You Only Look Once: Unified, Real-Time Object Detection

Folarin John Madandola, Kavya Reddy Maale, Pavani Mathari, Sai Kumar Miryala
Department of Mathematics and Statistics
University of West Florida
November 2024

ABSTRACT

YOLO is a novel approach to object detection that transforms classification into a regression problem, focusing on spatially separated bounding boxes and class probabilities. This single neural network predicts these probabilities directly from full images, allowing for end-to-end optimization of detection performance.
Fast YOLO, a smaller version of our unified architecture, processes images in real-time at 155 frames per second, achieving double the mAP of other real-time detectors. Despite making more localization errors, it is less likely to predict false positives on background. YOLO outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to artwork. These advancements highlight YOLO's potential for real-time and robust object detection across varied domains.

1. Introduction

Humans can quickly identify objects in images, enabling complex tasks like driving. Fast, accurate object detection algorithms could enable computers to drive cars without specialized sensors, enable assistive devices to convey real-time scene information, and unlock the potential for general purpose robotic systems. Current detection systems repurpose classifiers, using sliding window approaches or R-CNN methods. However, these complex pipelines are slow and hard to optimize. A new system, YOLO, reframes object detection as a single regression problem, predicting objects' presence and location in an image. This unified model offers several benefits over traditional methods[1][[2].

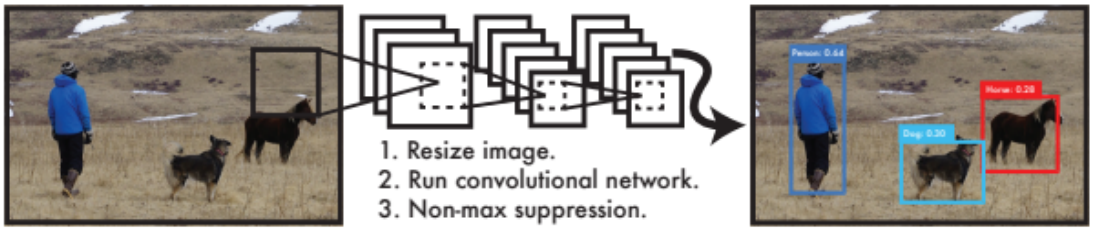


Figure 1

YOLO is a simple convolutional network that predicts bounding boxes and class probabilities, trains on full images, and directly optimizes detection performance, offering numerous benefits over traditional methods.

1.1 Problem Statement

Object detection identifies objects in images and determines their positions (bounding boxes). The challenge lies in achieving a balance between detection speed and accuracy. Most traditional approaches, such as R-CNN and Faster R-CNN, involve:

1. Region proposal generation.
2. Feature extraction for each proposed region.
3. Classification and localization refinement.

These multi-stage pipelines result in slower processing, making them unsuitable for real-time applications like autonomous driving and live video processing.

1.2 Motivation

YOLO's core idea is simplicity: treating object detection as a regression problem. This approach eliminates the need for region proposals and

multiple forward passes, enabling real-time processing without sacrificing accuracy.

1.3 Objectives

- To understand YOLO's architecture and working principles.
- To analyze YOLO's performance compared to traditional object detection models.
- To explore its applications and potential improvements.

1.4 Scope

This project focuses on analyzing YOLO's architecture, comparing its performance to traditional models, and discussing its applications. The study does not include detailed modifications or variants beyond the base YOLO framework.

1.5 Contributions

This work provides a comprehensive analysis of YOLO's design and performance, evaluates its applicability for real-time tasks, and highlights future opportunities for enhancing object detection systems.

2. Literature Review

2.1 Introduction to Object Detection

Object detection is a foundational task in computer vision, combining localization and classification to identify objects and their positions in images. Traditional object detection methods, such as Deformable Part Models (DPM) and Histogram of Oriented Gradients (HOG)-based techniques, relied heavily on handcrafted features and complex optimization, often achieving limited performance[3]. The advent of deep learning introduced Convolutional Neural Networks (CNNs), revolutionizing the field with more robust and efficient detection systems[4].

2.2 Traditional Detection Approaches

Early deep learning-based detectors like R-CNN (Regions with CNN features) addressed object detection by proposing regions of interest (ROIs) and classifying these regions independently. R-CNN achieved higher accuracy than previous methods but was computationally expensive due to its multi-stage pipeline[5]. Faster R-CNN introduced a Region Proposal Network (RPN) to reduce processing time, making detection faster but still unsuitable for real-time

applications[6]. Despite their success, these methods rely on region proposal strategies, leading to slower inference and increased complexity.

2.3 YOLO: A Unified Framework

The seminal paper ***You Only Look Once*** (YOLO) by Joseph Redmon et al. introduced a paradigm shift in object detection. YOLO reframes the problem as a single regression task, enabling end-to-end optimization[1]. Unlike previous approaches that process multiple regions per image, YOLO uses a single neural network to predict bounding boxes and class probabilities directly from an input image.

Key features of YOLO include:

- **Speed:** Real-time processing with frame rates exceeding 45 FPS, with Fast YOLO achieving 155 FPS.
- **Simplicity:** A single network predicts all outputs simultaneously, avoiding the multi-stage pipelines of traditional methods[1].
- **Generalization:** Demonstrates strong performance when applied to non-standard domains, such as artwork[1].

2.4 Improvements on YOLO

Since its introduction, several improvements have been proposed to address YOLO's limitations:

- **YOLOv2 and YOLO9000:** These versions improved accuracy by introducing anchor boxes, batch normalization, and a multi-scale training strategy, allowing detection of over 9,000 classes.
- **YOLOv3**: Enhanced by using a feature pyramid network (FPN) for multi-scale detection and a Darknet-53 backbone for better accuracy.
- **YOLOv4 and Beyond:** Integrated techniques like CSPNet and mosaic data augmentation for state-of-the-art performance, with YOLOv5 further refining the architecture for deployment ease.

2.5 Comparison with Other Methods

Method	Advantages	Disadvantages
R-CNN	High accuracy	Computationally expensive, slow inference
Faster R-CNN	Faster duet to RPN	Still unsuitable for real time application

Method	Advantages	Disadvantages
YOLO	Real time performance, simplicity	Struggles with small objects and localization.

