A Review on YOLOv8 and Its Advancements

3 authors, including:

Venkata Rami Reddy Ch
VIT-AP University, Amaravati
34 PUBLICATIONS   **401** CITATIONS

SEE PROFILE

A Review on YOLOv8 and its Advancements

[*]Mupparaju Sohan[1], Thotakura SaiRam[2], and Ch. Venkata RamiReddy[3]

[1,2,3]School of Computer Science and Engineering, VIT-AP University, Amaravati, India, 522237

[1*]sohanmupparaju@gmail.com
[2]sairam.thotakura2003@gmail.com
[3]chvrr58@gmail.com

* Corresponding Author

Abstract. Identifying objects is a crucial task in computer vision that finds its application in several fields like robotics, medical imaging, surveillance systems, and autonomous vehicles. The newest version of the YOLO model, YOLOv8, which is an advanced real-time object detection framework, has attracted the attention of the research community. Of all the popular object recognition machine learning models such as Faster R-CNN, SSD, and RetinaNet, YOLO is the most popular in terms of accuracy, speed, and efficiency. This review article provides an analysis of YOLO v8, highlighting its innovative features, improvements, applicability in different environments and performance metrics in comparison to other versions and models.

Keywords: YOLO, Object detection, Image classification, Instance segmentation, Computer vision, Review.

1 Introduction

Object detection is a fundamental objective in computer vision that involves identifying and classifying objects present in images or videos [1]. It has numerous practical applications across various domains such as robotics, autonomous vehicles, surveillance, and augmented reality [2]. It is now widely used in industries such as transportation, mining, and construction, which significantly improved safety measures [3]. One such application uses computer vision algorithms to detect if employees are in potentially dangerous situations by identifying whether employees are donning safety equipment, such as helmets, which can assist assure safety in dangerous areas [3]. It can also be utilized in autonomous vehicles to find pedestrians and other cars on the road [4]. Object detection in robotics can be used to locate and recognise items for manipulation or interaction [5]. Over time, various techniques have been developed to tackle the challenge of object detection, ranging from traditional machine-learning approaches to newer deep-learning models.

Initially, object detection was approached as a pipeline consisting of three main

steps: proposal generation, feature extraction, and region classification. Traditional machine learning techniques [6-8], including support vector machines (SVM), were popular due to their success on small training datasets. However, these methods had limited effectiveness, and detection accuracy improvements were marginal.

The emergence of deep learning brought about a significant change in object detection, with deep convolutional neural networks (CNNs) playing a crucial role in transforming the field of computer vision and enabling the development of more precise and resilient object detection methods [9]. Deep neural networks can generate hierarchical features, capture different scale information in different layers, and produce robust and discriminative features for classification.

This review article will discuss the most recent YOLO model YOLOv8, its development and implications in object detection along with the speed and accuracy that have emerged throughout the framework's development.

1.1   Existing Object Detection Models

Currently, deep learning-based object detection frameworks can be broadly classified into two families: two-stage detectors and one-stage detectors. Two-stage detectors, such as Region-based CNN (R-CNN) and its variants, first generate object proposals and then classify each proposal. One-stage detectors, such as You Only Look Once (YOLO) and its variants, directly predict the presence and location of objects in a single step.

RCNN (Region-based Convolutional Neural Network). Proposed by Ross Girshick et al. in 2014, is a two-stage object detection model [10]. It first generates region proposals using a selective search algorithm and then extracts features from these regions using a CNN. Finally, the extracted features are fed into an SVM for object classification. The main limitation of RCNN is its slow training and inference speed due to its two-stage approach and the selective search algorithm.

SPPNet (Spatial Pyramid Pooling Network). Proposed by Kaiming He et al. in 2014 [11], is an improvement over RCNN that addresses the slow speed issue. It introduces spatial pyramid pooling (SPP) to enable the network to take inputs of arbitrary sizes and output fixed-length feature vectors, thus eliminating the need for cropping or warping the input image. However, it still relies on a selective search algorithm for region proposals, which limits its performance.

Fast R-CNN. Proposed by Ross Girshick in 2015 [12], is an improvement over RCNN and SPPNet that eliminates the separate feature extraction step by introducing an RoI pooling layer that shares the feature map across all RoIs (region of interests). This significantly reduces the computation time and improves accuracy compared to RCNN and SPPNet.

Faster R-CNN. proposed by Shaoqing Ren et al. in 2015 [13], is a further improvement over Fast RCNN that replaces the selective search algorithm with a Region Proposal Network (RPN) to generate region proposals in a single forward pass. This leads to a significant reduction in computation time and improves the accuracy of object detection.

FPN (Feature Pyramid Networks). Proposed by Tsung-Yi Lin et al. in 2017 [14], is an extension of Faster RCNN that addresses the issue of detecting objects at different scales. It introduces a top-down pathway and lateral connections that combine features at different levels of a CNN to form a feature pyramid. This enables the network to detect objects at different scales and improves the accuracy of object detection.

YOLOv1. Proposed by Joseph Redmon et al. in 2015 [15], is a one-stage object detection model that uses a single convolutional neural network to predict object classes and bounding boxes directly from full images. It divides the image into a grid of cells and predicts multiple bounding boxes and object classes for each cell. The main limitation of YOLOv1 is its poor performance on small objects.

