STEP 1 - Set Up Environment in Colab

```
# Install necessary packages
!pip install -q torch torchvision
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 95.2 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 30.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 53.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.4 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 13.1 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.6 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 5.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 98.9 MB/s eta 0:00:00
```

STEP 2 - Download and Extract Pascal VOC Dataset

```
# Download VOC 2007 train/val
!wget -q http://host.robots.ox.ac.uk/pascal/VOC/voc2007/VOCtrainval_06-Nov-2007.tar
!tar -xf VOCtrainval_06-Nov-2007.tar
```

STEP 3 - Load and Transform the Dataset

```
import torchvision
from torchvision.datasets import VOCDetection
import torchvision.transforms as T

def get_transform():
    return T.Compose([
        T.ToTensor()
    ])
```

STEP 4 - Prepare the Dataset Class (Custom Wrapper for Faster R-CNN)

```
import os
import numpy as np
import torch
from PIL import Image
import xml.etree.ElementTree as ET

class VOCDataset(torch.utils.data.Dataset):
    def __init__(self, root, image_set='train', transforms=None):
        self.root = os.path.join(root, 'VOC2007')
        self.image_dir = os.path.join(self.root, 'JPEGImages')
        self.annotation_dir = os.path.join(self.root, 'Annotations')
        self.transforms = transforms
        with open(os.path.join(self.root, 'ImageSets/Main', image_set + '.txt')) as f:
            self.ids = [line.strip() for line in f.readlines()]

        self.classes = ['__background__'] + [
            'aeroplane', 'bicycle', 'bird', 'boat', 'bottle',
            'bus', 'car', 'cat', 'chair', 'cow', 'diningtable',
            'dog', 'horse', 'motorbike', 'person', 'pottedplant',
            'sheep', 'sofa', 'train', 'tvmonitor'
        ]

    def __getitem__(self, index):
        image_id = self.ids[index]
        img_path = os.path.join(self.image_dir, image_id + ".jpg")
        ann_path = os.path.join(self.annotation_dir, image_id + ".xml")

        img = Image.open(img_path).convert("RGB")
        boxes = []
        labels = []

        tree = ET.parse(ann_path)
        for obj in tree.findall("object"):
            name = obj.find("name").text
            bbox = obj.find("bndbox")
            box = [
                float(bbox.find("xmin").text),
                float(bbox.find("ymin").text),
                float(bbox.find("xmax").text),
                float(bbox.find("ymax").text)
            ]
            boxes.append(box)
            labels.append(self.classes.index(name))

        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.as_tensor(labels, dtype=torch.int64)
        target = {
            'boxes': boxes,
            'labels': labels,
            'image_id': torch.tensor([index])
        }

        if self.transforms:
            img = self.transforms(img)

        return img, target
```

```
    def __len__(self):
        return len(self.ids)
```

STEP 5 - Load Pretrained Backbone + Faster R-CNN

```
import torchvision
from torchvision.models.detection import FasterRCNN
from torchvision.models.detection.rpn import AnchorGenerator

# Load ResNet-50 backbone (excluding classifier head)
backbone = torchvision.models.resnet50(pretrained=True)
modules = list(backbone.children())[:-2]  # remove avgpool and fc
backbone = torch.nn.Sequential(*modules)
backbone.out_channels = 2048

# Define AnchorGenerator
anchor_generator = AnchorGenerator(
    sizes=((32, 64, 128, 256, 512),),  # size per feature map
    aspect_ratios=((0.5, 1.0, 2.0),)   # aspect ratios per feature map
)

# Create Faster R-CNN model with anchor generator
model = FasterRCNN(
    backbone=backbone,
    num_classes=21,  # 20 VOC classes + background
    rpn_anchor_generator=anchor_generator
)
```

```
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/torch/hub/checkpoints/resnet50-0676ba61.pth
100%|██████████| 97.8M/97.8M [00:00<00:00, 161MB/s]
```

