Gated Scene-Specific YOLO: A Pruning and Gating Approach for Efficient Object Detection

Hector Acosta[1][0009−0003−6784−8882] and Soon Ki Jung[1][0000−0003−0239−6785]

Kyungpook National University, Republic of Korea

Abstract. In object detection, YOLO (You Only Look Once) has significantly advanced real-time applications. However, in dynamic environments, particularly on edge devices like security cameras, optimization is crucial. Our work introduces "Gated Scene-Specific YOLO", an adaptation of YOLO that includes a dynamic gating mechanism for improved efficiency. Traditional YOLO models often process excessive, irrelevant data. Our approach counters this by activating neural pathways relevant to the scene. This is achieved through dynamic gate generation during training and analyzing scenes to deactivate consistently inactive gates, streamlining the model for the environment's specific needs. This method reduces computational load while having a minimal impact on detection accuracy, suitable for resource-constrained devices. Our results show that Gated Scene-Specific YOLO improves processing speed and maintains high accuracy, demonstrating the potential to enhance real-time object detection in various settings. This research contributes by showing a practical approach to optimize deep learning models for specific contexts, leading to more adaptable, resource-efficient detection solutions.

Keywords: Object Detection· YOLO Architecture· Scene-Specific Optimization· Gated Neural Networks· Model Pruning

1 Introduction

Object detection is a cornerstone of computer vision, impacting a wide array of applications from surveillance to autonomous driving. The evolution of deep learning has propelled advancements in this domain, particularly through the YOLO (You Only Look Once) architecture by Redmon [13]. Renowned for its efficiency in real-time processing, YOLO is a testament to the power of modern object detection systems from high-powered computing environments to more performance-constrained devices. While YOLO excels in real-time processing across these settings, its adaptability still faces challenges, especially when deployed on edge devices where computational resources are inherently limited. To address these challenges, our research specifically focuses on YOLOv6 by Li et al. [9, 10], known for its hardware-focused design and efficiency in real-time applications, forming an ideal base for our enhancements. In these scenarios, even the slightest optimization can yield significant improvements in speed and efficiency. Addressing this limitation, our research introduces "Gated Scene-Specific

YOLO," a novel approach that integrates dynamic gating with scene-specific model pruning within the YOLO architecture.

In the world of neural network optimization, model pruning emerges as a promising solution to alleviate computational intensity. This technique focuses on trimming redundant or less significant parameters from a neural network, thus streamlining its structure with minimal impact on performance. Our methodology advances this concept by employing a dynamic gating mechanism that adapts to the unique features of the input scene, enhancing the object detection process's efficiency without compromising accuracy.

A key innovation in our approach is the use of Improved SemHash, a method initially introduced by Kaiser and Bengio [6] and further explored by Chen et al. [2]. This technique allows for the generation of binary gates during training, crucial for selectively activating or deactivating specific network filters in response to input variations. The process involves dynamically generating gates during training and then conducting an analysis phase for a specific scene. This analysis identifies gates that are consistently inactive and exports them for static application in real-world scenarios. As a result, the Gater Network, typically part of the backbone during training, can be excluded during actual deployment, relying instead on these statically determined gates for efficient and tailored object detection.

Our research introduces a significant advancement in the YOLO architecture through the integration of a gating network, intelligently deactivating filters based on the unique features of each input scene. This novel approach, enhanced by the incorporation of Improved SemHash [1, 2], not only allows for precise network activity control but also brings considerable improvements in computational efficiency and detection accuracy. Extensive experimental validation, focusing on metrics such as FLOPs, FPS, and mAP@0.5:0.95, demonstrates the efficacy of our strategy as we observed a significant increase in FPS compared to the YOLO counterpart of our pruned model, all without compromising the robustness of detection as evidenced by stable mAP scores.

2 Related Work

The pursuit of efficiency in neural network architectures has become increasingly critical, especially in contexts where computational resources are limited. This pursuit is often centered around the concept of sparsity and conditional computation in neu-ral networks. One of the pioneering ideas in this domain, as introduced by Bengio [1], focuses on the selective activation of neural pathways similar to decision nodes in decision trees. This approach aims to reduce computational load while preserving the benefits of distributed representations.

