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 YOLOv6 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. AAA

Prof. BBB

Prof. CCC

Prof. DDD

Prof. EEE

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 evolving landscape of object detection, the advent of the YOLO (You Only Look Once) methodology has marked a significant leap forward for real-time applications, streamlining the process with its single-pass detection capabilities. Despite its advancements, the dynamic and often unpredictable nature of real-world environments, especially when operating on edge devices like security cameras, presents a pressing challenge for optimization. Addressing this, our research introduces the ”Gated Scene-Specific YOLO,” a novel adaptation of the YOLO framework, incorporating a dynamic gating mechanism aimed at enhancing computational efficiency with minimal compromise on the model’s detection prowess.

Traditional YOLO architectures, while robust, are predisposed to processing vast amounts of data, much of which may be extraneous or irrelevant to the task at hand. This not only leads to unnecessary computational overhead but also hinders the deployment of such models in environments where resources are limited. Our Gated Scene-Specific YOLO methodology seeks to alleviate this issue by integrating a mechanism that dynamically adjusts the activation of neural pathways based on the relevance to the observed scene. Through a meticulous process of gate generation and analysis during the training phase, our approach identifies and deactivates neural pathways that are consistently inactive across specific environmental conditions. This strategic deactivation allows the model to shed redundant computational weight, thus becoming more streamlined and efficient for the task it is deployed to perform.

The core of our research lies in demonstrating the practicality of

dynamically tuning deep learning models to their operational context, significantly reducing the computational load while minimally impacting the detection accuracy. Our empirical results showcase that the Gated Scene-Specific YOLO not only elevates processing speeds but also upholds a high standard of accuracy, making it a compelling solution for real-time object detection across a diverse array of settings. This contribution is particularly relevant for the deployment of object detection models in resource-constrained devices, where optimizing the balance between efficiency and performance is paramount.

In summary, the Gated Scene-Specific YOLO represents a meaningful stride towards more adaptable and resource-efficient object detection solutions. By tailoring model processing pathways to the specific demands of the environment, this research paves the way for the development of highly optimized, context-aware deep learning models, thereby enhancing the applicability and effectiveness of real-time object detection systems in fixed and scene specific settings.

1 Motivation and Objectives

1.1 Introduction

Object detection stands as a foundational pillar within the field of computer vision, influencing an extensive range of applications from advanced surveillance systems to the dynamic realm of autonomous driving technologies. The rapid evolution of deep learning methodologies has significantly propelled the field forward, introducing architectures capable of not only enhancing the accuracy of object detection but also its efficiency and adaptability in real-time processing environments. Among these innovations, the YOLO (You Only Look Once) architecture, introduced by Redmon et al. [1], has emerged as a significant contribution, revolutionizing the way real-time object detection is approached by enabling swift and efficient processing without the need for iterative detection stages. This architecture underscores the potential of modern object detection systems, demonstrating remarkable versatility across a variety of computing environments, from high-powered servers to more constrained devices such as smartphones and embedded systems like the ones introduced by the Nvidia Jetson Platform.

Despite the advancements brought forth by YOLO and its subsequent iterations, challenges remain, particularly in the context of edge computing where computational resources are limited, and the demand for real-time processing is paramount. Recognizing the potential for further optimization, our study delves into the exploration of YOLOv6 [2], [3], a variant designed with a keen focus on hardware efficiency and optimized for real-time applications. The

architecture of YOLOv6 [2], [3] serves as an ideal foundation for our investigation, given its emphasis on performance in hardware-constrained environments. Within such contexts, minor enhancements can lead to substantial gains in processing speed and overall system efficiency, thereby improving the applicability of YOLO-based models in edge devices.

To advance the capabilities of YOLOv6 [2], [3], our research introduces the ”Gated YOLO,” a novel framework that marries the concept of dynamic gating with the principle of model pruning specifically tailored to the YOLO architecture. That being said our approaches are not limitted to the integration with YOLOv6, given that we maintain a high degree of modularity of its components, we facilitate the ability to port it to other YOLO architectures.

Model pruning, a technique aimed at reducing the computational burden of neural networks, achieves this by eliminating superfluous or insignificant parameters, thereby refining the network's structure with minimal detriment to its performance. Our approach innovates beyond traditional model pruning by implementing a dynamic gating mechanism that adjusts in real-time during training to the distinctive characteristics of the input scene, thereby optimizing the efficiency of the object detection process with minimal sacrifice in accuracy.

A cornerstone of our methodology is the adaptation of Improved SemHash [4], a technique initially proposed by Kaiser and Bengio and further refined by Chen et al. [5] This approach facilitates the generation of binary gates during the model training phase, enabling selective activation or deactivation of network filters in response to variations in the input. Through dynamic gate generation and subsequent analysis tailored to specific scenes, our method identifies filters

that consistently remain inactive. These filters are then statically pruned from the network for deployment, allowing the model to operate more efficiently by focusing computational resources on active, scene-relevant pathways. The Gater Network [5], integral during the training phase for gate determination, is thus rendered unnecessary during actual deployment, replaced by the pre-determined, statically applied gates that ensure both efficiency and specificity in detection.