STEP 6 - Train the Model

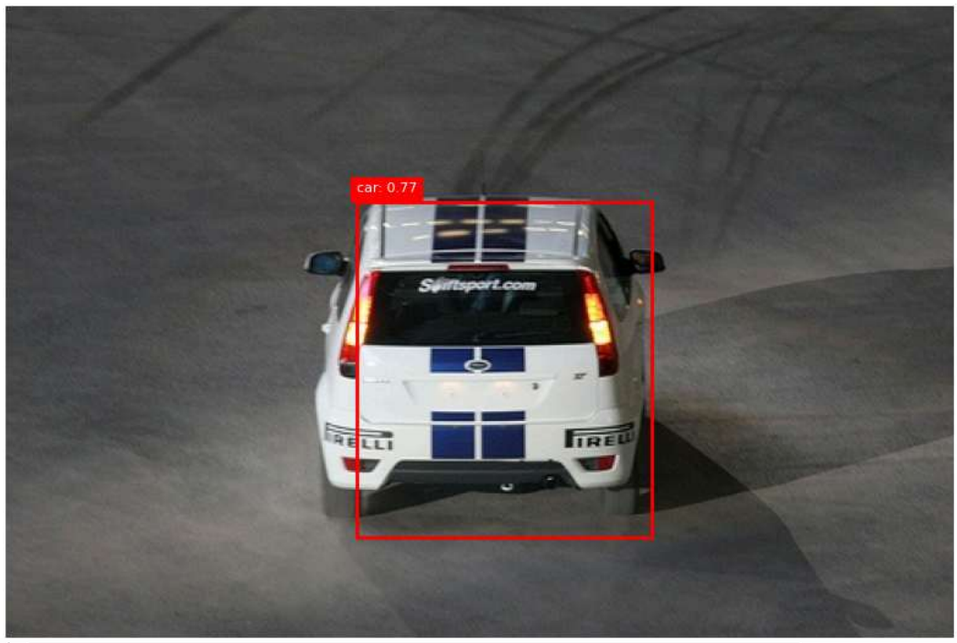
```
from torch.utils.data import DataLoader
import torchvision
import torch

def collate_fn(batch):
    return tuple(zip(*batch))

dataset = VOCDataset(root="VOCdevkit", image_set="train", transforms=get_transform())
dataloader = DataLoader(dataset, batch_size=2, shuffle=True, collate_fn=collate_fn)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Optimizer
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)

# Training loop
num_epochs = 5
model.train()
for epoch in range(num_epochs):
    for images, targets in dataloader:
        images = list(img.to(device) for img in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())

        optimizer.zero_grad()
        losses.backward()
        optimizer.step()

    print(f"Epoch {epoch+1}, Loss: {losses.item():.4f}")
```

```
Epoch 1, Loss: 1.0691
Epoch 2, Loss: 0.3049
Epoch 3, Loss: 1.5103
Epoch 4, Loss: 0.1116
Epoch 5, Loss: 0.1293
```

STEP 7 - Evaluate and Visualize

```
import torchvision.transforms.functional as F
from torchvision.ops import nms
import matplotlib.pyplot as plt
import matplotlib.patches as patches

# Optional: replace with your custom class labels if needed
VOC_CLASSES = [
    "__background__", "aeroplane", "bicycle", "bird", "boat", "bottle",
    "bus", "car", "cat", "chair", "cow", "diningtable", "dog", "horse",
    "motorbike", "person", "pottedplant", "sheep", "sofa", "train", "tvmonitor"
]
```

```
model.eval()
with torch.no_grad():
    image, _ = dataset[0]  # Test on the first image
    image = image.to(device)
    prediction = model([image])[0]

# Apply score threshold and NMS
boxes = prediction['boxes']
scores = prediction['scores']
labels = prediction['labels']

score_thresh = 0.5
nms_thresh = 0.4

keep = scores > score_thresh
boxes = boxes[keep]
scores = scores[keep]
labels = labels[keep]

keep_nms = nms(boxes, scores, nms_thresh)
boxes = boxes[keep_nms]
scores = scores[keep_nms]
labels = labels[keep_nms]

# Plot
img = F.to_pil_image(image.cpu())
fig, ax = plt.subplots(1, figsize=(10, 7))
ax.imshow(img)

for box, label, score in zip(boxes.cpu(), labels.cpu(), scores.cpu()):
    xmin, ymin, xmax, ymax = box
    rect = patches.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin, linewidth=2, edgecolor='red', facecolor='none')
    ax.add_patch(rect)
    ax.text(xmin, ymin - 5, f"{VOC_CLASSES[label]}: {score:.2f}", color='white', backgroundcolor='red', fontsize=8)

ax.axis('off')
plt.show()
```



