به نام خدا

گزارش پیاده سازی بخش های اول تا سوم:

```
import torch
import torch.nn as nn
import torchvision.models as models
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader, random_split
from PIL import Image
import os
import cv2
import zipfile
```

torch.nn: ماژول شبکه عصبی **PyTorch**. کلاس هایی را برای ساختن لایه ها (مانند **nn.Linear** برای لایه های کاملاً متصل، **nn.Conv2d** برای لایه های کانولوشن)، توابع فعال سازی (**nn.ReLU**) و توابع خطا (**nn.CrossEntropyLoss**) ارائه می دهد.

torchvision.models: مدل های از پیش آموزش دیده ای مانند **ResNet**، **VGG** و غیره را ارائه می دهد که می توانید مستقیماً از آنها استفاده کرد.

torchvision.transforms: تبدیل‌های رایج تصویر مانند تغییر اندازه (تغییر اندازه، تغییر اندازه)، تبدیل به تنسور (**transforms.ToTensor**) و نرمال‌سازی (**transforms.Normalize**) را ارائه می‌دهد.

torch.utils.data.Dataset: یک کلاس انتزاعی که یک مجموعه داده را نشان می دهد. شما مجموعه داده های سفارشی را با ارث بردن از این کلاس ایجاد می کنید.

torch.utils.data.DataLoader: یک داده قابل تکرار را روی یک مجموعه داده ارائه می دهد، دسته بندی، جابجایی، و بارگذاری موازی داده را مدیریت می کند.

torch.utils.data.random_split: یک تابع ابزار برای تقسیم تصادفی یک مجموعه داده به مجموعه های آموزشی و آزمایشی (یا اعتبار سنجی).

PIL (Pillow): کتابخانه ای برای دستکاری تصویر (باز کردن، ذخیره و غیره).

os: توابعی را برای تعامل با سیستم عامل ارائه می دهد (به عنوان مثال، فهرست کردن فایل ها در یک فهرست).

cv2 (OpenCV): کتابخانه ای برای وظایف بینایی کامپیوتر، از جمله پردازش ویدئو.

zipfile: ماژولی برای کار با آرشیوهای ZIP.

```
def unzip_dataset(zip_file_path, extract_path):

    with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
        zip_ref.extractall(extract_path)
    print(f"Dataset extracted to: {extract_path}")
```

در این قسمت صرفا لایه ورودی را از مسیری که مشخص کردیم دریافت میکنیم و آن را **unzip** میکنیم و داده های آن قابل استفاده خواهند بود.

کلاس Hockey:

در این کلاس ما با دریافت ورودی ها به دنبال استخراج فریم ها هستیم که با استفاده از متغیر extract_frames سراغ آن میرویم. ویدیو ها را از طریق cv2 میخوانیم و درون فریم ها iterate میکنیم و فریم ها را استخراج میکنیم و به شکل RGB تبدیل میکنیم. و در نهایت به PIL عکس تبدیل کرده و به لیست اضافه میکنیم. و لیست فریم های استخراج شده را برمیگردانیم.

```
data_transform = transforms.Compose([
    transforms.Resize((224, 224)),  # ResNet50 input size
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]), # ImageNet normalization
])
```

در این قسمت یک دنباله ای از تبدیلات عکس میسازیم. سایز فریم ها را به ۲۲٤ پیکسل تغییر میدهیم و در نهایت هر PIL را به تنسور تبدیل میکنیم. در مرحله بعد نرمال سازی را انجام میدهیم.

در مرحله بعد صرفا داده را unzip کردیم برای اینکه بنده از google colab استفاده میکردیم و مجبور بودم با خود فایل zip مستقیم کار کنم.

```
    num_samples = len(hockey_dataset)
    train_size = num_samples // 2
    test_size = num_samples - train_size  # The rest for testing
    train_dataset, test_dataset = random_split(hockey_dataset, [train_size, test_size])

    train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=8, shuffle=False)
```

در این قسمت داده ها را همانطور که از ما خواسته بودند به مجموعه های **train, test** تقسیم کردیم. برای مجموعه آموزشی داده ها را درهم سازی میکنیم.

```
resnet50 = models.resnet50(pretrained=True)

feature_extractor = nn.Sequential(*list(resnet50.children())[:-1])

feature_extractor.eval()
```

مدل از پیش آموزش دیده شده **resnet** را لود کردیم.

یک مدل جدید ساختیم که همه لایه های آن جز لایه آخری قرار دارند. این کار را برای استخراج ویژگی انجام داده ایم.

در خط بعدی مدل را به حالت **evalutation** میبریم زیرا که اگر به لایه های دارای **dropout** برخورد کرد تاثیر نپذیرد.

```
def extract_features(data_loader, feature_extractor):
    features_list = []
    labels_list = []
    with torch.no_grad():
        for video_frames, labels in data_loader:
            batch_size, num_frames, C, H, W = video_frames.shape
            video_frames = video_frames.view(batch_size * num_frames, C, H, W)
            features = feature_extractor(video_frames)
            features = features.view(batch_size, num_frames, -1)
            video_features = torch.mean(features, dim=1) # Average pooling
            features_list.append(video_features)
            labels_list.append(labels)
    all_features = torch.cat(features_list, dim=0)
    all_labels = torch.cat(labels_list, dim=0)
    return all_features, all_labels
```

Extract_features(data_loader، feature_extractor): این تابع ویژگی ها را از تمام ویدیوها در data_loader داده شده استخراج می کند.

با torch.no_grad():: محاسبات گرادیان را در حین استخراج ویژگی غیرفعال می کند (چون ما مدل ResNet50 را آموزش نمی دهیم).

حلقه از طریق دسته ها: کد از طریق دسته های ارائه شده توسط data_loader تکرار می شود.

Reshape: video_frames به گونه‌ای تغییر شکل داده می‌شود که تمام فریم‌ها از همه ویدیوها در دسته به‌عنوان یک دسته بزرگ از تصاویر در نظر گرفته می‌شوند (این امر ضروری است زیرا ResNet50 ورودی ٤ بعدی را انتظار دارد: batch_size، کانال‌ها، ارتفاع، عرض).

استخراج ویژگی: feature_extractor(video_frames) فریم ها را از feature_extractor عبور می دهد تا ویژگی ها را به دست آورد.

Reshape back: ویژگی‌ها تغییر شکل داده می‌شوند تا ساختار ویدئو حفظ شود.

میانگین : torch.mean (ویژگی‌ها، dim=1) میانگین ویژگی‌ها را در امتداد بعد num_frames محاسبه می‌کند و به طور موثر یک بردار ویژگی واحد برای هر ویدیو ایجاد می‌کند.

افزودن به لیست : ویژگی ها و برچسب های هر دسته به **features_list** و **labels_list** اضافه می شوند.

Concatenate: در نهایت، **torch.cat** تمام ویژگی‌ها و برچسب‌ها را از دسته‌های مختلف به تنسورهای منفرد (**all_features**، **all_labels**) متصل می‌کند.

در قسمت بعدی مقادیر را بروز رسانی کردیم.

در قسمت بعدی کلاس بندی جدید خودمان را ساختیم و خروجی را گزارش دادیم.