



YOLOv8 to YOLO11 Performance Benchmark and Comprehensive Architectural Comparative Review

Priyanto Hidayatullah^{1*}, Nurjannah Syakrani², Muhammad Rizqi Sholahuddin³, Trisna Gelar⁴, Refdinal Tubagus⁵

^{1,2,3,4}Computer Engineering and Informatics Department, Politeknik Negeri Bandung, Bandung, Indonesia

⁵Stunning Vision AI, Bandung, Indonesia

*priyanto@polban.ac.id

Abstract

In the domain of deep learning-driven computer vision, YOLO is revolutionary. However, not all YOLO models are accompanied by academic articles and architectural diagrams. It complicates the comprehension of the model's operation. Moreover, the existing review papers fail to examine each model comprehensively. This work aims to provide a thorough comparative analysis of the architectures from YOLOv8 to YOLO11, allowing readers to swiftly understand the operational mechanisms and differences among the models. We analyzed the architecture of each YOLO version by reviewing relevant scholarly articles, official documentation, and examining the source code. In particular, we discovered that YOLOv8 through YOLO11 differ in novelty while sharing similarities in the anchor-free and Non-Maximum Suppression (NMS) aspects, except YOLOv10 (NMS-free). Each also has drawbacks, such as differing levels of complexity in the way features are connected (v8), architectural structure and training (v9), training methods or dual assignments (v10), inference, and code implementation (v11). While each version improves architecture, some blocks remain unchanged. This study helps readers understand different YOLO version architectures and inspires how to improve their performance. It also provides readers with a comprehensive architecture diagram and detailed descriptions of each block, serving as a reference for both academic and practical applications. In terms of performance, a benchmark using the Roboflow 100 dataset reveals that YOLOv9 achieves superior accuracy; however, it is eight times slower owing to its NMS mechanism. YOLOv10 is the fastest but least accurate, whereas YOLOv8 and YOLO11 provide a balanced compromise between speed and accuracy.

Keywords: artificial intelligence; computer vision; deep learning; YOLO architecture

How to Cite: P. Hidayatullah, N. Syakrani, M. R. Sholahuddin, T. Gelar, and R. Tubagus, "YOLOv8 to YOLO11 Performance Benchmark and Comprehensive Architectural Comparative Review", *J. RESTI (Rekayasa Sist. Teknol. Inf.)*, vol. 10, no. 2, pp. 341 - 354, Apr. 2026.
Permalink/DOI: <https://doi.org/10.29207/resti.v10i2.6598>

Received: April 28, 2025

Accepted: March 17, 2026

Available Online: April 21, 2026

*This is an open-access article under the [CC BY 4.0 License](#)
Published by [Ikatan Ahli Informatika Indonesia](#)*

1. Introduction

The You Only Look Once (YOLO) framework is fundamental to real-time object detection, significantly influencing the domain of computer vision. The series has had notable and ongoing innovation since its introduction, with each iteration presenting substantial architectural improvements for better performance and efficiency. Multiple reviews have documented this progression, analyzing the historical evolution of the initial versions and the emergence of YOLO as a preferred solution for many applications [1-5]. These publications jointly validate YOLO's status as a premier system that harmonizes speed and accuracy, crucial for implementation in various real-world contexts.

Despite this extensive body of research, a significant obstacle for scholars and practitioners is the absence of comprehensive scholarly articles and detailed architectural schematics for the most recent editions. Although several review papers are available, they sometimes lack depth in architectural details and offer limited insights, necessitating reliance on secondary sources and empirical investigation within the community. Numerous studies have investigated the utilization of YOLO across various domains, such as weed identification in agriculture [6], animal phenotyping [7], medical image analysis [8-10], industrial defect inspection [3], [11], urban environments for sophisticated driver-assistance systems [12], and fall detection [13]. Previous publications have included comprehensive summaries

of YOLO's development, emphasizing its applications [14-17] or its resilience against adversarial attacks [18]. Papers concentrating on particular models or model groups, such as YOLOv8 and its advancements [19-21], the progression to YOLOv10 [22], or an extensive benchmark up to YOLOv12 [23], generally summarize essential features instead of offering a detailed, side-by-side comparison of the fundamental architectural components that enhance performance. These problems jointly highlight the imperative for a comprehensive architectural comparison to bridge this significant knowledge gap.

YOLOv8 was introduced in 2023. It employed an anchor-free detection method that improved performance on small objects and simplified the model design. The unifying architecture for various computer vision tasks enhances its application across numerous domains [24]. YOLOv8 covers various tasks, such as object detection, instance segmentation, and image classification, supported by an anchor-free detection approach. Ultralytics released it for the first time in 2023. The latest YOLOv8 supports a wide range of visual AI tasks, along with additional capabilities such as object tracking, pose estimation, and oriented bounding boxes [25]. In 2024, YOLOv9 introduced the GELAN (Generalized Efficient Layer Aggregation Network) architecture and the PGI (Programmable Gradient Information) framework. They increase the accuracy of lightweight models and resolve the issue of information bottlenecks [26].

YOLOv10, which was introduced in 2024, adopting a Non-Maximum Suppression-free (NMS-free) training technique, which enables more effective deployment while lowering processing needs, represents a significant advancement. Large-kernel convolutions and spatial-channel decoupled downsampling have significantly improved efficiency and performance. For NMS-free training, the dual assignments approach continuously improved accuracy, reduced latency, and enabled real-time object detection [27]. YOLO11, the latest Ultralytics' YOLO model, was introduced in 2025, performs incredibly well in a number of computer vision tasks, such as object detection, feature extraction, instance segmentation, pose estimation, object tracking, and image classification. It also marks a significant breakthrough in real-time object detection [28]. According to [29], YOLO11 incorporates the C3k2 block, C2PSA (Convolutional Block with Parallel Spatial Attention), and SPPF (Spatial Pyramid Pooling - Fast) components, which enhance the capacity to extract features and locate objects [14].

The evolution and advancement of YOLO architectures have been the subject of numerous review studies. A thorough examination of YOLO's development, with a particular emphasis on YOLO-NAS and versions 1 through 8. They look at the speed-accuracy trade-off, give a brief historical overview, and stress the importance of application demands when choosing a YOLO version [5]. However, they just look at

YOLOv8, and they do not go into great detail about each iteration's architecture. A thorough analysis covering versions 1 through 11 [29], whereas [14] concentrates on versions 1 through 10, with a particular emphasis on the agriculture industry. Additionally, it covers versions 1 through 10 and gives a brief overview of the architectural development [1]. Others specifically examines YOLOv5, YOLOv8, and YOLOv10 [14], whereas [28] focuses only on YOLO11.

The potential of YOLOv1 through YOLOv10 variations for enhancing photovoltaic (PV) defect detection has been thoroughly reviewed. This analysis emphasizes the impact of evolving YOLO architectures on quality control in the PV industry. The effectiveness of each variant is shaped by several fundamental architectural components, including programmable gradient information, path aggregation networks, and generalized efficient layer aggregation structures. The review suggests a future research direction focused on expanding the applicability of YOLO variations to a wider range of PV defects, beyond the current focus on micro-cracks. Furthermore, the exploration of attention mechanisms within YOLO architectures is proposed as a promising avenue for significantly improving the detection of subtle and intricate defects [30].

While existing reviews provide valuable insights into the evolution of YOLO architectures, thorough analyses of the most recent iterations, specifically versions v8 through v11, is often lacking. For instance, a detailed examination of YOLO's development from its inception up to YOLOv8 is offered; however, the complexities of the newer architectures that introduce significant enhancements in network design and training are not explored [5]. Similarly, a complete architectural comparison of YOLOv10 with its earlier or later counterparts is absent, making it challenging to discern its unique contributions and improvements [14]. Although a general overview of YOLO11 is provided, an in-depth exploration of its architectural components and their impact on performance is not included [29]. The comparison summary is presented in Table 1.

Table 1. Existing review papers comparison

Reference	YOLO Version	Focus of The Paper	Architectural Depth
[5]	YOLOv1 to YOLOv8	General architecture review	High-level overview, not a detailed comparison.
[14]	YOLOv1 to YOLOv10	Applications in the agricultural domain	Low, focuses on applications.
[15]	Up to YOLO11	General framework and benchmarks	Low, focuses on applications and benchmarks.
[22]	YOLOv5, YOLOv8, YOLOv10	Real-time performance analysis	Low, focuses on performance metrics.
[23]	YOLOv3 to YOLOv12	Comprehensive benchmark analysis	Low, focuses on performance metrics.

This study seeks to overcome these limitations by offering a comprehensive and systematic comparison of the architectures of YOLOv8, YOLOv9, YOLOv10, and YOLO11. We explore the distinctive architectural innovations of each model, expanding on the underlying knowledge outlined in recent publications regarding these specific versions [6], [11], [22], [28]. Our effort is to clarify the operation of each model by analyzing the fundamental components and design philosophies, thereby enhancing the community's understanding of YOLO's evolution. This profound comprehension will offer a definitive framework for model selection and optimization, enhancing the creation of more efficient and resilient computer vision systems. This paper methodically outlines the evolution of the YOLO architecture to foster innovation and advancement in real-time object identification.

