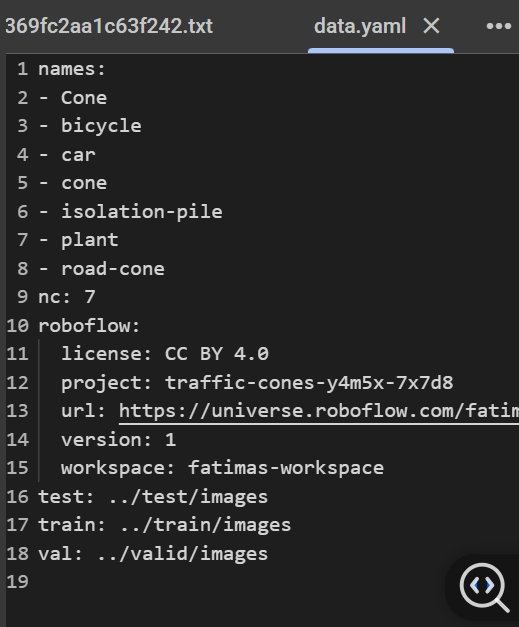
**Documentation of YOLO-v8 Detecting Traffic Cones**

**Observations data.yaml**



This is the **list of class labels** in my dataset.

Each class gets an **ID/index** starting from 0:

* 0 → Cone
* 1 → bicycle
* 2 → car
* 3 → cone ( duplicate of "Cone")
* 4 → isolation-pile
* 5 → plant
* 6 → road-cone

2. Nc

**Number of classes** in my dataset.

3. roboflow block

roboflow:

license: CC BY 4.0

project: traffic-cones-y4m5x-7x7d8

url: https://universe.roboflow.com/fatimas-workspace/traffic-cones-y4m5x-7x7d8/dataset/1

version: 1

workspace: fatimas-workspace

Metadata from Roboflow about my dataset:

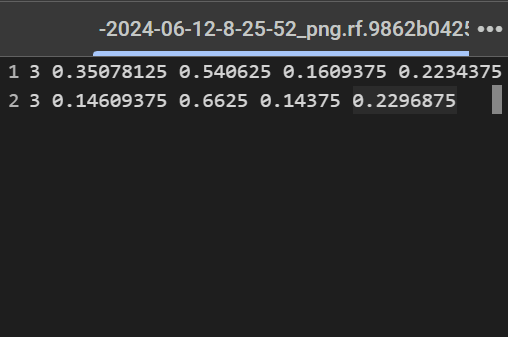
* license: dataset license.
* project: project name in Roboflow.
* url: link to dataset page.
* version: dataset version you exported.
* workspace: your Roboflow workspace.

4. Dataset splits

* These tell YOLO where to find the dataset images for each split:  
  + train: training images.
  + val: validation images (used for evaluation during training).
  + test: optional, used for testing after training.

Each folder should contain images + a labels/ folder (with .txt annotation files).

**Observation about trains/labels folder**



Each line corresponds to **one object in the image**. The format is:

<class\_id> <x\_center> <y\_center> <width> <height>

class\_id: the index of the object class (from your names: list in data.yaml).

* In my case 3 means the **4th class** (since indexing starts at 0). From your names, that’s cone.

x\_center: normalized x-coordinate of the bounding box center (value between 0 and 1, relative to image width).

y\_center: normalized y-coordinate of the bounding box center (value between 0 and 1, relative to image height).

width: normalized width of the bounding box (relative to image width).

height: normalized height of the bounding box (relative to image height).

**Why is it important to point the training command to data.yaml instead of directly to the image folders?**

**Class Mapping (names + indices)**

* The .yaml file contains the **class definitions** (names and nc).
* Without it, YOLO wouldn’t know how to map class IDs in your .txt annotation files to actual class names.

**Dataset Split Info**

* data.yaml tells YOLO **where the train, val, and test sets are located**.
* If you only give YOLO an image folder, it won’t know which images are for training vs validation.