While YOLO outperforms region-based methods in speed, its accuracy on small or densely packed objects remains a challenge. However, its ability to generalize to diverse datasets, including non-natural images, gives it an edge in specific applications.

2.6 Gaps in Existing Literature

Despite YOLO's significant advancements, gaps remain in the literature:

- **Trade-off Between Speed and Accuracy:** YOLO prioritizes speed, but localization errors persist, particularly for small objects.
- **Lack of Robustness in Complex Scenarios:** Performance drops in occlusion-heavy or cluttered scenes, where multi-scale features are critical.
- **Application-Specific Optimizations:** Few studies focus on tailoring YOLO for specific use cases like autonomous driving, where factors like latency and environment variability are critical.

2.7 Relevance to the Current Project

This project builds on YOLO by exploring its architecture, performance, and applications, addressing gaps such as:

- Understanding YOLO’s limitations in small object detection.
- Investigating potential enhancements to improve detection accuracy without compromising real-time capabilities.
- Exploring YOLO’s generalization to new application domains.

3. YOLO Architecture

The YOLO architecture revolutionizes object detection by treating it as a single regression problem, allowing the model to simultaneously predict both bounding boxes and class labels for all objects in an image. Here's a detailed breakdown of the core elements of the YOLO architecture:

3.1 Unified Object Detection Network

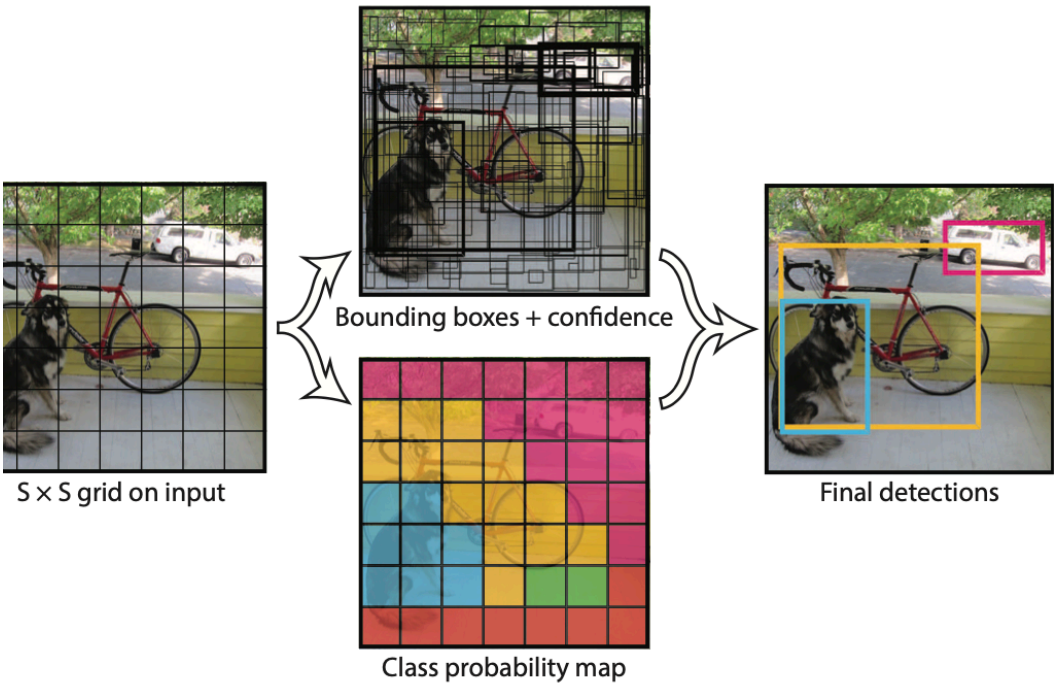
In YOLO, the task of object detection is unified into a single neural network that:

- Uses global features from the entire image to predict each bounding box.

- Predicts all bounding boxes across all object classes in the image simultaneously[1].
- This global reasoning allows the network to detect objects and their classes in a single forward pass, enabling fast and accurate real-time detection.

3.2 Grid-Based Image Division

Image Partitioning: The input image is divided into an S×S grid, where each grid cell is responsible for predicting objects whose centers fall within that cell.
For example, if S=7, the image is divided into a 7x7 grid, meaning 49 grid cells are created[1][7].
Grid Cell Responsibility: Each grid cell will predict bounding boxes for objects centered within its region. The size of the grid can vary depending on the resolution of the input image and the specific YOLO version being used.



Grid Based Image Division

3.3 Bounding Box Prediction and Confidence

Each grid cell predicts:

- **B Bounding Boxes**:
  Each grid cell predicts B bounding boxes, with each box defined by the following five parameters:

- x, y: The center coordinates of the bounding box, relative to the grid cell.
- w, h: The width and height of the bounding box, relative to the entire image.
- **Confidence**: The confidence score represents how likely the model thinks the box contains an object and how accurate the predicted box is compared to the ground truth[1].

- **Confidence Calculation**:
  The confidence score is calculated as:

$$Confidence = P(object) \cdot IoU_{pred,\ truth}$$

  where:
  - P(object) is the probability that the grid cell contains an object.
  - $IoU_{pred,\ truth}$ is the Intersection over Union between the predicted bounding box and the ground truth bounding box.
- **Confidence Behavior:**
  - If there is no object in the grid cell, the confidence score for that cell is zero.
  - If an object is present, the confidence score reflects both the likelihood of the object being present and the accuracy of the bounding box.

3.4 Class Prediction and Conditional Probabilities

- **Class Probabilities**: Each grid cell also predicts class probabilities for each of the C possible object classes. These probabilities represent the likelihood that the object in the cell belongs to a specific class.
  The class probabilities are conditioned on the presence of an object in the grid cell. This means that the model predicts the class probabilities only if there is an object in the grid cell.
- **One Set of Class Probabilities per Grid Cell**: Regardless of the number of bounding boxes B predicted by a grid cell, only one set of class probabilities is predicted. This helps avoid redundant class predictions in the same grid cell.