2. Methods

This study applies a structured methodology consisting of several stages (Figure 1) to ensure that the comparison from YOLOv8 to YOLO11 is comprehensive and reproducible. The research process is divided into four sequential stages: (i) systematic literature review, (ii) source code analysis, (iii) architecture diagram development, and (iv) validation and verification process.

2.1 Phase 1: Systematic Literature Review

In the initial phase, the research focused on collecting theoretical foundations and exploring relevant knowledge gaps. Literature was searched through several main sources, including IEEE Xplore, ACM Digital Library, arXiv, and official YOLO documentation. To obtain more focused results, the search used a combination of keywords with Boolean logic. For example, the query “YOLO architecture” AND “YOLO review” was used to find studies that discussed both aspects simultaneously.

The search was limited to publications since 2022, with a particular focus on research discussing architectural innovations, implementation methods, or benchmark-based evaluations. From the collected literature, the information recorded includes architectural contributions, model performance, technical constraints, and code repositories provided. The review results indicate that there are still limitations or gaps in the documentation for YOLOv8 to YOLO11, so further analysis was shifted to the code analysis phase in the next stage.

2.2 Phase 2: Source Code Analysis

The official repositories for each model were searched to obtain valid architectural details: YOLOv8 and YOLO11 from Ultralytics [31], YOLOv9 from WongKinYiu [32], and YOLOv10 from THU-MIG [33]. The analysis was conducted by examining the YAML configuration files to identify scaling parameters, such as `depth_multiple`, `width_multiple`,

and `max_channels`. Following that, several implementation files, such as `conv.py`, `block.py`, `head.py`, and `common.py`, were opened one by one to examine how the blocks are structured (C2f, C3k2, PSA, C2PSA, etc), the tensor sizes used, the order of convolutions and pooling, and the attention modules added. All differences between versions are then summarized in a comparison table for easier reading.

During the source code analysis phase, the focus is on how each block is actually implemented in the code, such as class definitions, initialization arguments, and the forward pass flow. Each function is investigated to better understand the tensor transformations that occur within it, such as dimensionality changes, repetitive convolution operations, and feature pooling processes. Furthermore, the implementations in each version are compared to identify structural changes and alterations.

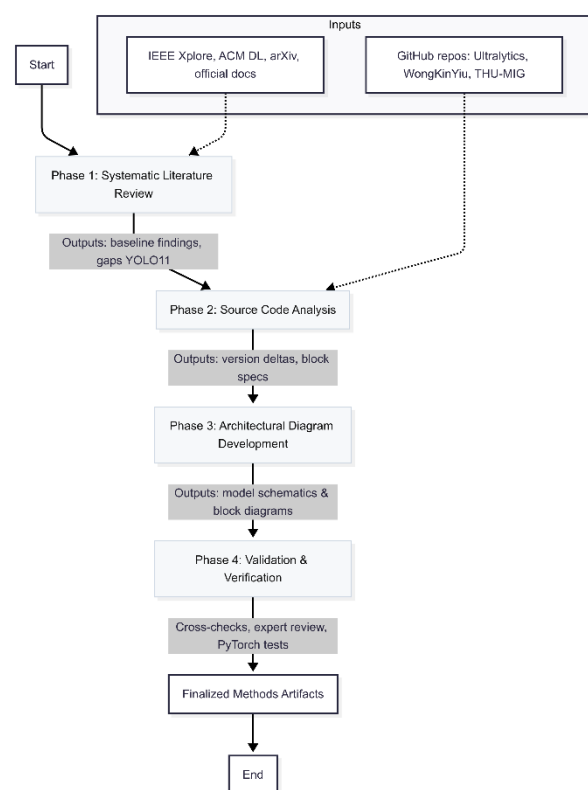


Figure 1. Research methodology workflow for YOLOv8–YOLO11 comparison.

2.3 Phase 3: Architectural Diagram Development

Phases 1 and 2 findings are used to create the overall architecture diagram (backbone–neck–head). Hierarchical layouts, color coding, and consistent symbols ensure clarity. Draw.io was used to create scalable vector diagram graphics for publishing and reuse.

The Diagram depicts the flow of data from input to output. Each block is neatly labeled and color-coded to facilitate identification. Arrows identify relationships between blocks, allowing users to readily understand the model's structure and process flow.

2.4 Phase 4: Validation and Verification

Accuracy was confirmed by verifying diagrams against academic sources, official documents, and source code. We reimplement the most fundamental component in YOLO, namely the Conv block. This block was employed across all YOLO versions analyzed in this study. We assessed all the other blocks through training and examined the output of each block to ensure the accuracy of the values depicted in our diagrams.

To verify accuracy, we ran tests on multiple model variants. Each variant was examined to determine the output produced by each block. This method ensured that the diagrammatic representations remained valid and relevant across the variants, as well as accurately reflecting the architecture's overall characteristics.

3. Results and Discussions

This section presents a detailed discussion of the overall architecture of each YOLO version and its constituent blocks, based on the respective papers, source code, and technical documentation.

3.1 YOLO Overall Architecture Introduction

The YOLO architecture consists of three main parts: the backbone, neck, and head. Among them, the backbone plays a key role in extracting multi-scale features from the input image. This process generates feature maps at multiple resolutions through the successive stacking of convolutional layers and specialized blocks. Features at different scales are incorporated by the neck and sent to the head for prediction. In order for the model to efficiently extract multi-scale information from the input, this process usually entails upsampling and concatenating feature maps from several levels. The final predictions for object detection and classification are produced by the head. The model eventually produces bounding boxes and matching class labels for the detected objects in the image after receiving and analyzing the feature maps from the neck. [28].

YOLO is composed of several parts, the stem, downsampling layers, stages consisting of basic blocks, and head [27]. The stem, the first part of the network, is in charge of receiving and processing raw input data, such as images or videos. The downsampling layers are intended to gradually reduce the feature maps' spatial resolution in order to increase computational efficiency and extend the receptive field of the model. Several convolutional blocks, referred to as fundamental building blocks or structures, usually make up each stage of YOLO's design. Each stage typically processes data at a different resolution and generates features at a higher level of abstraction.

3.2 YOLOv8 Overall Architecture

The diagram was made using yolov8.yaml, which can be found at ultralytics/cfg/models/v8 [34]. Three parameters: `depth_multiple`, `width_multiple`, and

`max_channels`, determine the YOLOv8 variant. The number of bottleneck blocks in the C2f block is determined by the `depth_multiple` parameters. The output channels are specified by the `width_multiple` and `max_channels` parameters. Two convolution blocks with a stride of two and a kernel size of three make up the stem component of YOLOv8. These two stages lower input resolution and transform data into initial features. Figure 2 shows the YOLOv8 architecture.

The C2f block is used in YOLOv8's stage component. Blocks 2, 4, 6, 8, 12, 15, 18, and 21 comprise the eight stages. Shortcuts are used by the backbone (blocks 2, 4, 6, and 8) but not by the neck (blocks 12, 15, 18, and 21). Empirical findings from trial-and-error trials are used to determine whether or not shortcuts should be used [35]. A convolution block with a three-kernel size and a two-stride is used in YOLOv8's downsampling process. The spatial resolution of the output will be cut in half when stride 2 is used.

The SPPF (Spatial Pyramid Pooling Fast) block at the neck, which occurs immediately after the final block on the backbone, provides a multi-scale representation of the feature map. SPPF enables the model to capture data at many levels of abstraction by pooling at various sizes [36]. The neck has a few concat and upsample blocks. Upsampling increases the resolution of the feature map. YOLOv8 employs the nearest neighbor upsampling method. This method repeats the values of nearby pixels to fill the newly produced pixels in a larger feature map. Feature maps can be combined using concat block. Combining feature maps will increase the overall number of channels while maintaining the same resolution.

There are three heads on YOLOv8. Small objects are detected by the head linked to block No. 15. Medium objects are detected by the head attached to block No. 18, while large objects are detected by the head linked to block No. 21. It's crucial to remember that the object size in each YOLO variant is dependent on the image or video frame.

3.3 YOLOv9 Overall Architecture

The diagram is generated based on yolov9-c architecture file, as defined in the yolov9-c.yaml file located in the models/detect/ folder. There is an extra part known as the auxiliary in YOLOv9. Network parameters are updated and dependable gradients are produced in this section. The issue of information loss during a deep learning network's layers can be mitigated by offering extra information that links the input data with the output target [26]. The auxiliary component is used for training procedure only. To speed up the model without compromising accuracy, this portion can be removed during the inference phase. YOLOv9 architecture is presented in Figure 3.