An important development related to this concept is Dynamic Sparse Training (DST) introduced by Liu et al. [12], which dynamically adjusts the sparsity of neural networks during training. DST techniques, as explored in various studies, modify the active connections within a network based on the training data, allowing for a more efficient use of model capacity and computational resources.

However, DST often involves continuous adjustment during training, which may still pose challenges in resource-constrained environments.

A significant stride in this area is the concept of dynamic filter selection in Convolutional Neural Networks (CNNs), as explored by Chen et al. [2] in "You Look Twice: GaterNet for Dynamic Filter Selection in CNNs". This methodology dynamically selects filters based on the input, tailoring the network's complexity to the requirements of the specific task, thus enhancing computational efficiency.

In parallel, "Neural Network Pruning by Gradient Descent" by Zhang et al. [15] presents a methodical approach to reduce network complexity through gradient descent. This technique effectively streamlines the network, ensuring that performance degradation is kept to a minimum. It aligns with the broader goal of achieving efficient yet robust neural network models.

Furthermore, recent research has broadened the scope of conditional computation beyond gating mechanisms. Techniques such as adaptive dropout rate adjustments and tree-structured neural networks demonstrate novel ways of input-dependent computation. Additionally, methods like SE-Net by Hu et al. [5], which dynamically scale components of the network, offer new insights into real-time efficiency optimization.

Moreover, the development of efficient object detection models for resource-constrained environments has been exemplified by approaches like MobileNet YOLO. Introduced by Howard et al. [4], MobileNet employs depth-wise separable convolutions to construct lightweight neural networks, offering a significant reduction in computational requirements without substantial loss in performance. The integration of MobileNet with the YOLO (You Only Look Once) framework, as in MobileNet YOLO, further demonstrates the feasibility of real-time object detection in embedded systems. These models have set benchmarks in achieving high efficiency, making them particularly suitable for applications in mobile and embedded devices.

Recently, advancements in applying neural networks to challenging conditions have been made, as seen in the ICRA2023 paper 'GDIP: Object-Detection in Adverse Weather Conditions Using Gated Differentiable Image Processing' by Kalwar et al. [8]. This work addresses object detection under adverse weather conditions using a novel gated image processing technique. While focusing on a different aspect of gating mechanisms, it presents an interesting parallel to our approach. Future work could involve comparing these methodologies to explore their respective strengths and limitations in various environmental contexts.

Extending beyond these methods, our approach uniquely capitalizes on the dynamic analysis of neural pathways to establish a static gating mechanism for inference. This strategy not only aligns with the objectives of dynamic filter selection and pruning but also innovates by solidifying the gate configurations post-training. This results in significantly reduced computational demands during inference, particularly beneficial in consistent environmental contexts, setting our method apart in the realm of efficient, real-time object detection.

3 Methodology

In this section, we outline the "Gated Scene-Specific YOLO" model, comprising the Gater Network, YOLO architecture modifications, and scene-specific analysis. These elements collectively boost efficiency in resource-constrained settings. The Gater Network adjusts dynamically to scene features, the YOLO architecture integrates these adjustments, and the analysis process fine-tunes the model for particular environments.

3.1 Gater Network

The GaterNet's initial task is to process the input image for feature extraction. Utilizing ResNet-18 by He et al. [3], known for its shallow yet efficient architecture, we extract key features while maintaining a balance between representational quality and computational speed.

Subsequently, it is tasked to generate a fixed set of gates based on the extracted features. For this, the ResNet-18's output in our implementation, is processed to be align with the channel dimensions required by our YOLO network and later turned into binary gates that will essentially turn off or on specific channels from our YOLO architecture. The alignment is crucial to ensure that the gating set serves as a direct input to the YOLO architecture, maintaining the integrity of the training and detection process. This gating strategy can be represented in (1) as bellow:

$$GatedFeatures(x) = F_{gate}(F_{extract}(x)). \tag{1}$$

Here, $F_{extract}$ is the feature extraction function defined as $F_{extract} : x \rightarrow f$, where the input image $x \in \mathbb{R}^{h_0 \times w_0 \times c_0}$ is transformed into a feature vector $\mathbf{f} \in \mathbb{R}^h$. The parameters h_0, w_0, and c_0 represent the height, width, and channel number of the input image, respectively, and h signifies the dimensionality of the extracted features.

