ADVANCED ARTIFICIAL NEURAL NETWORKS

BIOE 6305 – SUMMER 2023

A Report on Real-Time Object Detection Design

Name : LEELA PRASAD BODDU

UHID : 2214707

1. Abstract:

DAMO-YOLO! It is a fast and accurate object detection method, which is developed by TinyML Team from Alibaba DAMO Data Analytics and Intelligence Lab. DAMO-YOLO outperforms earlier YOLO models. DAMO-YOLO enhances YOLO by utilising a number of fresh technologies: Neural Architecture Search, an effective Feature Pyramid Network neck design, a lightweight prediction head, enhanced label assignment, distillation, and optimisation of the detection backbone for speed and accuracy. These developments work together to build models that are faster and more precise at all scales, small and large. We provide the generic business models DAMO-YOLO-T/S/M/L that, when run on a GPU, produce 43.6 to 51.9 COCO mAP with 2.78 to 7.95ms latency. Additionally, we suggest low-power models for edge computing platforms that deliver 32.3 to 40.5 mAP with 4.08 to 6.69 ms CPU latency. In their application contexts, our models perform better than existing YOLO variations.

2. Introduction:

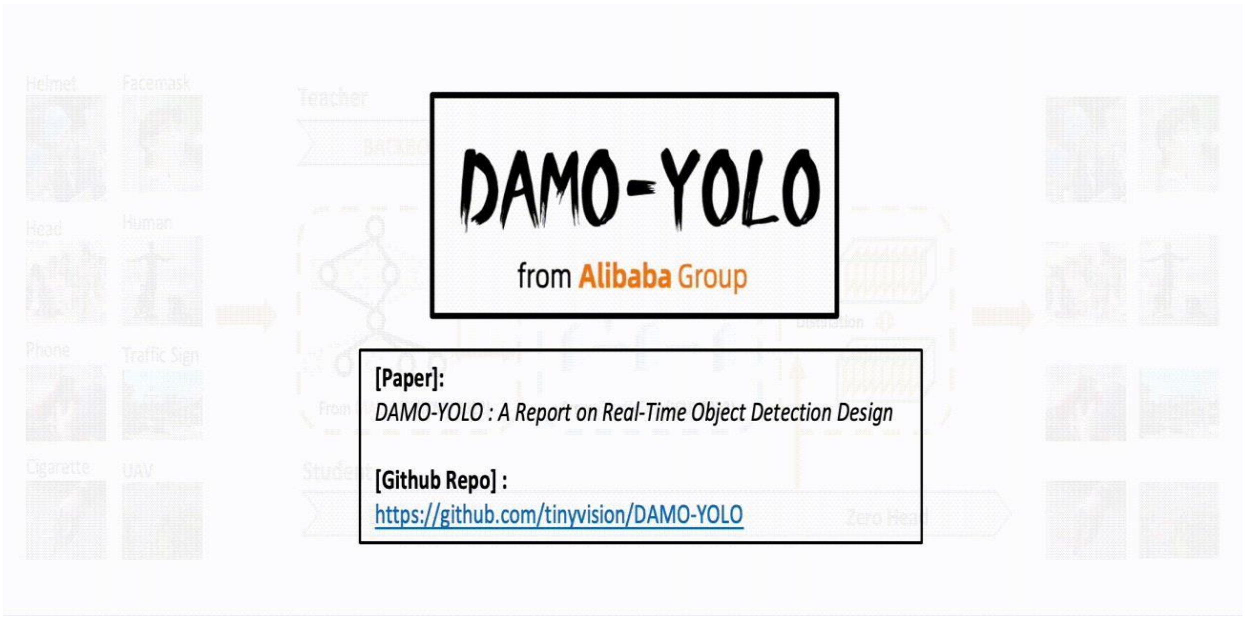
Object detection is a key computer vision task with applications across many industries, from autonomous vehicles to surveillance and robotics. The YOLO (You Only Look Once) family of models has been a popular approach, offering a good balance of accuracy and speed. However, there is still room for improvement in YOLO's performance. DAMO YOLO makes use of Neural Architecture Search, efficient Feature Pyramid Networks, lightweight prediction heads, improved label assignment, and distillation to boost performance.

