IJIE

INTERNATIONAL JOURNAL OF INTEGRATED ENGINEERING
ISSN: 2229-838X e-ISSN: 2600-7916

Vol. 18 No. 3 (2026) 389-409
https://publisher.uthm.edu.my/ojs/index.php/ijie

GCB-YOLO7: Enhancing Lightweight Face Detection with Ghost Module and Convolutional Block Attention Module

Ibrahim Al Amoudi[1], Dzati Athiar Ramli[1]*

[1] *School of Electrical and Electronic Engineering, USM Engineering Campus, Universiti Sains Malaysia, Nibong Tebal, 14300, Penang, Malaysia*

*Corresponding Author: dzati@usm.my
DOI: https://doi.org/10.30880/ijie.2026.18.03.026

Article Info

Received: 21 November 2025
Accepted: 15 April 2026
Available online: 30 April 2026

Keywords

Efficient neural networks, lightweight face detection, attention mechanism, object detection

Abstract

Convolutional Neural Networks (CNNs) have significantly advanced object detection recently. However, achieving the right balance between model size and detection accuracy is a persistent challenge, particularly for face detection in resource-constrained environments. While large models provide high accuracy, their computational demands make them impractical for real-time or embedded applications. Conversely, lightweight models are more efficient but often sacrifice accuracy. Existing lightweight YOLO variants trade accuracy for efficiency, particularly in detecting small or occluded faces. To address this, we proposed GCB-YOLO, a lightweight face detection model that balances compactness and performance. The design incorporates the Ghost Module (GM) to reduce model size, the Convolutional Block Attention Module (CBAM) to enhance feature representation, and the SiLU activation function to improve detection accuracy. Our evaluation on the Wider Face dataset demonstrates the effectiveness of GCB-YOLO. On the medium subset, it achieved 79.6% mAP50 and an 81.04% F1 score, surpassing other state-of-the-art lightweight models. The model maintained strong performance on the challenging hard subset with 60.5% mAP50, and a 65.37% F1 score. For the efficiency metrics, the parameter count was reduced by 49.44% (from 6.015M to 3.041M), and CPU inference speed improved from 4.265 FPS to 5.130 FPS. These results highlight GCB-YOLO's suitability for real-time face detection on resource-constrained devices.

1. Introduction

Face detection, a fundamental task in computer vision, has become increasingly crucial in various applications, including face recognition [1], facial emotion analysis [2], and more. Despite the significant advances in face detection applications, deploying these models in real-world applications remains a major challenge, particularly in environments constrained by limited computational resources. From surveillance systems and smartphones to AI-powered IoT devices, face detection is a fundamental component in many emerging technologies, yet many of these applications run on single-board computers (SBCs) and embedded systems, which offer limited computational power, storage, and memory [3].

Historically, face detection methods have transitioned from classical machine learning approaches, such as Haar Cascades [4], [5], which struggled to overcome challenges related to pose variations, illumination, and occlusions [6]. Recently, Convolutional Neural Networks (CNNs) have emerged as the superior approach, demonstrating robust performance across a wide array of detection tasks. Object detection models have proven

effective when adapted for face detection applications [9].However, CNN-based models typically fall into two distinct categories: heavyweight models and lightweight models [7]. Traditional heavyweight models excel in detecting faces under challenging conditions, but their large size and high computational requirements make them unsuitable for resource-constrained devices. Conversely, lightweight models, while addressing the need for real-time processing, often sacrifice accuracy, particularly in difficult detection scenarios such as small or partially obscured faces.

Furthermore, deployment on resource-constrained devices introduces additional challenges, including limited memory, processing power, and energy consumption [8]. Many existing models prioritize either accuracy or efficiency. For instance, YOLOv7 achieves high accuracy but comes with a larger model size, whereas YOLOv8 offers a more compact model but sacrifices some accuracy [10]. This growing divide between accuracy and efficiency underscores the pressing need for a model that strikes a balance between the two, particularly for real-time applications in resource-constrained environments. Addressing this challenge is critical for enabling the next generation of AI-driven technologies, from smart surveillance to edge computing devices, which rely on both accurate and efficient face detection.

This research addresses the need for efficient and accurate face detection on resource-constrained devices by presenting Ghost CBAM YOLOv7 (GCB-YOLOv7), a lightweight model derived from YOLOv7-tiny. Our approach integrates three existing techniques to reduce model size while maintaining or enhancing performance. First, the Ghost Module (GM) [30] was incorporated into the ELAN block, forming the proposed ELAN-GM block, which improves feature extraction while significantly reducing the model's overall size. Second, the Convolutional Block Attention Module (CBAM) [32] was integrated within the detector module to enhance feature representation, particularly in challenging detection scenarios. Our experiments determined that this placement yielded the most optimal results. Lastly, the SiLU activation function replaced Leaky-ReLU, providing further accuracy improvements without increasing the model size. The contributions of this research are summarized as follows:

1. We substitute selected convolution operations in the ELAN block with Ghost Modules [30], forming the ELAN-GM architecture to achieve parameter efficiency.
2. We replace convolution blocks in the detector module with CBAM [32] to enhance feature representation for challenging face detection scenarios.
3. We adopt SiLU activation function to increase detection accuracy without compromising model size.

These modifications collectively aim to create a balanced, lightweight face detection model capable of real-time processing on resource-limited devices while maintaining robust detection performance across various challenging scenarios. This work advances the state of lightweight face detection by addressing both computational constraints and the need for high accuracy in practical applications.

2. Related Work

2.1 Machine Learning Face Detection Models

The Viola-Jones algorithm, introduced in 2001, pioneered face detection using hand-crafted Haar-like features and the AdaBoost classifier in a cascaded structure [4], [5]. Known for its efficiency in low-powered CPUs, the algorithm achieved real-time performance. Despite its speed, Viola-Jones struggled in real-world conditions such as variations in pose, illumination, and facial expressions, limiting its robustness in complex scenarios. Efforts to improve its performance included modifications by Huang et al. (2019), which increased robustness to different face orientations and lighting conditions [11].

Other machine learning approaches, such as Principal Component Analysis (PCA) [12] and Histogram of Oriented Gradients (HOG) [13], were explored but faced similar challenges with non-frontal faces and uncontrolled environments. While lightweight and efficient, these approaches lacked the accuracy needed for real-world applications, highlighting the limitations of traditional methods in handling diverse face detection conditions [6]

2.2 Heavyweight Face Detection Models

With the rise of Convolutional Neural Networks (CNNs), deep learning models revolutionized face detection. Early models like FacenessNet introduced part-based detection, detecting facial components such as eyes and noses before assembling full faces [14], [15]. This approach significantly improved robustness against occlusions and pose variations, addressing major limitations of machine learning methods. However, FacenessNet struggled with detecting small faces due to the limited receptive field in its early layers, and its multi-stage detection process was computationally demanding, limiting its real-time applicability.

The introduction of single-stage object detectors advanced face detection. S3FD (Single Shot Scale-invariant Face Detector) introduced a scale-compensation anchor matching strategy and a max-out background label to address the challenges of detecting faces at various scales. S3FD's architecture allows it to handle a wide range of

face scales, particularly improving the detection of small faces compared to previous models [16]. While S3FD showed significant improvements in handling multi-scale face detection, the model still faced challenges with highly occluded faces and maintained a relatively large size, which limits its application in resource-constrained environments.

PyramidBox, built on SSD, introduced context-sensitive predictions and data-anchor sampling to further improve accuracy in detecting small and occluded faces [17]. RetinaFace took this further by incorporating multi-task learning, performing face detection, landmark localization, and 3D face reconstruction simultaneously [18]. TinaFace improved detection accuracy using innovations like Distance-IoU (DIoU) loss but maintained a heavyweight architecture [9]. YOLO-face, built on YOLOv3, introduced face-specific anchor boxes learned through k-means clustering and a hybrid regression loss function combining MSE and GIoU loss to improve multi-scale face detection performance [19]. These advancements, while improving accuracy, came at the cost of increased model size and computational demands, limiting their utility in real-time scenarios or on resource-limited devices.

The evolution of deep learning models in face detection has brought substantial improvements in accuracy over traditional machine learning approaches. Many of these models excel at face detection and incorporate features like facial keypoint detection and 3D face reconstruction, enhancing their versatility. However, this increased capability has come at the cost of larger model sizes and greater computational complexity. Consequently, while these deep learning models deliver superior performance in controlled environments with ample computational resources, they face challenges in edge deployments or real-time processing scenarios. This trade-off between accuracy and efficiency has driven the development of lightweight face detection models.

2.3 Lightweight Learning Face Detection Models

In response to the challenges posed by deep learning models in resource-constrained environments, lightweight face detection models have been developed to balance accuracy and efficiency. One of the early deep learning models is Multi-task Cascaded Convolutional Networks (MTCNN). It offers joint face detection and alignment through a cascaded structure of three simple cascaded networks—P-Net (Proposal Network), R-Net (Refine Network), and O-Net (Output Network)—which detect facial landmarks and bounding boxes in a unified framework. It can identify five facial landmarks: two eyes, a nose, and two mouth corners [20]. MTCNN was developed as a deep learning alternative to the machine learning models. It can achieve higher accuracy compared to traditional machine learning models. However, it struggles with detecting small and occluded faces.

FaceBoxes, optimized for CPU-based detection, uses a Rapidly Digested Convolutional Layers (RDCL) structure for fast inference, achieving real-time performance on CPUs [21], [22]. While effective for mobile applications, FaceBoxes suffers from accuracy issues when dealing with small faces and occlusions. BlazeFace, designed for mobile devices, achieves sub-millisecond inference times by utilizing Blaze Blocks, which are lightweight modules based on Depthwise Separable Convolutions (DSCs) [23]. While highly efficient, BlazeFace's simplified architecture limits its performance in detecting small faces or faces in crowded scenes.

Improved TinyYOLOv3, developed in 2021, incorporated DSCs, a CIoU loss function, and a channel attention mechanism to enhance small face detection while maintaining the efficiency of the YOLO architecture [7]. The improved model shows better performance on face detection tasks compared to the original YOLOv3-tiny, but it may still struggle with highly occluded faces or extreme profile views due to its compact design.

