***Abstract***: In the world of Machine Learning, Classification and Detection models are the stepping stone and serve as fundamental tools in various computer vision tasks, such as image recognition, object detection and pattern analysis. This research paper compares different detection and classification models' findings, assessing their performance. The models under consideration for classification are Googlenet, Resnet, VGGnet and a custom CNN model. In parallel, for object detection, we assess the effectiveness of YOLOv8, YOLOv9,YOLOv10 and their subsequent versions and their differences.

Concurrently, both classification and detection models will be compared with an emphasis on the ability to accurately identify and localize objects within the same dataset. The comparison of different frameworks allows us to gain a deeper insight into the trade-offs, and also highlight the specific areas of excellence for each model. We aim to inform researchers' and practitioners' model selection by highlighting how different models perform based on specific criteria.

**Key Words:**

**Introduction**

Machine learning has become an indispensable tool in computer vision, driving advancements in diverse applications such as object detection, pattern recognition, and image classification. Within this domain, classification and detection models play a critical role in interpreting visual data. Classification models excel at categorising images into predefined classes, while detection models not only identify objects within an image but also pinpoint their exact locations.

The application of these models is particularly valuable in environmental monitoring. Early and accurate detection and classification of events like forest fires can significantly mitigate potential damage and enhance response strategies. The existing methods of forest fire monitoring like manual on-ground inspections require extensive use of resources E.g. labour, and have many drawbacks regarding logistics and personnel safety. Other monitoring methods such as detection sensors need excessive capital to be installed and maintained for them to work properly. The outdated and existing methods have prove dangerous and unreliable, as the number of forest fires has surged significantly in recent years. The Bay Area Fire in California, 2020, is a prime example, with devastating consequences in terms of lives lost and land area destroyed. Losses of about 19 Billion $ were incurred and 33 lives were lost in the catastrophic forest fire. This research undertakes a thorough comparative analysis of leading classification and detection models specifically designed for forest fire analysis, aimed at preventing disasters like the Bay Area Fire and assisting firefighters in more effectively controlling and mitigating forest fires.

For classification, we have chosen GoogLeNet, ResNet, VGGNet, and a custom Convolutional Neural Network (CNN) model. These models are renowned for their robust architectures and proven effectiveness in various image classification tasks. Their successful deployment across diverse computer vision applications underscores their ability to accurately categorize images.

For object detection, we will assess the models' ability to accurately identify and localize forest fires within a consistent dataset. This task necessitates the detection of flames, which are crucial for timely intervention and damage control. Implementing these models for fire detection involves data collection and annotation, model training, and evaluation using metrics such as precision and accuracy.

By evaluating these models separately, our study aims to provide a deeper understanding of their strengths and weaknesses. We will compare the classification models based on their accuracy in categorizing images, while detection models will be evaluated for their precision in fire detection and localization. This two-pronged approach allows us to highlight the specific advantages of each model and the trade-offs involved in their use.

Our findings aim to guide researchers and practitioners in selecting the optimal model for their specific needs, providing insights into how different models perform under various criteria. This study not only contributes to the academic literature but also has practical implications for improving forest fire detection systems, ultimately leading to better environmental management and disaster response.

**2. Related Work**

The field of object detection has seen substantial advancements over the years, particularly with the integration of deep learning techniques. Traditional methods, which relied heavily on handcrafted features and conventional image processing algorithms, were often limited in their ability to accurately detect and classify objects across varying conditions. These limitations highlighted the need for more sophisticated models capable of handling complex visual data with greater precision and efficiency.

The advent of Convolutional Neural Networks (CNNs) marked a turning point in object detection, leading to the development of several key models. The R-CNN (Regions with CNN features) family of models—comprising R-CNN, Fast R-CNN, and Faster R-CNN—played a pivotal role in this evolution. R-CNN introduced the concept of region-based object detection by generating region proposals that were then classified using CNNs. However, the model's processing speed was a significant bottleneck. Fast R-CNN addressed this issue by integrating the region proposal and classification stages, but the model still required considerable computational resources.

Faster R-CNN further improved on its predecessors by introducing the Region Proposal Network (RPN), which generated region proposals more efficiently, thereby enhancing the model's speed without sacrificing precision. Despite these improvements, Faster R-CNN remained somewhat constrained by its processing speed, particularly in applications requiring real-time performance (Ren et al., 2015).

The YOLO (You Only Look Once) framework revolutionized object detection by approaching the problem as a single regression task rather than a classification problem with multiple stages. YOLO predicts bounding boxes and class probabilities directly from entire images in one pass, significantly improving detection speed. The initial YOLO model utilized GoogLeNet as its base network, which struck a balance between speed and accuracy, making it highly suitable for real-time applications. YOLOv1's innovative approach involved dividing the input image into a grid and predicting bounding boxes and class probabilities for each cell, which allowed for faster processing compared to the region-based methods (Redmon et al., 2016).

Subsequent versions of YOLO introduced further enhancements. YOLOv2, for example, incorporated multi-scale training, which improved mean Average Precision (mAP) values while maintaining the model’s speed advantage. These improvements solidified YOLO’s position as a leading framework for object detection, particularly in scenarios where both speed and accuracy are crucial (Redmon & Farhadi, 2017).

In addition to the standard YOLO models, variations like PP-YOLO have emerged to further optimize the framework's performance and efficiency. PP-YOLO introduced specific enhancements, including better optimization techniques and more efficient implementations, making it an even more robust option for various object detection tasks (Long et al., 2020). This versatility has allowed YOLO and its variants to be applied across a wide range of real-world scenarios, from simple object detection tasks to more complex applications.