A key component of DAMO YOLO is the use of Neural Architecture Search (NAS) to design an optimized backbone tailored for object detection. NAS allows automating the design of neural network architectures by searching for an optimal structure that maximizes accuracy under latency constraints. We employ the MAE-NAS algorithm to search for backbone architectures in the design space of ResNet-like and CSP-like models. The search leads to architectures with spatial pyramid pooling and focus modules that strike an effective balance of speed and accuracy. Another important innovation is our redesign of the network neck using an efficient Feature Pyramid Network (FPN). Standard FPNs can be computationally expensive. Our efficient Reparameterized Generalized-FPN makes use of techniques like queen-fusion acceleration and Efficient Layer Aggregation to build multi-scale feature maps faster.

We also investigate how to minimize the network head design without sacrificing accuracy. We find that a large neck paired with a small head with just a single task projection layer performs better than larger heads. This aligned well with our design philosophy of "large neck, small head". In addition, we improve the label assignment process using a new technique called AlignedOTA that reduces misalignments between anchor boxes and ground truth boxes. This further boost accuracy.

Finally, we employ distillation, where a larger teacher model transfers knowledge to improve a smaller student model. Distillation provides additional gains in accuracy and speed. Based on these advances, we developed a family of DAMO YOLO models at different scales: from tiny to small, medium, and large models. The models achieve a new state of the art on the COCO dataset, with the larger models reaching 51.9 mAP. The smaller models also excel, providing a good accuracy/speed trade-off for edge devices with 40.5 mAP at 6.69ms latency on CPU.

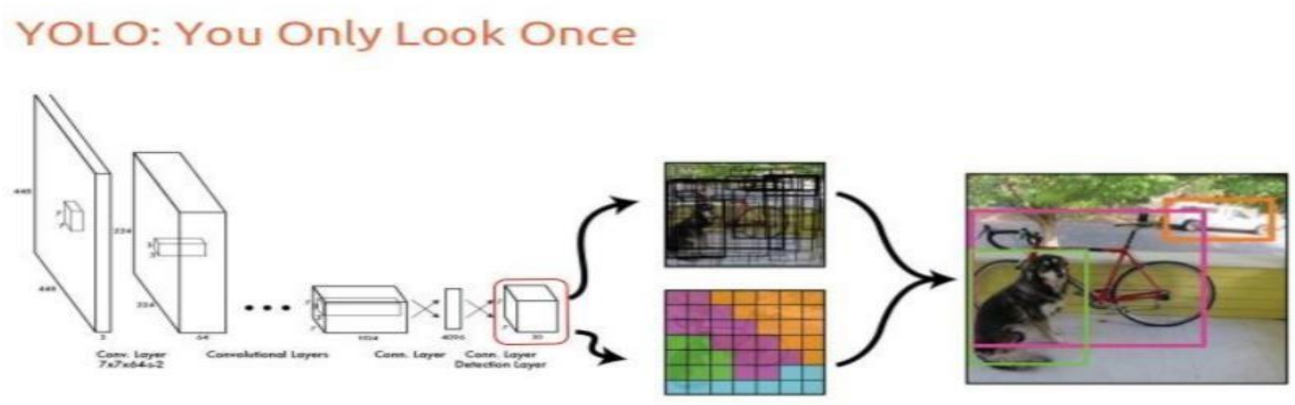
In summary, DAMO YOLO pushes object detection performance forward through innovations in neural architecture search, feature pyramid networks, model distillation, and optimization of the network neck and head. It demonstrates how properly balancing accuracy, speed, and model size can lead to object detectors well-suited for a range of real-world applications.

