SENet_CIFAR10_classification

刘天润 SC24219058

Table of contents

数据读取和处理

加载CIFAR-10数据集

CIFAR-10 是由 Hinton 的学生 Alex Krizhevsky 和 Ilya Sutskever 整理的一个用于识别普适物体的小型数据集。一共包含 10 个类别的 RGB 彩色图 片：飞机（ arplane ）、汽车（ automobile ）、鸟类（ bird ）、猫（ cat ）、鹿（ deer ）、狗（ dog ）、蛙类（ frog ）、马（ horse ）、船（ ship ）和卡车（ truck ）。图片的尺寸为 32×32 ，数据集中一共有 50000 张训练圆片和 10000 张测试图片。

使用 `torchvision` 加载和归一化训练数据和测试数据:

a、 `torchvision` 实现了常用的一些深度学习的相关的图像数据的加载功能，比如cifar10、Imagenet、Mnist等等的，保存在 `torchvision.datasets` 模块中。

b、同时，也封装了一些处理数据的方法。保存在 `torchvision.transforms` 模块中

c、还封装了一些模型和工具封装在相应模型中,比如 `torchvision.models` 当中就包含了AlexNet，VGG，ResNet，SqueezeNet等模型。

In [2]:
```
import os
os.environ["CUDA_VISIBLE_DEVICES"] = "0" # 在指定GPU运行，比如第一个GPU就是0，如果多

import torch
import torch.nn as nn
import torchvision
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn.functional as F
from torchsummary import summary
# 1. Configure memory allocator (BEFORE any CUDA operations)
os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "max_split_size_mb:32"

# 2. Clear cache
torch.cuda.empty_cache()

# 3. Optional: Limit GPU memory usage
torch.cuda.set_per_process_memory_fraction(0.8)  # Use 80% of GPU memory
```

```
d:\anaconda\envs\cuda102\lib\site-packages\tqdm\auto.py:21: TqdmWarning: IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

由于torchvision的datasets的输出是[0,1]的PILImage，所以先归一化为[-1,1]的Tensor

首先定义了一个变换transform，利用的是上面提到的transforms模块中的Compose( )把多个变换组合在一起，可以看到这里面组合了ToTensor和Normalize这两个变换

`transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))` 前面的（0.5，0.5，0.5） 是 R G B 三个通道上的均值， 后面(0.5, 0.5, 0.5)是三个通道的标准差，注意通道顺序是 R G B ，用过opencv的同学应该知道openCV读出来的图像是 BRG顺序。这两个tuple数据是用来对RGB 图像做归一化的，如其名称 Normalize 所示这里都取0.5只是一个近似的操作，实际上其均值和方差并不是这么多，但是就这个示例而言 影响可不计。精确值是通过分别计算R,G,B三个通道的数据算出来的。

In [2]:
```
transform = transforms.Compose([
#     transforms.CenterCrop(224),
    # transforms.RandomCrop(32,padding=4), # 数据增广
    # transforms.RandomHorizontalFlip(),  # 数据增广
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])
```

通过 `trainloader` 把数据传入网络，这里的 `trainloader` 是个变量名，可以随便取，重点是后面的 `torch.utils.data.DataLoader()` 定义，其来源于 `torch.utils.data` 模块

In [ ]:
```
Batch_Size = 128
trainset = datasets.CIFAR10(root='.', train=True,download=False, transform=trans
testset = datasets.CIFAR10(root='.',train=False,download=False,transform=transfo
trainloader = torch.utils.data.DataLoader(trainset, batch_size=Batch_Size,shuffl
testloader = torch.utils.data.DataLoader(testset, batch_size=Batch_Size,shuffle=
classes = ('plane', 'car', 'bird', 'cat','deer', 'dog', 'frog', 'horse', 'ship',
```

查看数据（格式，大小，形状）

In [ ]:
```
classes = trainset.classes
classes
```

Out[ ]:
```
['airplane',
 'automobile',
 'bird',
 'cat',
 'deer',
 'dog',
 'frog',
 'horse',
 'ship',
 'truck']
```

In [6]:
```
trainset.class_to_idx
```

Out[6]:
```
{'airplane': 0,
 'automobile': 1,
 'bird': 2,
 'cat': 3,
 'deer': 4,
 'dog': 5,
 'frog': 6,
 'horse': 7,
 'ship': 8,
 'truck': 9}
```

In [7]:
```
trainset.data.shape #50000是图片数量，32x32是图片大小，3是通道数量RGB
```

Out[7]:
```
(50000, 32, 32, 3)
```

In [8]:
```
#查看数据类型
print(type(trainset.data))
print(type(trainset))
```

```
<class 'numpy.ndarray'>
<class 'torchvision.datasets.cifar.CIFAR10'>
```

`trainset.data`是标准的numpy.ndarray类型，其中50000是图片数量，32x32是图片大小，3是通道数量RGB；

查看图片

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
# plt.imshow(trainset.data[0])
#从trainloader获取下一批数据
im,label = next(iter(trainloader))
```

np.ndarray转为torch.Tensor

在深度学习中，原始图像需要转换为深度学习框架自定义的数据格式，在pytorch中，需要转为`torch.Tensor`。 pytorch提供了`torch.Tensor`与`numpy.ndarray`转换为接

口:

方法名	作用
`torch.from_numpy(xxx)`	`numpy.ndarray` 转为torch.Tensor
`tensor1.numpy()`	获取tensor1对象的numpy格式数据