YOLO5Face, based on YOLOv5, added a five-point landmark regression head and employed Wing loss for improved landmark accuracy [24]. With multiple variants of differing sizes, YOLO5Face heavyweight versions excelled in accuracy, but their increased complexity limited their real-time and edge deployment feasibility, and YOLO5Face lightweight versions sacrificed performance in more complex detection tasks.

EfficientFace leverages EfficientNet, a scalable CNN architecture used in various models. Furthermore, it introduced cross-scale feature fusion and a CBAM attention mechanism to improve performance on occluded faces [25]. While EfficientFace offers competitive accuracy with reduced computational costs, it remains relatively heavyweight compared to other lightweight models.

Also based on YOLOv5 object detection model, the second version of YOLO-Face was built. YOLO-FaceV2 is another scalable model with several variants ranging from lightweight to heavyweight variants. YOLO-FaceV2 introduces a Receptive Field Enhancement (RFE) module that uses dilated convolutions to better detect multi-scale faces, a Separated and Enhancement Attention Module (SEAM) that focuses on occluded facial regions, and a Slide Weight Function (SWF) that adaptively balances hard and easy training samples [26].

FeatherFace, built on RetinaFace with MobileNet-0.25 backbone, introduced an ultra-lightweight architecture incorporating bidirectional feature pyramid networks, attention mechanisms, deformable convolutions, and channel shuffling to achieve optimal feature integration with minimal computational overhead [27]. Although the model demonstrated promising accuracy among ultralightweight models, its accuracy decreases when compared to heavier models that have more than 1 million parameters. This makes this model suitable for deployment in extremely resource-constrained environments.

As shown in Table 1, the review of existing face detection models reveals a clear trade-off: heavyweight models deliver superior accuracy but are impractical for real-time deployment in resource-constrained environments due to their high computational demands and slow inference speed. Conversely, lightweight models, while more suitable for such environments and offering high inference speeds, often compromise accuracy in favor of efficiency. This dichotomy underscores the pressing need for a balanced approach that can bridge the gap between accuracy and efficiency.

Table 1 *Comparison of the face detection models*

Model	Year	Key Features	Key Innovations	Limitations
Viola-Jones [4], [5]	2001	Haar-like features, AdaBoost, and cascade structure.	Fast, efficient on CPUs.	Poor performance on non-frontal faces and lighting variations.
FacenessNet [15]	2015	Part-based detection (facial components).	Robust to occlusions, pose variations	Struggles with small faces, computationally demanding.
MTCNN [20]	2016	Cascaded networks (P-Net, R-Net, O-Net)	Joint detection and alignment.	Issues with small, occluded faces.
S3FD [16]	2017	SDD-based, scale-compensation strategy.	Handles multi-scale detection to detect the small faces.	Large model size, occlusion issues.
FaceBoxes [21], [22]	2017	RDCL structure, multiple-scale convolutional layers.	Real-time CPU performance and anchor densification for small faces.	Struggles with small faces and accuracy limitations.
PyramidBox [17]	2018	SSD-based, context-sensitive predictions, pyramid anchors.	Improved detection of small, occluded faces.	Complex, resource-intensive architecture.
BlazeFace [23]	2019	Blaze Blocks, Double Blaze Blocks (DSC-based).	Sub-millisecond inference.	Issues with small faces and complex scenes.
RetinaFace [18]	2020	Multi-task learning (detection, landmark, 3D reconstruction).	Handles pose variations and occlusions.	Computationally heavy.
TinaFace [9]	2021	ResNet-50 backbone, DIoU loss, anchor-free.	High detection accuracy, handles small and occluded faces.	Large model size, high computational cost.
Improved TinyYOLOv3 [7]	2021	DSCs, CIoU loss, and channel attention.	Improve the identification of small faces.	Struggles with occlusions and extreme profiles.
YOLO-face [19]	2021	YOLOv3-based, face-specific anchor boxes, hybrid regression loss	Face-specific anchor boxes learned through k-means clustering, hybrid regression loss (MSE and GIoU)	Increased model size and computational demands, limiting real-time deployment
YOLO5Face [24]	2023	YOLOv5-based, five-point landmark regression.	Scalable model size and accuracy.	Lightweight variants sacrifice accuracy.
EfficientFace [25]	2023	EfficientNet backbone, cross-scale fusion, and CBAM attention.	Competitive accuracy with efficiency.	The size is still relatively large for lightweight models.
YOLO-FaceV2 [26]	2024	YOLOv5-based, RFE module, SEAM, SWF	RFE with dilated convolutions, SEAM for occluded regions, SWF for sample balancing	Trade-offs between lightweight and heavyweight variants
FeatherFace [27]	2025	RetinaFace with MobileNet-0.25, ultra-lightweight	Bidirectional FPN, attention mechanisms, deformable convolutions, channel shuffling	Accuracy drops compared to models with >1M parameters

Int. Journal of Integrated Engineering Vol. 18 No. 3 (2026) p. 389-409

3. Methods

3.1 YOLOv7-tiny Network

The choice of YOLOv7-tiny as the base model is motivated by its superior accuracy in the COCO2017 object detection dataset compared to other YOLO series models, despite its larger parameter count [28]. In the context of lightweight face detection, YOLOv7-tiny has demonstrated higher accuracy than YOLOv8n, although YOLOv8n is nearly half the size [29]. This trade-off between accuracy and model size presents an opportunity for optimization, which GCB-YOLO7 aims to address. Fig. 1 shows the full architecture of YOLOv7-tiny.

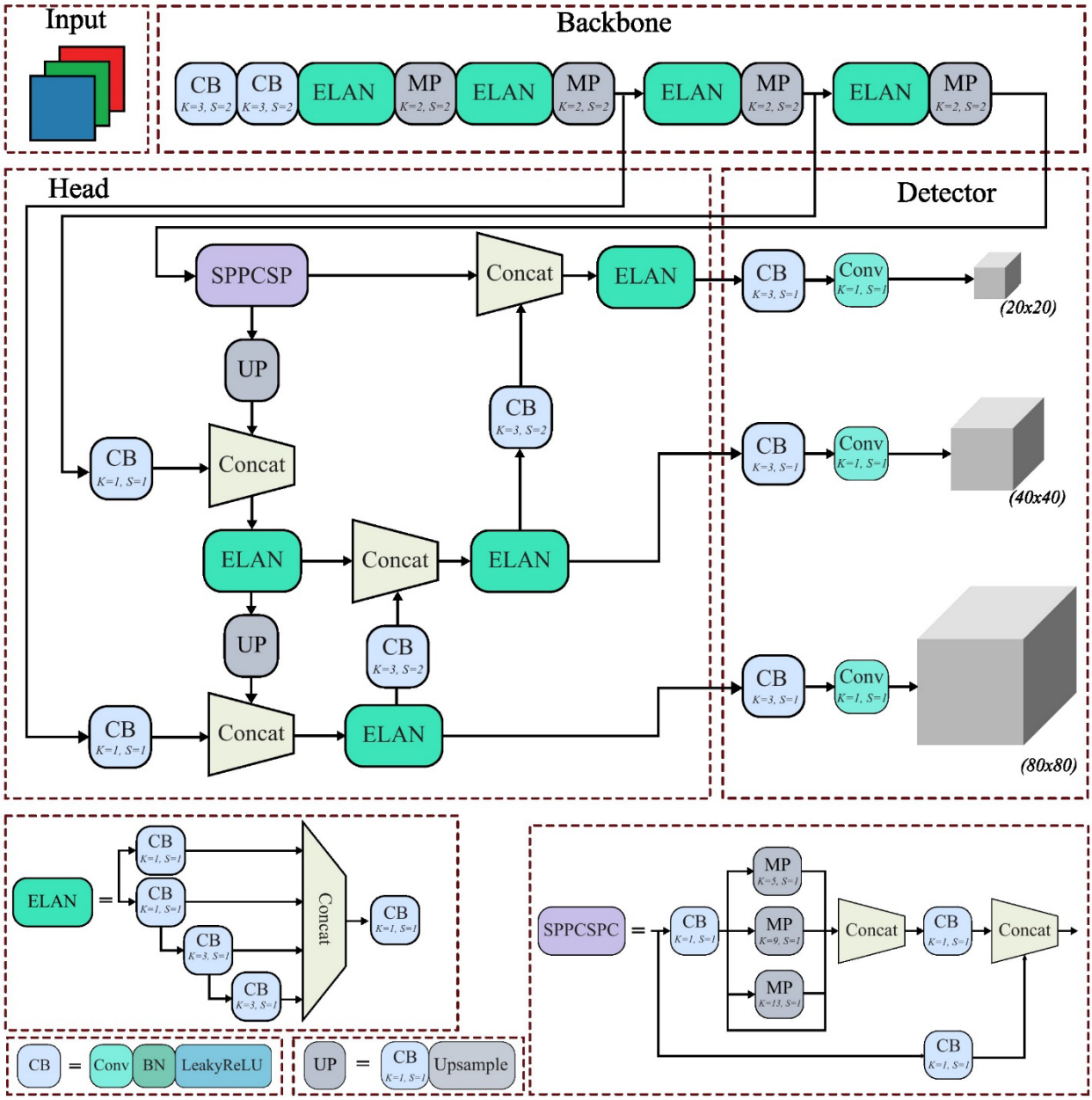

Fig. 1 *YOLOv7-tiny overall architecture*

The YOLO7 architecture consists of two main components: the backbone and the head, following the standard structure of object detection models. YOLOv7-tiny is the smallest variant of YOLOv7. Its backbone architecture uses a modified ELAN (Efficient Layer Aggregation Network) structure, which is simpler than the E-ELAN used in the full YOLOv7. The backbone is built using Convolutional Blocks (CB), a simplified version of ELAN modules, and max-pooling (MP) operations. The CB blocks consist of a convolution layer, followed by batch normalization and an activation function. The simplified ELAN module in YOLOv7-tiny features two main branches: one passes through a 1x1 CB, and the other passes through a 1x1 CB and two 3x3 CBs. The outputs from both branches and the input are concatenated and passed through a final 1x1 CB. This structure allows the network to extract diverse

feature information efficiently. For down-sampling, YOLOv7-tiny uses MP modules with a kernel size and stride of 2, effectively halving the spatial dimensions of the feature maps. The backbone progressively increases the number of channels (32→ 64→ 128→ 256→ 512) while reducing spatial dimensions and creating a feature pyramid.