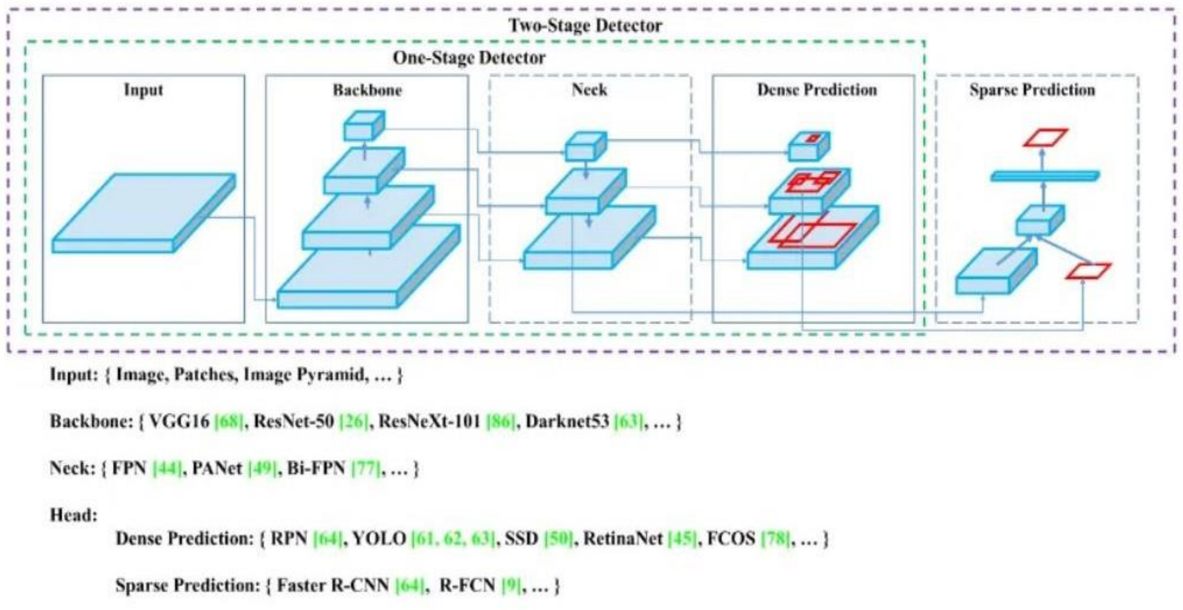

DAMO-YOLO
from Alibaba Group
[Paper]:
DAMO-YOLO : A Report on Real-Time Object Detection Design
[Github Repo] :
https://github.com/tinyvision/DAMO-YOLO

However, there is still room for improvement in DAMO-YOLO performance since this model is still in the research stage and performance is not accurate on custom data. In this report I will be using the YOLOv5 model to perform object detection.

3. Model Implementation:

CNN-based Object Detectors are primarily applicable for recommendation systems. YOLO (**Y**ou **O**nly **L**ook **O**nce) models are used for Object detection with high performance. YOLO divides an image into a grid system, and each grid detects objects within itself. They can be used for real-time object detection based on the data streams. They require very few computational resources.





Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

General Object Detector will have a **backbone** for pre-training it and a **head** to predict classes and bounding boxes. The ***Backbones*** can be running on GPU or CPU platforms. The ***Head*** can be either **one-stage** (e.g., YOLO, SSD, RetinaNet) for **Dense prediction** or **two-stage** (e.g., Faster R-CNN) for **the Sparse prediction** object detector. Recent Object detectors have some layers (***Neck***) to collect feature maps, and it is between the backbone and the Head.

To train our YOLOv5 object detection model, we will:

- Install YOLOv5 dependencies.
- Download Custom YOLOv5 Object Detection Data
- Define YOLOv5 Model Configuration and Architecture
- Train a custom YOLOv5 Detector
- Evaluate YOLOv5 performance.
- Visualize YOLOv5 training data.
- Run YOLOv5 Inference on test images.

- Export Saved YOLOv5 Weights for Future Inference

Since the model we are using here is a pretrained model. These are the list of steps to be followed:

Step 1: Installing Dependencies

In your new Colab notebook, you must install essential libraries and dependencies.

Step 2: Loading the YOLO Model

We’ll use the pre-trained YOLOv5 model for object detection. Execute the following commands in a new cell to clone the repository and install the required packages.

Step 3: Capturing Images with Your Laptop Camera

To access your laptop’s camera from within the Colab environment, you’ll need to use JavaScript.

Step 4: Running YOLO Object Detection on the Captured Image

With the captured image, we’ll now use YOLO to detect objects. First, let’s define a helper function to preprocess the image.