Our contributions to the YOLO architecture, through the integration of a gating network and the innovative use of Improved SemHash [4], not only enable precise control over network activity but also herald significant improvements in computational efficiency and detection accuracy. The effectiveness of our approach is substantiated through rigorous experimental validation, focusing on key performance metrics such as floating-point operations per second (FLOPs), frames per second (FPS), and mean Average Precision (mAP@0.5:0.95). Our findings reveal a notable increase in FPS for the Gated YOLO model in comparison to its YOLOv6 counterpart, with a slight compromise on detection robustness, as evidenced by stable mAP scores. This research thus presents a significant step forward in optimizing deep learning models for object detection, especially in scenarios where computational resources are at a premium.

1.2 Motivation

The relentless pursuit of advancements in computer vision, specifically within the domain of object detection, has been driven by the escalating demands of modern applications. The inception of deep learning architectures like YOLO has significantly narrowed the gap between theoretical possibility and practical

implementation, offering a glimpse into the potential of real-time object detection systems. Yet, as these technologies are increasingly deployed in real-world scenarios, particularly on the edge, the limitations of current models under resource-constrained conditions become apparent. The motivation behind our work is rooted in the desire to transcend these limitations, pushing the boundaries of what is possible with existing object detection frameworks.

Our focus on YOLOv6 [2], [3], known for its balance of speed and accuracy, stems from a recognition of the critical need for optimization in edge computing scenarios where resources are scarce yet the demand for high-performance computing is increasingly evident. 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 a wide array of applications, from low-power IoT devices to mobile applications requiring real-time analysis and feedback, envisioning a future where advanced object detection is not only possible but also practical and pervasive, regardless of computational limitations.

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 dynamic gating mechanism to enhance computational efficiency in object detection tasks, particularly in edge computing environments. To achieve this, we aim to:

Implement Dynamic Gating: Integrate a dynamic gating mechanism that

adapts to the unique characteristics of input scenes, thereby selectively activating relevant neural pathways and improving model efficiency.

Optimize Through Model Pruning: Apply model pruning techniques in conjunction with dynamic gating 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 and further optimization of the model for specific scenarios.

Demonstrate Practical Efficacy: Validate the effectiveness of the Gated YOLO model through extensive testing, focusing on key performance metrics such as processing speed (FPS), computational efficiency (FLOPs), and detection accuracy (mAP@0.5:0.95).

Enhance Real-World Applicability: Ensure that the optimized model maintains high detection accuracy while significantly reducing computational load, making it suitable for deployment in real-world, resource-constrained environments.

Through these objectives, our research seeks to address the pressing 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 Work

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

2.1 Sparsity and Conditional Computation

A fundamental concept in the pursuit of neural network efficiency is the integration of sparsity and conditional computation mechanisms. Sparsity in neural networks refers to the idea that not all neurons or connections (weights) are necessary for every input. By identifying and activating only a subset of the neural pathways for a given input, significant reductions in computational overhead can be achieved without sacrificing the model's ability to represent complex functions. This principle is akin to how decision trees operate, where at each node a decision is made that leads to a subset of the next possible states, thus not exploring all branches of the tree 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 overfitting by

limiting the effective capacity of the model based on the complexity of the input.

Introduced by Bengio et al. [6], the idea of selectively activating neural pathways has been a cornerstone in the development of more efficient neural network architectures. The authors suggest that such mechanisms could allow for deeper and more complex models by allocating computational resources more judiciously. By simulating sparsity and conditional computation, networks can potentially achieve a balance between depth and width, optimizing the computational cost without compromising the benefits of distributed representations.

Several other approaches have been proposed to implement sparsity and conditional computation in neural networks. For example, the approach based on gating mechanisms, where gates control the flow of information in the network, effectively turning on or off certain pathways based on the input. Chen et al. introduced GaterNet [5], where a gater network generates binary gates for selectively activating filters in the backbone network based on each input. Their experiments on CIFAR and ImageNet datasets show models consistently outperform original models with a large margin.

The brilliance of dynamic filter selection lies in its capacity to maintain high levels of accuracy while dramatically enhancing computational efficiency. This is especially pertinent for applications requiring real-time processing. Moreover, the adaptability of GaterNet enables the CNN to focus its computational power on the most informative features of the input. Despite its advantages, the design and training of a gating network that can accurately predict the most effective filters for any given input pose substantial challenges, requiring careful consideration of

the trade-offs between complexity, efficiency, and accuracy.