The YOLOv7-tiny head network enhances multi-scale feature fusion by employing SPPCSP (Spatial Pyramid Pooling Cross Stage Partial), simplified ELAN, and CB modules. The SPPCSP module starts with two parallel 1x1 CBs, both reducing the number of channels to 256. The output of one branch is fed into three parallel MP operations with kernel sizes of 5, 9, and 13 and a stride of 1. The outputs of the MP operations and the original CB are concatenated, passed through another 1x1 CB, then concatenated with the other initial branch, and finally processed by a 1x1 CB. This allows the SPPCSP block to capture multi-scale information effectively. The head then uses up-sampling and concatenation operations to fuse features from different scales, followed by simplified ELAN modules for further processing. The network produces feature maps at three different scales (P3, P4, P5) for detecting objects of various sizes. These feature maps are then processed by the final Detect layer, which applies convolutional layers to produce the final output tensors. The Detect layer handles the conversion of these outputs into bounding box predictions, object confidence scores, and class probabilities. It also manages anchor-based detection and grid-based offset calculations. This sophisticated detection mechanism allows YOLOv7-tiny to efficiently predict object locations and classifications across multiple scales while maintaining a balance between computational efficiency and detection accuracy.

3.2 Ghost Module

The Ghost Module (GM) is an innovative, lightweight block designed for efficient feature extraction in convolutional neural networks. It addresses the computational expense of conventional convolutions, especially when dealing with large kernel sizes or numerous channels. The core idea of GM is to generate more features using fewer parameters and calculations. The GM operates in two stages:First, intrinsic feature extraction, which is an ordinary convolution with a reduced number of output channels, is applied to extract intrinsic feature maps. This is typically implemented as a pointwise 9 (1x1) convolution and can be expressed in (1).

$$Y' = X * f_{1x1} \tag{1}$$

Where $Y' \in R^{H \times W \times C'out}$ represents the intrinsic feature maps, X is the input, and f_{1x1} is the pointwise convolution filter. Second Subsequently, GM applies a series of cheap linear operations to generate the remaining features, called "ghost features." In the original GhostNet design, these cheap operations are primarily depth-wise convolutions. This step can be represented as in (2).

$$Y = Concat([Y', Y' * F_{dp}]) \tag{2}$$

Where f_{dp} is the depth-wise convolutional filter, and $Y \in R^{H \times W \times Cout}$ Cout is the output feature map. The parameter count for GM is determined by summing the parameter count of its two components: intrinsic feature extraction and ghost operation. To calculate the parameter count for intrinsic feature extraction, we can use equation (3).

$$P_{Intrinsic} = C_{in} \times C' \tag{3}$$

Where C_{in} is the number of input channels and C' is the number of intrinsic feature maps. The ghost operation's parameter count, which in this case is the depth-wise convolution, is calculated using (4).

$$P_{ghost} = C' \times k^2 \times (s-1) \tag{4}$$

Where k is the kernel size of the depth-wise convolution. The total parameter count for the GM is the sum of both parameters count as in (5).

$$P_{GM} = P_{Intrinsic} + P_{ghost} = C'(C_{in} \times k^2 \times (s-1)) \tag{5}$$

In contrast, a standard convolution would require (6):

$$P_{Conv} = C_{in} \times C_{out} \times k^2 \tag{6}$$

Int. Journal of Integrated Engineering Vol. 18 No. 3 (2026) p. 389-409

The primary advantage of GM lies in its ability to achieve performance comparable to ordinary convolution at a significantly lower computational cost. By generating "ghost" features through cheap operations, the GM reduces the number of parameters and calculations required, making it particularly useful for deployment in resource-constrained environments. This two-step approach allows the Ghost Module to maintain feature richness while substantially reducing computational complexity. The module effectively mimics the output of standard convolutions by learning a set of intrinsic features and then applying linear transformations to create ghost features, resulting in a more efficient feature extraction process [30], [31]. The difference between the GM process and normal convolution is illustrated in Fig. 2.

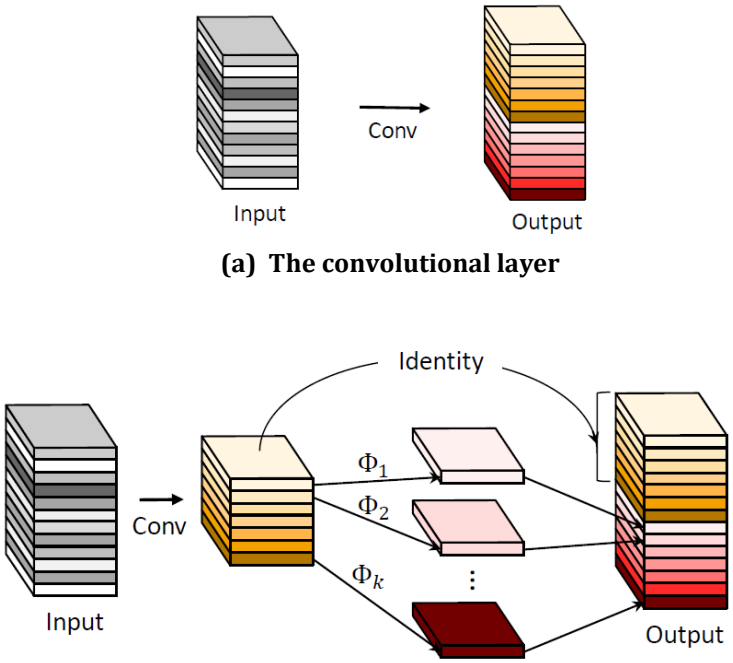

Fig. 2 *An illustration of the convolutional layer and the GM for outputting the same number of feature maps. Φ represents the cheap operation [30]*

3.3 ELAN Ghost Module (ELAN-GM) Block

As described earlier in the original ELAN module, the second branch consists of a CB with a 1x1 kernel size and two 3x3 CB modules. The 3x3 CB modules are larger in size and parameter count compared with the 1x1 CB module. In ELAN-GM, the 3x3 module was replaced with the GM module to reduce the number of parameters. The 1x1 CB modules in both branches remained the same. Also, the CB module after the concat layer has a large number of input channels, which also increases the number of parameters and model size. These models were replaced with GM. This design was implemented to reduce the size of the large module without affecting the accuracy of the model. The difference between ELAN and ELAN-GM is shown in Fig. 3.

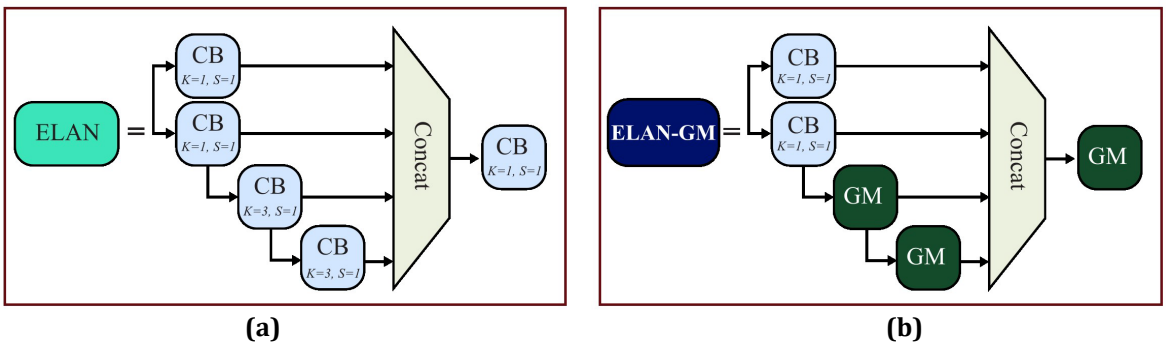

Fig. 3 *The difference between ELAN and ELAN-GM blocks. (a) The original ELAN block; (b) The proposed ELAN-GM block*

3.4 Convolutional Block Attention Module

Convolutional Block Attention Module (CBAM) [32] is a dual attention module that incorporates both channel and spatial attention modules. The channel attention module (CAM) in CBAM focuses on what information is important within a feature map. It does this by applying max pooling and average pooling operations across the width and height dimensions of the original feature map (F). This reduces the spatial information, resulting in two 1D vectors with a length equal to the number of channels (C) in the feature map. These two 1D vectors are fed into the multi-layer perceptron (MLP) separately. The MLP typically has one hidden layer, which performs linear transformations and applies a non-linear activation function on each input vector (max-pooled and average-pooled features) independently. Each input vector goes through its own transformation within the MLP, resulting in two separate 1D output vectors with the same dimensionality as the number of channels (C). These vectors represent the transformed and weighted channel attention scores based on max pooling and average pooling, respectively. These two 1D vectors are merged using element-wise summation. The equation of CAM is as follows (7):

$$M_c(F) = \sigma(MLP(AvgPool(F)) + MLP(MaxPool(F)) \quad (7)$$

Where F is the input feature map, $M_{ci}(F)$ is the channel attention map $M_c \in R^{C\times1\times1}$, and σ is the sigmoid activation function. Fig. 4 shows the general architecture of the CAM module of CBAM.

