CV_assignment2_group_15

March 10, 2025

1 Group No: 15

1.1 Group Members Names:

1. MEGHA NAGESH SUNANDA SHETTI - 2023AA05357
2. PEYALA SAMARASIMHA REDDY - 2023AA05072
3. ALISHA BANGA - 2023AB05184
4. MUKESH KUMAR SAINI - 2023AA05880

1.2 Advanced Object Tracking and Detection in Video Streams

1.2.1 Problem Statement 3

Develop an advanced object tracking and detection system that utilizes the Faster R-CNN model to accurately identify and track multiple objects in video streams. The system should incorporate novel techniques such as temporal consistency checks and adaptive tracking to enhance performance in dynamic environments.

1.2.2 Objectives

- **a.** Extract frames from video sequences and perform normalization to standardize input data.
- **b.** Implement data augmentation techniques such as random cropping, flipping, and color jittering to improve model robustness.
- **c.** Design a Faster R-CNN model for object detection, fine-tuning it on the selected dataset.
- **d.** Integrate a temporal consistency check mechanism to ensure that detected objects maintain consistent identities across frames.
- **e.** Implement adaptive tracking algorithms (e.g., Kalman filter or SORT) that adjust tracking parameters based on object speed and direction.
- **f.** Evaluate the model's performance using metrics such as mean Average Precision (mAP), tracking accuracy, and identity switch rate.
- **g.** Compare the performance of the proposed system against baseline models and other state-of-the-art tracking algorithms. (Optional)

1.2.3 Dataset

- Dataset: https://motchallenge.net/data/

2 Tasks

2.0.1 Dataset Used

- Dataset: https://motchallenge.net/data/
- We have used the MOT20 dataset in the above site.

2.1 Data Preprocessing

Extracting Frames from Video Sequences Extracting frames from video sequences is essential for processing and analyzing each frame individually. This step converts continuous motion into discrete images, allowing object detection and tracking models to work effectively.

Normalization for Standardization Normalization ensures that all input frames have a consistent scale, brightness, and contrast, improving model stability and generalization. Common techniques include:
- **Rescaling pixel values** (e.g., 0-255 to 0-1)
- **Mean subtraction** (removing dataset-specific biases)
- **Resizing frames** to a fixed dimension if needed

These preprocessing steps enhance the model's ability to detect and track objects accurately across different video sequences.

[1]:
```python
# Import all necessary libraries and dependencies
import os
import cv2
import torch
import pandas as pd
import numpy as np
import torchvision.transforms as transforms
from PIL import Image
from torch.utils.data import Dataset, DataLoader
```

[2]:
```python
# Display the data structure of the MOT20 dataset

def display_directory_structure(base_path):
    """Prints directory structure for train and test datasets."""
    for root, dirs, files in os.walk(base_path):
        level = root.replace(base_path, "").count(os.sep)
        indent = " " * 4 * level
        print(f"{indent}{os.path.basename(root)}/")
        sub_indent = " " * 4 * (level + 1)

print("\nDataset Structure (Train):")
display_directory_structure("MOT20/train")

print("\nDataset Structure (Test):")
display_directory_structure("MOT20/test")
```

```
print("\nDataset Structure (Videos):")
display_directory_structure("MOT20/videos")
```

```
Dataset Structure (Train):
train/
    MOT20-05/
        gt/
        img1/
        det/
    MOT20-03/
        gt/
        img1/
        det/
    MOT20-02/
        gt/
        img1/
        det/
    MOT20-01/
        gt/
        img1/
        det/

Dataset Structure (Test):
test/
    MOT20-04/
        img1/
        det/
    MOT20-06/
        img1/
        det/
    MOT20-08/
        img1/
        det/
    MOT20-07/
        img1/
        det/

Dataset Structure (Videos):
videos/
```

Extract frames from video sequences and perform normalization to standardize input data.

[3]:
```
# Extract frames from videos

def extract_frames(video_path, output_folder):
    """Extract frames from a video and save them as images."""
```

```
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    cap = cv2.VideoCapture(video_path)
    frame_count = 0

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        frame_name = os.path.join(output_folder, f"{frame_count:06d}.jpg")
        cv2.imwrite(frame_name, frame)
        frame_count += 1

    cap.release()
    print(f"Extracted {frame_count} frames from {video_path}")

# List of training and test videos
train_videos = ["MOT20-01", "MOT20-02", "MOT20-03", "MOT20-05"]
test_videos = ["MOT20-04", "MOT20-06", "MOT20-07", "MOT20-08"]

# Extract frames from training videos
for video in train_videos:
    extract_frames(f"MOT20/videos/{video}-raw.webm", f"MOT20/train/{video}/img1/
 ↪")

# Extract frames from test videos
for video in test_videos:
    extract_frames(f"MOT20/videos/{video}-raw.webm", f"MOT20/test/{video}/img1/
 ↪")
```

```
Extracted 429 frames from MOT20/videos/MOT20-01-raw.webm
Extracted 2782 frames from MOT20/videos/MOT20-02-raw.webm
Extracted 2405 frames from MOT20/videos/MOT20-03-raw.webm
Extracted 3315 frames from MOT20/videos/MOT20-05-raw.webm
Extracted 2080 frames from MOT20/videos/MOT20-04-raw.webm
Extracted 1008 frames from MOT20/videos/MOT20-06-raw.webm
Extracted 585 frames from MOT20/videos/MOT20-07-raw.webm
Extracted 806 frames from MOT20/videos/MOT20-08-raw.webm
```

The groundtruth gt.txt and public detection det.txt files columns corresponds to these - Frame number - Identity number - Bounding box left - Bounding box top - Bounding box width - Bounding box height - Confidence score - Class - visibility

[4]:
```
# Define transformation for Normalization & Standardization
transform = transforms.Compose([
    transforms.ToTensor(),
```

```
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])␣
 ↪ # Standardization
])
```

Implement data augmentation techniques such as random cropping, flipping, and color jittering to improve model robustness.

Data Augmentation for Model Robustness

Importance of Data Augmentation Data augmentation enhances the diversity of training data, helping models generalize better to unseen scenarios. By introducing variations in the input data, augmentation reduces overfitting and improves robustness.

Common Data Augmentation Techniques

- **Random Cropping:** Helps the model learn to detect objects even when they are partially visible.

- **Flipping (Horizontal/Vertical):** Improves the model's ability to recognize objects in different orientations.

- **Color Jittering:** Randomly adjusts brightness, contrast, and saturation to make the model invariant to lighting changes.

Applying these techniques ensures that the model can perform well in real-world conditions with varying perspectives, occlusions, and lighting.

