**Object Detection Using YOLOv8**

*Experience Report by Gavinolla Chaitanya Reddy*

## How the Code Works

- **Dataset Preparation**:
  - I downloaded the Pascal VOC 2007 and 2012 datasets. These contain images and information about where objects like cars, people, and animals are in the images.
  - The object information was in XML files, so I converted them into YOLO format, which YOLO models understand better.
  - Then, I organized everything into folders for training and testing.
- **YOLOv8 Model Training**:
  - I used a pre-trained YOLOv8 model, which means it already knew how to detect many objects.
  - I trained it on the Pascal VOC data for 10 rounds (called epochs) so it could learn even better.
  - After training, it saved the best version of the model in a file called `best.pt`.
- **Model Evaluation**:
  - I checked how good the model was by testing it on new images.
  - It gave results like how accurate it was (mAP), how many correct objects it found (precision), and how many total objects it caught (recall).
- **Inference and Visualization**:
  - I picked some test images and ran the trained model on them.
  - It drew boxes around the objects it found and showed the labels, which I displayed using simple Python code.

## **Challenges I Faced**

The first challenge I faced was finding the right dataset. Initially, I tried using the COCO dataset, but it was too large and had too many classes, which made it difficult to handle. So, I decided to switch to the Pascal VOC dataset, which was smaller and more manageable.

Training the model also came with challenges. Even though Pascal VOC is smaller than COCO, it still took a long time to train. At one point, I even ran out of free GPU time on Google Colab, which slowed down the process further.

## How I Used AI Tools to Help with Coding

I used AI tools to help me convert the VOC XML files into YOLO TXT format, which was needed for training the model. AI also helped me build the dataset folders correctly by showing me how to arrange the images and labels in the right way.

AI also helped me choose the right dataset by giving me information about how many classes each dataset had and how large they were. This helped me switch from COCO to Pascal VOC, which was easier to work with.

I also used AI for debugging issues when the code didn't run properly. It helped me fix errors and understand what went wrong in my scripts.

Overall, AI tools made the coding part easier and helped me learn faster by showing me solutions and explaining things in a simple way.

## What I Learned from the Project

I learned how to use the YOLOv8 model to detect objects in images. I started by preparing the dataset, changing the labels into the format that YOLO understands, and putting the images into folders for training and testing.

Then I trained the model and saved the best version. I also tested it to see how well it could find objects in new images. I used tools like Ultralytics and Matplotlib to help with this.

AI tools helped me when I didn't know how to write some parts of the code or when I got errors. I didn't let the AI do everything—I used it to understand things better. This project taught me how to solve problems step by step and made me feel more confident with coding and machine learning.

## How I Feel About the Balance Between Writing Code Myself vs. Using AI Assistance

I think using AI to help with coding is really helpful, especially when I get stuck or don't know how to start. Sometimes I know what I want to do, but I don't know the exact code. That's when AI gives me a push in the right direction.

But I also try to understand the code and write parts by myself. If I just copy everything, I won't learn. So I use AI like a guide or a teacher that helps me when I need it. I feel it's a good balance. AI makes things easier, but I still try to learn and do things on my own too.

## Suggestions for Improving This Assignment

One suggestion for improving this assignment would be to use a larger and more diverse dataset like COCO, which has around 80 classes. This would give the model more variety to learn from and make the object detection more powerful.

Another suggestion is to increase the number of training epochs from 10 to at least 25. This would help the model train longer and improve its accuracy and performance on test images. I wasn't able to train for 25 epochs because of GPU limitations on Google Colab, but it would definitely improve the results if more resources were available.

### AI Assistance and Prompting Strategy

Convert VOC XML files to YOLOv8 txt by reading bounding boxes and normalizing with image size, then save labels, also copy images and organize train, val, test folders by processing splits for VOC 2007 and 2012.

Code generated:

```
def voc_to_yolo(xml_path, txt_path, img_w, img_h):
    root_xml = ET.parse(xml_path).getroot()
    lines = []
    for obj in root_xml.iter("object"):
        cls = obj.find("name").text
        if cls not in cls2id: continue
        bbox = obj.find("bndbox")
        xmin, ymin, xmax, ymax = [float(bbox.find(t).text) for t in
('xmin','ymin','xmax','ymax')]
        # convert to YOLO (cx,cy,w,h) normalized 0-1
        cx = (xmin + xmax) / 2 / img_w
        cy = (ymin + ymax) / 2 / img_h
        w  = (xmax - xmin) / img_w
        h  = (ymax - ymin) / img_h
        lines.append(f"{cls2id[cls]} {cx:.6f} {cy:.6f} {w:.6f} {h:.6f}")
    txt_path.write_text("\n".join(lines))
```

```python
def process_split(split, year):
    img_dir  = voc_root/f"VOCdevkit/VOC{year}/JPEGImages"
    ann_dir  = voc_root/f"VOCdevkit/VOC{year}/Annotations"
    list_txt = voc_root/f"VOCdevkit/VOC{year}/ImageSets/Main/{split}.txt"

    dst_img = root/f"images/{split}"
    dst_lbl = root/f"labels/{split}"
    dst_img.mkdir(parents=True, exist_ok=True)
    dst_lbl.mkdir(parents=True, exist_ok=True)

    ids = list_txt.read_text().strip().split()
    for img_id in tqdm(ids, desc=f"VOC{year}-{split}"):
        src_img = img_dir/f"{img_id}.jpg"
        src_xml = ann_dir/f"{img_id}.xml"
        dst_img_file = dst_img/f"{img_id}.jpg"
        dst_lbl_file = dst_lbl/f"{img_id}.txt"

        # copy image
        if not dst_img_file.exists():
            shutil.copy2(src_img, dst_img_file)

        # parse size (needed for normalization)
        tree = ET.parse(src_xml).getroot()
        w = int(tree.find("size/width").text)
        h = int(tree.find("size/height").text)
        voc_to_yolo(src_xml, dst_lbl_file, w, h)

# build dataset (train = VOC07+12 trainval, val = VOC07 test)
process_split("trainval", "2007")
process_split("trainval", "2012")
process_split("test",     "2007")
```

Make yaml file for YOLO dataset with train and val folder paths, number of classes, and class names, then save it to voc20.yaml using yaml.dump, so model can read the dataset info easily , and train properly.

Code generated:

```python
# -------------------------------------------------------------------
# 3. create dataset YAML --------------------------------------------
```

```python
yaml_path = root/"voc20.yaml"
yaml_dict = {
    'path': str(root),
    'train': 'images/trainval',
    'val'  : 'images/test',
    'nc'   : len(classes),
    'names': classes,
}
yaml_path.write_text(yaml.dump(yaml_dict))

print("\n✅ VOC converted → YOLO format")
print("Dataset root :", root)
print("YAML file    :", yaml_path)
```