**CODE 1**

import os

import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import models, transforms, datasets

from torch.utils.data import DataLoader

import matplotlib.pyplot as plt

def main():

# Set the device (CPU or GPU)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Define data transformations for training dataset

data\_transforms = transforms.Compose([

transforms.RandomResizedCrop(224),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

# Specify the data directory and create a data loader for training

data\_dir = 'D:/EDI\_code/Images/dataset 2'

train\_dataset = datasets.ImageFolder(os.path.join(data\_dir, 'train'), transform=data\_transforms)

train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True, num\_workers=4)

# Load the pre-trained ResNet-50 model

model = models.resnet50(pretrained=True)

# Fine-tune: Unfreeze certain layers for training

for param in model.parameters():

param.requires\_grad = False # Freeze all layers by default

for param in model.layer4.parameters():

param.requires\_grad = True # Unfreeze the last residual block for fine-tuning

num\_classes = 5

model.fc = nn.Linear(model.fc.in\_features, num\_classes)

# Send the model to the GPU if available

model = model.to(device)

# Define the loss function and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(filter(lambda p: p.requires\_grad, model.parameters()), lr=0.001, momentum=0.9)

# Training loop

num\_epochs = 5

train\_losses = [] # List to store train losses

for epoch in range(num\_epochs):

model.train()

running\_loss = 0.0

for inputs, labels in train\_loader:

inputs = inputs.to(device)

labels = labels.to(device)

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item() \* inputs.size(0)

epoch\_loss = running\_loss / len(train\_dataset)

train\_losses.append(epoch\_loss) # Append train loss to the list

print(f'Epoch {epoch + 1}/{num\_epochs} | Train Loss: {epoch\_loss:.4f}')

# Save the trained model weights

torch.save(model.state\_dict(), 'resnet50\_model\_finetuned.pth')

print("Training complete! Model saved as 'resnet50\_model\_finetuned.pth'")

# Plot the loss curve

plt.plot(range(1, num\_epochs + 1), train\_losses)

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.title('Train Loss Curve')

plt.show()

if \_name\_ == '\_main\_':

# Call the main function

main()

**CODE 2**

import os

import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, models, transforms

from torch.utils.data import DataLoader

def main():

# Set the device (CPU or GPU)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Rest of your code remains the same...

# Define data transformations for the test dataset

data\_transforms = transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

# Specify the path to the test dataset directory

test\_data\_dir = 'D:/EDI\_code/Images/dataset 2'

# Create a dataset and dataloader for testing

test\_dataset = datasets.ImageFolder(os.path.join(test\_data\_dir, 'test'), transform=data\_transforms)

test\_loader = DataLoader(test\_dataset, batch\_size=32, shuffle=False, num\_workers=4)

# Load the pre-trained ResNet-50 model

model = models.resnet50(pretrained=True)

num\_classes = 5

model.fc = nn.Linear(model.fc.in\_features, num\_classes)

# Load the trained model weights

model\_weights\_path = 'resnet50\_model\_finetuned.pth '

model.load\_state\_dict(torch.load(model\_weights\_path))

# Send the model to the GPU if available

model = model.to(device)

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

# Testing the trained model

test\_corrects = 0

with torch.no\_grad():

# Testing loop (assuming you have a test\_loader defined)

for inputs, labels in test\_loader:

inputs = inputs.to(device)

labels = labels.to(device)

outputs = model(inputs)

\_, preds = torch.max(outputs, 1)

test\_corrects += torch.sum(preds == labels.data)

test\_acc = 100 \* test\_corrects.double() / len(test\_dataset)

print(f'Test Accuracy: {test\_acc:.4f}%')

if \_name\_ == '\_main\_':

# Call the main function

main()

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**CODE 3**

import torch

import torch.nn as nn

from torchvision import models, transforms

from PIL import Image

# Define the device (GPU or CPU)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Define data transformations for inference

data\_transform = transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

# Load the pre-trained ResNet-50 model

model = models.resnet50(pretrained=True)

# Modify the fc layer to match the number of output classes (5)

num\_classes = 5

model.fc = nn.Linear(model.fc.in\_features, num\_classes)

# Load the trained model weights

model\_weights\_path = 'resnet50\_model\_finetuned.pth'

model.load\_state\_dict(torch.load(model\_weights\_path, map\_location=device))

# Send the model to the GPU if available

model = model.to(device)

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

# Load and preprocess the image you want to classify

image\_path = 'D:/EDI\_code/Images/dataset 2/train/class\_2(Traffic sign)/00000\_00004\_00016.png' # Replace with the path to your image

image = Image.open(image\_path)

image = data\_transform(image).unsqueeze(0) # Add a batch dimension

# Perform inference

with torch.no\_grad():

image = image.to(device)

output = model(image)

\_, predicted\_class = torch.max(output, 1)

# Map the predicted class index to a label (if you have a label mapping)

label\_mapping = {0: ' pedestrian', 1: 'Traffic sign', 2: 'car', 3: 'bike', 4: 'zebra crossing'}

predicted\_label = label\_mapping[predicted\_class.item()]

# Print the result

print(f"Predicted class index: {predicted\_class.item()}")

# Uncomment the line below if you have a label mapping

print(f"Predicted class label: {predicted\_label}")

