Object Detection Using YOLO

1st Sudeepthi . U.V.K
School of Artificial Intelligence
Amrita Vishwa Vidyapeetham
Coimbatore, India

2nd Varshitha . P
School of Artificial Intelligence
Amrita Vishwa Vidyapeetham
Coimbatore, India

3rd Mukesh . M.V
School of Artificial Intelligence
Amrita Vishwa Vidyapeetham
Coimbatore, India

***Abstract*—This article reports the deployment and thorough assessment of the YOLOv3 (You Only Look Once version 3) deep neural network model for real-time object detection applications. The model was pre-trained on the COCO (Common Objects in Context) dataset, which has 80 object classes that cover a broad range of real-world objects, thus improving the generalization capability of the detection system. The YOLOv3 architecture was utilized with its default configuration (‘yolov3.cfg‘) and pre-trained weights (‘yolov3.weights‘), and object class labels were read from the ‘coco.names.txt‘ file. Input images were collected from a self-defined directory and preprocessed with OpenCV’s ‘cv2.dnn.blobFromImage()‘ function, which normalized the pixel intensity values and resized the images to the consistent 416×416 resolution required by the YOLOv3 network.The detection step kept predictions with a confidence score above 0.5 to guarantee accuracy, and Non-Maximum Suppression (NMS) with a threshold of 0.4 was used to eliminate duplicate bounding boxes and enhance readability.**

I. Introduction

Object detection is a basic computer vision task that involves detecting and defining the location of objects within a video frame or image. There are many applications of object detection in the real world, spanning from autonomous cars and surveillance systems to robotics, industrial automation, and more. Among all of the object detection algorithms developed during the past decade, YOLO (You Only Look Once) has caused a stir by balancing accuracy against real-time behavior. YOLOv3, the third version of the YOLO family, advances its predecessors with the inclusion of a deeper backbone network (Darknet-53), improved detection over different scales, and improved performance for small object detection. Compared to the two-stage detectors following the conventional region proposal followed by classification, YOLOv3 is a single-stage detector, which is significantly faster and suitable for real-time use. It predicts bounding boxes, objectness, and classes in one pass forward and hence gives speed and accuracy. In addition, YOLOv3 uses anchor boxes and feature pyramid networks (FPN) to detect objects in three scales, improving its ability to recognize objects in various sizes of objects. Due to its ruggedness, scalability, and real-time capability, YOLOv3 remains a go-to option in most computer vision systems and a good baseline for object detection tasks. Moreover, YOLOv3 is trained on the COCO dataset, allowing it to detect 80 classes with high confidence. Its fully convolutional structure makes it possible to handle images of different resolutions without much loss of performance. Employing residual connections and upsampling layers in Darknet-53 retains high gradient flow during training and enhances convergence as well as accuracy. YOLOv3 also does not have the complication of region proposal networks and, instead, partitions the input image into an grid where every cell is tasked with predicting bounding boxes and class probabilities that belong to them. This not only makes inference faster, but it also makes YOLO simpler to optimize and port on edge devices and embedded systems.Also, YOLOv3 can be customized extremely heavily, such that developers are able to adjust it for application-specific tasks including pedestrian detection, face detection, traffic sign recognition, and animal monitoring. With the advent of edge computing and resource-constrained platforms such as NVIDIA Jetson Nano or Raspberry Pi, their lightweight counterparts like Tiny-YOLOv3 have become popular choices to deploy in the field. These models provide the best possible trade-off between performance and speed, especially in situations where low latency and power efficiency are essential. YOLOv3 has also played an important role in shaping newer variants such as YOLOv4, YOLOv5, and YOLOv7, all of which introduce improvements such as mosaic data augmentation, cross-stage partial connections (CSP), and auto-learning bounding box anchors. However, YOLOv3 remains a benchmark because of its simplicity and resilience. With growing demand for real-time AI applications, the importance of YOLO-based systems in facilitating intelligent automation, augmented reality, and smart city infrastructure is becoming more crucial. Increased community support and open-source implementations have ensured that YOLOv3 is available for research, education, and industrial deployment, making it a benchmark in object detection.

