Team The Mavericks

Josanth Smilan A 2347228
Kalpana N 2347229

Output Structure

Introduction:

In this process, we are training a YOLOv5 (You Only Look Once) model for object detection, a state-of-the-art real-time computer vision model used to detect objects in images and videos. YOLOv5 is a highly efficient and popular version of the YOLO series that offers good performance for various use cases. The training process begins by setting up the environment, installing necessary dependencies, configuring model settings, and running the training script. The training configuration includes selecting hyperparameters such as image size, batch size, and the number of epochs. We also use pre-trained weights (from `yolov5s.pt`) as a starting point, which helps the model learn faster and achieve better performance by leveraging prior knowledge from similar tasks. The model is trained using a custom dataset, which is specified in a `data.yaml` file. Once trained, the model can be used for real-time object detection tasks.

Steps for YOLOv5 Model Training:

1. **Set up the environment**: Install dependencies using `pip install -U -r requirements.txt` and clone the YOLOv5 repository.
2. **Prepare dataset**: Organize the dataset in YOLO format and create a `data.yaml` file specifying the dataset paths and class names.
3. **Configure model settings**: Choose the model architecture (`yolov5s`), set hyperparameters such as image size, batch size, and epochs.

4. **Train the model**: Run the training command using `python train.py --data data.yaml --cfg yolov5s.yaml --epochs 50 --batch-size 16 --img-size 640`.
5. **Monitor training**: Track training progress, loss, and mAP metrics using the generated logs.
6. **Save the model**: The trained model is saved to the `runs/train/exp` directory for further use.
7. **Test the model**: Run inference on test images using `python detect.py --weights best.pt --img-size 640 --source test/images/`.

Yaml Source File

```
train: C:/Users/Kalpana/Documents/CV/Hackathon/dataset1/dataset1/train/images
val: C:/Users/Kalpana/Documents/CV/Hackathon/dataset1/dataset1/valid/images
test: C:/Users/Kalpana/Documents/CV/Hackathon/dataset1/dataset1/test/images

nc: 1
names: ['LicensePlate']
```

Model Trained Output:

```
train: weights=yolov5s.pt, cfg=, data=C:/Users/Kalpana/Documents/CV/Hackathon/yolov5/data.yaml, hyp=data\hyps\hyp.scratch-low.yaml, epochs=5, batch_size=8, imgsz=
From https://github.com/ultralytics/yolov5
   81ac034a..882c35fc  master     -> origin/master
github:  YOLOv5 is out of date by 5 commits. Use 'git pull' or 'git clone https://github.com/ultralytics/yolov5' to update.
YOLOv5  v7.0-383-g1435a8ee Python-3.11.4 torch-2.5.1+cpu CPU

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.
Comet: run 'pip install comet_ml' to automatically track and visualize YOLOv5  runs in Comet
TensorBoard: Start with 'tensorboard --logdir runs\train', view at http://localhost:6006/
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/Arial.ttf to C:\Users\Kalpana\AppData\Roaming\Ultralytics\Arial.ttf...

  0%|          | 0.00/755k [00:00<?, ?B/s]
 17%|▉         | 128k/755k [00:00<00:01, 580kB/s]
 34%|███▍      | 256k/755k [00:00<00:01, 264kB/s]
 51%|█████     | 384k/755k [00:01<00:02, 184kB/s]
 68%|██████▊   | 512k/755k [00:03<00:01, 140kB/s]
 85%|████████▍ | 640k/755k [00:04<00:00, 123kB/s]
100%|██████████| 755k/755k [00:05<00:00, 129kB/s]
100%|██████████| 755k/755k [00:05<00:00, 146kB/s]
Overriding model.yaml nc=80 with nc=1

                from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Conv                      [3, 32, 6, 2, 2]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     18816  models.common.C3                        [64, 64, 1]
...
                Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 63/63 [03:18<00:00,  3.13s/it]
                Class     Images  Instances          P          R      mAP50   mAP50-95: 100%|██████████| 63/63 [03:18<00:00,  3.15s/it]
                  all       1008       1468      0.809      0.733      0.738      0.494
Results saved to runs\train\exp2
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

The process of Extracting the characters from the image

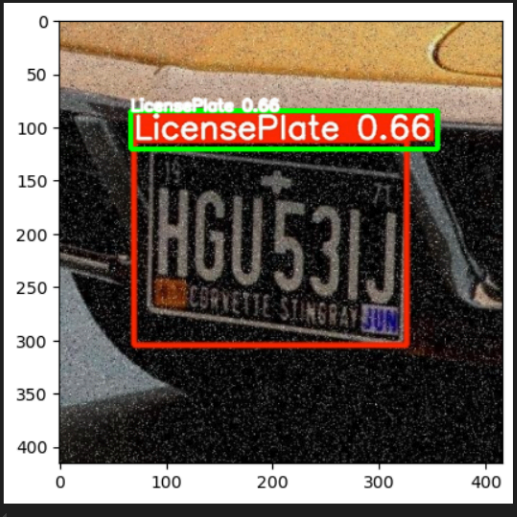
```
reader = easyocr.Reader(['en'])
result = reader.readtext(IMAGE_PATH)
result
```

49.9s

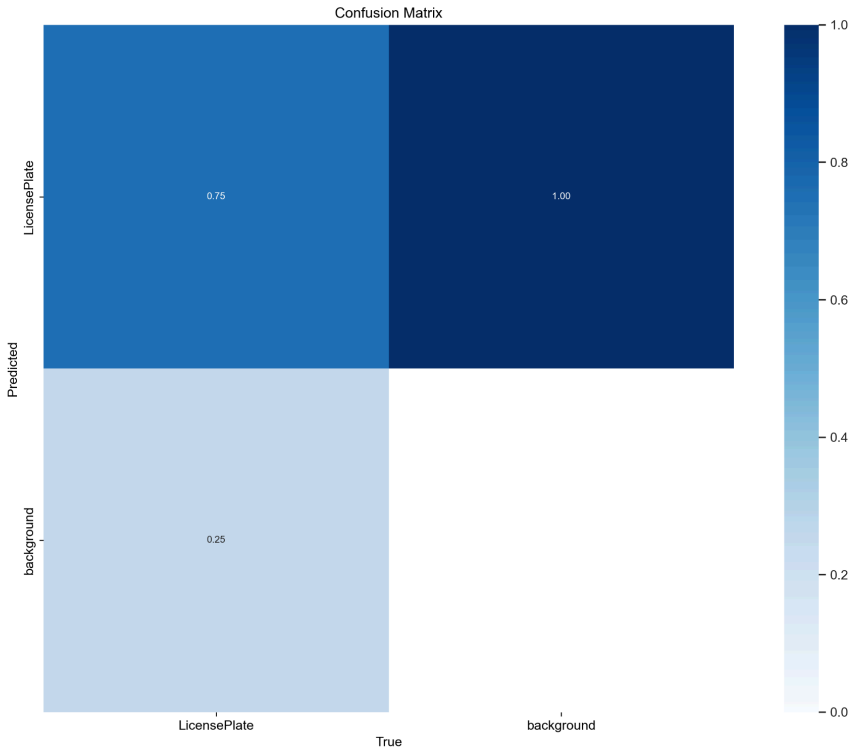
```
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is much faster with a GPU.
Downloading detection model, please wait. This may take several minutes depending upon your network connection.