SSD (Single Shot MultiBox Detector). Proposed by Wei Liu et al. in 2016 [16], is also a one-stage object detection model that uses a similar approach to YOLOv1. However, it introduces additional convolutional layers to predict object categories and offsets for default boxes of different scales and aspect ratios. This enables it to better handle objects at different scales and aspect ratios.

YOLOv2. Proposed by Joseph Redmon and Ali Farhadi in 2017 [17], is an improvement over YOLOv1 that addresses its limitations. It introduces anchor boxes and batch normalization to improve the accuracy of object detection. Anchor boxes are used to better handle objects at different scales and aspect ratios, while batch normalization improves the stability of the network during training.

YOLOv3. Proposed by Joseph Redmon and Ali Farhadi in 2018 [18], is a further improvement over YOLOv2 that introduces a number of changes to improve accuracy and speed. It uses a feature pyramid network and predicts object categories and bounding boxes at three different scales to better handle objects of different sizes. It also introduces new techniques such as multi-scale training and dynamic anchor assignment to improve the accuracy of object detection.

YOLOv4. proposed by Alexey Bochkovskiy et al. in 2020 [19], is a significant improvement over YOLOv3 that introduces a number of new techniques to improve accuracy and speed. It uses a CSPDarknet backbone and introduces new techniques such as spatial attention, Mish activation function, and GIoU loss to improve accuracy. It also introduces new training techniques such as the self-attention mechanism and Mosaic data augmentation to improve the robustness of the network.

YOLOv5. proposed by Glenn Jocher et al. in 2020 [20], is another significant improvement over YOLOv3 that introduces a new architecture and new techniques to improve accuracy and speed. It uses a novel backbone architecture and introduces new techniques such as focal loss, label smoothing, and auto-augmentation to improve accuracy. It also introduces new training techniques such as CutMix data augmentation and learning rate schedulers to improve the convergence rate of the network.

YOLOv6. Developed by the Meituan researchers in 2022 [21], is an object detection model designed primarily for industrial applications. Its hardware-efficient design and improved performance surpass that of YOLOv5 in terms of both detection accuracy and inference speed. It uses a EfficientRep backbone and techniques like an Anchor-free paradigm, SimOTA tag assignment, and SIoU box regression loss to improve speed and accuracy.

YOLOv7. Proposed by Chien-Yao Wang and Alexey Bochkovskiy in 2022 [22], is an improvement over Scaled-YOLOv4 and YOLOR which are based on YOLOv4. It introduces architectural reforms such as E-ELAN and techniques such as Model Scaling Techniques, Re-parameterization Planning and Coarse-to-fine auxiliary head supervision to improve its efficiency.

1.2   Overview of YOLOv8

YOLOv8 is the latest version [23-24] of the YOLO (You Only Look Once) models. The YOLO models are popular for their accuracy and compact size. It is a state-of-the-art model that could be trained on any powerful or low-end hardware. Alternatively, they can also be trained and deployed on the cloud. The first YOLO model was introduced in a C repository called Darknet in 2015 by Joseph Redmond [15] when he was working on it as PHD at the University of Washington. It has since been developed by the community for subsequent versions.
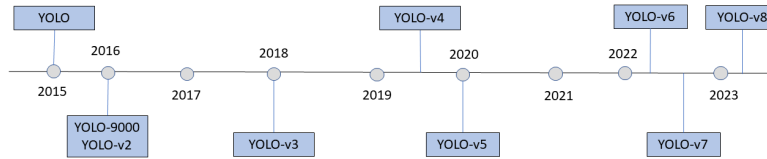

Fig. 1. Timeline of YOLO Advancements

YOLOv8 is developed by Ultralytics, a team known for its innovative YOLOv5 model [20]. It was introduced on January 10th, 2023. YOLOv8 is used to detect objects in images, classify images, and distinguish objects from each other. Ultralytics has made numerous enhancements to YOLOv8, making it better

and more user-friendly than YOLOv5. It is an advanced model that improves upon the success of YOLOv5 by incorporating modifications that enhance its power and user-friendliness in various computer vision tasks. These enhancements include a modified backbone network, an anchor-free detection head, and a new loss function. Furthermore, it provides built-in support for image classification tasks. YOLOv8 is distinctive in that it delivers unmatched speed and accuracy performance while maintaining a streamlined design that makes it suitable for different applications and easy to adapt to various hardware platforms.

2   Architecture of YOLOv8

As of the current writing, there is no published paper yet on YOLOv8, so detailed insights into the research techniques and ablation studies conducted during its development are unavailable. However, an analysis of the YOLOv8 repository [24] and its documentation [23] over its predecessor YOLOv5 [20], reveals several key features and architectural improvements.