The application of YOLO in fire and smoke detection has garnered significant attention in recent research due to its potential in enhancing real-time monitoring systems. For instance, Chetoui and Akhloufi (2023) conducted a study where they fine-tuned YOLOv8 and YOLOv7 models to improve their performance in wildfire detection. Their models demonstrated remarkable precision and recall, achieving a mean Average Precision (mAP) of 92.6% with YOLOv8, which outperformed other models such as YOLOv6, Faster R-CNN, and DEtection TRansformer (DETR). This study underscores the effectiveness of YOLO models in critical applications like wildfire detection, where timely and accurate detection is essential.

Another noteworthy contribution to this field is the work by Bahhar et al. (2024), who proposed a novel approach combining a staged YOLO model with an ensemble of CNNs for wildfire and smoke detection. This innovative architecture was designed to enhance the early detection of wildfires by accurately identifying and localizing both fire and smoke. The model achieved high classification accuracy and a strong F1-score, indicating its robustness in real-world applications. However, the study also highlighted several challenges, including the need for high-quality, real-world datasets, and the importance of addressing the potential for false alarms and ensuring the system's generalization across diverse environments.

These advancements illustrate the ongoing evolution of object detection models, particularly within the YOLO framework. As research continues to refine these models, their applicability in complex and critical real-world scenarios, such as wildfire detection, becomes increasingly evident. The focus on balancing speed, accuracy, and generalization in these models is crucial for their successful deployment in real-time applications where precision and responsiveness are paramount.

**2. Materials and Methods**

This comparative study presents the findings of 4 classification models namely GoogLeNet, ResNet, VGG-16 and a custom CNN model, various detection models that are ,YOLOv8 ,YOLOv9 ,YOLOv10 and their subsequent versions. The study explores the classification of fire and non-fire images in a given dataset using Convolution Neural networks (CNN) and the detection of fire for real time data labelling and monitoring.

**2.1 *Classification Models***

Classification models are essential in machine learning and computer vision for categorizing input data into specific classes based on learned patterns. In the realm of fire detection and classification, these models analyse and categorize images to identify fire. By leveraging deep learning architectures, such as VGG-16, ResNet, and custom CNN models, these systems effectively learn and generalize complex features from large datasets. This enables them to accurately detect and classify fire-related patterns, which is critical for applications in safety monitoring and emergency response.

**2.1.1 GoogLeNet**

GoogLeNet, presented by Szegedy et al., is a landmark in deep convolutional neural network design due to its innovative inception modules. These modules enable the network to handle multiple feature extraction processes in parallel, capturing features at various scales and complexities. This architectural approach enhances the network's ability to recognize complex patterns in images, making GoogLeNet well-suited for tasks such as fire detection where diverse and intricate features need to be identified.

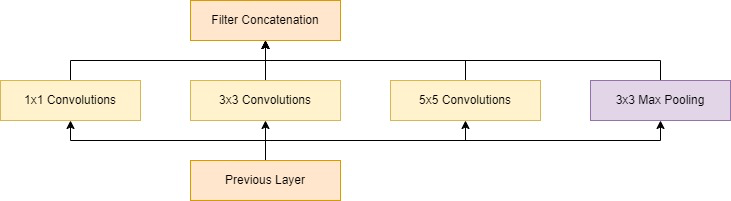
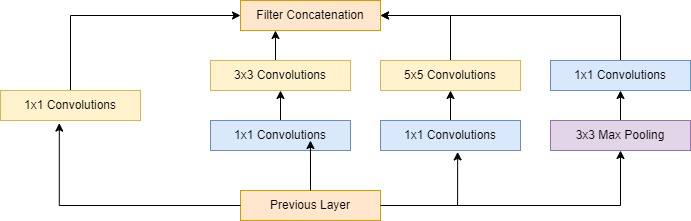


Figure 1 GoogLeNet Inception Modules

GoogLeNet employs inception modules that consist of multiple parallel convolutional layers with different kernel sizes—typically 1x1, 3x3, and 5x5—as well as max-pooling operations. The inception module's 1x1 convolutions serve to reduce dimensionality and computational complexity, while the 3x3 and 5x5 convolutions capture features at different spatial scales. The output from each convolutional path and pooling operation is concatenated along the depth dimension, resulting in a comprehensive feature map that integrates multi-scale information.

The network is organized into several inception modules stacked sequentially, interspersed with pooling layers to reduce spatial dimensions and control computational load. The final feature maps are processed through a global average pooling layer, which reduces each feature map to a single value by averaging all its spatial dimensions. This vector is then passed through fully connected layers and a final softmax layer that outputs classification probabilities for various fire types.

The input image is first processed by a convolutional layer with a large kernel (e.g., 7x7) to extract basic features. This output is fed into multiple inception modules. Each module processes the image through parallel paths: 1x1 convolutions for dimensionality reduction, 3x3 convolutions for medium-scale features, and 5x5 convolutions for large-scale features, alongside max-pooling for downsampling. The results are concatenated and passed through successive inception modules, refining the features at each stage. After processing through all inception modules, global average pooling converts the feature maps into a fixed-size vector. This vector undergoes further processing in fully connected layers before the softmax layer provides the final classification probabilities.

**2.1.2 ResNet (Residual Network)**

ResNet, introduced by He et al., revolutionized deep neural network training by incorporating residual connections, or skip connections, that bypass one or more convolutional layers. These connections allow the network to learn residual mappings, making it feasible to train very deep networks while addressing issues such as vanishing gradients and feature degradation.