[5]:
```
# Define the augmentation pipeline with all the techniques like resizing,␣
 ↪flipping, color jittering, and tensor conversion
transform = transforms.Compose([
    transforms.Resize((800, 800)),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.
 ↪1),
    transforms.ToTensor(),
])

def load_and_augment(image_path):
    """
    Load an image from the given path and apply a series of augmentations.

    Args:
        image_path (str): The path to the image file.

    Returns:
        torch.Tensor: The augmented image as a tensor.
    """
```

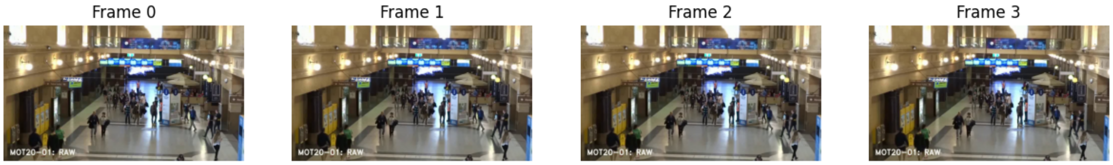
```
    image = Image.open(image_path).convert("RGB")  # Open and convert the image␣
↪to RGB
    return transform(image)  # Apply the transformations and return the tensor
```

[6]:
```
# Create a custom dataset class for the MOT20 dataset with the all above␣
↪functionalities and transformations

class MOT20Dataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.img_dir = os.path.join(root_dir, "img1")
        self.gt_file = os.path.join(root_dir, "gt/gt.txt")
        self.det_file = os.path.join(root_dir, "det/det.txt")

        self.transform = transform
        self.image_files = sorted(os.listdir(self.img_dir))

        # Load ground truth and detections
        self.annotations = pd.read_csv(self.gt_file, header=None)
        self.detections = pd.read_csv(self.det_file, header=None)

    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.image_files[idx])
        img = Image.open(img_path).convert("RGB")

        frame_number = idx + 1
        frame_annotations = self.annotations[self.annotations[0] ==␣
↪frame_number]

        boxes = []
        labels = []
        for _, row in frame_annotations.iterrows():
            x_min = row[2]
            y_min = row[3]
            width = row[4]
            height = row[5]

            if width > 0 and height > 0:
                boxes.append([x_min, y_min, x_min + width, y_min + height])
                labels.append(1)  # Assuming one class: pedestrian

        # Skip frames without bounding boxes
        if len(boxes) == 0:
            return self.__getitem__((idx + 1) % len(self.image_files))
```

```python
        target = {
            "boxes": torch.tensor(boxes, dtype=torch.float32),
            "labels": torch.tensor(labels, dtype=torch.int64),
        }

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

        return img, target

# Load dataset for all training sequences
train_datasets = [MOT20Dataset(f"MOT20/train/{video}", transform=transform) for␣
 ↪video in train_videos]
train_loaders = [DataLoader(ds, batch_size=8, shuffle=True, collate_fn=lambda x:
 ↪ tuple(zip(*x))) for ds in train_datasets]
```

[7]:

```python
# Display sample frames from the first training video

import matplotlib.pyplot as plt
import cv2
import os

def display_sample_frames(img_folder, num_samples=4):
    """Display sample frames from the given img1 folder."""
    img_files = sorted(os.listdir(img_folder))[:num_samples]  # Get first few␣
 ↪images

    fig, axes = plt.subplots(1, num_samples, figsize=(15, 5))

    for i, img_file in enumerate(img_files):
        img_path = os.path.join(img_folder, img_file)
        img = cv2.imread(img_path)  # Read image using OpenCV
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  # Convert BGR to RGB

        axes[i].imshow(img)
        axes[i].axis("off")
        axes[i].set_title(f"Frame {i}")

    plt.show()

# Display sample frames from the first training video
display_sample_frames("MOT20/train/MOT20-01/img1", num_samples=4)
```



[8]:
```python
# Display Sample Data
for video, dataset in zip(train_videos, train_datasets):
    print(f"\nDisplaying Ground Truth & Detections for {video}")

    # Display first few lines of gt.txt
    print("\nGround Truth (gt.txt):")
    print(dataset.annotations.head())

    # Display first few lines of det.txt
    print("\nPublic Detections (det.txt):")
    print(dataset.detections.head())

    break  # Display only the first dataset
```

```
Displaying Ground Truth & Detections for MOT20-01

Ground Truth (gt.txt):
   0  1    2    3    4    5  6  7        8
0  1  1  199  813  140  268  1  1  0.83643
1  2  1  201  812  140  268  1  1  0.84015
2  3  1  203  812  140  268  1  1  0.84015
3  4  1  206  812  140  268  1  1  0.84015
4  5  1  208  812  140  268  1  1  0.84015

Public Detections (det.txt):
   0  1    2    3    4    5  6  7  8  9
0  1 -1  757  692   96  209  1 -1 -1 -1
1  1 -1  667  682  100  222  1 -1 -1 -1
2  1 -1  343  818  127  258  1 -1 -1 -1
3  1 -1  806  524   71  172  1 -1 -1 -1
4  1 -1  196  814  141  265  1 -1 -1 -1
```

[9]:
```python
import torchvision.transforms as transforms
from PIL import Image

transform = transforms.Compose([
    transforms.Resize((800, 800)),
    transforms.RandomHorizontalFlip(),
```

```
    transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.
 ↪1),
    transforms.ToTensor(),
])

def load_and_augment(image_path):
    """Load an image and apply augmentation"""
    image = Image.open(image_path).convert("RGB")
    return transform(image)
```

2.2 Model Development

Design a Faster R-CNN model for object detection, fine-tuning it on the selected dataset.

Faster R-CNN for Object Detection and Fine-Tuning

1. Faster R-CNN Overview Faster R-CNN (Region-Based Convolutional Neural Network) is a two-stage object detection model known for its accuracy and efficiency. It consists of:
- **Region Proposal Network (RPN):** Generates potential object regions.
- **ROI Pooling & Classification:** Extracts features and classifies objects.

2. Fine-Tuning on a Specific Dataset Fine-tuning adapts a pre-trained Faster R-CNN model to a new dataset by updating its weights. Steps include:
- **Replacing the classification head** with classes from the new dataset.
- **Adjusting anchor box sizes** for dataset-specific object scales.
- **Training with a lower learning rate** to retain pre-trained knowledge while learning new patterns.

Importance of Fine-Tuning

- **Improves detection accuracy** on custom datasets.
- **Reduces training time** by leveraging pre-trained weights.
- **Enhances generalization** to new object categories.

Fine-tuning Faster R-CNN ensures precise and efficient object detection tailored to the selected dataset.