The gating function F_{gate} is defined as $F_{\text{gate}} : \mathbf{f} \rightarrow \mathbf{g}$, mapping the feature vector $\mathbf{f}$ to a gated vector $\mathbf{g}$. In the case of binary gating, $\mathbf{g}$ belongs to the binary space $\{0, 1\}^c$, where c is the number of channels in the YOLO network being modulated. An overview of this process can be evaluated in Fig. 1.

Feature Extraction. In our Gater Network, the F_{extract} function, as shown in Equation (1), is designed for efficient representation extraction from input images. Since we only require the extracted features and not the classifications from our feature extractor, we utilize ResNet-18 [3] without its final classification layer. Subsequently, an adaptive pooling layer precedes the feature extraction, standardizing input dimensions across different samples. We observed that larger dimensions post-pooling improve feature representation but at the cost of increased inference time. The extracted features are then flattened into

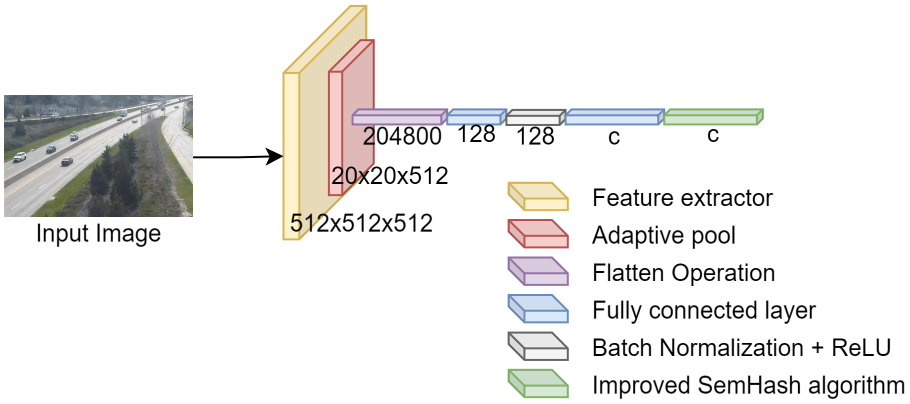

Fig. 1. This diagram depicts the transformation of an input image data through our GaterNet pipeline. It includes our feature extractor, which output dimensionality is reduced and standardized by the adaptive pooling, to subsequently be flattened for our fully connected layers culminating in the Improved SemHash layer that renders the output differentiable.

a one-dimensional vector, making them compatible with the subsequent fully connected layers. This process can be summarized in the equation (2) as below:

$$F_{\text{extract}}(x) = \text{Flatten}(\text{AdaptivePool}(f_{\text{net}}(x))), \tag{2}$$

where $f_{\text{net}}(x)$ represents the adapted neural network function, in our specific case, the ResNet-18 model [3] excluding its classification layer, used for initial processing of the input image x. *AdaptivePool* refers to the adaptive pooling operation standardizing the output size, and *Flatten* denotes the process of converting the pooled features into a one-dimensional vector suitable for further processing in the network.

Binary Gates. For the binary gates' generation, we leverage the method proposed by Chen et al. [2]. Similarly to their approach, we implemented a dual-layer strategy for mapping the high dimensional feature vector of size h to the YOLO network channels of size c. This structure uses two fully connected layers, incorporating a bottleneck design. The rationale for this design choice is to effectively manage the parameter count. A single-layer architecture, directly mapping from h to c, would result in an excessively large parameter matrix of size $h \times c$. The two-layer approach, with an intermediary bottleneck layer, significantly reduces the total number of parameters. These considerations can be examined in the equations (3) and (4) as below:

$$f_0 = \text{ReLU}(\text{BatchNorm}(\text{FC1}(f))), \textit{and} \tag{3}$$

$$g_0 = \text{FC2}(f_0). \tag{4}$$

In Equation (3) we defined the initial mapping and non-linear transformation of the feature vector f of size h, to our bottleneck of size b. Where FC1 refers to our first fully connected layer, BatchNorm denotes the process of batch normalization, and the process culminates with the rectified linear unit (ReLU) activation. Subsequently, in Equation (4) we depict the mapping of the bottleneck representation f_0 to a real-value vector g_0 of size c. Where FC2 refers to our second fully connected layer.