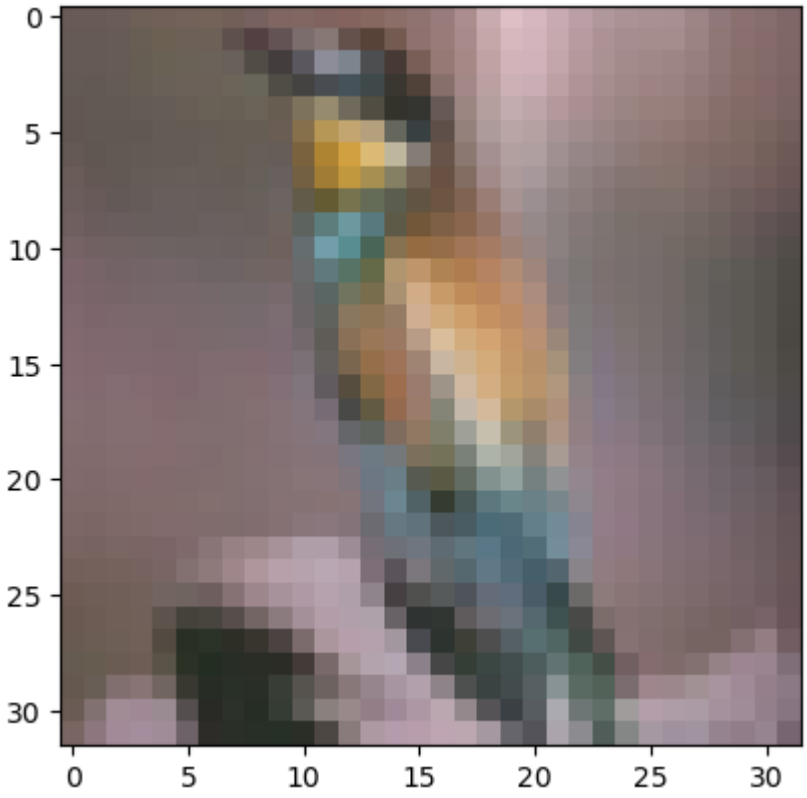
`torch.Tensor` 高维矩阵的表示： N x C x H x W

`numpy.ndarray` 高维矩阵的表示：N x H x W x C

因此在两者转换的时候需要使用 `numpy.transpose( )` 方法 。

In [ ]:
```
def imshow(img):
    img = img / 2 + 0.5
    img = np.transpose(img.numpy(),(1,2,0))
    plt.imshow(img)
```

In [11]:
```
imshow(im[0])
```



In [12]:
```
im[0].shape
```

Out[12]:
```
torch.Size([3, 32, 32])
```

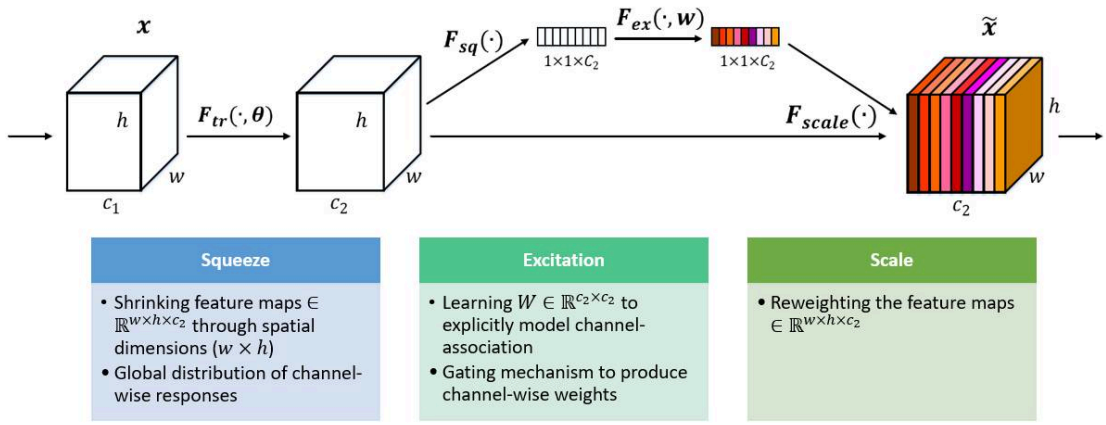
In [ ]:
```
plt.figure(figsize=(8,12))
imshow(torchvision.utils.make_grid(im[:32]))
```

定义网络（SENet）

Squeeze-and-Excitation Networks（SENet）是由自动驾驶公司Momenta在2017年公布的一种全新的图像识别结构，它通过对特征通道间的相关性进行建模，把重要的特征进行强化来提升准确率。SENet 以极大的优势获得了最后一届 ImageNet 2017 竞赛 Image Classification 任务的冠军，top5的错误率达到了2.251%，比2016年的第一名还要低25%，可谓提升巨大。下面就来具体学习一下SENet。

SE Block

SE Block是SENet的Block单元，图中的Ftr是传统的卷积结构，X和U是Ftr的输入（C'xH'xW'）和输出（CxHxW），这些都是以往结构中已存在的。SENet增加的部分是U后的结构：对U先做一个Global Average Pooling（图中的Fsq(.)，作者称为Squeeze过程），输出的1x1xC数据再经过两级全连接（图中的Fex(.)，作者称为Excitation过程），最后用sigmoid（论文中的self-gating mechanism）限制到[0，1]的范围，把这个值作为scale乘到U的C个通道上， 作为下一级的输入数据。这种结构的原理是想通过控制scale的大小，把重要的特征增强，不重要的特征减弱，从而让提取的特征指向性更强。



上图是SE 模块的示意图。给定一个输入 x，其特征通道数为 c_1，通过一系列卷积等一般变换后得到一个特征通道数为 c_2 的特征。与传统的 CNN 不一样的是，接下来通过三个操作来重标定前面得到的特征。

首先是 Squeeze 操作，顺着空间维度来进行特征压缩，将每个二维的特征通道变成一个实数，这个实数某种程度上具有全局的感受野，并且输出的维度和输入的特征通道数相匹配。它表征着在特征通道上响应的全局分布，而且使得靠近输入的层也可以获得全局的感受野，这一点在很多任务中都是非常有用的。

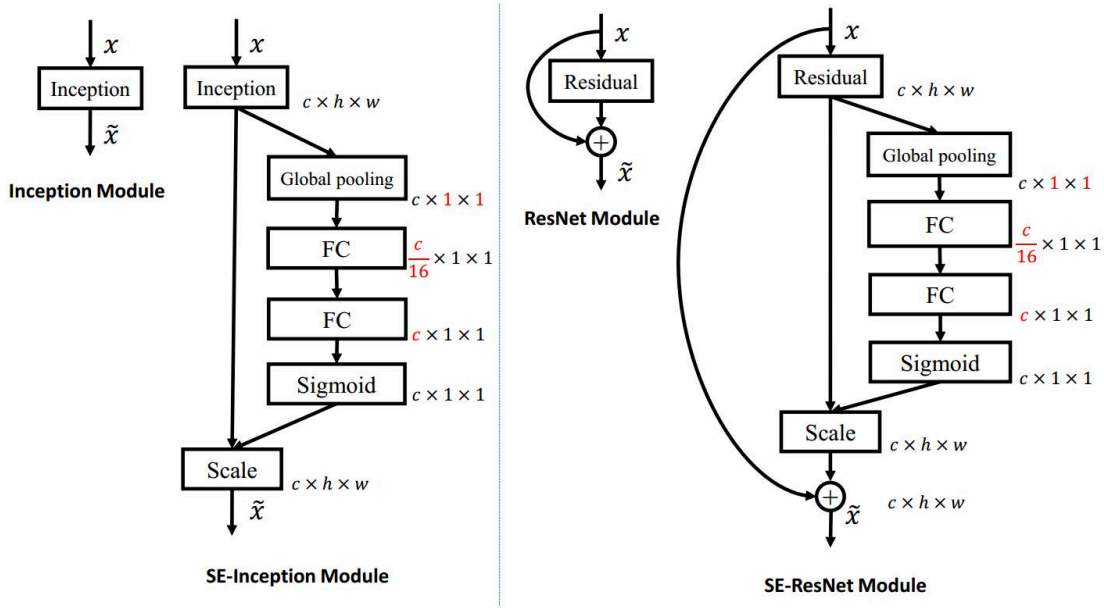
其次是 Excitation 操作，它是一个类似于循环神经网络中门的机制。通过参数 w 来为每个特征通道生成权重，其中参数 w 被学习用来显式地建模特征通道间的相关性。