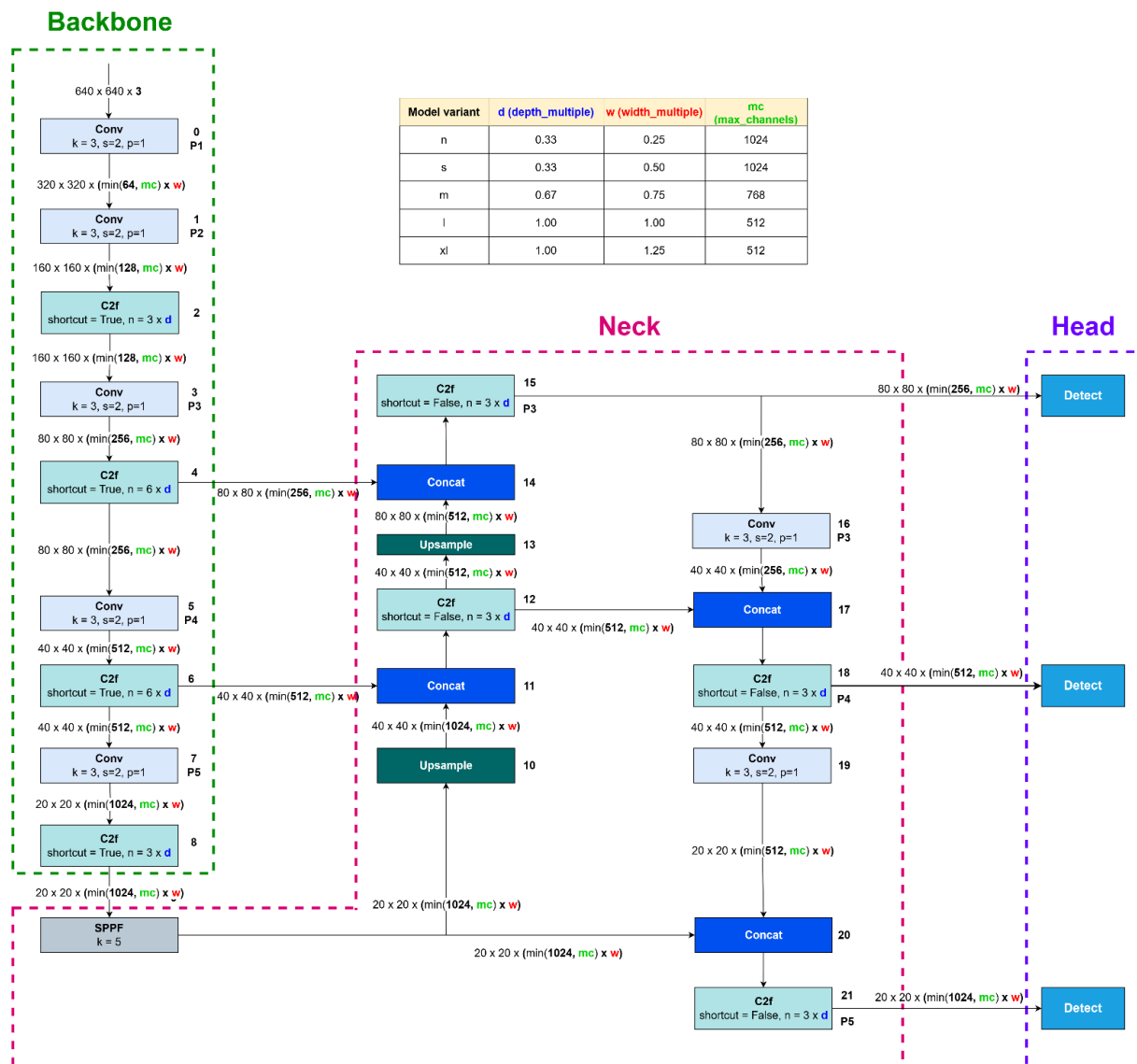


Figure 2. YOLOv8 Architecture [37], [38]

YOLOv9 begins with a Silence block. This block does not perform any transformation operations; it only returns the input. The auxiliary and backbone parts are connected to the YOLOv9 input by this block.

YOLOv9's stem component consists of two convolution blocks with a kernel size of three and a stride of two. These two blocks convert data into initial features and reduce input resolution.

YOLOv9's stage component makes use of the RepNCSPPELAN4 block. The GELAN concept is implemented in YOLOv9 via RepNCSPPELAN4. With gradient path planning, GELAN integrates CSPNet and ELAN (on YOLOv7) while taking accuracy, speed, and lightweight into account. ELAN capabilities are generalized by GELAN. While GELAN can employ a variety of computation blocks, including residual blocks or more complicated blocks, ELAN exclusively employs stacking of convolutional layers [1]. The following blocks: 3, 5, 7, 9, 13, 16, 19, and 22 comprise

the eight stages of YOLOv9. The ADown block is used in YOLOv9's downsampling procedure.

SPPELAN block is placed in the neck, immediately following the last block of the backbone. This block is identical to YOLOv8's SPPF (Spatial Pyramid Pooling Fast). Additionally, there are a few concat and upsample blocks. YOLOv9 employs nearest neighbor upsampling, similar to YOLOv8.

YOLOv9 has three heads. Attached to block No. 16, the first head's function is to identify small objects. Medium objects are detected by the head linked to block No. 19. Large objects are detected by the head connected to block No. 22.

Additional details that connect the input data to the desired output are given in the auxiliary section. As a result, this part includes multiple blocks that are identical to those in the backbone, including two convolutions, RepNCSPPELAN4, and ADown, using input from the Silence block.

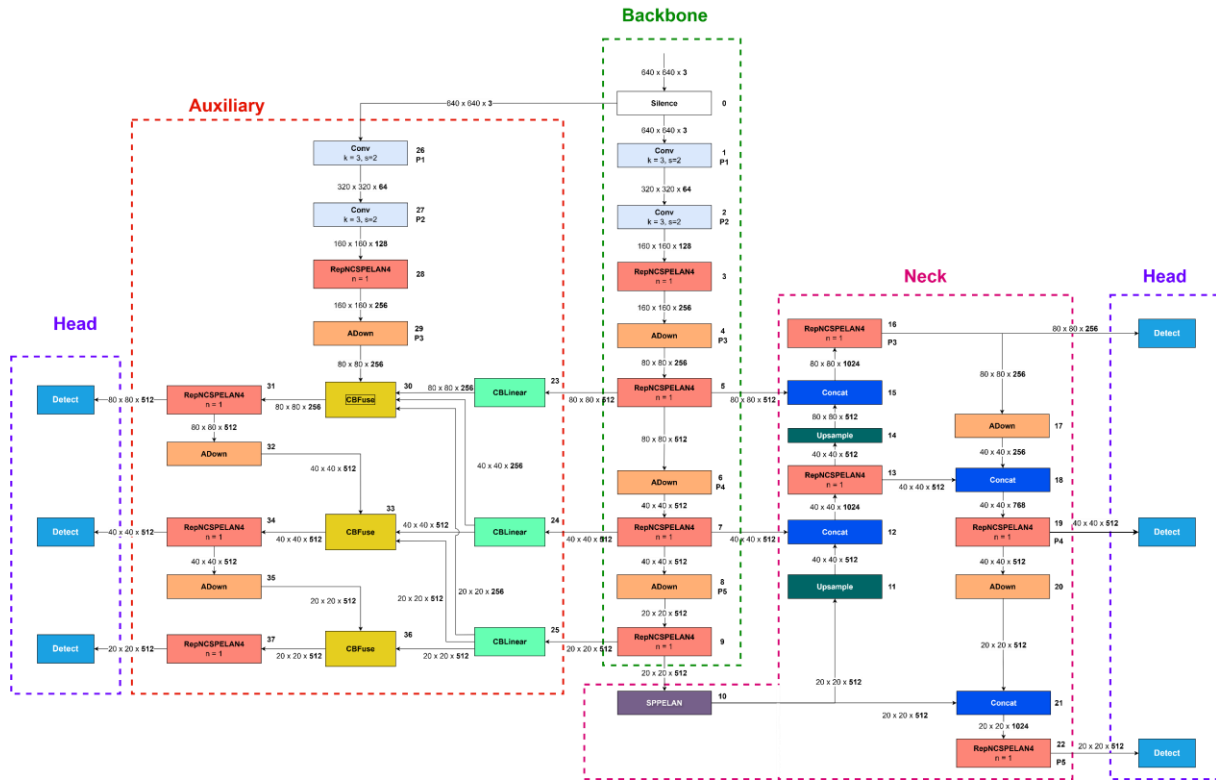


Figure 3. YOLOv9-c Architecture [39]

Three CBLinear blocks are present. These blocks are used to extract higher-level characteristics from the first backbone and to construct various pyramidal feature maps. Reversible properties that integrate high-level features from the first backbone and low-level features from the second backbone are common in CBFuse blocks.