2.1   Architecture components

The YOLOv8 architecture is composed of two major parts, namely the backbone and the head, both of which use a fully convolutional neural network.
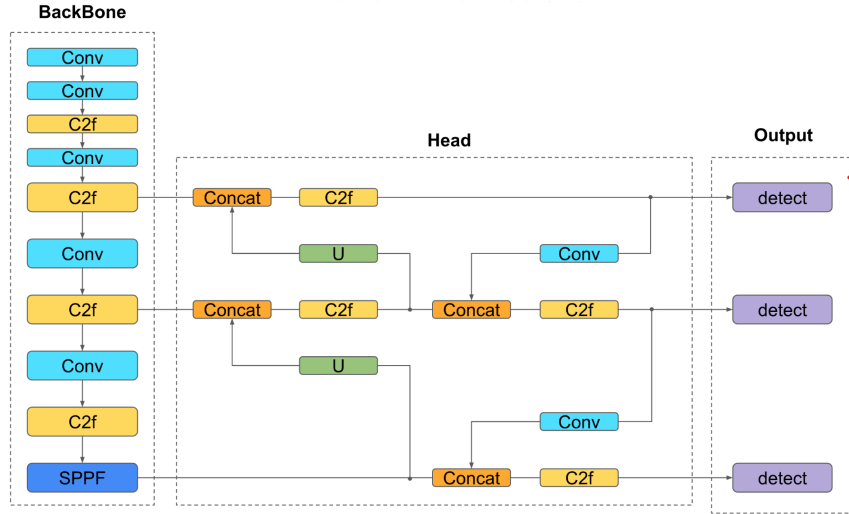

Fig. 2. YOLOv8 Architecture visualization, Arrows represent data flow between layers

Backbone. YOLOv8 features a new backbone network which is a modified version of the CSPDarknet53 architecture [26] which consists of 53 convolutional layers and employs a technique called cross-stage partial connections to enhance

the transmission of information across the various levels of the network.

This Backbone of YOLOv8 consists of multiple convolutional layers organized in a sequential manner that extract relevant features from the input image. The new C2f module integrates high-level features with contextual information to enhance detection accuracy. The SPPF [11] (spatial pyramid pooling faster) module, and the other following convolution layers, process features at various scales. [25]

Head. The head then takes the feature maps produced by the Backbone and further processes them to provide the model's final output in the form of bounding boxes and object classes. In YOLOv8, the head is created to be detachable, which implies that it manages objectness scores, classification, and regression duties in an independent manner. This approach allows each branch to focus on its own task while improving the model's overall accuracy. The U layers (Upsample layers) in Fig. 2. increase the resolution of the feature maps. [25] The head uses a sequence of convolutional layers to analyse the feature maps, followed by a linear layer for predicting the bounding boxes and class probabilities. The head's design is optimised for speed and accuracy, with special consideration given to the number of channels and kernel sizes of each layer to maximise performance.

Finally, the Detection module uses a set of convolution and linear layers to map the high-dimensional features to the output bounding boxes and object classes. The entire structure is designed to be quick and effective, yet it maintains high precision in object detection.

2.2 Architectural Advancements

Anchor-Free Detection. Similar to YOLOv6 and YOLOv7, YOLOv8 is a model that does not rely on anchors. This means that it predicts the centre of an object directly rather than the offset from a known anchor box. Anchor boxes were a well-known challenging aspect of early YOLO models (YOLOv5 and earlier) since these could represent the target benchmark's box distribution but not the distribution of the custom dataset. The use of anchor-free detection minimises the number of box predictions, which speeds up Non-Maximum Suppression (NMS), a complex post-processing phase that sifts through candidate detections following inference. [27]

New Convolution layer. The convolutional (conv) layers in YOLO architecture are responsible for detecting features in input images using learnable filters. These layers detect features at different scales and resolutions, allowing the network to detect objects of varying sizes and shapes. The output of these layers is then passed through other layers to generate bounding boxes and class predictions for each object detected in the image.

Unlike YOLOv5, YOLOv8 uses a different convolution layer called C2f. This

new layer replaces the C3 layer of YOLOv5. The C2f layer in YOLOv8 concatenates the outputs of all the Bottleneck layers, while in the C3 layer of YOLOv5, only the output of the last Bottleneck layer is utilized.
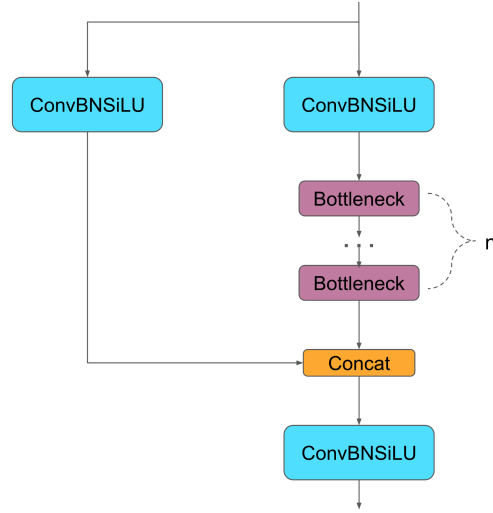

Fig. 3. C3 Module of YOLOv5, the number of bottleneck layers are n. ConvBNSiLU is a block composed of a Conv, a BatchNorm and a SiLU layer.
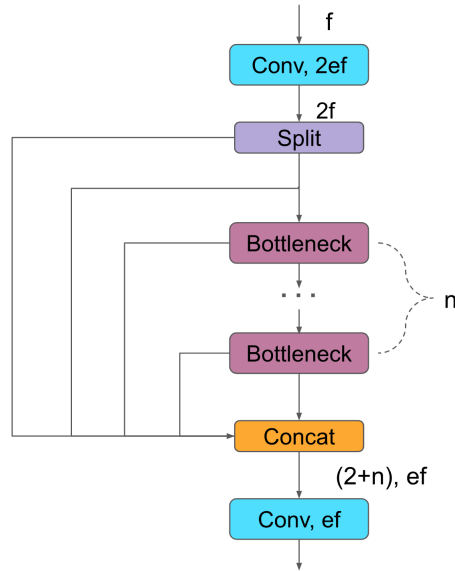

Fig. 4. C2f Module of YOLOv8, Conv is a block composed of a Conv2d, a BatchNorm and a SiLU layer.