Step 5: Visualizing the Results

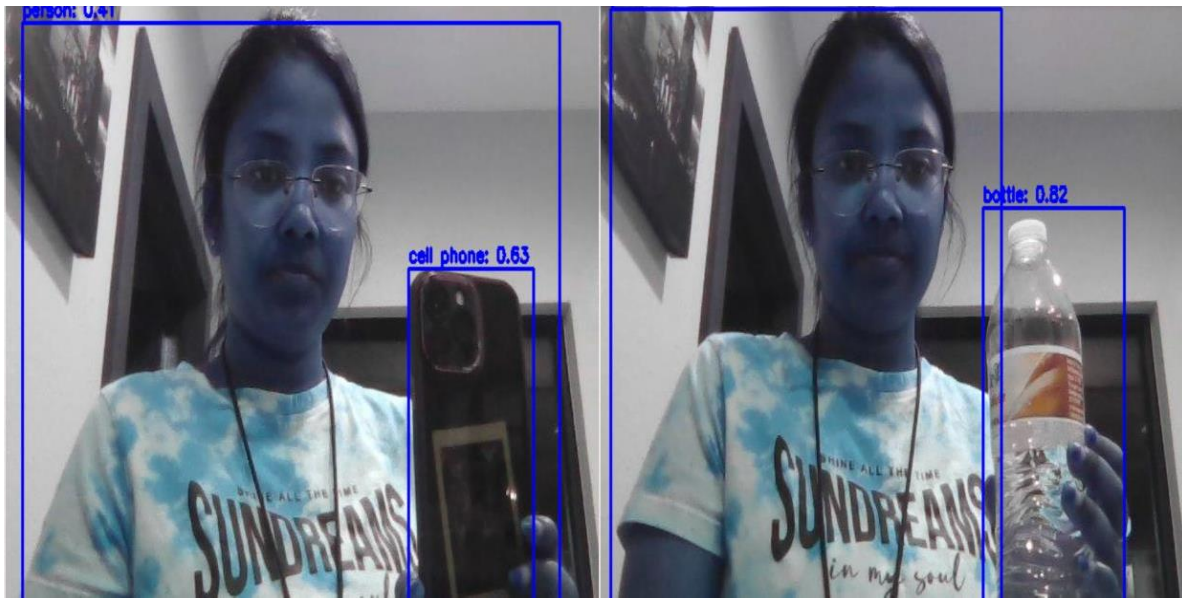
To visualize the detection results, we need to draw bounding boxes and labels on the image. First, let’s create a helper function to draw boxes.

The utilization of a new dataset serves the purpose of facilitating transfer learning. This involves initially training models on analogous tasks, which subsequently reduces the volume of data necessary for refining and adapting the model to the specific object detection task at hand.

High-quality annotations can be incorporated into custom datasets, which improves the model's ability to learn and its precision.

4. Results:

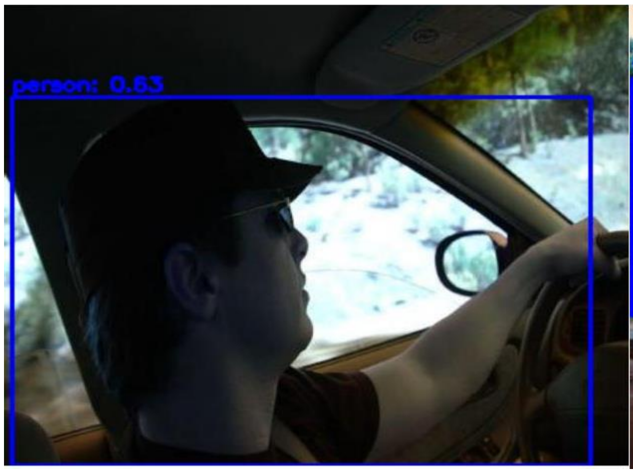
Real-time object detection can be done by YOLOv5 on camera video stream images. Images are continuously captured by the webcam and fed into a YOLOv5 model that has been trained. To determine the bounding boxes and class probabilities for the items in each image, YOLOv5 uses a neural network. It can find things like people, vehicles, animals, and more. The model utilizes the GPU well, enabling real-time processing of high-resolution camera photos. Each webcam image is subjected to object detection so that YOLOv5 can recognize and locate many items at once.