One novelty in YOLOv9, aside from the auxiliary section, is extra heads. Three more heads are present. Small, medium, and large objects are detected by heads attached to blocks No. 31, No. 34, and No. 37, accordingly.

3.4 YOLOv10 Overall Architecture

The diagram was made using yolov10l.yaml, which can be found in the ultralytics/cfg/models/v10 folder [40]. YOLOv8 serves as the foundation for YOLOv10 architecture. As a result, its stem component and characteristics resemble those of YOLOv8. The difference is that in YOLOv10, the depth_multiple parameter specifies the number of bottleneck blocks within C2f as well as CIBs within the C2fCIB block.

The YOLOv10l architecture is depicted in Figure . The use of the C2fCIB block is where the other variations diverge from one another. Table 2 shows specifics on how each variant uses C2fCIB blocks.

The C2f or C2fCIB block is used in the stage component of YOLOv10. Blocks 2, 4, 6, 8, 13, 16, 19, and 22 comprise the eight stages. YOLOv10 incorporates the idea of rank-guided block design, which seeks to optimize the YOLO architecture by

considering the intrinsic rank analysis-based complexity of the basic blocks at each level of the model. A numerical value called intrinsic rank is used to indicate the number of significant singular values at the end of a basic block. The deeper stages of the YOLO model and larger-scale models usually include greater redundancy. At high redundancy stages, the bottleneck block in C2f will be replaced by CIB (Compact Inverted Block), converting it into C2fCIB [27]. For example, in the YOLOv10l variant shown in Figure 4, blocks 8, 13, 19, and 22 employ the C2fCIB. Table 2 summarizes the use of C2fCIB across different YOLOv10 variant.

Table 2. YOLOv10l Architecture [41]

Model Variant	C2fCIB
n	22
s	8, 22
m	8, 9, 12
b	8, 13, 19, 22
l	8, 13, 19, 22
x	6, 8, 13, 19, 22

YOLOv10x, the largest variant, contains the highest quantity of C2fCIB blocks. The C2fCIB blocks are located in blocks 6, 8, 13, 19, and 22. The Table indicates a reduction in the quantity of C2fCIB blocks in the smaller variant. The variant n model exemplifies the substitution of a C2f block for the designated C2fCIB block in the smaller variant. This model contains a C2fCIB block in block number 22; thus, blocks number 6, 8, 13, and 19 are occupied by C2f blocks. In terms of the downsampling procedure, YOLOv10 employs Spatial-Channel Decoupled Downsampling.

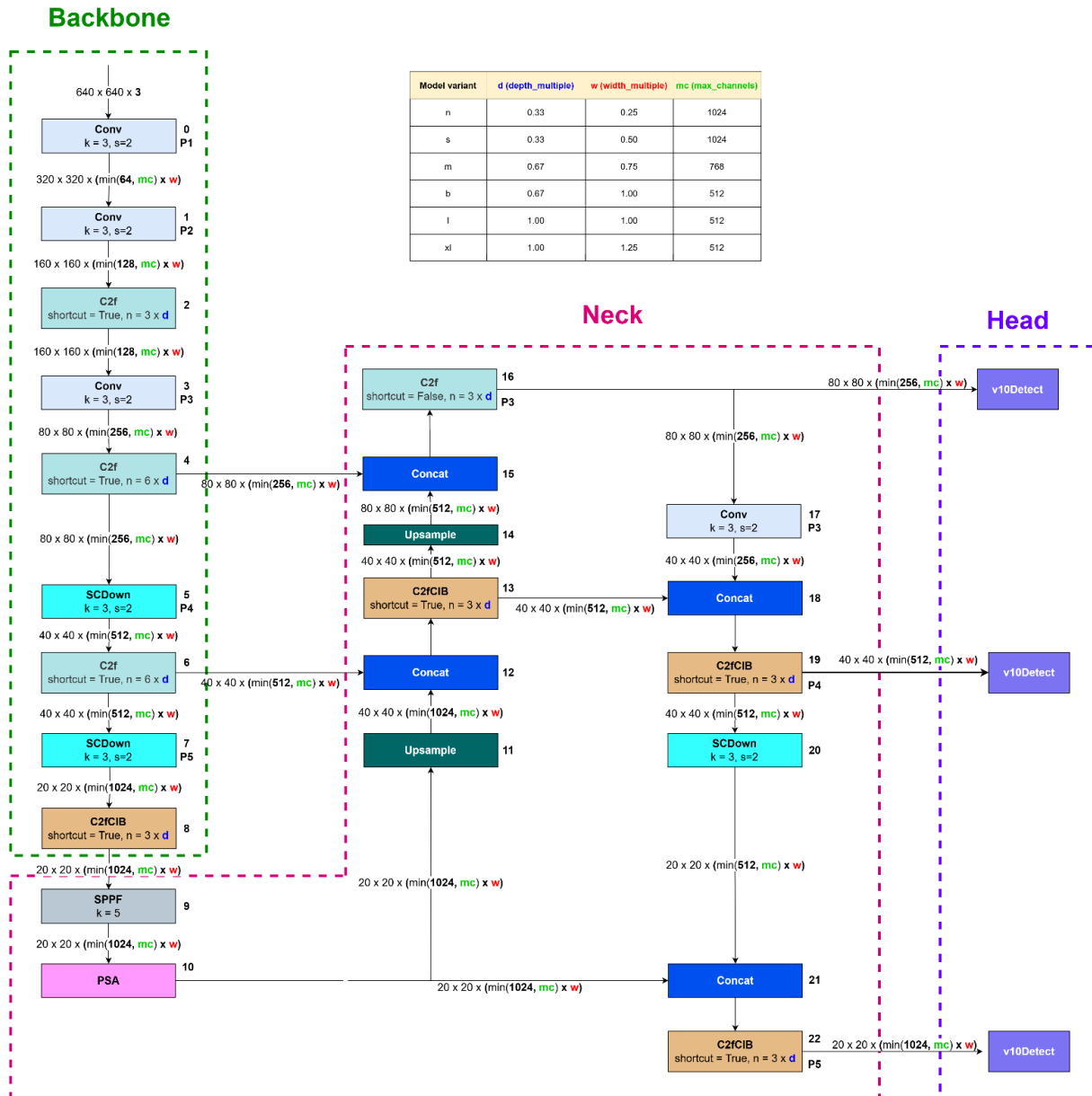


Figure 4. YOLOv10l Architecture [41]

Similar to the YOLOv8 architecture, a Spatial Pyramid Pooling Fast (SPPF) block is placed inside the neck immediately after the backbone's last block. The Partial Self Attention (PSA) block is then presented. To improve performance, PSA integrates a self-attention mechanism with global modeling capabilities [27]. To reduce the heavy computational overhead caused by the complexity of quadratic self-attention, PSA is only used at the lowest resolution after stage 4 [27].

Several concat and upsample blocks are present in the necks, similar to YOLOv8. The nearest neighbor upsampling is also used in the upsample block. Three heads are present. Small, medium, and large objects are detected by heads attached to blocks No. 16, No. 19, and No. 22, respectively.

3.5 YOLO11 Overall Architecture

The diagram was made using yolo11.yaml, which can be found in ultralytics/cfg/models/11 [42]. Three variables determine the YOLO11 variant. These parameters are the same as those used in YOLOv8, depth_multiple, width_multiple, and max_channels. The distinction is found in the function of depth_multiple, which determines the number of PSA blocks in C2PSA and bottleneck or C3k blocks in block C3k2. The output channels are determined by the width_multiple and max_channels variables. YOLO11's stem components match those of YOLOv8. An illustration of the YOLO11 architecture is shown in Figure 5.