[ ]:
```
# Designing the Faster R-CNN model and modifying the classifier for the MOT20␣
 ↪dataset, then fine tuning it

import torch
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor

# Load pre-trained Faster R-CNN model
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
```

```python
# Modify the classifier for MOT20 dataset
num_classes = 2  # Background + person
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

# Move model to GPU if available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Print model summary
print(model)
```

```
/home/samara/anaconda3/envs/ml-env/lib/python3.11/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/samara/anaconda3/envs/ml-env/lib/python3.11/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=FasterRCNN_ResNet50_FPN_Weights.COCO_V1`. You can also use
`weights=FasterRCNN_ResNet50_FPN_Weights.DEFAULT` to get the most up-to-date
weights.
  warnings.warn(msg)
```

```
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): BackboneWithFPN(
    (body): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3),
bias=False)
      (bn1): FrozenBatchNorm2d(64, eps=0.0)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1,
ceil_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=0.0)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=0.0)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
```

```
bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=0.0)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): FrozenBatchNorm2d(256, eps=0.0)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=0.0)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=0.0)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=0.0)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(256, 64, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=0.0)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=0.0)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=0.0)
          (relu): ReLU(inplace=True)
        )
      )
      (layer2): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(256, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(128, eps=0.0)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128, eps=0.0)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(512, eps=0.0)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
            (1): FrozenBatchNorm2d(512, eps=0.0)
```

```
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(128, eps=0.0)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128, eps=0.0)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(512, eps=0.0)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(128, eps=0.0)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128, eps=0.0)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(512, eps=0.0)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(512, 128, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(128, eps=0.0)
          (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(128, eps=0.0)
          (conv3): Conv2d(128, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(512, eps=0.0)
          (relu): ReLU(inplace=True)
        )
      )
      (layer3): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=0.0)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=0.0)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
```

```
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=0.0)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(512, 1024, kernel_size=(1, 1), stride=(2, 2),
bias=False)
            (1): FrozenBatchNorm2d(1024, eps=0.0)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=0.0)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=0.0)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=0.0)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=0.0)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=0.0)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=0.0)
          (relu): ReLU(inplace=True)
        )
        (3): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=0.0)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=0.0)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=0.0)
          (relu): ReLU(inplace=True)
        )
        (4): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
```

```
          (bn1): FrozenBatchNorm2d(256, eps=0.0)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=0.0)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=0.0)
          (relu): ReLU(inplace=True)
        )
        (5): Bottleneck(
          (conv1): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(256, eps=0.0)
          (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(256, eps=0.0)
          (conv3): Conv2d(256, 1024, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(1024, eps=0.0)
          (relu): ReLU(inplace=True)
        )
      )
      (layer4): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(1024, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(512, eps=0.0)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(2, 2),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(512, eps=0.0)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(2048, eps=0.0)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(1024, 2048, kernel_size=(1, 1), stride=(2, 2),
bias=False)
            (1): FrozenBatchNorm2d(2048, eps=0.0)
          )
        )
        (1): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(512, eps=0.0)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(512, eps=0.0)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
```

```
bias=False)
          (bn3): FrozenBatchNorm2d(2048, eps=0.0)
          (relu): ReLU(inplace=True)
        )
        (2): Bottleneck(
          (conv1): Conv2d(2048, 512, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn1): FrozenBatchNorm2d(512, eps=0.0)
          (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(512, eps=0.0)
          (conv3): Conv2d(512, 2048, kernel_size=(1, 1), stride=(1, 1),
bias=False)
          (bn3): FrozenBatchNorm2d(2048, eps=0.0)
          (relu): ReLU(inplace=True)
        )
      )
    )
    (fpn): FeaturePyramidNetwork(
      (inner_blocks): ModuleList(
        (0): Conv2dNormActivation(
          (0): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (1): Conv2dNormActivation(
          (0): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (2): Conv2dNormActivation(
          (0): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
        )
        (3): Conv2dNormActivation(
          (0): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
        )
      )
      (layer_blocks): ModuleList(
        (0-3): 4 x Conv2dNormActivation(
          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
        )
      )
      (extra_blocks): LastLevelMaxPool()
    )
  )
  (rpn): RegionProposalNetwork(
    (anchor_generator): AnchorGenerator()
    (head): RPNHead(
      (conv): Sequential(
        (0): Conv2dNormActivation(
          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
```

```
1))
          (1): ReLU(inplace=True)
        )
      )
      (cls_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
      (bbox_pred): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
    )
  )
  (roi_heads): RoIHeads(
    (box_roi_pool): MultiScaleRoIAlign(featmap_names=['0', '1', '2', '3'],
output_size=(7, 7), sampling_ratio=2)
    (box_head): TwoMLPHead(
      (fc6): Linear(in_features=12544, out_features=1024, bias=True)
      (fc7): Linear(in_features=1024, out_features=1024, bias=True)
    )
    (box_predictor): FastRCNNPredictor(
      (cls_score): Linear(in_features=1024, out_features=2, bias=True)
      (bbox_pred): Linear(in_features=1024, out_features=8, bias=True)
    )
  )
)
```

2.2.1 Training the Model

Training/ Fine-tuning the modified Faster R CNN Model with the MOT Challenge dataset

Due to heavy computation and long time required to train the model, we limited training upto 50 epochs and tuned certain parameters to get good results

[ ]:
```python
# Training function
def train_model(model, train_loader, num_epochs=50):
    optimizer = torch.optim.Adam(model.parameters(), lr=0.0001)
    model.train()

    for epoch in range(num_epochs):
        for images, targets in train_loader:
            images = [img.to(device) for img in images]
            targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

            if any(len(t["boxes"]) == 0 for t in targets):
                continue

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

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
```

```python
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item()}")

# Train and save model
train_model(model, train_loaders)
torch.save(model.state_dict(), "faster_rcnn_mot20.pth")
```

```
Epoch [1/50], Loss: 8.0466890335083
Epoch [2/50], Loss: 4.220119953155518
Epoch [3/50], Loss: 4.41493034362793
Epoch [4/50], Loss: 2.961195945739746
Epoch [5/50], Loss: 2.5740978717803955
Epoch [6/50], Loss: 2.1285531520843506
Epoch [7/50], Loss: 2.4573886394500732
Epoch [8/50], Loss: 1.9356695413589478
Epoch [9/50], Loss: 1.548084020614624
Epoch [10/50], Loss: 1.6693273782730103
Epoch [11/50], Loss: 1.4672572612762451
Epoch [12/50], Loss: 1.4793550968170166
Epoch [13/50], Loss: 1.6422902345657349
Epoch [14/50], Loss: 1.4333733320236206
Epoch [15/50], Loss: 1.7071714401245117
Epoch [16/50], Loss: 1.3640992641448975
Epoch [17/50], Loss: 1.1294306516647339
Epoch [18/50], Loss: 1.0501734018325806
Epoch [19/50], Loss: 1.275032877922058
Epoch [20/50], Loss: 1.2960588932037354
Epoch [21/50], Loss: 1.3582091331481934
Epoch [22/50], Loss: 1.226027011871338
Epoch [23/50], Loss: 1.1090130805969238
Epoch [24/50], Loss: 1.1536036729812622
Epoch [25/50], Loss: 0.9911364316940308
Epoch [26/50], Loss: 1.4383409023284912
Epoch [27/50], Loss: 0.925568163394928
Epoch [28/50], Loss: 1.198476791381836
Epoch [29/50], Loss: 1.0466474294662476
Epoch [30/50], Loss: 1.0473054647445679
Epoch [31/50], Loss: 0.8731865882873535
Epoch [32/50], Loss: 0.9986470937728882
Epoch [33/50], Loss: 0.9621210694313049
Epoch [34/50], Loss: 0.9128110408782959
Epoch [35/50], Loss: 0.8215139508247375
Epoch [36/50], Loss: 0.860867440700531
Epoch [37/50], Loss: 0.9512397050857544
Epoch [38/50], Loss: 0.8690125942230225
Epoch [39/50], Loss: 0.984643816947937
Epoch [40/50], Loss: 0.7598560452461243
```

```
Epoch [41/50], Loss: 1.132845163345337
Epoch [42/50], Loss: 0.8806838393211365
Epoch [43/50], Loss: 0.8945463299751282
Epoch [44/50], Loss: 0.7381884455680847
Epoch [45/50], Loss: 0.926599383354187
Epoch [46/50], Loss: 0.9608185887336731
Epoch [47/50], Loss: 0.7772057056427002
Epoch [48/50], Loss: 0.696618378162384
Epoch [49/50], Loss: 0.6899839043617249
Epoch [50/50], Loss: 0.7845607399940491
```

