# 6.3 卷积神经网络识别fashion-MNIST

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import time

import copy

import torch

from torch import nn

import torch.nn.functional as F

import torch.optim as optim

import torchvision

import torch.utils.data as Data

from torchvision import transforms

import hiddenlayer as hl

from torchvision.datasets import FashionMNIST

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

train\_data = FashionMNIST(

root="./deep Learning/FashionMNIST", # 数据的路径

train=True, # 只使用训练数据集

# 将数据转化为torch使用的张量,取汁范围为［0，1］

transform=transforms.ToTensor(),

download=True # 因为数据已经下载过，所以这里不再下载

)

# 定义一个数据加载器

train\_loader = Data.DataLoader(

dataset=train\_data, # 使用的数据集

batch\_size=64, # 批处理样本大小

shuffle=True, # 每次迭代前打乱数据

num\_workers=0, # 使用两个进程

)

print('train\_loader的batch数量为：', len(train\_loader))

# train\_loader的batch数量为： 938

# 可视化训练数据集的一个batch的样本来查看图像内容

for step, (b\_x, b\_y) in enumerate(train\_loader):

if step > 0:

break

# 输出训练图像的尺寸和标签的尺寸，都是torch格式的数据

print(b\_x.shape)

print(b\_y.shape)

# torch.Size([64, 1, 28, 28])

# torch.Size([64])

# 显示一个batch的图像

batch\_x = b\_x.squeeze().numpy()

batch\_y = b\_y.numpy()

class\_label = train\_data.classes

class\_label[0] = 'T-shirt'

plt.figure(figsize=(12, 5))

for ii in np.arange(len(batch\_y)):

plt.subplot(4, 16, ii + 1)

plt.imshow(batch\_x[ii], cmap='gray')

plt.title(class\_label[batch\_y[ii]], fontsize=9)

plt.axis('off')

plt.subplots\_adjust(wspace=0.05)

plt.show() # 课本缺了一行show



test\_data = FashionMNIST(root="./deep Learning/FashionMNIST", train=False)

test\_data\_x = test\_data.data.type(torch.FloatTensor) / 255.0

test\_data\_x = torch.unsqueeze(test\_data\_x, dim=1)

test\_data\_y = test\_data.targets

print(test\_data\_x.shape)

print(test\_data\_y.shape)

# torch.Size([10000, 1, 28, 28])

# torch.Size([10000])

class MyConvNet(nn.Module):

def \_\_init\_\_(self):

super(MyConvNet, self).\_\_init\_\_()

self.conv1 = nn.Sequential(

nn.Conv2d(

in\_channels=1,

out\_channels=16,

kernel\_size=3,

stride=1,

padding=1

),

nn.ReLU(),

nn.AvgPool2d(

kernel\_size=2,

stride=2,

),

)

self.conv2 = nn.Sequential(

nn.Conv2d(16, 32, 3, 1, 0),

nn.ReLU(),

nn.AvgPool2d(2, 2)

)

self.classifier = nn.Sequential(

nn.Linear(32 \* 6 \* 6, 256),

# nn.Linear(32\*7\*7,128),

nn.ReLU(),

nn.Linear(256, 128),

nn.ReLU(),

nn.Linear(128, 10)

)

def forward(self, x):

x = self.conv1(x)

x = self.conv2(x)

x = x.view(x.size(0), -1) # 展开卷积层

output = self.classifier(x)

return output

myconvnet = MyConvNet()

print(myconvnet)

