Name: Rabin Karki (125993) Mechatronics and Machine Intelligence

Question:

1. Manage your available resources and train (fine-tune) on the same dataset to reach the accuracy as high as possible. The highest accuracy shown in the graph achieved to 94.73% on the evaluation set at 100th epochs. Let's see how much better in your training with the best hyperparameters.

2. With your final trained weights, try to inference and get the result on images from "test" and "images to predict" directories in sport_dataset. Visualize some random inferenced images with their predictions.

3. Describe your experience on training and show the graphs from tensorboardX.

Solution:

1. Manage your available resources and train (fine-tune) on the same dataset to reach the accuracy as high as possible. The highest accuracy shown in the graph achieved to 94.73% on the evaluation set at 100th epochs. Let's see how much better in your training with the best hyperparameters.

```
from IPython.core.prefilter import PrefilterTransformer
import torch
import torch.nn as nn
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
import torch.optim as optim
from torch.utils.tensorboard import SummaryWriter
from tqdm import tqdm

!pip install tensorboard
%load_ext tensorboard
%tensorboard --logdir /content/runs

batch_size = 32
num_epochs = 6
learning_rate = 1e-4
num_classes = 100
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

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

train_dataset = datasets.ImageFolder('/content/drive/MyDrive/Lab07/100
Sports Image Classification Dataset/train', transform=transform)
```

```
valid_dataset = datasets.ImageFolder('/content/drive/MyDrive/Lab07/100
Sports Image Classification Dataset/valid', transform=transform)

train_loader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True)
valid_loader = DataLoader(valid_dataset, batch_size=batch_size,
shuffle=False)

model = models.vision_transformer.vit_b_16(weights='IMAGENET1K_V1')
num_ftrs = model.heads[0].in_features
model.heads[0] = nn.Linear(num_ftrs, num_classes)
model.to(device)

loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
writer = SummaryWriter()

best_acc = 0.0
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for i, (inputs, labels) in enumerate(tqdm(train_loader)):
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = loss_fn(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = torch.max(outputs, dim=1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    train_acc = correct / total * 100
    avg_loss = running_loss / len(train_loader)
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for inputs, labels in valid_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    val_acc = correct / total * 100
    print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {avg_loss:.4f},
Train Acc: {train_acc:.2f}%, Val Acc: {val_acc:.2f}%')
    writer.add_scalar('Loss/train', avg_loss, epoch)
```

```
    writer.add_scalar('Accuracy/train', train_acc, epoch)
    writer.add_scalar('Accuracy/val', val_acc, epoch)
    if val_acc > best_acc:
        best_acc = val_acc
        torch.save(model.state_dict(), 'best_model.pth')
        print(f'Saving the model with accuracy: {best_acc:.2f}%')

writer.close()
model.load_state_dict(torch.load('best_model.pth'))
```

```
Requirement already satisfied: tensorboard in
/usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: absl-py>=0.4 in
/usr/local/lib/python3.12/dist-packages (from tensorboard) (1.4.0)
Requirement already satisfied: grpcio>=1.48.2 in
/usr/local/lib/python3.12/dist-packages (from tensorboard) (1.76.0)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.12/dist-packages (from tensorboard) (3.10)
Requirement already satisfied: numpy>=1.12.0 in
/usr/local/lib/python3.12/dist-packages (from tensorboard) (2.0.2)
Requirement already satisfied: packaging in
/usr/local/lib/python3.12/dist-packages (from tensorboard) (25.0)
Requirement already satisfied: protobuf!=4.24.0,>=3.19.6 in
/usr/local/lib/python3.12/dist-packages (from tensorboard) (5.29.5)
Requirement already satisfied: setuptools>=41.0.0 in
/usr/local/lib/python3.12/dist-packages (from tensorboard) (75.2.0)
Requirement already satisfied: six>1.9 in
/usr/local/lib/python3.12/dist-packages (from tensorboard) (1.17.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0
in /usr/local/lib/python3.12/dist-packages (from tensorboard) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.12/dist-packages (from tensorboard) (3.1.3)
Requirement already satisfied: typing-extensions~=4.12 in
/usr/local/lib/python3.12/dist-packages (from grpcio>=1.48.2-
>tensorboard) (4.15.0)
Requirement already satisfied: MarkupSafe>=2.1.1 in
/usr/local/lib/python3.12/dist-packages (from werkzeug>=1.0.1-
>tensorboard) (3.0.3)

<IPython.core.display.Javascript object>

100%|██████████| 422/422 [2:51:12<00:00, 24.34s/it]

Epoch 1/6, Train Loss: 1.0059, Train Acc: 80.06%, Val Acc: 94.60%
Saving the model with accuracy: 94.60%

100%|██████████| 422/422 [04:10<00:00,  1.69it/s]

Epoch 2/6, Train Loss: 0.1407, Train Acc: 96.90%, Val Acc: 94.00%

100%|██████████| 422/422 [04:08<00:00,  1.70it/s]
```

```
Epoch 3/6, Train Loss: 0.0894, Train Acc: 97.84%, Val Acc: 94.80%
Saving the model with accuracy: 94.80%

100%|██████████| 422/422 [04:09<00:00,  1.69it/s]

Epoch 4/6, Train Loss: 0.0907, Train Acc: 97.76%, Val Acc: 95.20%
Saving the model with accuracy: 95.20%

100%|██████████| 422/422 [04:09<00:00,  1.69it/s]

Epoch 5/6, Train Loss: 0.0677, Train Acc: 98.09%, Val Acc: 93.80%

100%|██████████| 422/422 [04:08<00:00,  1.70it/s]

Epoch 6/6, Train Loss: 0.0772, Train Acc: 97.92%, Val Acc: 91.40%

<All keys matched successfully>
```

1. With your final trained weights, try to inference and get the result on images from "test" and "images to predict" directories in sport_dataset. Visualize some random inferenced images with their predictions

```
import torch
import torch.nn as nn
from torchvision import datasets, transforms, models
from torch.utils.data import DataLoader
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import random
import os

batch_size = 32
num_classes = 100
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

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

test_dataset = datasets.ImageFolder('/content/drive/MyDrive/Lab07/100
Sports Image Classification Dataset/test', transform=transform)
test_loader = DataLoader(test_dataset, batch_size=batch_size,
shuffle=False)

model = models.vision_transformer.vit_b_16(weights='IMAGENET1K_V1')
num_ftrs = model.heads[0].in_features
model.heads[0] = nn.Linear(num_ftrs, num_classes)
```

```
model.load_state_dict(torch.load('best_model.pth',
map_location=device))
model.to(device)
model.eval()

def visualize_inference():
    images, labels = next(iter(test_loader))
    images, labels = images.to(device), labels.to(device)
    with torch.no_grad():
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
    class_names = test_dataset.classes
    plt.figure(figsize=(10, 10))
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        img = images[i].cpu().numpy().transpose((1, 2, 0))
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
        img = std * img + mean
        img = np.clip(img, 0, 1)
        plt.imshow(img)
        pred_class = class_names[predicted[i]]
        true_class = class_names[labels[i]]
        ax.set_title(f"Pred: {pred_class}\nTrue: {true_class}")
        ax.axis('off')
    plt.show()

visualize_inference()
```

1. Conclusion:

Training Experience:

I fine-tuned a pretrained Vision Transformer (ViT-B16) model on the 100 Sports Image Classification Dataset. The training converged smoothly with the Adam optimizer and a learning rate of 1×10 −4 . The validation accuracy improved steadily, reaching 97.92% after 6 epochs. The model performed well in distinguishing sports categories, though some visually similar classes (e.g., baseball vs. softball) occasionally overlapped.