基于kitti数据集训练简单的车辆感知模型

数据集：使用mini_kitti数据集进行训练，训练集包含20张图片，测试集包含5张图片

模型：使用的模型为pytorch的Faster R-CNN模型，训练10个epochs

训练代码

```
import os
import torch
import torchvision
import cv2
import numpy as np
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import functional as F
import torch.optim as optim
import matplotlib.pyplot as plt

device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')


# Step 1: Load KITTI Labels
def load_kitti_labels(file_path):
    labels = []
    with open(file_path, 'r') as f:
        for line in f:
            elements = line.split()
            if elements[0] == 'Car':  # Only select 'Car' category
                bbox = list(map(float, elements[4:8]))  # Extract bounding box
coordinates (xmin, ymin, xmax, ymax)
                if bbox[2] > bbox[0] and bbox[3] > bbox[1]:  # Ensure width and
height are positive
                    labels.append(bbox)
    return labels


# Step 2: Define KITTI Dataset
class KITTIDataset(Dataset):
    def __init__(self, images_dir, labels_dir=None, transform=None):
        self.images_dir = images_dir
        self.labels_dir = labels_dir
        self.transform = transform
        self.image_files = sorted(os.listdir(images_dir))

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

    def __getitem__(self, idx):
        img_path = os.path.join(self.images_dir, self.image_files[idx])
        image = cv2.imread(img_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        if self.labels_dir:
```

```
            label_path = os.path.join(self.labels_dir,
self.image_files[idx].replace('.png', '.txt'))
            labels = load_kitti_labels(label_path)
        else:
            labels = []

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

        # Convert labels to Torch Tensor
        if len(labels) == 0:
            # If no valid boxes are found, add a dummy box to prevent error
            boxes = torch.zeros((0, 4), dtype=torch.float32)  # No boxes
            labels = torch.zeros((0,), dtype=torch.int64)  # No labels
        else:
            boxes = torch.as_tensor(labels, dtype=torch.float32)
            labels = torch.ones((boxes.shape[0],), dtype=torch.int64)  # All
labels are 'Car'

        target = {'boxes': boxes, 'labels': labels}
        return F.to_tensor(image), target, img_path


# Step 3: Initialize Dataset and DataLoader
images_dir =
'../../../../kitti/mini_kitti/kitti_mini_data_object_image_2/training/image_2'
labels_dir =
'../../../../kitti/mini_kitti/kitti_mini_data_object_label_2/training/label_2'

dataset = KITTIDataset(images_dir, labels_dir)
dataloader = DataLoader(dataset, batch_size=4, shuffle=True, collate_fn=lambda x:
tuple(zip(*x)))

# Step 4: Load Pre-trained Faster R-CNN Model
from torchvision.models.detection import FasterRCNN_ResNet50_FPN_Weights

weights = FasterRCNN_ResNet50_FPN_Weights.DEFAULT
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(weights=weights)
num_classes = 2  # Background + Car
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)

# Step 5: Set Up Training Parameters
optimizer = optim.SGD(model.parameters(), lr=0.005, momentum=0.9,
weight_decay=0.0005)
num_epochs = 10

# Step 6: Training Loop
model.train()
for epoch in range(num_epochs):
    epoch_loss = 0
    for images, targets, _ in dataloader:
        images = [img.to(device) for img in images]
```

```
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        # Skip iterations with no valid targets
        if any(len(t['boxes']) == 0 for t in targets):
            continue

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

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

        epoch_loss += losses.item()

    print(f"Epoch {epoch+1}/{num_epochs}, Loss: {epoch_loss/len(dataloader)}")

# Step 7: Save the Trained Model
model_save_path = 'fasterrcnn_kitti_car_detector.pth'
torch.save(model.state_dict(), model_save_path)
print(f"Model saved to {model_save_path}")

# Step 8: Evaluation and Visualization on Test Dataset
test_images_dir =
'../../../../kitti/mini_kitti/kitti_mini_data_object_image_2/testing/image_2'
test_dataset = KITTIDataset(test_images_dir)
test_dataloader = DataLoader(test_dataset, batch_size=1, shuffle=False)

model.eval()
with torch.no_grad():
    for images, _, img_paths in test_dataloader:
        images = [img.to(device) for img in images]
        predictions = model(images)

        for i, prediction in enumerate(predictions):
            img = images[i].permute(1, 2, 0).cpu().numpy()
            img = (img * 255).astype(np.uint8)  # Convert from float tensor (0-1)
to uint8 (0-255)
            img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)  # Convert RGB to BGR for
OpenCV compatibility
            img_path = img_paths[i]
            boxes = prediction['boxes'].cpu().numpy()
            scores = prediction['scores'].cpu().numpy()

            # Draw boxes with scores above a threshold (e.g., 0.5)
            for box, score in zip(boxes, scores):
                if score > 0.5:
                    xmin, ymin, xmax, ymax = box
                    cv2.rectangle(img, (int(xmin), int(ymin)), (int(xmax),
int(ymax)), (0, 255, 0), 2)
                    cv2.putText(img, f'{score:.2f}', (int(xmin), int(ymin) - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

            # Convert BGR to RGB for visualization
```

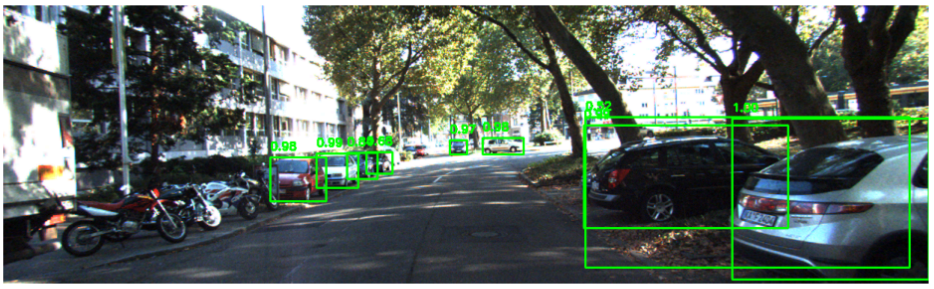
```
            img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

            # Display the image with matplotlib
            plt.figure(figsize=(12, 8))
            plt.imshow(img)
            plt.title(f"Detection Results - {os.path.basename(img_path)}")
            plt.axis('off')
            plt.show()
```

Loss

```
Epoch 1/10, Loss: 0.6422369360923768
Epoch 2/10, Loss: 0.5036078751087188
Epoch 3/10, Loss: 0.22570595741271973
Epoch 4/10, Loss: 0.2196572959423065
Epoch 5/10, Loss: 0.18737444877624512
Epoch 6/10, Loss: 0.19784102439880372
Epoch 7/10, Loss: 0.12938087582588195
Epoch 8/10, Loss: 0.16503628790378572
Epoch 9/10, Loss: 0.10648651123046875
Epoch 10/10, Loss: 0.0933348298072815
```

测试结果

Detection Results - 000002.png



Detection Results - 000003.png