The bottleneck in this system is similar to that of YOLOv5 but with a change in the first convolution layer. The kernel size has been increased from 1x1 to 3x3, which is similar to the ResNet block described in 2015. In the neck of the system, the features are concatenated without requiring them to have the same channel dimensions, which reduces the number of parameters and the total size of the tensors.

3 Training and Inference

3.1 Downloadable Python Package via pip

YOLOv8 can now be installed through a PIP package, making it easy for users to install and manage YOLOv5 for training and inference. This simplifies the installation process and allows for easy updates and compatibility with other Python libraries. Users can simply use the pip package manager to install YOLOv8 and start using it for their computer vision tasks, further increasing the accessibility and usability of the model.

YOLOv8 can also be installed from the source on GitHub.

3.2 Command Line Interface (CLI)

One of the most useful features of YOLOv8 is its ultralytics package distributed with a CLI. It supports easy single-line commands without requiring a Python environment. CLI does not require any customisation or Python code. The CLI provides options for specifying dataset paths, model architecture, training parameters, and output directories. This allows users to easily customize the training process according to their specific requirements. Compared to previous versions of YOLO, the CLI in YOLOv8 offers additional options for fine-tuning, model evaluation, and distributed training, providing enhanced flexibility and control over the training process.

3.3 YOLOv8 Python SDK

The YOLOv8 model also comes with a Pythonic Model and Trainer interface, making it easier to integrate the YOLO model into custom Python scripts with just a few lines of code. This enables users to leverage the power of YOLOv8 for object detection, image classification, and instance segmentation tasks with minimal effort. This streamlined integration process is a significant advancement, as it eliminates the need for complex configurations or lengthy setup procedures, making YOLOv8 more accessible and convenient for developers to use in their Python-based projects.

3.4 YOLOv8 Tasks and Modes

The YOLOv8 framework can be used to perform computer vision tasks such as detection, segmentation, classification, and pose estimation. It comes with pre-trained models for each task. The pretrained models for detection, Segmentation and Pose are pretrained on the COCO dataset [25-26], while Classification models are pretrained on the ImageNet dataset.

YOLOv8 introduces scaled versions such as YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra big). These several versions provide variable model sizes and capabilities, catering to various requirements and use scenarios.

For Segmentation, Classification and Pose estimation; these various scaled versions use suffixes such as -seg, -cls and -pose respectively. These tasks don't require additional commands and scripts for making masks, contours or for classifying the images.

With a well-labelled and sufficient dataset, the accuracy can be high. Also using a GPU over a CPU is recommended for the training process to further enhance the performance by decreasing computation time.

YOLOv8 offers multiple modes that can be used either through a command line interface (CLI) or through Python scripting, allowing users to perform different tasks based on their specific needs and requirements. These modes are

Train. This mode is used to train a custom model on a dataset with specified hyperparameters. During the training process, YOLOv8 employs adaptive techniques to optimize the learning rate and balance the loss function. This leads to enhanced model performance.

Val. This mode is used to evaluate a trained model on a validation set to measure its accuracy and generalization performance. This mode can help in tuning the hyperparameters of the model for improved performance.

Predict. This mode is used to make predictions using a trained model on new images or videos. The model is loaded from a checkpoint file, and users can input images or videos for inference. The model predicts object classes and locations in the input file.

Export. This mode is used to convert a trained model to a format suitable for deployment in other software applications or hardware devices. This mode is useful for deploying the model in production environments. Commonly used YOLOv8 export formats are PyTorch, TorchScript, TensorRT, CoreML, and PaddlePaddle.

Track. This mode is used to perform real-time object tracking in live video streams. The model is loaded from a checkpoint file and can be used for applications like surveillance systems or self-driving cars.

Benchmark. This mode is used to profile the performance of different export formats in terms of speed and accuracy. It provides information on the size of the exported format, mAP50-95 metrics for object detection, segmentation, and pose, or accuracy_top5 metrics for classification, as well as inference time per image. This enables users to select the most suitable export format for their particular use case, considering their requirements for speed and accuracy.

3.5 User Experience (UX) Enhancements

YOLOv8 improves on prior versions by introducing new modules such as spatial attention, feature fusion, and context aggregation.

Each YOLO task has its own Trainers, Validators and Predictors which can be customized to support user custom tasks or research and development ideas. Callbacks are utilized as points of entry at strategic stages during the train, val, export, and predict modes in YOLOv8. These callbacks accept an object of either a trainer, validator, or predictor, depending on the type of operation being performed.

The model's performance, speed, and accuracy are heavily influenced by YOLO settings, hyperparameters, and augmentation which can also influence model behaviour at various stages of model development, such as training, validation, and prediction. These can be configured by using either CLI or Python scripts.

4 Performance Evaluation

4.1 Object Detection Metrics

mAPval. mAPval stands for Mean Average Precision on the validation set. It is a popular metric used to evaluate the accuracy of an object detection model. Average precision (AP) is calculated for each class in the validation set, and then the mean is taken across all classes to obtain the mAPval score. A higher mAPval score indicates better accuracy of the object detection model in detecting objects of different classes in the validation set.

