Kaggle上的狗品种识别（ImageNet Dogs）

在本节中，我们将解决Kaggle竞赛中的犬种识别挑战，比赛的网址是https://www.kaggle.com/c/dog-breed-identification 在这项比赛中，我们尝试确定120种不同的狗。该比赛中使用的数据集实际上是著名的ImageNet数据集的子集。

In [2]:

```
# 在本节notebook中，使用后续设置的参数在完整训练集上训练模型，大致需要40-50分钟
# 请大家合理安排GPU时长，尽量只在训练时切换到GPU资源
# 也可以在Kaggle上访问本节notebook：
# https://www.kaggle.com/boyuai/boyu-d2l-dog-breed-identification-imagenet-dogs
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import torchvision.models as models
import os
import shutil
import time
import pandas as pd
import random
```

In [3]:

```
# 设置随机数种子
random.seed(0)
torch.manual_seed(0)
torch.cuda.manual_seed(0)
```

整理数据集

我们可以从比赛网址上下载数据集，其目录结构为：

```
| Dog Breed Identification
    | train
    |   | 000bec180eb18c7604dcecc8fe0dba07.jpg
    |   | 00a338a92e4e7bf543340dc849230e75.jpg
    |   | ...
    | test
    |   | 00a3edd22dc7859c487a64777fc8d093.jpg
    |   | 00a6892e5c7f92c1f465e213fd904582.jpg
    |   | ...
    | labels.csv
    | sample_submission.csv
```

train和test目录下分别是训练集和测试集的图像，训练集包含10,222张图像，测试集包含10,357张图像，图像格式都是JPEG，每张图像的文件名是一个唯一的id。labels.csv包含训练集图像的标签，文件包含10,222行，每行包含两列，第一列是图像id，第二列是狗的类别。狗的类别一共有120种。

我们希望对数据进行整理，方便后续的读取，我们的主要目标是：

- 从训练集中划分出验证数据集，用于调整超参数。划分之后，数据集应该包含4个部分：划分后的训练集、划分后的验证集、完整训练集、完整测试集
- 对于4个部分，建立4个文件夹：train, valid, train_valid, test。在上述文件夹中，对每个类别都建立一个文件夹，在其中存放属于该类别的图像。前三个部分的标签已知，所以各有120个子文件夹，而测试集的标签未知，所以仅建立一个名为unknown的子文件夹，存放所有测试数据。

我们希望整理后的数据集目录结构为：

```
| train_valid_test
    | train
    |   | affenpinscher
    |   |   | 00ca18751837cd6a22813f8e221f7819.jpg
    |   |   | ...
    |   | afghan_hound
    |   |   | 0a4f1e17d720cdff35814651402b7cf4.jpg
    |   |   | ...
    |   | ...
    | valid
    |   | affenpinscher
    |   |   | 56af8255b46eb1fa5722f37729525405.jpg
    |   |   | ...
    |   | afghan_hound
    |   |   | 0df400016a7e7ab4abff824bf2743f02.jpg
    |   |   | ...
    |   | ...
    | train_valid
    |   | affenpinscher
    |   |   | 00ca18751837cd6a22813f8e221f7819.jpg
    |   |   | ...
    |   | afghan_hound
    |   |   | 0a4f1e17d720cdff35814651402b7cf4.jpg
    |   |   | ...
    |   | ...
    | test
    |   | unknown
    |   |   | 00a3edd22dc7859c487a64777fc8d093.jpg
    |   |   | ...
```

In [4]:

```
data_dir = '/home/kesci/input/kaggledog9219/dog-breed-identification/dog-breed-identification'  # 数据集目录
label_file, train_dir, test_dir = 'labels.csv', 'train', 'test'  # data_dir中的文件夹、文件
new_data_dir = './train_valid_test'  # 整理之后的数据存放的目录
valid_ratio = 0.1  # 验证集所占比例
```