Progress: |                                        -| 98.6% Complete
```

OCR (Object Character Detection) on License plate

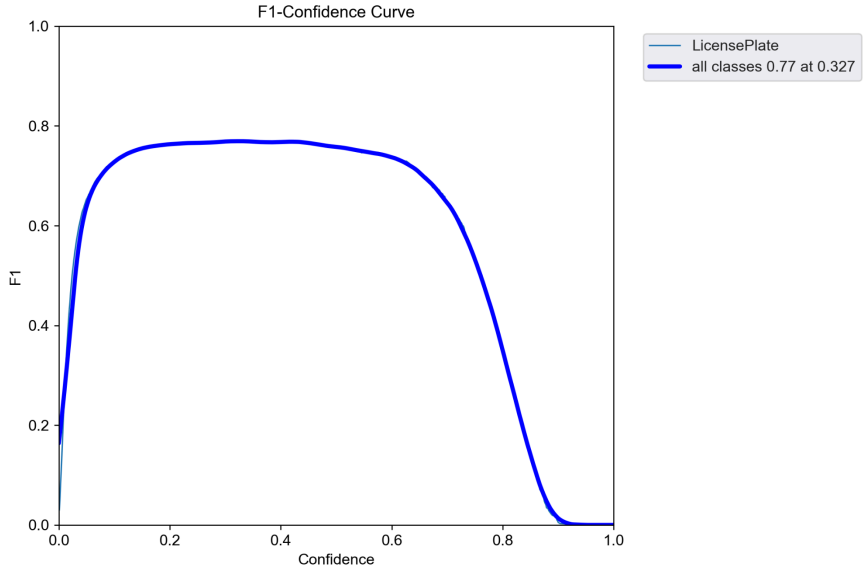
```
img = cv2.imread(image_path)
img = cv2.rectangle(img,top_left,bottom_right,(0,255,0),3)
img = cv2.putText(img,text,top_left, font, 0.5,(255,255,255),2,cv2.LINE_AA)
plt.imshow(img)
plt.show()
```



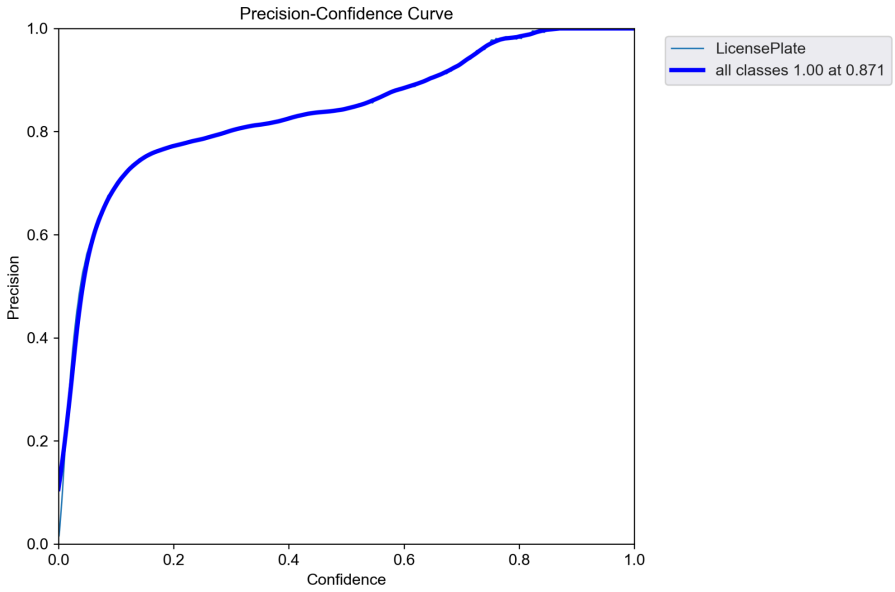
Confusion Matrix


Confusion Matrix
LicensePlate
background
Predicted
0.75
1.00
0.25
LicensePlate
background
True
1.0
0.8
0.6
0.4
0.2
0.0

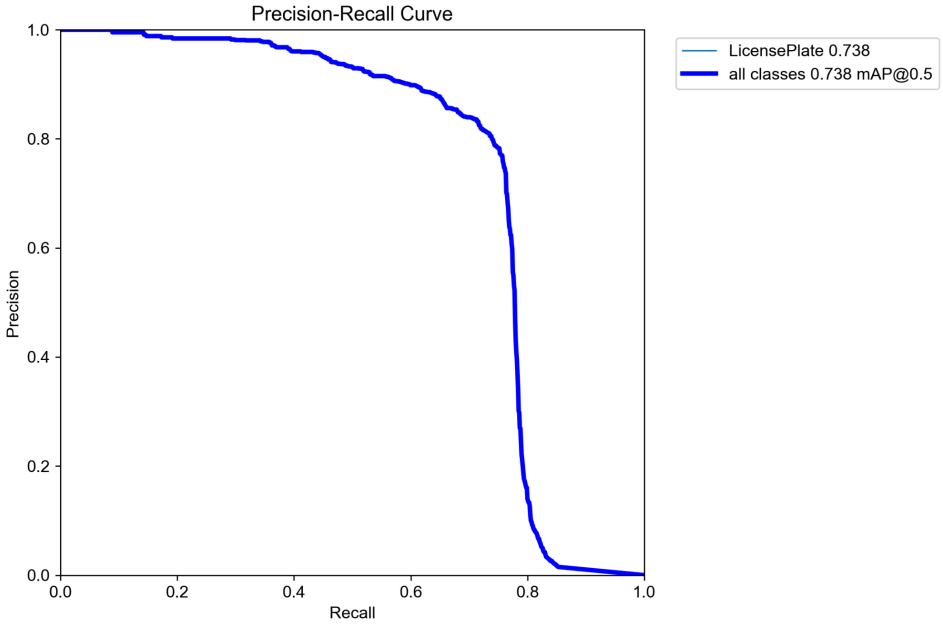
F1 - Curve


F1-Confidence Curve
