卷积神经网络结构与应用介绍

摘要

卷积神经网络(CNN)是一类包含卷积计算切具有深度结构的神经网络，是深度学习中的经典算法。本报告介绍了 CNN 的基本结构、CNN 网络的四种经典架构、CNN 网络的两种应用，并在最后给出使用 PyTorch 基于 ResNet101 迁移学习的情绪图片识别案例。

1 CNN 网络基本结构

与传统计算机视觉技术的人工特征提取不同，在卷积神经网络中，卷积运算是为了让算法自学习地提取矩阵（图片）的特征信息，以自适应地完成特征工程。在卷积层与池化层对图片的特征信息进行提取后，再由全连接层通过特征进行分类/回归，这便是卷积神经网络的整体结构。

1.1 卷积层（Convolutional Layer）

1.1.1 卷积运算公式

卷积运算的定义为：设:$f(x), g(x)$是$R1$上的两个可积函数，作积分

$$\int_{-\infty}^{+\infty} f(t) \cdot g(x-t)dt \tag{1.1}$$

可以证明，关于几乎所有的实数x，上述积分是存在的。这样，随着x的不同取值，这个积分就定义了一个新函数$h(x)$，称为函数f与g的卷积。

在卷积神经网络的应用中，通常使用该公式的离散形式，即为：

$$Z^{l+1} = \sum_{k=1}^{K_l} \sum_{i=1}^{L} \sum_{j=1}^{L} \left(Z_{i,j,k}^{l} w_k^{l+1} \right) \tag{1.2}$$

其以矩阵形式进行表示即如下图所示：

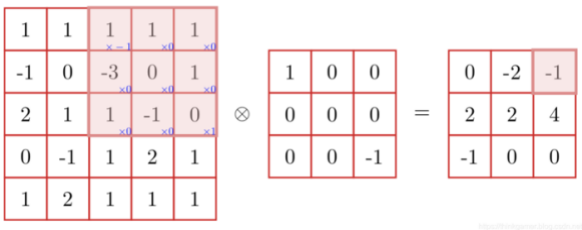

图 1

1.1.2 卷积层超参数

在卷积层中，除了卷积核的数值为算法通过优化算法自学习外，其中的卷积核大小、通道数（Channel）、步长（Stride）、填充（Padding）都是需要调节的超参数。

卷积核大小与卷积层的感受野（Receptive Field）相关，卷积核大小越大则单层内的感受野越大，同时也可以通过加深网络层数来增大感受野。卷积核大小通常设置为 3*3 或 5*5.

由通道数的不同，可分为单通道卷积与多通道卷积两种。单通道卷积即在二维平面中进行一次卷积运算得到输出。而多通道卷积则是在每个通道层面上进行卷积运算，并最终将多通道求和进行输出。由此不难发现，以 1*1 为卷积核的 n 通道卷积其实与 n 个神经元的全连接层等价。多通道卷积示意图如下：

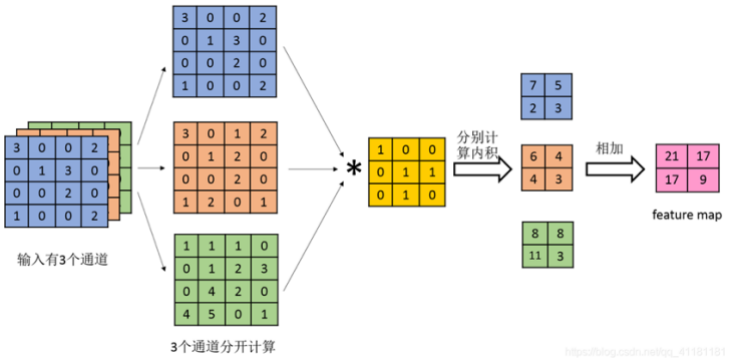

图 2

步长（Stride）为卷积核窗口在输入矩阵上每次移动的元素个数，调整步长值的大小可影响输出矩阵的形状。

填充（Padding）为在输入矩阵四周进行填充一些元素，超参数中可以设置填充的元素层数与填充元素的数值大小。例填充 1 层、元素为 0 的矩阵如下图所示：

0	0	0	0	0	0	0	0
0	0	0	1	2	0	0	0
0	0	1	1	3	0	0	0
0	1	2	2	3	1	1	0
0	1	2	1	3	1	1	0
0	1	1	1	3	1	1	0
0	0	0	0	2	0	0	0
0	0	0	0	0	0	0	0