Improved Semantic Hashing. A key component of our methodology is rendering the binary gates differentiable, a challenge we address using the SemHash introduced by Kaiser and Bengio in [6] and later used by Chen et al. [2] in their approach. In this technique, during training, noise is drawn from a Gaussian distribution with mean 0 and standard deviation of 1 to create a c-dimensional vector which is later added to our g_0 turning it into g_{noisy}. Based on this we can get two vectors g_α and g_β which are defined in Equations (5), (6) and (7) as follows:

$$g_{\text{noisy}} = g_0 + \epsilon, \tag{5}$$

$$g_\alpha(i) = \text{clamp}(1.2 \times \sigma(g_{\text{noisy}}(i)) - 0.1, 0, 1), and \tag{6}$$

$$g_\beta(i) = \begin{cases} 1 & \text{if } g_{\text{noisy}}(i) > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

Equation (5) introduces the variable g_{noisy}, which represents the noisy version of the gating vector g_0. The noise term ϵ is added to g_0, which perturbs the original gating signal by a random amount drawn from a Gaussian distribution.

Equation (6) defines the gating vector g_α, derived from g_{noisy}. Where σ denotes the standard sigmoid activation function, $\sigma(x) = \frac{1}{1+e^{-x}}$, which maps the input x to a value between 0 and 1. The term $g_\alpha(i)$ represents the activation for the i-th element of the gating vector after applying a modified sigmoid function as described by Kaiser and Sutskever [7], defined by scaling σ by 1.2, subtracting 0.1, and then clamping the result to the interval [0, 1].

Equation (7) defines the binary gating vector g_β, which represents a hard thresholding operation that binarizes the noisy gating vector g_{noisy}. Each element of $g_\beta(i)$ is set to 1 if the corresponding element of g_{noisy} is positive and 0 otherwise.

Loss Function. We've enhanced the original loss function from the YOLOv6 architecture by Li et al. [9, 10] to factor in our GaterNet loss. It consists of classification loss, bounding box regression loss, and a gating loss to encourage sparsity in the gating mechanism. The loss function L is formulated in Equation (8) as follows:

$$L = \alpha_{\text{cls}} L_{\text{cls}} + \alpha_{\text{iou}} L_{\text{iou}} + \alpha_{\text{dfl}} L_{\text{dfl}} + \lambda \cdot L_{\text{gate}}, \tag{8}$$

where L_{cls} denotes the classification loss normalized by the sum of target scores to mitigate division by zero errors, alongside its respective weight α_{cls}. L_{iou} and L_{dfl} comprise the bounding box regression loss, weighted by the target scores

α_{iou} and α_{dfl}. And L_{gate} represents the gating loss, encouraging sparsity via an L_1 regularization term scaled by the batch size in practice and weighted by λ. The L_{gate} can be defined in Equation (9) as follows:

$$L_{\text{gate}} = \frac{1}{c}\|g\|_1 = \frac{1}{c}\sum_i |g_i|. \tag{9}$$

Here, c is a scaling factor, which is the size of the gating vector g, and g_i are the elements of the gating vector. The division by c normalizes the regularization term to account for the size of the data.

3.2 YOLO Architecture

In our model, the YOLO architecture, particularly YOLOv6 as proposed by Li et al. [9, 10], forms the secondary principal module. YOLOv6 is chosen for its real-time performance and effective use of hardware resources. Our significant contribution to this framework is the incorporation of GaterNet. This auxiliary network, specifically designed for adaptive gating mechanisms, allows for dynamic feature selection, thereby enhancing computational efficiency. Figure 2 illustrates our modified YOLO architecture.