LicensePlate
all classes 0.77 at 0.327
F1
Confidence
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

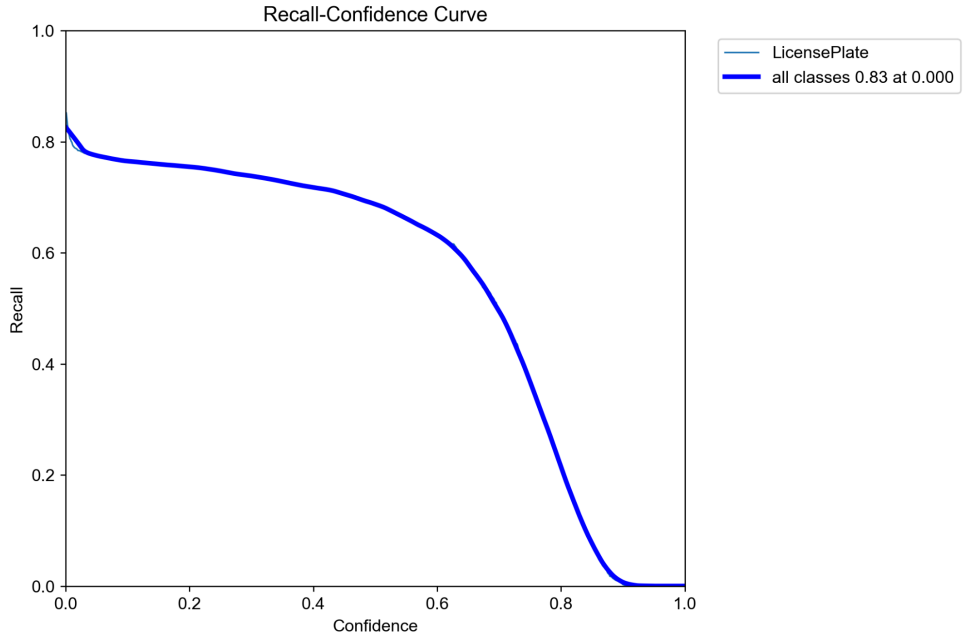
P Curve



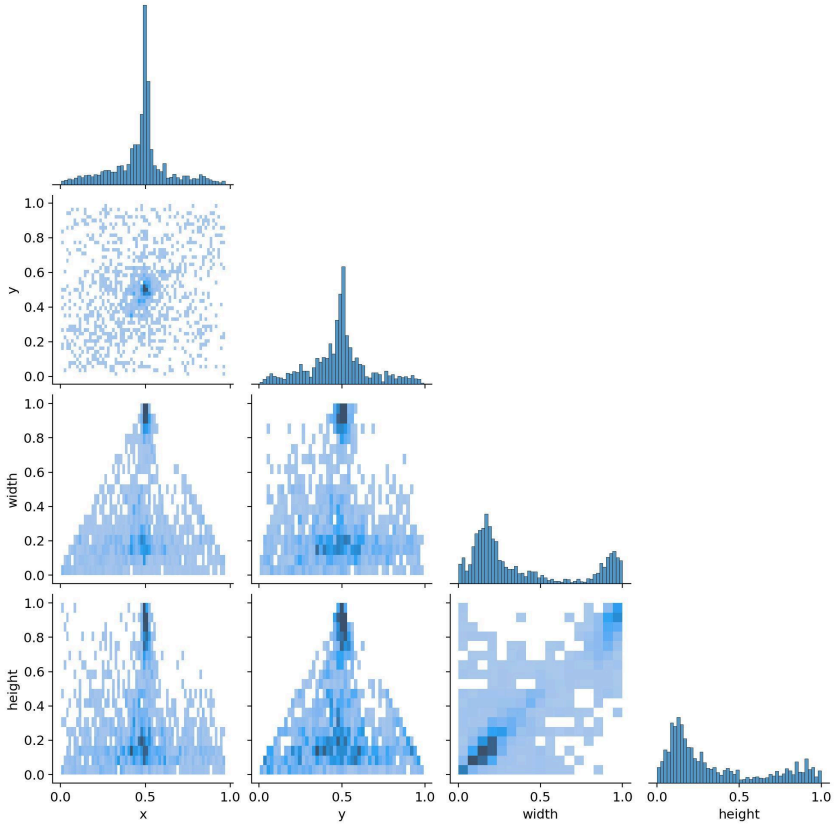
PR Curve



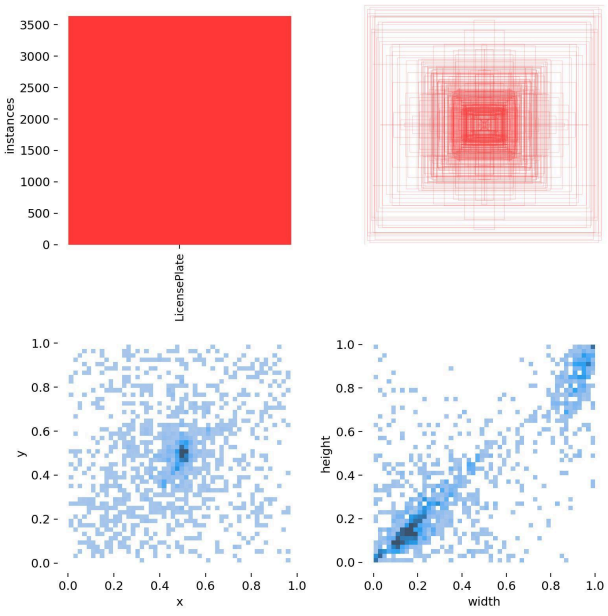
R Curve



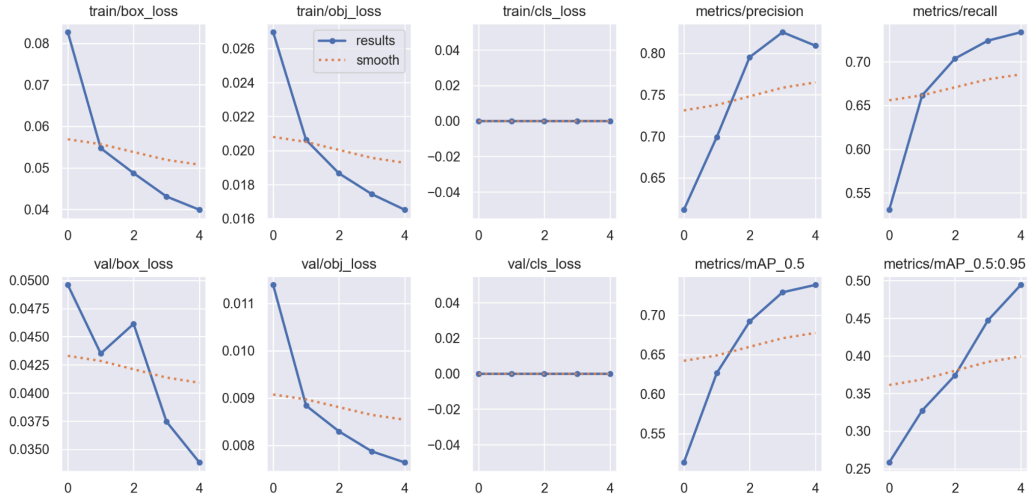
Labels Correlogram



Number Plate Labels Images Graphs

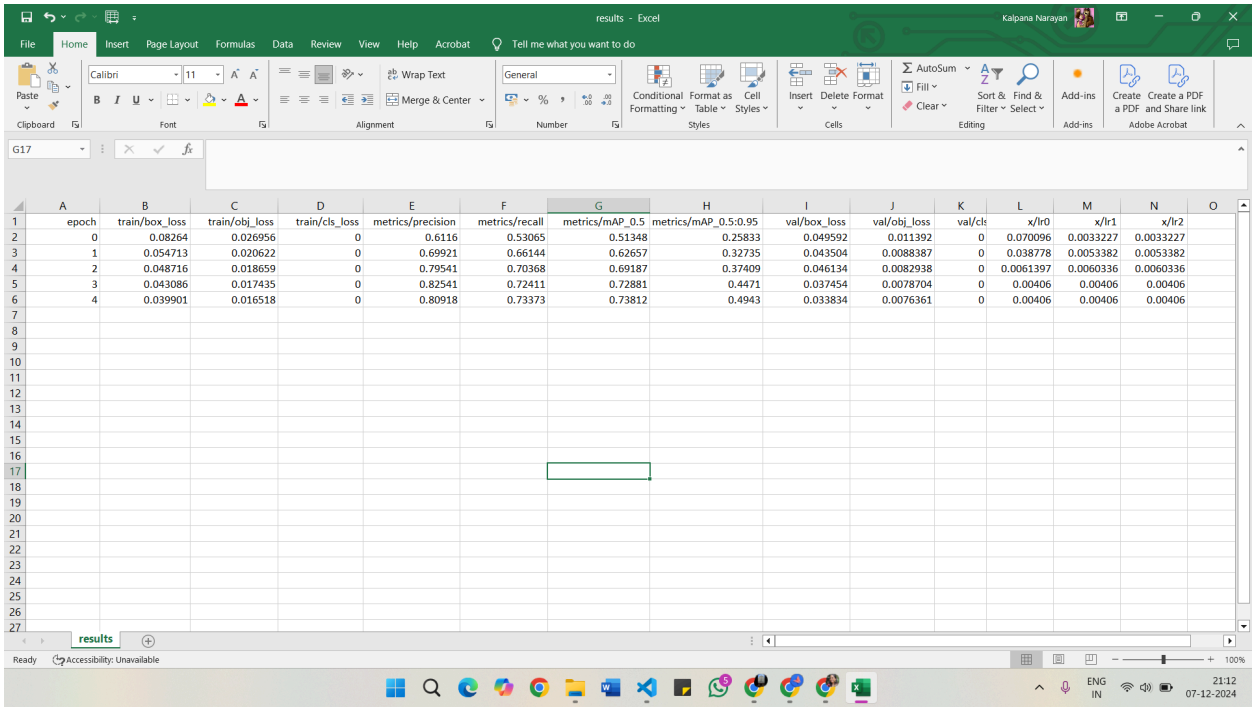


Results



Validation Output

Accuracy Results



epoch	train/box_loss	train/obj_loss	train/cls_loss	metrics/precision	metrics/recall	metrics/mAP_0.5	metrics/mAP_0.5:0.95	val/box_loss	val/obj_loss	val/cls	x/lr0	x/lr1	x/lr2