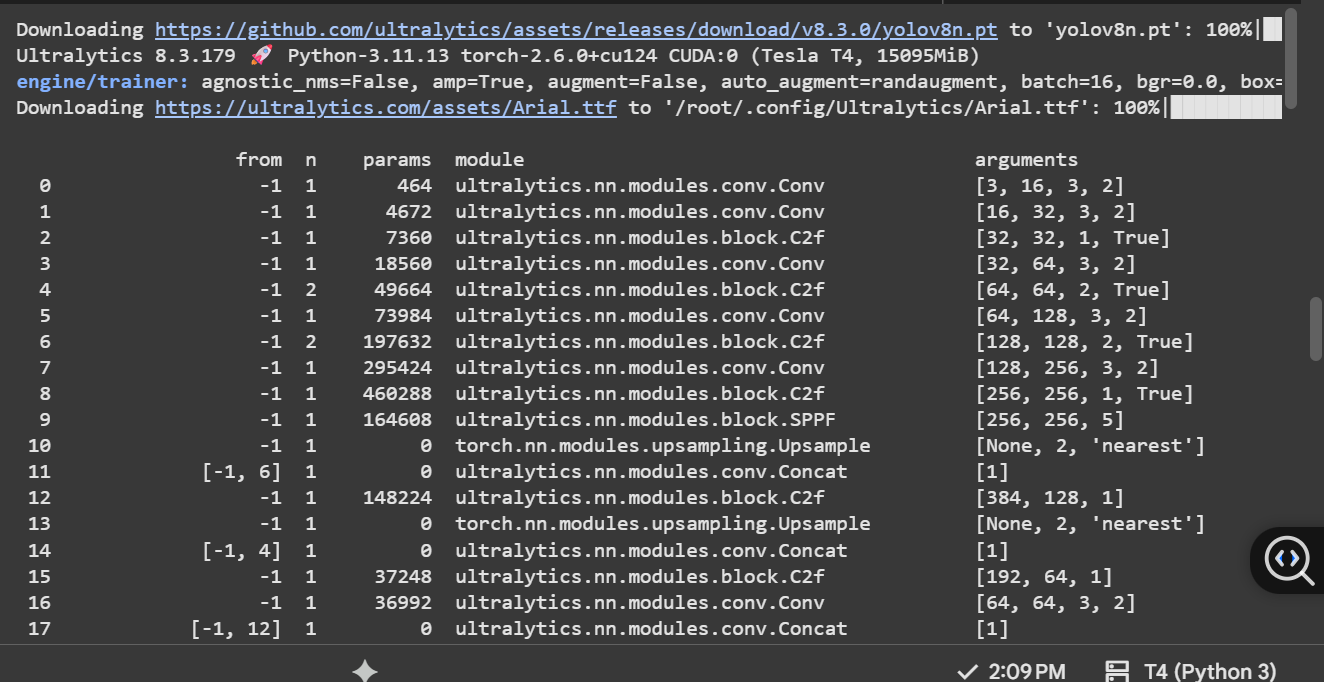
**Consistency Across Training**

* YOLO training, validation, and testing all read from the **same source (data.yaml)**, ensuring consistency.

**Robustness**

* data.yaml also stores metadata like dataset source (Roboflow, license, etc.).
* This makes your training reproducible and sharable with others.

