به نام خدا

توضیح بخش د)

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
from torchvision import models
```

از optim برای بهینه سازی SGD و Adam استفاده کرده ایم.

از torchvision برای استفاده از دیتاست ها و مدل های از قبل آموزش دیده استفاده کردیم.

torchvision.transforms: برای تبدیل تصویر مانند تغییر اندازه، برش، عادی سازی، و غیره استفاده می شود.

matplotlib.pyplot: برای رسم نمودارها (منحنی خطا و دقت).

torchvision.models: مدل های از پیش آموزش دیده مانند VGG16، ResNet و غیره را ارائه می دهد (models.vgg16).

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

این خط بررسی می‌کند که آیا یک GPU با قابلیت CUDA در دسترس است یا خیر.

اگر یک GPU در دسترس باشد، دستگاه روی "cuda" تنظیم می شود تا PyTorch از GPU برای محاسبات استفاده کند.

در غیر این صورت، دستگاه برای استفاده از CPU روی "cpu" تنظیم شده است.

```python
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

full_dataset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                            download=True, transform=transform)
```

Python

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100%|██████████| 170M/170M [00:13<00:00, 12.9MB/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
```

Empty markdown cell, double-click or press enter to edit.

```python
train_size = int(0.8 * len(full_dataset))
val_size = int(0.1 * len(full_dataset))
test_size = len(full_dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = torch.utils.data.random_split(full_dataset, [train_size, val_size, test_size])

trainloader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
                                          shuffle=True, num_workers=2)
valloader = torch.utils.data.DataLoader(val_dataset, batch_size=32,
                                        shuffle=False, num_workers=2)
testloader = torch.utils.data.DataLoader(test_dataset, batch_size=32,
                                         shuffle=False, num_workers=2)
```

transform: دنباله ای از تبدیل ها را برای اعمال بر روی تصاویر تعریف می کند:

transforms.ToTensor(): تصاویر را از فرمت PIL به تنسور (Tensor) PyTorch تبدیل می کند.

transforms.Normalize((۰.۵، ۰.۵، ۰.۵)، (۰.۵، ۰.۵، ۰.۵)): مقادیر پیکسل هر کانال (R, G, B) را با استفاده از میانگین ۰.۵ و انحراف ۰.۵ نرمال می کند. این به بهبود آموزش کمک می کند.

full_dataset: مجموعه داده CIFAR-10 را با استفاده از torchvision.datasets.CIFAR10 بارگیری می کند.

train=True: بخش آموزشی مجموعه داده را بارگیری می کند.

download=True: مجموعه داده را در صورتی که قبلاً در فهرست اصلی وجود نداشته باشد دانلود می کند.

transform=transform: تبدیل های تعریف شده را برای هر تصویر اعمال می کند.

❖ تقسیم مجموعه داده:

این کد اندازه مجموعه های آموزشی (۸۰٪)، اعتبارسنجی (۱۰٪) و تست (۱۰٪) را محاسبه میکند.

torch.utils.data.random_split برای تقسیم تصادفی full_dataset به این سه زیر مجموعه استفاده می شود.

DataLoader: اشیاء DataLoader را برای هر تقسیم مجموعه داده (trainloader، valloader، testloader) ایجاد می کند.

batch_size=32: تعداد نمونه ها در هر دسته را مشخص می کند.

shuffle=True (برای آموزش): داده های آموزشی را در هر دوره به هم می ریزد.

shuffle=False (برای اعتبارسنجی و آزمایش): بدون نیاز به زدن اعتبارسنجی و آزمایش داده ها.

num_workers=2: از ۲ زیر فرآیند برای بارگذاری داده ها استفاده می کند (می تواند روند را تسریع کند).

model = models.vgg16(pretrained=True): مدل VGG16 را از torchvision.models بارگیری می کند.

pretrained=True: وزن های از پیش آموزش دیده را که در مجموعه داده ImageNet آموخته اند بارگیری و بارگیری می کند.

for param در model.parameters(): param.requires_grad = False: وزن تمام لایه ها را در مدل VGG16 ثابت می کند. این کار به این دلیل انجام می شود که می خواهیم از ویژگی های از پیش آموزش دیده استفاده کنیم و فقط لایه های طبقه بندی جدیدی را که اضافه می کنیم آموزش دهیم.

