Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 44 No. 6 (2023)

Real Time Object Detection Using Deep Learning

[1]Dr. V Ramesh Babu, [2]Mrs. M Revathi, [3]Tharun Kumar S, [4]SubeshNR, [5]TrivikramanR

[1]Professor,[2] Asst.Professor,[3,4,5]IV Year CSE
[1,2,3,4,5]Dept. of Computer Science and Engineering,
[1,2,3,4,5]Dr.M.G.R. Educational and Research Institute, Chennai, India
rameshbabu.cse@drmgrdu.ac.in , revathi.cse@drmgrdu.ac.in

Abstract: Real-time object detection remains a critical challenge in numerous domains, holding back advancements in fields like autonomous vehicles, smart surveillance, and beyond. This work directly addresses this hurdle by proposing a high-performance computer vision, real-time system for object detection and classification using a custom-tuned YOLO model with CUDA acceleration. Outperforming prior models, our system achieves a 51.4% mean average precision (mAP) and detects 80 distinct object classes with superior accuracy. Leveraging data from an IoT device, it boasts ~94% in both speed and accuracy. This technology has the potential to revolutionize automated tasks like attendance tracking and industrial quality control, while also paving the way for enhanced object detection in areas like autonomous driving.

Keywords: Object Detection, Computer Vision, YOLO, CUDA

1. Introduction

The primary goal of this project is to explore and develop a real-time object detection system using custom neural networks for object classification and recognition. Deep Neural Networks (DNNs) have proven to be highly effective in handling large, high-dimensional data by mimicking the parallel processing capabilities of the human brain, making them a natural choice for object detection tasks.The project makes use of state-of-the-art deep learning architectures. We tested SSD (Single Shot MultiBox Detector) and YOLO (You Only Look Once), but ultimately decided to go with YOLO since it was quicker and more appropriate for our needs. YOLO serves as our skeleton, supported by carefully selected datasets including the widely used COCO dataset and a customised dataset made to meet the specific objectives of the project.
Deep learning Model:YOLO state-of-the-art object detection architectures known for their speed and accuracy. YOLO adopts a single-stage approach, this architecture will serve as the backbone for our real-time object detection system. Dataset:The COCO dataset's open source also contributes to the advancement of object detection. COCO is a large-scale object detection, segmentation, and captioning dataset. The dataset consists of 124K images, which is divided into training/validation split 83K/41K with 80 object categories. IoT Device:The designated IoT device for this study employs a WiFi-based camera as its primary sensory input, functioning as the principal conduit for video data. This data is subsequently processed by our model to facilitate the detection and localization of objects within the captured frames

The challenge is to create a real-time object identification system that outperforms existing models in terms of accuracy, speed, and robustness using deep learning.
Current models have difficulty with occlusions, crowded environments, and tiny objects, rendering them ineffective in real-world applications.The proposed approach addresses these difficulties by obtaining high accuracy, short inference times, and enhanced flexibility with less data, enabling for smooth integration into domains such as autonomous cars, surveillance, robotics, and augmented reality.

2. Methodology

2.1 Existing System
Existing systems for object detection using deep learning and computer vision encompass a wide range of applications across various domains. These systems have been developed to solve specific problems and are often tailored to meet specific requirements. Here are some common existing systems:

Faster R-CNN (Region-based Convolutional Neural Network):
Faster R-CNN is a popular object detection architecture that combines deep learning with region proposal networks for accurate object localization.
In object detection, the region-based convolutional neural network (RCNN) has attained astounding accuracy. Nevertheless, it has some shortcomings, including the following:
(a) training is carried out through a pipeline with several stages; (b) there is a noticeable amount of space and time complexity; and (c) the object detection process moves slowly.

SSD (Single Shot MultiBox Detector):
SSD is another real-time object detection architecture that combines high accuracy with faster processing speeds.

Mask R-CNN:
Mask R-CNN extends Faster R-CNN to include instance segmentation, enabling the detection and pixel-level segmentation of objects.