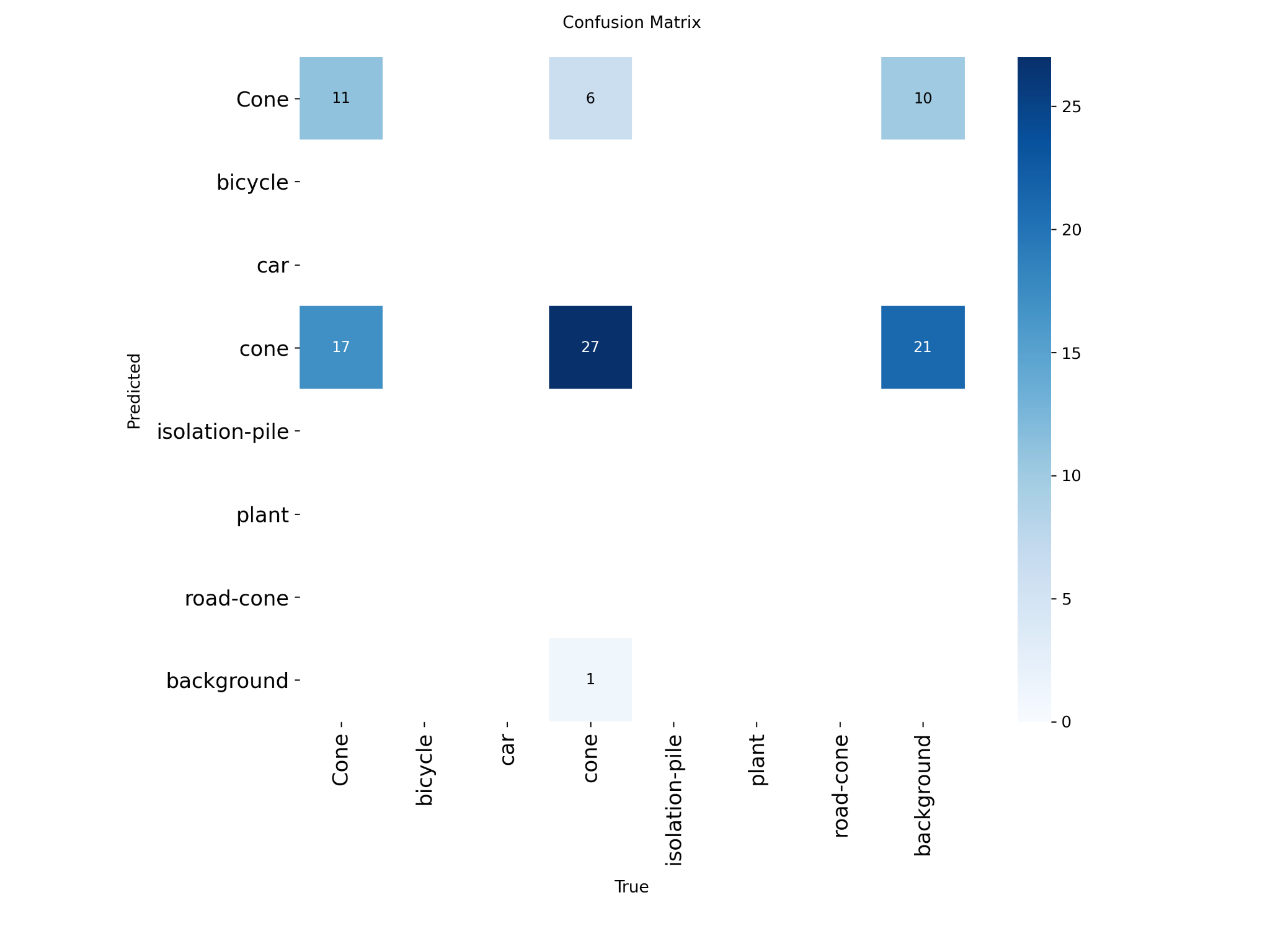
**What is the output in your terminal? Watch for the training progress, loss values, and metrics.**



During training, YOLOv8 logs each epoch with loss values (box, classification, DFL), precision, recall, and mAP scores. Initially, losses were high, but they decreased as training progressed, while precision, recall, and mAP improved.

At the end of training, the best model was saved at runs/detect/train/weights/[best.pt](http://best.pt)

**Confusion Matrix**



* Query successful

The confusion matrix you provided shows the performance of a classification model. The x-axis (horizontal) represents the **True** labels, which are the actual classes of the data points. The y-axis (vertical) represents the **Predicted** labels, which are the classes assigned by the model.

Based on the image, we can analyze the model's predictions:

* **Correct Predictions (True Positives):** The numbers on the diagonal from the top-left to the bottom-right represent the instances where the model's prediction matched the true label.
  + The model correctly classified **11** instances as 'cone'.
  + The model correctly classified **27** instances as 'car'.
  + There appear to be no correct predictions for 'bicycle', 'isolation-pile', 'plant', 'road-cone', or 'background'. This is indicated by the absence of non-zero values on the diagonal for these classes.
* **Incorrect Predictions (False Positives and False Negatives):** The off-diagonal numbers represent misclassifications.
  + **False Positives (Type I error):** These are cases where the model incorrectly predicted a class. For example, the row labeled 'cone' shows that the model predicted 'cone' for 6 instances that were actually 'bicycle' and 10 instances that were actually 'background'.
  + **False Negatives (Type II error):** These are cases where the model failed to predict the correct class. For example, the column labeled 'car' shows that the model misclassified 17 instances that were actually 'car' as 'cone' and 21 instances as 'background'. The model struggled significantly with classifying 'car' and 'background', as shown by the large off-diagonal values.