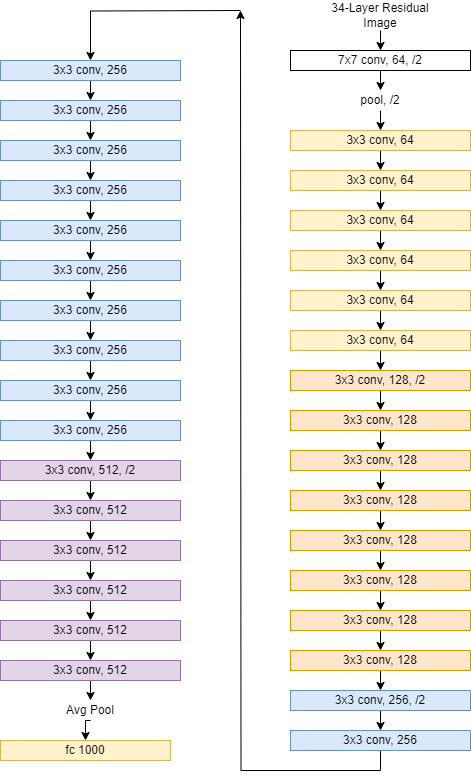
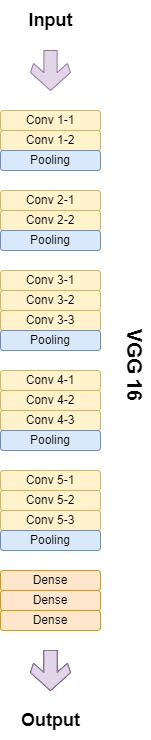


Figure 2 ResNet 34 layered Architecture

The ResNet architecture is built upon residual blocks, each comprising several convolutional layers with 3x3 kernels. A key feature of these blocks is the skip connection that directly adds the input of the block to its output, allowing the network to learn residuals instead of direct mappings. This design enhances gradient flow and facilitates training of very deep networks. ResNet networks are typically composed of multiple stages, with each stage consisting of a series of residual blocks.

Each residual block consists of two or more convolutional layers followed by batch normalization and ReLU activation functions. The skip connection adds the original input of the block to the output of the convolutional layers before applying a ReLU activation. The network uses global average pooling after the last residual block to compress the feature maps into a fixed-size vector. This vector is then fed into fully connected layers, leading to a softmax layer for classification.

An image enters through an initial convolutional layer to extract basic features. It is then processed through a series of residual blocks. Within each block, the image data flows through several convolutional layers with residual connections that bypass these layers. The residual connections add the input of the block directly to its output before activation, allowing the network to learn residual mappings. After passing through all residual blocks, global average pooling reduces the feature maps to a fixed-size vector. This vector is processed by fully connected layers, followed by a softmax layer that outputs classification probabilities.



**2.1.3 VGG-16**

VGG-16, developed by Simonyan and Zisserman, is a deep convolutional neural network renowned for its uniform architecture and effective performance in image classification tasks. The network is characterized by its use of small 3x3 convolutional filters stacked in multiple layers. This design enables VGG-16 to capture intricate patterns and hierarchical features in images, making it highly effective for tasks such as fire detection and classification.

Figure 3 VGG-16 Architecture

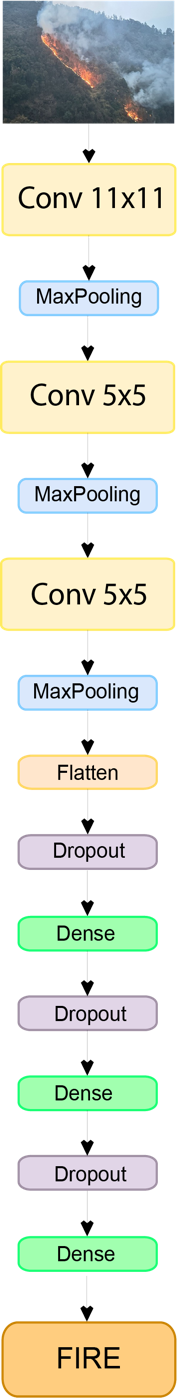
VGG-16 is structured into a series of convolutional blocks, each consisting of multiple convolutional layers with 3x3 kernels, followed by max-pooling operations. The network begins with a series of convolutional layers that progressively increase the depth of feature maps while reducing their spatial dimensions. The architecture is divided into five blocks, with the first two blocks containing two convolutional layers each, and the subsequent blocks containing three convolutional layers each.

Each convolutional layer in VGG-16 uses small 3x3 filters with a stride of 1, allowing the network to learn fine-grained features. Max-pooling layers with 2x2 kernels and a stride of 2 are used to downsample the feature maps, reducing their spatial dimensions and increasing the depth of the feature representations. After the convolutional blocks, the network uses flattening to convert the 3D feature maps into a 1D vector. This vector is then processed through three fully connected layers, with the final layer being a softmax classifier that outputs the probabilities for different classes.

The input image is first processed by a series of convolutional layers with 3x3 filters. Each convolutional layer is followed by a ReLU activation function to introduce non-linearity. After a few convolutional layers, max-pooling operations are applied to reduce the spatial dimensions of the feature maps. This process is repeated through the five convolutional blocks, progressively capturing more complex features.

Once the image has passed through all the convolutional and pooling layers, the resulting feature maps are flattened into a one-dimensional vector. This vector is then fed into fully connected layers, which aggregate the features and provide a final classification. The final layer is a softmax classifier that outputs the probability distribution over the classes, allowing the network to classify the input image into one of the predefined categories.

**2.1.4 Custom Convolution Neural Network(CNN)**:

The custom Convolutional Neural Network (CNN) architecture is tailored for specific classification tasks, including fire detection and classification. This model combines various convolutional and pooling layers with dense layers and dropout regularization to extract and classify features from images effectively. Its design aims to balance complexity and performance, leveraging deep learning techniques to achieve high accuracy in distinguishing between different classes.

The custom CNN model starts with an initial convolutional layer featuring 96 filters of size 11x11, with a stride of 4x4, and uses ReLU activation to introduce non-linearity and capture essential features from the input images. This is followed by a max-pooling layer with a pool size of 3x3 and a stride of 2x2 to reduce spatial dimensions and retain important features.

Figure 4 Custom CNN Architecture

The network continues with additional convolutional layers: the second layer uses 256 filters of size 5x5, and the third layer uses 384 filters of the same size. These convolutional layers are interspersed with max-pooling layers, each with a pool size of 3x3 and a stride of 2x2, to further reduce dimensions and enhance feature representation.