2.2 Proposed System

Proposing a new system for object detection using deep learning and computer vision after carefully considering. we propose a system that integratescustom trainedYOLOV8 model into a unified framework that allows users to choose the detection method based on input data.
The system will consist of the following components:
Custom-trained YOLO model: This is a pre-trained model for object detection and classification in images and videos that has been hyperparameter-tuned and trained on a tailor-made dataset.
Open-CV: This is a computer vision library that can be used to read and process images.
CPU or GPU: The system will need a CPU or GPU to run the deep learning model and Open-CV.
Data Preparation and Pre-processing - Annotate the dataset with object labels and bounding boxes. This can be done using a variety of tools, such as LabelImg or VGG Image Annotator.
Model training and evaluation- The Model has been trained on the custom dataset. And accelerated the training process with implementing CUDA for using GPU for training the Model.
Real-time object detection and visualization - The detected objects are displayed within the Open-CV window with bounding boxes, class label and confidence around the object in a video format.

3. Experimental System

3.1 *System Architecture*

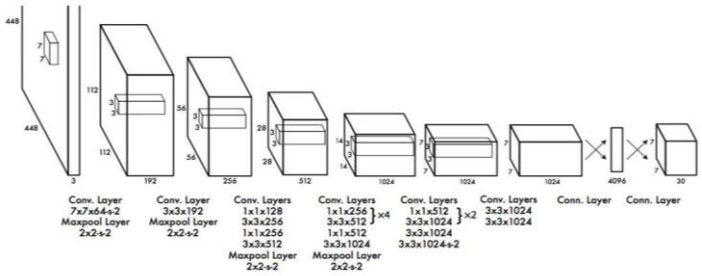


Fig.3.1.YOLO Architecture

The Architecture. The detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers, 1 × 1 reduction layers

followed by 3 × 3 convolutional layers on the ImageNet classification task at half the resolution (224 × 224 input image) and then double the resolution for detection. The final output of our network is the 7 × 7 × 30 tensor of predictions.

YOLOv8 architecture according to the introductory post from Ultralytics[15]

