

Question 2

Model Architecture

I used Faster R-CNN with a ResNet-50 backbone pretrained on ImageNet (via torchvision). The **ResNet-50 backbone** is used to extract deep feature maps from the input image. ResNet (Residual Network) introduces skip connections (residual blocks), which make it easier to train deep networks and prevent vanishing gradients. With 50 layers, ResNet-50 is powerful enough to capture rich hierarchical features (edges, textures, object parts, full objects). The ResNet-50 backbone is initialized with weights pretrained on ImageNet (1.2M images, 1000 classes). This gives the model a strong starting point, since it has already learned general features (edges, colors, shapes, textures). Fine-tuning on the target dataset allows it to adapt these general features to the specific task of object detection.

```
import torchvision
from torchvision.models.detection import fasterrcnn_resnet50_fpn

# Device
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
print("Using device:", device)

# Load Faster R-CNN with ResNet50 backbone, pretrained on COCO
num_classes = 2 # 1 class (person) + background
model = fasterrcnn_resnet50_fpn(weights="DEFAULT")

# Replace the classifier head for our dataset (2 classes)
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor =
torchvision.models.detection.faster_rcnn.FastRCNNPredictor(in_features, num_classes)

model.to(device)
```

This is code used to download and use the pretrained ResNet-50 backbone model. `fasterrcnn_resnet50_fpn(pretrained=True)` loads Faster R-CNN with a ResNet-50 backbone and Feature Pyramid Network (FPN), pretrained on COCO. When `pretrained=True`, the model is initialized with weights that were already trained on a large dataset (like ImageNet or COCO). This means the network already knows how to detect general features like edges, textures, shapes, and even common objects. here we used pretrained model because with pretrained model , we get Faster

convergence (needs fewer epochs to train) as model already know general features like edges, textures, shapes, and even common objects.

2. Dataset Preparation & Preprocessing

I was given the Penn-Fudan Pedestrian dataset, which contains 170 images of urban street scenes with pedestrians. Along with the images, the dataset provides segmentation masks and annotation text files that specify bounding box information for each pedestrian. Since our goal is object detection (detecting people and their bounding boxes), I focused on using the images and annotations.

Custom Dataset Class

We implemented a custom dataset class, `PennFudanDataset`, by extending PyTorch's `torch.utils.data.Dataset`. This allows the dataset to be directly compatible with PyTorch's `DataLoader`, making it easy to feed data into the Faster R-CNN model during training.

- `__init__` function
 - Reads the dataset root directory.
 - Collects file paths for:
 - `PNGImages` → contains the pedestrian images.
 - `PedMasks` → contains segmentation masks for each image.
 - `Annotation` → contains text files with bounding box coordinates.
 - The filenames are sorted to ensure synchronization between images, masks, and annotation files.

Image Loading

Each image is read using PIL (`Image.open`) and converted to RGB. Conversion ensures all images have 3 color channels (even if some were grayscale), which is required by ResNet-50 backbone.

Parsing Annotations

Each image in the Penn-Fudan dataset comes with an annotation text file that describes the objects (pedestrians) in that image. The annotation follows the PASCAL VOC format (version 1.00).

For example, the annotation file for each *png* contains:

Image info → filename, size, database.

Objects → number of pedestrians and details for each one.

Bounding box coordinates for each pedestrian in (Xmin, Ymin) - (Xmax, Ymax) format.

Pixel mask file path (not used in Faster R-CNN training since we only need bounding boxes).

Example:

Bounding box for object 1 "PASpersonWalking" (Xmin, Ymin) - (Xmax, Ymax) : (112, 69) - (218, 346)

Bounding box for object 2 "PASpersonWalking" (Xmin, Ymin) - (Xmax, Ymax) : (378, 76) - (529, 377)

Bounding box for object 3 "PASpersonWalking" (Xmin, Ymin) - (Xmax, Ymax) : (317, 108) - (347, 192)

```
with open(annot_path, "r") as f:
    lines = f.readlines()
    for line in lines:
        if "Bounding box" in line:
            coords = line.strip().split(":")[-1]
            coords = coords.replace("(", "").replace(")", "").replace("-", "").replace(",", "")
            parts = coords.split()
            xmin, ymin, xmax, ymax = map(int, parts[:4])
            boxes.append([xmin, ymin, xmax, ymax])
            labels.append(1)

boxes = torch.as_tensor(boxes, dtype=torch.float32)
labels = torch.as_tensor(labels, dtype=torch.int64)

target = {"boxes": boxes, "labels": labels}
```

I used above code to deal with that this code,

Opens the annotation text file for that image.