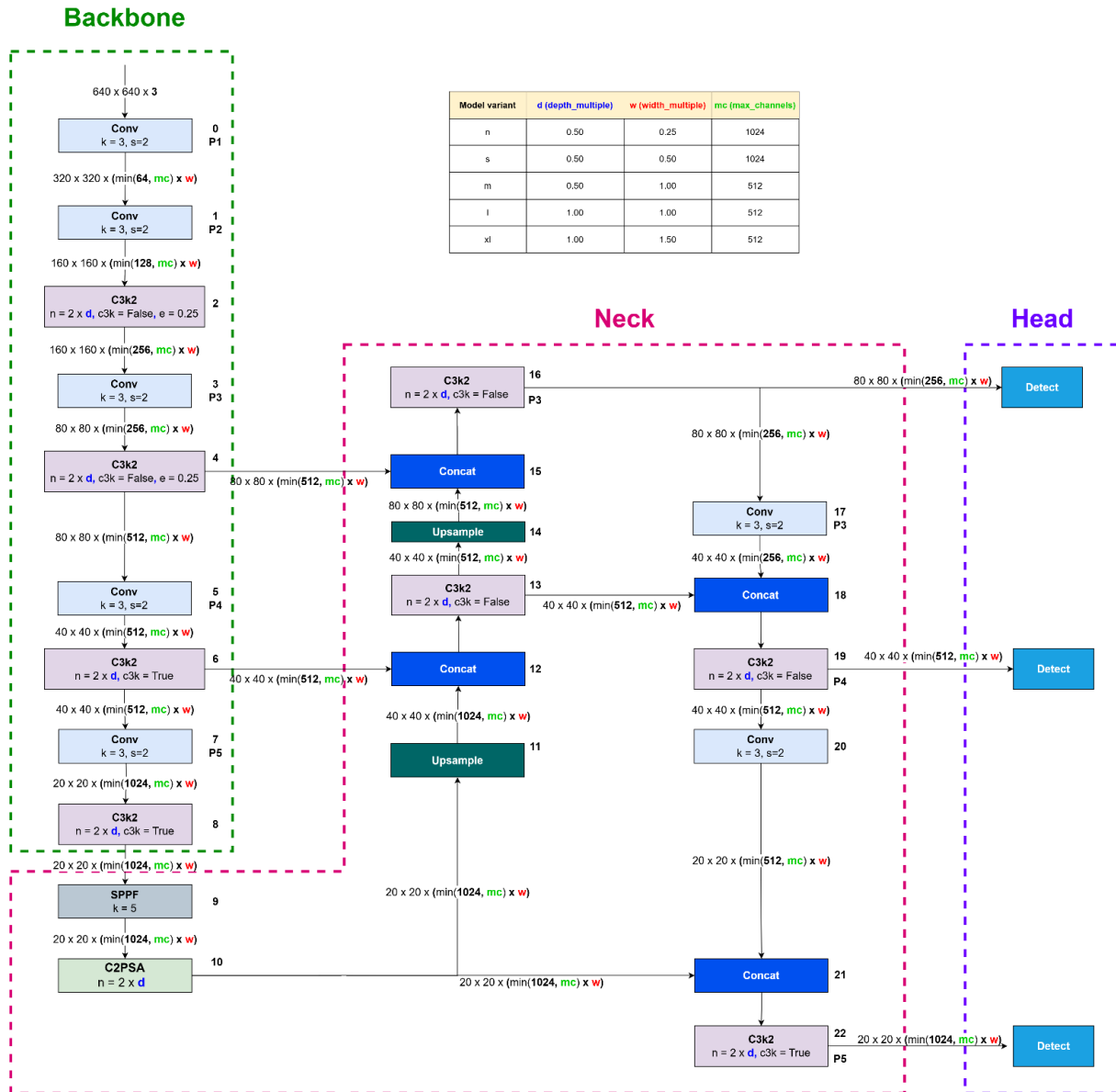


Figure 5. YOLO11 Architecture [43]

YOLO11 uses the C3k2 block in its stage component. C3k2 is an improvement of C2f in YOLOv8 and YOLOv10. When compared to C2f, C3k2 has less parameters. The implementation of the C3k block is determined by the c3k parameter found in the C3k2 block. It will use the C3k block if c3k parameter True. On the other hand, as seen in C2f, if c3k is False, it will make use of bottleneck. Nonetheless, c3k is always True in the m, l, and x variants. The YOLO11 documentation does not include this finding [25]. Rather, we discovered it by examining the source code. This experience led to the conclusion that understanding the YOLO model requires a thorough understanding of the source code, not only reading the technical documentation. Block numbers 2, 4, 6, 8, 13, 16, 19, and 22 correspond to the eight stages. YOLO11 performs downsampling using a convolutional block with stride 2 and kernel size 3. The spatial resolution that results from using a stride of 2 will be reduced by

50%. Immediately after the backbone's final stage, the neck incorporates an SPPF (Spatial Pyramid Pooling Fast) block, which is also found in YOLOv8. The C2PSA block comes next. Similar to the PSA in YOLOv10, C2PSA uses a self-attention technique that combines global modeling capabilities to improve accuracy. C2PSA is only placed with little resolution following stage 4. Similar to YOLOv8 and YOLOv10, multiple concatenation and upsampling blocks are used in the neck. Nearest-neighbor upsampling is also used in the upsample block. In terms of functionality and block number, the head portion of YOLO11 and YOLO10 are identical.

3.6 YOLO Architecture Blocks Introduction

In the following section, we explain in detail every YOLO architecture block. Prior to explaining the architectural blocks, it is essential to address the input image resizing process of the input image in YOLO.

When an image is resized into a lower dimension, critical information and fine features may be lost [44]. However, YOLO resizes the image to achieve real-time performance and to reduce training time [45].

In YOLOv8 through YOLO11, during resizing, the aspect ratio of the input images is kept unchanged, in contrast to YOLOv4 and its predecessors. Gray pixels will be added to the image to maintain its aspect ratio. The input image will be scaled without padding if it is square. Figure 6 shows an example of how to resize an image.

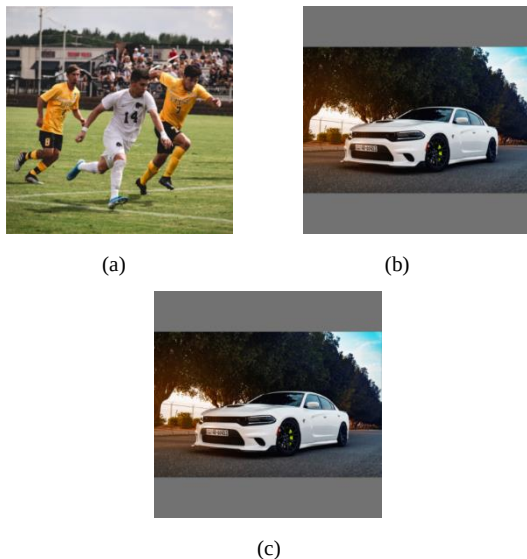


Figure 6. Image resizing in YOLO for different aspect ratios: (a) Square-shaped Image (b) Landscape-oriented Image (c) Portrait-oriented Image

3.7 Convolutional (Conv) Block

Across all YOLO versions examined in this study, the convolutional block remains consistent, consisting of a 2D convolution layer, 2D batch normalization, and a SiLU activation function. These components are fused to form a single convolutional block. But there are two new convolutional blocks in YOLO11. The convolutional block without an activation function and the depthwise convolutional block (DWConv).

A Conv block devoid of an activation function is used by the feed-forward network (FFN) in PSABlock (Position-Sensitive Attention Block). DWConv is lighter since depthwise convolution only uses one kernel to work on a single channel. Every kernel has a different matrix value.

3.8 Downsampling Block

For downsampling, the YOLOv8 and YOLO11 variants both use standard 3x3 convolution with a stride of 2. However, because of its computational expenses and comparatively high number of factors, this conventional approach is resource-intensive and may be less effective.

YOLOv10 utilizes Spatial-Channel Decoupled Downsampling (SCDown), which separates spatial

reduction from channel expansion. First, pointwise convolution (1×1) is applied to adjust the channel dimension while preserving spatial resolution. Followed by depthwise convolution (e.g., 3×3) that reduces spatial dimensions without altering the number of channels. Both the number of parameters and the computational cost are decreased by this method [27].

Adaptive Downsampling (ADown) is used by YOLOv9. This block performs downsampling by combining Average Pooling with Max Pooling. Since pooling requires no parameters, ADown has a slightly lower parameter count [46].

3.9 Bottleneck

A block that resembles the ResNet block is called the bottleneck. This block utilized in YOLOv8, YOLOv10, as well as YOLO11. Deeper networks are made possible by bottlenecks, which stack more layers without significantly raising computational costs. There are bottlenecks that use shortcuts and those that do not.

The block structure of a shortcut-using bottleneck is made up of two parallel paths: the skip connection path and the main path. The skip connection path transfers the input straight to the output unaltered, whereas the main path has two convolutional blocks. After then, element-by-element summation is used to merge the outputs of these two routes. In contrast, the block structure in a bottleneck without shortcuts is made up of just two convolutional blocks without any extra connection path.

3.10 C2f Block

C2f is designed as an optimized version of the CSP bottleneck utilizing two convolution layers. At every stage, C2f is used for feature extraction. By using feature vector switching and multilayer convolution, C2f blocks are capable of learning multiscale features while broadening the receptive field. C2f is available in YOLOv8 and YOLOv10.

The Cross-Stage Partial (CSP) principle is used by the C2f block to process feature maps. This block divides the feature map into two parts, one of which is kept as a shortcut and the other of which is processed more thoroughly utilizing the bottleneck. A richer feature representation is then created by reassembling the two components.

3.11 C2fCIB, CIB, RepVGGDW

YOLOv10, which uses a rank-guided block design paradigm, introduced C2fCIB in addition to C2f. This approach uses intrinsic rank analysis to adjust the fundamental blocks' complexity at each model step in order to optimize the YOLO model architecture. The intrinsic rank provides a numerical measure of how many singular values in the last convolution of a basic block are significant.