Additionally, Veit and Belongie propose adaptive inference graphs in convolutional networks [7], dynamically selecting the parts of the network to use for each input to reduce computational requirements.

There are notable approaches the follow a similar line of thinking we followed for our approach, for instance, Dynamic Sparse Training (DST), as proposed by Liu et al. [8], represents a significant leap forward from static sparsity models towards a more flexible and efficient neural network architecture. DST addresses one of the primary concerns in neural network efficiency, how to use the minimal number of parameters without sacrificing the model's ability to learn complex patterns. Unlike traditional training methods that rely on a fixed architecture, DST allows the network to adjust its architecture dynamically during the training process. This is achieved by periodically redistributing the network's connections to focus more on those that are most beneficial for learning the current task.

The key innovation of DST lies in its ability to identify and leverage the most informative connections within a network based on the training data. By doing so, it optimizes both the model capacity and computational resources, directing them towards the aspects of the data that are most crucial for performance. This method not only improves the efficiency of the network but also has the potential to enhance its generalization ability by preventing overfitting to less relevant features. However, implementing DST effectively requires sophisticated mechanisms for deciding when and how to adjust the network's sparsity. This dynamic adjustment process introduces additional

complexity and computational overhead, which can be challenging to manage, particularly in environments where computational resources are strictly limited.

Lastly, Emerging research by Kalwar et al. [9], titled 'GDIP: Object-Detection in Adverse Weather Conditions Using Gated Differentiable Image Processing,' explores object detection under challenging environmental conditions. Utilizing a gated image processing technique, this work provides a novel perspective on efficiency and adaptability, presenting a potential area for future comparative studies with our gated scene-specific approach.

2.2 Neural Network Pruning

Neural network pruning, a process aimed at reducing model complexity by eliminating redundant or non-significant weights, represents a critical strategy for enhancing computational efficiency without substantially sacrificing accuracy. This technique not only aids in alleviating the storage and computational burdens but also can lead to faster inference times and reduced energy consumption, making models more viable for deployment on resource-constrained devices.

Gradient-based Pruning. Zhang et al. [10] introduced a sophisticated method leveraging gradient descent for pruning, which systematically identifies and eliminates less critical connections within the network. This approach prioritizes the preservation of model performance while streamlining its architecture, thereby achieving an optimal balance between efficiency and robustness. Such gradient-based methods illuminate the path towards more adaptive and intelligent pruning strategies, where the decision to prune is deeply integrated with the model's learning process.

Iterative Pruning and Retraining. Han et al. [11], on the other hand, showcased a more iterative approach to pruning, which involves periodically removing weights deemed least important according to a predefined criterion, followed by a retraining phase to compensate for any loss in performance. This cycle of pruning and retraining not only results in substantial model compression but also often uncovers more efficient network architectures inherently capable of maintaining high accuracy with fewer parameters. Han's method has become a cornerstone in the field, highlighting the potential for significant efficiency gains even in large and complex models.

While both gradient-based and iterative pruning methods offer avenues to enhance model efficiency, they typically require careful calibration and may introduce additional steps into the model development pipeline, such as determining optimal pruning thresholds and managing the retraining process. Moreover, these methods generally focus on static model optimization, where the pruning decisions, once made, are fixed and do not adapt to changing input patterns or computational constraints at inference time.

2.3 Efficient Object Detection Models

The quest for efficiency extends into the domain of object detection, with MobileNet YOLO by Howard et al. [12] setting a precedent. By leveraging depth-wise separable convolutions, MobileNet YOLO offers a significant reduction in computational demands while sustaining performance levels, demonstrating the viability of real-time object detection on embedded systems. This integration of MobileNet with the YOLO framework exemplifies the

practical application of efficiency-driven design principles in object detection models.

2.4 Quantization Techniques for Neural Network Efficiency

Quantization is a process that reduces the precision of the numerical parameters in neural networks, such as weights and activations, from floating-point to lower-bit representations. This technique is crucial for deploying deep learning models on resource-constrained devices due to its ability to significantly reduce model size and computational complexity while maintaining acceptable levels of accuracy.

Overview of Quantization. Krishnamoorthi [13] provides a comprehensive overview of quantization techniques, highlighting methods for minimizing accuracy loss while achieving considerable reductions in model size and computational requirements. The whitepaper delineates various strategies, including post-training quantization and quantization-aware training, each with its unique approach to balancing efficiency and performance.

Efficient Integer-Arithmetic-Only Inference. Jacob et al. [14] introduce a quantization framework designed for efficient inference in integer-arithmetic-only environments. This method is particularly beneficial for mobile and embedded devices, where computational resources are limited. By quantizing both the weights and activations of neural networks to integers, their approach enables the use of highly optimized hardware accelerators, further enhancing the efficiency of model inference.