# MyConvNet(

# (conv1): Sequential(

# (0): Conv2d(1, 16, kernel\_size=(3, 3), stride=(1, 1), padding=(1, 1))

# (1): ReLU()

# (2): AvgPool2d(kernel\_size=2, stride=2, padding=0)

# )

# (conv2): Sequential(

# (0): Conv2d(16, 32, kernel\_size=(3, 3), stride=(1, 1))

# (1): ReLU()

# (2): AvgPool2d(kernel\_size=2, stride=2, padding=0)

# )

# (classifier): Sequential(

# (0): Linear(in\_features=1152, out\_features=256, bias=True)

# (1): ReLU()

# (2): Linear(in\_features=256, out\_features=128, bias=True)

# (3): ReLU()

# (4): Linear(in\_features=128, out\_features=10, bias=True)

# )

# )

def train\_model(model, traindataloader, train\_rate, criterion, optimizer, num\_epochs):

batch\_num = len(traindataloader)

train\_batch\_num = round(train\_rate \* batch\_num)

best\_model\_wts = copy.deepcopy(model.state\_dict())

best\_acc = 0.0

train\_loss\_all = []

train\_acc\_all = []

val\_loss\_all = []

val\_acc\_all = []

since = time.time()

for epoch in range(num\_epochs):

print('Epoch {}/{}'.format(epoch, num\_epochs - 1))

print('-' \* 10)

train\_loss = 0.0

train\_corrects = 0

train\_num = 0

val\_loss = 0

val\_corrects = 0

val\_num = 0

for step, (b\_x, b\_y) in enumerate(traindataloader):

if step < train\_batch\_num:

model.train()

output = model(b\_x)

pre\_lab = torch.argmax(output, 1)

loss = criterion(output, b\_y)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

train\_loss += loss.item() \* b\_x.size(0)

train\_corrects += torch.sum(pre\_lab == b\_y.data)

train\_num += b\_x.size(0)

else:

model.eval()

output = model(b\_x)

pre\_lab = torch.argmax(output, 1)

loss = criterion(output, b\_y)

val\_loss += loss.item() \* b\_x.size(0)

val\_corrects += torch.sum(pre\_lab == b\_y.data)

val\_num += b\_x.size(0)

train\_loss\_all.append(train\_loss / train\_num)

train\_acc\_all.append(train\_corrects.double().item() / train\_num)

val\_loss\_all.append(val\_loss / val\_num)

val\_acc\_all.append(val\_corrects.double().item() / val\_num)

print('{} Train Loss: {:.4f}, Train Acc: {:.4f},'.format(epoch, train\_loss\_all[-1], train\_acc\_all[-1]))

print('{} Val Loss: {:.4f} , Val Acc: {:.4f}'.format(epoch, val\_loss\_all[-1], val\_acc\_all[-1]))

if val\_acc\_all[-1] > best\_acc:

best\_acc = val\_acc\_all[-1]

best\_model\_wts = copy.deepcopy(model.state\_dict())

print('save model')

time\_use = time.time() - since

print('Train and val complete in {:.0f}m {:.0f}s'.format(time\_use // 60, time\_use % 60))

model.load\_state\_dict(best\_model\_wts)

train\_process = pd.DataFrame(data={'epoch': range(num\_epochs),

'train\_loss': train\_loss\_all, 'train\_acc': train\_acc\_all,

'val\_loss': val\_loss\_all, 'val\_acc': val\_acc\_all})

return model, train\_process

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(myconvnet.parameters(), lr=0.0003)

myconvnet, train\_process = train\_model(myconvnet, train\_loader, 0.8, criterion, optimizer, num\_epochs=25)

plt.figure(figsize=(12, 4))

# 绘制损失曲线

plt.subplot(1, 2, 1)

plt.plot(train\_process['epoch'], train\_process['train\_loss'], 'ro-', label='Train loss')

plt.plot(train\_process['epoch'], train\_process['val\_loss'], 'bs-', label='Val loss')

plt.legend()

plt.xlabel('Epoch')

plt.ylabel('Loss')

# 绘制准确率曲线

plt.subplot(1, 2, 2)

plt.plot(train\_process['epoch'], train\_process['train\_acc'], 'ro-', label='Train accuracy')

plt.plot(train\_process['epoch'], train\_process['val\_acc'], 'bs-', label='Val accuracy')