As they get deeper, YOLO and other large-scale models typically have more redundancy. The bottleneck block

in C2f will be swapped out for CIB (Compact Inverted Block) at stages with substantial redundancy to create C2fCIB. In the case of YOLOv10l, blocks 8, 13, 19, and 22 employ C2fCIB. In comparison to YOLOv8, this approach boosts the model's training speed, hence improving its efficiency [27].

Depthwise convolution is used in CIB. Its structure follows an alternating pattern of depthwise convolution and standard convolution blocks. While some CIBs use shortcuts, others do not.

YOLOv10 introduces the idea of large-kernel convolution. This technique allows for the expansion of the input region that influences the output of the neurons in the convolution layer. Only the model's deep stages employ large-kernel convolution. The second depthwise convolution in CIB is replaced with RepVGWDW during implementation. RepVGWDW includes two depthwise convolution blocks with kernel sizes of 3 and 7. Large-kernel convolution is applied only in the YOLOv10 n and s variants, as it does not yield improvements in larger variants.

3.12 C3k2 Block

In YOLO11, this block is a recent addition. C3k2 serves as a replacement for the C2f block and exhibits a comparable structure. Additionally, the first version C3k2 and C2f are identical. The bottlenecks were swapped out for a single C3k block in the second version of C3k2.

C3k is a block that has multiple bottlenecks and three convolutional blocks. There are two variants of the bottleneck: one that has a shortcut (shortcut=true) and another that does not (shortcut=false). The C3k2 block is suitable for objects of different sizes since it efficiently captures more intricate details.

3.13 SPPF & SPPElan

The block known as Spatial Pyramid Pooling – Fast, or SPPF, allows feature maps to be rendered at various scales. SPPF allows the model to capture information at multiple levels of abstraction through multi-scale pooling. Initially included in YOLOv8, this block was later adapted for YOLOv10 and YOLO11.

A variation of Spatial Pyramid Pooling (SPP), which was present in the YOLO model prior to YOLOv8, is represented by SPPF. In SPP, the maximum kernel sizes are 3, 5, and 9. By comparison, SPPF uses a kernel size of 5. By making this change, SPPF uses fewer floating-point operations (FLOP) than SPP. The structure and functionality of several of the YOLOv9 blocks are similar to those of SPPF. However, it is called SPPELAN in YOLOv9.

3.14 C2PSA & PSABlock

The attention module is used to determine the relationships between pixels in a picture or video frame. Each pixel in the image preserves three parameters: query, key, and value. To run queries on a specific pixel,

the dot product is applied to all keys from other pixels and used accordingly. To determine which pixels require attention, the likelihood that results from the dot product will be utilized. In other terms, the pixels that are observed are similar to these pixels.

In YOLOv10 and YOLO11, attention block is initially introduced. The attention process is depicted as C2PSA in YOLO11. Two convolutional blocks are located at the beginning and the end of this block. The first convolutional block's result is split. Multiple PSABlocks will process the first segment of the findings to determine which components require attention. The second part of the findings will be combined with the PSABlock results using a concat block.

FFN (Feed Forward Network) and an Attention block are included in PSABlock. Two successive convolutional blocks make up FFN. However, an activation function is not included in the second convolutional block.

Similarly, YOLOv10 has PSA (Partial Self Attention) modules that help highlight areas of the image that require attention. Though it has a simpler structure, this block is comparable to C2PSA. It starts with a convolutional block that divides the output into two parts. One part is passed directly to the concatenation block, whereas the other is processed through both a convolutional block and an attention block. The concatenated results are processed by a convolutional block at the end of the block.

In YOLOv10, the PSA consists of a single module with one feed-forward layer and one attention layer. C2PSA is a sophisticated module made up of several PSA blocks. Because it executes the PSA procedure iteratively, collecting deeper feature correlations, C2PSA is more sophisticated than PSA. Because the initial split output convolution divides the input into two pathways, shortcut and deep processing, C2PSA effectively catches deeper characteristics without requiring additional time.

3.15 Detect Block

The detect block in YOLOv8 and YOLOv9 consists of two parts: the regression head and the classification head. While the regression head generates bounding box coordinate predictions, the classification head generates class probabilities. Two convolutional blocks make up each head, followed by a two-dimensional convolution layer.

The regression head and classification head in the YOLOv10 detect block have different structures. The difference in computational expenses made this separation necessary. The classification head may be twice as large as the regression head.

However, in determining performance, the regression head is more crucial. As a result, the classification head architecture can be optimized and made lighter by using

depthwise convolutions. This approach decreases the number of model parameters.

For NMS-free training, YOLOv10 presents the concept of Consistent Dual Assignment. This concept uses consistent matching metrics to combine one-to-many and one-to-one strategies in the training process. During training, this approach uses two heads: one uses one-to-many assignment for robust supervision, while the other uses one-to-one matching. Only the head using one-to-one matching is used during inference, removing the need for NMS and increasing inference efficiency without compromising performance [27].

Similar to YOLOv10, YOLO11's detect block employs depthwise convolutions in the classification head. However, YOLO11 uses a novel block, DWConv.

The analysis presents a detailed comparison of the four YOLO versions as outlined in Table 3. Table 4 presents a comparison of the advantages and disadvantages of the four YOLO versions. Current YOLO versions reveal clear trends characterized by an anchor-free approach and the incorporation of attention mechanisms.

Table 3. Comprehensive Comparison of YOLO Models

Aspect	YOLOv8	YOLOv9	YOLOv10	YOLO11
NMS	NMS	NMS	NMS-free	NMS
Anchor	Anchor-free	Anchor-free	Anchor-free	Anchor-free
Novelty	C2f SPPF	PGI GELAN ADown	SCDown PSA Conv(DepthWise)	C3k2 C2PSA DWConv

Table 4. Different YOLO Versions Advantages and Disadvantages

Model	Advantages	Disadvantages
YOLOv8	Simplifies the architecture by implementing anchor-free detection.	It is less effective at capturing the global relationship between features since it does not use the attention mechanism.
YOLOv9	In order to produce more reliable gradients during training without compromising performance during inference, an auxiliary section was added. Using Adaptive Downsampling (ADown), which is more parameter-efficient.	Auxiliary sections increase the complexity of the architectural structure. More Training Complexity and Resource Consumption
YOLOv10	Adopt NMS-free training to facilitate faster inference and easier deployment. Rank-guided block design (C2fCIB) modifies the block complexity at every level to increase efficiency. Partial Self Attention (PSA) was incorporated to improve the model's capacity to identify global feature correlations.	Implementing NMS-free requires a more complex training methodology (dual assignment).
YOLO11	Incorporated supplementary blocks that enhance feature extraction and object detection functionalities, such as C3k2 and C2PSA. Spatial attention capabilities are strengthened by the C2PSA (Convolutional Block with Parallel Spatial Attention) block.	The architecture is inherently complex. The complexity resides in the code implementation, making understanding and alteration more challenging.

The anchor-free approach enhances the accuracy of bounding box localization and accelerates predictions. The attention mechanism captures more intricate feature relationships and effectively acquires deeper features without incurring additional time costs. Conversely, there is a distinction among YOLO versions regarding the implementation of NMS. YOLOv10 is the sole model that does not employ NMS.

Each model introduces its own novelties. YOLOv8 introduces C2f that replaces C3 block in YOLOv5 [47]. It also introduces SPPF, which is faster than the SPP block in YOLOv5 [47]. Although YOLOv8 is the oldest in this study, it remains the most popular YOLO version [48].

YOLOv9 incorporates PGI to address information bottlenecks by integrating an auxiliary section into the training process. GELAN employs gradient path planning in its design to enhance the efficiency of lightweight models. Furthermore, YOLOv9 utilizes ADown, an innovative downsampling method that integrates max pooling and average pooling.

YOLOv10 introduces C2fCIB, which substitutes certain C2f blocks, particularly in stages characterized by significant redundancy. It employs SCDown (Spatial-Channel Decoupled Downsampling), which separates spatial resolution from channel reduction to enhance efficiency. Additionally, YOLOv10 introduces PSA (Partial Self-Attention), an attention mechanism

implemented at low resolution. During the detection phase, YOLOv10 utilizes a dual assignment training technique (one-to-many and one-to-one), enabling inference without Non-Maximum Suppression (NMS-free).

YOLO11 advances prior concepts by incorporating C3k2 instead of C2f/C2fCIB. This module aims to be more lightweight while preserving its capability to extract complex features. YOLO11 employs C2PSA, an enhanced version of PSA, for its attention module. This module provides in-depth processing of global features. In the classification section of the head, YOLO11 employs DWConv rather than the conventional Conv block.

Performance benchmarking was conducted utilizing the Roboflow 100 dataset, which consists of 224,714 images [49]. Benchmarking using this dataset provides a novel perspective for the computer vision community on the performance of the four YOLO models, as the majority of existing literature benchmarks against the MS-COCO dataset. According to [49], MS-COCO largely comprises images sourced from the web, while Roboflow 100 is developed from the web application Roboflow Universe, contributed by practitioners in computer vision, which makes the dataset represent more realistic domains. The benchmark is presented in Tables 5 and 6.