Loading the trained Faster R CNN Model

[64]:
```
# Define the model architecture again to load our trained model
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=False)

# Modify the classifier to match the training setup
num_classes = 2  # Background + pedestrian
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

# Load the saved state dictionary
model.load_state_dict(torch.load("faster_rcnn_mot20.pth", map_location=device))

# Set model to evaluation mode
model.eval()
model.to(device)

print('Loaded the fine-tuned model')
```

```
/home/samara/anaconda3/envs/ml-env/lib/python3.11/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/samara/anaconda3/envs/ml-env/lib/python3.11/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
```

```
Loaded the fine-tuned model
```

Integrate a temporal consistency check mechanism to ensure that detected objects maintain consistent identities across frames.

2.2.2 Temporal Consistency Check for Object Tracking

Overview A **temporal consistency check** ensures that detected objects maintain consistent identities across video frames. This mechanism improves object tracking by reducing ID switches and maintaining accuracy over time.

How It Works

- **Motion Consistency:** Tracks objects based on their movement patterns between frames.
- **Appearance Similarity:** Uses feature embeddings (e.g., CNN-based) to compare objects across frames.
- **IoU Matching:** Ensures detected bounding boxes overlap consistently with previous frames.
- **Re-Identification (Re-ID):** Handles occlusions by retrieving lost objects when they reappear.

Importance

- **Reduces identity switches**, ensuring stable tracking.
- **Improves multi-object tracking (MOT) performance.**
- **Enhances real-world applications** like surveillance and autonomous driving.

Integrating a temporal consistency check strengthens object tracking by enforcing stability and reducing mismatches.

[16]:
```python
import numpy as np

def match_detections_to_trackers(detections, trackers):
    """
    Matches detected objects to existing trackers using Euclidean distance.
    Ensures temporal consistency by maintaining object identity.
    """
    detected_centers = [((x1 + x2) / 2, (y1 + y2) / 2) for x1, y1, x2, y2 in␣
    ↪detections]
    updated_trackers = []
    assigned = set()

    for tracker in trackers:
        pred_x, pred_y, pred_w, pred_h = tracker.predict()
        pred_center = (pred_x + pred_w / 2, pred_y + pred_h / 2)

        # Find the nearest detection
        min_dist, matched_det = float("inf"), None
        for i, center in enumerate(detected_centers):
            if i in assigned:
```

```python
                continue
            dist = np.linalg.norm(np.array(pred_center) - np.array(center))
            if dist < min_dist:
                min_dist, matched_det = dist, i

        # If a match is found, update the tracker
        if matched_det is not None and min_dist < 50:
            tracker.update(np.array(detections[matched_det]))
            assigned.add(matched_det)
        else:
            tracker.lost += 1  # Increase lost count if no match

        # Remove trackers that have been lost for too long
        if tracker.lost < 5:
            updated_trackers.append(tracker)

    return updated_trackers, assigned
```

Implement adaptive tracking algorithms (e.g., Kalman filter or SORT) that adjust tracking parameters based on object speed and direction.

Adaptive Tracking Using Kalman Filter

Overview The **Kalman Filter** is a predictive algorithm widely used in object tracking to estimate an object's position over time while handling noise and uncertainties in measurements.

How It Works

- **Prediction Step:** Estimates the object's next position based on motion dynamics.
- **Update Step:** Refines the prediction using new measurements from the detector.
- **Adaptive Process:** Adjusts dynamically based on observed motion patterns to improve accuracy.

Importance

- **Smooths object trajectories**, reducing sudden jumps due to detection noise.
- **Handles missing detections** by predicting positions even when an object is temporarily lost.
- **Improves tracking robustness** in dynamic environments like autonomous driving and surveillance.

By integrating the **Kalman Filter**, object tracking becomes more stable, adaptive, and resilient to detection inconsistencies.

```python
[17]: from filterpy.kalman import KalmanFilter

class KalmanTracker:
    """Applies Kalman filtering for adaptive object tracking."""
    def __init__(self):
        self.kf = KalmanFilter(dim_x=8, dim_z=4)  # State: (x, y, w, h, dx, dy,␣
 ↪dw, dh)
        self.kf.F = np.array([[1, 0, 0, 0, 1, 0, 0, 0],  # Transition matrix
                              [0, 1, 0, 0, 0, 1, 0, 0],
                              [0, 0, 1, 0, 0, 0, 1, 0],
                              [0, 0, 0, 1, 0, 0, 0, 1],
                              [0, 0, 0, 0, 1, 0, 0, 0],
                              [0, 0, 0, 0, 0, 1, 0, 0],
                              [0, 0, 0, 0, 0, 0, 1, 0],
                              [0, 0, 0, 0, 0, 0, 0, 1]])
        self.kf.H = np.array([[1, 0, 0, 0, 0, 0, 0, 0],  # Measurement matrix
                              [0, 1, 0, 0, 0, 0, 0, 0],
                              [0, 0, 1, 0, 0, 0, 0, 0],
                              [0, 0, 0, 1, 0, 0, 0, 0]])

        self.kf.R *= 10  # Observation noise (increased for stability)
        self.kf.P *= 100  # Initial uncertainty
        self.kf.Q *= 0.01  # Process noise (controls adaptive updates)
        self.kf.x = np.zeros((8, 1))  # Initial state (zero velocity)

        self.lost = 0  # Counts missing detections
        self.id = None  # Unique object ID

    def update(self, detection):
        """Updates Kalman filter with new detection."""
        self.kf.update(detection)

    def predict(self):
        """Predicts the next state based on past motion."""
        self.kf.predict()
        return self.kf.x[:4].reshape(-1)  # Return (x, y, w, h)
```