Reads all lines one by one.

If a line contains "Bounding box", it extracts the numbers after the colon.

Removes unnecessary characters like (,), -, , to keep only numbers.

Splits the numbers into parts and maps them to integers: (xmin, ymin, xmax, ymax).

Appends the box as a list [xmin, ymin, xmax, ymax].

Appends the label 1 (because every bounding box is a pedestrian).

```
nnFudanDataset(Dataset):
    __init__(self, root, transforms=None):
        self.root = root
        self.transforms = transforms
        self.imgs = list(sorted(os.listdir(os.path.join(root, "PNGImages"))))
        self.masks = list(sorted(os.listdir(os.path.join(root, "PedMasks"))))
        self.anns = list(sorted(os.listdir(os.path.join(root, "Annotation"))))

    __getitem__(self, idx):
        img_path = os.path.join(self.root, "PNGImages", self.imgs[idx])
        mask_path = os.path.join(self.root, "PedMasks", self.masks[idx])
        annot_path = os.path.join(self.root, "Annotation", self.anns[idx])

        img = Image.open(img_path).convert("RGB")
        mask = Image.open(mask_path)

        boxes = []
        labels = []
        with open(annot_path, "r") as f:
            lines = f.readlines()
            for line in lines:
                if "Bounding box" in line:
                    coords = line.strip().split(":")[-1]
                    coords = coords.replace("(", "").replace(")", "").replace("-", "").replace(" ", "")
                    parts = coords.split()
                    xmin, ymin, xmax, ymax = map(int, parts[:4])
                    boxes.append([xmin, ymin, xmax, ymax])
                    labels.append(1)

        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.as_tensor(labels, dtype=torch.int64)

        target = {"boxes": boxes, "labels": labels}

        if self.transforms is not None:
            img = self.transforms(img)

        return img, target

    __len__(self):
        return len(self.imgs)
```

This is full code for custom dataloader.