Table 5. Accuracy Benchmark

Model	mAP@50	Precision	Recall
YOLOv8m	0.72	0.76	0.71
YOLOv9m	0.78	0.79	0.78
YOLOv10m	0.70	0.73	0.69
YOLO11m	0.72	0.76	0.71

Table 6. Speed Benchmark

Model	Params (M)	FLOPs (G)	Average Inference time (ms)
YOLOv8m	25.9	78.9	58.84
YOLOv9m	20.0	76.3	505.91
YOLOv10m	15.4	59.1	55.16
YOLO11m	20.1	68.0	73.22

According to our benchmark, YOLOv9 achieved the highest accuracy of all models evaluated. The auxiliary section has proven to be effective despite having fewer parameters and floating-point operations per second (FLOPs). Unfortunately, its training required nearly twice the resources compared to the other models. Moreover, the inference duration is markedly greater than that of the others, exceeding the average inference time by more than eight times. We attribute the excessive inference time of YOLOv9 to the NMS algorithm employed by it. The algorithm encountered significant difficulties, particularly with images containing numerous objects. For instance, YOLOv9 encountered difficulties in 8 out of 100 datasets within the Roboflow 100 dataset due to the presence of hundreds of objects in each image. We encountered numerous warnings during inference, indicating that the NMS process exceeded the time limit of 3 seconds. We also observed this warning during inference with

YOLOv8 and YOLO11, although not as frequently as with YOLOv9.

YOLOv10 is the fastest model. It is evidently clear due to its minimal number of parameters and FLOPs. Furthermore, it implemented an NMS-free mechanism. YOLOv8 and YOLO11 provide a balance between accuracy and speed.

4. Conclusions

According to our thorough study, every version adds architectural enhancements and feature extraction improvements. Nonetheless, we found some architectural blocks that are exactly the same as in the earlier version. Additionally, we found that YOLO has evolved to include an attention layer starting with YOLOv10.

The lack of official architecture diagrams and academic papers makes it difficult for scholars and practitioners to comprehend how the model works. This problem also makes it difficult to improve the model in the future. To help scholars, practitioners, and developers better understand the model and improve its performance going forward, we encourage future YOLO developers to provide both architectural diagrams and scientific papers.

Specifically, we found that, except for YOLOv10 (NMS-free), YOLOv8 through YOLO11 share commonalities in anchor-free and NMS features but differ in novelty. The various levels of complexity in the code implementation (v11), training methodology or dual assignment (v10), architectural structure and training time (v9), and the link between features (v8) are some of the disadvantages of each YOLO version in addition to its advantages. In the future, a more comprehensive YOLO analysis using a mathematical model is encouraged. It will be very beneficial to the computer vision community to have such an analysis.

In terms of performance, YOLOv9 demonstrates the highest accuracy. We assert that the auxiliary is responsible for achieving its highest level of accuracy. However, this model is the slowest, operating at a rate eight times slower than its counterparts. Our observation indicates that the YOLOv9 NMS mechanism is the primary cause of this issue.

YOLOv10 is the fastest model. It possesses the smallest number of parameters and employs an NMS-free mechanism. Unfortunately, it exhibits the lowest accuracy. YOLOv8 and YOLO11 achieve a trade-off between accuracy and inference speed.

References

- [1] C.-Y. Wang and H.-Y. M. Liao, "YOLOv1 to YOLOv10: The Fastest and Most Accurate Real-time Object Detection Systems," *APSIPA Transactions on Signal and Information Processing*, vol. 13, no. 1, p. Article 1, 2024, doi: [10.1561/116.20240058](https://doi.org/10.1561/116.20240058).