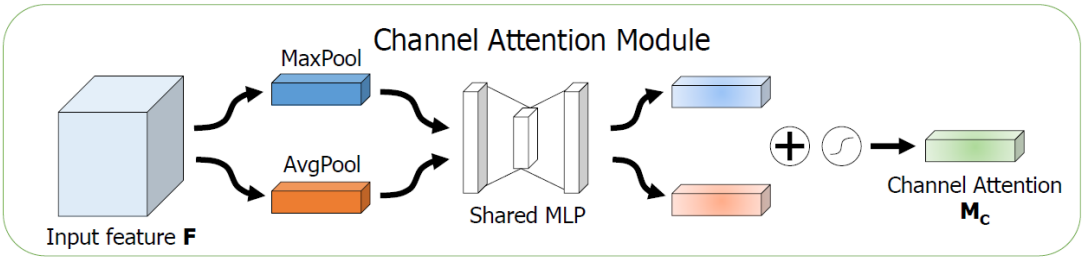

Fig. 4 *General architecture of CAM in CBAM [32]*

The spatial attention module (SAM) in CBAM complements the CAM module by focusing on where the important information resides within a feature map. It achieves this by analyzing the feature map along the channel dimension. First, it performs separate average pooling and max pooling along the channel axis. These operations result in two separate 2D feature maps, retaining the original height and width but with only one channel each. These feature maps are then concatenated, creating a single 2D descriptor with two channels. Finally, a small convolution layer with one output channel is applied to this descriptor. This step allows the module to learn spatial relationships between the combined channel information. Finally, a small convolution layer with one output channel is applied to this descriptor. This step allows the module to learn spatial relationships between the combined channel information. The equation of SAM can be defined as follows (8):

$$M_s(F) = \sigma(F_{7\times7}([AvgPool(F); MaxPool(F)])) \quad (8)$$

where $M_s(F) \in R^{1\times H\times W}$, σ is the sigmoid function, and $F_{7\times7}$ is a convolution layer with a 7×7 kernel size. Fig. 5 shows the CAM and SAM used in CBAM architecture.

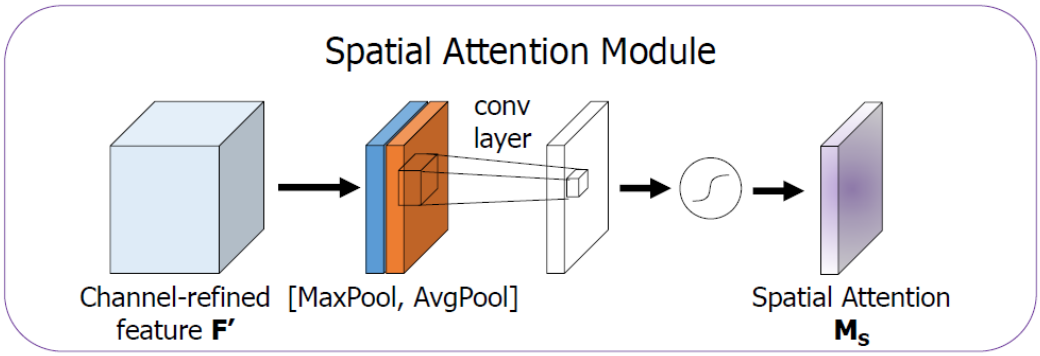

Fig. 5 *The general architecture of SAM in CBAM [32]*

In CBAM, these two modules' arrangement has been done sequentially, as it showed higher accuracy than the parallel arrangement. Furthermore, when the CAM is placed first, the model shows slightly higher accuracy than when placing the SAM in front. Therefore, the arrangement of CBAM's final architecture is as follows: First, there are the input features, and then it goes to the CAM. Thereafter, it goes to the Spatial Attention Module. Finally, we acquire the output features. Figure 6 shows the order of the modules in the CBAM architecture.

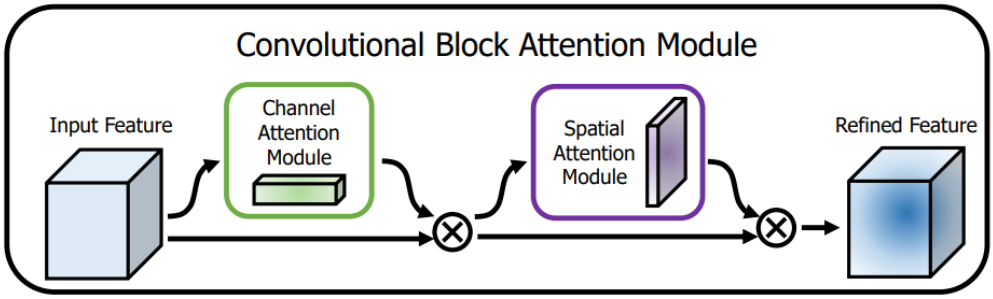

Fig. 6 *The general architecture of the CBAM block [32]*

A key aspect of our methodology involves the investigation of the optimal placement for the CBAM block within the architecture. While our initial research considered multiple potential locations for attention mechanism integration, including the backbone and detector components, our methodology ultimately focused on the detector placement. By integrating CBAM modules within the detector, the model could refine high-level features crucial for accurate object detection while minimizing the impact on model size and computational requirements.

3.5 Activation Function

Activation functions play a crucial role in neural network models, serving as non-linear transformations that enable the network to learn complex patterns. In the YOLOv7 series, the Sigmoid Linear Unit (SiLU) activation function was predominantly used, with one notable exception: the YOLOv7-tiny model, which employed LeakyReLU. This choice was made due to the trade-off between speed and accuracy in activation functions.

LeakyReLU, a variant of the Rectified Linear Unit (ReLU), is known for its computational efficiency, making it an attractive option for smaller, speed-oriented models like YOLOv7-tiny. It addresses the "dying ReLU" problem by allowing a small, non-zero gradient when the input is negative. On the other hand, SiLU, also known as the swish activation function, has gained popularity recently due to its superior performance in terms of accuracy. SiLU is smooth and non-monotonic and has been shown to work particularly well in deep learning architectures, often outperforming ReLU and its variants in various tasks.

In our proposed model, we opted for the SiLU activation function despite its slightly higher computational cost. This decision was based on the fact that our model has a smaller overall size compared to the original YOLOv7-tiny. The reduced model size should compensate for the additional computational overhead of SiLU, potentially resulting in a model that is both faster and more accurate than its predecessor. By leveraging the benefits of SiLU's nonmonotonicity and smoothness, we aim to enhance the model's ability to learn complex features while maintaining efficient inference times.

3.6 GCB-YOLO7

The GCB-YOLO architecture incorporates three modifications to YOLOv7-tiny, each designed based on established architectural principles. We replaced all ELAN blocks with ELAN-GM blocks because Ghost Modules reduce computational overhead by generating feature maps through cheap linear operations rather than expensive convolutions. Ghost Modules maintain feature diversity by splitting channels and applying different transformations, preserving representational capacity while reducing parameter count.

Additionally, CB blocks in the detector were replaced with CBAM attention modules because attention mechanisms enable selective feature refinement, which is more beneficial in detection layers where precise localization and classification decisions occur. CBAM's dual-branch design addresses both channel and spatial attention, allowing the network to learn both what and where to focus, addressing the detector's need for semantic understanding and spatial precision.

We also adopted SiLU activation to replace LeakyReLU because SiLU exhibits self-gating properties that provide smoother gradient flow and better optimization dynamics. Unlike LeakyReLU's linear negative region, SiLU's smooth, non-monotonic curve enables more nuanced feature learning. These modifications work complementarily, where Ghost Modules reduce computational burden in feature extraction, CBAM enhances

feature selectivity for detection tasks, and SiLU provides improved optimization characteristics for the combined architecture, as illustrated in Fig. 7.

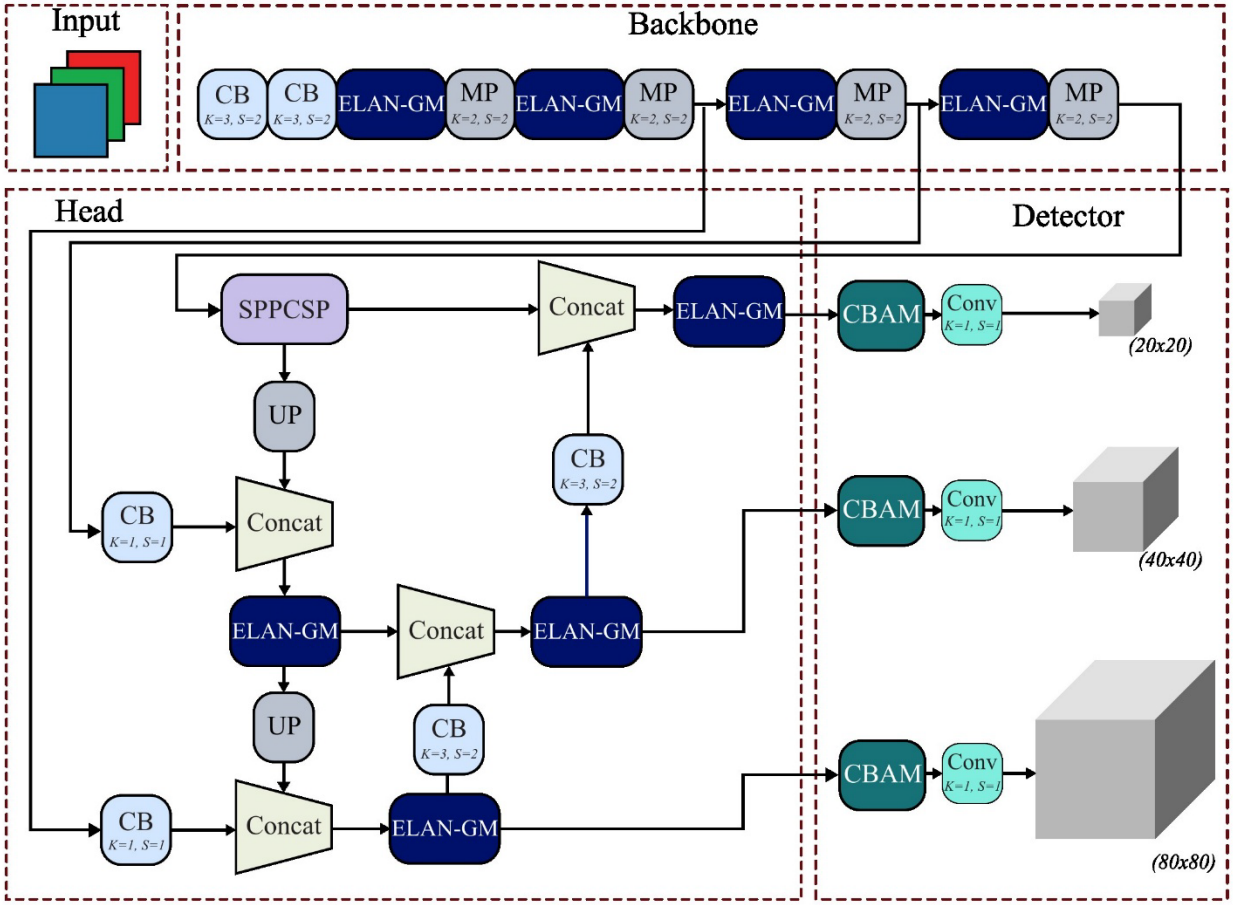

Fig. 7 *The overall architecture of GCB-YOLOv7*

3.7 Experimental Setting

3.7.1 Dataset

To verify the effectiveness of the targeted models, we used the Wider Face dataset. The Wider Face dataset is a comprehensive and challenging benchmark for face detection algorithms. Developed by researchers at the National University of Singapore, it addresses the limitations of existing face detection datasets by providing a diverse and large-scale collection of face images captured in various real-world scenarios. It consists of 32,203 images with 393,703 labeled faces, making it one of the largest publicly available face detection datasets. It was partitioned into three distinct subsets: training, validation, and testing sets, comprising 40%, 10%, and 50% of the total data, respectively. The images were meticulously selected from multiple sources, including the web, movies, and real-world scenarios, to capture a wide range of variations. Unlike prior datasets focused on individual face images, Wider Face organizes images based on 60 event categories. This categorization reflects the context in which faces appear (e.g., sporting events, parties, and street scenes). These categories are further divided into three difficulty subsets based on face detection performance: Hard (categories 1-20) includes challenging scenarios like traffic scenes and crowded festivals where faces are typically small, occluded, or in extreme poses; Medium (categories 21-40) encompasses moderately difficult contexts such as sports events and school settings; and Easy (categories 41-60) contains more favorable conditions like portrait-style scenarios, dance performances, and spa settings where faces are generally larger, well-lit, and unoccluded. This approach helps to evaluate model performance on individual faces and assess their ability to detect faces within specific environments, offering a more comprehensive evaluation [33].

3.7.2 Training Setup

We used the YOLOv7 codebase as the starting point for all the modifications described earlier. We implemented these changes using the Google Colab cloud-based editor. The hardware specification of Google Colab is shown in Table 2.

Table 2 *Google Colab specifications*

Component	Specification
CPU	Intel(R) Xeon(R) CPU @ 2.20GHz
GPU	NVIDIA Tesla T4
RAM	13 GB
Operating System	Ubuntu 22.04
Python Version	3.10.12
Pytorch Version	2.3.0
CUDA Version	12.2

3.7.3 Implementation Setup

To assess the real-world applicability of the lightweight model, it was deployed on a laptop equipped with a Core i5 CPU. While the Core i5 processor may be capable of running larger CNNs, its architecture is not specifically optimized for deep learning tasks. Consequently, even if such a network could be executed, the processing speed wouldn't be sufficient for real-time applications. The full specifications of the laptop are shown in Table 3

Table 3 *Implementation hardware specifications*

Component	Specification
Laptop model	DELL G3 3579
CPU	Intel(R) Core(TM) i5-8300H @ 2.30GHz,
RAM	12 GB
Operating System	Windows 11
Python Version	3.12.3
Pytorch Version	2.3.0

3.7.4 Evaluation Metrics

We evaluated our models using several metrics: mean Average Precision (mAP), F1 score, Giga Floating Point Operations Per Second (GFLOPs), number of parameters, and Frames Per Second (FPS).

Precision, Recall, mAP, and F1 score assess the models' detection accuracy. The F1 score provides a balanced measure of precision and recall. We used mAP50 (mean Average Precision at 50% IoU threshold) and mAP50-95 (mean Average Precision over IoU thresholds from 50% to 95%) to provide a comprehensive view of model performance across different levels of detection precision.

GFLOPs and the number of parameters measure the computational complexity and size of the models, respectively. These metrics are crucial for assessing the models' suitability for deployment in resource-constrained environments. FPS (Frames Per Second) indicates the models' real-time processing capability, which is essential for many practical applications of face detection.

3.8 Multi-Criteria Decision Analysis (MCDA)

Our analysis requires selecting the optimal model based on multiple evaluation metrics. These metrics vary, with some ensuring model efficiency and others demonstrating model accuracy. We need a mathematical approach to assist in selecting the model that offers the best trade-off between model size and accuracy. MCDA provides a practical framework in this regard. MCDA offers a structured approach to decision-making based on multiple criteria [34]. In this instance, we utilize the evaluation metrics as the criteria for performing the analysis. We perform the analysis using these steps:

1. Identify the criteria, which are the previously discussed evaluation metrics.
2. Assign weights to the criteria based on their importance. We employ the Analytic Hierarchy Process (AHP) for this purpose.
3. Normalize the data for all metrics to a common scale (0-1), where 1 represents the best performance.

4. Calculate weighted scores by multiplying the normalized values of each criterion by their respective weights.

To ensure accurate weight assignment, we perform certain analyses. Firstly, we rank the metrics based on their importance in developing a lightweight face detector. We consider both the model's efficiency and its performance when sorting the importance of these metrics for developing a lightweight, accurate model. We utilize the following ranking:

1. Parameters and mAP50 are considered crucial measures of accuracy and efficiency for face detection tasks, directly reflecting the model's performance. These are tied because, for a lightweight model, the number of parameters is critical, as it directly impacts the model size and computational requirements. Additionally, mAP50 is the main metric for measuring model accuracy.
2. Model size is closely related to the number of parameters but can sometimes differ due to model architecture. It is deemed crucial for deployment, especially on memory-constrained devices.
3. FPS is considered vital for real-time applications. It indicates how quickly the model can process images, which is critical for many practical applications.
4. GFLOPS represents the computational complexity of the model. It is important for understanding the model's efficiency.
5. The F1 score balances between precision and recall, offering a single metric for model performance. It is ranked lower than mAP50 because mAP50 is more commonly used in face detection tasks.
6. mAP50:95 provides a more comprehensive view of the model's performance across different IoU thresholds. However, it is ranked lower because mAP50 is often sufficient for many applications and is more widely used.
7. Precision and Recall are components of the F1 score and mAP. They are ranked lower because the combined metrics (F1 and mAP) provide more comprehensive information. This ranking prioritizes a balance between model lightness (parameters, size, FPS, GFLOPS) and accuracy (mAP50, F1, mAP50:95, P, R). The top metrics directly address the goal of a lightweight and accurate model. To calculate the weight, we employ AHP, and we use the average value from the two as the weight for the MCDA process. The weights are presented in Table 4.

Table 4 *Weights for MCDA analysis*

Criteria	Average weight
mAP50	0.2663
Parameters	0.2663
Size	0.1511
FPS	0.1077
GFLOPS	0.0768
mAP50:95	0.0543
P	0.0375
R	0.0249

In MCDA, data normalization is important because it guarantees that criteria with different units and scales can be compared on a common scale. We utilize Min-Max normalization as it is straightforward and commonly used. We can classify the metrics into two types: those where higher is better and those where lower is better. In Min-Max normalization, two formulas can be used based on the metric type. Formula (9) is used when the metric is of the first type:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{9}$$

When the metric is of the second type, we use the formula (10):

$$x' = \frac{\max(x) - x}{\max(x) - \min(x)} \tag{10}$$

Where x' is the normalized value, x is the original value, max(x) is the highest value of x, and min(x) is the lowest value of x. The final step in performing MCDA is to multiply the weights of each category with the

normalized values and then sum the values for each model. The model that scores higher in MCDA shows the best results in the trade-off between model size and accuracy.

4. Results and Discussion

4.1 Lightweight Convolution Analysis

A comprehensive series of experiments validated the effectiveness of our proposed modifications to YOLOv7-tiny. Our inspection of the model's convolution layers revealed optimal strategies for reducing weight without compromising accuracy. By leveraging the GM's ability to generate feature maps through cheaper operations on existing features, we maintained representational power while decreasing computational demands. This strategic replacement proved particularly effective in layers with large input and output channel dimensions, especially those with 3x3 kernels and layers following concatenation operations. Within the ELAN block, we identified two 3x3 convolution layers and the 1x1 convolution layer after concatenation as major contributors to the parameter count. We tested both versions of Ghost Modules, GM and GMv2, to assess their parameter reduction capabilities. Table 5 demonstrates GM's effectiveness in reducing parameters across layers with varying Cin, Cout, and kernel sizes. In contrast, GMv2 showed a higher parameter count than the original CB modules due to its incorporation of the DFC attention module. Based on these results, we selected GM for further testing. Notably, the extent of parameter reduction achieved by substituting GM for standard convolutions directly correlated with the layer's initial parameter count. This observation suggests that layers with higher parameter densities benefit most from this optimization technique.

Table 5 *Parameter difference between CB and GM blocks in ELAN blocks of the backbone*

Layer	Cin	Cout	K	CB	GM	GMv2
4	32	32	3	9280	5072	14544
7	128	64	1	8320	5024	13728
11	64	64	3	36992	19360	56736
14	256	128	1	33024	18240	52032
18	128	128	3	147712	75584	224064
21	512	256	1	131584	69248	202368
25	256	256	3	590336	298624	890496
28	1024	512	1	525312	269568	797952

We tested the model's accuracy in two scenarios: replacing all convolution modules with Ghost Modules (Full-GM) and replacing only the ELAN block layers with a large number of parameters (ELAN-GM). The Full-GM approach significantly reduced the number of parameters by 48.21%, from 6.015 to 3.115 million. In contrast, the ELAN-GM approach reduced the model size by only 23.72%, from 6.015 to 4.588 million parameters. However, using GM exclusively severely impacted accuracy, causing it to drop from 68.3% to 59.2%. Regarding model speed, Full-GM demonstrated the fastest performance at 5.136 FPS, closely followed by ELAN-GM at 5.044 FPS. The original YOLOv7-tiny lagged at 4.265 FPS.

MCDA results further highlighted the effectiveness of our convolution modifications. The original YOLOv7-tiny achieved an MCDA score of only 0.3323, while Full-GM scored 0.6139. The best result came from the ELAN-GM approach, with a score of 0.6463. Table 6 presents the complete results of this analysis.