最后是一个 Reweight 的操作，将 Excitation 的输出的权重看做是进过特征选择后的每个特征通道的重要性，然后通过乘法逐通道加权到先前的特征上，完成在通道维度上的对原始特征的重标定。

我们可以总结一下**中心思想**：对于每个输出 channel，预测一个常数权重，对每个 channel 加权一下，本质上，SE模块是在 channel 维度上做 attention 或者 gating 操作，这种注意力机制让模型可以更加关注信息量最大的 channel 特征，而抑制那些不重要的 channel 特征。SENet 一个很大的优点就是可以很方便地集成到现有网络中，提升网络性能，并且代价很小。

SE模块的应用

这是两个SENet实际应用的例子，左侧是SE-Inception的结构，即Inception模块和SENet组和在一起；右侧是SE-ResNet，ResNet和SENet的组合，这种结构scale放到了直连相加之前。



SE模块可以嵌入到现在几乎所有的网络结构中。通过在原始网络结构的 building block 单元中嵌入 SE模块，我们可以获得不同种类的 SENet。如SE-BN-Inception，SE-ResNet，SE-ReNeXt，SE-Inception-ResNet-v2等等

Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 (32 × 4d)
112 × 112	conv, 7 × 7, 64, stride 2		
56 × 56	max pool, 3 × 3, stride 2		
	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \\ \text{conv}, 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 64 \\ \text{conv}, 3 \times 3, 64 \\ \text{conv}, 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 \\ \text{conv}, 3 \times 3, 128 \quad C = 32 \\ \text{conv}, 1 \times 1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
28 × 28	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 \\ \text{conv}, 3 \times 3, 128 \\ \text{conv}, 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1 \times 1, 128 \\ \text{conv}, 3 \times 3, 128 \\ \text{conv}, 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 \\ \text{conv}, 3 \times 3, 256 \quad C = 32 \\ \text{conv}, 1 \times 1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
14 × 14	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 \\ \text{conv}, 3 \times 3, 256 \\ \text{conv}, 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1 \times 1, 256 \\ \text{conv}, 3 \times 3, 256 \\ \text{conv}, 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 \\ \text{conv}, 3 \times 3, 512 \quad C = 32 \\ \text{conv}, 1 \times 1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
7×7	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 \\ \text{conv}, 3 \times 3, 512 \\ \text{conv}, 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 512 \\ \text{conv}, 3 \times 3, 512 \\ \text{conv}, 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1 \times 1, 1024 \\ \text{conv}, 3 \times 3, 1024 \quad C = 32 \\ \text{conv}, 1 \times 1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
1 × 1	global average pool, 1000-d fc, softmax		

从上面的介绍中可以发现，SENet构造非常简单，而且很容易被部署，不需要引入新的函数或者层。除此之外，它还在模型和计算复杂度上具有良好的特性。拿 ResNet-50 和 SE-ResNet-50 对比举例来说，SE-ResNet-50 相对于 ResNet-50有着 10% 模型参数的增长，额外的模型参数都存在于 Bottleneck 设计的两个 Fully Connected 中。而 ResNet 结构中最后一个 stage 的特征通道数目为 2048，导致模型参数有着较大的增长，实现发现移除掉最后一个 stage 中 3个 build block 上的 SE设定，可以将 10%参数量的增长减少到 2%。此时模型的精度几乎无损失。

Model and Computational Complexity

SE-ResNet-50 vs. ResNet-50

- Parameters: 2%~10% additional parameters
- Computation cost: <1% additional computation (theoretical)
- GPU inference time: 10% additional time
- CPU inference time: <2% additional time

判断是否使用GPU

In [ ]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
```

In [7]:
```
class BasicBlock(nn.Module):
    def __init__(self, in_planes, planes, stride=1):
        """
        构造函数，初始化BasicBlock模块的各个组件

        :param in_planes: 输入通道数
        :param planes: 输出通道数
        :param stride: 步长
        """
        super(BasicBlock, self).__init__()
        # 第一个卷积层
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride,
        self.bn1 = nn.BatchNorm2d(planes)
        # 第二个卷积层
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=
        self.bn2 = nn.BatchNorm2d(planes)

        # shortcut
        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != planes:
            # 如果输入输出通道数不同或者步长不为1，则使用1x1卷积层
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, planes, kernel_size=1, stride=stride, bias=
                nn.BatchNorm2d(planes)
            )

        # SE layers
        self.fc1 = nn.Conv2d(planes, planes//16, kernel_size=1)  # Use nn.Conv2a
        self.fc2 = nn.Conv2d(planes//16, planes, kernel_size=1)

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))

        # Squeeze
        w = F.avg_pool2d(out, out.size(2))
        w = F.relu(self.fc1(w))
        w = F.sigmoid(self.fc2(w))
        # Excitation
        out = out * w
```

```
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

In [8]:
```
class PreActBlock(nn.Module):
    def __init__(self, in_planes, planes, stride=1):
        super(PreActBlock, self).__init__()
        self.bn1 = nn.BatchNorm2d(in_planes)
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=3, stride=stride,
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=1, padding=

        if stride != 1 or in_planes != planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, planes, kernel_size=1, stride=stride, bias=
            )

        # SE layers
        self.fc1 = nn.Conv2d(planes, planes//16, kernel_size=1)
        self.fc2 = nn.Conv2d(planes//16, planes, kernel_size=1)

    def forward(self, x):
        out = F.relu(self.bn1(x))
        shortcut = self.shortcut(out) if hasattr(self, 'shortcut') else x
        out = self.conv1(out)
        out = self.conv2(F.relu(self.bn2(out)))

        # Squeeze
        w = F.avg_pool2d(out, int(out.size(2)))
        w = F.relu(self.fc1(w))
        w = F.sigmoid(self.fc2(w))
        # Excitation
        out = out * w

        out += shortcut
        return out
```