- [2] M. Hussain, "YOLOv1 to v8: Unveiling Each Variant—A Comprehensive Review of YOLO," *IEEE Access*, vol. 12, pp. 42816–42833, 2024, doi: [10.1109/ACCESS.2024.3378568](https://doi.org/10.1109/ACCESS.2024.3378568).
- [3] M. Hussain, "YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection," *Machines*, vol. 11, no. 7, p. 677, Jun. 2023, doi: [10.3390/machines11070677](https://doi.org/10.3390/machines11070677).
- [4] R. Sapkota et al., "YOLO advances to its genesis: a decadal and comprehensive review of the You Only Look Once (YOLO) series," *Artif Intell Rev*, vol. 58, no. 9, p. 274, Jun. 2025, doi: [10.1007/s10462-025-11253-3](https://doi.org/10.1007/s10462-025-11253-3).
- [5] J. Terven, D.-M. Cordova-Esparza, and J.-A. Romero-Gonzalez, "A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023, doi: [10.3390/make5040083](https://doi.org/10.3390/make5040083).
- [6] A. Allmendinger, A. O. Saltık, G. G. Peteinatos, A. Stein, and R. Gerhards, "Assessing the capability of YOLO- and transformer-based object detectors for real-time weed detection," *Precision Agric*, vol. 26, no. 3, p. 52, Jun. 2025, doi: [10.1007/s11119-025-10246-0](https://doi.org/10.1007/s11119-025-10246-0).
- [7] G. Li, R. Jian, X. Jun, and G. Shi, "A Review of You Only Look Once Algorithms in Animal Phenotyping Applications," *Animals*, vol. 15, no. 8, p. 1126, Apr. 2025, doi: [10.3390/ani15081126](https://doi.org/10.3390/ani15081126).
- [8] Ahealisi Yeerjiang et al., "YOLOv1 to YOLOv10: A Comprehensive Review of YOLO Variants and Their Application in Medical Image Detection," *jaip*, vol. 7, no. 3, 2024, doi: [10.23977/jaip.2024.070314](https://doi.org/10.23977/jaip.2024.070314).
- [9] M. G. Ragab et al., "A Comprehensive Systematic Review of YOLO for Medical Object Detection (2018 to 2023)," *IEEE Access*, vol. 12, pp. 57815–57836, 2024, doi: [10.1109/ACCESS.2024.3386826](https://doi.org/10.1109/ACCESS.2024.3386826).
- [10] S. S. Debsarkar, B. Aronow, and V. B. S. Prasath, "Advancements in automated nuclei segmentation for histopathology using you only look once-driven approaches: A systematic review," *Computers in Biology and Medicine*, vol. 190, p. 110072, May 2025, doi: [10.1016/j.compbiomed.2025.110072](https://doi.org/10.1016/j.compbiomed.2025.110072).
- [11] M. Mao and M. Hong, "YOLO Object Detection for Real-Time Fabric Defect Inspection in the Textile Industry: A Review of YOLOv1 to YOLOv11," *Sensors*, vol. 25, no. 7, p. 2270, Apr. 2025, doi: [10.3390/s25072270](https://doi.org/10.3390/s25072270).
- [12] R. Ayachi, Y. Said, M. Afif, A. Alshammari, M. Hleili, and A. Ben Abdelali, "Assessing YOLO models for real-time object detection in urban environments for advanced driver-assistance systems (ADAS)," *Alexandria Engineering Journal*, vol. 123, pp. 530–549, Jun. 2025, doi: [10.1016/j.aej.2025.03.077](https://doi.org/10.1016/j.aej.2025.03.077).
- [13] A. R. Khakan, H. S. Aghdasi, and P. Salehpour, "The Impact of YOLO Algorithms Within Fall Detection Application: A Review," *IEEE Access*, vol. 13, pp. 6793–6809, 2025, doi: [10.1109/ACCESS.2024.3496823](https://doi.org/10.1109/ACCESS.2024.3496823).
- [14] M. A. R. Alif and M. Hussain, "YOLOv1 to YOLOv10: A comprehensive review of YOLO variants and their application in the agricultural domain (Version 1)," *arXiv*, 2024, doi: [10.48550/ARXIV.2406.10139](https://doi.org/10.48550/ARXIV.2406.10139).
- [15] M. L. Ali and Z. Zhang, "The YOLO Framework: A Comprehensive Review of Evolution, Applications, and Benchmarks in Object Detection," *Computers*, vol. 13, no. 12, p. 336, Dec. 2024, doi: [10.3390/computers13120336](https://doi.org/10.3390/computers13120336).
- [16] C. M. Badgujar, A. Poulse, and H. Gan, "Agricultural object detection with You Only Look Once (YOLO) Algorithm: A bibliometric and systematic literature review," *Computers and Electronics in Agriculture*, vol. 223, p. 109090, Aug. 2024, doi: [10.1016/j.compag.2024.109090](https://doi.org/10.1016/j.compag.2024.109090).
- [17] A. Vijayakumar and S. Vairavasundaram, "YOLO-based Object Detection Models: A Review and its Applications," *Multimed Tools Appl*, vol. 83, no. 35, pp. 83535–83574, Mar. 2024, doi: [10.1007/s11042-024-18872-y](https://doi.org/10.1007/s11042-024-18872-y).
- [18] K. D. Apostolidis and G. A. Papakostas, "Delving into YOLO Object Detection Models: Insights into Adversarial Robustness," *Electronics*, vol. 14, no. 8, p. 1624, Apr. 2025, doi: [10.3390/electronics14081624](https://doi.org/10.3390/electronics14081624).
- [19] R. Varghese and S. M., "YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness," in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, Chennai, India: IEEE, Apr. 2024, pp. 1–6. doi: [10.1109/ADICS58448.2024.10533619](https://doi.org/10.1109/ADICS58448.2024.10533619).
- [20] M. Sohan, T. Sai Ram, and Ch. V. Rami Reddy, "A Review on YOLOv8 and Its Advancements," in *Data Intelligence and Cognitive Informatics*, I. J. Jacob, S. Piramuthu, and P. Falkowski-Gilski, Eds., in *Algorithms for Intelligent Systems*, Singapore: Springer Nature Singapore, 2024, pp. 529–545. doi: [10.1007/978-981-99-7962-2_39](https://doi.org/10.1007/978-981-99-7962-2_39).
- [21] M. Yaseen, "What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector," Aug. 28, 2024, *arXiv*: arXiv:2408.15857. doi: [10.48550/arXiv.2408.15857](https://doi.org/10.48550/arXiv.2408.15857).
- [22] M. Hussain, "YOLOv5, YOLOv8 and YOLOv10: The Go-To Detectors for Real-time Vision," 2024, *arXiv*. doi: [10.48550/ARXIV.2407.02988](https://doi.org/10.48550/ARXIV.2407.02988).
- [23] N. Jegham, C. Y. Koh, M. Abdelatti, and A. Hendawi, "YOLO Evolution: A Comprehensive Benchmark and Architectural Review of YOLOv12, YOLO11, and Their Previous Versions," Mar. 17, 2025, *arXiv*: arXiv:2411.00201. doi: [10.48550/arXiv.2411.00201](https://doi.org/10.48550/arXiv.2411.00201).
- [24] Y. Chen, X. Yuan, R. Wu, J. Wang, Q. Hou, and M.-M. Cheng, "YOLO-MS: Rethinking Multi-Scale Representation Learning for Real-time Object Detection," 2023, *arXiv*. doi: [10.48550/ARXIV.2308.05480](https://doi.org/10.48550/ARXIV.2308.05480).
- [25] Glenn Jocher, Paula Derrenger, and Muhammad Rizwan Munawar, "Home - Ultralytics YOLO Docs." Accessed: Jan. 13, 2025. [Online]. Available: <https://docs.ultralytics.com/>
- [26] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, "YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information (Version 2)," *arXiv*, 2024, doi: [10.48550/ARXIV.2402.13616](https://doi.org/10.48550/ARXIV.2402.13616).
- [27] A. Wang et al., "YOLOv10: Real-Time End-to-End Object Detection," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., Curran Associates, Inc., 2024, pp. 107984–108011. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/c34ddd05eb089991f06f3c5dc36836e0-Paper-Conference.pdf
- [28] R. Khanam and M. Hussain, "YOLOv11: An Overview of the Key Architectural Enhancements," 2024, *arXiv*. doi: [10.48550/ARXIV.2410.17725](https://doi.org/10.48550/ARXIV.2410.17725).
- [29] R. Sapkota et al., "YOLO11 to Its Genesis: A Decadal and Comprehensive Review of The You Only Look Once (YOLO) Series," 2024, *arXiv*. doi: [10.48550/ARXIV.2406.19407](https://doi.org/10.48550/ARXIV.2406.19407).
- [30] M. Hussain and R. Khanam, "In-Depth Review of YOLOv1 to YOLOv10 Variants for Enhanced Photovoltaic Defect Detection," *Solar*, vol. 4, no. 3, pp. 351–386, Jun. 2024, doi: [10.3390/solar4030016](https://doi.org/10.3390/solar4030016).
- [31] G. Jocher, J. Qiu, and A. Chaurasia, *Ultralytics YOLO*. (Jan. 2023). Python. Accessed: Aug. 22, 2025. [Online]. Available: <https://github.com/ultralytics/ultralytics>
- [32] K.-Y. Wong, *WongKinYiu/yolov9*. (Aug. 22, 2025). Python. Accessed: Aug. 22, 2025. [Online]. Available: <https://github.com/WongKinYiu/yolov9>
- [33] *THU-MIG/yolov10*. (Aug. 22, 2025). Python. THU-MIG. Accessed: Aug. 22, 2025. [Online]. Available: <https://github.com/THU-MIG/yolov10>
- [34] Glenn Jocher and Paula Derrenger, "yolov8.yaml." Accessed: Jan. 20, 2025. [Online]. Available: <https://github.com/ultralytics/ultralytics/blob/main/ultralytics/cfg/models/v8/yolov8.yaml>
- [35] G. Jocher, "YOLO by Ultralytics." [Online]. Available: <https://github.com/ultralytics/ultralytics/issues/1200#issuecomment-1454873251>
- [36] Glenn Jocher, "Understanding SPP and SPPF implementation · Issue #8785 · ultralytics/yolov5," GitHub. Accessed: Nov. 06, 2024. [Online]. Available: <https://github.com/ultralytics/yolov5/issues/8785>
- [37] Priyanto Hidayatullah and Refdinal Tubagus, *YOLOv8 Architecture Detailed Explanation - A Complete Breakdown*, (Oct. 28, 2023). Accessed: Jan. 20, 2025. [Online Video]. Available: <https://www.youtube.com/watch?v=HQXhDO7COj8>
- [38] RangeKing, "RangeKing," GitHub. Accessed: Jul. 31, 2023. [Online]. Available: <https://github.com/RangeKing>

- [39] Priyanto Hidayatullah and Refdinal Tubagus, *YOLOv9 Architecture Explained*, (Apr. 19, 2024). Accessed: Jan. 20, 2025. [Online Video]. Available: <https://www.youtube.com/watch?v=oZ6I1VHpil0>
- [40] W. Ao, "Yolov10l.yaml." May 23, 2024. [Online]. Available: <https://github.com/THU-MIG/yolov10/blob/main/ultralytics/cfg/models/v10/yolov10l.yaml>
- [41] Priyanto Hidayatullah and Refdinal Tubagus, *YOLOv10 Architecture Explained - A Complete Breakdown*, (Sep. 03, 2024). Accessed: Jan. 20, 2025. [Online Video]. Available: <https://www.youtube.com/watch?v=A6rHMzRvs98>
- [42] Glenn Jocher and Paula Derrenger, "yolo11.yaml." Accessed: Jan. 20, 2025. [Online]. Available: <https://github.com/ultralytics/ultralytics/blob/main/ultralytics/cfg/models/11/yolo11.yaml>
- [43] P. Hidayatullah and R. Tubagus, "YOLO11 Architecture-Detailed Explanation [Video recording]." Oct. 28, 2024. [Online]. Available: <https://www.youtube.com/watch?v=L9Va7Y9UT8E>
- [44] Y. Ying and L. KeXin, "Effect analysis of Self built low-quality scene image set super-resolution reconstruction based on deep learning technology," in *2024 International Conference on Electronic Engineering and Information Systems (EEISS)*, Changsha, China: IEEE, Jan. 2024, pp. 203–206. doi: [10.1109/eeiss62553.2024.00043](https://doi.org/10.1109/eeiss62553.2024.00043).
- [45] S. Saponara and A. Elhanashi, "Impact of Image Resizing on Deep Learning Detectors for Training Time and Model Performance," in *Applications in Electronics Pervading Industry, Environment and Society*, vol. 866, S. Saponara and A. De Gloria, Eds., in *Lecture Notes in Electrical Engineering*, vol. 866, Cham: Springer International Publishing, 2022, pp. 10–17. doi: [10.1007/978-3-030-95498-7_2](https://doi.org/10.1007/978-3-030-95498-7_2).
- [46] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023.
- [47] Glenn Jocher, Ayush Chaurasia, Alex Stoken, and Jirka Borovec, *YOLOv5*. (Nov. 22, 2022). Ultralytics. [Online]. Available: <https://github.com/ultralytics/yolov5>
- [48] "Google Trends," Google Trends. Accessed: Jul. 31, 2025. [Online]. Available: <https://trends.google.com/trends/explore?date=today%205-y&q=YOLOv8,YOLOv9,YOLOv10,YOLO11,YOLOv11&hl=en>
- [49] F. Ciaglia, F. S. Zuppichini, P. Guerrie, M. McQuade, and J. Solawetz, "Roboflow 100: A Rich, Multi-Domain Object Detection Benchmark," 2022, *arXiv*. doi: [10.48550/ARXIV.2211.13523](https://doi.org/10.48550/ARXIV.2211.13523).