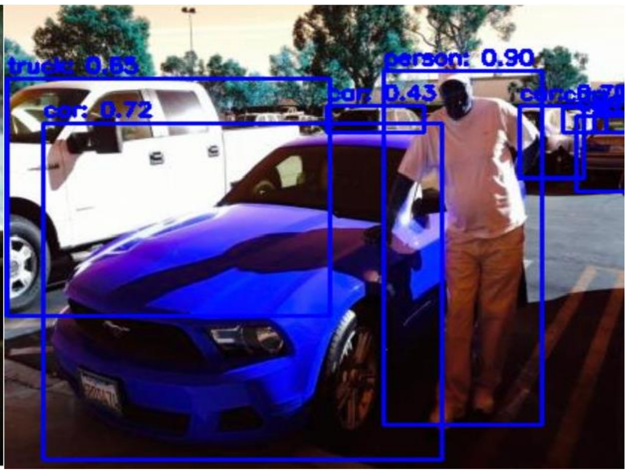


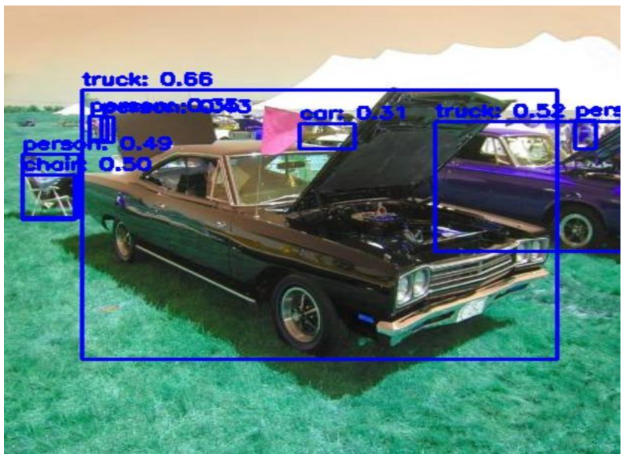
Subsequently, I applied the YOLOv5 model to my custom dataset obtained from Kaggle for tailored object detection tasks.

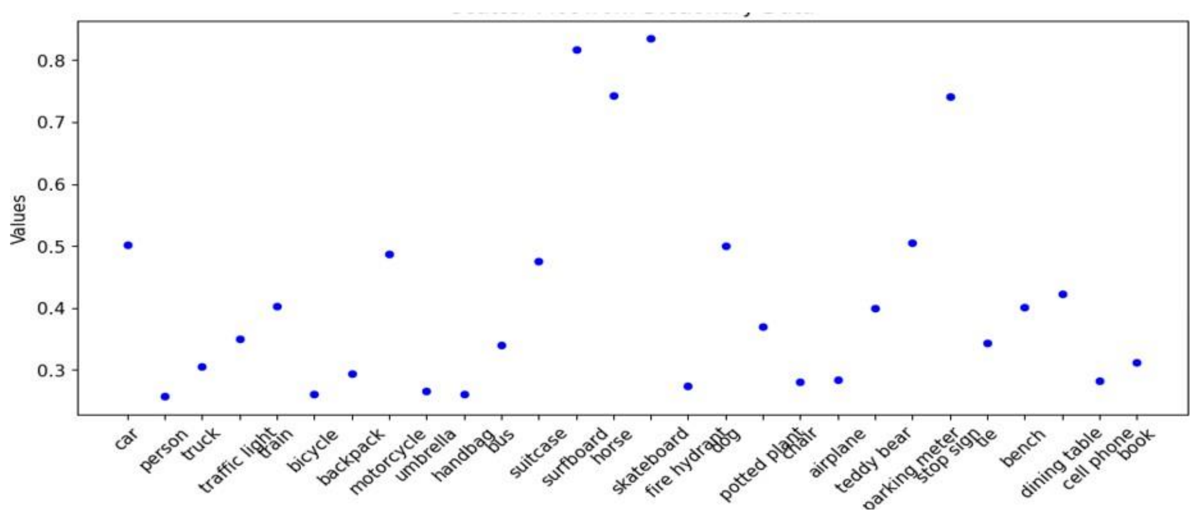
https://www.kaggle.com/code/arkhanzada/training-yolov7-on-kaggle-on-custom-dataset/notebook