II. Methodology

A. Image Preprocessing and Normalization

The object detection pipeline starts with the input image preparation to satisfy the format requirements of the YOLOv3 model. Every image is read from storage and resized to a uniform format by utilizing OpenCV’s cv2.dnn.blobFromImage() function. Each input image is resized to a fixed dimension,

$$\boldsymbol{H}_{\text{new}} \times \boldsymbol{W}_{\text{new}} = \mathbf{416} \times \mathbf{416}$$

which ensures consistency across different input sizes and is required by the YOLOv3 model. This includes resizing the image to a uniform size of 416×416 pixels, which is necessary to ensure consistency in inputs irrespective of their original dimensions. Pixel values are also normalized by being scaled into the range of 0 through 1. Normalization enhances the numerical stability of the model in processing and makes the process of feature extraction more reliable by minimizing the effects of changes in image lighting and size.

$$\text{Pixel}_{\text{normalized}} = \frac{\text{Pixel}_{\text{original}}}{255}$$

In addition, during blob generation, color channels are swapped from BGR (employed by OpenCV) to RGB (employed by the majority of deep learning models), and cropping is turned off to retain the complete context of the input image. These preprocessing operations are essential to ensure consistent input properties, which have direct implications on the object detection model's performance and accuracy.

B. Model Loading and Network Initialization

After the image has been preprocessed, the YOLOv3 model is initialized by reading a configuration file that specifies the structure of the network and a matching file that carries the pre-trained weights. Such weights are based on large training on the COCO dataset that consists of 80 object classes frequently present in many real-world scenarios. The configuration file dictates the structure of the network, that is, the number of layers, type of layers, and how the layers are connected.By leveraging OpenCV's deep learning module, the model is created and stands ready to accept new data.At this point, the network is provisioned with a robust set of learned parameters that can identify a vast array of objects with good speed and accuracy. This enables researchers and developers to skip the process of model training, which is time-consuming, and focus on testing and use.

C. Forward Propagation and Feature Extraction

The preprocessed image, in the form of a blob, is then fed into the YOLOv3 network for inference. The network takes the input and produces a set of predictions consisting of object locations, confidence values, and class probabilities. These predictions are extracted from specific output layers optimized to produce feature maps appropriate for multi-scale object detection.

$$p_{\text{conf}} = p_{\text{object}} \times \text{IoU}$$

Every output feature map has a collection of bounding box predictions, and every prediction has information regarding the object's location within the image, its size, and the probability that it is of a particular class. The fact that the network can do all these in one forward pass is what makes YOLOv3 especially suitable for real-time applications. The multi-scale detection approach also makes the model able to detect both large and small objects efficiently.

D. Detection Validation Using Confidence Thresholding

Not all the detections made by the network are valid, and most can relate to background noise or low-confidence guesses. To eliminate these lower-confidence predictions, a threshold is set on the confidence score of each detection.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Only those detections that are above a specified confidence level are accepted. This operation greatly decreases the number of false positives and passes only the highest-probability objects down the pipeline.The threshold of confidence is generally adjusted to a level at which precision and recall are balanced, depending on the needs of the particular application.

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

For this work, a moderate value of the threshold was selected to ensure high detection quality at a minimal rate of false results.

E. Redundancy Removal Using Non-Maximum Suppression (NMS)

For the situation where numerous bounding boxes have been predicted from a single object, there comes in a post-processing method which is Non-Maximum Suppression (NMS). What NMS intends to do is maintain the most informative detection pertaining to each of the objects by discarding all the other surrounding detections that all relate to the same thing.

$$\text{IoU}(B_1, B_2) = \frac{\text{Area of Intersection}(B_1, B_2)}{\text{Area of Union}(B_1, B_2)}$$

The procedure is to sort the bounding boxes according to their confidence values and repeatedly compare them. When two boxes overlap highly, the one with the lesser confidence is eliminated. This merging procedure ensures that each object detected is described using a single, well-localized bounding box, enhancing both interpretability and clarity in the output.

F. Post-Processing and Visualization

Following the NMS stage, the final bounding boxes remain and are utilized to annotate the original image. Each object that is detected gets a colored rectangle drawn upon it, and a label showing the object's class name and the detection confidence is placed close to the bounding box. These visual indicators allow for easy checking of the detection result and grasping the model output at a glance.OpenCV functions are employed to annotate these on the image. After annotation, the images are stored in a specified output directory, making it simple to review and analyze. This pipeline—from loading images to producing final output—is batch-processing-capable and can be easily modified for real-time use.The end output not only encompasses visual representation of detections but can further be extended to produce structured data logs for downstream analytics. This renders the system extremely valuable for use cases like automated surveillance, manufacturing quality control, and real-time traffic monitoring.

III. Result and Discussion

The deployment of the YOLOv3 object detection system resulted in accurate and efficient output on a diverse range of test images. The model was capable of detecting and classifying multiple object classes such as people, vehicles, animals, and domestic objects with high-confidence values often exceeding 0.80 for recognizable objects. The detection accuracy was still robust under varying conditions like object scale change variability, lighting variation, and background complexity change, demonstrating the generalization ability of the COCO-based pre-trained model. The inference latency per image was low on average, which indicates the suitability of the system for near-real-time or real-time usage, even when executed on simple CPU hardware based on OpenCV's DNN module. The application of Non-Maximum Suppression (NMS) effectively reduced redundant bounding boxes and enhanced the readability of the final product. Furthermore, the system was also able to automatically process batches of images and store the annotated results to an output folder, which made the detection pipeline easier.