In [5]:

```
def mkdir_if_not_exist(path):
    # 若目录path不存在，则创建目录
    if not os.path.exists(os.path.join(*path)):
        os.makedirs(os.path.join(*path))

def reorg_dog_data(data_dir, label_file, train_dir, test_dir, new_data_dir, valid_ratio):
    # 读取训练数据标签
    labels = pd.read_csv(os.path.join(data_dir, label_file))
    id2label = {Id: label for Id, label in labels.values}  # (key: value): (id: label)

    # 随机打乱训练数据
    train_files = os.listdir(os.path.join(data_dir, train_dir))
    random.shuffle(train_files)

    # 原训练集
    valid_ds_size = int(len(train_files) * valid_ratio)  # 验证集大小
    for i, file in enumerate(train_files):
        img_id = file.split('.')[0]  # file是形式为id.jpg的字符串
        img_label = id2label[img_id]
        if i < valid_ds_size:
            mkdir_if_not_exist([new_data_dir, 'valid', img_label])
            shutil.copy(os.path.join(data_dir, train_dir, file),
                        os.path.join(new_data_dir, 'valid', img_label))
        else:
            mkdir_if_not_exist([new_data_dir, 'train', img_label])
            shutil.copy(os.path.join(data_dir, train_dir, file),
                        os.path.join(new_data_dir, 'train', img_label))
        mkdir_if_not_exist([new_data_dir, 'train_valid', img_label])
        shutil.copy(os.path.join(data_dir, train_dir, file),
                    os.path.join(new_data_dir, 'train_valid', img_label))

    # 测试集
    mkdir_if_not_exist([new_data_dir, 'test', 'unknown'])
    for test_file in os.listdir(os.path.join(data_dir, test_dir)):
        shutil.copy(os.path.join(data_dir, test_dir, test_file),
                    os.path.join(new_data_dir, 'test', 'unknown'))
```

In [6]:

```
reorg_dog_data(data_dir, label_file, train_dir, test_dir, new_data_dir, valid_ratio)
```

图像增强

In [7]:

```
transform_train = transforms.Compose([
    # 随机对图像裁剪出面积为原图像面积0.08~1倍、且高和宽之比在3/4~4/3的图像，再放缩为高和宽均为224像素的新图像
    transforms.RandomResizedCrop(224, scale=(0.08, 1.0),
                                 ratio=(3.0/4.0, 4.0/3.0)),
    # 以0.5的概率随机水平翻转
    transforms.RandomHorizontalFlip(),
    # 随机更改亮度、对比度和饱和度
    transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4),
    transforms.ToTensor(),
    # 对各个通道做标准化，(0.485, 0.456, 0.406)和(0.229, 0.224, 0.225)是在ImageNet上计算得的各通道均值与方差
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])  # ImageNet上的均值和方差
])

# 在测试集上的图像增强只做确定性的操作
transform_test = transforms.Compose([
    transforms.Resize(256),
    # 将图像中央的高和宽均为224的正方形区域裁剪出来
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
```

读取数据

In [8]:

```
# new_data_dir目录下有train, valid, train_valid, test四个目录
# 这四个目录中，每个子目录表示一种类别，目录中是属于该类别的所有图像
train_ds = torchvision.datasets.ImageFolder(root=os.path.join(new_data_dir, 'train'),
                                            transform=transform_train)
valid_ds = torchvision.datasets.ImageFolder(root=os.path.join(new_data_dir, 'valid'),
                                            transform=transform_test)
train_valid_ds = torchvision.datasets.ImageFolder(root=os.path.join(new_data_dir, 'train_valid'),
                                            transform=transform_train)
test_ds = torchvision.datasets.ImageFolder(root=os.path.join(new_data_dir, 'test'),
                                           transform=transform_test)
```

In [9]:

```
batch_size = 128
train_iter = torch.utils.data.DataLoader(train_ds, batch_size=batch_size, shuffle=True)
valid_iter = torch.utils.data.DataLoader(valid_ds, batch_size=batch_size, shuffle=True)
train_valid_iter = torch.utils.data.DataLoader(train_valid_ds, batch_size=batch_size, shuffle=True)
test_iter = torch.utils.data.DataLoader(test_ds, batch_size=batch_size, shuffle=False)  # shuffle=False
```

定义模型

这个比赛的数据属于ImageNet数据集的子集，我们使用微调的方法，选用在ImageNet完整数据集上预训练的模型来抽取图像特征，以作为自定义小规模输出网络的输入。

此处我们使用与训练的ResNet-34模型，直接复用预训练模型在输出层的输入，即抽取的特征，然后我们重新定义输出层，本次我们仅对重定义的输出层的参数进行训练，而对于用于抽取特征的部分，我们保留预训练模型的参数。

In [10]:

```
def get_net(device):
    finetune_net = models.resnet34(pretrained=False)  # 预训练的resnet34网络
    finetune_net.load_state_dict(torch.load('//home/kesci/input/resnet345000/resnet34-333f7ec4.pth'))
    for param in finetune_net.parameters():  # 冻结参数
        param.requires_grad = False
    # 原finetune_net.fc是一个输入单元数为512，输出单元数为1000的全连接层
    # 替换掉原finetune_net.fc，新finetuen_net.fc中的模型参数会记录梯度
    finetune_net.fc = nn.Sequential(
        nn.Linear(in_features=512, out_features=256),
        nn.ReLU(),
        nn.Linear(in_features=256, out_features=120)  # 120是输出类别数
    )
    return finetune_net
```