Following the convolutional and pooling stages, the model includes a flattening layer to convert the 3D feature maps into a 1D vector. This vector is then passed through several fully connected dense layers. The first dense layer has 2048 units with ReLU activation, followed by a dropout layer with a dropout rate of 0.25 to mitigate overfitting. The second dense layer contains 1024 units, also with ReLU activation, and is followed by another dropout layer with a dropout rate of 0.2. The final dense layer, with 2 units and a softmax activation function, produces the output probabilities for classification into two classes.

The input image, sized 224x224 pixels with 3 color channels, first undergoes convolution with 96 filters of size 11x11 and a stride of 4x4. The resulting feature maps are then downsampled using max-pooling with a pool size of 3x3 and a stride of 2x2. This process is repeated with additional convolutional layers (256 filters of 5x5, and 384 filters of 5x5) and corresponding max-pooling layers. After extracting hierarchical features through these layers, the feature maps are flattened into a 1D vector. This vector is processed through dense layers with ReLU activation, interspersed with dropout layers to prevent overfitting. The final output layer uses softmax activation to classify the input image into one of the two classes, based on the learned features.

**2.2 *Detection Models***

**2.2.1 YOLO**

YOLO (You Only Look Once) is the state of the art object detection algorithm renowned for its efficiency and precision. Released by ***Joseph Redmon et al. and his team under*** *titled “You Only Look Once: Unified, Real-Time Object Detection”.* Since its inception in 2016 until the present year (2024), the YOLO family has continued to evolve at a rapid pace. Releasing *yolov8(2023), yolov9(2024) and yolov10(2024).* YOLO treats object detection as a single regression task, predicting bounding boxes and class probabilities directly from entire images. It divides the image into an S×S×S grid, with each grid cell predicting a set number of bounding boxes, their confidence scores, and class probabilities. This method enables YOLO to analyse images in a single pass, greatly enhancing detection speed compared to traditional techniques that repeatedly apply models to different image sections. As a result, YOLO is ideal for applications demanding rapid and accurate object detection, such as real-time video processing and autonomous driving.

**2.2.2 YOLOv8**

The YOLOv8 algorithm, builds upon the principles of its predecessors, integrating improvements in both accuracy and computational efficiency. It maintains the core principle of single-stage detection but refines the architecture with a more sophisticated backbone, enhanced detection head, and advanced loss functions. With its improved grid division, enhanced confidence scoring, and refined model architecture, YOLOv8 processes images more efficiently in a single pass. YOLOv8 offers multiple model variants, including YOLOv8n, YOLOv8s, YOLOv8m, YOLOv8l, and YOLOv8x, each designed to meet different performance requirements. These improvements make YOLOv8 ideal for applications that require fast and accurate object detection, such as real-time video processing, autonomous driving, and various AI-driven monitoring systems.

The process begins with the input image undergoing pre-processing steps like resizing and normalization to prepare it for the model. This ensures consistency in how the image data is presented to the network, improving the model’s ability to generalize across different types of inputs.

Once pre-processed, the image is fed into the backbone of the YOLOv8 model, which is responsible for extracting hierarchical features from the input image. This feature extraction is performed by a series of convolutional layers combined with advanced modules such as C2f (Cross Stage Partial Network with Efficient Layer Aggregation Network) and SPPF (Spatial Pyramid Pooling-Fast), where each layer applies a series of filters to capture different aspects of the image, such as edges, textures, and more complex patterns. The convolutional layers are followed by batch normalization and an activation function, specifically SiLU (Sigmoid Linear Unit), to introduce non-linearity and stabilize the learning process.

The backbone also includes the C2f module, derived from the *C3 module of YOLOv5 and the ELAN module of YOLOv7*, which enhances feature extraction by splitting the feature map into parts, processing them independently through bottleneck blocks, and then concatenating the results. This approach not only preserves computational efficiency but also allows the model to capture more complex patterns by leveraging the strength of both depth and width in the network’s architecture.