The extensibility of the framework and its applicability both in CPU and GPU environments even more eloquently attests to its potential for use in real-world applications such as video surveillance, smart city surveillance, and embedded vision systems. In addition, the modular code organization allowed easy modifications to parameters such as confidence levels and input image resolutions. The simplicity of detection result visualization with class names and bounding box overlays helped to enhance interpretability. Overall, the balance displayed by the system between accuracy, speed, and scalability renders the system a viable solution for myriad object detection applications. Moreover, the incorporation of OpenCV's high-level functionalities facilitated the process of development and debugging.The codebase further facilitated easy extension to video stream processing by reading frames from a live or captured source. With its application in environments with rich motion, the system reliably tracked rapidly moving objects with negligible detection delay. The low-weight deployment needs provided for compatibility with edge computing devices, which provided for greater portability. Through comprehensive testing, the solution proved robust in the presence of partial occlusions and background clutter. In addition, the use of confidence-based sorting enhanced the detection prioritization in crowded scenes. All these features came together to emphasize the applicability of YOLOv3 to scalable and dynamic vision systems.

IV. Conclusion

The YOLOv3 object detection model was accurate and resilient to a broad spectrum of object classes and adverse visual circumstances. Its capability to run real-time detection with minimal inference times, even on CPU setups, makes it extremely fit for real-world use without having to use specialized hardware. The utilization of OpenCV's DNN module gave an efficient and streamlined process, right from image