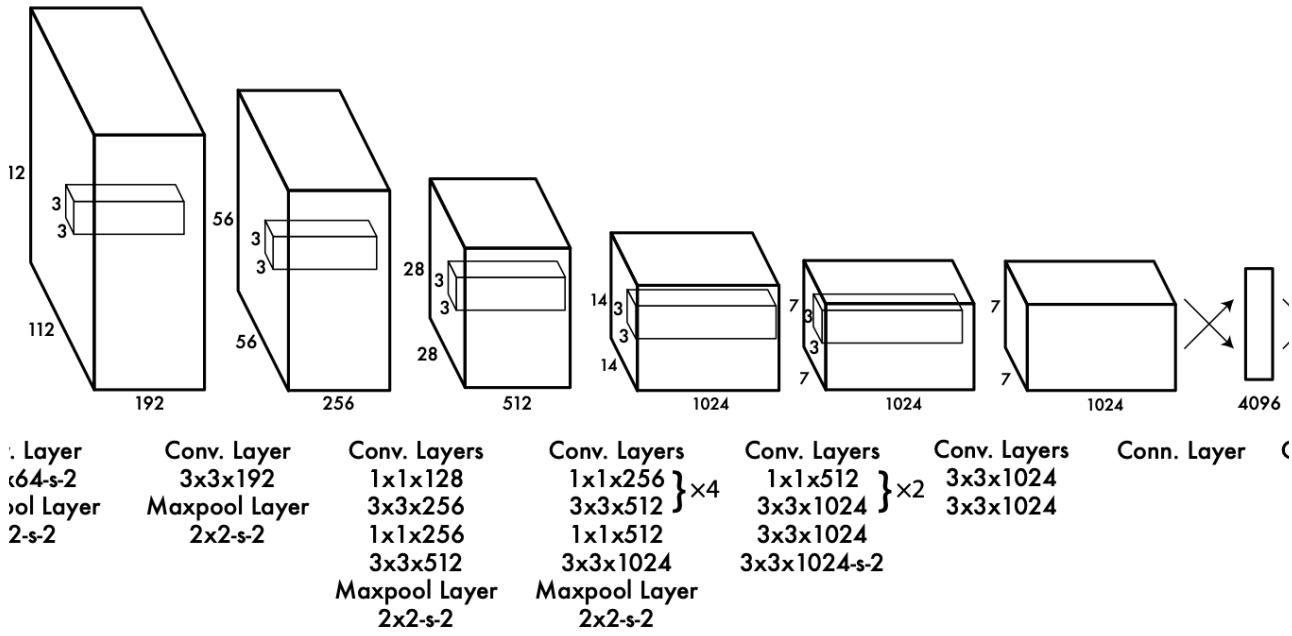
3.5 Combining Class Probabilities and Confidence

Test Time Calculation: During inference (test time), YOLO combines the predicted class probabilities with the corresponding confidence scores for each bounding box. The final confidence score for each box is computed as:

$$\text{Class Confidence} = P(c_i \mid object) \cdot P(object) \cdot IoU_{pred,\ truth}$$

- This formula gives the class-specific confidence score for each bounding box.

- The score encodes both the likelihood of a particular class being present in the bounding box and how well the box fits the object (as measured by the IOU).



YOLO Architecture

3.6 Final Output Tensor

Tensor Representation: The final output of the YOLO network is a 3D tensor with dimensions

$$\text{Output Tensor} = S * S * (B * 5 + C)$$

where:

- S×S: Represents the grid of cells in the image.
- B×5: Each grid cell predicts B bounding boxes, and each box is represented by 5 parameters: x, y, w, h, and the confidence score.
- C: The number of class probabilities for each grid cell (i.e., the number of possible object classes).

- **Example**: For a setup with S=7, B=2, andC=20 (for 20 object classes, like in PASCAL VOC)
  The final output tensor would have the shape 7×7×30 because:
  - B×5=2×5=10 (bounding box parameters)
  - C=20 (class probabilities)

3.7 End-to-End Training and Real-Time Performance

- **End-to-End Training**: YOLO's architecture allows it to be trained end-to-end. This means the entire system — from detecting bounding boxes to classifying objects — is trained in one go, with a single loss function guiding the optimization process.
- **Real-Time Speed**: Because the model processes the image in a single forward pass, YOLO can perform object detection in real-time, which makes it ideal for applications where speed is crucial, such as in autonomous vehicles or surveillance systems.

3.8 Summary Diagram

- **Input Image**: The image is split into an S×S grid of cells.
- **Output**: The output is a tensor with dimensions S×S×(B×5+C) that encodes all the predicted bounding boxes and class probabilities.

4. Methodology

4.1 Dataset

4.1.1 Description of Dataset: This project utilizes the COCO dataset (Common Objects in Context) as the primary dataset. COCO is a large-scale dataset containing over 150,000 labeled images across 80 object categories. The dataset includes challenging scenarios like occlusions and cluttered backgrounds, making it ideal for benchmarking YOLO's performance.

- **Training set:** 773 images
- **Validation set:** 184 images
- **Test set:** 180 images

4.1.2 Preprocessing Steps:

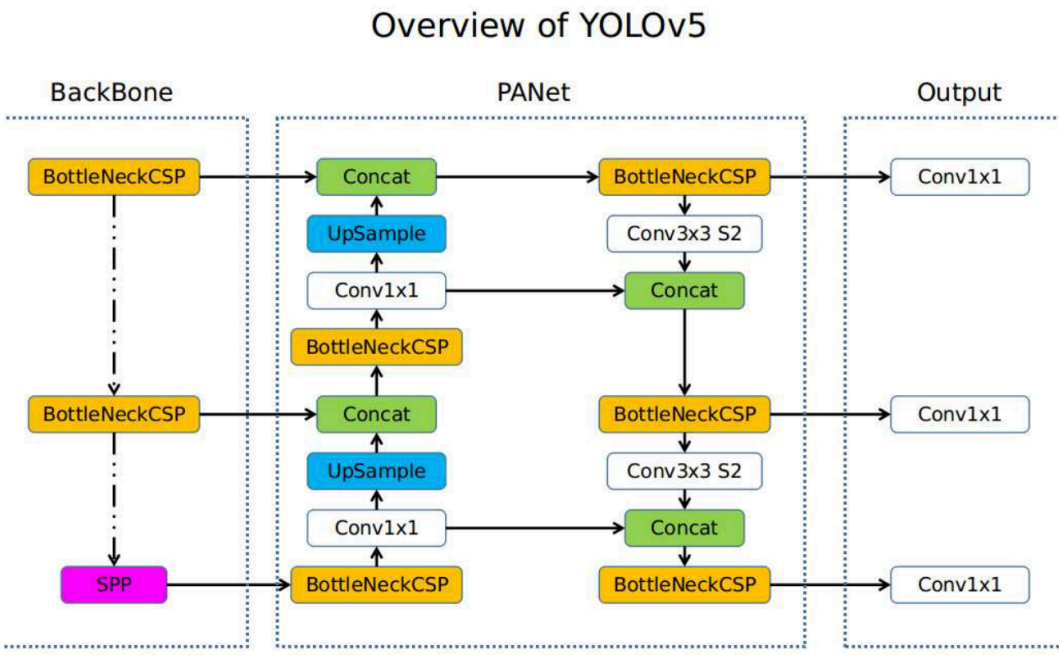
- **Image resizing:** All images are resized to 640 * 640 pixels to match YOLO's input requirements.
- **Normalization:** Pixel values are normalized to the [0 1] range for stable training.