**CODE 4**

import torch

from torchvision.models.detection import fasterrcnn\_resnet50\_fpn

from torchvision.transforms import functional as F

from PIL import Image

import cv2

from datetime import datetime

# Define the device (GPU or CPU)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Load the pre-trained Faster R-CNN model

model = fasterrcnn\_resnet50\_fpn(pretrained=True)

model = model.to(device)

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

# Open the video file for reading

video\_path = 'your\_video\_path.mp4'

cap = cv2.VideoCapture(video\_path)

# Check if the video file is opened successfully

if not cap.isOpened():

print("Error: Could not open video file.")

exit()

# Create the VideoWriter object to save the output video

output\_path = 'output\_video.mp4'

fps = int(cap.get(cv2.CAP\_PROP\_FPS))

frame\_width = int(cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

frame\_height = int(cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

fourcc = cv2.VideoWriter\_fourcc(\*'mp4v')

out = cv2.VideoWriter(output\_path, fourcc, fps, (frame\_width, frame\_height))

# Set a score threshold for detected objects

score\_threshold = 0.7

# Skip frames if needed (adjust this based on your real-time requirements)

frame\_skip = 2

frame\_count = 0

batch\_size = 4 # Adjust the batch size based on your hardware capabilities

time\_now\_1 = datetime.now()

print("Time when the video is given to test:", time\_now\_1)

frame\_buffer = [] # Buffer for frames to be processed in a batch

while cap.isOpened():

ret, frame = cap.read()

if not ret:

break

frame\_count += 1

# Skip frames based on frame\_skip value

if frame\_count % frame\_skip != 0:

continue

# Convert the frame to PIL Image and preprocess

image = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)).convert("RGB")

image\_tensor = F.to\_tensor(image).to(device)

# Append the frame to the buffer

frame\_buffer.append(image\_tensor)

# Process frames in batches

if len(frame\_buffer) >= batch\_size:

with torch.no\_grad():

predictions = model([frame.to(device) for frame in frame\_buffer])

# Iterate through batched predictions

for i, prediction in enumerate(predictions):

boxes = prediction[0]['boxes']

labels = prediction[0]['labels']

scores = prediction[0]['scores']

# Draw bounding boxes on the frame

for box, label, score in zip(boxes, labels, scores):

if score > score\_threshold:

box = [round(coord, 2) for coord in box.tolist()]

cv2.rectangle(frame, (int(box[0]), int(box[1]), int(box[2]), int(box[3]), (255, 0, 0), 2))

label\_text = f"Class {label.item()}",

cv2.putText(frame, label\_text, (int(box[0]), int(box[1]) - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5,

(0, 0, 255), 2),

# Write the frame with bounding boxes to the output video

out.write(frame),

# Clear the frame buffer

frame\_buffer.clear(),

# Release the video capture and video writer

cap.release(),

out.release(),

--------------------------------------------------------------------

**CODE 5**

import os

import torch

import torch.nn as nn

from torchvision import models, transforms, datasets

from torch.utils.data import DataLoader

from sklearn.metrics import classification\_report, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

def evaluate\_model(model, dataloader, device):

model.eval()

all\_preds = []

all\_labels = []

with torch.no\_grad():

for inputs, labels in dataloader:

inputs, labels = inputs.to(device), labels.to(device)

outputs = model(inputs)

\_, predicted = torch.max(outputs, 1)

all\_preds.extend(predicted.cpu().numpy())

all\_labels.extend(labels.cpu().numpy())

return all\_labels, all\_preds

def print\_metrics\_and\_confusion\_matrix(all\_labels, all\_preds, class\_names):

# Calculate metrics

print("Classification Report:")

classification\_rep = classification\_report(all\_labels, all\_preds, target\_names=class\_names)

print(classification\_rep)

# Create confusion matrix

conf\_matrix = confusion\_matrix(all\_labels, all\_preds)

# Plot confusion matrix

plt.figure(figsize=(10, 8))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues',

xticklabels=class\_names, yticklabels=class\_names)

plt.xlabel('Predicted')

plt.ylabel('True')

plt.title('Confusion Matrix')

plt.show()

def main():

# Set the device (CPU or GPU)

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Define data transformations for test dataset

data\_transforms = transforms.Compose([

transforms.Resize(256),

transforms.CenterCrop(224),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

# Specify the data directory and create a data loader for test

data\_dir = 'D:/EDI\_code/Images/dataset 2'

test\_dataset = datasets.ImageFolder(os.path.join(data\_dir, 'test'), transform=data\_transforms)

test\_loader = DataLoader(test\_dataset, batch\_size=32, shuffle=False, num\_workers=4)

# Load your pre-trained model

model = models.resnet50(pretrained=False) # Assuming you're using ResNet-50 architecture

num\_classes = len(test\_dataset.classes)

model.fc = nn.Linear(model.fc.in\_features, num\_classes)

model.load\_state\_dict(torch.load('resnet50\_model\_finetuned.pth')) # Replace 'your\_model.pth' with the actual file path

model = model.to(device)

# Evaluate the model on the test dataset

all\_labels, all\_preds = evaluate\_model(model, test\_loader, device)

# Print metrics and confusion matrix for each class

print\_metrics\_and\_confusion\_matrix(all\_labels, all\_preds, test\_dataset.classes)

if \_name\_ == '\_main\_':

# Call the main function

main()