Table 6 *Comparison results between original YOLOv7-tiny and convolution modification*

Model	Param (M)	GFLOPS	Speed (FPS)	Size (MB)	mAP50	mAP50:95	F1	MCDA
YOLOv7-tiny (base model)	6.015M	13.2	4.265	12.3	68.3%	35.4%	71.44%	0.3439
Full GM	3.115M	7.2	5.136	6.6	59.2%	29.3%	64.48%	0.6019
ELAN-GM	4.588M	10.0	5.044	9.5	65.7%	33.4%	69.38%	0.6453

From these results, we can conclude that the best trade-off between model size and accuracy is achieved by modifying the ELAN block, where we replace the original CB blocks that generate the largest number of

parameters with GM blocks. Both Full-GM and ELAN-GM showed faster FPS results than the original model, which means this modification moved the research closer to the desired objectives.

4.2 Attention Mechanism

Initial experiments placed attention mechanisms in the backbone, retaining the original CB module in the detector. Contrary to expectations, this modification resulted in reduced accuracy and increased model size. As shown in Table 7, the ELAN-GM model, without additional attention mechanisms, consistently outperformed variants with attention layers in the backbone. SE attention achieved the best result among these variants, with an mAP50 of 64.7% and 4.590 million parameters, while GAM attention showed the poorest performance.

Table 7 *Results of attention layer integration into the backbone*

Model	Param (M)	GFLOPS	Speed (FPS)	Size (MB)	mAP50	mAP50:95	F1
SE	4.590M	10.0	4.756	9.5	64.7%	32.7	68.59%
ECA	4588M	10.0	4.593	9.5	62.5%	32.3%	68.03%
CBAM	4.590M	10.0	4.429	9.5	64.2%	32.6%	68.25%
GAM	4.697M	11.4	4.170	9.7	57.3%	28.1%	66.16%

In subsequent investigations, we targeted the CB blocks in the detector, identified as significant contributors to the model's high parameter count. Replacing these with lightweight alternatives, including GM and various attention mechanisms, yielded substantial reductions in parameter count. The ECA mechanism demonstrated a unique advantage, maintaining a constant parameter count regardless of channel numbers due to its 1D convolution approach. The results of the number of parameter count for the detector when using different convolutions and attention mechanisms are in Table 8. Comparative analysis of full models revealed similar parameter numbers for SE, ECA, and CBAM modules when used as detectors, with ECA showing a slight advantage. GAM and GM modules exhibited significant differences in model size. GAM achieved the highest accuracy scores across most metrics, while GM, despite having the largest model size, showed the lowest accuracy.

Table 8 *Number of parameters in each detector layer*

Module	Detector 1	Detector 2	Detector 3	Total parameter
CB	73984	295424	1180672	1550080
GM	38720	151168	597248	787136
SE	580	2184	8464	11228
ECA	3	3	3	9
CBAM	679	2283	8563	11525
GAM	27456	109184	435456	572096

Implementing attention mechanisms in the detector improved both model size and accuracy metrics, contrasting with the results observed in the backbone. The ECA detector emerged as the most efficient in terms of speed, closely followed by CBAM. The GAM detector, despite not being the largest model, exhibited the slowest performance at 4.220 FPS, highlighting the complex relationship between model size, architectural design, and computational efficiency in deep learning systems.

We employed MCDA to determine the optimal balance between model size and accuracy. The MCDA evaluation ranked the attention mechanisms as follows: CBAM achieved the highest score 0.8266, outperforming ECA 0.8098, SE 0.7637, GAM 0.5289 and the lowest score was achieved when using GM 0.0493 in the detector instead of the attention mechanism. This analysis confirmed CBAM as offering the optimal trade-off between model size and accuracy for the face detection task. Based on these results, we selected the model incorporating the CBAM attention mechanism in the detector as offering the optimal trade-off between model size and accuracy for the face detection task. Table 9 displays the full results of using attention mechanisms in the detector.

Int. Journal of Integrated Engineering Vol. 18 No. 3 (2026) p. 389-409

Table 9 *Full results of the detector test*

Model	Param (M)	GFLOPS	Speed (FPS)	Size (MB)	mAP50	mAP50:95	F1	MCDA
GM	3.825M	8.6	4.784	8.0	64.9%	32.8%	68.66%	0.0493
CBAM	3.041M	7.1	5.311	6.4	65.9%	33.5%	69.59%	0.8266
ECA	3.030M	7.1	5.451	6.4	65.8%	33.1%	69.37%	0.8098
SE	3.041M	7.1	5.097	6.4	65.8%	33.4%	69.28%	0.7637
GAM	3.602M	8.2	4.220	7.5	66.6%	33.8%	70.04	0.5289

4.3 Activation Function

The selection of an appropriate activation function in neural networks is a critical factor that significantly influences model performance. This section presents a thorough examination of various activation functions and their impact on the model's effectiveness. We subjected a range of activation functions to rigorous testing, including ReLU (Rectified Linear Unit), LeakyReLU, SiLU (also known as Swish), Hardswish, and Mish. The systematic evaluation of these activation functions across the model architecture aims to identify the most effective option for enhancing the face detection capabilities of the modified YOLOv7-tiny model. While activation functions can influence accuracy without altering the model size, they may impact processing speed. By comparing performance metrics of models employing different activation functions, we can determine which function provides the optimal balance between computational efficiency and detection accuracy. This assessment is crucial for refining the model's performance in real-world face detection scenarios.

As shown in Table 10 The accuracy metrics revealed distinct differences among the activation functions. The highest mAP50 score of 67.9% was achieved by the model using the Mish activation function. This was followed closely by SiLU with 67.7%, while the original LeakyReLU function scored 65.9%, which was the lowest accuracy. These results suggest that more advanced activation functions like Mish and SiLU offer improved feature learning capabilities compared to the original LeakyReLU. The superior performance of Mish, in particular, maybe attributed to its smooth, nonmonotonic nature, which could allow for more nuanced feature representations in the face detection task. The minimal difference in accuracy between Mish and SiLU (0.2%) indicates that both functions are highly effective for face detection applications.

Table 10 *Activation function comparison results*

Model	Param (M)	GFLOPS	Speed (FPS)	Size (MB)	mAP50	mAP50:95	F1	MCDA
ReLU	3.041	7.1	5.863	6.4	66.2%	33.5%	69.66%	0.1532
LeakyReLU	3.041	7.1	5.318	6.4	65.9%	33.5%	69.59%	0.0973
SiLU	3.041	7.1	5.130	6.4	67.7%	34.8%	70.96%	0.4421
Hardswish	3.041	7.1	4.341	6.4	67.3%	34.5%	70.74%	0.3276
Mish	3.041	7.1	2.718	6.4	67.9%	34.9%	71.09%	0.3871

The consistent model size across different activation functions highlights that performance improvements can be achieved without increasing the model's complexity or memory requirements. This is particularly advantageous for deployment in resource-constrained environments where model size is a critical consideration.

The impact of activation function choice on inference speed was observed to be more pronounced than its effect on model size. The analysis revealed significant variations in computational efficiency. The fastest inference times were recorded for ReLU, followed by LeakyReLU, due to their simplicity and computational efficiency. while Mish was found to be computationally more expensive, resulting in slower inference speeds despite its high accuracy. To determine the optimal activation function considering both accuracy and computational efficiency, we conducted MCDA analysis. In this analysis, SiLU emerged as the best performer with a score of 0.4369, followed closely by Mish at 0.3746. The original LeakyReLU function scored 0.1044, which was the lowest MCDA score. Based on this comprehensive analysis, we selected the SiLU activation function for the proposed model. This choice reflects a strategic balance between enhancing detection accuracy and maintaining computational efficiency, which is crucial for real-world face detection applications.

4.4 Modifications Summary

To demonstrate the effectiveness of the modifications applied to the model, we summarized the final results of all selected modifications in an ablation experiment style. The ablation models consisted of (i) the original YOLOv7-

tiny, (ii) YOLOv7-tiny with GM blocks in the ELAN module (ELAN-GM), (iii) the replacement of CB with CBAM attention in the model's detector, and (iv) the use of the SiLU activation function. The ablation results are illustrated in Table 11.

Table 11 *Ablation experiment results for GCB-YOLO-tiny*

Model	Param (M)	Speed (FPS)	mAP50	mAP50:95
YOLOv7-tiny (base model)	6.015	4.265	68.3%	35.4%
+GM block (ELAN-GM)	4.588	5.044	65.7%	33.4%
+ CBAM attention as detector	3.041	5.318	65.9%	33.5%
+ SiLU activation (GCB-YOLO)	3.041	5.130	67.7%	34.8%

The original YOLOv7-tiny exhibited high accuracy with a large model size compared to the other models. In the first modification, we replaced CB blocks with a high number of parameters with GM blocks to reduce the parameter count, while the detector of the model still utilized CB blocks at this stage. This modification resulted in a 23.72% reduction in model size from 6.015M to 4.588M, with accuracy decreasing from 68.3% mAP50 to 65.7%. While this represents an accuracy reduction, our MCDA analysis in Table 6 confirmed this configuration achieved the highest optimization score 0.6463, making it the optimal foundation for subsequent enhancements in our multi-stage approach. The model's speed was also affected, increasing from 4.265 FPS to 5.044 FPS. Further replacement of the detector's CB blocks led to an additional reduction in model size and increased accuracy. The number of parameters was decreased to 3.014M, and the FPS was increased to 5.318. The mAP slightly increased to 65.9% and 33.5% at 0.5 and 0.95 thresholds, respectively. The activation function used in YOLOv7-tiny, LeakyReLU, was initially chosen for its speed. Replacing LeakyReLU with SiLU resulted in a further increase in accuracy to 67.7%. However, we observed a slight decrease in speed from 5.318 FPS to 5.130 FPS.

4.5 Comparison With YOLO-Based Lightweight Models

4.5.1 Result Using Full Wider Face Dataset

To ensure the effectiveness of GCB-YOLO, we compared it against other lightweight YOLO models. The models under comparison include YOLOv5n, YOLOv6n, YOLOv7-tiny, and YOLOv8n. We conducted the comparison with the full Wider Face dataset and Wider Face sub-datasets (easy, medium, and hard). The first experiment included the full Wider Face dataset. The full results are documented in Table 12.

