latency and mean Average Precision (mAP) between various versions of YOLO models (YOLOv5, YOLOv6, YOLOX) and our proposed G-YOLO model. The comparisons are conducted on two datasets: the VOC dataset and the RoadSec dataset. The results are illustrated through bar graphs and line charts.

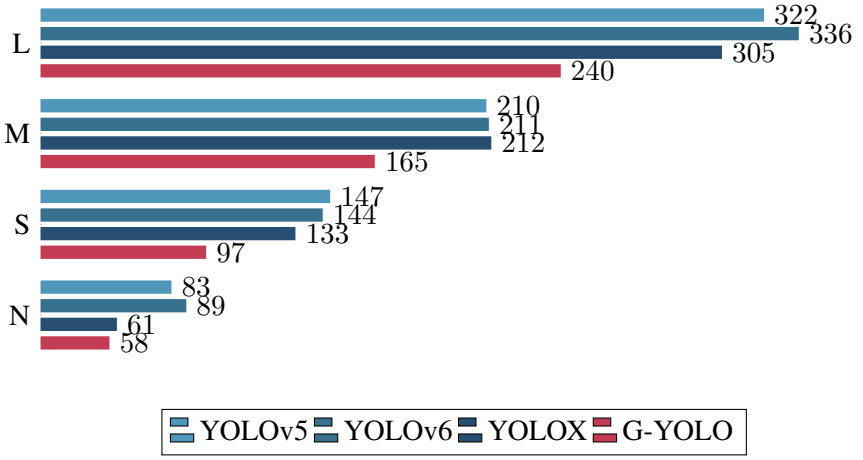

Fig. A.1: Latency comparison between various YOLO and G-YOLO models on VOC dataset.

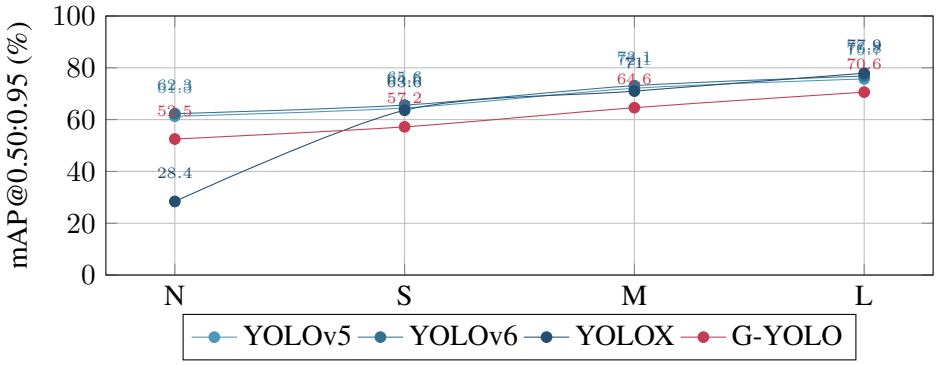

Fig. A.2: mAP comparison between various YOLO and G-YOLO models on VOC dataset.

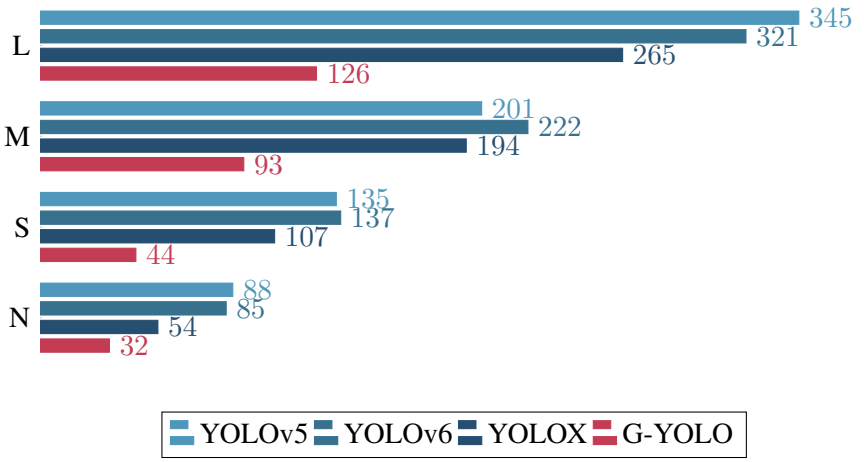

Fig. A.3: Latency comparison between various YOLO and G-YOLO models on RoadSec dataset.