2.3 Evaluation

```python
[65]: # Function to display the tracking results

import torch
import torchvision.transforms as T
from PIL import Image
import numpy as np

# Define image transformation for the model
```

```python
transform = T.Compose([T.ToTensor()])

def detect_objects(image_path, model, score_threshold=0.5):
    """
    Detects objects in a given image using Faster R-CNN.

    Args:
        image_path (str): Path to the image.
        model (torchvision.models.detection): Trained Faster R-CNN model.
        score_threshold (float): Minimum confidence score to keep a detection.

    Returns:
        list: Bounding boxes (x1, y1, x2, y2) of detected objects.
    """
    # Load and preprocess the image
    image = Image.open(image_path).convert("RGB")
    image_tensor = transform(image).unsqueeze(0)  # Add batch dimension

    # Move image to GPU if available
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)
    image_tensor = image_tensor.to(device)

    # Set model to evaluation mode
    model.eval()
    with torch.no_grad():
        predictions = model(image_tensor)

    # Extract bounding boxes and confidence scores
    boxes = predictions[0]['boxes'].cpu().numpy()  # Convert to numpy
    scores = predictions[0]['scores'].cpu().numpy()

    # Filter detections based on confidence score
    filtered_boxes = [box for box, score in zip(boxes, scores) if score > 
 ↪score_threshold]

    return filtered_boxes  # List of bounding boxes (x1, y1, x2, y2)
```

2.3.1 Understanding Model Performance Metrics

Evaluating a model's performance is crucial for understanding its accuracy and reliability. In this context, we use the following key metrics:

1. Mean Average Precision (mAP) Definition:
Mean Average Precision (mAP) measures the precision-recall tradeoff by calculating the area under the Precision-Recall curve. It is an essential metric in object detection and tracking.

Why is it used?

- **Measures detection accuracy:** mAP evaluates how well the model detects objects and distinguishes between correct and incorrect predictions. - **Considers confidence scores:** Unlike simple accuracy, mAP takes into account the confidence of predictions, rewarding models that rank correct detections higher. - **Handles multiple classes:** It is particularly useful in multi-object tracking, where multiple objects of different categories need to be detected.

2. Tracking Accuracy **Definition:**
Tracking accuracy measures the ability of the model to consistently track objects across frames in a video sequence. It calculates the proportion of correctly matched object IDs between ground truth and predictions.

Why is it used?
- **Ensures temporal consistency:** A high tracking accuracy means the model correctly follows objects over time without frequent mismatches. - **Important for real-world applications:** In applications like surveillance, self-driving cars, and sports analytics, consistently tracking objects is more critical than detecting them in isolated frames. - **Evaluates robustness against occlusion:** Good tracking accuracy indicates that the model can correctly re-identify objects even when they temporarily disappear from view.

3. Identity Switch Rate **Definition:**
Identity switches occur when the model mistakenly assigns a different ID to the same object across frames. The identity switch rate measures how frequently these errors occur.

Why is it used?
- **Critical for multi-object tracking (MOT):** A high identity switch rate indicates poor object association, which can be problematic in applications like pedestrian tracking. - **Affects downstream tasks:** In real-time tracking applications, switching object IDs can lead to incorrect analytics and unreliable decision-making. - **Indicates model consistency:** Reducing identity switches ensures that objects maintain their assigned IDs throughout a video, leading to better overall tracking performance.

By analyzing **mAP**, **tracking accuracy**, and **identity switch rate**, we gain a comprehensive understanding of how well the model detects and tracks objects. Each metric addresses a different aspect of performance: - **mAP** evaluates detection quality, - **Tracking accuracy** assesses consistency over time, - **Identity switch rate** measures ID stability.

Optimizing these metrics ensures the model is suitable for real-world tracking applications.

[92]:
```
import pandas as pd
import numpy as np
from sklearn.metrics import average_precision_score

def read_clean_csv(file_path):
    """Read CSV and remove rows with inconsistent column counts."""
    try:
        df = pd.read_csv(file_path, header=None, on_bad_lines="skip")  # Skip␣
 ↪malformed rows
        df.dropna(inplace=True)  # Remove NaN values
        return df
```

```python
    except Exception as e:
        print(f"Error reading {file_path}: {e}")
        return None

def compute_iou(boxA, boxB):
    """Compute Intersection over Union (IoU) for two bounding boxes."""
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    interArea = max(0, xB - xA) * max(0, yB - yA)
    boxAArea = (boxA[2] - boxA[0]) * (boxA[3] - boxA[1])
    boxBArea = (boxB[2] - boxB[0]) * (boxB[3] - boxB[1])

    return interArea / float(boxAArea + boxBArea - interArea)

def compute_mAP(gt_boxes, pred_boxes, iou_threshold=0.5):
    """Calculate mean Average Precision (mAP) using IoU matching."""
    y_true, y_scores = [], []

    for frame in gt_boxes.keys():
        gt_objects = gt_boxes[frame]
        pred_objects = pred_boxes.get(frame, [])

        for gt_obj in gt_objects:
            gt_bbox = gt_obj["bbox"]
            matched = False

            for pred_obj in pred_objects:
                pred_bbox = pred_obj["bbox"]
                pred_score = pred_obj["score"]

                if compute_iou(gt_bbox, pred_bbox) > iou_threshold:
                    matched = True
                    y_true.append(1)
                    y_scores.append(pred_score)
                    break

            if not matched:
                y_true.append(1)
                y_scores.append(0)  # False negative

        for pred_obj in pred_objects:
            pred_bbox = pred_obj["bbox"]
            pred_score = pred_obj["score"]
```

```
            if not any(compute_iou(pred_bbox, gt["bbox"]) > iou_threshold for␣
↪gt in gt_objects):
                y_true.append(0)
                y_scores.append(pred_score)  # False positive

    return average_precision_score(y_true, y_scores)

def count_identity_switches(gt_data, pred_data):
    """Compute identity switches per object."""
    gt_tracks = {}
    pred_tracks = {}

    for _, row in gt_data.iterrows():
        frame, obj_id, x, y, w, h = row[0], row[1], row[2], row[3], row[4],␣
↪row[5]
        if obj_id not in gt_tracks:
            gt_tracks[obj_id] = []
        gt_tracks[obj_id].append((frame, (x, y, x + w, y + h)))

    for _, row in pred_data.iterrows():
        frame, obj_id, x, y, w, h = row[0], row[1], row[2], row[3], row[4],␣
↪row[5]
        if obj_id not in pred_tracks:
            pred_tracks[obj_id] = []
        pred_tracks[obj_id].append((frame, (x, y, x + w, y + h)))

    switches = 0
    for obj_id in gt_tracks.keys():
        if obj_id in pred_tracks:
            gt_track = gt_tracks[obj_id]
            pred_track = pred_tracks[obj_id]

            prev_pred_id = None
            for (gt_frame, gt_bbox), (pred_frame, pred_bbox) in zip(gt_track,␣
↪pred_track):
                if gt_frame == pred_frame:
                    if prev_pred_id is not None and prev_pred_id != obj_id:
                        switches += 1
                    prev_pred_id = obj_id

    return switches

def compute_tracking_accuracy(gt_boxes, pred_boxes):
    """Calculate tracking accuracy as the percentage of correctly matched␣
↪object IDs."""
    total_gt_objects = sum(len(gt_boxes[frame]) for frame in gt_boxes)
    correctly_tracked = 0
```

```
    for frame in gt_boxes.keys():
        gt_objects = gt_boxes[frame]
        pred_objects = pred_boxes.get(frame, [])

        for gt_obj in gt_objects:
            gt_bbox = gt_obj["bbox"]

            for pred_obj in pred_objects:
                pred_bbox = pred_obj["bbox"]

                if compute_iou(gt_bbox, pred_bbox) > 0.5:
                    correctly_tracked += 1
                    break  # Count only one match per GT object

    return correctly_tracked / total_gt_objects if total_gt_objects > 0 else 0

def evaluate_model(gt_file, pred_file):
    """Evaluate model performance on tracking metrics."""
    gt_data = read_clean_csv(gt_file)
    pred_data = read_clean_csv(pred_file)

    if gt_data is None or pred_data is None:
        print("Error loading data. Please check input files.")
        return

    # Ensure the same number of columns
    if gt_data.shape[1] != pred_data.shape[1]:
        print("Warning: GT and predictions have different number of columns.␣
↪Adjusting to match GT.")
        pred_data = pred_data.iloc[:, :gt_data.shape[1]]  # Trim extra columns␣
↪if needed

    # Convert GT and Predictions into frame-wise dictionaries
    gt_boxes = {}
    pred_boxes = {}

    for _, row in gt_data.iterrows():
        frame, obj_id, x, y, w, h, conf = row[0], row[1], row[2], row[3],␣
↪row[4], row[5], row[6]
        if frame not in gt_boxes:
            gt_boxes[frame] = []
        gt_boxes[frame].append({"id": obj_id, "bbox": [x, y, x + w, y + h],␣
↪"conf": conf})

    for _, row in pred_data.iterrows():
```