num_ftrs = model.classifier[6].in_features: تعداد ویژگی های ورودی آخرین لایه کاملاً متصل را در طبقه بندی کننده اصلی VGG16 دریافت می کند.

model.classifier = model.classifier[:-1]: آخرین لایه کاملا متصل طبقه بندی کننده اصلی VGG16 را حذف می کند.

new_classifier = nn.Sequential(...): با استفاده از nn.Sequential یک طبقه بندی جدید ایجاد می کند.

nn.Linear(num_ftrs, 512): یک لایه کاملا متصل که ویژگی های ورودی num_ftrs را می گیرد و ۵۱۲ ویژگی را خروجی می دهد.

nn.ReLU(): تابع فعال سازی ReLU را اعمال می کند.

nn.Linear(۵۱۲, ۱۰): یکی دیگر از لایه های کاملا متصل که ۵۱۲ ویژگی را به ۱۰ کلاس خروجی نگاشت می کند (برای CIFAR-10).

model.classifier = nn.Sequential(model.classifier, new_classifier): طبقه بندی کننده جدید را به قسمت طبقه بندی کننده موجود مدل VGG16 اضافه می کند.

model = model.to (دستگاه): مدل را به دستگاه مشخص شده (CPU یا GPU) منتقل می کند.

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.classifier.parameters(), lr=0.001, momentum=0.9)
```

criterion = nn.CrossEntropyLoss(): تابع خطا را به صورت CrossEntropyLoss تعریف می کند. این یک تابع از دست دادن متداول برای مسائل طبقه بندی چند کلاسه است.

optimizer = optim.SGD(...): با استفاده از Stochastic Gradient Descent (SGD) یک شی بهینه ساز ایجاد می کند.

model.classifier.parameters(): مشخص می کند که ما فقط می خواهیم پارامترهای لایه های طبقه بندی کننده جدید اضافه شده را بهینه کنیم (نه لایه های منجمد VGG16).

lr=0.001: نرخ یادگیری را روی ۰.۰۰۱ تنظیم می کند.

مومنتوم=۰.۹: مومنتوم را روی ۰.۹ تنظیم می کند (به شتاب دادن SGD در جهت مربوطه کمک می کند و نوسانات را کاهش می دهد).

بخش training and validation:

num_epochs = 10: تعداد دوره های آموزشی (چند بار تکرار در کل مجموعه داده آموزشی) را تنظیم می کند.

train_losses، val_losses، train_accuracies، val_accuracies: فهرست‌هایی برای ذخیره تلفات آموزشی و اعتبارسنجی و دقت برای هر دوره (برای ترسیم بعداً استفاده می‌شود).

حلقه آموزشی (حلقه بیرونی در طول دوره ها تکرار می شود):

model.train(): مدل را روی حالت آموزش قرار می دهد (برای لایه هایی مانند افت تحصیل و عادی سازی دسته ای که در طول آموزش و ارزیابی رفتار متفاوتی دارند مهم است).

حلقه داخلی (تکرار بر روی دسته در داده های آموزشی):

ورودی ها، برچسب ها = داده ها: دسته ای از تصاویر (ورودی ها) و برچسب های مربوط به آنها (برچسب ها) را دریافت می کند.

inputs, labels = inputs.to (دستگاه)، labels.to (دستگاه): داده ها را در صورت وجود به GPU منتقل می کند.

optimizer.zero_grad(): گرادیان های بهینه ساز را صفر می کند (مهم قبل از محاسبه گرادیان در هر تکرار).

خروجی ها = مدل(ورودی ها): برای بدست آوردن خروجی های پیش بینی شده، یک گذر رو به جلو از مدل انجام می دهد.

خطا = معیار (خروجی ها، برچسب ها): تلفات بین خروجی های پیش بینی شده و برچسب های واقعی را محاسبه می کند.

loss.backward(): پس انتشار را برای محاسبه گرادیان تلفات با توجه به پارامترهای مدل انجام می دهد.

optimizer.step(): پارامترهای مدل را بر اساس گرادیان های محاسبه شده با استفاده از بهینه ساز انتخاب شده (SGD در این مورد) به روز می کند.

run_loss += loss.item(): خطا را برای دسته فعلی جمع می کند.

_, predicted = torch.max(outputs.data , 1): برچسب های کلاس پیش بینی شده را با یافتن شاخص حداکثر مقدار در هر سطر خروجی دریافت می کند.

total += labels.size(۰): تعداد کل نمونه های پردازش شده را به روز می کند.

صحیح += (پیش‌بینی شده == برچسب‌ها).sum().item(): تعداد نمونه‌های طبقه‌بندی شده به‌درستی در دسته فعلی را شمارش می‌کند.

بعد از حلقه داخلی (پایان هر دوره):

train_loss = running_loss / len(trainloader): میانگین تلفات تمرین را برای دوره محاسبه می کند.

train_accuracy = 100 * correct / total: دقت تمرین را برای دوره محاسبه می کند.

train_losses.append(...), train_accuracies.append(...): تلفات و دقت محاسبه شده را ذخیره می کند.

❖ حلقه اعتبارسنجی:

model.eval(): مدل را روی حالت ارزیابی قرار می دهد.

با torch.no_grad():: محاسبات گرادیان را غیرفعال می کند (زیرا در حین اعتبارسنجی نیازی به آپدیت وزن مدل نداریم).

بقیه حلقه اعتبارسنجی بسیار شبیه به حلقه آموزشی است، اما از valloader استفاده می کند و وزن مدل را به روز نمی کند. از دست دادن اعتبار و دقت را محاسبه و ذخیره می کند.

بخش پلات کردن:

در این بخش صرفا از کتابخانه matplotlib استفاده کردیم و بر اساس خطا و accuracy را نمایش میدهیم.

```python
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        images, labels = data
        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()

test_accuracy = 100 * correct / total
print(f'Test Accuracy: {test_accuracy:.2f}%')
```

Python

```
Test Accuracy: 63.50%
```

مدل را روی حالت ارزیابی قرار می دهد.

محاسبات گرادیان را غیرفعال می کند.

از طریق تست لودر تکرار می شود.

دقت تست را به روشی مشابه حلقه اعتبارسنجی محاسبه می کند.

دقت تست نهایی را چاپ می کند.

```python
num_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print(f'Number of trainable parameters in the network: {num_params}')
```

Python

```
Number of trainable parameters in the network: 2102794
```

در اینجا پارامتر را از خروجی فایل قبلی بدست آوردیم

markdown

```python
xception_num_params = 20861480

print(f'Number of trainable parameters in VGG16 (after modification): {num_params}')
print(f'Number of trainable parameters in Xception (before modification): {xception_num_params}')

if num_params < xception_num_params:
    print("VGG16 has fewer trainable parameters than Xception (after our modifications).")
elif num_params > xception_num_params:
    print("VGG16 has more trainable parameters than Xception (after our modifications).")
else:
    print("VGG16 has the same number of trainable parameters as Xception (after our modifications).")
```

Python

num_params: برای p تعداد پارامترهای قابل آموزش در مدل را محاسبه می کند. فقط پارامترهایی را می‌شمارد که دارای requires_grad=True هستند (که در این مورد پارامترهای طبقه‌بندی‌کننده جدید هستند).

بقیه کد، تعداد پارامترها را در مدل اصلاح شده VGG16 با تعداد پارامترهای یک مدل Xception که قبلا آموزش داده شده مقایسه می کند (شما باید xception_num_params را با مقدار واقعی که از آموزش یک مدل Xception به دست آورده اید، جایگزین کنید).

خلاصه:

به طور کلی این کد با لود کردن مدل از قبل آموزش دیده شده VGG16 و ثابت نگه داشتن لایه های ابتدایی آن و لایه های کانولوشنی آن و حذف کردن لایه طبقه بندی آن برای اینکه با داده های خودمان مورد استفاده قرار بدهیم و کلاس بندی کنیم. همچنین یک طبقه بند مناسب برای داده های CIFAR-10 پیاده سازی کرده ایم.