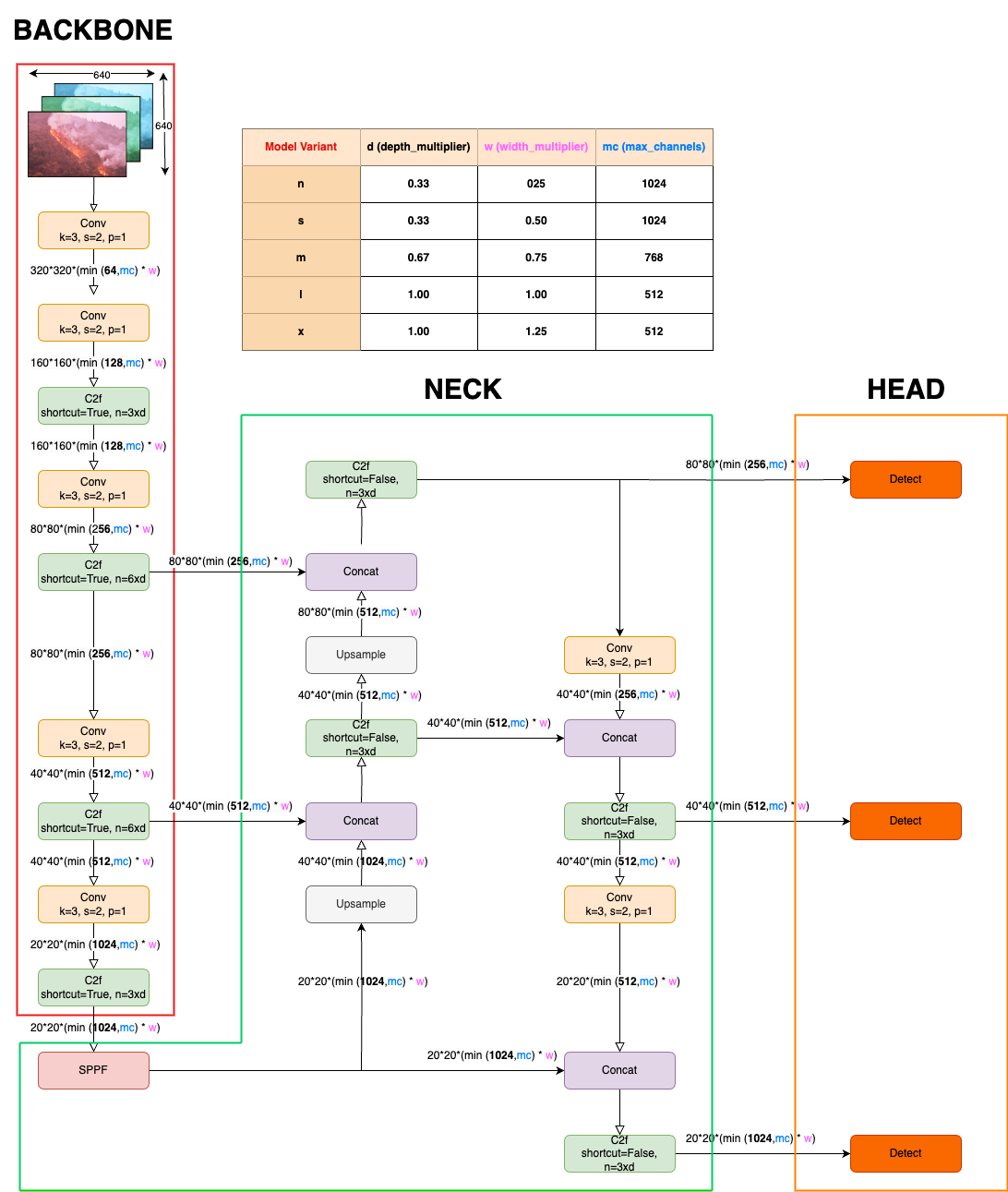
To further refine the feature extraction process, YOLOv8 incorporates the Spatial Pyramid Pooling-Fast (SPPF) module, which applies pooling operations with different kernel sizes to capture spatial hierarchies within the image. This is particularly useful for detecting objects like wildfires, which can vary significantly in size and shape. The SPPF module pools the input feature map at multiple scales, and then concatenates these pooled outputs to create a final feature map that encodes multi-scale information:

Figure 5 YOLOv8 Architecture

Following the backbone, the extracted features are passed to the neck of the YOLOv8 model, which further refines these features and prepares them for detection. It employs a PAN-FPN (Path Aggregation Network with Feature Pyramid Network) structure, which enhances the model’s ability to detect objects of varying scales and complexities. The PAN component aggregates features from different layers of the backbone, ensuring that fine-grained details from earlier layers are preserved and combined with higher-level abstract features. The FPN component, in turn, propagates strong semantic features from the deeper layers of the network to the shallower ones through a top-down pathway. In YOLOv8, the upsampling stages replace the traditional C3 modules with the more efficient C2f modules, omitting the need for the CBS (Convolutional, Batch Normalization, and SiLU activation) 1×1 convolutional structure. By upsampling the feature maps, the model ensures that information from the higher-resolution layers (which capture fine details) is retained and combined with the more abstract, lower-resolution features.

Interp is the interpolation method and scale factor that determines the ratio of upsampling. This combination of upsampled and concatenated features enhances the model’s capacity to detect objects that appear at different scales within the same image.

Finally, the processed features are passed to the head of the YOLOv8 model, where the actual object detection takes place. The head uses a decoupled approach, which separates the tasks of object classification and bounding box regression into different branches. This decoupling allows the model to optimize each task independently, leading to more precise and reliable predictions. For bounding box regression, the model predicts the coordinates of the boxes surrounding detected objects using a loss function like Complete Intersection over Union (CIoU) to measure the accuracy of these predictions

For the classification task, the model assigns probability scores to each detected object, indicating the likelihood that the object belongs to a particular class, such as "fire". The classification loss is typically calculated using binary cross-entropy, which measures the difference between the predicted probability and the true label.

Additionally, YOLOv8 transitions from an anchor-based to an anchor-free approach, which simplifies the network architecture by eliminating the need for predefined anchor boxes. Instead, the model directly regresses bounding box coordinates without relying on these anchors, making the detection process more flexible and adaptable to various object sizes and shapes. This anchor-free design is particularly advantageous in wildfire detection, where the objects (e.g., flames) do not conform to fixed aspect ratios, and the bounding boxes must be dynamically adjusted to fit the objects' unique shapes and sizes.

**2.2.2.1 YOLOv8m**

YOLOv8m, or YOLOv8 Medium, is a variant of the YOLOv8 object detection algorithm designed to balance performance and computational efficiency. It achieves this balance by utilizing a backbone with a depth and width multiplier that is greater than that of YOLOv8s but less than YOLOv8l or YOLOv8x. This configuration allows YOLOv8m to process images faster than larger models while still maintaining a relatively high detection accuracy. This makes YOLOv8m a flexible option for applications needing real-time object detection with moderate hardware demands.

YOLOv8m Depth Multiplier=0.67 YOLOv8m Width Multiplier=0.75

**2.2.2.2 YOLOv8n**

YOLOv8n, the nano version of the YOLOv8 series, is the most lightweight model, specifically designed for ultra-low-power devices and applications where computational resources are extremely limited.YOLOv8n uses the smallest depth and width multipliers in the series, which significantly reduces the number of layers and parameters in the network. This variant is ideal for edge computing scenarios, where the model must run efficiently on devices like IoT sensors or small drones, often with minimal hardware capabilities such as in remote wildfire detection systems deployed in forests or other hard-to-reach areas.