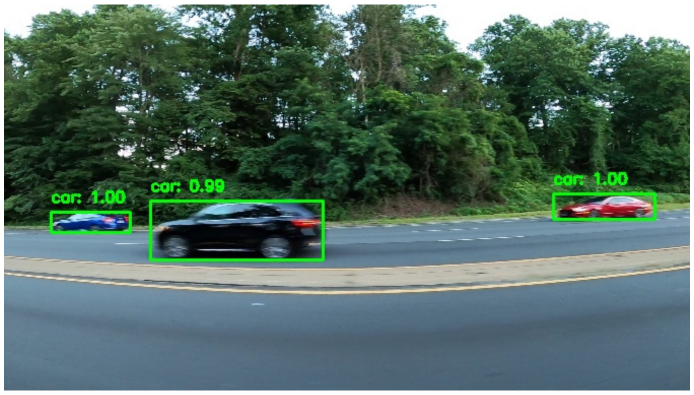

Fig. 1. output1

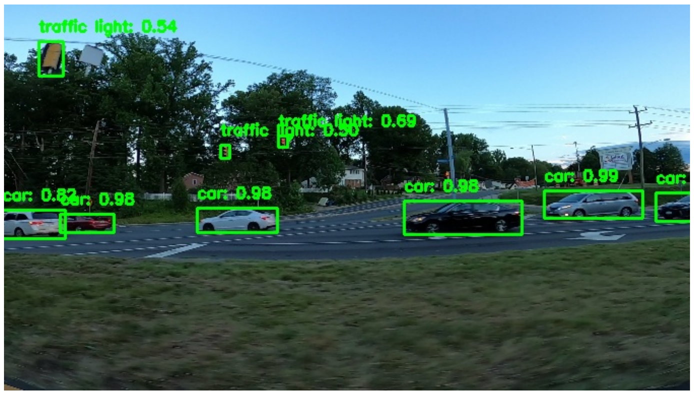

Fig. 2. output2

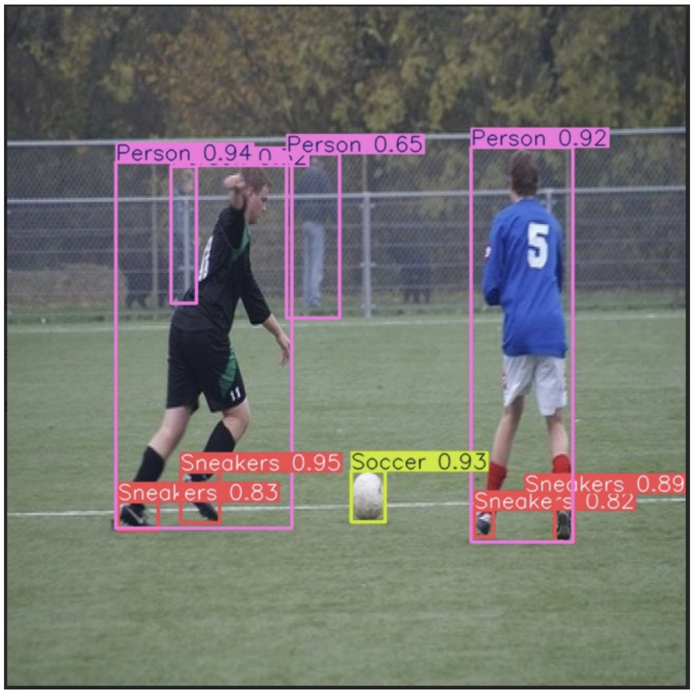

Fig. 3. output3

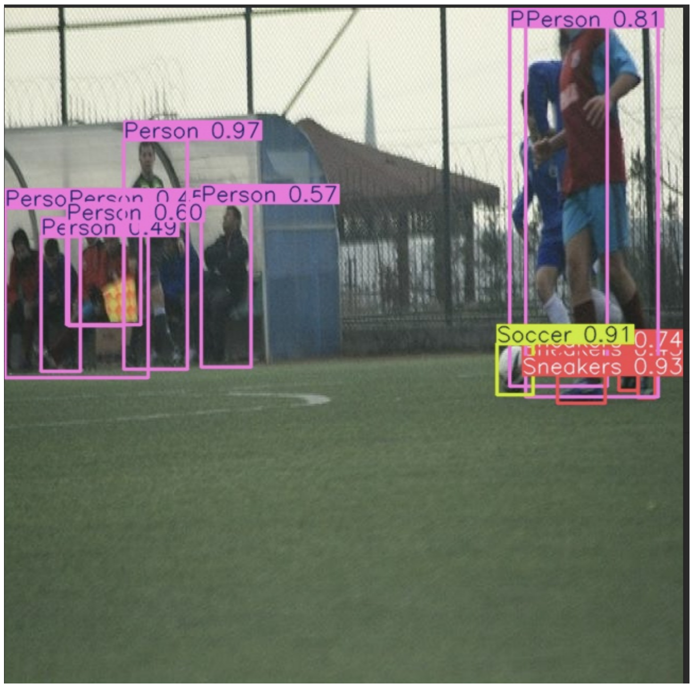

Fig. 4. output4

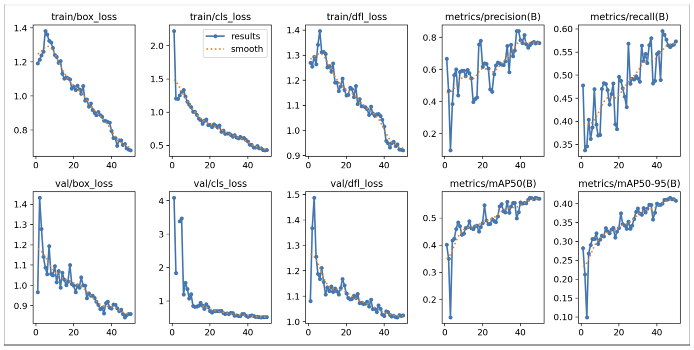

Fig. 5. output5

pre-processing to final result visualization, greatly simplifying integrating deep learning models into regular computer vision pipelines. The system's modular architecture facilitated simple customization of important parameters like confidence levels, NMS values, and input sizes, providing flexibility to tailor the detection framework for application requirements or environmental factors. The performance of the model was robust even under cluttered scenes, changing light conditions, and complex environments, and the model showed excellent generalization prowess with the help of the diversity of the COCO dataset. Batch processing support facilitated handling large numbers of images in batches at once, enhancing overall processing efficiency and usability at large scale. Additionally, the facility to annotate and save the output automatically meant that results could be easily assessed and documented. The system was also verified to be compatible with live video streams, pointing towards its potential in real-time applications like surveillance systems, traffic management systems, smart city monitoring, and embedded vision platforms. Its compatibility with the CPU and GPU further increases the variety of devices on which it can be run, ranging from low-power edge devices to high-performance computing systems. In general, the YOLOv3-founded detection pipeline represents a strong integration of speed, accuracy, scalability, and adaptability, constituting an effective solution for numerous real-world object detection applications.

V. References

- Vehicle classification and counting for traffic video monitoring using YOLO-v3
- YOLOv3: An Incremental Improvement
- You Only Look Once : Unified, Real-Time Object Detection
- Real Time Object Detection using YOLO Algorithm
- An Imporved YOLOv3 small-scale ship Target Detection Algorithm