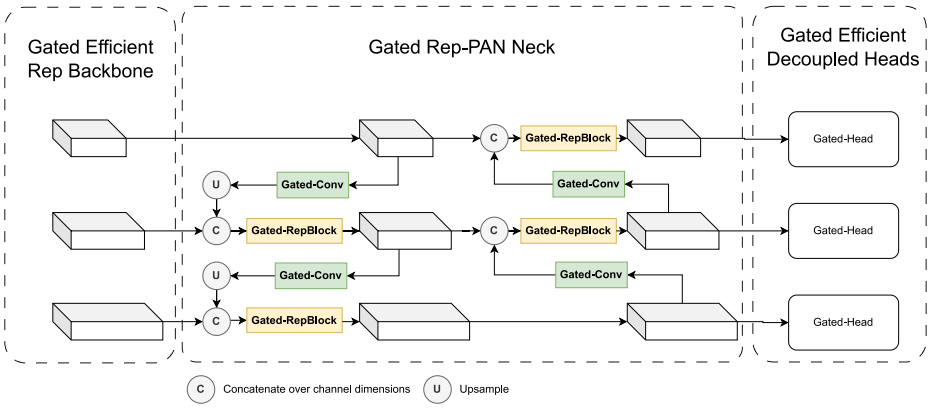

Fig. 2. The diagram showcases our YOLOv6 architecture augmented with GaterNet. It includes a 'Gated Efficient Re-parameterizable Backbone (EfficientRep)', a 'Gated Rep-Pan Neck', and 'Gated Efficient Decoupled Heads'. Each section is purpose-built to optimize performance and efficiency.

The 'Gated Efficient Reparameterizable Backbone', also known as EfficientRep, is where the initial and crucial process of feature extraction occurs. The gates within this backbone are responsible for selecting and emphasizing valuable features while suppressing unnecessary ones, thus setting the stage for enhanced processing in subsequent layers.

In the 'Gated Rep-Pan Neck', inspired by the traditional PAN topology and enhanced with Rep blocks, gates play a critical role. They adjust resolution and scale dynamically, ensuring that the flow and integration of features from the backbone to the detection heads are optimized. This section is crucial for refining and aggregating features for accurate object detection.

In the 'Gated Efficient Decoupled Heads' of our YOLOv6 architecture, gates play a crucial role in enhancing detection efficiency. These heads, drawing inspiration from the original YOLOv6 model, are designed with a hybrid-channel strategy, which reduces the number of convolutional layers. The gates within these heads dynamically filter and prioritize features, focusing computational resources on the most relevant elements for accurate object detection. This selective processing significantly cuts down on unnecessary computations, thereby reducing inference latency.

YOLO Gate module. The Gate Module significantly augments the YOLO architecture, adapting its behavior for training and inference. During training, it modulates the network's output via element-wise multiplication based on GaterNet's signals, either preserving or attenuating layer outputs. During inference, it evaluates active gates' proportion, bypassing or permitting convolutional operations accordingly. This dual functionality optimizes feature propagation, tailoring network performance to specific operational needs.

$$\mathbf{G} = [g_1, g_2, \ldots, g_c], \tag{10}$$

$$g_{\text{closed}} = \frac{\sum_{i=1}^{c} \mathbf{1}(G_i = 0)}{c}, and \tag{11}$$

$$g(x) = \begin{cases} 0, & \text{if } g_{\text{closed}} > 0.99, \\ \text{Conv}(x) \odot \mathbf{G}, & \text{otherwise.} \end{cases} \tag{12}$$

In our gate module, the vector $\mathbf{G}$ represents the array of gating decisions, as defined in Equation (10), where each gate g_c within the array can either enable or suppress the corresponding feature map in the network's layer. Equation (11) quantifies the percentage of gates that are closed, g_{closed}, by dividing the count of gates with a value of zero by the total number of gates c. This metric provides insight into the scarcity of activations. Equation (12) details the conditional operation applied to the input tensor x: if g_{closed} exceeds 99%, indicating almost all gates are closed, the output is a tensor of zeros, effectively bypassing the convolution; otherwise, the convolution $\text{Conv}(x)$ is performed and element-wise multiplied by $\mathbf{G}$, denoted by $\text{Conv}(x) \odot \mathbf{G}$, to produce the gated output. Figure 3 visualizes this gating mechanism the following way:

3.3 Analysis Step

The final component of our methodology is the analytical task. In this phase, we conduct inference on a designated scene over a set duration. This allows the gat-

