''' [CIFAR-100分类]
https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html
https://github.com/weiaicunzai/pytorch-cifar100

[环境要求]

- torch
- numpy
- torchvision
- matplotlib

[数据] 在 datasets.CIFAR100 中设置 download=True 即可自动下载 '''

In [ ]:
```
from __future__ import print_function, division
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import time
import os
import copy
```

CIFAR-100数据集图像的均值和方差

In [ ]:
```
CIFAR100_TRAIN_MEAN = (0.5070751592371323, 0.48654887331
CIFAR100_TRAIN_STD = (0.2673342858792401, 0.256438462917
```

训练数据使用随机切割、水平翻转、随机旋转的数据增强，并进行归一化处理

验证数据仅使用归一化，不进行数据增强

In [ ]:
```
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomCrop(32, padding=4),
        transforms.RandomHorizontalFlip(),
        transforms.RandomRotation(15),
        transforms.ToTensor(),
        transforms.Normalize(CIFAR100_TRAIN_MEAN, CIFAR1
    ]),
    'val': transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(CIFAR100_TRAIN_MEAN, CIFAR1
    ]),
}
```

搭建训练和验证数据集

In [ ]:
```
Mode = {'train':True, 'val':False}

image_datasets = {x: datasets.CIFAR100(root=os.path.join
                                       transform=data
                                       download=True)
                  for x in ['train', 'val']}
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [ ]:
```
dataloaders = {x: torch.utils.data.DataLoader(image_data
                                              shuffle=Mc
               for x in ['train', 'val']}
```

获取数据集大小

In [ ]:
```
dataset_sizes = {x: len(image_datasets[x]) for x in ['tr
```

获取数据集的类别

In [ ]:
```
class_names = image_datasets['train'].classes
```

图像展示函数

In [ ]:
```
def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.1)
```

从数据集中取出一组样本

In [ ]:
```
inputs, classes = next(iter(dataloaders['train']))
```

将一组样本拼成一幅图像

In [ ]:
```
out = torchvision.utils.make_grid(inputs)
print(out.shape)
```

```
torch.Size([3, 546, 274])
```

在屏幕中展示图像

In [ ]:
```
plt.ion()
imshow(out, title=[class_names[x] for x in classes])
```

ResNet18的基本模块

In [ ]:
```python
class BasicBlock(nn.Module):

    expansion = 1

    def __init__(self, in_channels, out_channels, stride
        super().__init__()

        # 定义残差学习函数
        self.residual_function = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels * Basic
            nn.BatchNorm2d(out_channels * BasicBlock.exp
        )

        # 定义短路连接
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != BasicBlock.expa
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels * Ba
                nn.BatchNorm2d(out_channels * BasicBlock
            )

    def forward(self, x):
        return nn.ReLU(inplace=True)(self.residual_funct
```

定义通用的ResNet结构

In [ ]:
```
class ResNet(nn.Module):
    def __init__(self, block, num_block, num_classes=100
        super().__init__()
        self.in_channels = 64
        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1, b
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True))
        self.conv2_x = self._make_layer(block, 64, num_b
        self.conv3_x = self._make_layer(block, 128, num_
        self.conv4_x = self._make_layer(block, 256, num_
        self.conv5_x = self._make_layer(block, 512, num_
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * block.expansion, num_c

    def _make_layer(self, block, out_channels, num_block
        # 生成一个ResNet的基本单元
        strides = [stride] + [1] * (num_blocks - 1)
        layers = []
        for stride in strides:
            layers.append(block(self.in_channels, out_ch
            self.in_channels = out_channels * block.expa

        return nn.Sequential(*layers)