图 3

卷积层输入层与输出层的形状计算公式为：

$$n_{out} = \frac{n_{in} + 2p - k}{s} + 1 \tag{1.3}$$

其中n_{in}为输入层形状，n_{out}为输出层形状，k为卷积核形状，p为填充形状，s为步长大小。

1.2 池化层（Pooling Layer）

在卷积层进行特征提取后，输出的特征图会被传递至池化层进行特征选择和信息过滤。由于池化操作不可导，因此池化层不参与反向传播。池化操作后的结果相比其输入缩小了。池化层的引入是仿照人的视觉系统对视觉输入对象进行降维和抽象。

1.2.1 池化层种类

最大池化（max-pooling）为在池化范围内，求整个矩阵区间内的元素最大值，作为当前池化范围内的输出值，如下图所示。

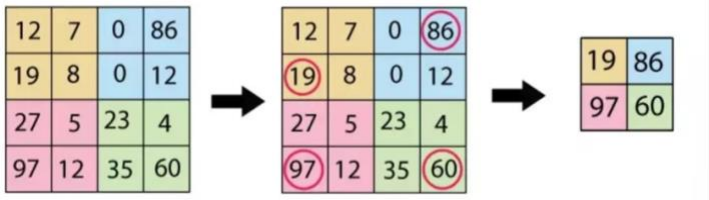

图 4

平均池化（mean-pooling）为在池化范围内，对整个池化区间内的元素求和进而求平均值，作为当前池化范围内的输出值，如下图所示。

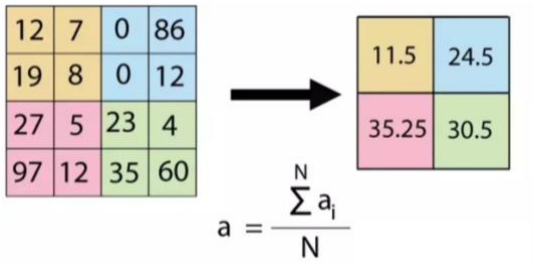

图 5

除此之外，常见的池化层还有全局平均池化、全局最大池化、重叠池化（Overlapping Pooling）、空金字塔池化（Spatial Pyramid Pooling）等。