Table 12 *Comparison with state-of-the-art lightweight object detection models*

Model	Param (M)	GFLOPS	Speed (FPS)	Size	mAP50	mAP50:95	F1	MCDA
YOLOv5n	2.655	7.8	8.567	5.3	63.0%	33.8%	66.23%	0.5756
YOLOv6n	4.238	11.9	9.4	8.7	63.1%	34.2%	66.61%	0.3734
YOLOv7-tiny	6.015	13.2	4.265	12.3	68.3%	35.4%	71.44%	0.3961
YOLOv8n	3.011	8.2	5.488	6.2	64.5%	35.2%	67.93%	0.5998
GCB-YOLO	3.041	7.1	5.130	6.4	67.7%	34.8%	70.96%	0.8050

Regarding accuracy metrics, we observed similar performance across all models for mAP50:95. However, notable differences were recorded in the mAP50 metric. The highest mAP50 was achieved by YOLOv7-tiny at 68.3%, followed closely by the proposed GCB-YOLO at 67.7%. YOLOv8n, YOLOv6n, and YOLOv5n showed lower mAP50 scores of 64.5%, 63.1%, and 63.0%, respectively. The F1 score followed a similar trend. YOLOv7-tiny led with 71.44%, closely followed by GCB-YOLO at 70.96%. YOLOv8n, YOLOv6n, and YOLOv5n demonstrated lower F1 scores of 67.93%, 66.61%, and 66.23%, respectively.

We observed significant variations in the number of parameters across the models. YOLOv5n had the smallest parameter count at 2.655 million, followed by YOLOv8n with 3.011 million and the proposed GCB-YOLO with 3.041 million. YOLOv6n and YOLOv7-tiny had considerably larger parameter counts at 4.238 million and 6.015 million, respectively. Model size, measured in megabytes, showed a similar trend. YOLOv5n was the smallest at 5.3 MB, followed by YOLOv8n at 6.2 MB and GM-YOLOv7 at 6.4 MB. YOLOv6n and YOLOv7-tiny were substantially larger at 8.7 MB and 12.3 MB, respectively.

Int. Journal of Integrated Engineering Vol. 18 No. 3 (2026) p. 389-409

In terms of computational efficiency measured in GFLOPS, GCB-YOLO demonstrated the best performance at 7.1 GFLOPS, followed by YOLOv5n at 7.8 GFLOPS and YOLOv8n at 8.2 GFLOPS. YOLOv6n and YOLOv7-tiny showed higher GFLOPS at 11.9 and 13.2, respectively. GCB-YOLO's exceptional efficiency in terms of GFLOPS is due to its utilization of GM blocks and the CBAM attention mechanism. GM generates more features from cheaper operations, effectively reducing the computational cost while maintaining model expressiveness. This approach allows GCB-YOLO to achieve a lower GFLOPS count despite having a slightly higher number of parameters. Additionally, incorporating the CBAM attention mechanism can lead to more efficient feature utilization, potentially increasing the parameter count and reducing the overall computational burden during inference.

Remarkably, YOLOv6n demonstrated the fastest performance at 9.4 FPS, despite having a larger parameter count compared to other models. YOLOv5n followed closely with 8.567 FPS. The remaining models exhibited lower frame rates: YOLOv8n at 5.488 FPS, GCB-YOLO at 5.130 FPS, and YOLOv7-tiny at 4.265 FPS. It's crucial to note that we obtained the FPS measurements for YOLOv5n, YOLOv6n, and YOLOv8n using the Ultralytics Python library, while we evaluated YOLOv7-tiny and GCB-YOLO using the YOLOv7 GitHub repository implementation. This disparity in testing methodologies may significantly contribute to the substantial differences in FPS results. Furthermore, YOLOv6n's use of the ReLU activation function, which is computationally less complex than the SiLU activation employed by the other models, may also contribute to its performance.

To determine the best-performing model overall, considering all metrics, we conducted an MCDA analysis. In this evaluation, GCB YOLO emerged as the top performer with a score of 0.7990, significantly outperforming the other models. YOLOv8n and YOLOv5n followed with scores of 0.6009 and 0.5860, respectively. The base model, YOLOv7-tiny, scored 0.3840, while YOLOv6n showed the lowest overall performance with a score of 0.3838.

Based on these results, we can conclude that the proposed GCB-YOLO model offers a superior balance between accuracy, model size, and computational efficiency. While YOLOv7-tiny maintains a slight edge in accuracy metrics, GCB-YOLO significantly reduces the model size and computational requirements while maintaining comparable accuracy. This makes GCB-YOLO a highly competitive option for lightweight face detection tasks, particularly in resource-constrained environments.

4.5.2 Result Using Wider Face Sub-Datasets

The Wider Face dataset is known for its challenging nature, divided into easy, medium, and hard subsets based on the difficulty of face detection. We also tested the models presented above in the Wider Face subsets. The model size and computation cost are not affected when testing the model with different datasets. Therefore, the next experiments focus on accuracy metrics. In the evaluation using the Easy Dataset, YOLOv8n demonstrated the best performance in mAP metrics, achieving 83.2% mAP50 and 51.7% mAP50:95. The other YOLO models exhibited comparable mAP results, with YOLOv7-tiny scoring 82.9% mAP50 and 50.2% mAP50:95, YOLOv5n attaining 82.2% mAP50 and 50.6% mAP50:95, and YOLOv6n reaching 82.0% mAP50 and 50.2% mAP50:95. In terms of F1 scores, YOLOv7-tiny outperformed the others with 82.50%, closely followed by YOLOv8n at 82.33%. YOLOv5n and YOLOv6n achieved F1 scores of 81.89% and 81.55%, respectively. The proposed GCB-YOLO model demonstrated competitive performance, although it fell slightly behind the other models with 81.4% mAP50, 48.2% mAP50:95, and an F1 score of 81.11%. The full results are shown in Table 13.

Table 13 *Results on the Wider Face sub-datasets (easy, medium, hard)*

Model	Easy			Medium			Hard		
	mAP50	mAP50:95	F1	mAP50	mAP50:95	F1	mAP50	mAP50:95	F1
YOLOv5n	82.2%	50.6%	81.89%	76.5%	44.7%	77.50%	54.2%	26.1%	58.51%
YOLOv6n	82.0%	50.2%	81.55%	75.3%	44.3%	76.55%	54%	26.0%	58.42%
YOLOv7-tiny	82.9%	50.2%	82.51%	76.9%	43.3%	79.09%	58.8%	26.8%	63.61%
YOLOv8n	83.2%	51.7%	82.33%	76.6%	44.7%	77.87%	55.8%	26.7%	59.38%
GCB-YOLO	81.4%	48.2%	81.11%	79.6%	44.1%	81.04%	60.5%	27.4%	65.37%

In the medium subset, GCB-YOLO demonstrated a notable performance improvement, achieving the highest mAP50 of 79.6% and F1 score of 81.04%. This superior performance suggests that GCB-YOLO is particularly adept at handling moderately difficult scenarios. YOLOv8n maintained strong performance, achieving the highest mAP50:95 of 44.7%, while YOLOv5n also performed well. YOLOv7-tiny showed competitive results with an mAP50 of 76.9% and an F1 score of 79.09%, ranking second in both metrics. YOLOv5n and YOLOv6n also performed well, with YOLOv5n achieving 76.5% mAP50 and 77.50% F1 score, while YOLOv6n reached 75.3% mAP50 and 76.55% F1 score. Despite its high mAP50 and F1 score, GCB-YOLO's mAP50:95 of 44.1% was slightly lower than some of its counterparts.

In the evaluation using the hard subset, GCB-YOLO showed outstanding performance, achieving the highest mAP50 of 60.5% and F1 score of 65.37%. YOLOv8n maintained a strong performance in the mAP50:95 metric,

scoring 26.7%, though it was surpassed by GCB-YOLO with 27.4% in this category. YOLOv7-tiny demonstrated robust performance, achieving the second-highest scores across all metrics with 58.8% mAP50, 26.8% mAP50:95, and an F1 score of 63.61%. YOLOv5n and YOLOv6n showed similar performance levels, with YOLOv5n slightly outperforming YOLOv6n across all metrics. YOLOv5n achieved 54.2% mAP50, 26.1% mAP50:95, and 58.51% F1 score, while YOLOv6n reached 54% mAP50, 26% mAP50:95, and 58.42% F1 score. These results show GCB-YOLO's strength in detecting difficult faces.

In summary, GCB-YOLO showed a clear trend of improved performance as the difficulty increased, outperforming other models in the medium and hard subsets. This suggests that its architectural modifications are particularly effective for challenging detection scenarios. These results highlight the strengths of GCB-YOLO in handling challenging face detection scenarios, particularly in the medium and hard subsets of the Wider Face dataset. Despite a slight accuracy drop compared to YOLOv7-tiny, GCB-YOLO reduces parameters by ~50% while maintaining competitive accuracy, making it more practical for deployment on devices such as SBCs and IoT systems.

4.5.3 Results on the Wider Face Test Dataset

Two representative images from the Wider Face test set were selected to demonstrate GCB-YOLO's detection capabilities across diverse challenging scenarios. The confidence threshold was set to 0.25 across all models for fair comparison.

Fig. 8 *Detection results comparison: Image from the Handshaking category of the Wider Face dataset*

Int. Journal of Integrated Engineering Vol. 18 No. 3 (2026) p. 389-409

The first image, from the "Handshaking" category, depicts a social gathering scene with multiple detection challenges, including varied face orientations, occlusions, diverse lighting conditions, and individuals at different distances. GCB-YOLO achieved the highest performance, detecting 17 faces compared to YOLOv7-Tiny and YOLOv6n (14 faces each) and YOLOv5n and YOLOv8n (12 faces each). All models successfully detected the five clear foreground faces, but GCB-YOLO demonstrated superior capability in identifying blurry and partially obscured faces in the background, which differentiates high-performing models in real-world applications.