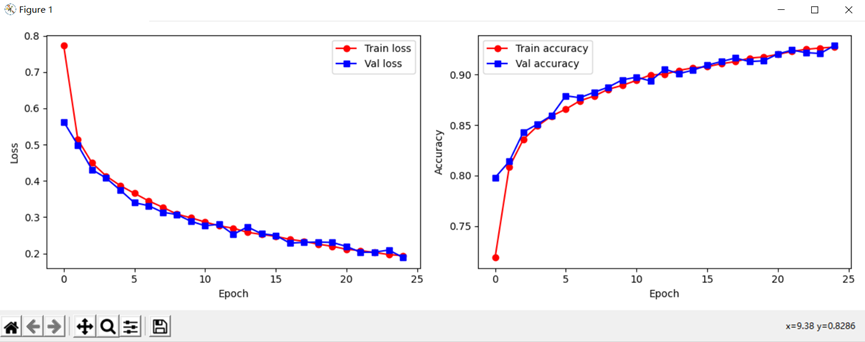
plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

# 显示图表

plt.tight\_layout() # 可选，用于调整子图布局

plt.show()

# plt.figure(figsize=(12, 4))

# plt.subplot(1, 2, 1)

# plt.plot(train\_process.epoch, train\_process.train\_loss\_all, 'ro-', label='Train loss')

# plt.plot(train\_process.epoch, train\_process.val\_loss\_all, 'bs-', label='Val loss')

# plt.legend()

# plt.xlabel('epoch')

# plt.ylabel('Loss')

# plt.subplot(1, 2, 2)

# plt.plot(train\_process.epoch, train\_process.train\_acc\_all, 'ro-', label='Train acc')

# plt.plot(train\_process.epoch, train\_process.val\_acc\_all, 'bs-', label='Val acc')

# plt.xlabel('epoch')

# plt.ylabel('acc')

# plt.legend()

# plt.show()

myconvnet.eval()

output = myconvnet(test\_data\_x)

pre\_lab = torch.argmax(output, 1)

acc = accuracy\_score(test\_data\_y, pre\_lab)

print('在测试集的精度', acc)

conf\_mat = confusion\_matrix(test\_data\_y, pre\_lab)

df\_cm = pd.DataFrame(conf\_mat, index=class\_label, columns=class\_label)

heatmap = sns.heatmap(df\_cm, annot=True, fmt='d', cmap='YlGnBu')

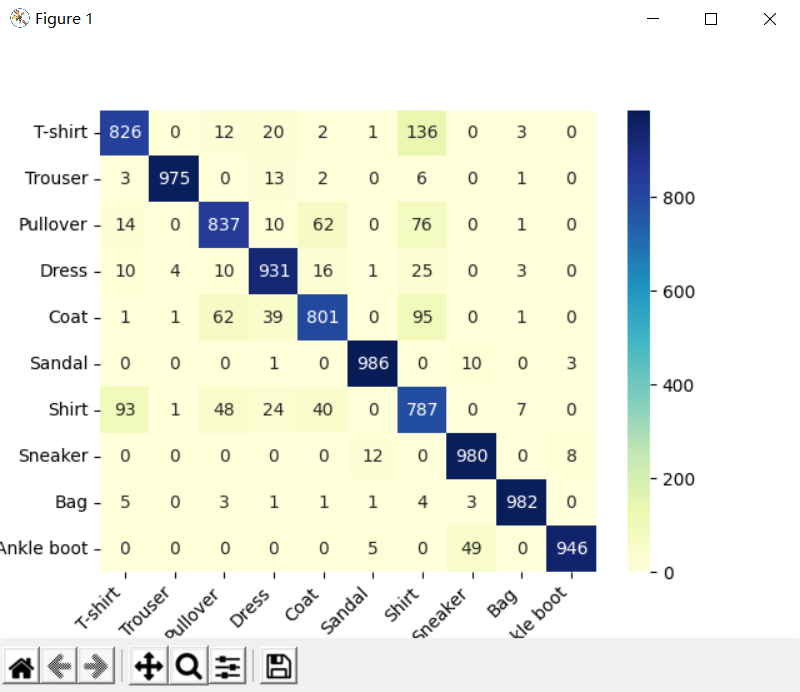
heatmap.yaxis.set\_ticklabels(heatmap.yaxis.get\_ticklabels(), rotation=0, ha='right')

heatmap.xaxis.set\_ticklabels(heatmap.xaxis.get\_ticklabels(), rotation=45, ha='right')

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.show()



# Epoch 24/24

# ----------

# 24 Train Loss: 0.1940, Train Acc: 0.9273,

# 24 Val Loss: 0.1891 , Val Acc: 0.9288

# save model

# Train and val complete in 28m 33s

# 在测试集的精度 0.9051

# Traceback (most recent call last):

# File "D:\pythoncode\learn\a\deep\_learning6.3.py", line 208, in <module>

# plt.plot(train\_process.epoch, train\_process.train\_loss\_all, 'ro-', label='train\_loss')

# File "D:\anaconda3\envs\deeplearning\lib\site-packages\pandas\core\generic.py", line 5141, in \_\_getattr\_\_

# return object.\_\_getattribute\_\_(self, name)

# AttributeError: 'DataFrame' object has no attribute 'train\_loss\_all'

# plt.plot(train\_process.epoch, train\_process.train\_loss\_all, 'ro-', label='Train loss')

# plt.plot(train\_process.epoch, train\_process.val\_loss\_all, 'bs-', label='Val loss')

# 把上面的代码换成下面的这种，不知道课本当时为什么这么写。。。

# plt.plot(train\_process['epoch'], train\_process['train\_acc'], 'ro-', label='Train accuracy')

# plt.plot(train\_process['epoch'], train\_process['val\_acc'], 'bs-', label='Val accuracy')

# 6.3.4 空洞卷积神经网络的搭建

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import time

import copy

from sklearn.metrics import accuracy\_score, confusion\_matrix

import seaborn as sns

import torch

from torch import nn

import torch.nn.functional as F

import torch.optim as optim

import torchvision

import torch.utils.data as Data

from torchvision import transforms

import hiddenlayer as hl

from torchvision.datasets import FashionMNIST

from sklearn.metrics import accuracy\_score

train\_data = FashionMNIST(

root="./deep Learning/FashionMNIST", # 数据的路径

train=True, # 只使用训练数据集

# 将数据转化为torch使用的张量,取汁范围为［0，1］

transform=transforms.ToTensor(),

download=True # 因为数据已经下载过，所以这里不再下载

)

test\_data = FashionMNIST(root="./deep Learning/FashionMNIST", train=False)

test\_data\_x = test\_data.data.type(torch.FloatTensor) / 255.0

test\_data\_x = torch.unsqueeze(test\_data\_x, dim=1)

test\_data\_y = test\_data.targets

class\_label = train\_data.classes

class\_label[0] = 'T-shirt'

train\_loader = Data.DataLoader(

dataset=train\_data, # 使用的数据集

batch\_size=64, # 批处理样本大小

shuffle=True, # 每次迭代前打乱数据

num\_workers=0, # 使用两个进程

)

class MyConvdilaNet(nn.Module):

def \_\_init\_\_(self):

super(MyConvdilaNet, self).\_\_init\_\_()

self.conv1 = nn.Sequential(

nn.Conv2d(

in\_channels=1,

out\_channels=16,

kernel\_size=3,

stride=1,

padding=1,

dilation=2

),

nn.ReLU(),

nn.AvgPool2d(

kernel\_size=2,

stride=2,

),

)

self.conv2 = nn.Sequential(

nn.Conv2d(16, 32, 3, 1, 0, dilation=2),

nn.ReLU(),

nn.AvgPool2d(2, 2)

)

self.classifier = nn.Sequential(

nn.Linear(32 \* 4 \* 4, 256),

# nn.Linear(32\*7\*7,128),

nn.ReLU(),

nn.Linear(256, 128),

nn.ReLU(),

nn.Linear(128, 10)