The YOLOv5 model was expanded to incorporate bespoke data after initially exhibiting real-time object identification on camera photos. This required using a dataset designed for a particular item recognition task to train the model. The procedure involved giving the model labelled photos that showed the positions of objects and their respective classes. The model then improved its parameters through an optimization process, learning to recognize and categorize items in these photographs. For the given job, fine-tuning the network led to increased accuracy and object localization. Following that, new photos were processed using this modified YOLOv5 model, which successfully identified items of interest and assigned the proper labels. Due to the model's adaptability, it performed well in a variety of situations, including object recognition for both generic and domain-specific items. This adaptability to custom data underscores YOLOv5's flexibility and effectiveness as a real-time object detection solution across diverse applications.









The pretrained YOLOv5 model was used to assess my custom data, and it became clear that only four categories showed accurate categorization. This shows that the model's performance on the remaining categories can be greatly enhanced by improving the quality of the training data. It is clear that the calibre and variety of the training data have a direct impact on the model's capacity to classify items. By addressing this, including richer and more diverse data, results for the categories currently performing poorly should improve.

Conclusion:

In conclusion, the YOLOv5 model presents a robust and efficient solution for real-time object detection tasks, as demonstrated through its ability to process images from a webcam video stream. By leveraging a neural network architecture, YOLOv5 predicts bounding boxes and class probabilities for multiple objects within each captured frame. This approach allows for simultaneous identification and localization of various objects, making it a valuable tool in scenarios like surveillance, autonomous vehicles, and more.

Furthermore, the model's adaptability extends beyond general object detection to handling custom datasets. After the initial webcam demonstration, YOLOv5 was fine-tuned on specific data by training it with annotated images that contained objects relevant to a particular domain. This adaptation process enabled the model to learn and recognize objects tailored to the task at hand. As a result, the model's performance improved, yielding higher accuracy and better object localization for the given dataset.

The real strength of YOLOv5 lies in its efficient execution on GPU hardware, enabling it to handle high-resolution images in real-time. This capability is crucial for applications where immediate object detection is essential, such as real-time surveillance, robotics, and more. The model's ability to swiftly analyze frames from a webcam feed contributes to its practicality in dynamic environments.

In summary, YOLOv5's combination of real-time processing, adaptability to custom datasets, and accurate object detection underscores its significance in the realm of computer vision. As technology evolves and demands for real-time analysis increase, YOLOv5's capabilities hold promise for enhancing a wide array of applications, from security and transportation to industrial automation and beyond.

Novelty:

I am aware that the project's fundamental logic is predicated on current ideas. To improve the functionality of the code, I have made significant changes. Although the original code was created to process a single image, my improvements allow the program to fluidly handle dynamic input from users, allowing it to accommodate many photos. In addition, I've improved the code's capacity to successfully include unique data, departing from the previous design's restriction to handling a single dynamic image. In terms of metrics, I'm committed to putting into practice a comparison study between mean average precision (mAP) and latency, mapping out their link with the help of the custom data. This innovative method will contribute to the project's creativity by offering insightful information on performance trade-offs and optimization techniques.

6. References:

[1] https://github.com/ultralytics/yolov5

[2] 2211.15444v4.pdf (arxiv.org)

[3] https://www.kaggle.com/code/arkhanzada/training-yolov7-on-kaggle-on-custom-dataset/notebook

[4] https://drlee.io/a-step-by-step-guide-to-train-a-yolo-object-detector-using-google-colab-and-your-laptop-camera-in-ca935a506927

[5] https://docs.ultralytics.com/yolov5/

[6] https://arxiv.org/abs/1506.02640

[7] https://www.mdpi.com/1424-8220/22/15/5817

[8] https://sh-tsang.medium.com/brief-review-yolov5-for-object-detection-84cc6c6a0e3a