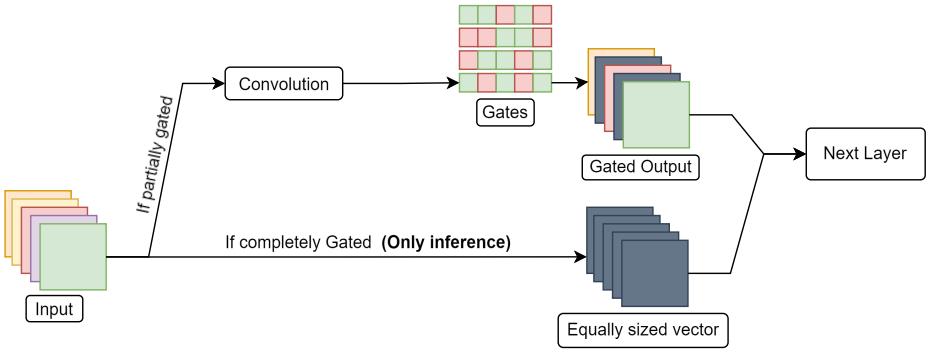

Fig. 3. This depicts the conditional modulation of the "Input" tensor by "Gates" following convolution, either allowing a gated output or bypassing the operation to produce a zeroed tensor for the next layer.

ing mechanism to ascertain which filters should be deactivated and which should remain operational, depending on the unique features of the current scene. Practically, we monitor the gating decisions for every frame processed, scrutinizing the closure rate of each section. For clarity and legibility in our main discussion, we present these decisions grouped by sections, as depicted in Figure 4.

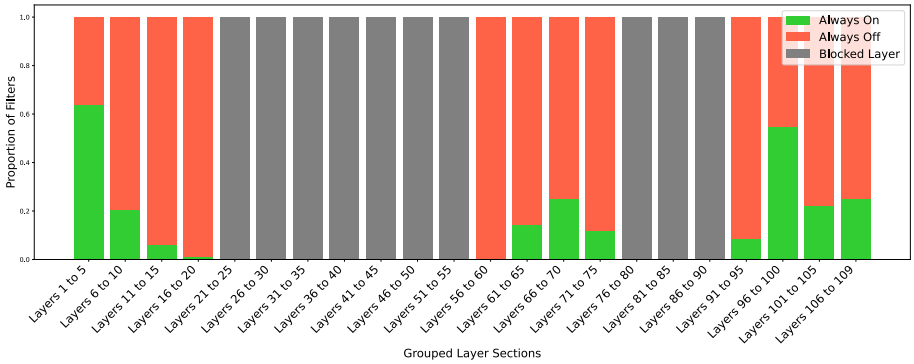

Fig. 4. The graph displays a simplified distribution of gating states across different sections of the network, grouped for clarity. It underscores filters that are persistently active ("Always On"), suppressed ("Always Off"), or sections that are completely deactivated ("Blocked Layer").

Our ultimate goal is to distill a static gating configuration specific to a scene, which can then be applied consistently for subsequent inferences on that scene or similar unseen data, eliminating the need for GaterNet to dynamically compute gates. While we had the option to assess gates that were predominantly active

or dependent on the input, for our objective of establishing a static gating pattern that prunes the model independently of the input, we classified such gates as "Always On" to preserve their functionality. As evident from the grouped representation in Figure 4, a significant number of layers have been effectively "Blocked", ensuring that their associated computations are no longer necessary.

4 Results

4.1 Dataset

Our experiments focused on object representation in stationary environments like those monitored by surveillance and traffic cameras. We used the STREETS dataset by Snyder and Do [14], containing over four million images from Lake County, IL. Lacking annotations, we enriched the dataset using the YOLOv6 Large model [9,10] for precise supervised learning. This enhancement allowed a thorough examination of various traffic patterns and incidents. The detailed annotations of the STREETS dataset provided a solid foundation for evaluating our approach, closely mimicking real-world traffic conditions.

4.2 Experiment Configuration

In our experiments, we employed YOLOv6 as the base model for its performance and hardware efficiency. We enhanced it with GaterNet, utilizing ResNet [3] for dynamic feature analysis. While other lightweight CNNs could replace ResNet, we preferred ResNet18 for its balance of power and simplicity. We used the STREETS dataset, annotated using YOLOv6 Large, to match our fixed scene research focus. Conducted on an RTX 2080 Super with 8GB RAM, our setup ensured robust computational capacity. Additionally, we pre-trained our model without gating to refine the weights, preparing for more effective gating integration later.

Furthermore, to validate the real-world applicability of our model, especially in embedded and resource-constrained environments, we deployed our enhanced YOLOv6 with GaterNet on the NVIDIA Jetson TX2 edge device. This deployment aimed to measure the model's efficiency in milliseconds per detection, a crucial metric for real-time applications in such environments. The results from the Jetson TX2 provide valuable insights into our model's performance outside of high-end computational settings, demonstrating its versatility and adaptability to different hardware configurations.