The second image, from the "Greeting People" category, presents a casual indoor scenario with varied lighting, multiple face orientations, and partial occlusions. GCB-YOLO detected all visible faces, including those in the far background and partially visible at image edges. YOLOv7-Tiny performed comparably but missed one background face. YOLOv8n showed reduced accuracy, missing the background person, edge-visible face, and an elderly individual. YOLOv6n produced similar results to YOLOv8n with additional duplicate bounding box errors. YOLOv5n generated a false positive by incorrectly classifying a neck region as a face.

These results demonstrate GCB-YOLO's ability to maintain competitive or superior detection accuracy across challenging real-world scenarios while achieving significant parameter reduction compared to baseline models.

Fig. 9 *Detection results comparison: Image from the Greeting People category of the Wider Face dataset*

5. Conclusion

This paper presents GCB-YOLO, a lightweight face detection model that addresses the trade-off between model complexity and detection accuracy. Integration of Ghost Modules with attention mechanisms achieves parameter reduction (49.44%) while maintaining competitive performance. Our findings demonstrate that attention mechanisms perform differently when applied to backbone versus detector components, with detector-level integration proving more effective for lightweight architectures.

Achieving real-time performance with reduced computational demands makes GCB-YOLO valuable for deployment in resource-constrained environments where traditional models would be impractical.

Future work could extend the approach to multitask settings, such as landmark detection or emotion recognition, though such applications require extensive validation and task-specific optimization. Investigating the architecture's performance across different object detection domains could lead to broader applicability.

Acknowledgement

Funding: This paper is supported under the Ministry of Higher Education Malaysia for Fundamental Research Grant Scheme with Project code: FRGS/1/2024/ICT03/USM/02/1

Conflict of Interest

Authors declare that there is no conflict of interest regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *Ibrahim Al-Amuodi, Dzati Athiar Ramli;* ***data collection:*** *Ibrahim Al-Amuodi;* ***analysis and interpretation of results:*** *Ibrahim Al-Amuodi, Dzati Athiar Ramli;* ***draft manuscript preparation:*** *Ibrahim Al-Amuodi, Dzati Athiar Ramli. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] Adjabi, I., Ouahabi, A., Benzaoui, A., & Taleb-Ahmed, A. (2020). Past, present, and future of face recognition: A review. *Electronics*, 9(8), Article 1188. https://doi.org/10.3390/electronics9081188

[2] Mellouk, W., & Handouzi, W. (2020). Facial emotion recognition using deep learning: Review and insights. *Procedia Computer Science*, *175*, 689–694. https://doi.org/10.1016/j.procs.2020.07.101

[3] Xu, X., Liang, W., Zhao, J., & Gao, H. (2021). Tiny FCOS: A lightweight anchor-free object detection algorithm for mobile scenarios. *Mobile Networks and Applications*, *26*(6), 2219–2229. https://doi.org/10.1007/s11036-021-01845-y

[4] Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. https://doi.org/10.1109/cvpr.2001.990517

[5] Viola, P., & Jones, M. J. (2004). Robust real-time face detection. *International Journal of Computer Vision*, *57*(2), 137–154. https://doi.org/10.1023/B:VISI.0000013087.49260.fb

[6] Jadhav, A., Lone, S., Matey, S., Madamwar, T., & Jakhete, S. (2021). Survey on face detection algorithms. *International Journal of Innovative Science and Research Technology*, *6*(2), 291–297.

[7] Gao, J., & Yang, T. (2022). Face detection algorithm based on improved TinyYOLOv3 and attention mechanism. *Computer Communications*, *181*, 329–337. https://doi.org/10.1016/j.comcom.2021.10.023

[8] Chen, J., & Ran, X. (2019). Deep learning with edge computing: A review. *Proceedings of the IEEE*, *107*(8), 1655–1674. https://doi.org/10.1109/JPROC.2019.2921977

[9] Zhu, Y., Cai, H., Zhang, S., Wang, C., & Xiong, Y. (2020). TinaFace: Strong but simple baseline for face detection. *arXiv preprint*. https://arxiv.org/abs/2011.13183

[10] Sirisha, U., Praveen, S. P., Srinivasu, P. N., Barsocchi, P., & Bhoi, A. K. (2023). Statistical analysis of design aspects of various YOLO-based deep learning models for object detection. *International Journal of Computational Intelligence Systems*, *16*(1), Article 16. https://doi.org/10.1007/s44196-023-00302-w

[11] Huang, J., Shang, Y., & Chen, H. (2019). Improved Viola-Jones face detection algorithm based on HoloLens. *EURASIP Journal on Image and Video Processing*, *2019*(1), Article 41. https://doi.org/10.1186/s13640-019-0435-6

[12] Zhang, J., Zhang, X. D., & Ha, S. W. (2008). A novel approach using PCA and SVM for face detection. In *Proceedings - 4th International Conference on Natural Computation, ICNC 2008* (pp. 375–378). https://doi.org/10.1109/ICNC.2008.257

[13] Mahajan, J. A., & Paithane, A. N. (2017). Face detection on distorted images by using quality HOG features. In *Proceedings of the International Conference on Inventive Communication and Computational Technologies, ICICCT 2017* (pp. 321–326). https://doi.org/10.1109/ICICCT.2017.7975236

[14] Yang, S., Luo, P., Loy, C.-C., & Tang, X. (2015). From facial parts responses to face detection: A deep learning approach. In *2015 IEEE International Conference on Computer Vision (ICCV)* (pp. 3676–3684). https://doi.org/10.1109/ICCV.2015.419

[15] Yang, S., Luo, P., Loy, C. C., & Tang, X. (2018). Faceness-Net: Face detection through deep facial part responses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *40*(8), 1845–1859. https://doi.org/10.1109/TPAMI.2017.2738644

[16] Zhang, S., Zhu, X., Lei, Z., Shi, H., Wang, X., & Li, S. Z. (2017). S3FD: Single shot scale-invariant face detector. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 192–201). https://doi.org/10.1109/ICCV.2017.30

[17] Tang, X., Du, D. K., He, Z., & Liu, J. (2018). PyramidBox: A context-assisted single shot face detector. In *Computer Vision – ECCV 2018* (pp. 812–828). Springer. https://doi.org/10.1007/978-3-030-01240-3_49

[18] Deng, J., Guo, J., Ververas, E., Kotsia, I., & Zafeiriou, S. (2020). RetinaFace: Single-shot multi-level face localisation in the wild. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 5203–5212). https://doi.org/10.1109/CVPR42600.2020.00525

[19] Chen, W., Huang, H., Peng, S., Zhou, C., & Zhang, C. (2021). YOLO-face: A real-time face detector. *The Visual Computer*, *37*(4), 805–813. https://doi.org/10.1007/s00371-020-01831-7

[20] Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint face detection and alignment using multi-task cascaded convolutional networks. *IEEE Signal Processing Letters*, *23*(10), 1499–1503. https://doi.org/10.1109/LSP.2016.2603342

[21] Zhang, S., Zhu, X., Lei, Z., Shi, H., Wang, X., & Li, S. Z. (2017). FaceBoxes: A CPU real-time face detector with high accuracy. In *IEEE International Joint Conference on Biometrics, IJCB 2017* (pp. 1–9). https://doi.org/10.1109/BTAS.2017.8272675

[22] Zhang, S., Wang, X., Lei, Z., & Li, S. Z. (2019). Faceboxes: A CPU real-time and accurate unconstrained face detector. *Neurocomputing*, *364*, 297–309. https://doi.org/10.1016/j.neucom.2019.07.064

[23] Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K., & Grundmann, M. (2019). BlazeFace: Sub-millisecond neural face detection on mobile GPUs. *arXiv preprint*. https://arxiv.org/abs/1907.05047

[24] Qi, D., Tan, W., Yao, Q., & Liu, J. (2023). YOLO5Face: Why reinventing a face detector. In *Computer Vision – ACCV 2022* (pp. 228–244). Springer. https://doi.org/10.1007/978-3-031-25072-9_15

[25] Wang, G., Li, J., Wu, Z., Xu, J., Shen, J., & Yang, W. (2023). EfficientFace: An efficient deep network with feature enhancement for accurate face detection. *Multimedia Systems*, *29*(5), 2595–2610. https://doi.org/10.1007/s00530-023-01134-6

[26] Yu, Z., Huang, H., Chen, W., Su, Y., Liu, Y., & Wang, X. (2024). YOLO-FaceV2: A scale and occlusion aware face detector. *Pattern Recognition*, *155*, Article 110714. https://doi.org/10.1016/j.patcog.2024.110714

[27] Kim, D., Jung, J., & Kim, J. (2025). FeatherFace: Robust and lightweight face detection via optimal feature integration. *Electronics*, *14*(3), Article 517. https://doi.org/10.3390/electronics14030517

[28] Terven, J., Córdova-Esparza, D. M., & Romero-González, J. A. (2023). A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS. *Machine Learning and Knowledge Extraction*, *5*(4), 1680–1716. https://doi.org/10.3390/make5040083

[29] Al Amoudi, I., & Ramli, D. A. (2024). YOLOv7-Tiny and YOLOv8n evaluation for face detection. In *12th International Conference on Robotics, Vision, Signal Processing and Power Applications. RoViSP 2023* (pp. 477–483). Springer. https://doi.org/10.1007/978-981-99-9005-4_60

[30] Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., & Xu, C. (2020). GhostNet: More features from cheap operations. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 1580–1589). https://doi.org/10.1109/CVPR42600.2020.00165

[31] Tang, Y., Han, K., Guo, J., Xu, C., Xu, C., & Wang, Y. (2022). GhostNetV2: Enhance cheap operation with long-range attention. *Advances in Neural Information Processing Systems*, *35*, 9969–9982.

[32] Woo, S., Park, J., Lee, J. Y., & Kweon, I. S. (2018). CBAM: Convolutional block attention module. In *Computer Vision – ECCV 2018* (pp. 1–17). Springer. https://doi.org/10.1007/978-3-030-01234-2_1

[33] Yang, S., Luo, P., Loy, C. C., & Tang, X. (2016). WIDER FACE: A face detection benchmark. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 5525–5533). https://doi.org/10.1109/CVPR.2016.596

[34] Guitouni, A., & Martel, J. M. (1998). Tentative guidelines to help choosing an appropriate MCDA method. *European Journal of Operational Research*, *109*(2), 501–521. https://doi.org/10.1016/S0377-2217(98)00073-3