## CNN搭建的整体流程

### 导库

import torch # 导入 torch

import torch.nn as nn # torch的网络组建

import torch.nn.functional as F # 搭建网络

import torch.optim as optim # 优化器

import torchvision # 数据相关

from torchvision import transforms # 数据处理与增强

### 数据处理

路径：

train\_dir = '你的训练集路径'

test\_dir = '你的测试集路径'

数据处理与增强：

1. 训练集的处理

train\_transform = transforms.Compose([

transforms.Resize((256,256)), # 输入尺寸

transforms.RandomCrop((224,224)), # 裁剪尺寸

transforms.RandomHorizontalFlip(p=0.5), # 随机水平翻转

transforms.ColorJitter(

brightness=(0.7,1.3), # 明暗程度

contrast=(0.7,1.3), # 对比度

saturation=(0.7,1.3), # 饱和度

hue=(-0.05,0.05)), # 颜色

transforms.ToTensor(), # 数据转为张量并做归一化

transforms.Normalize( # 数据标准化

mean = [.5,.5,.5], # RGB均值

std = [.5,.5,.5]) # RGB 方差 （可以使用经验值（查阅））

])

1. 测试集处理

test\_transform = transforms.Compose([

transforms.Resize((224,224)), # 输入尺寸（在预测中应遵循此尺寸）

transforms.ToTensor(), # 张量

transforms.Normalize( # 标准化

mean = [.5,.5,.5],

std = [.5,.5,.5])

])

1. 创建数据的batch打包、

train\_ds = torchvision.datasets.ImageFolder(

train\_dir, # 导入训练文件夹路径

transform=train\_transform # 数据处理与增强

)

test\_ds = torchvision.datasets.ImageFolder(

test\_dir, # 导入测试文件夹路径

transform=test\_transform # 数据处理与增强

)

# 创建dl

BATCHSIZE = 16 # batch\_size的批量尺寸

train\_dl = torch.utils.data.DataLoader(

train\_ds, # 将打包好的数据生成可迭代的batch\_size个数的批量数据

batch\_size=BATCHSIZE, # batch\_size 值得选择应对照本机配置

shuffle=True # 打乱（必须打乱否则模型会识别顺序）

)

test\_dl = torch.utils.data.DataLoader(

test\_ds,

batch\_size=BATCHSIZE

)

### 模型搭建

1. 自建模型：

class TestNetBN(nn.Module):

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

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

self.conv1 = nn.Conv2d(3,16,3) # 卷积

self.bn1 = nn.BatchNorm2d(16) # 批归一化

self.conv2 = nn.Conv2d(16,32,3)

self.bn2 = nn.BatchNorm2d(32)

self.conv3 = nn.Conv2d(32,64,3)

self.bn3 = nn.BatchNorm2d(64)

self.fc1 = nn.Linear(64\*10\*10,1024) # 全连接

self.fc2 = nn.Linear(1024,2) # 输出全连接

def forward(self,x):

x = F.relu(self.conv1(x)) # relu 激活函数

x = self.bn1(x) # 连接批归一化

x = F.max\_pool2d(x,2) # 最大池化

x = F.relu(self.conv2(x))

x = self.bn2(x)

x = F.max\_pool2d(x,2)

x = F.relu(self.conv3(x))

x = self.bn3(x)

x = F.max\_pool2d(x,2)

x = x.view(-1,64\*10\*10)

x = F.dropout(x) # dropout 正则惩罚

x = F.relu(self.fc1(x))

x = F.dropout(x)

x = self.fc2(x) # softmax

return x

1. 迁移学习：

参数：pretrained=True 表示下载 迁移模型的权重参数（imamgenet预训练参数）

model = torchvision.models.vgg16(pretrained=True)

冻结卷积基：（注意迁移模型的结构）

for param in model.features.parameters():

param.requires\_grad = False

重新定义输出结构（可训参数）：

model.classifier[-1].out\_features = 2 # 此处为2分类

### 损失函数与优化器

1. 损失函数（交叉熵（可选其他损失函数建议查阅官方API并理解其含义））

loss\_fn = nn.CrossEntropyLoss()

1. 优化器

Lr参数为初始的学习率值

optimizer = torch.optim.Adam(model.parameters(),lr=0.0001)

1. 学习率的衰减策略

optimizer：参数为待优化学习率的优化器

step\_size：优化的步伐

gamma：衰减系数 一般为 lr \* gamma（其他策略另算（余弦退火））

torch.optim.lr\_scheduler.StepLR(optimizer,step\_size=10,gamma=0.1)

### 模型训练

1. 训练模块的基本步骤及返回值

def train(dataloader,model,loss\_fn,optimizer):

# acc 当前样本一共有多少个

size = len(dataloader.dataset)

num\_batches = len(dataloader)

# 初始化loss值

train\_loss,correct = 0,0

model.train() # 模式为训练模式

for X,y in dataloader:

# 前向传播（推理）

pred = model(X)

# 产生误差

loss = loss\_fn(pred,y) # 返回平均误差

# loss = torch.nn.functional.cross\_entropy(pred, y)

# 归零权重(固定写法)

optimizer.zero\_grad()

# 反向传递

loss.backward()