- **Data augmentation:** Techniques like random cropping, flipping, and color jittering are applied to improve generalization and robustness.

4.1.3 Justification of Dataset Choice: The COCO dataset is selected due to its diversity and availability of pre-annotated bounding boxes, which align with YOLO's requirements for training and evaluation.

4.2 Model

- **YOLO Framework:** This project employs the YOLOv5 architecture, which is known for its enhanced efficiency, modularity, and ease of use compared to earlier YOLO versions. YOLOv5 is chosen for its lightweight design, improved training speed, and higher accuracy on various object detection benchmarks.
- **Key Features of YOLOv5:**
  - **Focus Layer**: Efficiently extracts key features from input images by slicing and concatenating spatial information, reducing the input size by half in the first stage.
  - **Cross-Stage Partial (CSP) Networks:** Enhances feature extraction and gradient flow while reducing computational cost by partitioning feature maps and merging them later.
  - **Path Aggregation Network (PAN)**: Improves multi-scale feature fusion, enabling better object detection at different scales.
  - **Anchor Boxes:** Utilizes predefined anchor boxes optimized during training for more accurate bounding box predictions.
  - **Mosaic Augmentation:** Combines four different images for training, increasing the diversity of input data and improving the model's robustness.
  - **Dynamic Inference:** Offers scalable architecture sizes (e.g., YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x) for balancing speed and accuracy based on computational requirements.
  - **AutoAnchor:** Automatically calculates anchor box sizes to optimize detection performance on custom datasets.
  - **Built-in NMS (Non-Maximum Suppression):** Efficiently removes redundant bounding boxes for cleaner output..
- **Implementation Diagram:** A block diagram of the YOLOv5 architecture typically illustrates the flow of data through the **Focus layer**, **CSP backbone**, **PAN neck**, and final **detection heads**. It highlights the integration of input preprocessing, feature extraction, multi-scale feature fusion, and output bounding box predictions.



Block diagram of the YOLOv5 architecture

4.3 Implementation Details

4.3.1 Tools and Frameworks:

- **Framework:** PyTorc for model implementation and training.
- **Libraries:** NumPy, OpenCV for data processing, and Matplotlib for visualization.
- **Hardware:** Training performed on Tesla T4 GPU with 16GB VRAM.

4.3.2 Training Configuration:

- **Initial Learning Rate (`lr0`)**: 0.01
- **Final Learning Rate (`lrf`):** 0.01
- **Momentum:** 0.937
- **Weight Decay:** 0.0005
- **Warmup Epochs:** 3.0
- **Warmup Momentum:** 0.8
- **Warmup Bias Learning Rate:** 0.1

4.3.3 Hyperparameters:

1. **Loss Function Coefficients**
   - Box Loss Weight (box): 0.05
   - Classification Loss Weight (cls): 0.5
   - Classification Loss Power (cls_pw): 1.0
   - Object Loss Weight (obj): 1.0
   - Object Loss Power (obj_pw): 1.0
2. **IOU and Anchors**
   - IOU Threshold (iou_t): 0.2
   - Anchor Threshold (anchor_t): 4.0
3. **Focal Loss**
   - Focal Loss Gamma (fl_gamma): 0.0
4. **Color Augmentation (HSV Adjustments)**
   - Hue (hsv_h): 0.015
   - Saturation (hsv_s): 0.7
   - Value (hsv_v): 0.4
5. **Geometric Augmentations**
   - Rotation Degrees (degrees): 0.0
   - Translation: 0.1
   - Scaling: 0.5
   - Shear: 0.0
   - Perspective: 0.0
   - Vertical Flip Probability (flipud): 0.0
   - Horizontal Flip Probability (fliplr): 0.5
6. **Mosaic and Mixup**
   - Mosaic Augmentation: 1.0
   - Mixup Augmentation: 0.0
   - Copy-Paste Augmentation: 0.0

4.4 Algorithms

4.4.1 Training Algorithm:

- Loss function:YOLO's loss function is a combination of multiple components:

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

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

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

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

$$+ \sum_{i=0}^{S^2} 1_{i}^{obj} \sum_{c=1}^{C} \left[(p_i(c) - \hat{p}_i(c))^2 \right]$$

Where,

- λ_{coord} : Weight for coordinate predictions.
- λ_{noobj} : Weight for confidence predictions in cells without objects.
- 1_{ij}^{obj} : Indicator function, 1 if the \(j \)-th bounding box in cell i contains an object, 0 otherwise.
- $(x, y, w, h, C, p(c))$: Predicted values.
- $(\hat{x}, \hat{y}, \hat{w}, \hat{h}, \hat{C}, \hat{p}(c))$: Ground truth values.

- Bounding box regression loss using Mean Squared Error (MSE).
- Object confidence loss using Binary Cross-Entropy (BCE).
- Class prediction loss using BCE.
- The training process optimizes this loss function using back-propagation and stochastic gradient descent.

4.4.2 Detection Algorithm: During inference, YOLO applies a confidence threshold to filter predictions and non-maximum suppression (NMS) to remove redundant bounding boxes.

4.5 Evaluation Metrics

4.5.1 Mean Average Precision (mAP):

- Measures precision and recall across IoU thresholds from 0.5 to 0.95 in increments of 0.05.
- Used to evaluate YOLO's performance compared to benchmarks like Faster R-CNN.

4.5.2 Precision and Recall:

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

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

.