In [9]:
```
class SENet(nn.Module):
    def __init__(self, block, num_blocks, num_classes=10):
        super(SENet, self).__init__()
        self.in_planes = 64

        self.conv1 = nn.Conv2d(3, 64, kernel_size=3, stride=1, padding=1, bias=F
        self.bn1 = nn.BatchNorm2d(64)
        self.layer1 = self._make_layer(block,  64, num_blocks[0], stride=1)
        self.layer2 = self._make_layer(block, 128, num_blocks[1], stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2], stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3], stride=2)
        self.linear = nn.Linear(512, num_classes)

    def _make_layer(self, block, planes, num_blocks, stride):
        strides = [stride] + [1]*(num_blocks-1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_planes, planes, stride))
            self.in_planes = planes
        return nn.Sequential(*layers)

    def forward(self, x):
```

```
        out = F.relu(self.bn1(self.conv1(x)))
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = F.avg_pool2d(out, 4)
        out = out.view(out.size(0), -1)
        out = self.linear(out)
        return out


def SENet18(num_classes=10):
    return SENet(PreActBlock, [2,2,2,2], num_classes=num_classes)
```

In [10]:
```
net = SENet18(num_classes=10).to(device)
```

In [11]:
```
summary(net,(3,32,32))
```

```
d:\anaconda\envs\cuda102\lib\site-packages\torch\nn\functional.py:1806: UserWarni
ng: nn.functional.sigmoid is deprecated. Use torch.sigmoid instead.
  warnings.warn("nn.functional.sigmoid is deprecated. Use torch.sigmoid instea
d.")
```

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,728
       BatchNorm2d-2           [-1, 64, 32, 32]             128
       BatchNorm2d-3           [-1, 64, 32, 32]             128
            Conv2d-4           [-1, 64, 32, 32]          36,864
       BatchNorm2d-5           [-1, 64, 32, 32]             128
            Conv2d-6           [-1, 64, 32, 32]          36,864
            Conv2d-7              [-1, 4, 1, 1]             260
            Conv2d-8             [-1, 64, 1, 1]             320
        PreActBlock-9           [-1, 64, 32, 32]               0
      BatchNorm2d-10           [-1, 64, 32, 32]             128
           Conv2d-11           [-1, 64, 32, 32]          36,864
      BatchNorm2d-12           [-1, 64, 32, 32]             128
           Conv2d-13           [-1, 64, 32, 32]          36,864
           Conv2d-14              [-1, 4, 1, 1]             260
           Conv2d-15             [-1, 64, 1, 1]             320
       PreActBlock-16           [-1, 64, 32, 32]               0
      BatchNorm2d-17           [-1, 64, 32, 32]             128
           Conv2d-18          [-1, 128, 16, 16]           8,192
           Conv2d-19          [-1, 128, 16, 16]          73,728
      BatchNorm2d-20          [-1, 128, 16, 16]             256
           Conv2d-21          [-1, 128, 16, 16]         147,456
           Conv2d-22              [-1, 8, 1, 1]           1,032
           Conv2d-23            [-1, 128, 1, 1]           1,152
       PreActBlock-24          [-1, 128, 16, 16]               0
      BatchNorm2d-25          [-1, 128, 16, 16]             256
           Conv2d-26          [-1, 128, 16, 16]         147,456
      BatchNorm2d-27          [-1, 128, 16, 16]             256
           Conv2d-28          [-1, 128, 16, 16]         147,456
           Conv2d-29              [-1, 8, 1, 1]           1,032
           Conv2d-30            [-1, 128, 1, 1]           1,152
       PreActBlock-31          [-1, 128, 16, 16]               0
      BatchNorm2d-32          [-1, 128, 16, 16]             256
           Conv2d-33            [-1, 256, 8, 8]          32,768
           Conv2d-34            [-1, 256, 8, 8]         294,912
      BatchNorm2d-35            [-1, 256, 8, 8]             512
           Conv2d-36            [-1, 256, 8, 8]         589,824
           Conv2d-37             [-1, 16, 1, 1]           4,112
           Conv2d-38            [-1, 256, 1, 1]           4,352
       PreActBlock-39            [-1, 256, 8, 8]               0
      BatchNorm2d-40            [-1, 256, 8, 8]             512
           Conv2d-41            [-1, 256, 8, 8]         589,824
      BatchNorm2d-42            [-1, 256, 8, 8]             512
           Conv2d-43            [-1, 256, 8, 8]         589,824
           Conv2d-44             [-1, 16, 1, 1]           4,112
           Conv2d-45            [-1, 256, 1, 1]           4,352
       PreActBlock-46            [-1, 256, 8, 8]               0
      BatchNorm2d-47            [-1, 256, 8, 8]             512
           Conv2d-48            [-1, 512, 4, 4]         131,072
           Conv2d-49            [-1, 512, 4, 4]       1,179,648
      BatchNorm2d-50            [-1, 512, 4, 4]           1,024
           Conv2d-51            [-1, 512, 4, 4]       2,359,296
           Conv2d-52             [-1, 32, 1, 1]          16,416
           Conv2d-53            [-1, 512, 1, 1]          16,896
       PreActBlock-54            [-1, 512, 4, 4]               0
      BatchNorm2d-55            [-1, 512, 4, 4]           1,024
           Conv2d-56            [-1, 512, 4, 4]       2,359,296
      BatchNorm2d-57            [-1, 512, 4, 4]           1,024
```

```
           Conv2d-58            [-1, 512, 4, 4]       2,359,296
           Conv2d-59             [-1, 32, 1, 1]          16,416
           Conv2d-60            [-1, 512, 1, 1]          16,896
      PreActBlock-61            [-1, 512, 4, 4]               0
           Linear-62                  [-1, 10]           5,130
================================================================
Total params: 11,260,354
Trainable params: 11,260,354
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 11.27
Params size (MB): 42.95
Estimated Total Size (MB): 54.23
----------------------------------------------------------------
```

In [12]:
```python
test_x = torch.randn(2,3,32,32).to(device)
test_y = net(test_x)
print(test_y.shape)
```

```
torch.Size([2, 10])
```

In [13]:
```python
# 检查 CUDA 是否可用
print("CUDA available:", torch.cuda.is_available())

# 获取当前 GPU 名称（如果有）
if torch.cuda.is_available():
    print("GPU:", torch.cuda.get_device_name(0))
else:
    print("No GPU found. Using CPU.")
```

```
CUDA available: True
GPU: GeForce MX350
```

In [ ]:
```python
# 定义模型并移动到 device
net = SENet18(num_classes=10).to(device)