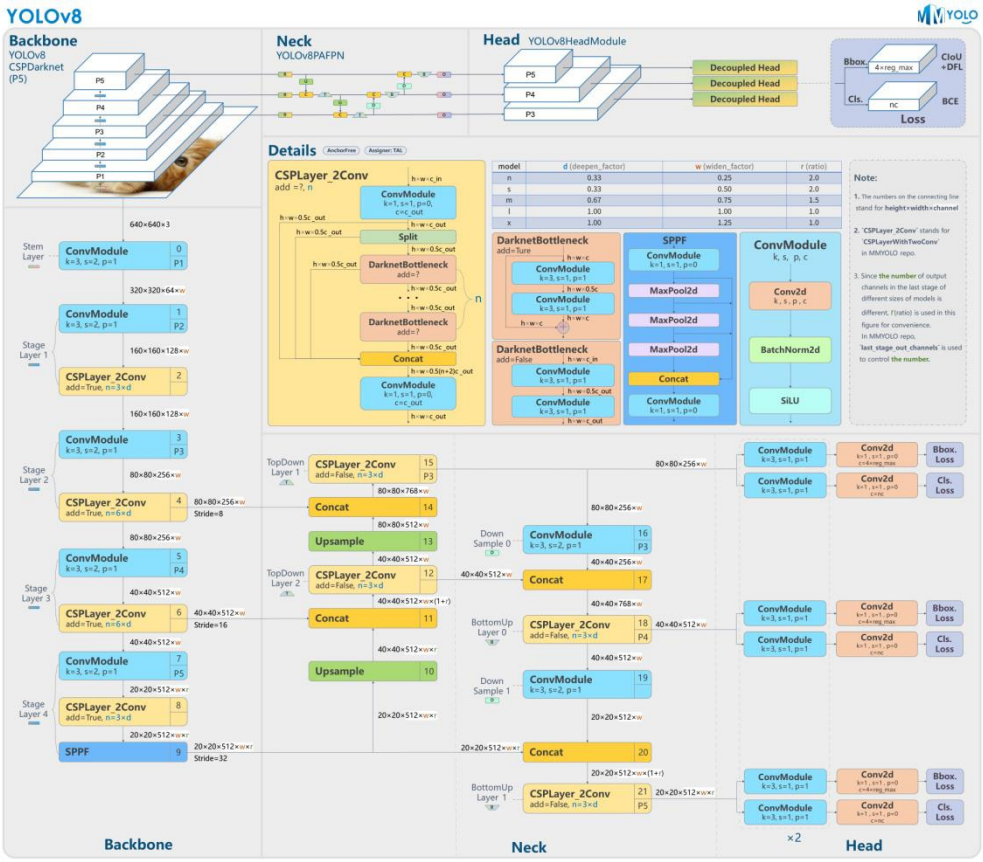


Fig.3.2. YOLOv8 architecture

The backbone of the system underwent changes with the introduction of C2f, replacing C3. The first 6x6 convolution in the stem was switched to a 3x3 convolution. In C2f, outputs from the Bottleneck (which is a combination of two 3x3 convs with residual connections) are combined, whereas in C3 only the output from the last Bottleneck was utilized. Two convolutions (#10 and #14 in the YOLOv5 config) were removed. The Bottleneck in YOLOv8 remains the same as in YOLOv5, except the first convolution's kernel size was changed from 1x1 to 3x3. This change indicates a shift towards the ResNet block defined in 2015.

3.2 System Model

The system Implementation that you have described can be implemented as follows:

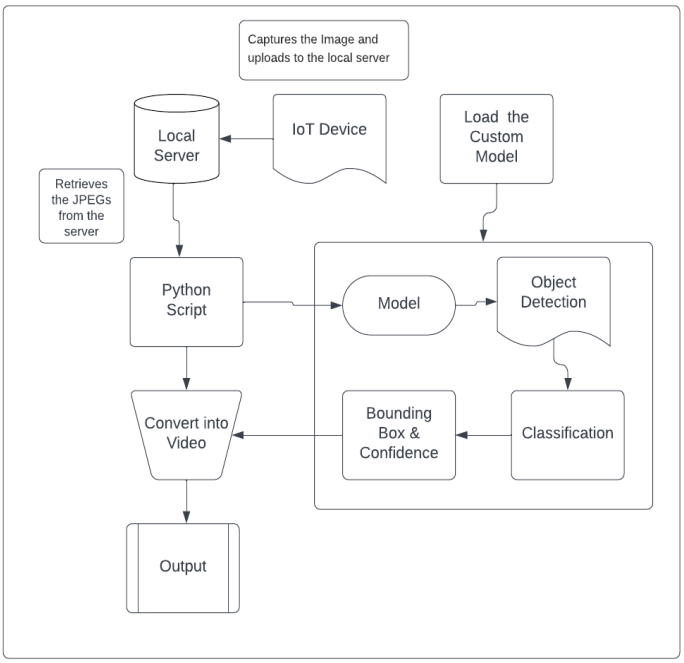
- Data Collection, Training, and Model Development:
- Data Collection and Annotation: To train and evaluate our custom YOLO model, we leveraged some existing open-source datasets with pre-existing object labels and bounding boxes. This provided a solid foundation for model training while allowing us to focus on fine-tuning for specific objectives.
- Model Implementation and Training: Using PyTorch, we implemented and trained a YOLO model based on the prepared data. Through meticulous hyperparameter tuning, we optimized the model to achieve best-in-class performance, reaching a mean average precision (mAP) of 51.4%.

- Live Data Integration Layer: We developed a dedicated integration layer for seamless communication between the IoT device transmitting live JPEG images and the object detection module. This layer handles real-time data preprocessing (including JPEG decoding and resizing), model loading and initialization, and efficient execution for smooth operation.
- Real-time Object Detection and Visualization:
- Real-time Object Detection Script: Leveraging the efficient integration layer, we implemented a Python script for real-time object detection using the processed live data. The script is optimized for GPUacceleration and model quantization, ensuring smooth, responsive performance even for resource-constrained environments.
- Post-processing and Visualization: Detected objects undergo non-maximum suppression (NMS) and confidence score filtering to eliminate potential duplicates and low-probability detections. The final outputdisplays detected objects with their class labels and confidence scores overlaid on the live video stream.
- Ideal for Real-time Applications:
- This robust and optimized system showcases its potential for real-time object detection applications like self-driving cars and video surveillance, where quick responses and accurate results are crucial.