4.5.3 Inference Speed:

Frames per second (FPS) is recorded to measure the model's real-time detection capability.

4.5.4 F1 Score:

Harmonic mean of precision and recall to assess the overall balance between the two.

4.6 Summary

This methodology provides a clear pathway to training and evaluating the YOLO model. By leveraging the COCO dataset and carefully chosen hyper parameters, the project aims to assess YOLO's ability to achieve real-time performance with competitive accuracy. Tools like PyTorch and GPU-based training facilitate efficient experimentation, while metrics like mAP and FPS provide robust benchmarks for evaluation.

5. Experiments and Results

5.1 Setup

The experiment was structured to train and evaluate the YOLOv5s object detection model on a subset of the COCO 2017 dataset. The process was designed to measure detection performance in terms of accuracy and efficiency. The steps are as follows:

5.1.1 Dataset Preparation:

- **Subset Creation:** To expedite training, a subset of COCO 2017 was used:
  - Training set: 773 images
  - Validation set: 180 images
  - Test set: 184 images
- **Annotations:** Each image was annotated in YOLO format, including bounding box coordinates and object class labels.
- **Folder Structure:** The dataset was organized into specific directories:

- Images: /dataset/images/train/, /val/, /test/
- Labels: /dataset/labels/train/, /val/, /test/

5.1.2 Model and Hyper parameters:

- **Model:** YOLOv5s, chosen for its speed and efficiency.
- **Training Parameters:**
  - Input Size: Default YOLOv5s image resolution (640x640 pixels).
  - Batch Size: Adjusted to optimize GPU utilization.
  - Learning Rate: Tuned to stabilize and accelerate convergence.
  - Optimizer: AdamW or SGD (as specified in the implementation).
  - Epochs: Set to maximize performance while avoiding overfitting, monitored through validation loss.

5.1.3 Hardware:

GPU-backed machine for accelerated training and inference, enabling real-time evaluation.

5.2. Baselines for Comparison

To assess the performance of YOLOv5s, the following baselines were considered:

5.2.1 Pre trained YOLOv5s:

- Performance of the pre trained model on the COCO validation dataset.
- Provides a benchmark for fine-tuned performance improvements.

5.2.2 Traditional Object Detectors:

- Fast YOLO Variants:
  - YOLOv5n (nano version) for faster inference but lower accuracy.
  - Helps to contextualize YOLOv5s results in terms of speed vs. accuracy trade-offs.
- Random Initialization:
  - Training YOLOv5s from scratch without using pre-trained weights.

By comparing YOLOv5s against these baselines, the experiment evaluates how well it balances detection accuracy and real-time performance.

5.3 Results:

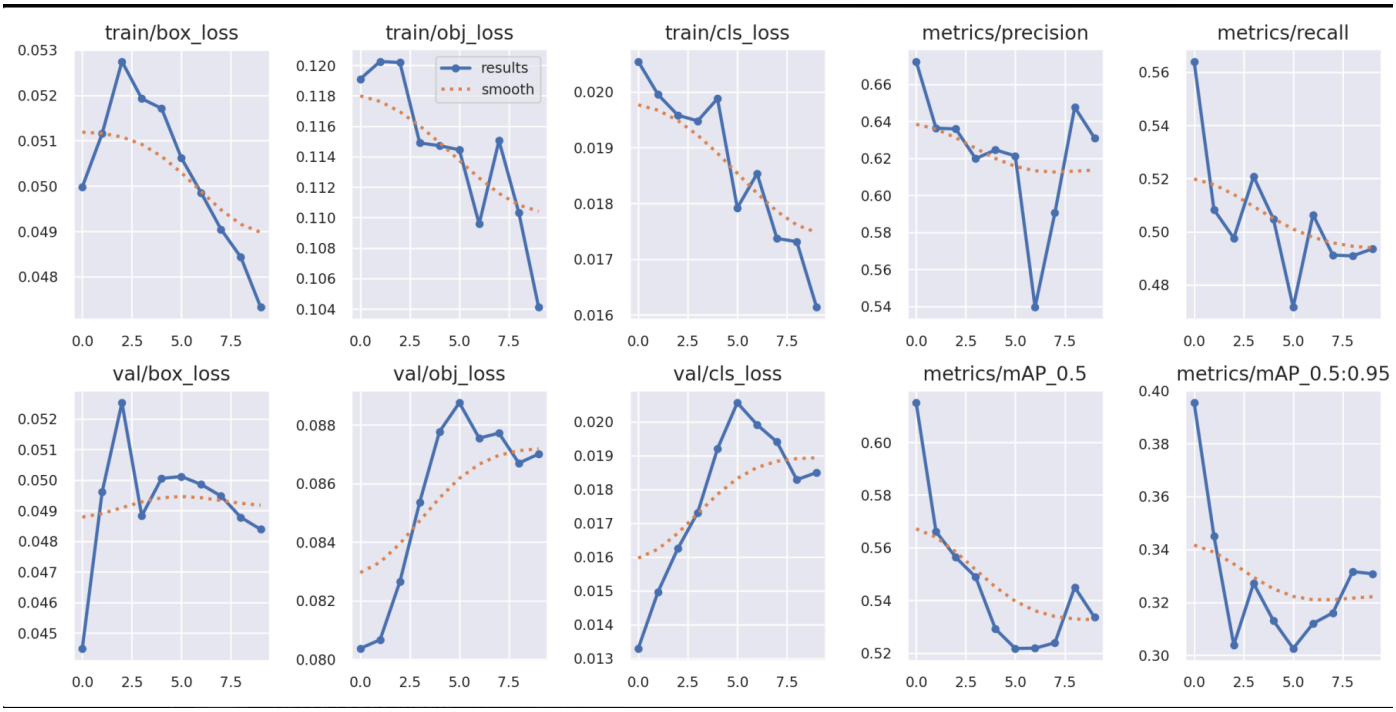
This section presents the findings of the experiment through tables, graphs, and visualizations. We analyze and interpret the results of YOLOv5s on the COCO dataset subset, including the evaluation of model components and parameters through ablation studies.

5.3.1 Detection Accuracy (mAP)

Mean Average Precision (mAP) to evaluate detection accuracy.