# 如果 CUDA 可用，使用 DataParallel 并启用 cudnn.benchmark
if device == 'cuda':
    net = torch.nn.DataParallel(net)
    torch.backends.cudnn.benchmark = True  # 仅当输入尺寸固定时启用
```

训练网络

定义损失函数和优化器

1. 优化器 (Optimizer):

pytorch将深度学习中常用的优化方法全部封装在torch.optim之中，所有的优化方法都是继承基类optim.Optimizier。

使用随机梯度下降(SGD)优化器
初始学习率(lr)为 0.1
动量(momentum)为 0.9，有助于加速收敛
权重衰减(weight_decay)为 0.0005，用于L2正则化防止过拟合

2. 损失函数 (Criterion):

损失函数封装在神经网络工具箱nn中。

In [ ]:
```
import torch.optim as optim
optimizer = optim.SGD(net.parameters(), lr=1e-1, momentum=0.9, weight_decay=5e-4
criterion = nn.CrossEntropyLoss()
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer, 'min', factor=0.94 ,
# scheduler = optim.lr_scheduler.MultiStepLR(optimizer, milestones=[75, 150], ga
import time
epoch = 20
```

In [ ]:
```
import os
if not os.path.exists('./model'):
    os.makedirs('./model')
else:
    print('文件已存在')
save_path = './model/SENet.pth'
```

文件已存在

可视化训练过程

这次更新了tensorboard的可视化，可以得到更好看的图片，并且能可视化出不错的结果

In [18]:
```
# 使用tensorboard
from torch.utils.tensorboard import SummaryWriter
os.makedirs("./logs", exist_ok=True)
tbwriter = SummaryWriter(log_dir='./logs/SENet18', comment='SENet18')  # 使用ten
tbwriter.add_graph(model= net, input_to_model=torch.randn(size=(1, 3, 32, 32)))
```

```
C:\Users\33171\AppData\Local\Temp\ipykernel_19172\2857566879.py:25: TracerWarnin
g: Converting a tensor to a Python integer might cause the trace to be incorrect.
We can't record the data flow of Python values, so this value will be treated as
a constant in the future. This means that the trace might not generalize to other
inputs!
  w = F.avg_pool2d(out, int(out.size(2)))
```

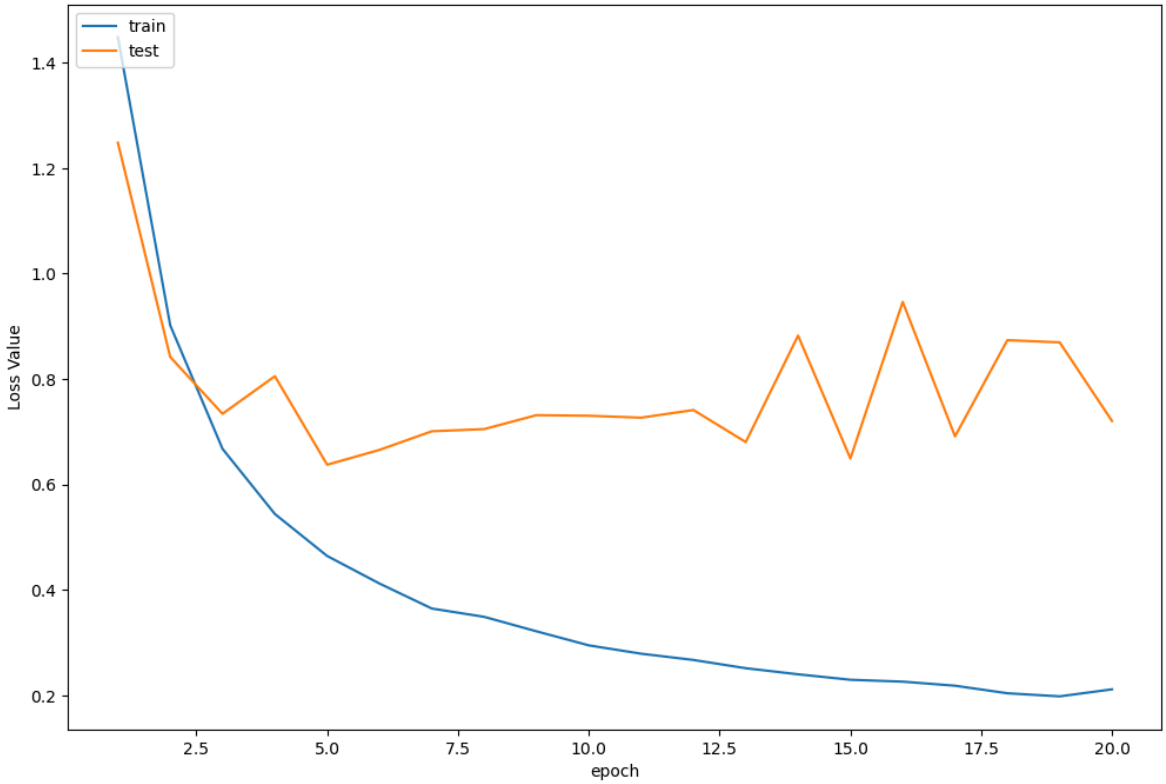
In [27]:
```
from utils import plot_history
from utils import train
Acc, Loss, Lr = train(net, trainloader, testloader, epoch, optimizer, criterion,
```

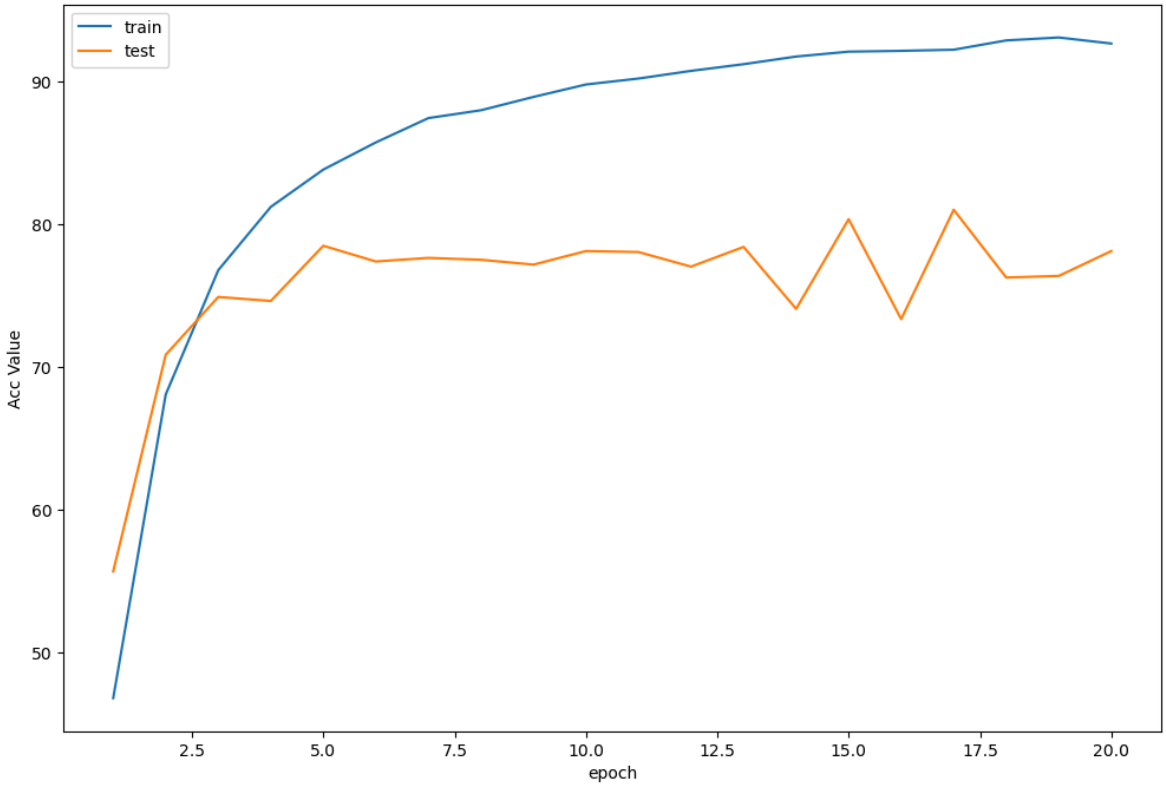
```
Train Epoch 1/20:   0%|          | 0/390 [00:00<?, ?it/s<class 'dict'>]Train Epoc
h 1/20: 100%|██████████| 390/390 [04:52<00:00,  1.34it/s, Train Acc=0.468, Train
Loss=1.45]
Test Epoch 1/20: 100%|██████████| 78/78 [00:24<00:00,  3.24it/s, Test Acc=0.557,
Test Loss=1.25]
Epoch [  1/ 20]  Train Loss:1.447881  Train Acc:46.82% Test Loss:1.248131  Test A
cc:55.70%  Learning Rate:0.100000
Train Epoch 2/20: 100%|██████████| 390/390 [04:48<00:00,  1.35it/s, Train Acc=0.6
81, Train Loss=0.902]
Test Epoch 2/20: 100%|██████████| 78/78 [00:20<00:00,  3.85it/s, Test Acc=0.709,
Test Loss=0.842]
Epoch [  2/ 20]  Train Loss:0.901953  Train Acc:68.09% Test Loss:0.841897  Test A
cc:70.87%  Learning Rate:0.100000
```

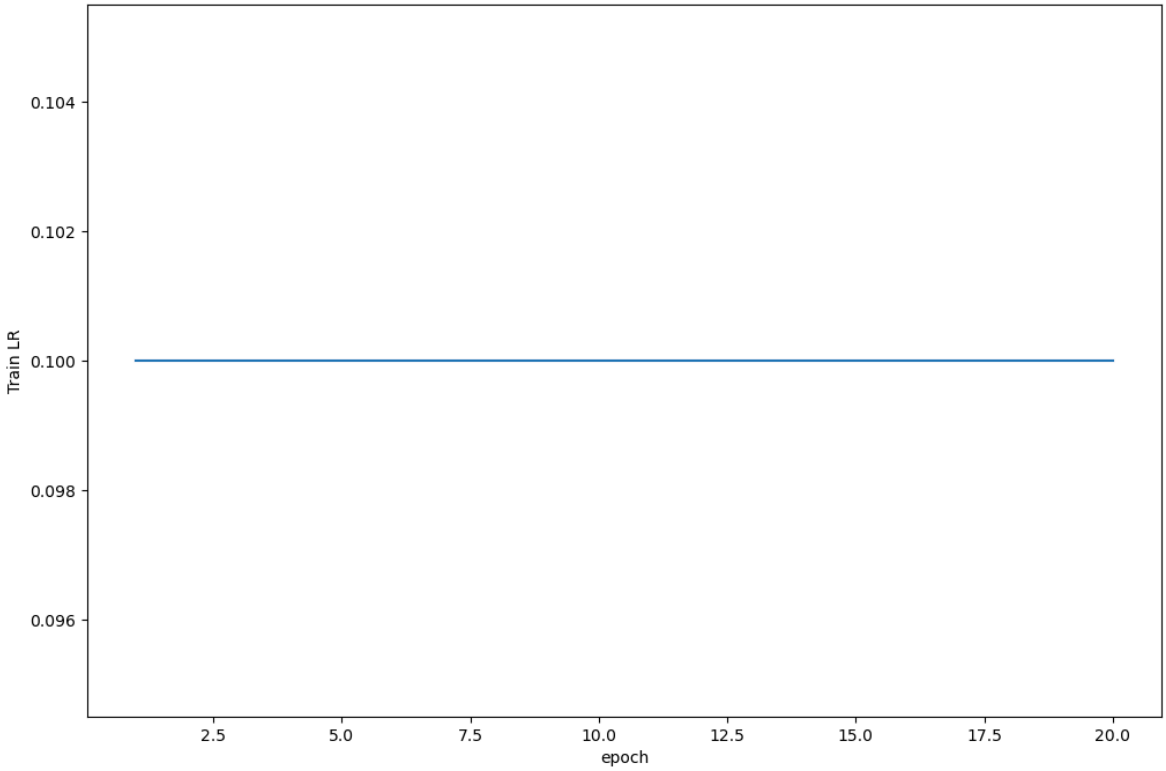
```
Train Epoch 3/20: 100%|██████████| 390/390 [04:51<00:00,  1.34it/s, Train Acc=0.768, Train Loss=0.668]
Test Epoch 3/20: 100%|██████████| 78/78 [00:18<00:00,  4.13it/s, Test Acc=0.749, Test Loss=0.734]
Epoch [  3/ 20]  Train Loss:0.667599  Train Acc:76.80% Test Loss:0.734083  Test Acc:74.92%  Learning Rate:0.100000
Train Epoch 4/20: 100%|██████████| 390/390 [04:58<00:00,  1.31it/s, Train Acc=0.812, Train Loss=0.544]
Test Epoch 4/20: 100%|██████████| 78/78 [00:19<00:00,  3.93it/s, Test Acc=0.746, Test Loss=0.805]
Epoch [  4/ 20]  Train Loss:0.544105  Train Acc:81.23% Test Loss:0.805343  Test Acc:74.64%  Learning Rate:0.100000
Train Epoch 5/20: 100%|██████████| 390/390 [04:40<00:00,  1.39it/s, Train Acc=0.838, Train Loss=0.464]
Test Epoch 5/20: 100%|██████████| 78/78 [00:18<00:00,  4.21it/s, Test Acc=0.785, Test Loss=0.637]
Epoch [  5/ 20]  Train Loss:0.464436  Train Acc:83.84% Test Loss:0.637488  Test Acc:78.51%  Learning Rate:0.100000
Train Epoch 6/20: 100%|██████████| 390/390 [04:35<00:00,  1.42it/s, Train Acc=0.857, Train Loss=0.412]
Test Epoch 6/20: 100%|██████████| 78/78 [00:18<00:00,  4.20it/s, Test Acc=0.774, Test Loss=0.666]
Epoch [  6/ 20]  Train Loss:0.412196  Train Acc:85.74% Test Loss:0.665640  Test Acc:77.40%  Learning Rate:0.100000
Train Epoch 7/20: 100%|██████████| 390/390 [04:47<00:00,  1.36it/s, Train Acc=0.874, Train Loss=0.365]
Test Epoch 7/20: 100%|██████████| 78/78 [00:20<00:00,  3.86it/s, Test Acc=0.777, Test Loss=0.701]
Epoch [  7/ 20]  Train Loss:0.364806  Train Acc:87.45% Test Loss:0.700921  Test Acc:77.65%  Learning Rate:0.100000
Train Epoch 8/20: 100%|██████████| 390/390 [04:31<00:00,  1.44it/s, Train Acc=0.88, Train Loss=0.349]
Test Epoch 8/20: 100%|██████████| 78/78 [00:18<00:00,  4.19it/s, Test Acc=0.775, Test Loss=0.705]
Epoch [  8/ 20]  Train Loss:0.349053  Train Acc:87.99% Test Loss:0.705037  Test Acc:77.52%  Learning Rate:0.100000
Train Epoch 9/20: 100%|██████████| 390/390 [04:42<00:00,  1.38it/s, Train Acc=0.889, Train Loss=0.322]
Test Epoch 9/20: 100%|██████████| 78/78 [00:47<00:00,  1.64it/s, Test Acc=0.772, Test Loss=0.731]
Epoch [  9/ 20]  Train Loss:0.321614  Train Acc:88.93% Test Loss:0.731482  Test Acc:77.18%  Learning Rate:0.100000
Train Epoch 10/20: 100%|██████████| 390/390 [08:28<00:00,  1.30s/it, Train Acc=0.898, Train Loss=0.295]
Test Epoch 10/20: 100%|██████████| 78/78 [00:18<00:00,  4.11it/s, Test Acc=0.781, Test Loss=0.73]
Epoch [ 10/ 20]  Train Loss:0.294959  Train Acc:89.80% Test Loss:0.730369  Test Acc:78.14%  Learning Rate:0.100000
Train Epoch 11/20: 100%|██████████| 390/390 [04:43<00:00,  1.37it/s, Train Acc=0.902, Train Loss=0.279]
Test Epoch 11/20: 100%|██████████| 78/78 [00:18<00:00,  4.19it/s, Test Acc=0.781, Test Loss=0.727]
Epoch [ 11/ 20]  Train Loss:0.279147  Train Acc:90.22% Test Loss:0.726721  Test Acc:78.06%  Learning Rate:0.100000
```

```
Train Epoch 12/20: 100%|██████████| 390/390 [04:47<00:00,  1.36it/s, Train Acc=0.908, Train Loss=0.267]
Test Epoch 12/20: 100%|██████████| 78/78 [00:19<00:00,  4.03it/s, Test Acc=0.77, Test Loss=0.741]
Epoch [ 12/ 20]  Train Loss:0.267259  Train Acc:90.75% Test Loss:0.741240  Test Acc:77.04%  Learning Rate:0.100000
Train Epoch 13/20: 100%|██████████| 390/390 [04:47<00:00,  1.35it/s, Train Acc=0.912, Train Loss=0.252]
Test Epoch 13/20: 100%|██████████| 78/78 [00:19<00:00,  4.07it/s, Test Acc=0.784, Test Loss=0.68]
Epoch [ 13/ 20]  Train Loss:0.251630  Train Acc:91.22% Test Loss:0.680468  Test Acc:78.43%  Learning Rate:0.100000
Train Epoch 14/20: 100%|██████████| 390/390 [04:46<00:00,  1.36it/s, Train Acc=0.918, Train Loss=0.24]
Test Epoch 14/20: 100%|██████████| 78/78 [00:18<00:00,  4.31it/s, Test Acc=0.741, Test Loss=0.882]
Epoch [ 14/ 20]  Train Loss:0.240072  Train Acc:91.76% Test Loss:0.882364  Test Acc:74.09%  Learning Rate:0.100000
Train Epoch 15/20: 100%|██████████| 390/390 [04:44<00:00,  1.37it/s, Train Acc=0.921, Train Loss=0.23]
Test Epoch 15/20: 100%|██████████| 78/78 [00:18<00:00,  4.23it/s, Test Acc=0.804, Test Loss=0.649]
Epoch [ 15/ 20]  Train Loss:0.229706  Train Acc:92.10% Test Loss:0.649343  Test Acc:80.37%  Learning Rate:0.100000
Train Epoch 16/20: 100%|██████████| 390/390 [04:51<00:00,  1.34it/s, Train Acc=0.922, Train Loss=0.226]
Test Epoch 16/20: 100%|██████████| 78/78 [00:18<00:00,  4.12it/s, Test Acc=0.734, Test Loss=0.946]
Epoch [ 16/ 20]  Train Loss:0.226149  Train Acc:92.16% Test Loss:0.946223  Test Acc:73.37%  Learning Rate:0.100000
Train Epoch 17/20: 100%|██████████| 390/390 [04:58<00:00,  1.31it/s, Train Acc=0.922, Train Loss=0.218]
Test Epoch 17/20: 100%|██████████| 78/78 [00:19<00:00,  4.10it/s, Test Acc=0.81, Test Loss=0.691]
Epoch [ 17/ 20]  Train Loss:0.218432  Train Acc:92.24% Test Loss:0.691324  Test Acc:81.03%  Learning Rate:0.100000
Train Epoch 18/20: 100%|██████████| 390/390 [05:03<00:00,  1.29it/s, Train Acc=0.929, Train Loss=0.204]
Test Epoch 18/20: 100%|██████████| 78/78 [00:20<00:00,  3.74it/s, Test Acc=0.763, Test Loss=0.874]
Epoch [ 18/ 20]  Train Loss:0.204105  Train Acc:92.89% Test Loss:0.873698  Test Acc:76.28%  Learning Rate:0.100000
Train Epoch 19/20: 100%|██████████| 390/390 [04:51<00:00,  1.34it/s, Train Acc=0.931, Train Loss=0.198]
Test Epoch 19/20: 100%|██████████| 78/78 [00:18<00:00,  4.17it/s, Test Acc=0.764, Test Loss=0.87]
Epoch [ 19/ 20]  Train Loss:0.198255  Train Acc:93.09% Test Loss:0.869534  Test Acc:76.39%  Learning Rate:0.100000
Train Epoch 20/20: 100%|██████████| 390/390 [04:57<00:00,  1.31it/s, Train Acc=0.927, Train Loss=0.212]
Test Epoch 20/20: 100%|██████████| 78/78 [00:17<00:00,  4.38it/s, Test Acc=0.781, Test Loss=0.72]
Epoch [ 20/ 20]  Train Loss:0.211568  Train Acc:92.67% Test Loss:0.720468  Test Acc:78.14%  Learning Rate:0.100000
```

In [ ]:
```
plot_history(epoch ,Acc, Loss, Lr)
```


train
test
Loss Value
epoch
1.4
1.2
1.0
0.8
0.6
0.4
0.2
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


train
test
Acc Value
epoch
90
80
70
60
50
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



测试

In [ ]:
```
correct = 0   # 定义预测正确的图片数，初始化为0
total = 0     # 总共参与测试的图片数，也初始化为0

torch.cuda.empty_cache()
print(f"GPU Memory allocated: {torch.cuda.memory_allocated()/1024**2:.2f} MB")
print(f"GPU Memory reserved: {torch.cuda.memory_reserved()/1024**2:.2f} MB")
```

```
GPU Memory allocated: 186.23 MB
GPU Memory reserved: 200.00 MB
```

In [ ]:
```
testloader = torch.utils.data.DataLoader(testset, batch_size=16,shuffle=True, nu
```

Accuracy

correct / total，标签预测正确的样本占所有样本的比例训练好的模型在测试集上的准确率

In [ ]:
```
for data in testloader:  # 循环每一个batch
    images, labels = data
    images = images.to(device)
    labels = labels.to(device)

    net = torch.load('./model/SENet.pth', map_location=device)
    net.eval()  # 把模型转为test模式

    if hasattr(torch.cuda, 'empty_cache'):
        torch.cuda.empty_cache()
    outputs = net(images)  # 输入网络进行测试

    # outputs.data是一个batch_sizex10张量，将每一行的最大的那一列的值和序号各自组成一
    _, predicted = torch.max(outputs.data, 1)
```

```
    total += labels.size(0)           # 更新测试图片的数量
    correct += (predicted == labels).sum() # 更新正确分类的图片的数量

print('Accuracy of the network on the 10000 test images: %.2f %%' % (100 * corre
```

```
Accuracy of the network on the 10000 test images: 81.01 %
```

单个类别的Precision

查准率：TP/(TP+FP)，被预测为该类别的样本中，有多少样本是预测正确的。

In [ ]:
```
# 初始化统计量
num_classes = 10
class_true_positives = [0] * num_classes  # 真正例（预测正确）
class_false_positives = [0] * num_classes  # 假正例（预测错误）

net.eval()
with torch.no_grad():
    for data in testloader:
        images, labels = data
        images = images.to(device)
        labels = labels.to(device)
        outputs = net(images)
        _, predicted = torch.max(outputs, 1)

        # 统计TP和FP
        for true_label, pred_label in zip(labels, predicted):
            true_label = true_label.item()
            pred_label = pred_label.item()

            if pred_label == true_label:
                class_true_positives[pred_label] += 1  # TP
            else:
                class_false_positives[pred_label] += 1  # FP

# 计算并输出每类查准率
print("{:<12} {:<12} {:<12} {:<12}".format(
    "Class", "TP", "FP", "Precision"))
print("-" * 45)
for i in range(num_classes):
    tp = class_true_positives[i]
    fp = class_false_positives[i]
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0.0  # 避免除以0

    print("{:<12} {:<12} {:<12} {:.2f}%".format(
        classes[i], tp, fp, precision * 100
    ))
```

```
Class        TP           FP           Precision
-----------------------------------------------
plane        899          312          74.24%
car          952          128          88.15%
bird         842          430          66.19%
cat          757          438          63.35%
deer         696          71           90.74%
dog          563          46           92.45%
frog         799          75           91.42%
horse        868          209          80.59%
ship         885          89           90.86%
truck        840          101          89.27%
```

保存模型

In [ ]:
```
torch.save(net,save_path[:-4]+'_'+str(epoch)+'.pth')
```

预测

In [ ]:
```
import torch
from PIL import Image
from torch.autograd import Variable
import torch.nn.functional as F
from torchvision import datasets, transforms
import numpy as np

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model = SENet18(num_classes=10)

model = torch.load(save_path)  # 加载模型
# model = model.to('cuda')
model.eval()  # 把模型转为test模式

# 读取要预测的图片
img = Image.open("./cat.jpg").convert('RGB') # 读取图像
```

输入CIFAR10数据集之外的图片，测试其预测效果

In [36]:
```
img
```

Out[36]:

In [37]:

```
trans = transforms.Compose([transforms.Resize((32,32)),
                            transforms.ToTensor(),
                            transforms.Normalize(mean=(0.5, 0.5, 0.5),
                                                 std=(0.5, 0.5, 0.5)),
                           ])

img = trans(img)
img = img.to(device)
# 图片扩展多一维,因为输入到保存的模型中是4维的[batch_size,通道,长，宽]，而普通图片只有
img = img.unsqueeze(0)
    # 扩展后，为[1，1，28，28]
output = model(img)
prob = F.softmax(output,dim=1) #prob是10个分类的概率
print("概率",prob)
value, predicted = torch.max(output.data, 1)
print("类别",predicted.item())
print(value)
pred_class = classes[predicted.item()]
print("分类",pred_class)
```

```
概率 tensor([[3.9969e-08, 1.3356e-07, 4.2326e-07, 9.9998e-01, 1.5980e-07, 1.5913e
-05,
         6.5965e-07, 1.7590e-06, 7.5294e-09, 7.7658e-08]], device='cuda:0',
       grad_fn=<SoftmaxBackward0>)
类别 3
tensor([13.6831], device='cuda:0')
分类 cat
```

分类正确

In [1]:

```
from IPython.display import display, Javascript

# 强制清除 Widget 状态
display(Javascript('''
    if (typeof Jupyter !== 'undefined') {
        Jupyter.notebook.metadata.widgets = {};
    }
```

```
'''))

# # 可选：重启内核以彻底清理
# from IPython import get_ipython
# get_ipython().kernel.restart()
```