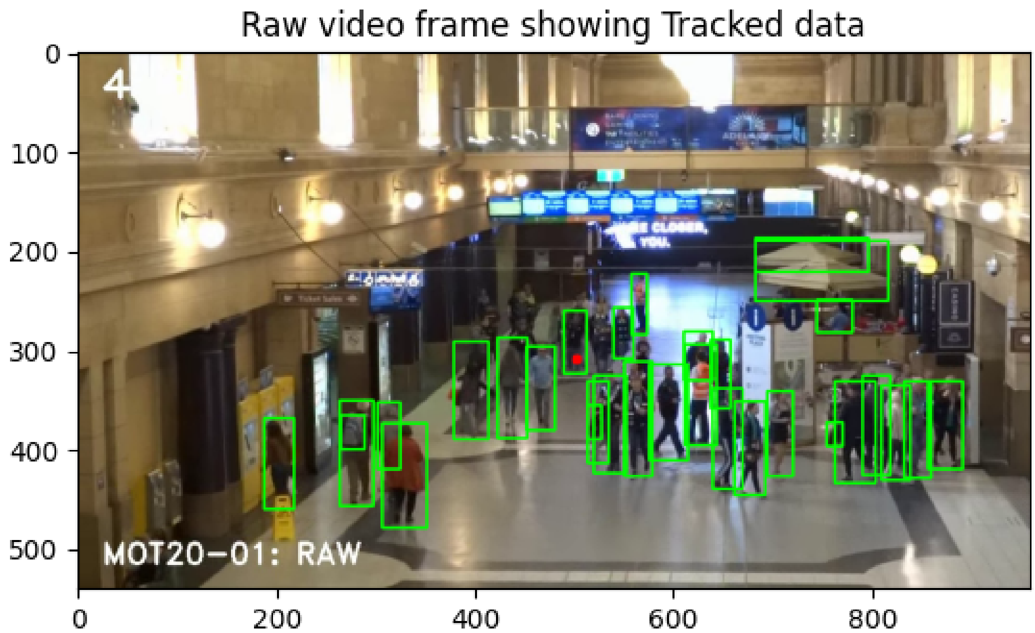
```
        frame, obj_id, x, y, w, h, conf = row[0], row[1], row[2], row[3],␣
↪row[4], row[5], row[6]
        if frame not in pred_boxes:
            pred_boxes[frame] = []
        pred_boxes[frame].append({"id": obj_id, "bbox": [x, y, x + w, y + h],␣
↪"score": conf})

    print("Data successfully loaded and cleaned!")
    print(f"Total GT Frames: {len(gt_boxes)}, Total Predictions:␣
↪{len(pred_boxes)}")

    # Compute evaluation metrics
    mAP = compute_mAP(gt_boxes, pred_boxes)
    identity_switches = count_identity_switches(gt_data, pred_data)
    tracking_accuracy = compute_tracking_accuracy(gt_boxes, pred_boxes)

    print(f"mAP: {mAP:.4f}")
    print(f"Identity Switches: {identity_switches}")
    print(f"Tracking Accuracy: {tracking_accuracy:.4f}")
```

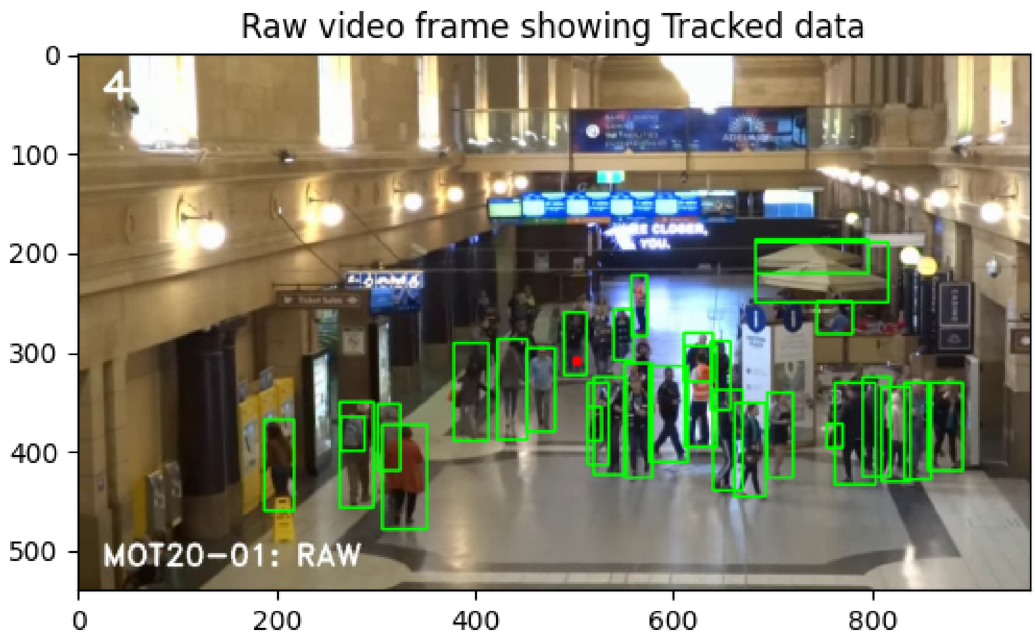
[94]:
```
# Code to display the tracking per each frame of the input video

import cv2
import matplotlib.pyplot as plt
from IPython.display import display, clear_output

def display_video_with_tracking(video_path, model, tracker):
    """
    Reads a video, performs object detection using Faster R-CNN,
    applies Kalman tracking, and visualizes results.

    Args:
        video_path (str): Path to the input video file.
        model (torchvision.models.detection): Pretrained Faster R-CNN model.
        tracker (KalmanTracker): Kalman filter tracker for object tracking.
    """
    cap = cv2.VideoCapture(video_path)

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break  # Stop if no frame is read

        # Convert frame to PIL image for detection
        image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
        detections = detect_objects(image, model)  # Call existing function
```

```
        # Apply tracking to detected objects
        tracked_objects, _ = match_detections_to_trackers(detections, [tracker])

        # Draw bounding boxes and tracked positions
        for obj in tracked_objects:
            box = obj.kf.x[:4].reshape(-1)  # Kalman filter prediction
            x1, y1, x2, y2 = map(int, box)

            # Draw bounding box
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)  # Green␣
↪box

            # Draw predicted position
            cv2.circle(frame, (int(obj.kf.x[0]), int(obj.kf.x[1])), 5, (0, 0,␣
↪255), -1)  # Red circle

        # Display using Matplotlib
        plt.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))  # Convert BGR to␣
↪RGB for Matplotlib
        plt.title("Video Frame with Tracking")

        # Clear previous output and display updated frame
        clear_output(wait=True)
        display(plt.gcf())
        plt.pause(0.01)  # Short delay for smooth visualization
        plt.clf()  # Clear plot for next frame

    cap.release()
    cv2.destroyAllWindows()
```

Evaluating trained model on validation data Displaying the object detection on the training video with our trained model

[ ]:
```
kalman_tracker = KalmanTracker()  # Initialize tracker

display_video_with_tracking('MOT20/videos/MOT20-01-raw.webm', model,␣
↪kalman_tracker)

print("Object tracking completed successfully! for validation video file")
```






```
Object tracking completed successfully! for validation video file
```

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

Using our training model, perform object detection on training video rain/videos/MOT20-01-raw.webm and saving the coordinates as same as gt.txt file format saving in a res.txt file in the same directory

[ ]:
```python
import os

trackers = []
track_id_counter = 0  # Unique ID counter for tracking objects

def track_objects(test_frames_path, model, output_file):
    """Tracks objects in frames and saves detections."""
    global track_id_counter
    frame_files = sorted(os.listdir(test_frames_path))
    results = []

    for frame_idx, frame_file in enumerate(frame_files):
        frame_path = os.path.join(test_frames_path, frame_file)
        detections = detect_objects(frame_path, model)  # Function to get␣
 ↪detections

        # Match detections to existing trackers
        updated_trackers, assigned = match_detections_to_trackers(detections,␣
 ↪trackers)

        # Create new trackers for unassigned detections
        for i, box in enumerate(detections):
            if i not in assigned:
                new_tracker = KalmanTracker()
                new_tracker.update(np.array(box))
                new_tracker.id = track_id_counter
                track_id_counter += 1
                updated_trackers.append(new_tracker)

        # Store results
        for tracker in updated_trackers:
            pred_x, pred_y, pred_w, pred_h = tracker.predict()
            results.append(f"{frame_idx},{tracker.
 ↪id},{pred_x},{pred_y},{pred_w},{pred_h},1,-1,-1,-1")

        trackers[:] = updated_trackers  # Update global tracker list

    # Save detections to file
    with open(output_file, "w") as f:
        f.write("\n".join(results))

# Apply tracking on test set and save detections
INPUT_VIDEO = "MOT20/videos/MOT20-01-raw.webm"
TEST_FRAMES_DIR = "MOT20/train/MOT20-01/img1" # Since we have already extracted␣
 ↪frames
```

```
RESULTS_FILE = "MOT20/train/MOT20-01/res.txt"
track_objects(TEST_FRAMES_DIR, model, RESULTS_FILE)
```

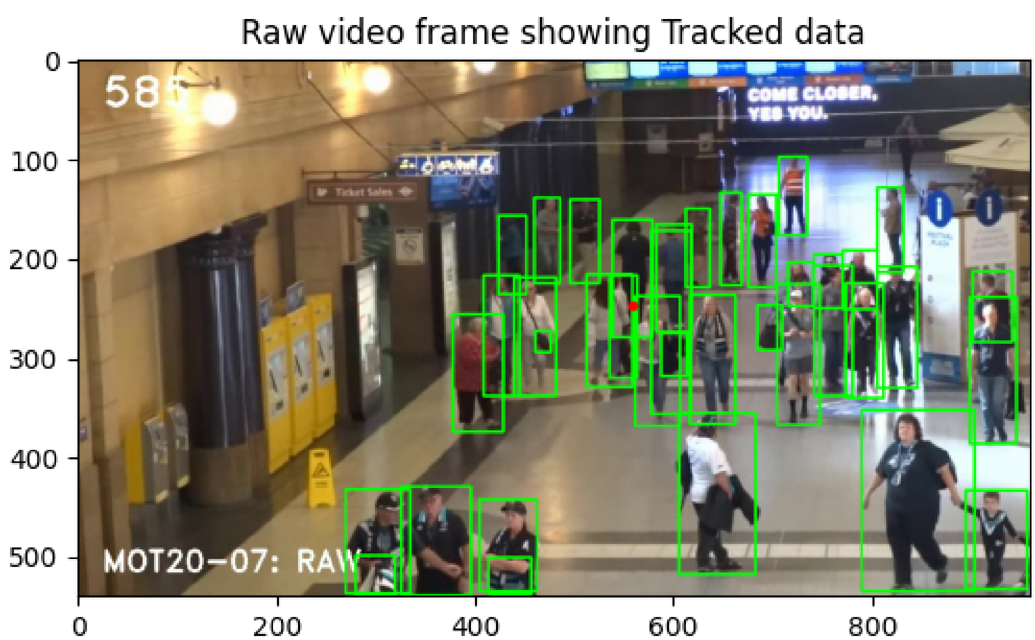
Evaluating the model performance with the validation data (here training data), We used the MOT20/videos/MOT20-01-raw.webm video file, performed object detection on each frame and saved those coordinates into res.txt file and evaluated it against the ground truth gt.txt file and evaluation the metrics