YOLOv8n Depth Multiplier=0.25 YOLOv8n Width Multiplier=0.25

**2.2.2.2 YOLOv8s**

YOLOv8s, or YOLOv8 Small, is a variant of the YOLOv8 object detection algorithm that offers a balance between performance and resource consumption. It is optimized for speed, making it ideal for real-time applications where latency is a critical factor. YOLOv8s achieves this by using a smaller depth and width multiplier compared to YOLOv8m, which reduces the overall complexity of the model. The architecture of YOLOv8s includes fewer layers and narrower channels in both the backbone and neck, which leads to faster inference times and process images at a high frame rate, making it suitable for applications such as real-time surveillance and mobile-based wildfire detection systems.

YOLOv8s Depth Multiplier=0.33 YOLOv8s Width Multiplier=0.50

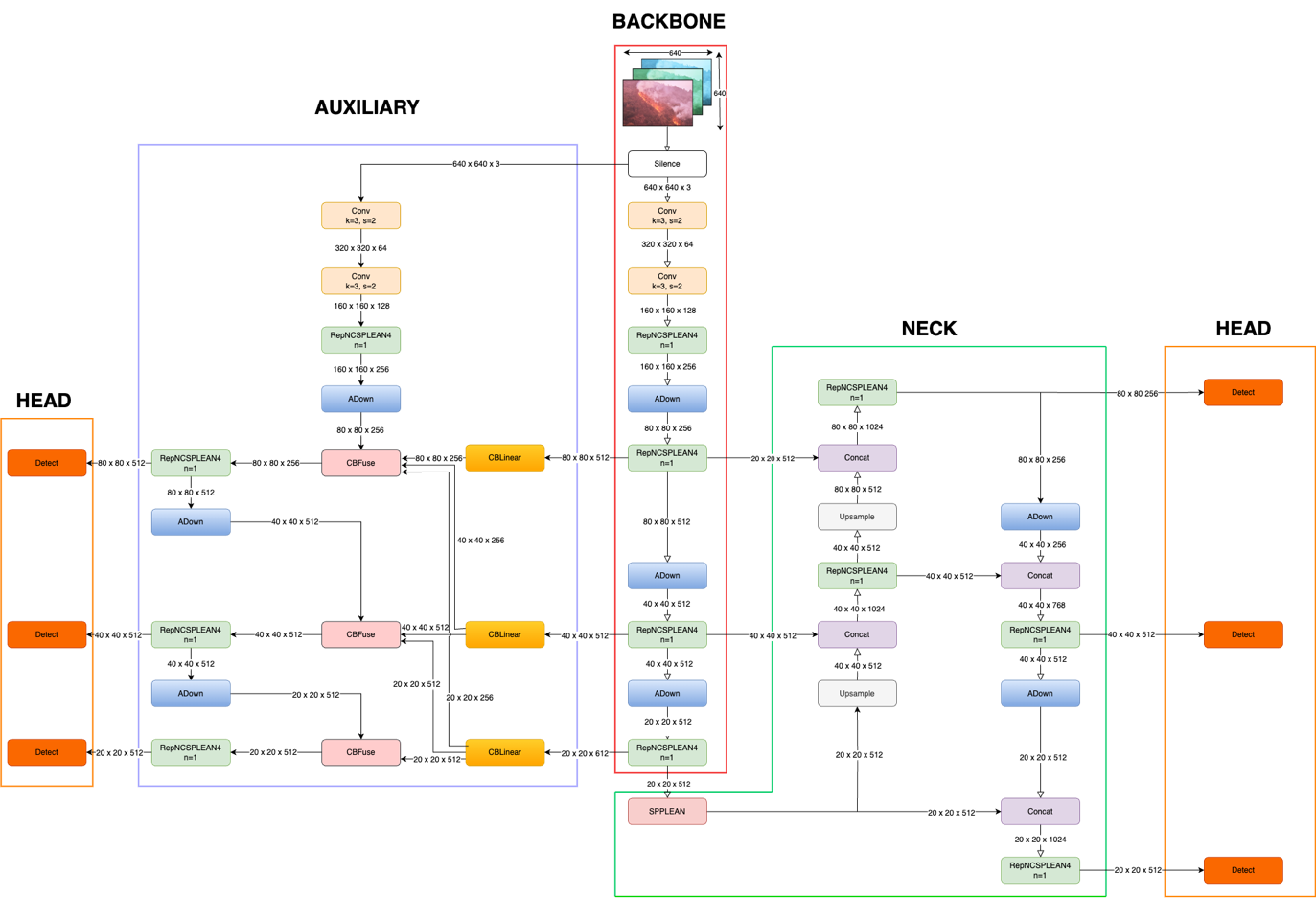
**2.2.3 YOLOv9**

Figure 6 YOLOv9c Architecture

The YOLOv9 architecture introduces several advancements over its predecessor YOLOv8, continues to incorporate the single-stage object detection paradigm, but with enhancements across the backbone, neck, and head components, by addressing the challenges of information loss through novel techniques such as Programmable Gradient Information (PGI) and the Generalized Efficient Layer Aggregation Network (GELAN) making it both more efficient and accurate in processing complex visual data. These innovations are built on the foundation of the Information Bottleneck Principle and the strategic use of Reversible Functions, ensuring enhanced efficiency and accuracy.

The input image undergoes pre-processing steps, including resizing and normalization, to ensure consistency and improve generalization across diverse input types. YOLOv9 then utilizes RepNcSPELAN4 modules for feature extraction, which split feature maps and process them through deeper bottleneck blocks before concatenating the outputs. This step enhances feature map richness and depth, crucial for complex detection tasks like forest fires.

Next, the CBLinear Fusion component aggregates features from different layers, preserving spatial details across scales. This is achieved through a linear fusion mechanism, which improves the retention of fine details and enhances the model's ability to detect small, irregular objects. In comparison to YOLOv8's PAN-FPN structure, YOLOv9's fusion mechanism is more effective.