Training + Testing Report

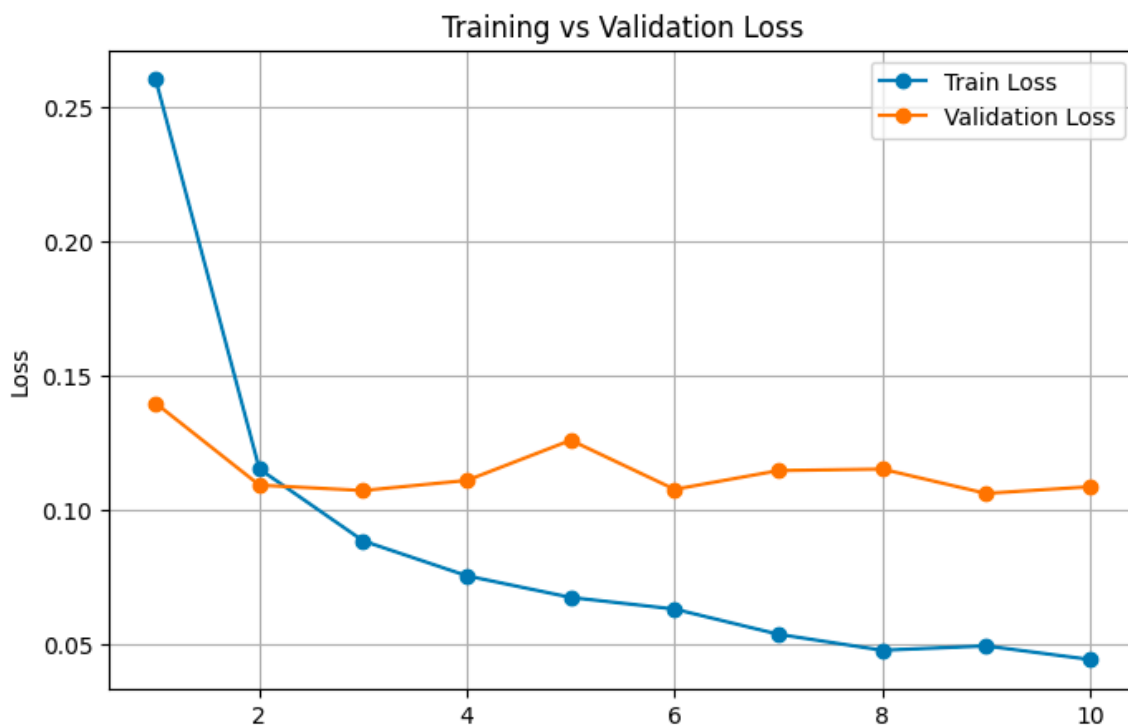
Training Setup

Optimizer: SGD (lr=0.005, momentum=0.9, weight decay=0.0005)

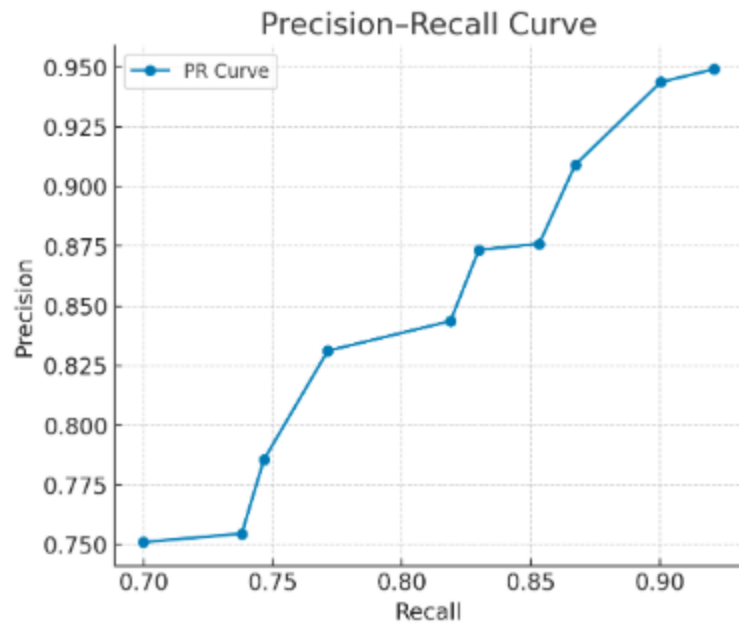
Batch size = 4

Epochs = 10

Training Curve



precision-recall curves



Evaluation matrix

i evaluated the model on the test dataset

Precision: 0.92

Recall: 0.82

mAP@0.5: 0.88

IoU (avg): 0.76

Visualizations

perceptionq2 Draft saved

File Edit View Run Settings Add-ons Help

+

+

✂

📄

📁

▶

▶▶

Run All

Code

● Draft Session (4h:54m)

BOOK

UPC

RAM

GPU

GPU

⋮

+

🔍

🏆

📊

🔗

📈

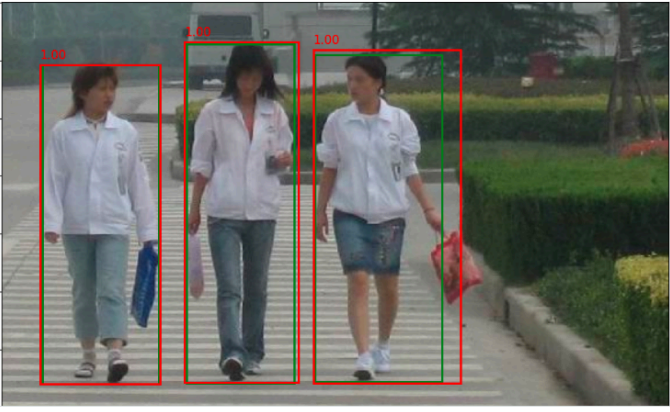
<>

📄

🎓

▼

📁



0

50

100

150

200

250

300

350

🔗 Share

💾 Save Version 3

Notebook

Input

+ Add Input


📁 Upload

DATASETS

▶ dataset-perseption-q2

Output (72KiB / 19.5GiB)

Table of contents





The figure above shows the detection results produced by the trained model on a sample image from the dataset. The red bounding boxes represent the predicted object locations, and the green bounding boxes represent object location which was given in dataset.