0	0.08264	0.026956	0	0.6116	0.53065	0.51348	0.25833	0.049592	0.011392	0	0.070096	0.0033227	0.0033227
1	0.054713	0.020622	0	0.69921	0.66144	0.62657	0.32735	0.043504	0.0088387	0	0.038778	0.0053382	0.0053382
2	0.048716	0.018659	0	0.79541	0.70368	0.69187	0.37409	0.046134	0.0082938	0	0.0061397	0.0060336	0.0060336
3	0.043086	0.017435	0	0.82541	0.72411	0.72881	0.4471	0.037454	0.0078704	0	0.00406	0.00406	0.00406
4	0.039901	0.016518	0	0.80918	0.73373	0.73812	0.4943	0.033834	0.0076361	0	0.00406	0.00406	0.00406

Conclusion:

The training of the YOLOv5 model involves several critical steps, including environment setup, dependency installation, configuration, and running the training process with proper hyperparameters. After training, the model learns to recognize objects in images, and the best-performing model is saved for further use. By fine-tuning a pre-trained YOLOv5 model on a custom dataset, we can leverage its powerful object detection capabilities for specific tasks. Monitoring the training process through logs and performance metrics such as loss and mAP is essential for evaluating the model's effectiveness. Once training is complete, the model is ready to be deployed for real-time object detection, providing efficient and accurate results.