The object detection stage in YOLOv9 involves an Advanced Detect Layer that separates classification and regression tasks for specialized processing. The bounding box regression employs an Adaptive Intersection over Union (AIoU) loss function, which provides better shape adaptation for irregular objects. The classification task uses Focal Loss, which focuses on hard-to-detect classes, ensuring that YOLOv9 maintains high accuracy even in challenging scenarios.

YOLOv9's architecture also includes an Auxiliary Network that provides additional pre-processed features to support the main detection pipeline. This auxiliary network enhances the robustness and accuracy of the final detections, further improving the model's performance.

At the core of YOLOv9's design are the Information Bottleneck Principle and Reversible Functions. The Information Bottleneck Principle addresses the issue of information loss as data passes through successive layers of the network. YOLOv9 counters this challenge by implementing PGI, which aids in preserving essential data across the network's depth, ensuring reliable gradient generation and better model convergence. Reversible Functions are integrated into the architecture to retain a complete information flow, enabling more accurate updates to the model's parameters and mitigating the risk of information degradation, particularly in deeper layers.

GELAN further enhances YOLOv9's architecture by enabling superior parameter utilization and computational efficiency. Its design allows for the flexible integration of various computational blocks, making YOLOv9 adaptable to a wide range of applications without sacrificing speed or accuracy. This strategic architectural advancement ensures that even lightweight models, which are often prone to information loss, can maintain essential data required for accurate object detection.

**2.2.3.1 YOLOv9m**

YOLOv9m is designed to offer a middle ground between the lighter YOLOv9s and the more computationally demanding YOLOv9c. Operating at a resolution of 640 pixels, it achieves a mAPval (50-95) of 51.4% and a mAPval (50) of 68.1% on COCO dataset, demonstrating its capability to detect objects with higher precision. The model contains 20.1 million parameters and demands 76.8 billion FLOPs, making it suitable for environments where computational resources are more abundant, such as desktop GPUs or high-performance embedded platforms. YOLOv9m provides a significant boost in accuracy over YOLOv9s, making it ideal for applications where detection quality is a priority, yet the computational load remains manageable.

**2.2.3.2 YOLOv9s**

The YOLOv9s model is tailored for scenarios where a balance between speed and accuracy is essential. With a resolution of 640 pixels, it achieves a mAPval (50-95) of 46.8% and a mAPval (50) of 63.4% on COCO dataset, making it a versatile choice for applications that demand quick processing without significant sacrifices in detection accuracy. The model contains 7.2 million parameters and requires 26.7 billion FLOPs, which makes it more computationally intensive than YOLOv9t but still feasible for deployment on moderately powerful hardware, such as embedded systems or mobile devices. Compared to its predecessor, YOLOv8s, YOLOv9s benefits from architectural enhancements that contribute to improved accuracy while maintaining a similar level of computational efficiency.

**2.2.3.2 YOLOv9c**

YOLOv9c represents the core model of the YOLOv9 series, designed for applications where detection accuracy is paramount, and the computational capacity to support it is available. With a resolution of 640 pixels, YOLOv9c achieves a mAPval (50-95) of 53.0% and a mAPval (50) of 70.2% on COCO dataset, reflecting its superior detection capabilities. The model includes 25.5 million parameters and requires 102.8 billion FLOPs, positioning it as a solution for high-performance tasks where accuracy cannot be compromised. YOLOv9c is particularly well-suited for complex detection tasks in fields such as autonomous driving, surveillance, or any application where precision in object detection is critical.

**2.2.4 YOLOv10**

YOLOv10 represents a significant advancement in the field of real-time object detection, building upon the strengths of its predecessors while addressing their limitations. The key innovation of YOLOv10 lies in its ability to perform object detection without relying on traditional Non-Maximum Suppression (NMS). This is achieved through a novel dual-head prediction system that streamlines both training and inference processes.

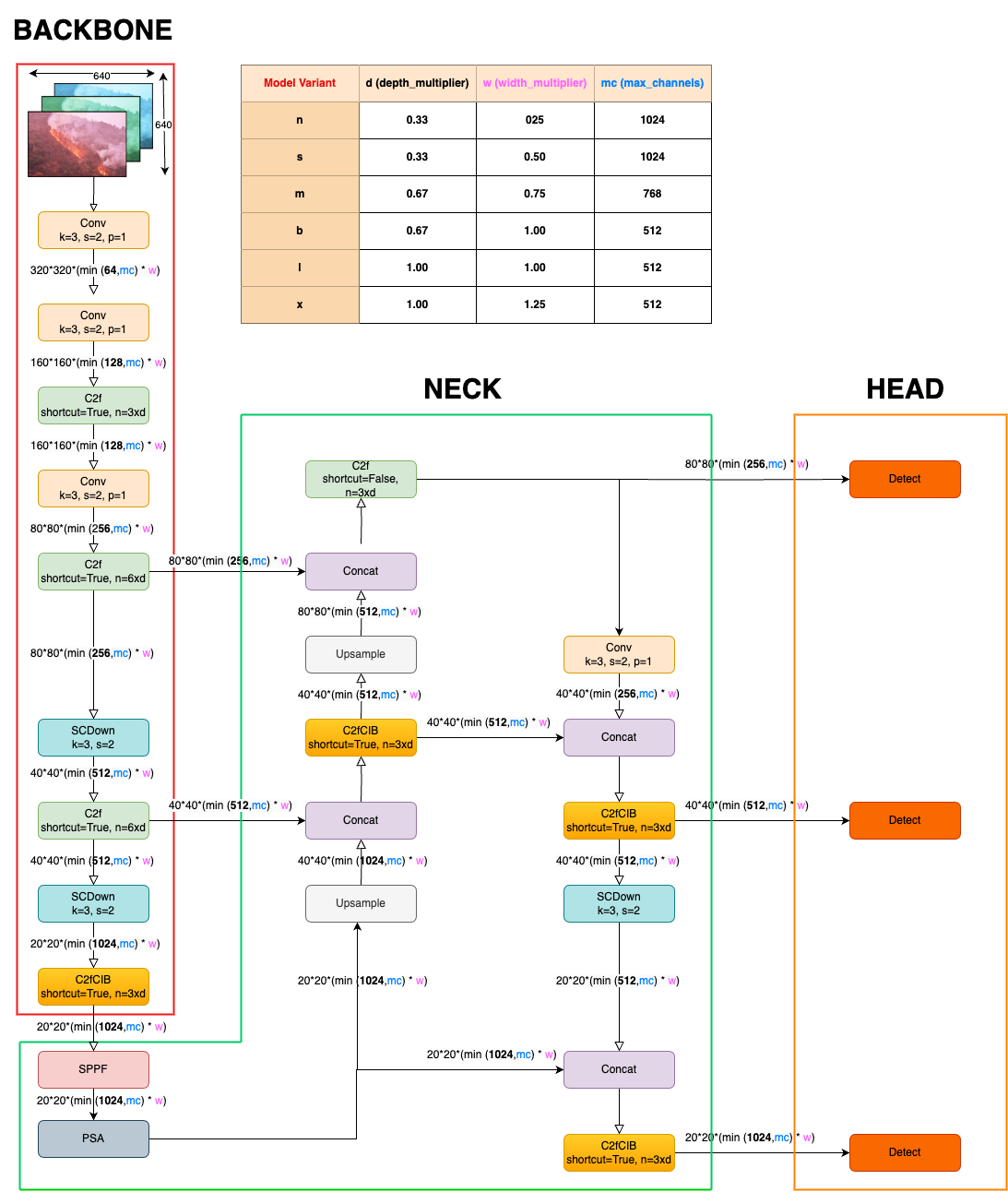
The architecture of YOLOv10 incorporates an optimized Cross Stage Partial Network (CSPNet) backbone, which enhances gradient flow and minimizes redundant computations during feature extraction. This backbone is crucial for generating high-quality feature maps that are then processed by the network. The feature maps are refined through the Path Aggregation Network (PAN) layers in the neck of the architecture, which aggregates features from multiple scales. This aggregation ensures that both local and global features are preserved, leading to more accurate object detection.