# 利用梯度下降优化参数

optimizer.step()

# # 放弃张量微分的追踪（小验证）

with torch.no\_grad():

train\_loss += loss.item()

correct += (pred.argmax(1) == y).type(torch.float).sum().item()

1. 测试模块的基本步骤及返回值

# 搭建测试 （test）

def test(dataloader,model):

# acc 当前样本一共有多少个

size = len(dataloader.dataset)

num\_batches = len(dataloader)

test\_loss,correct = 0,0

model.eval() # 推理模式

with torch.no\_grad():

for X,y in dataloader:

pred = model(X)

test\_loss += loss\_fn(pred,y)

correct += (pred.argmax(1) == y).type(torch.float).sum().item()

correct /= size

test\_loss /= num\_batches

return test\_loss,correct

1. 训练

# 定义训练函数

def fit(epochs,train\_dl,test\_dl,model,loss\_fn,optimizer,

exp\_lr\_scheduler=None):

train\_acc = []

train\_loss = []

test\_acc = []

test\_loss = []

for epoch in range(epochs):

start = time.time()

epoch\_loss,epoch\_acc = train(dataloader=train\_dl,model=model,loss\_fn=loss\_fn,optimizer=optimizer)

epoch\_test\_loss,epoch\_test\_acc = test(dataloader=test\_dl,model=model)

end = time.time()

times = end - start

train\_acc.append(epoch\_acc)

train\_loss.append(epoch\_loss)

test\_acc.append(epoch\_test\_acc)

test\_loss.append(epoch\_test\_loss)

if exp\_lr\_scheduler:

exp\_lr\_scheduler.step()

print('训练epoch{},训练集损失值:{:.2f},训练集的准确率:{:.2f}%,测试集损失值:{:.2f},测试集的准确率:{:.2f}%,消耗时间：{:.2f}s'.

format(epoch+1,epoch\_loss,epoch\_acc\*100,epoch\_test\_loss,epoch\_test\_acc\*100,times))

return train\_loss,test\_loss,train\_acc,test\_acc

1. 模型保存

权重保存：此保存方法只保留权重并注意参数有可能存在GPU中推理前应注意数据与模型保持同一个设备。

torch.save(model.state\_dict(),'cnnNet\_weights.pth')

检查点保存：此保存方法训练较大规模深度学习时的必备

PATH = 'cnnNet\_checkpoint.pt'

torch.save({

'poch':epoch,

'model\_state\_dict':model.state\_dict(),

'optimizer\_state\_dict':optimizer.state\_dict()

})

最优模型保存：

best\_model\_wts = copy.deepcopy(model.state\_dict()) # 用以记录模型最优参数

best\_acc = 0.0

if epoch\_test\_acc > best\_acc: # 如果当前的test\_acc 高于best\_acc 执行保存

best\_acc = epoch\_test\_acc

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

1. 推理

第一步：模型导入

第二步：新数据处理（注意应和模型输入的图像维度一样（b,c,h,w））

第三步：推理后应处理softmax的值，取出最大得分的位置，结合label给出推理答案。

### 附录（评估指标可视化）

## 训练曲线可视化

# 损失值

train\_loss\_min\_index = np.argmin(np.array(train\_loss))

train\_loss\_min\_value = round(train\_loss[train\_loss\_min\_index],6)

test\_loss\_min\_index = np.argmin(np.array(test\_loss))

test\_loss\_min\_value = round(test\_loss[test\_loss\_min\_index].item(),6)

s = train\_loss\_min\_value

s\_test = test\_loss\_min\_value

plt.plot(range(1,epoch+2),train\_loss,label='train\_loss')

plt.plot(range(1,epoch+2),test\_loss,label='test\_loss')

plt.text(train\_loss\_min\_index, train\_loss\_min\_value+0.1, s)

plt.text(test\_loss\_min\_index, test\_loss\_min\_value+0.1, s\_test)

plt.plot(train\_loss\_min\_index+1,train\_loss\_min\_value,'bo')

plt.plot(test\_loss\_min\_index+1,test\_loss\_min\_value,'yo')

plt.title('CNN(Dropout) Loss values')

plt.legend()

## 训练曲线可视化

# 准确率

train\_acc\_max\_index = np.argmax(np.array(train\_acc))

train\_acc\_max\_value = round(train\_acc[train\_acc\_max\_index],6)

test\_acc\_max\_index = np.argmax(np.array(test\_acc))

test\_acc\_max\_value = round(test\_acc[test\_acc\_max\_index],6)

s = train\_acc\_max\_value

s\_test = test\_acc\_max\_value

plt.plot(range(1,epoch+2),train\_acc,label='train\_loss')

plt.plot(range(1,epoch+2),test\_acc,label='test\_loss')

plt.text(train\_acc\_max\_index, train\_acc\_max\_value-0.02, s)

plt.text(test\_acc\_max\_index, test\_acc\_max\_value+0.01, s\_test)

plt.plot(train\_acc\_max\_index+1,train\_acc\_max\_value,'bo')

plt.plot(test\_acc\_max\_index+1,test\_acc\_max\_value,'go')

plt.title('CNN(Dropout) Acc values')

plt.ylim(0.5,1.08)

plt.xlim(0,31)

plt.legend()