4.3 Classification Performance

The classification performance of our model was assessed using a comprehensive set of metrics including FLOPs, FPS, mAP at different IoU thresholds, and critically, inference time in milliseconds (ms) on the NVIDIA Jetson TX2. This addition underscores our commitment to evaluating performance in real-world, resource-constrained environments. We examined various variants of the

YOLOv6 model: N (Nano), S (Small), M (Medium), and L (Large), both with and without gating mechanisms. The inclusion of inference times, particularly on an edge device like the Jetson TX2, provides insights into the practical deployment capabilities of our model. The summarized performance data is represented at Table 1.

In a comparative analysis, the performance of our YOLOv6 N model was juxtaposed with that of the MobileNet YOLOv3 implementation, referenced from Liu in [11], to evaluate efficiency in practical, resource-constrained scenarios. Both models were tested under similar conditions, using an image resolution of 352 pixels. Our YOLOv6 N model achieved a notably faster inference time of 32ms compared to the 55ms of MobileNet YOLOv3, while also demonstrating superior accuracy, with a mAP@0.50 of 0.8341 against MobileNet YOLOv3's 0.7144. This comparison underscores the effectiveness of our approach in delivering high efficiency without compromising on detection accuracy, marking a significant stride in real-time object detection for embedded systems.

Model	Size	Gates	FLOPs	FPS (RTX 2080)	mAP 0.50	mAP 0.50:0.95	Inference Time (TX2)
YOLOv6-N	640	-	3.38G	188.42	0.9341	0.7930	85ms
		5,632	1.54G	218.48	0.9057	0.7351	32ms
YOLOv6-S	640	-	13.46G	176.37	0.9450	0.8361	137ms
		11,248	7.65G	211.85	0.916	0.7712	44ms
YOLOv6-M	640	-	25.48G	101.10	0.9353	0.8249	222ms
		21,344	13.23G	173.27	0.9159	0.7830	93ms
YOLOv6-L	640	-	44.77G	64.72	0.9469	0.8579	321ms
		31,760	23.65G	129.96	0.9186	0.7878	126ms

Table 1. Classification performance of various YOLOv6 configurations with and without gating mechanisms.

4.4 Comparative Analysis of Gate Behavior Across Scene Variations

This section delves into the comparative analysis of gate behavior across different scene variations. We investigate how specific scene characteristics influence the activation patterns of the gates within our model. Notably, certain channels are activated based on these unique scene features, while others remain inactive.

The visual comparison in Figures 5 and 6 illustrates the distinctive characteristics of each scene.

The graph in Figure 7 illustrates the variance in gate activation patterns when exposed to distinct scene attributes. The findings reveal a correlation between scene elements and the activation or suppression of particular channels, emphasizing the model's adaptability and sensitivity to contextual nuances.

Fig. 5. Scene 1: An intersection with a broader spectrum of car sizes.

Fig. 6. Scene 2: An overpass highway camera focused on larger vehicles given its location.

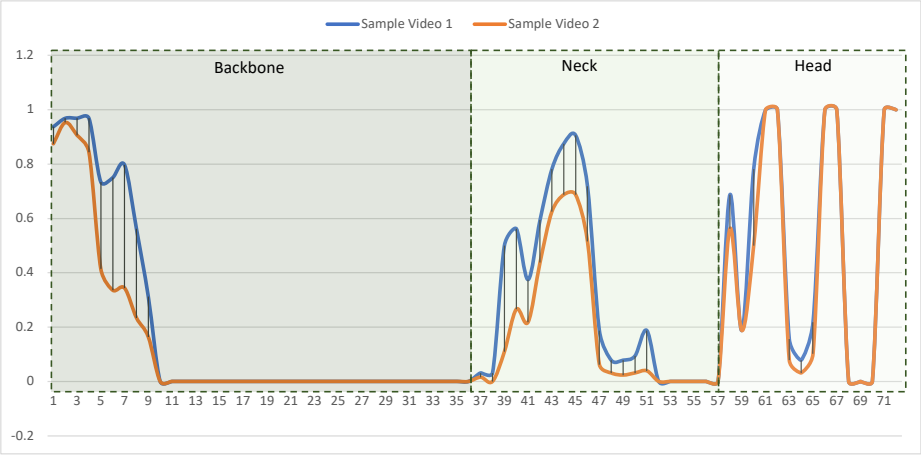

Fig. 7. Comparative graph showing gate activation across different scenes. This visualization underscores how specific channels are influenced by scene characteristics, reflecting the dynamic nature of gate behavior.