Figure 7 YOLO v10 Architecture

In YOLOv10, the prediction stage is designed to operate differently during training and inference. During training, the model employs a One-to-Many Head approach, where multiple predictions are generated for each object. This approach allows the model to receive comprehensive supervision and improves learning accuracy by providing multiple bounding box predictions per object. The One-to-Many Head processes these predictions to compute a combined loss function, which includes metrics for classification, localization, and consistency.

Conversely, during inference, YOLOv10 utilizes a One-to-One Head, which generates a single, most confident bounding box prediction per object. This transition from the One-to-Many Head to the One-to-One Head during inference eliminates the need for NMS, resulting in reduced computational overhead and faster inference times. The model's design ensures that only the most accurate predictions are considered, which enhances both speed and efficiency.

The image processing pipeline in YOLOv10 starts with the input image being passed through the CSPNet backbone, where initial feature maps are created. These feature maps are then processed by the PAN layers, which combine features from different scales to capture detailed information. The fused features are forwarded to the dual-head prediction system, where the One-to-Many Head generates multiple bounding box predictions for each object during training. These predictions are used to calculate the loss and refine the model's learning.

During the inference phase, the model relies solely on the One-to-One Head to produce the final predictions. This head simplifies the detection process by directly providing the most confident bounding boxes without additional post-processing steps like NMS. The reduction in post-processing requirements contributes to the model's efficiency, making YOLOv10 well-suited for real-time applications where quick response times are essential.

YOLOv10's design also includes several efficiency-oriented optimizations. The classification head uses depthwise separable convolutions to reduce computational costs while maintaining performance. Additionally, the decoupling of spatial downsampling and channel transformation operations helps preserve more information during the downsampling process, which improves detection accuracy. Advanced techniques such as large-kernel convolutions and partial self-attention modules further enhance the model's ability to capture broader contextual information and improve global representation learning.

**2.2.4.1 YOLOv10m**

YOLOv10 Medium, targets higher performance and accuracy for demanding tasks. It features an advanced CSPNet backbone with increased depth and width, allowing for comprehensive feature extraction. The PAN in YOLOv10m is more elaborate, with additional layers that capture fine-grained multi-scale features. The dual-head prediction mechanism is further enhanced with a larger number of anchor boxes and prediction channels, contributing to improved detection accuracy. YOLOv10m demonstrates a high mAP of 52.1 on the COCO dataset and achieves a latency of approximately 60 ms per frame at 640x640 pixels. This makes YOLOv10m suitable for high-resolution video processing and complex object detection tasks where precision is critical.

**2.2.4.2 YOLOv10s**

YOLOv10 Small, provides a balanced approach between efficiency and accuracy. It utilizes a moderately scaled CSPNet backbone, increasing the depth and width compared to YOLOv10n to capture more detailed features. The PAN in YOLOv10s includes additional aggregation layers, optimizing multi-scale feature integration while maintaining efficiency. The model benefits from an enhanced dual-head prediction mechanism with more anchor boxes, improving detection precision without a significant increase in computational cost. YOLOv10s achieves a mAP of 48.3 on the COCO dataset, with a latency of around 45 ms per frame at 640x640 pixels. This balance makes it suitable for applications requiring both accuracy and real-time performance.

**2.2.4.3 YOLOv10n**

YOLOv10 Nano, is designed to operate in environments where computational resources are limited. It employs a compact CSPNet backbone with reduced convolutional layers and parameters to ensure lightweight operation. The model features a simplified PAN to balance feature integration with minimal computational load. Depthwise separable convolutions are used extensively to enhance computational efficiency, while large-kernel convolutions improve feature extraction with minimal resource consumption. YOLOv10n delivers a remarkable performance of 41.2 mAP (mean Average Precision) with a latency of approximately 23 ms per frame at a resolution of 640x640 pixels, making it ideal for real-time applications on mobile and embedded devices.