Speed CPU ONNX. This relates to the object identification model's speed while running on a CPU (Central Processing Unit) using the ONNX (Open Neural Network Exchange) runtime. ONNX is a prominent deep learning model representation format, and model speed can be quantified in terms of inference time or frames per second (FPS). Higher values for Speed CPU ONNX indicate faster inference times on a CPU, which can be important for real-time or near-real-time applications.

Speed A100 TensorRT. This refers to the speed of the object detection model when running on an A100 GPU (Graphics Processing Unit) using TensorRT, which is an optimization library developed by NVIDIA for deep learning inference. Similar to Speed CPU ONNX, the speed can be measured in terms of inference time or frames per second (FPS). Higher values for Speed A100 TensorRT indicate faster inference times on a powerful GPU, which can be beneficial for applications that require high throughput or real-time processing.

Latency A100 TensorRT FP16 (ms/img). This refers to the latency or inference time of the object detection model when running on an NVIDIA A100 GPU with TensorRT optimization, using the FP16 (half-precision floating point) data type. It indicates how much time the model takes to process a single image, typically measured in milliseconds per image (ms/img). Lower values indicate faster inference times, which are desirable for real-time or low-latency applications.

Params (M). Params (M) refers to the number of model parameters in millions. It represents the size of the model, and generally larger models tend to have more capacity for learning complex patterns but may also require more computational resources for training and inference.

FLOPs (B). FLOPs (B) stands for Floating point operations per second in billions. It is a measure of the computational complexity of the model, indicating the number of floating-point operations the model performs per second during inference. Lower FLOPs (B) values indicate less computational complexity and can be desirable for resource-constrained environments, while higher values indicate more computational complexity and may require more powerful hardware for efficient inference.

4.2 Benchmark Datasets and Computational Efficiency

Performance of YOLOv8 on COCO. The COCO val2017 dataset [28-29] is a commonly used benchmark dataset for evaluating object detection models. It consists of a large collection of more than 5000 diverse images with 80 object categories, and it provides annotations for object instances, object categories, and other relevant information. The dataset is an Industry-Standard benchmark for object detection performance and for comparing the accuracy and speed of different object detection models.

Table 1. Performance of YOLOv8 Pretrained Models for Detection. Sourced from the Ultralytics GitHub Repository (https://github.com/ultralytics/ultralytics)

Model	size (pixels)	mAPval 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

The training and results were sourced from Ultralytics GitHub repository [24][1]. All scaled versions of YOLOv8 along with previous versions of YOLO i.e. YOLOv5, YOLOv7 were trained on COCO. Here mAPval values are for single-model single-scale on the COCO val2017 dataset and Speed is averaged over COCO val images using an Amazon EC2 P4d instance.

---

[1] Ulatralytics GitHub repository https://github.com/ultralytics/ultralytics
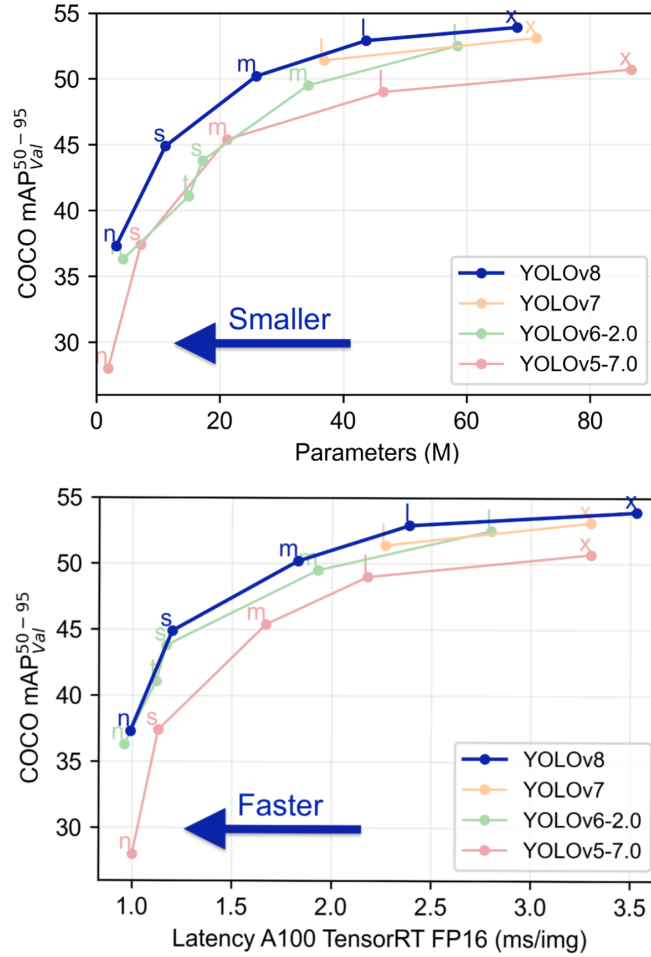
Fig. 5. Performance comparison of all scaled versions of YOLOv8, YOLOv7, YOLOv6 and YOLOv5 (Source: GitHub repository "ultralytics/ultralytics" by Ultralytics, `https://github.com/ultralytics/ultralytics`)

Performance of YOLOv8 on RF100. The Roboflow 100 (RF100) dataset [30-31] is a diverse, multi-domain benchmark comprising 100 datasets created by using over 90,000 public datasets and 60 million public images from the Roboflow Universe, a web application for computer vision practitioners. The dataset aims to provide a more comprehensive evaluation of object detection models by offering a wide range of real-life domains, including satellite, microscopic, and gaming images. With RF100, researchers can test their models' generalizability on semantically diverse data.