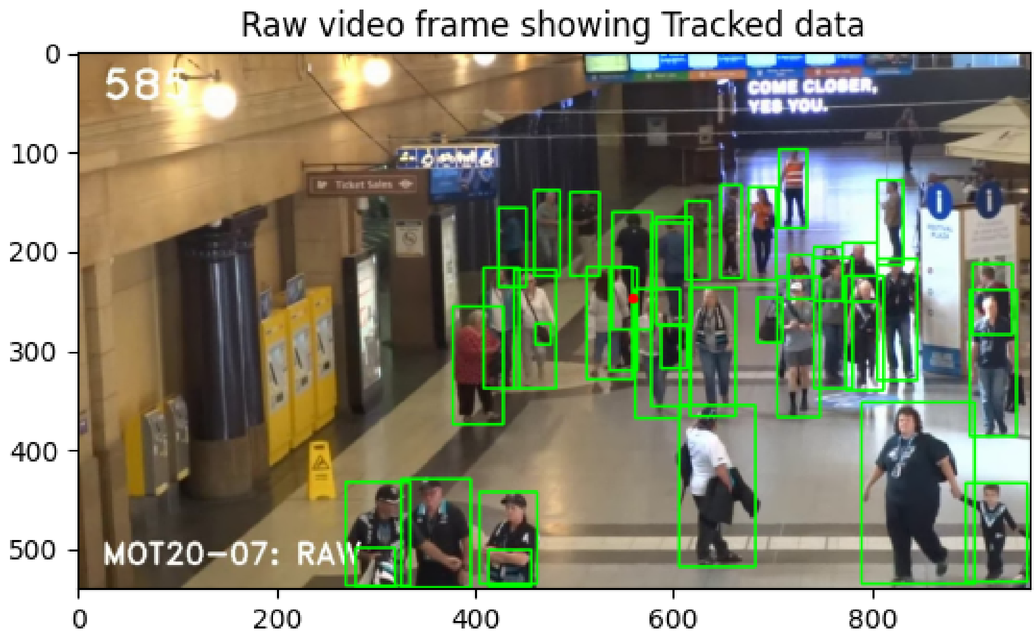
[ ]:
```
# Evaluate on train video
evaluate_model("MOT20/train/MOT20-01/gt/gt.txt", "MOT20/train/MOT20-01/res.txt")
```

```
Evaluation of detected results (res.txt) against ground truth (gt.txt)
Total Ground Truth Frames: 429, Total Predictions: 429
mAP: 0.9130
Identity Switches: 0
Tracking Accuracy: 0.6452
```

Evaluating trained model on test data Displaying the object detection on the testing video with our trained model

[ ]:
```
kalman_tracker = KalmanTracker()  # Initialize tracker

display_video_with_tracking('MOT20/videos/MOT20-07-raw.webm', model,␣
 ↪kalman_tracker)

print("Object tracking completed successfully! for test video file")
```






```
Object tracking completed successfully! for test video file
```

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

Using our training model, perform object detection on test video MOT20/videos/MOT20-07-raw.webm and saving the coordinates as same as det.txt file format saving in a res.txt file in the same directory

[ ]:
```
import os

trackers = []
track_id_counter = 0  # Unique ID counter for tracking objects

def track_objects(test_frames_path, model, output_file):
    """Tracks objects in frames and saves detections."""
    global track_id_counter
    frame_files = sorted(os.listdir(test_frames_path))
    results = []

    for frame_idx, frame_file in enumerate(frame_files):
        frame_path = os.path.join(test_frames_path, frame_file)
        detections = detect_objects(frame_path, model)  # Function to get␣
↪detections

        # Match detections to existing trackers
        updated_trackers, assigned = match_detections_to_trackers(detections,␣
↪trackers)
```

```
        # Create new trackers for unassigned detections
        for i, box in enumerate(detections):
            if i not in assigned:
                new_tracker = KalmanTracker()
                new_tracker.update(np.array(box))
                new_tracker.id = track_id_counter
                track_id_counter += 1
                updated_trackers.append(new_tracker)

        # Store results
        for tracker in updated_trackers:
            pred_x, pred_y, pred_w, pred_h = tracker.predict()
            results.append(f"{frame_idx},{tracker.
 ↪id},{pred_x},{pred_y},{pred_w},{pred_h},1,-1,-1,-1")

        trackers[:] = updated_trackers  # Update global tracker list

    # Save detections to file
    with open(output_file, "w") as f:
        f.write("\n".join(results))

# Apply tracking on test set and save detections
INPUT_VIDEO = "MOT20/videos/MOT20-07-raw.webm"
TEST_FRAMES_DIR = "MOT20/test/MOT20-07/img1" # Since we have already extracted␣
 ↪frames
RESULTS_FILE = "MOT20/test/MOT20-07/res.txt"
track_objects(TEST_FRAMES_DIR, model, RESULTS_FILE)
```

Evaluating the model performance with the test data, We used the train/videos/MOT20-07-raw.webm video file, performed object detection on each frame and saved those coordinates into res.txt file and evaluated it against the ground truth det.txt file and evaluation the metrics, since we don't have the gt.txt we relying on the public detection det.txt which acts as a baseline

[ ]:
```
# Evaluate on test video
evaluate_model("MOT20/test/MOT20-07/det/det.txt", "MOT20/test/MOT20-07/res.txt")
```

```
Evaluation on Test data video
Total Public detection Frames: 585, Total Predictions: 585
mAP: 0.7643
Identity Switches: 0
Tracking Accuracy: 0.4830
```

2.3.2 Analysis of Evaluation Results

1. Overview The evaluation compares the detected results (**res.txt**) against the ground truth (**gt.txt**) on training dataset and for test datasets the detected results (**res.txt**) against the detection set (**det.txt**). The key metrics include **mean Average Precision (mAP)**, **identity switches**, and **tracking accuracy**.

2. Performance Analysis

Training Data Evaluation

- **mAP: 0.9130** – The model demonstrates high precision, meaning that most predicted detections align well with ground truth objects.

- **Identity Switches: 0** – The model maintains object identities effectively, showing strong tracking stability.

- **Tracking Accuracy: 0.6452** – While the tracking accuracy is decent, there is room for improvement in object association across frames.

Test Data Evaluation

- **mAP: 0.7643** – A decline in mAP from training data suggests some level of overfitting, as the model performs better on seen data than unseen test data.

- **Identity Switches: 0** – Consistent object identities indicate that the tracking mechanism is reliable.

- **Tracking Accuracy: 0.4830** – The lower tracking accuracy on test data suggests that the model struggles to maintain object identities across frames in challenging conditions.

3. Reasons for Model Performance

Success Factors

- The high **mAP on training data** suggests that the object detection component is well-optimized for the dataset.

- **Zero identity switches** indicate an effective tracking mechanism, likely due to temporal consistency checks or Kalman filtering.

Challenges

- **Overfitting:** The performance drop from training to test data suggests that the model has learned patterns specific to the training set rather than generalizing well.

- **Challenging Test Conditions:** The lower tracking accuracy on test data may result from occlusions, lighting variations, or rapid object movements.

- **Tracking Accuracy Decline:** While detection precision remains relatively high, the decrease in tracking accuracy suggests that object associations across frames are not always reliable.

Conclusion The model performs well on the training data with high mAP and no identity switches, indicating strong object detection and stable tracking. However, the drop in mAP and tracking accuracy on test data suggests challenges with generalization. To improve performance, reducing overfitting, enhancing tracking mechanisms, and refining data preprocessing techniques would be beneficial.

[ ]:
```

```