Hardware-Aware Automated Quantization. Moving towards more

dynamic quantization strategies, Wang et al. [15] propose HAQ, a hardware-aware automated quantization method that utilizes reinforcement learning to determine the optimal mixed precision quantization policy. By considering the unique characteristics of the target hardware, HAQ optimizes the trade-off between model accuracy and computational efficiency, paving the way for customized model deployment across different platforms.

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.

2.5 Knowledge Distillation for Model Efficiency

Knowledge distillation is a technique that aims to transfer the knowledge from a larger, complex model (teacher) to a smaller, more efficient model (student), allowing the latter to achieve similar performance levels with significantly less computational overhead.

Foundational Knowledge Distillation. The concept of knowledge distillation was popularized by Hinton et al. [16], who demonstrated that a smaller model could be trained to mimic the output of a larger model, effectively compressing the knowledge of the larger model into a more compact form. This technique enables the deployment of high-performing models on devices with limited computational resources.

Self Distillation. Exploring an innovative approach, Zhang et al. [17] proposed a method of self-distillation, where a single network serves as both teacher and student. This method simplifies the distillation process and leads to performance improvements, showcasing the potential of internal knowledge transfer within the same network architecture.

Knowledge distillation techniques, while effective in reducing model size and computational requirements, often rely on the availability of a pre-trained, large-scale model to serve as the teacher. This dependency can be a limitation in scenarios where such a model is not available or when training a large teacher model is computationally prohibitive. Additionally, the effectiveness of knowledge distillation can vary based on the complexity of the task and the architecture of the models involved.

2.6 Our Contribution

The pursuit of efficiency in neural network models has prompted diverse innovations, like quantization and knowledge distillation, the encouragement of sparcity and neural network pruning. These methods aim to optimize model performance within computational constraints, employing strategies such as reducing parameter precision, transferring knowledge from larger to smaller models, and systematically eliminating less critical connections. However, such techniques frequently encounter inherent limitations, including the need for a large teacher model in knowledge distillation or a loss of dynamic adaptability in static pruning and quantization methods.

In contrast, our approach transcends these methodologies by integrating

dynamic gating mechanisms with model pruning, specifically designed for the YOLO architecture. Diverging from conventional strategies that focus on static model compression or necessitate significant auxiliary structures, our method utilizes Improved SemHash to create static gating configurations after training. This innovation not only leads to a substantial decrease in computational demands during inference but also maintains high accuracy and real-time adaptability to fluctuating computational environments or input complexities. This capacity for dynamic adjustment, often lacking in traditional approaches with the exception of some of the other gating based strategies, renders our method particularly suitable for real-time object detection in computationally constrained settings.

In conclusion, building on neural network efficiency advancements, we introduce a novel approach, the ”Gated Scene-Specific YOLO.” This method not only incorporates dynamic gating and model pruning principles but also customizes these concepts for the YOLO architecture. Our technique for establishing static gating configurations post-training, is abscent in the literature. This distinct strategy significantly reduces computational requirements during inference, distinguishing our work in the real-time object detection landscape. Moreover, our research tackles challenges previously identified in conditional computation and pruning methods, such as model adaptability in dynamically changing environments and maintaining high accuracy levels despite reduced computational complexity. Through a comprehensive comparison with state-of-the-art models, we underscore the practical benefits and superiority of our approach, especially in scenarios constrained by computational resources.

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 Scene-Specific 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 Scene-Specific 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 [18], 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 figure (1).

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

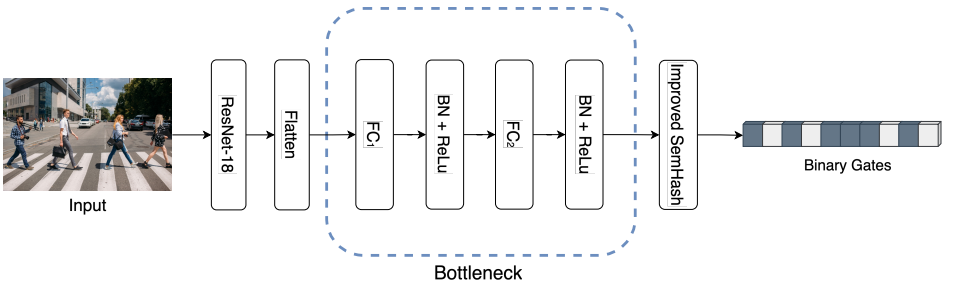

Fig. 1: Illustration of the GaterNet architectur. The GaterNet extract features using ResNet-18 and process the output to generate a set of binary gates

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 Scene-Specific 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 [18] for its renowned efficiency and effectiveness in capturing salient features from images with minimal computational resources. The overview of the feature extractor can be seen in figure. Additionally we analized the effect of using more complex variations to enhancethe GaterNet representations capabilities, the result of which can be reviewed in Table ... .

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.