**Observation: Why is it important to use best.pt instead of the original yolov8n.pt?**

The file yolov8n.pt is the pre-trained base model provided by Ultralytics.

After training on your custom dataset, YOLO saves two weights:

* last.pt → the weights from the final epoch.
* best.pt → the weights that achieved the best validation performance during training.

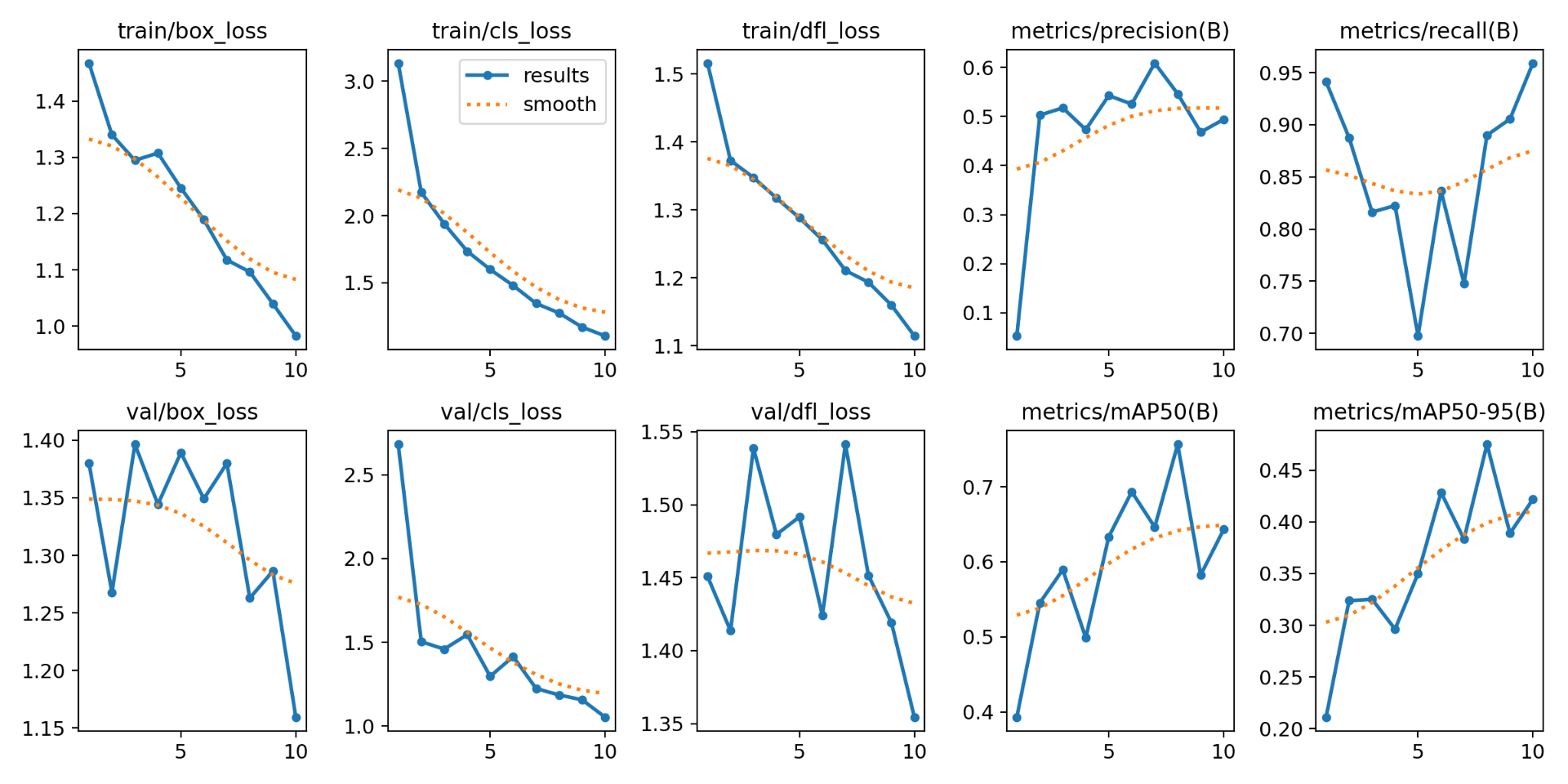
Using best.pt ensures you are running inference with the model that performed best on your validation set, not just the last saved version or the generic pretrained one.