    def forward(self, x):
        # 向前传播
        output = self.conv1(x)
        output = self.conv2_x(output)
        output = self.conv3_x(output)
        output = self.conv4_x(output)
        output = self.conv5_x(output)
        output = self.avg_pool(output)
        output = output.view(output.size(0), -1)
        output = self.fc(output)
        return output
```

定义ResNet18

In [ ]:
```
def resnet18():
    return ResNet(BasicBlock, [2, 2, 2, 2])
```

搭建模型

In [ ]:
```
model_ft = resnet18()
```

重新设置模型的最后一层全连接层的输出通道数

In [ ]:
```
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, len(class_names))
```

设置使用GPU或CPU

In [ ]:
```
device = torch.device("mps" if torch.backends.mps.is_ava
```

设置模型为GPU或CPU

In [ ]:
```
model_ft = model_ft.to(device)
```

定义损失函数

In [ ]:
```
criterion = nn.CrossEntropyLoss()
```

定义优化器

In [ ]:
```
optimizer_ft = optim.Adam(model_ft.parameters(), lr=2e-4
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, ste
```

模型训练函数

In [ ]:
```
from IPython.display import clear_output

def train_model(model, criterion, optimizer, scheduler,
    train_losses = []
    train_acc = []
    val_losses = []
    val_acc = []
    since = time.time()

    # 定义记录最优模型以及最高准确率的变量
    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1
        print('-' * 10)

        # 每一个epoch包括训练和验证两个阶段
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train()  # 设置模型为训练模式
            else:
                model.eval()   # 设置模型为验证模式
```

```
            running_loss = 0.0
            running_corrects = 0

            # 在训练数据上迭代
            for inputs, labels in dataloaders[phase]:
                # 设置输入图像和标签为GPU或CPU
                inputs = inputs.to(device)
                labels = labels.to(device)

                # 优化器的参数归零
                optimizer.zero_grad()

                # 向前传播
                with torch.set_grad_enabled(phase == 'tr
                    outputs = model(inputs)
                    _, preds = torch.max(outputs, 1)
                    loss = criterion(outputs, labels)

                    # 若为训练模式，则反传梯度并更新模型参数
                    if phase == 'train':
                        loss.backward()
                        optimizer.step()

                # 记录信息
                running_loss += loss.item() * inputs.siz
                running_corrects += torch.sum(preds == l

            # 若为训练模式，则更新优化器的参数
            if phase == 'train':
                scheduler.step()

            # 记录并打印信息
            clear_output()
            epoch_loss = running_loss / dataset_sizes[ph
            epoch_acc = running_corrects / dataset_sizes

            if phase == "train":
                train_losses.append(epoch_loss)
                train_acc.append(epoch_acc)
            else:
                val_losses.append(epoch_loss)
                val_acc.append(epoch_acc)

            print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                phase, epoch_loss, epoch_acc))

            # 保存在验证集上性能最优的模型
            if phase == 'val' and epoch_acc > best_acc:
                best_acc = epoch_acc
                best_model_wts = copy.deepcopy(model.sta
```

```
        print()

    # 打印信息
    time_elapsed = time.time() - since
    print('Training complete in {:.0f}m {:.0f}s'.format(
        time_elapsed // 60, time_elapsed % 60))
    print('Best val Acc: {:4f}'.format(best_acc))

    # 加载性能最优的模型
    model.load_state_dict(best_model_wts)
    return model, train_losses, train_acc, val_losses, v
```

In [ ]:
```
model_ft, train_losses, train_acc, val_losses, val_acc =
                        num_epochs=25)
```

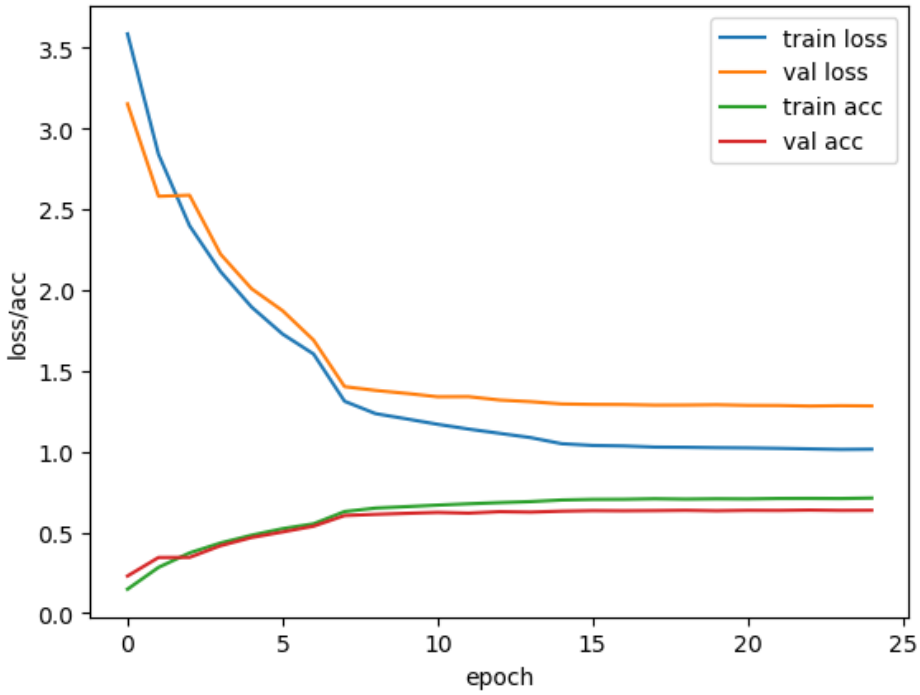
```
val Loss: 1.2824 Acc: 0.6363