YOLOv8 is evaluated on the RF100 benchmark alongside YOLOv5 and YOLOv7. mAP@.50 is a specific version of the mAP metric that measures the average precision of a model at a detection confidence threshold of 0.5. In other words, it measures how well the model is able to detect objects when it is at least 50% confident that an object is present in the image.

The process and results were sourced from the robofow blog [32][2]. Small versions of each model are trained for a total of 100 epochs. To minimize the effect of random initialization and ensure reproducibility, each experiment is run using a single seed.
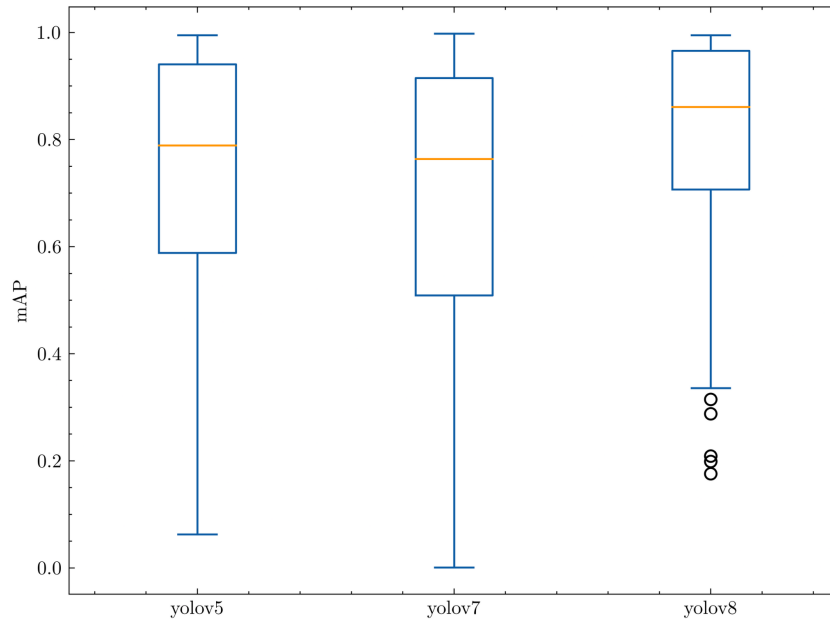

Fig. 6. Box plots show each model's mAP@.50. (Source: "What's New in YOLOv8?" by Jacob Solawetz and Francesco, published on the Roboflow blog on [March 27, 2023], https://blog.roboflow.com/whats-new-in-yolov8/)

According to the box plot in Fig. 6, YOLOv8 performed better than yolov7 and yolov5 on the Roboflow 100 benchmark in terms of mAP and had fewer outliers, indicating a more consistent performance.

The analytical results, which compare YOLOv5 with YOLOv8 performance in domain-specific tasks, show that YOLOv8 outperforms YOLOv5. Specifically, YOLOv8 obtained a mean average precision (mAP) score of 80.2% on Roboflow 100, while YOLOv5's score was 73.5%. These findings suggest that YOLOv8 is significantly more effective than YOLOv5 for domain-specific tasks.

---

[2] Roboflow blog https://blog.roboflow.com/whats-new-in-yolov8/

The bar plot in Fig. 7 shows the performance of YOLOv8, YOLOv7, and YOLOv8 evaluated on each category of the RF100 dataset. Small versions of each model are trained for a total of 100 epochs as done with the whole RF100 dataset above. From the plots in Fig. 7, it is observed that the mAP of YOLOv8 is similar to or significantly improved than its previous versions.
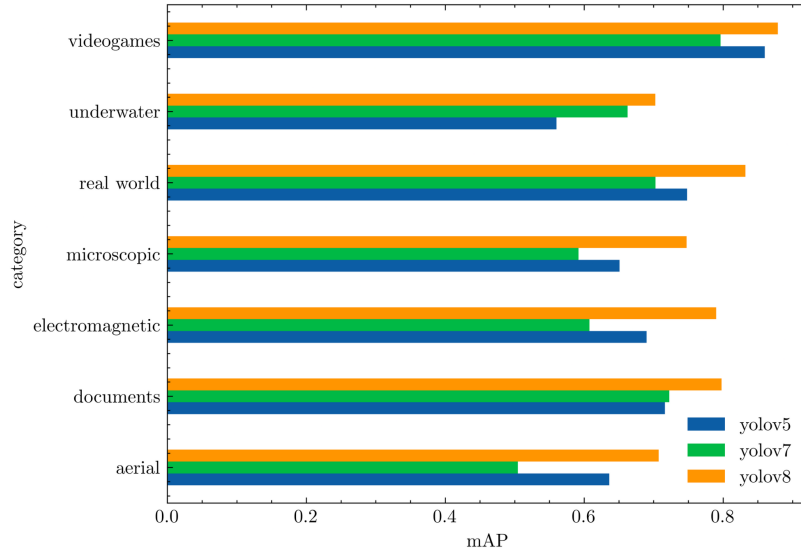

Fig. 7. Bar plot shows the average mAP@.50 for each RF100 category. (Source: "What's New in YOLOv8?" by Jacob Solawetz and Francesco, published on the Roboflow blog on [March 27, 2023], https://blog.roboflow.com/whats-new-in-yolov8/)

By comparison of the evaluation, it is observed that the YOLOv8 model outperforms YOLOv5 and YOLOv7 in each category of the dataset.

