

در این بخش شما باید یک شبکه عصبی کانولوشنال (CNN) را برای دسته‌بندی تصاویر CIFAR-10 با استفاده از PyTorch پیاده‌سازی کنید و عملکرد آن را با پرسپترون چندلایه قبلی مقایسه نمایید.

۱. تنظیم داده‌ها و بارگذاری CIFAR-10

مجموعه داده CIFAR-10 را با نرمال‌سازی مناسب بارگذاری کنید و های‌های DataLoader آموزش و تست را با اندازه بچ و شافل دلخواه ایجاد نمایید.

برنامه ۱: تعریف transforms و DataLoader

```
1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 import torch.nn.functional as F
5 from torchvision import datasets, transforms
6 from torch.utils.data import DataLoader
7
8 # Device configuration
9 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
10
11 # Hyper-parameters
12 num_epochs = 20
13 batch_size = 128
14 learning_rate = 0.01
15
16 # CIFAR-10 dataset transforms
17 transform = transforms.Compose([
18     transforms.ToTensor(),
19     transforms.Normalize((0.4914, 0.4822, 0.4465), (0.247, 0.243, 0.261))
20 ])
21
22 # CIFAR-10 dataset
23 train_dataset = datasets.CIFAR10(root='./data', train=True, download=True,
24     transform=transform)
25 test_dataset = datasets.CIFAR10(root='./data', train=False, download=True,
26     transform=transform)
27 print(train_dataset.classes)
28 # Data loaders
29 train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size,
30     shuffle=True, num_workers=4)
31 test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size,
32     shuffle=False, num_workers=4)
```

۲. تعریف معماری ساده CNN

یک کلاس PyTorch از نوع nn.Module با ساختار زیر پیاده‌سازی کنید:

- Conv2d با ۳۲ فیلتر، کرنل 3×3 ، فعال‌سازی ReLU
- MaxPool2d با کرنل 2×2
- Conv2d با ۶۴ فیلتر، کرنل 3×3 ، فعال‌سازی ReLU
- MaxPool2d با کرنل 2×2
- Flatten() و دو لایه تمام‌متصل ۱۲۸ و ۱۰ (نورونی Softmax خروجی)

برنامه ۲: کلاس SimpleCNN

```
1 class SimpleCNN(nn.Module):
2     def __init__(self):
3         super(SimpleCNN, self).__init__()
4         self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
5         self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
6         self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
7         self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
8         self.fc1 = nn.Linear(64 * 8 * 8, 128)
9         self.fc2 = nn.Linear(128, 10)
10
11     def forward(self, x):
12         x = F.relu(self.conv1(x))
13         x = self.pool1(x)
14         x = F.relu(self.conv2(x))
15         x = self.pool2(x)
16         x = x.view(x.size(0), -1)
17         x = F.relu(self.fc1(x))
18         x = self.fc2(x)
19         return x
```

۳. حلقه آموزش مدل

حلقه آموزشی را با تابع هزینه CrossEntropyLoss و بهینه‌ساز SGD با مومنتم پیاده کنید. در هر صد گام آموزشی، میانگین لوس را چاپ نمایید.

برنامه ۳: حلقه آموزش train loop

```
1 if __name__ == "__main__":
2     model = SimpleCNN().to(device)
3
4     # Loss and optimizer
5     criterion = nn.CrossEntropyLoss()
6     optimizer = optim.SGD(model.parameters(), lr=learning_rate, momentum=0.9)
7
8     # Training loop
9     for epoch in range(num_epochs):
10         model.train()
11         running_loss = 0.0
12         for i, (images, labels) in enumerate(train_loader):
13             images = images.to(device)
14             labels = labels.to(device)
15
16             # Forward pass
17             outputs = model(images)
18             loss = criterion(outputs, labels)
19
20             # Backward and optimize
21             optimizer.zero_grad()
22             loss.backward()
23             optimizer.step()
24
25             running_loss += loss.item()
26             if (i + 1) % 100 == 0:
27                 print(f'Epoch [{epoch+1}/{num_epochs}], Step
28                     [{i+1}/{len(train_loader)}], Loss: {running_loss / 100:.4f}')
```

۴. ارزیابی مدل و گزارش دقت

مدل آموزش دیده را روی مجموعه تست ارزیابی کنید و دقت نهایی را محاسبه و چاپ کنید. سپس وزن های مدل را ذخیره نمایید.

برنامه ۴: ارزیابی و ذخیره مدل

```

1 model.eval()
2 correct = 0
3 total = 0
4 with torch.no_grad():
5     for images, labels in test_loader:
6         images = images.to(device)
7         labels = labels.to(device)
8         outputs = model(images)
9         _, predicted = torch.max(outputs.data, 1)
10        total += labels.size(0)
11        correct += (predicted == labels).sum().item()
12
13    print(f'Test Accuracy of the model on the 10000 test images: {100 *
14    correct / total:.2f}%',)
15
16    # Save the model checkpoint
17    torch.save(model.state_dict(), 'simple_cnn_cifar10.pth')

```

۵. گزارش معماری و تحلیل نتایج

یک گزارش مختصر بنویسید که شامل معماری نهایی، جزئیات پیاده سازی، روند آموزش و نتایج ارزیابی باشد. همچنین مزایا و معایب استفاده از CNN را در مقایسه با پرسپترون چندلایه مورد بحث قرار دهید.

۶. مزایا و معایب استفاده از CNN در مقابل یک پرسپترون چندلایه

مزایا:

- **استخراج ویژگی های مکانی:** لایه های کانولوشنال با حفظ ساختار دوبعدی تصویر، الگوهای محلی مانند لبه ها و بافت ها را بهتر استخراج می کنند، در حالی که MLP تصویر را صاف شده دریافت می کند.
- **کاهش پارامترها:** با اشتراک وزن در فیلترها، تعداد پارامترها به طور قابل توجهی کمتر از MLP بوده و خطر بیش برآزش کاهش می یابد.
- **مقیاس پذیری:** استفاده از pooling و فیلترهای محلی باعث می شود CNN بتواند روی تصاویر بزرگ تر یا پیچیده تر نیز کارایی مناسبی داشته باشد.
- **تعمیم بهتر:** CNN در برابر تغییرات کوچک در تصویر (جابجایی، چرخش، تغییر نور) مقاوم تر است و به همین دلیل در داده های بصری عملکرد بهتری دارد.

معایب:

- **پیچیدگی پیاده سازی:** طراحی معماری CNN اعم از انتخاب تعداد لایه ها، فیلترها و سایر ابرپارامترها نیازمند تجربه و آزمون و خطای بیشتری است.
- **زمان آموزش:** محاسبات کانولوشن هزینه برتر از ضرب های برداری ساده در MLP بوده و ممکن است زمان آموزش طولانی تری داشته باشد.

- وابستگی به GPU: برای آموزش سریع و مؤثر CNN معمولاً نیاز به شتاب‌دهنده‌هایی مانند GPU است، در حالی که MLP را می‌توان حتی روی CPU اجرا کرد.