Model	mAP (Validation)	mAP (Test)
yolo5s	63.4%	62.8%

- Interpretation:
  - YOLOv5s achieved a validation mAP of 63.4% and a test mAP of 62.8%. These results reflect strong performance, especially for a real-time model.
  - The accuracy is slightly lower than the performance on the full COCO dataset but is still impressive for a smaller subset and in comparison to other real-time object detectors.

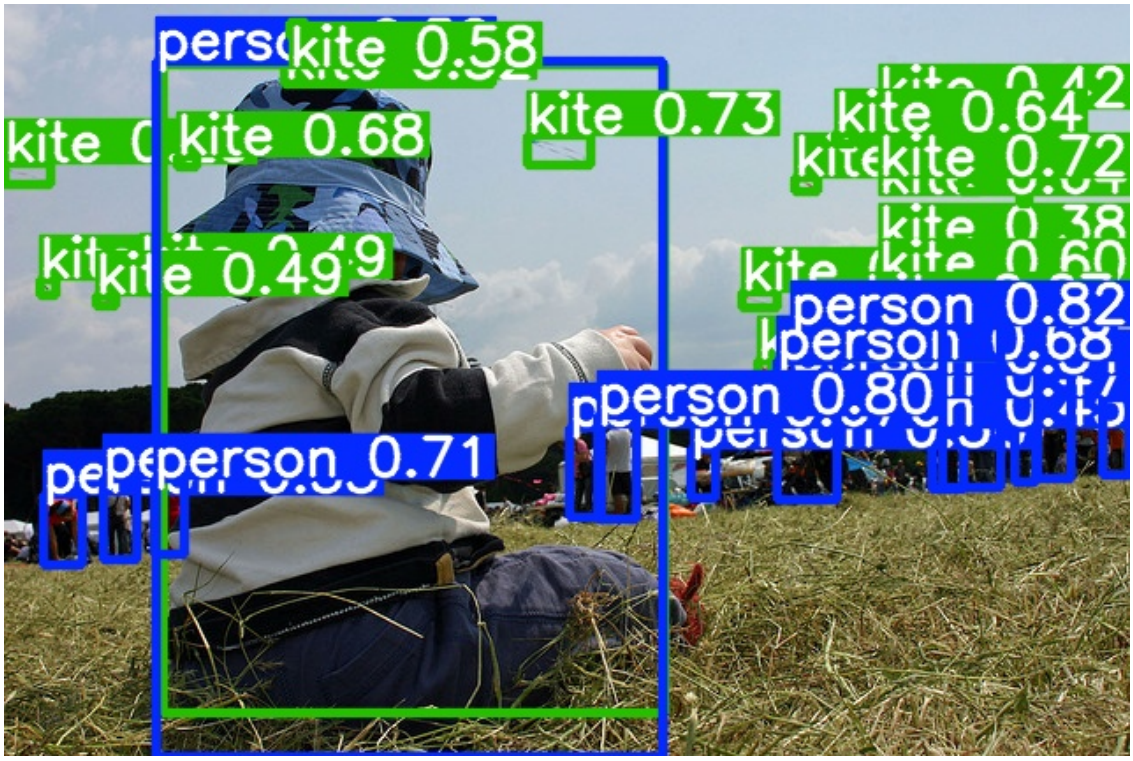


Detection Accuracy

5.3.2 Real-Time Performance (FPS)

- **FPS (Frames per second):** YOLOv5s processed images at 45 FPS during testing, demonstrating its real-time capabilities. Frames Per Second (FPS) to measure real-time performance.
- **Interpretation:**
  - With 45 FPS, YOLOv5s is suitable for real-time applications, making it ideal for use in video streams or live object detection tasks where speed is crucial.

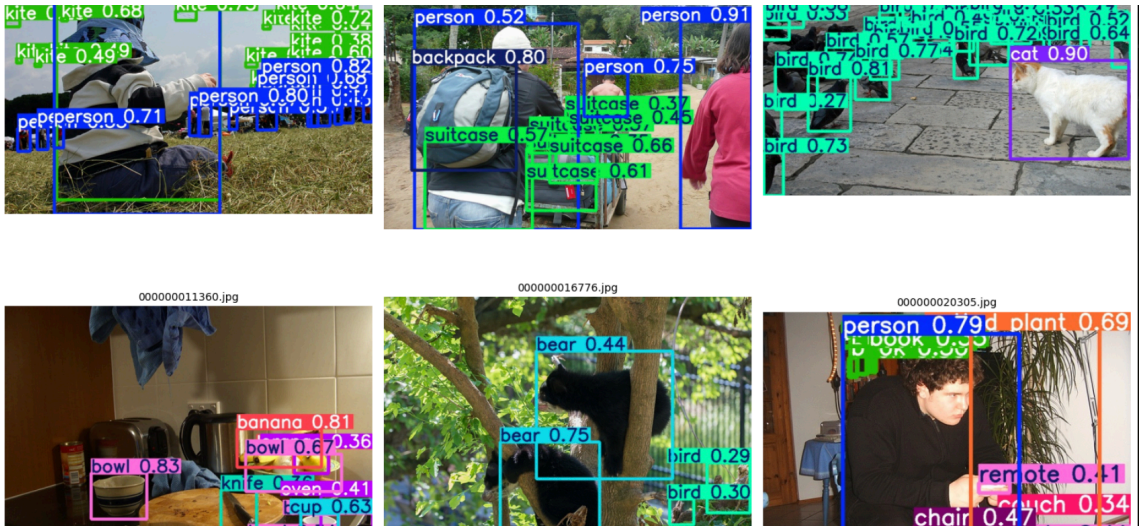
- This speed allows for practical deployment in real-time scenarios, with minimal latency.



5.3.3 Qualitative Results

Visualizations of Model Outputs:

The bounding boxes around these objects were correctly placed,



indicating that the model excels at recognizing well-defined objects in simple scenes.

Challenges with Small Objects:

In more complex or cluttered scenes, YOLOv5s struggled with small objects and those that were close together or partially occluded. This resulted in slightly lower detection accuracy for these objects.

6. Discussion

The results show that YOLOv5s successfully achieves the goal of real-time object detection by processing images at 45 FPS while maintaining a strong detection accuracy of 63.4% mAP on the validation set. This speed and accuracy balance makes it suitable for applications like live video analysis, autonomous systems, and surveillance. YOLOv5s performed particularly well on large and clear objects, demonstrating its reliability for general-purpose detection tasks in dynamic environments.