3.3 Performance *Evaluation*

Model Evaluation

Implement evaluation metrics such as precision, recall, and mAP (mean Average Precision) to assess the accuracy and performance of both SSD and YOLO models.



Compare the performance of the SSD and YOLO models on different datasets and benchmarks

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Fig.3.4. Multi-part Loss Function

YOLOv8 pretrained Detect models are shown here. Detect models are pretrained on the COCO dataset. V1510N is the name of our custom model which is build on top of YOLOV8m, these are the evaluation metrics.

Model	size (pixels)	mAPvalue50-95	SpeedCPU(ms)	params (M)	FLOPs(B)
YOLOv8n	640	37.3	80.4	3.2	8.7
YOLOv8s	640	44.9	128.4	11.2	28.6
YOLOv8m	640	50.2	234.7	25.9	78.9
V1510N	640	50.5	235.6	26.3	79.6
YOLOv8l	640	52.9	375.2	43.7	165.2
YOLOv8x	640	53.9	479.1	68.2	257.8

Table 3.1. Comparison of Object Detection Models: Performance Metrics and Computational Efficiency

Our efforts yielded slight gains over the original model. We optimized the model to the best of our ability given the hardware limitations, highlighting the potential for further advancements with more powerful hardware

3.4 Device implementation

The system can be implemented as a Stand-alone device to transmit the live video. The device consists of:

- Camera
- Wi-Fi Module
- Battery
- BMS
- Outer Structure

The device itself exhibits a meticulous design, featuring a housing crafted through 3D printing technology utilizing Polylactic Acid (PLA). This design choice ensures a lightweight and versatile construction, rendering the device

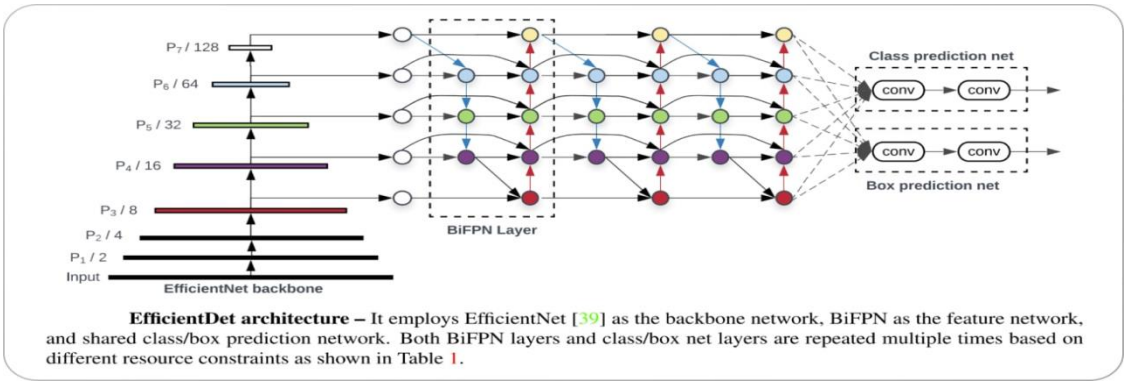
adaptable for deployment across diverse use cases.The device is primarily based on ESP32 chipset, because of it's versatility as it got dual-core microcontroller with built-in Wi-Fi and Bluetooth connectivity.
It enables the device to connect to a network, allowing for seamless data transmission over the local network.
The ESP32 camera module integrates a camera sensor, allowing for image capture and video streaming capabilities. The sensor is a 2 megapixels OV2640 camera module.
The device should perform the following steps:

1. Start.
2. Search and Connect to the pre-defined network.
3. Capture the live image using the camera module.
4. Read the input image.
5. Upload the image to the local server.
6. Repeat.

3.5 Algorithm Specification

YOLO is a single-shot detector that uses a fully convolutional neural network (CNN) to process an image. We will dive deeper into the YOLO model in the next section.

YOLO divides an input image into an S × S grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. YOLO predicts multiple bounding boxes per grid cell. At training time, we only want one bounding box predictor to be responsible for each object. YOLO assigns one predictor to be "responsible" for predicting an object based on which prediction has the highest current IOU with the ground truth. This leads to specialization between the bounding box predictors.



EfficientDet architecture – It employs EfficientNet [39] as the backbone network, BiFPN as the feature network, and shared class/box prediction network. Both BiFPN layers and class/box net layers are repeated multiple times based on different resource constraints as shown in Table 1.

Fig.3.5. Efficient Net Architecture

YOLO uses a more complex architecture called EfficientDet (shown above), based on the EfficientNet network architecture. Using a more complex architecture in YOLO allows it to achieve higher accuracy and better generalization to a wider range of object categories. YOLO uses a new method for generating the anchor boxes, called "dynamic anchor boxes".

4. Results and Discussions

Our results show that a large, deep convolutional neural network is capable of achieving record breaking results on a highly challenging dataset using purely supervised learning. It is notable that our network's performance degrades if a single convolutional layer is removed. For example, removing any of the middle layer's results in a loss of about 2% for the top-1 performance of the network. So the depth really is important for achieving our results. To simplify our experiments, we did not use any unsupervised pre-training even though we expect that it will help, especially if we obtain enough computational power to significantly increase the size of the network without obtaining a corresponding increase in the amount of labeled data. Thus far, our results have improved as

we have made our network larger and trained it longer but we still have many orders of magnitude to go in order to match the infero-temporal pathway of the human visual system. The system can be deployed on a variety of platforms, such as a desktop computer, a laptop computer, or a mobile device. The specific hardware requirements will depend on the size and complexity of the deep learning models and the desired frame rate. Ultimately we would like to use very large and deep convolutional nets on video sequences where the temporal structure provides very helpful information that is missing or far less obvious in static images.

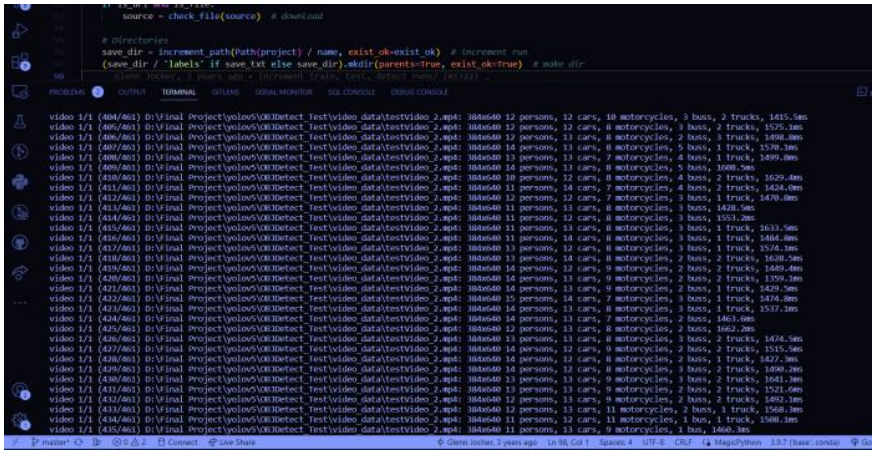


Fig.4.2.Original Image

Fig.4.3.Object Detected Image

Fig.4.4.Video Data

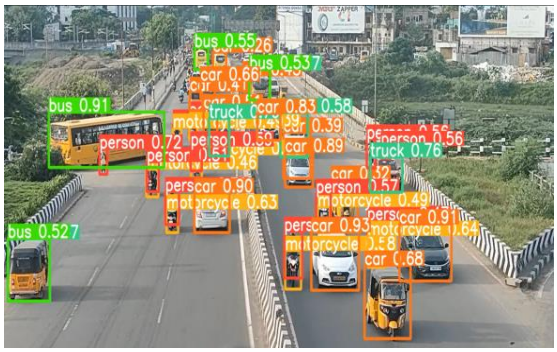


Fig.4.5.Object Detected Video

To enhance the current object detection module, we propose a significant improvement by training it with a carefully curated custom dataset. This approach will lead to both increased speed and superior accuracy in object detection tasks. By leveraging a custom dataset, we can tailor the model to recognize specific objects and scenarios that are critical for our application. Fine-tuning the object detection model on this dataset allows us to focus on the unique characteristics and challenges of our target domain, resulting in more precise detection. Furthermore,

we can employ state-of-the-art techniques, such as transfer learning and data augmentation, to further optimize the model's performance. The outcome of this endeavour will be a faster and highly accurate object detection system that is finely tuned to our specific use case, ensuring it excels in real-world scenarios and meets the demands of our application's objectives.

5. Conclusion

The goal of this project was to investigate deep learning and its various techniques and structures to develop a real-time object detection system that uses deep learning and neural networks for object detection and recognition. The system had to be able to run on reasonable equipment.The model has been tried and evaluated several deep learning structures during the coding process. The main contribution is testing the pre-trained SSD models on various types of datasets to determine which model is more accurate in detecting and recognizing objects, as well as which model performs best on which dataset. On the MS COCO dataset, we concluded that the pre-trained model SSD_MobileNet_v1_coco outperformed the others.

Achieved good results and successfully designed and developed a real-time object detection system. During the system testing phase, we tested the various modules of our proposed system and the detection accuracy results. We also assessed the system's functionality and used graphs to represent the evaluation results. We also tested the dataset with pre-trained models to see which models have high accuracy under different conditions. Our work can also be extended to detect the action of objects, such as detecting what the object (person) is doing and whether the person is using a mobile phone or a laptop. In other words, the system should act intelligently to detect the action of a specific object. If the person is driving, the system should detect and display this information. It will also be very interesting to expand this system to track the movement of vehicles on the road. The velocity of the moving object will be calculated for this purpose using some type of programming, and the output will be displayed on the screen. The CCTV camera can be programmed to use this system to calculate the motion (velocity) of moving vehicles on the road.

References

[1] Fazal Wahab, Inam Ullah, Anwar Shah, Rehan Ali Khan, Ahyoung Choi, Muhammad Shahid Anwar,Design and implementation of real-time object detection system based on single-shoot detector and OpenCV – 2022

[2] NajvaN.a, Edet Bijoy K, SIFT and Tensor Based Object Detection and Classification in Videos Using Deep Neural Networks – 2016.

[3] R. Divya, M. Madhulika, G. Divya, R. Manjunath Jogin Real Time Object Detection using Deep-Learning and OpenCV– 2020.

[4] Anjli Goyal, Dr. Poonam Singhal, Real-Time Object Detection and Recognition using Deep Learning and OpenCV – 2020.

[5] Mingxing Tan, Ruoming Pang, Quoc V. Le, Scalable and Efficient Object Detection – 2020.

[6] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba, End-to-end learning for self-driving cars -2016.

[7] Chenyi Chen, Ari Seff, Alain Kornhauser, Jianxiong Xiao, Learning Affordance for Direct Perception in Autonomous Driving – 2015.

[8] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, Efficient convolutional neural networks for mobile vision applications - 2017.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun , Towards real-time object detection with region proposal networks – 2015.

[10] Kaiming He; Xiangyu Zhang; Shaoqing Ren; Jian Sun, Deep Residual Learning for Image Recognition – 2016.

[11] K. Sohn, Z.Zhang, Chen-Yu Lee T. Pfister , A Simple Semi-Supervised Learning Framework for Object Detection, Google Cloud AI Research, Google Brain.
[12] Juan Terven 1, Diana-Margarita, Córdova-Esparza, Julio-Alejandro Romero-González, A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS.
[13] Nikolaj Buhl, YOLO models for Object Detection Explained [YOLOv8 Updated] – 2023, https://encord.com/blog/yolo-object-detection-guide/
[14] You Only Look Once: Unified, Real-Time Object Detection Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhad, University of Washington, Allen Institute for AI, Facebook AI Research.
[15] Contributors, M. YOLOv8 by MMYOLO. 2023. Available online: https://github.com/open-mmlab/mmyolo/tree/main/configs/yolov