)

def forward(self, x):

x = self.conv1(x)

x = self.conv2(x)

x = x.view(x.size(0), -1) # 展开卷积层

output = self.classifier(x)

return output

def train\_model(model, traindataloader, train\_rate, criterion, optimizer, num\_epochs):

batch\_num = len(traindataloader)

train\_batch\_num = round(train\_rate \* batch\_num)

best\_model\_wts = copy.deepcopy(model.state\_dict())

best\_acc = 0.0

train\_loss\_all = []

train\_acc\_all = []

val\_loss\_all = []

val\_acc\_all = []

since = time.time()

for epoch in range(num\_epochs):

print('Epoch {}/{}'.format(epoch, num\_epochs - 1))

print('-' \* 10)

train\_loss = 0.0

train\_corrects = 0

train\_num = 0

val\_loss = 0

val\_corrects = 0

val\_num = 0

for step, (b\_x, b\_y) in enumerate(traindataloader):

if step < train\_batch\_num:

model.train()

output = model(b\_x)

pre\_lab = torch.argmax(output, 1)

loss = criterion(output, b\_y)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

train\_loss += loss.item() \* b\_x.size(0)

train\_corrects += torch.sum(pre\_lab == b\_y.data)

train\_num += b\_x.size(0)

else:

model.eval()

output = model(b\_x)

pre\_lab = torch.argmax(output, 1)

loss = criterion(output, b\_y)

val\_loss += loss.item() \* b\_x.size(0)

val\_corrects += torch.sum(pre\_lab == b\_y.data)

val\_num += b\_x.size(0)

train\_loss\_all.append(train\_loss / train\_num)

train\_acc\_all.append(train\_corrects.double().item() / train\_num)

val\_loss\_all.append(val\_loss / val\_num)

val\_acc\_all.append(val\_corrects.double().item() / val\_num)

print('{} Train Loss: {:.4f}, Train Acc: {:.4f},'.format(epoch, train\_loss\_all[-1], train\_acc\_all[-1]))

print('{} Val Loss: {:.4f} , Val Acc: {:.4f}'.format(epoch, val\_loss\_all[-1], val\_acc\_all[-1]))

if val\_acc\_all[-1] > best\_acc:

best\_acc = val\_acc\_all[-1]

best\_model\_wts = copy.deepcopy(model.state\_dict())

print('save model')

time\_use = time.time() - since

print('Train and val complete in {:.0f}m {:.0f}s'.format(time\_use // 60, time\_use % 60))

model.load\_state\_dict(best\_model\_wts)

train\_process = pd.DataFrame(data={'epoch': range(num\_epochs),

'train\_loss': train\_loss\_all, 'train\_acc': train\_acc\_all,

'val\_loss': val\_loss\_all, 'val\_acc': val\_acc\_all})

return model, train\_process

myconvdilanet = MyConvdilaNet()

print(myconvdilanet)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(myconvdilanet.parameters(), lr=0.0003)

myconvdilanet, train\_process = train\_model(myconvdilanet, train\_loader, 0.8, criterion, optimizer, num\_epochs=25)

plt.figure(figsize=(12, 4))

# 绘制损失曲线

plt.subplot(1, 2, 1)

plt.plot(train\_process['epoch'], train\_process['train\_loss'], 'ro-', label='Train loss')

plt.plot(train\_process['epoch'], train\_process['val\_loss'], 'bs-', label='Val loss')

plt.legend()

plt.xlabel('Epoch')

plt.ylabel('Loss')

# 绘制准确率曲线

plt.subplot(1, 2, 2)

plt.plot(train\_process['epoch'], train\_process['train\_acc'], 'ro-', label='Train accuracy')

plt.plot(train\_process['epoch'], train\_process['val\_acc'], 'bs-', label='Val accuracy')