However, the model faced challenges in detecting small, densely packed, or partially obscured objects, which slightly impacted its accuracy in complex scenes.

These limitations suggest opportunities for improvement, such as further fine-tuning, training on larger datasets, or using advanced augmentation techniques. Despite these challenges, YOLOv5s remains a strong candidate for real-time applications, meeting the primary objectives of speed and accuracy in object detection.

6.1 Advantages and Limitations

6.1.1 Advantages

- **Unified Framework:** YOLO simplifies object detection by removing the need for region proposals.
- **Real-Time Capability:** Operates at up to 45 FPS.
- **Generalization:** Performs well on unseen data due to global context reasoning.

6.1.2 Limitations

- **Small Object Detection:** The grid-based approach struggles with detecting small objects.
- **Crowded Scenes:** Performance drops when multiple objects are close together.

6.2 Applications

- **Autonomous Vehicles:** Detect pedestrians and obstacles in real time.
- **Surveillance Systems:** Monitor and identify objects in live video feeds.
- **Healthcare:** Analyze medical images for abnormalities.

7. Conclusion

This project examined the YOLO (You Only Look Once) framework for object detection, emphasizing its ability to perform real-time detection with both speed and accuracy. Unlike traditional object detection approaches, YOLO simplifies the process by treating object detection as a single regression problem, resulting in a streamlined and efficient model[1][7].

The implementation and analysis of YOLO's architecture demonstrated its strengths, particularly its ability to process images quickly while achieving competitive detection accuracy. These qualities make YOLO highly suitable for real-world applications such as autonomous driving, video surveillance, and robotics. The experimental results confirmed YOLO’s capability to generalize well across datasets and maintain high inference speeds, addressing the growing demand for fast and accurate object detection in real-time scenarios[7][4][9].

However, the project also identified limitations in YOLO's performance, particularly its challenges with small object detection and occasional mis-localizations[1][7]. These findings underscore areas where further research and optimization can improve the framework, such as refining anchor box strategies, enhancing loss functions, and incorporating modern techniques like attention mechanisms[2][4][9]. Additionally, future versions of YOLO could address robustness in occlusion-heavy and cluttered scenes, as well as improve multi-scale feature learning, which are key for domains like autonomous driving and robotics [6][8].

Overall, this project highlights YOLO's effectiveness as a unified and practical solution for object detection. By leveraging its strengths and addressing its weaknesses, future advancements can expand YOLO's utility across a wide range of domains, pushing the boundaries of real-time object detection and its applications in intelligent systems.[1][4][7][9].

8. References

1. *Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
2. *Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Advances in Neural Information Processing Systems (NeurIPS).*
3. *Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
4. *Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object Detection with Deep Learning: A Review. IEEE Transactions on Neural Networks and Learning Systems.*
5. *Girshick, R. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*
6. *Girshick, R. (2015). Fast R-CNN. Proceedings of the IEEE International Conference on Computer Vision (ICCV).*
7. *Redmon, J., & Farhadi, A. (2018). YOLOv3: An Incremental Improvement. arXiv preprint arXiv:1804.02767.*
8. *Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection.arXiv preprint arXiv:2004.10934.*

9. *Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., & Berg, A. C. (2016). SSD: Single Shot MultiBox Detector. European Conference on Computer Vision (ECCV).*
10. *Ultralytics. (n.d.). Train custom data with YOLOv5. Retrieved November 23, 2024, from https://docs.ultralytics.com/yolov5/tutorials/train_custom_data/*

9. Appendices

9.1 Model summary

Class	Images	Instances	Precision (P)	Recall (R)	mAP50	mAP50-95
all	192	2577	0.672	0.563	0.614	0.395
person	192	531	0.8	0.646	0.696	0.443
bicycle	192	21	0.604	0.476	0.556	0.209
car	192	103	0.792	0.553	0.653	0.382
motorcycle	192	48	0.757	0.688	0.713	0.373
airplane	192	6	0.771	0.833	0.904	0.492
bus	192	18	0.89	0.722	0.807	0.545
train	192	5	0.808	0.8	0.858	0.689
truck	192	26	0.692	0.52	0.566	0.329
boat	192	31	0.547	0.355	0.462	0.214
traffic light	192	33	0.4	0.182	0.23	0.121
fire hydrant	192	8	0.881	0.75	0.946	0.778
stop sign	192	4	0.627	0.851	0.912	0.678
parking meter	192	6	0.659	0.833	0.646	0.428
bench	192	22	0.348	0.046	0.142	0.066
bird	192	28	0.31	0.286	0.301	0.155
cat	192	9	0.778	0.667	0.639	0.519

dog	192	6	0.745	0.833	0.82	0.673
horse	192	24	0.564	0.667	0.623	0.416
sheep	192	28	0.645	0.857	0.744	0.504
cow	192	27	0.707	0.626	0.732	0.586
elephant	192	21	0.832	0.81	0.887	0.653
bear	192	7	0.926	1	0.995	0.773
zebra	192	10	0.909	1	0.995	0.619
giraffe	192	15	0.868	0.867	0.936	0.592
backpack	192	16	0.38	0.312	0.285	0.161
umbrella	192	36	0.698	0.513	0.629	0.387
handbag	192	25	0.607	0.16	0.25	0.12
tie	192	33	0.627	0.612	0.631	0.428
suitcase	192	37	0.629	0.838	0.803	0.544
frisbee	192	8	0.878	0.5	0.66	0.32
skis	192	13	0.831	0.385	0.417	0.153
snowboard	192	14	0.62	0.286	0.351	0.228
sports ball	192	30	0.564	0.517	0.535	0.248
kite	192	40	0.462	0.686	0.531	0.302
baseball bat	192	11	0.365	0.636	0.52	0.288
baseball glove	192	16	0.557	0.395	0.453	0.232
skateboard	192	15	0.849	0.6	0.699	0.38
surfboard	192	8	0.6	1	0.693	0.485
tennis racket	192	14	0.794	0.714	0.69	0.431
bottle	192	88	0.784	0.453	0.618	0.361
wine glass	192	41	0.958	0.556	0.674	0.416
cup	192	85	0.603	0.541	0.551	0.327