定义训练函数

In [11]:

```
def evaluate_loss_acc(data_iter, net, device):
    # 计算data_iter上的平均损失与准确率
    loss = nn.CrossEntropyLoss()
    is_training = net.training  # Bool net是否处于train模式
    net.eval()
    l_sum, acc_sum, n = 0, 0, 0
    with torch.no_grad():
        for X, y in data_iter:
            X, y = X.to(device), y.to(device)
            y_hat = net(X)
            l = loss(y_hat, y)
            l_sum += l.item() * y.shape[0]
            acc_sum += (y_hat.argmax(dim=1) == y).sum().item()
            n += y.shape[0]
    net.train(is_training)  # 恢复net的train/eval状态
    return l_sum / n, acc_sum / n
```

In [12]:

```
def train(net, train_iter, valid_iter, num_epochs, lr, wd, device, lr_period,
          lr_decay):
    loss = nn.CrossEntropyLoss()
    optimizer = optim.SGD(net.fc.parameters(), lr=lr, momentum=0.9, weight_decay=wd)
    net = net.to(device)
    for epoch in range(num_epochs):
        train_l_sum, n, start = 0.0, 0, time.time()
        if epoch > 0 and epoch % lr_period == 0:  # 每lr_period个epoch，学习率衰减一次
            lr = lr * lr_decay
            for param_group in optimizer.param_groups:
                param_group['lr'] = lr
        for X, y in train_iter:
            X, y = X.to(device), y.to(device)
            optimizer.zero_grad()
            y_hat = net(X)
            l = loss(y_hat, y)
            l.backward()
            optimizer.step()
            train_l_sum += l.item() * y.shape[0]
            n += y.shape[0]
        time_s = "time %.2f sec" % (time.time() - start)
        if valid_iter is not None:
            valid_loss, valid_acc = evaluate_loss_acc(valid_iter, net, device)
            epoch_s = ("epoch %d, train loss %f, valid loss %f, valid acc %f, "
                       % (epoch + 1, train_l_sum / n, valid_loss, valid_acc))
        else:
            epoch_s = ("epoch %d, train loss %f, "
                       % (epoch + 1, train_l_sum / n))
        print(epoch_s + time_s + ', lr ' + str(lr))
```

调参

In [13]:

```
num_epochs, lr_period, lr_decay = 5, 10, 0.1
lr, wd = 0.03, 1e-4
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

In [14]:

```
net = get_net(device)
train(net, train_iter, valid_iter, num_epochs, lr, wd, device, lr_period, lr_decay)
```

```
epoch 1, train loss 3.478195, valid loss 1.346200, valid acc 0.602740, time 240.51 sec, lr 0.03
epoch 2, train loss 1.535601, valid loss 0.763173, valid acc 0.758317, time 241.27 sec, lr 0.03
epoch 3, train loss 1.320487, valid loss 0.674160, valid acc 0.785714, time 241.40 sec, lr 0.03
epoch 4, train loss 1.203045, valid loss 0.597432, valid acc 0.806262, time 241.73 sec, lr 0.03
epoch 5, train loss 1.161204, valid loss 0.598735, valid acc 0.810176, time 241.73 sec, lr 0.03
```

在完整数据集上训练模型

In [16]:

```
# 使用上面的参数设置，在完整数据集上训练模型大致需要40-50分钟的时间
net = get_net(device)
train(net, train_valid_iter, None, num_epochs, lr, wd, device, lr_period, lr_decay)
```

```
epoch 1, train loss 3.237881, time 1092.35 sec, lr 0.03
epoch 2, train loss 1.481752, time 741.81 sec, lr 0.03
epoch 3, train loss 1.279390, time 267.66 sec, lr 0.03
epoch 4, train loss 1.208638, time 267.86 sec, lr 0.03
```

对测试集分类并提交结果

用训练好的模型对测试数据进行预测。比赛要求对测试集中的每张图片，都要预测其属于各个类别的概率。

In [1]:

```
preds = []
for X, _ in test_iter:
    X = X.to(device)
    output = net(X)
    output = torch.softmax(output, dim=1)
    preds += output.tolist()
ids = sorted(os.listdir(os.path.join(new_data_dir, 'test/unknown')))
with open('submission.csv', 'w') as f:
    f.write('id,' + ','.join(train_valid_ds.classes) + '\n')
    for i, output in zip(ids, preds):
        f.write(i.split('.')[0] + ',' + ','.join(
            [str(num) for num in output]) + '\n')
```