STEP 8 - Download TorchMetrics

```
pip install torchmetrics
```

```
Collecting torchmetrics
  Downloading torchmetrics-1.7.1-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.11/dist-packages (from torchmetrics) (2.0.2)
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.11/dist-packages (from torchmetrics) (24.2)
Requirement already satisfied: torch>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from torchmetrics) (2.6.0+cu124)
Collecting lightning-utilities>=0.8.0 (from torchmetrics)
  Downloading lightning_utilities-0.14.3-py3-none-any.whl.metadata (5.6 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-packages (from lightning-utilities>=0.8.0->torchmetrics) (75
Requirement already satisfied: typing_extensions in /usr/local/lib/python3.11/dist-packages (from lightning-utilities>=0.8.0->torchmetri
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics) (3.18.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetr
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchme
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetr
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetric
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetric
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetr
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics) (2.
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics) (
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetri
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics) (3.2.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch>=2.0.0->torchmetrics) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch>=2.0.0->torchmet
```

```
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch>=2.0.0->torchmetrics) (3.0
Downloading torchmetrics-1.7.1-py3-none-any.whl (961 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 961.5/961.5 kB 13.6 MB/s eta 0:00:00
Downloading lightning_utilities-0.14.3-py3-none-any.whl (28 kB)
Installing collected packages: lightning-utilities, torchmetrics
Successfully installed lightning-utilities-0.14.3 torchmetrics-1.7.1
```

STEP 9 - Evaluation metrics (mAP, precision, recall)

```
from torchmetrics.detection.mean_ap import MeanAveragePrecision

metric = MeanAveragePrecision()
model.eval()
with torch.no_grad():
    for images, targets in dataloader:
        images = [img.to(device) for img in images]
        outputs = model(images)

        # Move to CPU and format for metric
        outputs = [{k: v.cpu() for k, v in out.items()} for out in outputs]
        targets = [{k: v.cpu() for k, v in t.items()} for t in targets]

        metric.update(outputs, targets)

final_result = metric.compute()
print(final_result)  # will show mAP, precision, recall, etc.
```

```
{'map': tensor(0.2764), 'map_50': tensor(0.6083), 'map_75': tensor(0.1961), 'map_small': tensor(0.0517), 'map_medium': tensor(0.2517), '
    19, 20], dtype=torch.int32)}
```

STEP 10 - Save the Model

```
torch.save(model.state_dict(), "fasterrcnn_voc.pth")

from google.colab import drive
drive.mount('/content/drive')
```

STEP 11- Load the Model

```
model.load_state_dict(torch.load("fasterrcnn_voc.pth"))
model.eval()
```

```
FasterRCNN(
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      Resize(min_size=(800,), max_size=1333, mode='bilinear')
  )
  (backbone): Sequential(
    (0): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
    (4): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        )
      )
      (1): Bottleneck(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
      (2): Bottleneck(
        (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
      )
    )
    (5): Sequential(
      (0): Bottleneck(
        (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (bn3): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
        (relu): ReLU(inplace=True)
        (downsample): Sequential(
          (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
          (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

```
      )
    )
    (1): Bottleneck(
```

STEP 12 - SAVE Demo: Detection on Test Images

```
plt.savefig("output_prediction.png")
```

```
<Figure size 640x480 with 0 Axes>
```

Experience Report – Object Detection Assignment

1. Challenges Faced

- Understanding the VOC dataset format and how to use the custom `VOCDataset` class.
- Tuning confidence threshold and NMS to reduce redundant boxes.
- Evaluating mAP and understanding how to use `torchmetrics` for metrics.

2. How I Used AI Tools

- Used ChatGPT to understand how to visualize object detection predictions with bounding boxes and labels.
- Asked for help writing a clean inference and NMS function.
- Used Copilot for autocompleting PyTorch loops and syntax.

3. What I Learned

- Structure of object detection pipelines (backbone + detection head).
- Role of Region Proposal Networks in Faster R-CNN.
- How to prepare datasets and evaluate with mAP.

4. What Surprised Me

- How many redundant boxes a raw model outputs without NMS.
- How easy it is to overfit small datasets without careful validation.
- How much AI tools can speed up development if guided correctly.

5. My Thoughts on AI-Assisted Coding

- I enjoyed using AI to speed up routine parts of the code, like boilerplate or fixing bugs.
- It's important to read and understand what AI outputs — blindly copying can lead to subtle bugs.
- Overall, it felt like pair programming with a super-fast assistant.

6. Suggestions for Improving the Assignment

- Provide a small working example with baseline mAP evaluation.
- Recommend tools like FiftyOne or torchmetrics early for evaluation.
- Include optional bonus challenges (e.g., try YOLOv5 or fine-tune on custom data).