1.2.2 池化层参数

池化层参数同样有池化窗口的大小、池化的填充以及池化的步长。

1.3 全连接层（Fully-Connected Layer）

全连接层即为线性层与多层感知机（MLP）等价，在卷积神经网络中通常用来在卷积层与池化层进行特征提取后，使用提取到的特征对图片进行分类或回归。

1.3.1 拉平操作（Flatten）

在卷积层与池化层的计算后，矩阵仍为二维形式，故而需要通过拉平（Flatten）操作将其化为一维向量。而后才可以作为全连接层的输入。

1.3.2 激活函数（Activation Function）

激活函数对于人工神经网络模型去学习、理解非常复杂和非线性的函数来说具有十分重要的作用。它们将非线性特性引入到我们的网络中。

Sigmoid 函数：

$$f(x) = \frac{1}{1 + e^{-x}} \tag{1.4}$$

函数图像：

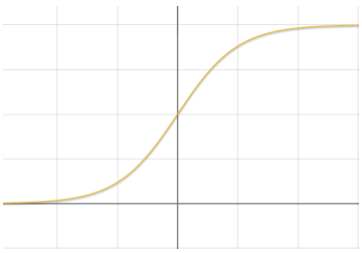

图 6

Tanh 函数：

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{1.5}$$

函数图像：

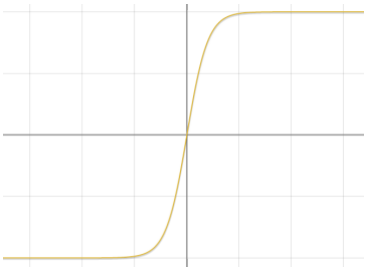

图 7

1.4 整体结构

在卷积神经网络中，卷积层与池化层负责特征工程中的特征提取与融合，而后由全连接层来进行分类或回归。如下图所示：

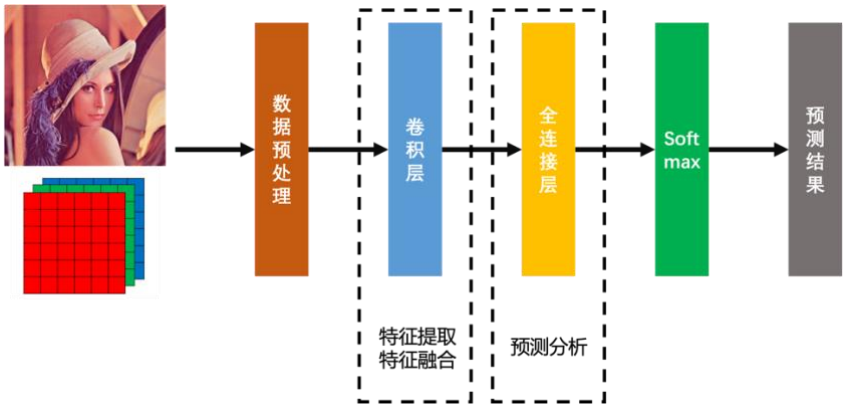

图 8

2 CNN 的经典架构

本节介绍 CNN 网络过程中的几个经典网络架构，借鉴李沐老师在 B 站课程【动手学深度学习 v2】，并查阅网络资料。

2.1 LeNet 网络

LeNet 是卷积神经网络的开山之作，也是将深度学习推向繁荣的一座里程碑。

Yann LeCun 于上世纪 90 年代提出了 LeNet，他首次采用了卷积层、池化层这两个全新的神经网络组件。LeNet 在手写字符识别任务上取得了瞩目的准确率。

网络架构图如下：

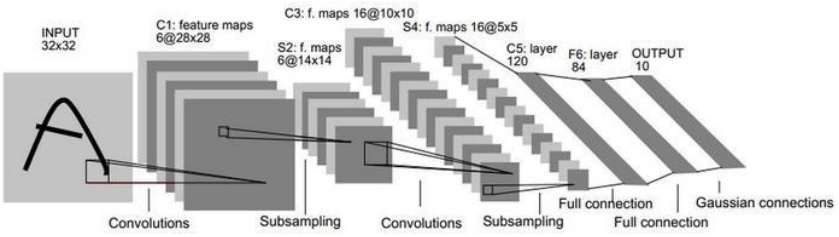

图 9

2.2 AlexNet 网络

AlexNet 由 Geoffrey 和他的学生 Alex 提出，并在 2012 年的 ILSVRC 竞赛中获得了第一名。Alexnet 共有 8 层结构，前 5 层为卷积层，后三层为全连接层。

AlexNet 网络结构具有如下特点：

1）AlexNet 在激活函数上选取了非线性非饱和的 ReLU 函数，在训练阶段梯度衰减快慢方面，ReLU 函数比传统神经网络所选取的非线性饱和函数（如 sigmoid 函数，tanh 函数）要快许多。

2）AlexNet 在双 GPU 上运行，每个 GPU 负责一半网络的运算。

3）采用局部响应归一化（LRN）。对于非饱和函数 ReLU 来说，不需要对其输入进行标准化，但 Alex 等人发现，在 ReLU 层加入 LRN，可形成某种形式的横向抑制，从而提高网络的泛华能力。

4）池化方式采用 overlapping pooling。即池化窗口的大小大于步长，使得每次池化都有重叠的部分。

网络架构图如下：

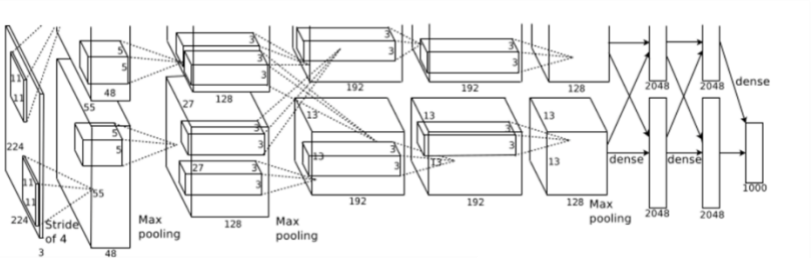

图 10

2.3 VGG 网络

VGG 是 Oxford 的 Visual Geometry Group 的组提出的。该网络是在 ILSVRC 2014 上的相关工作，主要工作是证明了增加网络的深度能够在一定程度上影响网络最终的性能。

VGG 网络的最大突破是提出了“VGG 块”概念，块（Block）结构十分整洁，从此成为了后面网络模型设计中的常用范式。

VGG 网络结构图：

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

图 11

2.4 ResNet 网络

残差神经网络(ResNet)是由微软研究院的何恺明、张祥雨、任少卿、孙剑等人提出的。ResNet 在 2015 年的 ILSVRC（ImageNet Large Scale Visual Recognition Challenge）中取得了冠军。

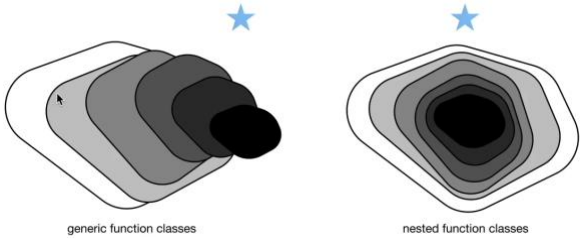

图 12

ResNet 为了使网络搭建地更深，引入了残差连接（skip connect）。当我们强行将一个输入添加到函数的输出的时候，虽然我们仍然可以用$G(x)$来描述输入输出的关系，但是这个$G(x)$却可以明确的拆分为$F(x)$和X的线性叠加。残差连接示意图如下：

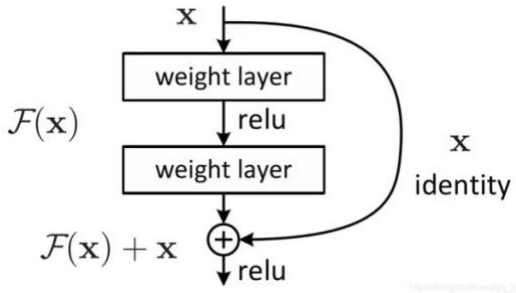

图 13

3 CNN 网络的应用

3.1 图像分类与识别

卷积神经网络在图像分类上一枝独秀，其中手写字体(Hand Written)的识别率已经超越人类的识别率，达到了 99.9%。国外众多快递公司已经开始应用卷积神经网络模型识别快递单上的手写字体，尽最大可能地节约企业成本、提高自身的系统运作效率 。

除了图像分类,还可以在一张图片中识别不止一个类别,这便是目标检测。

图 14

3.2 自然语言处理(NLP)

卷积神经网络不再是图像处理任务专用的神经网络模型。自然语言处理任务在卷积神经网络模型中的输入不再是像素点，大多数情况下是以矩阵表示的句子。矩阵的每一行对应一个元素，如果一个元素代表一个单词，那么每一行代表一个单词的向量。卷积神经网络模型应用在计算机视觉中，卷积核每次只对图像中的一小块区域进行卷积操作，但在处理自然语言时，卷积核通常覆盖上下几行(几个单词)。因此，卷积核的宽度和输入矩阵的宽度需要相同。

4 迁移学习与实现

迁移学习理念认为对于一个已经在庞大数据集（例如 ImageNet）上训练的模型而言，对其进行一定程度的微调，并在目标域训练集的少量样本中进行训练，便可以得到作用于目标域的新模型，这种技术对于少量样本的数据集建模提供了便利。

本节介绍编者使用 resnet101 预训练模型并通过微调变更为推特图片情绪识别模型（数据集：T4SA-Twitter1269）。模型的思路路线如下图所示：

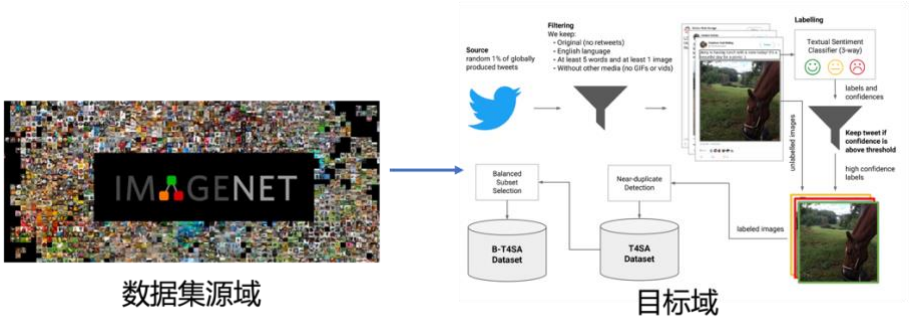

图 15

最终在测试集上得到的损失函数数值虽训练步数走向如下:

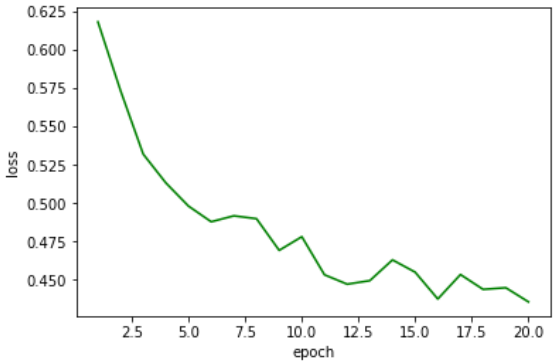

图 16

项目代码如下（代码是大创项目的一部分，源文件为 jupyter notebook 格式，时间有些长且疏于整理，因此比较乱，请老师见谅）:

```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch import optim
import torchvision
from torchvision import models
from torchvision import datasets
from torchvision import transforms
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torchvision.utils import make_grid
import visdom
from torch.utils.tensorboard import SummaryWriter
import copy
import time
```

```
from PIL import ImageFile
from tqdm import tqdm

BATCH_SIZE = 32
ImageFile.LOAD_TRUNCATED_IMAGES = True
model_pre = models.resnet101(pretrained=True)
data_transforms = {
    'train':
    transforms.Compose([
        transforms.RandomResizedCrop(size=300, scale=(0.8, 1.1
)),
        transforms.ColorJitter(0.4, 0.4, 0.4),
        transforms.RandomCrop((256, 256)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),

    'test':
    transforms.Compose([
        transforms.Resize((300, 300)),
        transforms.RandomCrop((256, 256)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
    ]),
}

root = r"/root/autodl-tmp/DeepSent/Twitter1269/"

# 自定义图片图片读取方式，可以自行增加 resize、数据增强等操作
def MyLoader(path):
    return Image.open(path).convert('RGB')

class MyDataset (Dataset):
    # 构造函数设置默认参数
    def __init__(self, txt, transform=None, target_transform=N
one,
                 loader=MyLoader, state=True):
        with open(txt, 'r') as fh:
            imgs = []
            if state == True:
                for line in fh:
```

```
                    line = line.strip('\n')  # 移除字符串首尾的
换行符
                    line = line.rstrip()   # 删除末尾空
                    words = line.split()  # 以空格为分隔符 将字
符串分成
                    words[0] = root + 'twitter1/' + words[0]
                    imgs.append((words[0], int(words[1])))  #
imgs 中包含有图像路径和标签
            else:
                for line in fh:
                    line = line.strip('\n')  # 移除字符串首尾的
换行符
                    line = line.rstrip()   # 删除末尾空
                    words = line.split()  # 以空格为分隔符 将字
符串分成
                    words[0] = '/root/autodl-
tmp/wsj_pic/' + words[0]
                    imgs.append((words[0], words[1]))  # imgs
中包含有图像路径和标签
        self.imgs = imgs
        self.transform = transform
        self.target_transform = target_transform
        self.loader = loader

    def __getitem__(self, index):
        fn, label = self.imgs[index]
        # 调用定义的 loader 方法
        img = self.loader(fn)
        if self.transform is not None:
            img = self.transform(img)
        return img, label

    def __len__(self):
        return len(self.imgs)


train_data = MyDataset(txt=root + 'train_1.txt', transform=
                       data_transforms['train'])
test_data = MyDataset(txt=root + 'test_1.txt', transform=
                      data_transforms['test'])
pes_data = MyDataset(txt='/root/autodl-
tmp/wsj_pic/val_set.txt', transform=
                      data_transforms['test'], state=False)
```

```
# train_data 和 test_data 包含多有的训练与测试数据，调用 DataLoader 批量加载
train_loader = DataLoader(dataset=train_data, batch_size=BATCH_SIZE,
                          shuffle=True, num_workers=0)
test_loader = DataLoader(dataset=test_data, batch_size=BATCH_SIZE, num_workers=0)
pes_loader = DataLoader(dataset=pes_data, batch_size=BATCH_SIZE, num_workers=0)

# 迁移学习
def get_model():
    # 获取欲训练模型 restnet50
    model = models.resnet50(pretrained=True)
    # 冻结模型参数
    for param in model.parameters():
        param.requires_grad = False
    # 全连接层
    model.fc = nn.Sequential(
        nn.Flatten(), # 拉平
        nn.BatchNorm1d(2048), # 加速神经网络的收敛过程，提高训练过程中的稳定性
        nn.Dropout(0.5), # 丢掉部分神经元
        nn.Linear(2048, 512), # 全连接层
        nn.ReLU(), # 激活函数
        nn.BatchNorm1d(512),
        nn.Dropout(0.5),
        nn.Linear(512, 2), # 2 个输出
        nn.LogSoftmax(dim=1) # 损失函数：将 input 转换成概率分布的形式，输出 2 个概率
    )
    # model.fc = nn.Sequential(nn.Linear(2048, 3), nn.LogSoftmax(dim=1))
    # nn.init.xavier_uniform(model.fc.weight);

    return model

def train(model, device, loader, criterion, optimizer, epoch, writer):
    model.train()
    total_loss = 0.0
    correct = 0.0
    for batch_id, (data, target) in enumerate(tqdm(loader)):
        data, target = data.to(device), target.to(device)
```

```
            optimizer.zero_grad()
            # output, _ = model(data)
            output = model(data)
            # output = torch.log(torch.softmax(output, dim=-
1)) # **
            loss = criterion(output, target)  # 计算损失
            loss.backward()  # 反向传播
            optimizer.step()  # 更新参数
            total_loss += loss
            time.sleep(0.003)
            #print(batch_id / len(loader))
        writer.add_scalar("Train Loss", total_loss / len(loader),
epoch)
        writer.flush()  # 刷新日志

        model.eval()
        with torch.no_grad():
            for data, target in tqdm(loader):
                data, target = data.to(device), target.to(device)
                output = model(data)
                _, preds = torch.max(output, dim=1)
                correct += preds.eq(target.view_as(preds)).sum().i
tem()
        acc = correct / BATCH_SIZE / len(loader) * 100
        return total_loss / len(loader), acc

    def test(model, device, loader, criterion, epoch, writer):
        model.eval()
        total_loss = 0.0
        correct = 0.0
        with torch.no_grad():
            for data, target in tqdm(loader):
                data, target = data.to(device), target.to(device)
                # 预测输出
                output = model(data)
                # 计算损失
                total_loss += criterion(output, target).item()
                _, preds = torch.max(output, dim=1)
                # preds = torch.log(preds)
                # print("preds:" + str(preds) + " " + str(target))
                correct += preds.eq(target.view_as(preds)).sum().i
tem()
            total_loss /= len(loader)
            # print("correct:" + str(correct.item()))
```

```
        accuracy = 100 * correct / len(loader) / BATCH_SIZE
        writer.add_scalar("Test Loss", total_loss, epoch)
        writer.add_scalar("Accuracy", accuracy, epoch)
        writer.flush()
        print("Test Loss : {:.4f}, Accuracy: {:.4f}" .format(t
otal_loss,
                                                              a
ccuracy))
        return total_loss, accuracy

def train_epochs(model, device, tra_loader, tes_loader, criter
ion, optimizer,
                 num_epochs, writer):
    print("{0:>20} | {1:>20} | {2:>20} | {3:>20} | {4:>20} |".
format('Epoch', 'Training Loss',
                                                'Test Loss',
'Train_ACC', 'Test_ACC'))
    best_score = np.inf
    start = time.time()
    # 开始循环读取数据并验证
    for epoch in num_epochs:
        train_loss, train_acc = train(model, device, tra_loade
r, criterion, optimizer,
                              epoch, writer)
        test_loss, accuracy = test(model, device, tes_loader,
criterion, epoch,
                                   writer)
        if test_loss < best_score:
            best_score = test_loss
            torch.save(model.state_dict(), model_path)
        print("{0:>20} | {1:>20} | {2:>20} | {3:>20.2f} | {4:>
20.2f} |"
              .format(epoch, train_loss, test_loss,train_acc ,
 accuracy))
        test_arr.append(accuracy)
        loss_arr.append(test_loss)
        writer.flush()
        # 训练玩所耗费的总时间
        time_all = time.time() - start
        print("Training complete in {:.2f}m {:.2f}s"
              .format(time_all // 60, time_all % 60))
        time.sleep(0.003)

def tb_writer():
```

```
    timestr = time.strftime("%Y%m%d_%H%M%S")
    writer = SummaryWriter('logdir/' + timestr)
    return writer

device = torch.device("cuda:0")
model_path = 'model/model_resnet50_.pth'
model = get_model().to(device)

writer = tb_writer()
criterion = nn.NLLLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001)
train_epochs(model, device, train_loader, test_loader, criteri
on, optimizer,
             range(0, 20), writer)

x = np.arange(1, 21)
plt.xlabel("epoch")
plt.ylabel("loss")
plt.plot(x, loss_arr, color='green')
           plt.show()
```