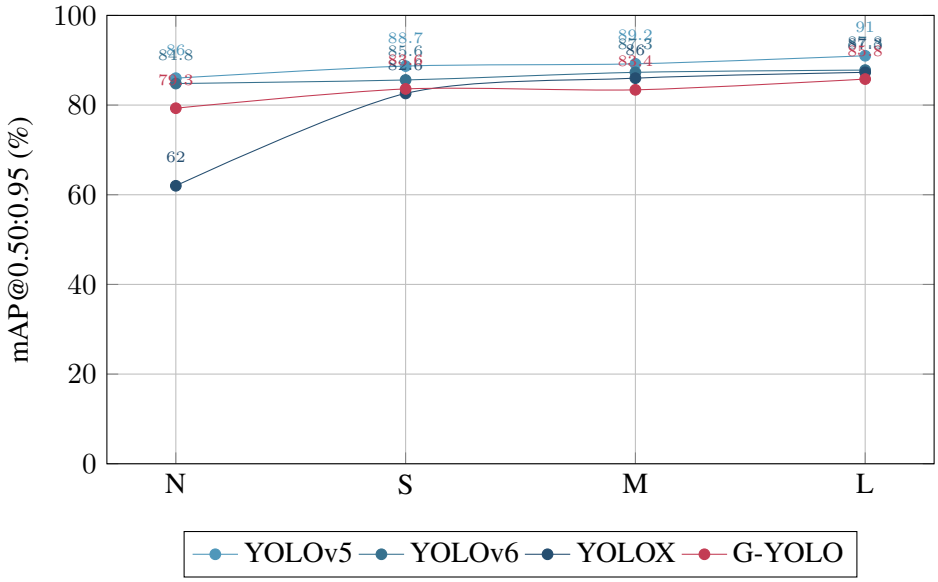

Fig. A.4: mAP comparison between various YOLO and G-YOLO models on RoadSec dataset.

Appendix B

This appendix provides a visual comparison of the object detection outputs from the G-YOLOv6 model and the YOLOv6 N model. The images demonstrate the performance of both models in terms of detection accuracy and frame rate (FPS) on the Jetson TX2 platform.

Fig. B.1: Output of G-YOLOv6

Fig. B.2: Output of YOLOv6 N

Fig. B.3: Comparison of Object Detection Outputs. The image on the left shows the result of our method, achieving over 30 FPS consistently on the Jetson TX2. The image on the right shows the result of the YOLOv6 N model, achieving an average of 11 FPS. Both models provide very similar results in terms of detection accuracy.

초록

객체 탐지 분야에서 YOLO(You Only Look Once) 방법론은 단일 패스 탐지 기능으로 실시간 응용 프로그램에 혁명을 일으켰습니다. 그러나 기존의 YOLO 구조는 방대한 양의 데이터를 처리하여 불필요한 계산을 초래하고 자원이 제한된 환경에서의 배포를 제한합니다. 우리 연구는 YOLO 프레임워크를 개조한 “Gated YOLO”를 도입하여 계산 효율성을 높이면서 탐지 정확도를 유지하는 게이트 메커니즘을 통합합니다.

이 방법은 관찰된 장면의 관련성에 따라 신경 경로의 활성화를 조정하는 메커니즘을 통합합니다. 훈련 단계에서 우리의 접근 방식은 특정 환경 조건에서 일관되게 비활성된 신경 경로를 식별하고 비활성화합니다. 이러한 전략적 비활성화는 불필요한 계산 부담을 줄여 지정된 모델 작업에 더 효율적으로 만듭니다.

실험 결과, 우리의 접근 방식은 탐지 정확도에 미치는 영향을 최소화하면서 계산 부담을 크게 줄여 처리 속도를 높이는 것으로 나타났습니다. 이는 보안 카메라 시스템과 같은 고정된 자원이 제한된 환경에서 실시간 객체 탐지에 효과적인 해결책이 됩니다.

요약하자면, Gated YOLO는 더 효율적인 객체 탐지 솔루션을 향한 중요한 진전을 나타냅니다. 모델 처리 경로를 특정 환경 요구에 맞춤으로써 이 연구는 정적 설정에서 실시간 객체 탐지 시스템의 적용 가능성과 효과를 향상시킵니다.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[2] C. Li, L. Li, H. Jiang, *et al.*, "Yolov6: A single-stage object detection framework for industrial applications," *arXiv preprint arXiv:2209.02976*, 2022.

[3] C. Li, L. Li, Y. Geng, *et al.*, "Yolov6 v3. 0: A full-scale reloading," *arXiv preprint arXiv:2301.05586*, 2023.

[4] Ł. Kaiser and S. Bengio, "Discrete autoencoders for sequence models," *arXiv preprint arXiv:1801.09797*, 2018.

[5] Z. Chen, Y. Li, S. Bengio, and S. Si, "You look twice: Gaternet for dynamic filter selection in cnns," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9172–9180.

[6] Y. Bengio, "Deep learning of representations: Looking forward," in *International conference on statistical language and speech processing*, Springer, 2013, pp. 1–37.

[7] A. Veit and S. Belongie, *Convolutional networks with adaptive inference graphs*, 2020. arXiv: `1711.11503 [cs.CV]`.

[8] J. Liu, Z. Xu, R. Shi, R. C. Cheung, and H. K. So, "Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers," *arXiv preprint arXiv:2005.06870*, 2020.

[9] S. Kalwar, D. Patel, A. Aanegola, K. R. Konda, S. Garg, and K. M. Krishna, "Gdip: Gated differentiable image processing for object detection in adverse conditions," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2023, pp. 7083–7089.

[10] Z. Zhang, R. Tao, and J. Zhang, "Neural network pruning by gradient descent," *arXiv preprint arXiv:2311.12526*, 2023.

[11] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding*, 2016. arXiv: 1510.00149 [cs.CV].

[12] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *arXiv preprint arXiv:1806.08342*, 2018.

[13] B. Jacob, S. Kligys, B. Chen, *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[14] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 8612–8620.

[15] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[16] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma, "Be your own teacher: Improve the performance of convolutional neural networks via self distillation," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 3713–3722.

[17] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[18] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*, http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[19] Ultralytics, *YOLOv5: A state-of-the-art real-time object detection system*, `https://docs.ultralytics.com`, Accessed: 2024-04-01, 2021.

[20] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, *Yolox: Exceeding yolo series in 2021*, 2021. arXiv: `2107.08430 [cs.CV]`.