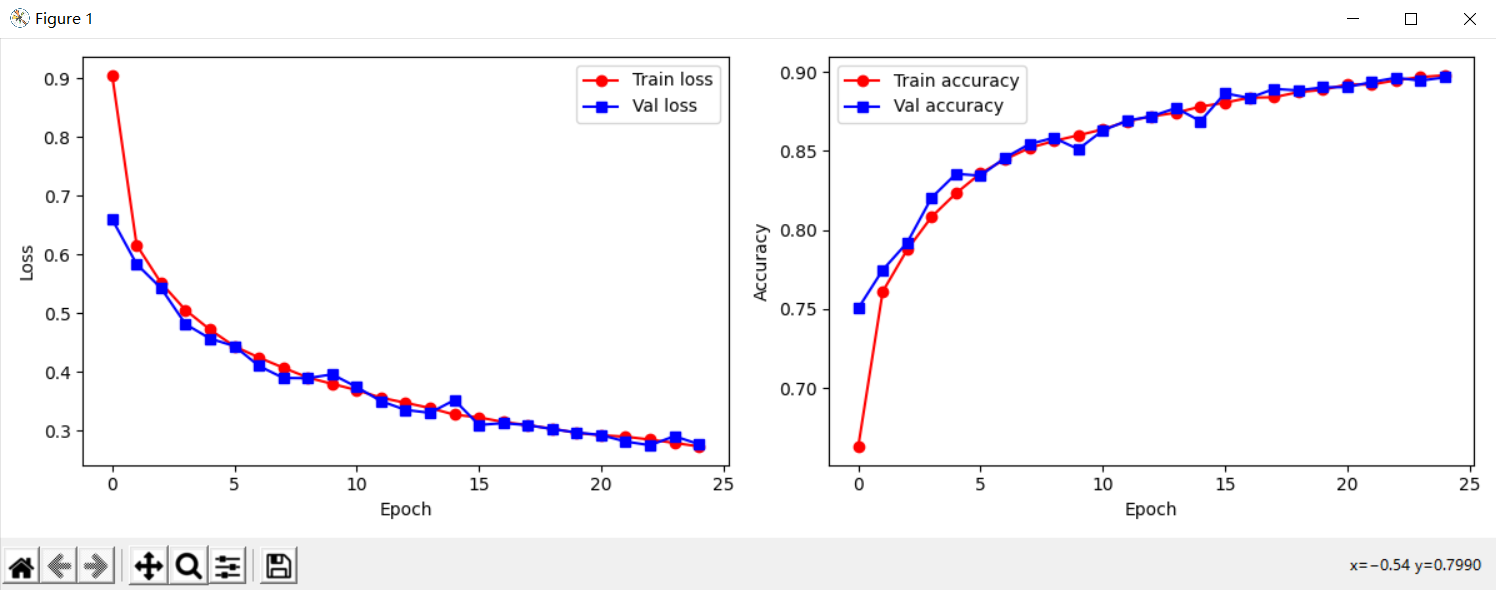
plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

# 显示图表

plt.tight\_layout() # 可选，用于调整子图布局

plt.show()

myconvdilanet.eval()

output = myconvdilanet(test\_data\_x)

pre\_lab = torch.argmax(output, 1)

acc = accuracy\_score(test\_data\_y, pre\_lab)

print('在测试集的精度', acc)

conf\_mat = confusion\_matrix(test\_data\_y, pre\_lab)

df\_cm = pd.DataFrame(conf\_mat, index=class\_label, columns=class\_label)

heatmap = sns.heatmap(df\_cm, annot=True, fmt='d', cmap='YlGnBu')

heatmap.yaxis.set\_ticklabels(heatmap.yaxis.get\_ticklabels(), rotation=0, ha='right')

heatmap.xaxis.set\_ticklabels(heatmap.xaxis.get\_ticklabels(), rotation=45, ha='right')

plt.ylabel('True label')

plt.xlabel('Predicted label')

plt.show()

# Epoch 24/24

# ----------

# 24 Train Loss: 0.2729, Train Acc: 0.8980,

# 24 Val Loss: 0.2767 , Val Acc: 0.8967

# save model

# Train and val complete in 21m 49s

# 在测试集的精度 0.8826