From the above performance, YOLOv8 is observed to be not only faster and more accurate than previous versions, but it also requires fewer parameters to attain its performance.

5   Applications and Use Cases

Autonomous Vehicles. YOLOv8's real-time object detection capabilities make it a powerful tool for enhancing the safety of autonomous vehicles. By accurately detecting and tracking other vehicles, pedestrians, and traffic signals, YOLOv8 can help self-driving cars navigate complex traffic scenarios with greater efficiency and safety.

Medical Imaging. YOLOv8 can be used in medical imaging to detect and classify numerous anomalies and diseases such as cancer, tumours, and fractures. It can also be utilised for surgical planning and guidance, as well as tracking medical tools in real-time during surgery. YOLOv8's improved accuracy and speed can assist medical practitioners in making faster and more accurate diagnoses, therefore improving patient outcomes.

Manufacturing. In the field of manufacturing, YOLOv8 can identify product flaws by detecting deviations in shape, size, or colour. It can also verify that the appropriate parts are being used in assembly line processes and monitor inventory levels to prevent shortages.

Security. YOLOv8 can be used for surveillance to identify individuals and objects in restricted areas, enabling the detection of intruders and unauthorized access. It can also monitor crowd movement and traffic flow in congested public spaces such as airports and train stations. Additionally, it can detect potentially threatening behaviour to aid in identifying security risks.

Sports Analysis. The sports analysis field can utilize YOLOv8 to track the movements of players, detect the location of the ball, and classify the actions of players. This data can help assess player performance, formulate game plans, and identify areas that require improvement.

Autonomous Drones. Autonomous drones can leverage YOLOv8 for detecting and tracking objects, including moving targets, to enable applications like surveillance, search and rescue, and structural or infrastructure inspection.

Agriculture. Regarding agriculture, YOLOv8 can track crop growth, detect crop diseases, and recognize pests. It can also facilitate precision agriculture by identifying areas of a field that require varying degrees of water or fertilizer. By providing faster and more precise data, YOLOv8 can support farmers in making more informed decisions, increasing crop yields, and decreasing waste.

Robotics. In the domain of robotics, YOLOv8 can assist robots in recognizing and interacting with objects in their surroundings. It can facilitate object tracking, manipulation, real-time navigation, and obstacle avoidance. By providing greater speed and precision, YOLOv8 can enable robots to undertake more intricate tasks, including warehouse automation, manufacturing, and search and rescue missions.

Environmental Monitoring. YOLOv8 can identify and categorize animals, track deforestation, and monitor pollution levels as such it can be used for environmental monitoring. This data can facilitate the

development of conservation plans, highlight environmental concerns, and assess the effects of human activity on the environment.

Traffic Management. YOLOv8 can assist in traffic management to recognize and trail vehicles, monitor traffic buildup, and manage traffic lights. These applications can contribute to decreasing the occurrence of traffic accidents, improving traffic flow, and reducing the amount of time commuters spend travelling. YOLOv8 can be instrumental in achieving these objectives.

From the above applications, it can be observed that YOLOv8 is well-suited to a wide range of computer vision applications because of its improved speed and precision. It can be utilised in both traditional applications like medical imaging and surveillance as well as emergent ones like gaming and augmented reality.

6 Conclusion

In this paper, the YOLOv8 with its architecture and its advancements along with an analysis of its performance has been portrayed on various datasets in comparison with previous models of YOLO.

The introduction of YOLO v8 is a noteworthy achievement in the progress of object detection models. YOLO's latest edition leverages the advantages of its prior versions and incorporates various novel elements that boost its efficacy and effectiveness. YOLO v8 incorporates cutting-edge techniques that have been shown to improve object detection accuracy and speed while reducing computation and memory requirements, such as the addition of attention modules and self-attention mechanisms and the use of spatial pyramid pooling and deformable convolutions.

Overall, YOLO v8 exhibits great potential as an object detection model that can enhance real-time detection capabilities. This latest version of YOLO is a notable advancement in the realm of computer vision and is likely to stimulate additional exploration and progress in this domain.

References

1. Deng, J., Xuan, X., Wang, W., et al. "A Review of Research on Object Detection Based on Deep Learning." Journal of Physics: Conference Series 1684 (2020): 012028. doi: 10.1088/1742-6596/1684/1/012028.
2. Bianchini, M., Simic, M., Ghosh, A., Shaw, R. N. In Machine Learning for Robotics Applications. Springer Verlag, Singapore, S.l., 2022.
3. Agrawal, T., Kirkpatrick, C., Imran, K., Figus, M. "Automatically Detecting Personal Protective Equipment on Persons in Images Using Amazon Rekognition." Amazon, 2020.

https://aws.amazon.com/blogs/machine-learning/automatically-detecting-personal-protective-equipment-on-persons-in-images-using-amazon-rekognition/. Accessed April 27, 2023.

