Manahil Fatima Anwar

20K-0134

BAI-7A

Lab 11 - Vision in Transformers

```
import torch
import torch.nn as nn
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
from torchvision.datasets import CIFAR10
from sklearn.model_selection import train_test_split
import timm
```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])
```

```
cifar_dataset = CIFAR10(root="./data", train=True, transform=transform, download=True)
```

```
Files already downloaded and verified
```

```
subset_size = 5000
cifar_dataset = torch.utils.data.random_split(cifar_dataset, [subset_size, len(cifar_dataset) - subset_size])[0]
```

```
train_set, val_set = train_test_split(cifar_dataset, test_size=0.1, random_state=42)
```

```
batch_size = 32
train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=4)
val_loader = DataLoader(val_set, batch_size=batch_size, shuffle=False, num_workers=4)
```

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: UserWarning: This DataLoader will create 4 worker processes
  warnings.warn(_create_warning_msg(
```

```
class ViT(nn.Module):
    def __init__(self, num_classes=10):
        super(ViT, self).__init__()

        self.model = timm.create_model("vit_small_patch16_224", pretrained=True)
        self.model.head = nn.Linear(self.model.head.in_features, num_classes)

    def forward(self, x):
        return self.model(x)

model = ViT().to(device)
```

```
model.safetensors:                                88.2M/88.2M [00:01<00:00,
```

```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=2e-4)
```

```
num_epochs = 5
accumulation_steps = 4

for epoch in range(num_epochs):
    model.train()
    for i, (images, labels) in enumerate(train_loader):
        images, labels = images.to(device), labels.to(device)

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

        if (i + 1) % accumulation_steps == 0:
            optimizer.step()
            optimizer.zero_grad()

    model.eval()
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)

            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()

        accuracy = correct / total
        print(f"Epoch {epoch+1}/{num_epochs}, Loss: {loss.item():.4f}, Accuracy: {accuracy:.4f}")

print("Training complete!")
```

```
Epoch 1/5, Loss: 0.2216, Accuracy: 0.8820
Epoch 2/5, Loss: 0.3163, Accuracy: 0.9240
Epoch 3/5, Loss: 0.4203, Accuracy: 0.9200
Epoch 4/5, Loss: 0.2218, Accuracy: 0.8920
Epoch 5/5, Loss: 0.0050, Accuracy: 0.9120
Training complete!
```

```
from PIL import Image


def classify_image(model, image_path, transform, class_labels):
    image = Image.open(image_path).convert("RGB")
    image = transform(image).unsqueeze(0).to(device)

    model.eval()
    with torch.no_grad():
        output = model(image)

    _, predicted_class_index = torch.max(output, 1)

    predicted_class_label = class_labels[predicted_class_index.item()]

    return predicted_class_label


cifar10_classes = [
    'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck'
]


image_path = "cat.jpg"
predicted_class_label = classify_image(model, image_path, transform, cifar10_classes)


print(f"The predicted class label for the image is: {predicted_class_label}")
```

```
The predicted class label for the image is: cat
```

```
image_path = "truck.jpg"
predicted_class_label = classify_image(model, image_path, transform, cifar10_classes)
print(f"The predicted class label for the image is: {predicted_class_label}")
```

```
The predicted class label for the image is: truck
```