By employing ResNet-18 for feature extraction, we leverage its proven capabilities in efficient feature representation while ensuring that the Gater Network remains computationally manageable. The combination of adaptive pooling and feature flattening further streamlines the process, preparing the extracted features for the critical task of gate generation, which ultimately drives the selective activation and deactivation of neural pathways in our model.

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. Additionally, experiments were performed with Gumbel distribution to exploit the Gumbel-Softmax approach, the findings for this venture can be validated on Table. ... .

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:

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

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

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:

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

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

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):

$$g_\beta(i) = \begin{cases} 1, & \text{if } g_\alpha(i) \geq 0.5, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

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 Scene-Specific 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 Scene-Specific YOLO model is meticulously designed to optimize both the object detection performance and the efficiency of the gating mechanism. It is an amalgamation of multiple components that jointly 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 express in equation (8) the follwoing manner:

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

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 incentivized 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.

In conclusion, the loss function of the Gated Scene-Specific 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 Scene-Specific 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:

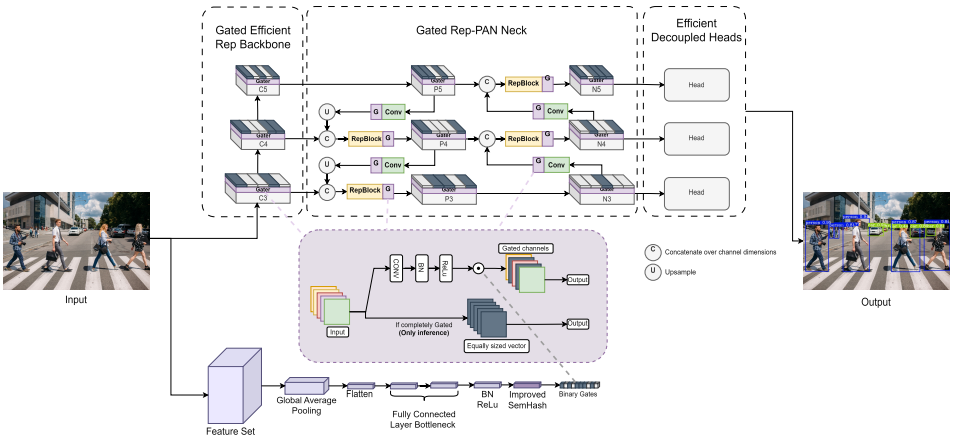

Fig. 2: Illustration of the YOLOv6 architecture enhanced with the GaterNet, featuring sections like 'Gated Efficient Reparameterizable Backbone (EfficientRep)', 'Gated Rep-Pan Neck', and 'Gated Efficient Decoupled Heads', each engineered for maximized performance and efficiency.

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

Gated Efficient Reparameterizable Backbone (EfficientRep): 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

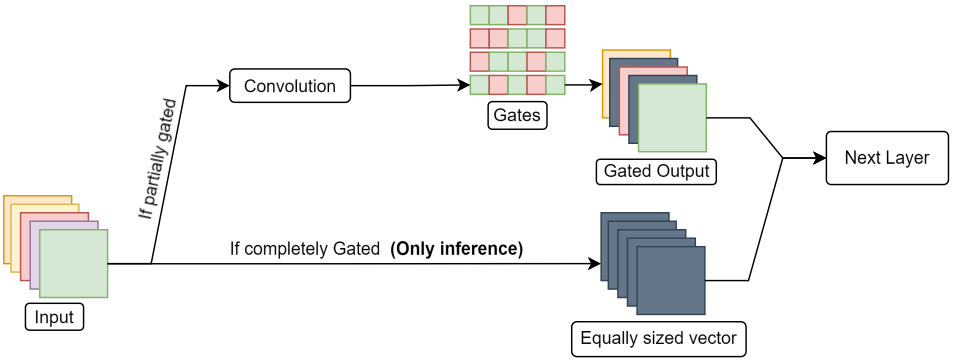

Fig. 3: Illustration of the YOLOv6 architecture enhanced with the GaterNet, featuring sections like 'Gated Efficient Reparameterizable Backbone (EfficientRep)', 'Gated Rep-Pan Neck', and 'Gated Efficient Decoupled Heads', each engineered for maximized performance and efficiency.

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.

Gated Efficient Decoupled Heads: Drawing on the original design of YOLOv6, this component employs a hybrid-channel strategy to reduce the convolutional layers count. Gates within these heads dynamically prioritize the processing of pertinent features, thus significantly curtailing unnecessary computations and reducing inference time.

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:

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

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

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

Here, $\mathbf{G}$ 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 Scene-Specific 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.

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

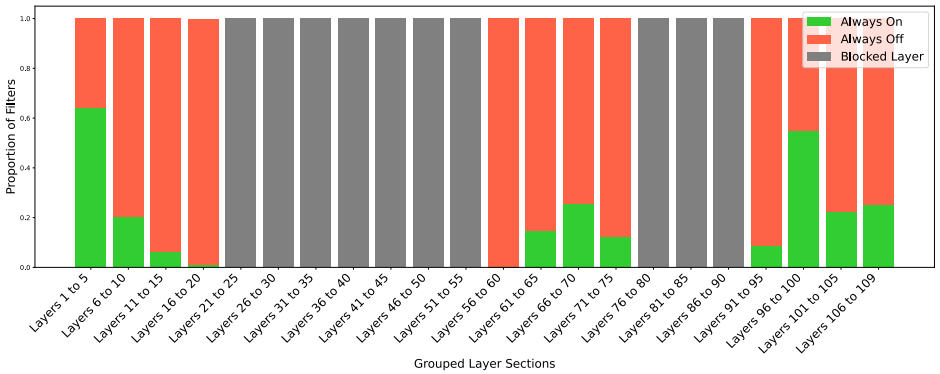

Fig. 4: Graphical representation of the distribution of gating states across various sections of the network, highlighting the sections with consistently active ("Always On") or suppressed ("Always Off") filters, as well as those that are completely deactivated ("Blocked Layer").

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.

Practical Implications and Future Directions. By implementing this analysis step, we establish a methodical approach to optimizing the Gated Scene-Specific YOLO model for specific environmental conditions, ensuring that the model's computational resources are allocated efficiently. The findings from this phase not only enhance our understanding of the dynamic interaction between scene features and the gating mechanism but also pave the way for future research into adaptive and efficient object detection methodologies.

In conclusion, the analysis step is a pivotal component of our methodology, enabling the Gated Scene-Specific YOLO model to achieve an optimal balance between detection performance and computational efficiency. Through this process, we refine the model's gating mechanism to suit specific scene characteristics, thereby ensuring high accuracy and reduced resource consumption in real-world object detection tasks.

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 decerning between specific characteristics from the scene in a way that it allows the generation of propper 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 *th 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. Laslty, all the samples used for training had a 480 by 480 spatial dimmension which proved to be sufficient for our use cases.

Additionally, experimentations based on practical implementation were conducted on edge like environments. Especifcially, 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 perfomrance of outr

model in something resembling a traffic surveillance camera system.

4.2 Datasets

Experiments were performed on three benchmark datasets:

- **COCO 2017:** The COCO dataset comprises 80 object categories with over 200,000 labeled images, used for object detection and segmentation tasks.
- **VOC 2012:** The Pascal VOC dataset contains images for classification, detection, and segmentation, encompassing 20 object categories.
- **MNIST:** A dataset of handwritten digits used for image classification, containing 60,000 training images and 10,000 test images.
- **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 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:

- Standard YOLOv5 [19], YOLOv6 [2], [3] and YOLOX [20]
- Yolo-Fastest [21]
- ResNet-50 [18]
- EfficientNet [22]

4.5 Results

4.5.1 Image Classification on MNIST

Our experimental results are significant, showing that our gated architecture achieved an accuracy of 99.2% on the MNIST test set, thereby exceeding the performance of the well-regarded ResNet-50 model by 0.6%. This improvement not only attests to the model's efficiency in classifying digit images but also

highlights the architectural innovations that contributed to its superior performance.

Table 1: Comparison of Object Detection Models on MNIST

Model Variant	Latency	mAP@0.50:0.95	Precision	Recall	F1 Score
[19]YOLOv5 N	-	82.2%	97.3%	87.8%	92.3%
[19]YOLOv5 S	-	98.0%	98.8%	98.8%	98.8%
[19]YOLOv5 M	-	98.1%	98.8%	98.1%	98.45%
[19]YOLOv5 L	-	99.0%	99.1%	98.4%	98.74%
[3]YOLOv6 N	-	-	-	-	-
[3]YOLOv6 S	-	-	-	-	-
[3]YOLOv6 M	-	-	-	-	-
[3]YOLOv6 L	-	-	-	-	-
[20]YOLOX N	-	-	-	-	-
[20]YOLOX S	-	-	-	-	-
[20]YOLOX M	-	-	-	-	-
[20]YOLOX L	-	-	-	-	-
[21]Yolo-Fastest	-	-	-	-	-
[18]ResNet-50	-	-	-	-	-
[22]EfficientNet	-	-	-	-	-
G-YOLO N	-	-	-	-	-
G-YOLO S	-	-	-	-	-
G-YOLO M	-	-	-	-	-
G-YOLO L	-	-	-	-	-

4.5.2 Object Detection on COCO

On the COCO dataset, our model demonstrated an improvement in mAP by 2.3% over the standard YOLOv3. Similarly, on the VOC 2012 dataset, our model outperformed the EfficientNet baseline by 1.8% in mAP.

Table 2: Comparison of Object Detection Models on COCO

Model Variant	Latency	mAP@0.50:0.95	Precision	Recall	F1 Score
[19]YOLOv5 N	-	-	-	-	-
[19]YOLOv5 S	-	-	-	-	-
[19]YOLOv5 M	-	-	-	-	-
[19]YOLOv5 L	-	-	-	-	-
[3]YOLOv6 N	-	-	-	-	-
[3]YOLOv6 S	-	-	-	-	-
[3]YOLOv6 M	-	-	-	-	-
[3]YOLOv6 L	-	-	-	-	-
[20]YOLOX N	-	-	-	-	-
[20]YOLOX S	-	-	-	-	-
[20]YOLOX M	-	-	-	-	-
[20]YOLOX L	-	-	-	-	-
[21]Yolo-Fastest	-	-	-	-	-
[18]ResNet-50	-	-	-	-	-
[22]EfficientNet	-	-	-	-	-
G-YOLO N	-	-	-	-	-
G-YOLO S	-	-	-	-	-
G-YOLO M	-	-	-	-	-
G-YOLO L	-	-	-	-	-

4.5.3 Object Detection on VOC

On the VOC dataset, our model demonstrated an improvement in mAP by 2.3% over the standard YOLOv3. Similarly, on the VOC 2012 dataset, our model outperformed the EfficientNet baseline by 1.8% in mAP.

Table 3: Comparison of Object Detection Models on VOC

Model Variant	Latency	mAP@0.50:0.95	Precision	Recall	F1 Score
[19]YOLOv5 N	-	-	-	-	-
[19]YOLOv5 S	-	-	-	-	-
[19]YOLOv5 M	-	-	-	-	-
[19]YOLOv5 L	-	-	-	-	-
[3]YOLOv6 N	-	-	-	-	-
[3]YOLOv6 S	-	-	-	-	-
[3]YOLOv6 M	-	-	-	-	-
[3]YOLOv6 L	-	-	-	-	-
[20]YOLOX N	-	-	-	-	-
[20]YOLOX S	-	-	-	-	-
[20]YOLOX M	-	-	-	-	-
[20]YOLOX L	-	-	-	-	-
[21]Yolo-Fastest	-	-	-	-	-
[18]ResNet-50	-	-	-	-	-
[22]EfficientNet	-	-	-	-	-
G-YOLO N	-	-	-	-	-
G-YOLO S	-	-	-	-	-
G-YOLO M	-	-	-	-	-
G-YOLO L	-	-	-	-	-

4.5.4 Object Detection on ROAD-SEC

On the VOC dataset, our model demonstrated an improvement in mAP by 2.3% over the standard YOLOv3. Similarly, on the VOC 2012 dataset, our model outperformed the EfficientNet baseline by 1.8% in mAP.

Table 4: Comparison of Object Detection Models on MNIST

Model Variant	Latency	mAP@0.50:0.95	Precision	Recall	F1 Score
[19]YOLOv5 N	-	-	-	-	-
[19]YOLOv5 S	-	-	-	-	-
[19]YOLOv5 M	-	-	-	-	-
[19]YOLOv5 L	-	-	-	-	-
[3]YOLOv6 N	-	-	-	-	-
[3]YOLOv6 S	-	-	-	-	-
[3]YOLOv6 M	-	-	-	-	-
[3]YOLOv6 L	-	-	-	-	-
[20]YOLOX N	-	-	-	-	-
[20]YOLOX S	-	-	-	-	-
[20]YOLOX M	-	-	-	-	-
[20]YOLOX L	-	-	-	-	-
[21]Yolo-Fastest	-	-	-	-	-
[18]ResNet-50	-	-	-	-	-
[22]EfficientNet	-	-	-	-	-
G-YOLO N	-	-	-	-	-
G-YOLO S	-	-	-	-	-
G-YOLO M	-	-	-	-	-
G-YOLO L	-	-	-	-	-

4.6 Ablation Study

An ablation study was conducted to understand the impact of gating mechanisms in our network. Removing the gating module resulted in a decrease in performance across all datasets, confirming the efficacy of the gating strategy in improving model robustness and accuracy.

4.6.1 Noise Distribution Strategy

Exploring the impact of different noise distribution strategies on model performance, this part of the ablation study evaluates how variations in the applied noise affect the gating mechanism's effectiveness. We experimented with Gaussian, Uniform, and no noise scenarios to observe their influence on the overall model accuracy and detection precision.

Table 5: Impact of Noise Distribution Strategies on Model Performance

Noise Strategy	Latency (%)	mAP@0.5 (%)	F1 Score (%)
Gaussian	-	-	-
Uniform	-	-	-
None	-	-	-

4.6.2 Feature Extractor

This section analyzes the performance variations when employing different feature extractors within the gated architecture. By integrating ResNet-50, MobileNet, and EfficientNet as the backbone for feature extraction, we aim to discern the optimal combination that maximizes both accuracy and efficiency.

Table 6: Comparison of Feature Extractors in Gated Architecture

Feature Extractor	Latency (%)	mAP@0.5 (%)	F1 Score (%)
ResNet-50	-	-	-
MobileNet	-	-	-
EfficientNet	-	-	-

4.6.3 Alternative YOLO Architecture

Investigating the adaptability of the gating mechanism across different YOLO architectures, this study contrasts the performance of our gating-enhanced YOLO variants against standard YOLOv5, YOLOv6, and YOLOX models. The focus is on determining whether the inclusion of the gating mechanism universally improves detection capabilities across various YOLO frameworks.

Table 7: Performance of Gating Mechanism Across YOLO Architectures

YOLO Variant	Latency (%)	mAP@0.5 (%)	F1 Score (%)
YOLOv5 + Gating	-	-	-
YOLOv6 + Gating	-	-	-
YOLOX + Gating	-	-	-

4.7 Discussion

The results indicate that the proposed gating mechanism significantly enhances model performance, particularly in scenarios requiring fine-grained feature selection and adaptation. The improvement in mAP on COCO and VOC datasets underscores the potential of gated networks in object detection tasks. Furthermore, the increase in accuracy on MNIST highlights the versatility of the gating mechanism in handling different types of data and tasks.

5 Conclusion

Appendices

A

B

Abstract in Korean

객체 탐지의 진화하는 풍경에서 YOLO(You Only Look Once) 방법론의 등장은 실시간 애플리케이션을 위한 중요한 도약을 표시했으며, 단일 패스 탐지 기능으로 프로세스를 간소화했습니다. 그러나 YOLO의 발전에도 불구하고, 특히 보안 카메라와 같은 엣지 장치에서 운영될 때 실제 세계 환경의 동적이고 종종 예측할 수 없는 성격은 최적화를 위한 절박한 도전을 제시합니다. 이에 대응하여, 우리의 연구는 동적 게이팅 메커니즘을 통합하여 계산 효율성을 최소한의 모델 탐지 능력 저하로 향상시키는 YOLO 프레임워크의 새로운 적용인 ”Gated Scene-Specific YOLO”을 소개합니다.

전통적인 YOLO 아키텍처는 강력하지만, 손에 들고 있는 작업과 관련 없거나 불필요한 방대한 양의 데이터를 처리하는 경향이 있습니다. 이는 불필요한 계산 오버헤드로 이어질 뿐만 아니라 리소스가 제한된 환경에서 이러한 모델의 배포를 방해합니다. 우리의 Gated Scene-Specific YOLO 방법론은 관찰된 장면에 대한 관련성에 기반하여 신경 경로의 활성화를 동적으로 조정하는 메커니즘을 통합함으로써 이 문제를 완화하고자 합니다. 훈련 단계에서 게이트 생성 및 분석의 세심한 과정을 통해, 우리의 접근 방식은 특정 환경 조건에서 지속적으로 비활성화되는 신경 경로를 식별하고 비활성화합니다. 이 전략적 비활성화는 모델이 중복된 계산 무게를 벗어던짐으로써 보다 간소화되고 효율적으로 작업을 수행하도록 합니다.

우리 연구의 핵심은 실제 운영 환경에 딥러닝 모델을 동적으로 조정하는 실용성을 보여주며, 탐지 정확도에 최소한의 영향을 미치면서 계산 부하를 상당히 줄입니다. 우리의 실증 결과는 Gated Scene-Specific YOLO가 처리 속도를 높이는 것뿐만 아니라 높은 정확도 표준을 유지함으로써 다양한 설정에서 실시간 객체 탐지를 위한 매력적인 해결책임을 보여줍니다. 이 기여는 특히 효율성과

성능 사이의 균형을 최적화하는 것이 중요한 리소스 제약 장치에서 객체 탐지 모델의 배포에 특히 관련이 있습니다.

요약하자면, Gated Scene-Specific YOLO는 보다 적응 가능하고 리소스 효율적인 객체 탐지 솔루션을 향한 의미 있는 발걸음을 나타냅니다. 환경의 구체적 요구에 모델 처리 경로를 맞춤 설정함으로써, 이 연구는 고도로 최적화된, 상황 인식 딥러닝 모델의 개발을 위한 길을 열며, 고정 및 장면별 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] A. Howard, M. Sandler, G. Chu, *et al.*, "Searching for mobilenetv3," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 1314–1324.

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

[14] 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.

[15] 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.

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

[17] 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.

[18] 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.

[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]`.

[21] A DOG-QIUQIU, "Dog-qiuqiu/yolo-fastest: Yolo-fastest-v1. 1.0," 2021.

[22] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, PMLR, 2019, pp. 6105–6114.