4. Rasouli, A., Tsotsos, J. K. "Autonomous Vehicles That Interact with Pedestrians: A Survey of Theory and Practice." IEEE Transactions on Intelligent Transportation Systems 21 (2019): 900–918. doi: 10.1109/TITS.2019.2901817.

5. Martinez-Martin, E., del Pobil, A. P. "Object Detection and Recognition for Assistive Robots: Experimentation and Implementation." IEEE Robotics & Automation Magazine 24 (2017): 123–138. doi: 10.1109/MRA.2016.2615329.

6. Viola, P., and M. Jones. "Rapid Object Detection Using a Boosted Cascade of Simple Features." Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, n.d. https://doi.org/10.1109/cvpr.2001.990517.

7. Dalal, N., and B. Triggs. "Histograms of Oriented Gradients for Human Detection." 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), n.d. https://doi.org/10.1109/cvpr.2005.177.

8. Felzenszwalb, Pedro F., Ross B. Girshick, and David McAllester. "Cascade Object Detection with Deformable Part Models." 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010. https://doi.org/10.1109/cvpr.2010.5539906.

9. Nash, R. "An Introduction to Convolutional Neural Networks." arXiv preprint arXiv:1511.08458 (2015).

10. Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation." 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014. https://doi.org/10.1109/cvpr.2014.81.

11. He, K., Zhang, X., Ren, S., Sun, J. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition." IEEE Transactions on Pattern Analysis and Machine Intelligence 37 (2015): 1904–1916. doi: 10.1109/TPAMI.2015.2389824.

12. Girshick, Ross. "Fast R-CNN." 2015 IEEE International Conference on Computer Vision (ICCV), 2015. https://doi.org/10.1109/iccv.2015.169.

13. Ren, S., He, K., Girshick, R., Sun, J. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks." IEEE Transactions on Pattern Analysis and Machine Intelligence 39 (2017): 1137–1149. doi: 10.1109/TPAMI.2016.2577031.

14. Lin, Tsung-Yi, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature Pyramid Networks for Object Detection." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017. https://doi.org/10.1109/cvpr.2017.106.

15. Redmon, J., Divvala, S., Girshick, R., Farhadi, A. "You Only Look Once: Unified, Real-Time Object Detection." 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016. doi: 10.1109/CVPR.2016.91.

16. Liu, W., Anguelov, D., Erhan, D., et al. "SSD: Single Shot Multibox Detector." Computer Vision – ECCV 2016 (2016): 21–37. doi: 10.1007/978-3-319-46448-0_2.

17. Redmon, Joseph, and Ali Farhadi. "Yolo9000: Better, Faster, Stronger." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017. https://doi.org/10.1109/cvpr.2017.690.

18. Redmon, Joseph, and Ali Farhadi. "YOLOv3: An Incremental Improvement." arXiv preprint arXiv:1804.02767 (2018).

19. Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection." arXiv preprint arXiv:2004.10934 (2020).

20. Jocher, G. 2020. "YOLOv5 by Ultralytics (Version 7.0)." Computer software. doi: 10.5281/zenodo.3908559.

21. Li, Chuyi, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. "YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications." arXiv preprint arXiv:2209.02976 (2022).

22. Wang, Chien-Yao, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors." arXiv preprint arXiv:2207.02696 (2022).

23. Ultralytics. 2023. "YOLOv8 Docs." https://docs.ultralytics.com/. Accessed April 27, 2023.

24. Jocher, G., Chaurasia, A., and Qiu, J. 2023. "YOLO by Ultralytics (Version 8.0.0)." Computer software. GitHub. https://github.com/ultralytics/ultralytics.

25. RangeKing and Jocher, G. 2023. "Brief summary of YOLOv8 model structure." GitHub Issue. https://github.com/ultralytics/ultralytics/issues/189. Accessed April 27, 2023.

26. Bochkovskiy, A., Wang, C., and Liao, H. M. 2020. "YOLOv4: Optimal Speed and Accuracy of Object Detection." arXiv preprint arXiv:2004.10934.

27. Liu, W., Hasan, I., and Liao, S. 2023. "Center and Scale Prediction: Anchor-free Approach for Pedestrian and Face Detection." Pattern Recognition 135: 109071. doi: 10.1016/j.patcog.2022.109071.

28. Lin, T-Y, Maire, M., Belongie, S., et al. 2014. "Microsoft Coco: Common Objects in Context." Computer Vision – ECCV 2014: 740–755. doi: 10.1007/978-3-319-10602-1_48.

29. Common Objects in Context. 2023. COCO. https://cocodataset.org/. Accessed April 27, 2023.

30. Ciaglia, F., Zuppichini, F. S., Guerrie, P., McQuade, M., and Solawetz, J. 2022. "Roboflow 100: A Rich, Multi-Domain Object Detection Benchmark." arXiv preprint arXiv:2211.13523.

31. Roboflow 100: A new object detection benchmark. 2023. RF100. https://www.rf100.org/. Accessed April 27, 2023.

32. Solawetz, J. Francesco. 2023. "What is YOLOv8? The Ultimate Guide." Blog post. https://blog.roboflow.com/whats-new-in-yolov8/. Accessed April 27, 2023.