```
import os
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms, models
from torch.utils.data import Dataset, DataLoader
import cv2
import matplotlib.pyplot as plt
from PIL import Image

labeled_data_dir = "C:/Users/Admin/Downloads/data1/train_with"
unlabeled_data_dir = "C:/Users/Admin/Downloads/res"
model_save_path = "C:/Users/Admin/Downloads/tennis_ball_model.pth"
batch_size = 32
num_epochs = 10
fine_tuning_epochs = 5
learning_rate = 0.001

# _____Преобразования данных
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# _____Кастомный датасет с метками
class LabeledDataset(Dataset):
    def __init__(self, image_dir, label_dir, transform=None):
        self.image_dir = image_dir
        self.label_dir = label_dir
        self.transform = transform
        self.image_files = [f for f in os.listdir(image_dir) if f.endswith('.jpg')]

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

    def __getitem__(self, idx):
        image_path = os.path.join(self.image_dir, self.image_files[idx])
        label_path = os.path.join(self.label_dir, self.image_files[idx].replace('.jpg', '.t

        # _____Чтение изображения с использованием OpenCV#
        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = Image.fromarray(image)
        labels = []
        with open(label_path, 'r') as f:
            for line in f:
                parts = line.strip().split()
                class_id = int(parts[0])
```

```
                labels.append(class_id)

        labels = torch.tensor(labels[0], dtype=torch.long)

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

        return image, labels

# _____________________________________________________Датасет без меток#
class UnlabeledDataset(Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        self.image_files = [os.path.join(data_dir, f) for f in os.listdir(data_dir) if f.en

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

    def __getitem__(self, idx):
        image_path = self.image_files[idx]

        image = cv2.imread(image_path)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        image = Image.fromarray(image)

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

        return image

# ____________________________________Загрузка данных#
train_dataset = LabeledDataset(
    image_dir=os.path.join(labeled_data_dir, "train", "images"),
    label_dir=os.path.join(labeled_data_dir, "train", "labels"),
    transform=transform
)
val_dataset = LabeledDataset(
    image_dir=os.path.join(labeled_data_dir, "valid", "images"),
    label_dir=os.path.join(labeled_data_dir, "valid", "labels"),
    transform=transform
)
unlabeled_dataset = UnlabeledDataset(
    data_dir=unlabeled_data_dir,
    transform=transform
)

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False)
unlabeled_loader = DataLoader(unlabeled_dataset, batch_size=batch_size, shuffle=False)
```

```
unlabeled_loader = DataLoader(unlabeled_dataset, batch_size=batch_size, shuffle=False)

# _____________________________________________Проверка загрузки#
print(f"Количество тренировочных данных: {len(train_dataset)}")
print(f"Количество валидационных данных: {len(val_dataset)}")
print(f"Количество данных без меток: {len(unlabeled_dataset)}")

# _____________________________________________Определение модели#
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)
model.fc = nn.Linear(model.fc.in_features, 2)
model = model.to(device)

# _____________________________________________Функция потерь и оптимизатор#
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# _____________________________________________Функция обучения#
def train_epoch(model, loader, criterion, optimizer):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for inputs, labels in loader:
        inputs, labels = inputs.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = outputs.max(1)
        total += labels.size(0)
        correct += predicted.eq(labels).sum().item()

    epoch_loss = running_loss / len(loader)
    epoch_acc = 100.0 * correct / total

    return epoch_loss, epoch_acc

# _____________________________________________Функция валидации#
def validate_epoch(model, loader, criterion):
    model.eval()
    running_loss = 0.0
    correct = 0
    total = 0

    with torch.no_grad():
```

```
        for inputs, labels in loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)

            running_loss += loss.item()
            _, predicted = outputs.max(1)
            total += labels.size(0)
            correct += predicted.eq(labels).sum().item()

    epoch_loss = running_loss / len(loader)
    epoch_acc = 100.0 * correct / total

    return epoch_loss, epoch_acc

# _____________________________________________________________Цикл обучения
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []

for epoch in range(num_epochs):
    train_loss, train_acc = train_epoch(model, train_loader, criterion, optimizer)
    val_loss, val_acc = validate_epoch(model, val_loader, criterion)

    train_losses.append(train_loss)
    val_losses.append(val_loss)
    train_accuracies.append(train_acc)
    val_accuracies.append(val_acc)

    print(f"Эпоха {epoch + 1}/{num_epochs}")
    print(f"  Train Loss: {train_loss:.4f}, Train Accuracy: {train_acc:.2f}%")
    print(f"  Val Loss: {val_loss:.4f}, Val Accuracy: {val_acc:.2f}%")

# _____________________________________________________Визуализация обучения
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Val Loss')
plt.title('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, label='Train Accuracy')
plt.plot(val_accuracies, label='Val Accuracy')
plt.title('Accuracy')
plt.legend()

plt.show()

# _____________________________________________________Сохранение модели
torch.save(model.state_dict(), model_save_path)
print(f"Модель сохранена по пути: {model_save_path}")
```

```
print(f"Модель сохранена по пути: {model_save_path}")
```