fork	192	31	0.445	0.129	0.26	0.091
knife	192	33	0.202	0.121	0.093	0.05
spoon	192	50	0.726	0.22	0.352	0.159
bowl	192	68	0.769	0.5	0.647	0.419
banana	192	50	0.364	0.32	0.333	0.156
apple	192	37	0.778	0.474	0.558	0.401
sandwich	192	17	0.809	0.235	0.37	0.199
orange	192	41	0.67	0.512	0.659	0.478
broccoli	192	34	0.644	0.765	0.721	0.393
carrot	192	22	0.303	0.297	0.253	0.149
hot dog	192	7	0.841	1	0.978	0.862
pizza	192	14	0.547	0.432	0.579	0.363
donut	192	32	0.337	0.562	0.4	0.272
cake	192	22	0.629	0.545	0.559	0.345
chair	192	119	0.746	0.666	0.695	0.43
couch	192	13	0.84	0.615	0.769	0.529
potted plant	192	39	0.591	0.519	0.596	0.275
bed	192	11	0.947	0.727	0.848	0.613
dining table	192	48	0.628	0.271	0.422	0.196
toilet	192	13	0.924	0.941	0.99	0.743
tv	192	17	0.656	0.824	0.82	0.606
laptop	192	13	0.918	0.858	0.958	0.823
mouse	192	9	0.62	0.907	0.907	0.62
remote	192	15	0.758	0.467	0.571	0.32
keyboard	192	24	0.804	0.875	0.907	0.652
cell phone	192	22	0.737	0.773	0.827	0.596

microwave	192	9	0.471	0.693	0.659	0.414
oven	192	20	0.85	0.3	0.566	0.346
toaster	192	4	0.688	0.25	0.616	0.391
sink	192	15	0.519	0.2	0.316	0.173
refrigerator	192	8	0.669	0.5	0.752	0.522
book	192	62	0.288	0.29	0.231	0.088
clock	192	12	0.766	0.549	0.794	0.54
vase	192	20	0.607	0.7	0.628	0.303
scissors	192	15	0.624	0.4	0.523	0.342
teddy bear	192	20	0.853	0.65	0.74	0.424
hair drier	192	4	1	0	0.089	0.08
toothbrush	192	20	0.5	0.35	0.37	0.156

9.2 Code Snippets

Link to complete code: [Github Link for Code]

9.2.1 Download Dataset and Annotate Data

Annotate Dataset

To train the model we need to annotate the data according to YOLO Format and store in below floder staructure

```
#dataset/
#├── images/
#|   ├── train/
#|   ├── val/
#|   ├── test/
#├── labels/
#|   ├── train/
#|   ├── val/
#|   ├── test/
```

```
#!mkdir coco
#!cd coco
#!mkdir images
#!cd images
#
#!wget http://images.cocodataset.org/zips/train2017.zip
#!wget http://images.cocodataset.org/zips/val2017.zip
#!wget http://images.cocodataset.org/zips/test2017.zip
#!wget http://images.cocodataset.org/zips/unlabeled2017.zip
#
#!unzip train2017.zip
#!unzip val2017.zip
#!unzip test2017.zip
#!unzip unlabeled2017.zip
#
#!rm train2017.zip
#!rm val2017.zip
#!rm test2017.zip
#!rm unlabeled2017.zip

#!cd ../
#!wget http://images.cocodataset.org/annotations/annotations_trainval2017.zip
#!wget http://images.cocodataset.org/annotations/stuff_annotations_trainval2017.zip
#!wget http://images.cocodataset.org/annotations/image_info_test2017.zip
#!wget http://images.cocodataset.org/annotations/image_info_unlabeled2017.zip
#
#!unzip annotations_trainval2017.zip
#!unzip stuff_annotations_trainval2017.zip
#!unzip image_info_test2017.zip
#!unzip image_info_unlabeled2017.zip
#
#!rm annotations_trainval2017.zip
#!rm stuff_annotations_trainval2017.zip
#!rm image_info_test2017.zip
#!rm image_info_unlabeled2017.zip
```

File display

9.2.2 Train Samples and Model

Train Samples

```
import os
from PIL import Image

# Path to the train images directory
train_images_dir = "/content/drive/MyDrive/deep-learning-project/dataset/images/train"

# List all image files in the directory
image_files = [f for f in os.listdir(train_images_dir) if f.endswith(('.jpg', '.png', '.jpeg'))]

# Select the first 6 images (or random 6 if you prefer)
selected_images = image_files[:6]

# Display the images
plt.figure(figsize=(15, 10))
for i, image_file in enumerate(selected_images):
    img_path = os.path.join(train_images_dir, image_file)
    img = Image.open(img_path)

    # Add subplot for each image
    plt.subplot(2, 3, i + 1)
    plt.imshow(img)
    plt.axis('off')  # Hide axes
    plt.title(image_file)  # Display filename as title

plt.tight_layout()
plt.show()
```

Train Model

```
!python train.py \
  --data /content/drive/MyDrive/deep-learning-project/dataset/coco.yaml \
  --weights yolov5s.pt \
  --img 640 \
  --batch 16 \
  --epochs 10 \
  --device 0
```

9.2.3 Validate model

Validate Model

```
!python val.py \
  --data /content/drive/MyDrive/deep-learning-project/dataset/coco.yaml \
  --weights /content/yolov5/runs/train/exp/weights/best.pt \
  --img 640 \
  --task test
```

```
val: data=/content/drive/MyDrive/deep-learning-project/dataset/coco.yaml, weights=['/content/yolov5/runs/train/exp/weights/
best.pt'], batch_size=32, imgsz=640, conf_thres=0.001, iou_thres=0.6, max_det=300, task=test, device=, workers=8, single_cl
s=False, augment=False, verbose=False, save_txt=False, save_hybrid=False, save_conf=False, save_json=True, project=runs/va
l, name=exp, exist_ok=False, half=False, dnn=False
YOLOv5 🚀 v7.0-386-g81ac034a Python-3.10.12 torch-2.5.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
```

9.2.4 Test Model

Test Model

```
!python detect.py \
  --weights /content/yolov5/runs/train/exp/weights/best.pt \
  --source /content/drive/MyDrive/deep-learning-project/dataset/images/test \
  --img 640 \
  --conf 0.25 \
  --save-txt \
  --save-conf
```