**FinalCone Detected Image**

****

Yes, the displayed image correctly shows that the model has detected a traffic cone. The bounding box is drawn around the cone, and the label “cone” is assigned. The confidence score is 0.60 (60%), which means the model is moderately confident about its prediction. While it correctly identifies the cone, the confidence is not very high, suggesting that further training or fine-tuning may be needed to improve detection reliability.

**Observation about Results.PNG**

****

### **Top Row (Training Losses + Metrics)**

* **train/box\_loss** → measures how well bounding boxes fit the objects.  
   Decreasing → model is learning to localize objects better.
* **train/cls\_loss** → measures classification error (wrong class predictions).  
   Decreasing steadily → your model is improving in classifying cones vs other objects.
* **train/dfl\_loss** → Distribution Focal Loss (used for bounding box regression).  
   Decreasing smoothly → bounding box predictions are stabilizing.
* **metrics/precision(B)** → percentage of predicted cones that were correct.  
   Fluctuating but improving → precision around **0.5–0.6** by epoch 10.
* **metrics/recall(B)** → percentage of actual cones your model found.  
   Dropped around epoch 5 but recovered → around **0.9+ recall** at epoch 10 (good).

### **🔹 Bottom Row (Validation Losses + Metrics)**

* **val/box\_loss, val/cls\_loss, val/dfl\_loss** → how well your model generalizes to unseen validation images.  
  + They fluctuate more (normal for small datasets).
  + Still trending downward overall → improvement is happening.
* **metrics/mAP50(B)** → mean Average Precision at IoU=0.5 (how well predictions overlap with ground truth).  
   Increased from ~0.3 → ~0.7 by epoch 10 → strong progress.
* **metrics/mAP50-95(B)** → stricter metric (IoU 0.5–0.95).  
   Increased from ~0.2 → ~0.42 → improving, though harder to optimize.

### **Overall Interpretation**

* Training losses are **consistently decreasing** → model is learning.
* Validation losses fluctuate but **trend downward** → generalization is okay.
* mAP50 (~0.7) and mAP50-95 (~0.42) after just 10 epochs → promising results! With more epochs (50–100), performance will likely improve a lot.
* Precision and recall are balancing out → the model finds most cones but sometimes predicts extras.

**Conclusion**

Through this project, I successfully fine-tuned a YOLOv8 model to detect traffic cones. By carefully structuring the dataset, configuring the data.yaml file, and running the training process, I observed how losses decreased and performance metrics (precision, recall, and mAP) improved over time.

The evaluation metrics and visualizations (results plot, confusion matrix, and inference output) clearly demonstrate the power of **transfer learning**: starting from a pretrained model (yolov8n.pt) and adapting it to a specialized dataset yielded promising results in just a few epochs.

Key takeaways:

* **data.yaml** is critical as it links dataset structure with class definitions, ensuring YOLO knows how to train and validate correctly.
* Using **best.pt** guarantees inference with the most effective model checkpoint rather than just the last trained weights.
* Even with a relatively small dataset and short training time (10 epochs), the model was able to detect cones with moderate confidence (~60%).

Overall, this exercise highlights how YOLOv8 can be fine-tuned efficiently for domain-specific tasks. With more data, additional epochs, and hyperparameter tuning, the model can achieve higher confidence and stronger generalization. This demonstrates how deep learning can be quickly adapted to real-world object detection problems with limited resources.