5 Discussion

Our study has successfully demonstrated that Gated Scene-Specific YOLO significantly enhances computational efficiency in environments with consistent features, such as stationary surveillance systems and traffic monitoring cameras. By focusing on fixed settings, the model's selective gating mechanism strategically deactivates irrelevant filters, reducing computational overhead. This approach aligns with the growing need for resource-efficient models in embedded systems and edge devices, where consistent environmental conditions prevail. While the model shows less adaptability to new or changing scenes, its performance in familiar settings suggests a substantial reduction in resource consumption while minimally impacting accuracy. This characteristic makes it particularly valuable in applications where computational resources are limited, and operational consistency is guaranteed. Future work could focus on refining the gating mecha-

nism to ensure minimal performance degradation over time and exploring similar efficiency-oriented approaches in other fixed-setting applications.

6 Conclusion

In this paper, we introduced the Gated Scene-Specific YOLO, a novel adaptation of the YOLO architecture that incorporates a dynamic gating mechanism to improve efficiency in fixed-setting object detection tasks. Our approach represents a significant stride towards resource-efficient deep learning, particularly for applications operating under consistent environmental conditions. The substantial computational savings achieved, underscore the potential of selective gating in enhancing the performance of object detection models. While our model is tailored for fixed settings, its underlying principles may inspire broader research into resource optimization in various deep learning applications. As the demand for intelligent systems in resource-constrained environments grows, approaches like Gated Scene-Specific YOLO will be crucial in paving the way for more sustainable and efficient solutions.

7 Acknowledgement

This work was supported by Innovative Human Resource Development for Local Intellectualization program through the Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT)(IITP-2024-RS-2022-00156389) and also was supported by the BK21 FOUR project (AI-driven Convergence Software Education Research Program) funded by the Ministry of Education, School of Computer Science and Engineering, Kyungpook National University, Korea (4199990214394).

References

1. Bengio, Y.: Deep learning of representations: Looking forward. In: International conference on statistical language and speech processing. pp. 1–37. Springer (2013)
2. Chen, Z., Li, Y., Bengio, S., Si, S.: You look twice: Gaternet for dynamic filter selection in cnns. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9172–9180 (2019)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
4. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 1314–1324 (2019)
5. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 7132–7141 (2018)
6. Kaiser, Ł., Bengio, S.: Discrete autoencoders for sequence models. arXiv preprint arXiv:1801.09797 (2018)

7. Kaiser, Ł., Sutskever, I.: Neural gpus learn algorithms. arXiv preprint arXiv:1511.08228 (2015)
8. Kalwar, S., Patel, D., Aanegola, A., Konda, K.R., Garg, S., Krishna, K.M.: Gdip: gated differentiable image processing for object detection in adverse conditions. In: 2023 IEEE International Conference on Robotics and Automation (ICRA). pp. 7083–7089. IEEE (2023)
9. Li, C., Li, L., Geng, Y., Jiang, H., Cheng, M., Zhang, B., Ke, Z., Xu, X., Chu, X.: Yolov6 v3. 0: A full-scale reloading. arXiv preprint arXiv:2301.05586 (2023)
10. Li, C., Li, L., Jiang, H., Weng, K., Geng, Y., Li, L., Ke, Z., Li, Q., Cheng, M., Nie, W., et al.: Yolov6: A single-stage object detection framework for industrial applications. arXiv preprint arXiv:2209.02976 (2022)
11. Liu, E.: Mobilenet-yolo caffe. https://github.com/eric612/MobileNet-YOLO (2018)
12. Liu, J., Xu, Z., Shi, R., Cheung, R.C., So, H.K.: Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. arXiv preprint arXiv:2005.06870 (2020)
13. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)
14. Snyder, C., Do, M.: Streets: A novel camera network dataset for traffic flow. Advances in Neural Information Processing Systems **32** (2019)
15. Zhang, Z., Tao, R., Zhang, J.: Neural network pruning by gradient descent. arXiv preprint arXiv:2311.12526 (2023)