Training complete in 22m 60s
Best val Acc: 0.637700
```

In [ ]:
```
for i in range(len(train_acc)):
    train_acc[i] = train_acc[i].cpu().numpy()

for i in range(len(val_acc)):
    val_acc[i] = val_acc[i].cpu().numpy()
```

In [ ]:
```
plt.figure()
plt.plot(train_losses, label="train loss")
plt.plot(val_losses, label="val loss")
plt.plot(train_acc, label="train acc")
plt.plot(val_acc, label="val acc")
plt.legend()
plt.xlabel("epoch")
plt.ylabel("loss/acc")
plt.savefig("pic/resnet18.png")
plt.show()
```



val Loss: 1.2824 Acc: 0.6363

Training complete in 22m 60s

Best val Acc: 0.637700

实验结果

- 显然，实验效果并不理想，且有轻微过拟合
- 但显然曲线也并未收敛到最佳效果，理论上继续训练，会逐渐收敛到最佳效果
- 但是，由于时间有限，暂时不再继续训练

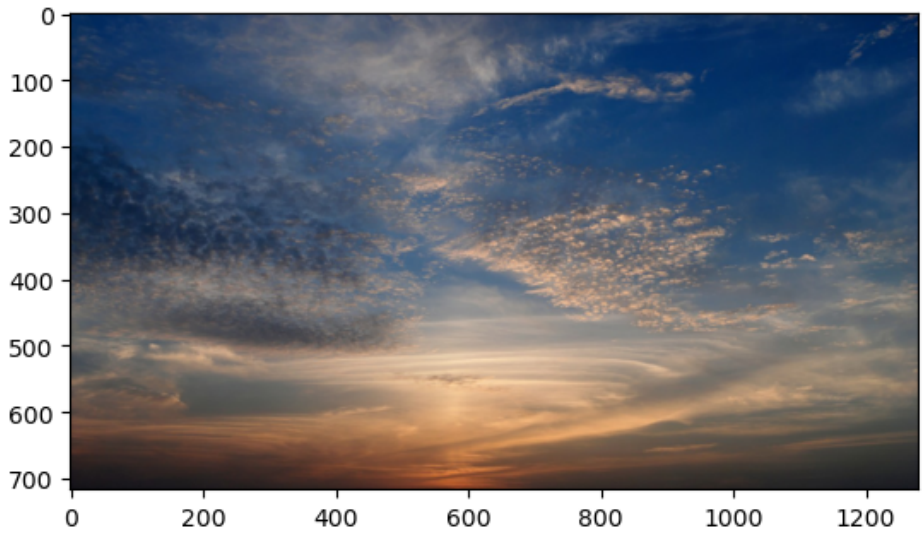
In [ ]:
```
# Load an image and convert it to a PyTorch tensor
from PIL import Image
img = Image.open("pic/test.jpg")

img_tensor = data_transforms['val'](img)

# Expand dimensions to match the model's input shape
img_tensor = img_tensor.unsqueeze(0)

# Predict the image's class
model_ft.eval()
predictions = model_ft.forward(img_tensor.to(device))
print(class_names[predictions.argmax()])
plt.imshow(img)
```

Out[ ]:
```
cloud
<matplotlib.image.AxesImage at 0x2cc8e2c50>
```



In [ ]: