# 一、安装与准备数据集

## 1.1 安装Pytorch 和 Torchvision

### 1.1.1 什么是Pytorch？

PyTorch是一个[开源](https://baike.baidu.com/item/开源/246339?fromModule=lemma_inlink)的[Python](https://baike.baidu.com/item/Python?fromModule=lemma_inlink)机器学习库，基于Torch，用于自然语言处理等应用程序。是一个动态的框架，在运算过程中，会根据不同的数值，按照最优方式进行合理安排。

### 1.1.2 什么是Torchvision？

torchvision 包含了目前流行的数据集，模型结构和常用的图片转换工具。

### 1.1.3 安装命令

pip install torch==1.11.0  
pip install torchvision==0.12.0

## 1.2 引入库

import matplotlib  
import numpy as np # linear algebra  
import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)  
import os  
import torch  
import torch.nn as nn  
import cv2  
import matplotlib.pyplot as plt  
import torchvision  
from torch.utils.data import Dataset, DataLoader, ConcatDataset  
from torchvision import transforms, models  
from torch.optim.lr\_scheduler import \*  
import copy  
import random  
import tqdm  
from PIL import Image  
import torch.nn.functional as F

# 二、准备数据集

## 2.1 下载数据集

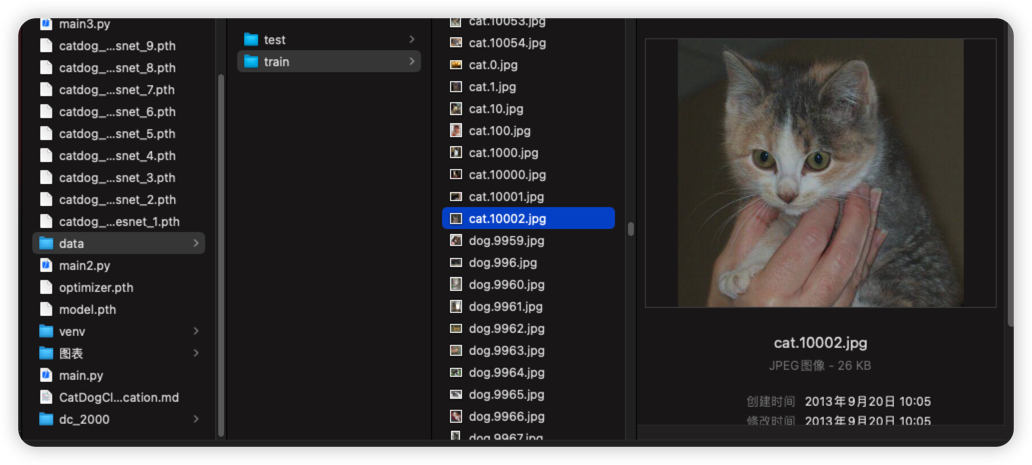
数据集描述：

The train folder contains 25,000 images of dogs and cats. Each image in this folder has the label as part of the filename. The test folder contains 12,500 images, named according to a numeric id. For each image in the test set, you should predict a probability that the image is a dog (1 = dog, 0 = cat).

train 文件夹包含 25,000 张狗和猫的图像。此文件夹中的每个图像都有标签作为文件名的一部分。测试文件夹包含 12,500 张图像，根据数字 id 命名。对于测试集中的每个图像，您应该预测图像是狗的概率（1 = 狗，0 = 猫）。

数据集下载链接：<https://www.kaggle.com/competitions/dogs-vs-cats-redux-kernels-edition/data>

下载并解压到到项目根目录中的 data 中，结构如下图所示：



## 2.2 定义超参数及配置

BATCH\_SIZE = 20 # 每批次的大小  
EPOCHS = 10 # 迭代次数  
DEVICE = torch.device("cuda" if torch.cuda.is\_available() else "cpu") # 使用cpu或者gpu  
  
cPath = os.getcwd() # 获取当前路径  
train\_dir = cPath + '/data/train' # 训练集路径  
test\_dir = cPath + '/data/test' # 测试集路径  
train\_files = os.listdir(train\_dir) # 训练集文件名下的所有文件  
test\_files = os.listdir(test\_dir) # 测试集文件名下的所有文件

## 2.3 数据处理与构建加载器

### 2.3.1 数据预处理

class CatDogDataset(Dataset):  
 def \_\_init\_\_(self, file\_list, dir, mode='train', transform=None):  
 self.file\_list = file\_list  
 self.dir = dir  
 self.mode = mode  
 self.transform = transform  
 if self.mode == 'train':  
 if 'dog' in self.file\_list[0]:  
 self.label = 1  
 else:  
 self.label = 0  
  
 def \_\_len\_\_(self):  
 return len(self.file\_list)  
  
 def \_\_getitem\_\_(self, idx):  
 img = Image.open(os.path.join(self.dir, self.file\_list[idx]))  
 if self.transform:  
 img = self.transform(img)  
 if self.mode == 'train':  
 img = img.numpy()  
 return img.astype('float32'), self.label  
 else:  
 img = img.numpy()  
 return img.astype('float32'), self.file\_list[idx]

### 2.3.2 训练数据加载器 - Train\_Loader

使用自定义的 transform 进行数据增强，它是对训练集进行变换，使训练集更丰富，从而让模型更具泛化能力，以及数据处理统一输入图片格式大小和归一化。

1. train\_transforms 先调整图片大小至 256x256 重置图像分辨率
2. 再按照 224x224 随机剪裁
3. 然后随机的图像水平翻转
4. 转化成 tensor，
5. 最后采用 ImageNet 给出的数值归一化。

接着构造 train dataloader，目的是为了方便读取和使用，设置 batch 大 小，采用多线程，shuffle=True 设置在每个 epoch 重新打乱数据，保证数据的随机性。

train\_transform = transforms.Compose([  
 transforms.Resize((256, 256)), # 先调整图片大小至256x256  
 transforms.RandomCrop((224, 224)), # 再随机裁剪到224x224  
 transforms.RandomHorizontalFlip(), # 随机的图像水平翻转，通俗讲就是图像的左右对调  
 transforms.ToTensor(),  
 transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)) # 归一化，数值是用ImageNet给出的数值  
])  
  
# 把猫的图片和狗的图片分开  
cat\_files = [tf for tf in train\_files if 'cat' in tf]  
dog\_files = [tf for tf in train\_files if 'dog' in tf]  
  
cats = CatDogDataset(cat\_files, train\_dir, transform=train\_transform) # 猫的数据集类  
dogs = CatDogDataset(dog\_files, train\_dir, transform=train\_transform) # 狗的数据集类  
  
train\_set = ConcatDataset([cats, dogs]) # 把猫和狗的数据集合并  
train\_loader = DataLoader(train\_set, batch\_size=BATCH\_SIZE, shuffle=True, num\_workers=0) # 训练集数据加载器

### 2.3.3 测试数据加载器 - Test\_Loader

test\_transform 重置图片分辨率 224x224，转化成 tensor，同样采用 ImageNet 给出的数值归一化。 接着构造 test dataloader，设置 batch size，采用多线程，shuffle=False。

test\_transform = transforms.Compose([  
 transforms.Resize((224, 224)),  
 transforms.ToTensor(),  
 transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))  
])  
  
test\_set = CatDogDataset(test\_files, test\_dir, mode='test', transform=test\_transform)  
test\_loader = DataLoader(test\_set, batch\_size=BATCH\_SIZE, shuffle=False, num\_workers=0)

## 2.4 预览一下图片

查看训练集经过 transform 处理的一个 batch 的图片集。在张量里, image 是 (batch, width, height), 所以我们需要转置成 (width, height, batch) 来展示。

samples, labels = iter(train\_loader).next()  
plt.figure(figsize=(16, 24))  
grid\_imgs = torchvision.utils.make\_grid(samples[:BATCH\_SIZE])  
np\_grid\_imgs = grid\_imgs.numpy()  
# in tensor, image is (batch, width, height), so you have to transpose it to (width, height, batch) in numpy to show it.  
plt.imshow(np.transpose(np\_grid\_imgs, (1, 2, 0)))

# 三、定义模型

## 3.1 模型方案选择

两种方案：

1. 采用迁移学习可使用**resnet50**、**resnet18**、**vgg16**和**vgg19**。
2. 或者**自己构建神经网络**（自己构建的准确率比较低，但是目的在于熟悉图片维度的计算）

### **自建**模型

class MineNet(nn.Module):  
 def \_\_init\_\_(self, num\_classes=2):  
 super().\_\_init\_\_()  
 self.features = nn.Sequential(  
 nn.Conv2d(3, 64, kernel\_size=11, stride=4, padding=2), # (224+2\*2-11)/4+1=55  
 nn.ReLU(inplace=True),  
 nn.MaxPool2d(kernel\_size=3, stride=2), # (55-3)/2+1=27  
 nn.Conv2d(64, 128, kernel\_size=5, stride=1, padding=2), # (27+2\*2-5)/1+1=27  
 nn.ReLU(inplace=True),  
 nn.MaxPool2d(kernel\_size=3, stride=2), # (27-3)/2+1=13  
 nn.Conv2d(128, 256, kernel\_size=3, stride=1, padding=1), # (13+1\*2-3)/1+1=13  
 nn.ReLU(inplace=True),  
 nn.Conv2d(256, 128, kernel\_size=3, stride=1, padding=1), # (13+1\*2-3)/1+1=13  
 nn.ReLU(inplace=True),  
 nn.Conv2d(128, 128, kernel\_size=3, stride=1, padding=1), # 13+1\*2-3)/1+1=13  
 nn.ReLU(inplace=True),  
 nn.MaxPool2d(kernel\_size=3, stride=2), # (13-3)/2+1=6  
 ) # 6\*6\*128=9126  
  
 self.avgpool = nn.AdaptiveAvgPool2d((6, 6))  
 self.classifier = nn.Sequential(  
 nn.Dropout(),  
 nn.Linear(128 \* 6 \* 6, 2048),  
 nn.ReLU(inplace=True),  
 nn.Dropout(),  
 nn.Linear(2048, 512),  
 nn.ReLU(inplace=True),  
 nn.Linear(512, num\_classes),  
 )  
 # softmax  
 self.logsoftmax = nn.LogSoftmax(dim=1)  
  
 def forward(self, x):  
 x = self.features(x)  
 x = self.avgpool(x)  
 x = x.view(x.size(0), -1)  
 x = self.classifier(x)  
 x = self.logsoftmax(x)  
 return x

model = MineNet()  
# model = MyConvNet().to(DEVICE)  
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, momentum=0.9, weight\_decay=5e-4) # 设置训练细节  
scheduler = StepLR(optimizer, step\_size=5) # 设置学习率下降策略  
criterion = nn.CrossEntropyLoss() # 设置损失函数

或者可以使用**迁移模型**方式：

### ResNet18

Pytorch 的 ResNet18 接口的最后一层全连接层的输出维度是 1000。这不符合猫狗大战数据集， 因为猫狗大战数据集是二分类的，所以最后一层全连接层输出的维度应该是 2 才对。因此我们需要对 ResNet18 进行最后一层的修改。取掉 ResNet18 model 的后 1 层，加上一层参数修改好的全连接层，输出为 2。训练、验证方法不变。

class Net(nn.Module):  
 def \_\_init\_\_(self, model):  
 super(Net, self).\_\_init\_\_()  
 self.resnet\_layer = nn.Sequential(\*list(model.children())[:-1])  
 self.Linear\_layer = nn.Linear(512, 2)  
 def forward(self, x):  
 x = self.resnet\_layer(x)   
 x = x.view(x.size(0), -1)   
 x = self.Linear\_layer(x)  
 return x  
   
from torchvision.models.resnet import resnet18  
resnet = resnet18(pretrained=True)  
model = Net(resnet)  
model = model.to(DEVICE)  
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9, weight\_decay=5e-4) # 设置训练细节  
scheduler = StepLR(optimizer, step\_size=3)  
criterion = nn.CrossEntropyLoss()

### VGG16 网络

因为猫狗大战数据集是二分类的，所以最后一层全连接层输出的维度应该是 2 才对。因此我们需 要对 VGG16 进行最后一层的修改。把 Pytorch 的 VGG16 接口 model 的 classifier 替换成输出为 2 分类的。训练、验证方法不变。

from torchvision.models.vgg import vgg16  
model = vgg16(pretrained=True)  
for parma in model.parameters():  
 parma.requires\_grad = False  
  
model.classifier = nn.Sequential(nn.Linear(25088, 4096),  
 nn.ReLU(),  
 nn.Dropout(p=0.5),  
 nn.Linear(4096, 4096),  
 nn.ReLU(),  
 nn.Dropout(p=0.5),  
 nn.Linear(4096, 2))  
  
for index, parma in enumerate(model.classifier.parameters()):  
 if index == 6:  
 parma.requires\_grad = True  
   
model = model.to(DEVICE)  
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9, weight\_decay=5e-4) # 设置训练细节  
scheduler = StepLR(optimizer, step\_size=3)  
criterion = nn.CrossEntropyLoss()

下文均使用**自建模型**的形式进行学习。

## 3.2 定义模型训练流水线

### 3.2.1 定义刷新数据流程

由于测试集没有 label，所以我把训练集按 9:1 划分成训练集和验证集，其中验证集是每轮 epoch 开始之前调用 refreshdataloader 在猫狗子集合中各抽 10%，意思就是验证集中猫狗图片各占一半，训 练集 22500 张图片，验证集 2500 张图片，验证集数据使用 test\_transform 处理。

def refreshdataloader():  
 cat\_files = [tf for tf in train\_files if 'cat' in tf]  
 dog\_files = [tf for tf in train\_files if 'dog' in tf]  
  
 val\_cat\_files = []  
 val\_dog\_files = []  
  
 for i in range(0, 1250):  
 r = random.randint(0, len(cat\_files) - 1)  
 val\_cat\_files.append(cat\_files[r])  
 val\_dog\_files.append(dog\_files[r])  
 cat\_files.remove(cat\_files[r])  
 dog\_files.remove(dog\_files[r])  
  
 cats = CatDogDataset(cat\_files, train\_dir, transform=train\_transform)  
 dogs = CatDogDataset(dog\_files, train\_dir, transform=train\_transform)  
  
 train\_set = ConcatDataset([cats, dogs])  
 train\_loader = DataLoader(train\_set, batch\_size=BATCH\_SIZE, shuffle=True, num\_workers=0)  
  
 val\_cats = CatDogDataset(val\_cat\_files, train\_dir, transform=test\_transform)  
 val\_dogs = CatDogDataset(val\_dog\_files, train\_dir, transform=test\_transform)  
  
 val\_set = ConcatDataset([val\_cats, val\_dogs])  
 val\_loader = DataLoader(val\_set, batch\_size=BATCH\_SIZE, shuffle=True, num\_workers=0)  
  
 return train\_loader, val\_loader

### 3.2.1 定义训练过程

def train(model, device, train\_loader, optimizer, epoch):  
 model.train()  
 train\_loss = 0.0  
 train\_acc = 0.0  
 percent = 10  
  
 for batch\_idx, (sample, target) in enumerate(train\_loader):  
 sample, target = sample.to(device), target.to(device)  
 optimizer.zero\_grad()  
 output = model(sample)  
 loss = criterion(output, target)  
 loss.backward()  
 optimizer.step()  
 loss = loss.item()  
 train\_loss += loss  
 pred = output.max(1, keepdim=True)[1]  
 train\_acc += pred.eq(target.view\_as(pred)).sum().item()  
  
 if (batch\_idx + 1) % percent == 0:  
 print('train epoch: {} [{}/{} ({:.0f}%)]\tloss: {:.6f}\t'.format(  
 epoch, (batch\_idx + 1) \* len(sample), len(train\_loader.dataset),  
 100. \* batch\_idx / len(train\_loader), loss))  
  
 train\_loss \*= BATCH\_SIZE  
 train\_loss /= len(train\_loader.dataset)  
 train\_acc = train\_acc / len(train\_loader.dataset)  
 print('\ntrain epoch: {}\tloss: {:.6f}\taccuracy:{:.4f}% '.format(epoch, train\_loss, 100. \* train\_acc))  
 scheduler.step()  
  
 return train\_loss, train\_acc

### 3.2.3 定义验证数据流程

def val(model, device, val\_loader, epoch):  
 model.eval()  
 val\_loss = 0.0  
 correct = 0  
 for sample, target in val\_loader:  
 with torch.no\_grad():  
 sample, target = sample.to(device), target.to(device)  
 output = model(sample)  
  
 val\_loss += criterion(output, target).item()  
 pred = output.max(1, keepdim=True)[1]  
 correct += pred.eq(target.view\_as(pred)).sum().item()  
  
 val\_loss \*= BATCH\_SIZE  
 val\_loss /= len(val\_loader.dataset)  
 val\_acc = correct / len(val\_loader.dataset)  
 print("\nval set: epoch{} average loss: {:.4f}, accuracy: {}/{} ({:.4f}%) \n"  
 .format(epoch, val\_loss, correct, len(val\_loader.dataset), 100. \* val\_acc))  
 return val\_loss, 100. \* val\_acc

### 3.2.4 定义测试数据流程

def test(model, device, test\_loader, epoch):  
 model.eval()  
 filename\_list = []  
 pred\_list = []  
 for sample, filename in test\_loader:  
 with torch.no\_grad():  
 sample = sample.to(device)  
 output = model(sample)  
 pred = torch.argmax(output, dim=1)  
  
 filename\_list += [n[:-4] for n in filename]  
 pred\_list += [p.item() for p in pred]  
  
 print("\ntest epoch: {}\n".format(epoch))  
  
 submission = pd.DataFrame({"id": filename\_list, "label": pred\_list})  
 submission.to\_csv('preds\_' + str(epoch) + '.csv', index=False)

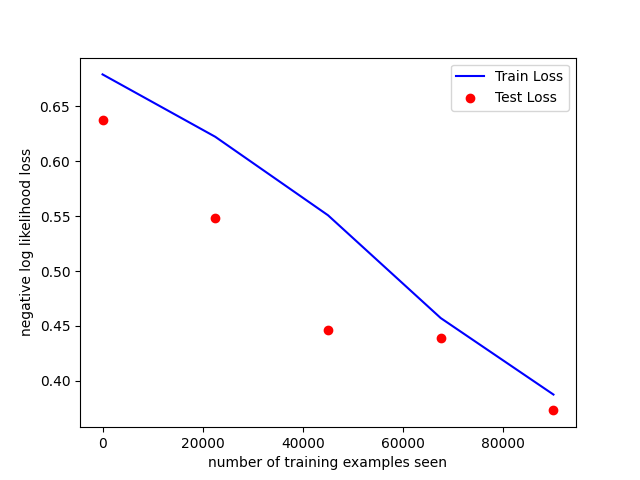
# 四、训练模型

train\_counter = [] # 训练集数量  
train\_losses = [] # 训练集损失  
train\_acces = [] # 训练集准确率  
val\_counter = [] # 验证集数量  
val\_losses = [] # 验证集损失  
val\_acces = [] # 验证集准确率  
  
for epoch in range(1, EPOCHS + 1):  
 # 刷新读取数据集  
 train\_loader, val\_loader = refreshdataloader()  
 # 开始训练并记录训练数据  
 tr\_loss, tr\_acc = train(model, DEVICE, train\_loader, optimizer, epoch)  
 train\_counter.append((epoch - 1) \* len(train\_loader.dataset))  
 train\_losses.append(tr\_loss)  
 train\_acces.append(tr\_acc)  
  
 # 验证当前训练的预测效果  
 vl, va = val(model, DEVICE, val\_loader, epoch)  
 val\_counter.append((epoch - 1) \* len(val\_loader.dataset))  
 val\_losses.append(vl)  
 val\_acces.append(va)  
  
 # 将当前批次模型保存下来  
 filename\_pth = 'catdog\_mineresnet\_' + str(epoch) + '.pth'  
 torch.save(model.state\_dict(), filename\_pth)  
  
test(model, DEVICE, test\_loader, 1)

展示训练过程图：

fig = plt.figure() # 创建图像  
plt.plot(train\_counter, train\_losses, color='blue') # 画出训练损失曲线  
plt.scatter(val\_counter, val\_losses, color='red') # 画出测试损失散点图  
plt.legend(['Train Loss', 'Test Loss'], loc='upper right') # 图例标识  
plt.xlabel('number of training examples seen') # x轴标签  
plt.ylabel('negative log likelihood loss') # y轴标签

展示结果如下：



# 五、模型预测结果展示

def showDemo():  
 examples = enumerate(test\_loader) # 枚举测试集  
 batch\_idx, (example\_data, example\_targets) = next(examples) # 获取第一个批次的数据  
 with torch.no\_grad(): # 不计算梯度，节省内存  
 y\_pred = model(example\_data) # 将数据输入网络，得到输出  
 \_, pred = torch.max(y\_pred.data, 1)  
 fig = plt.figure() # 创建图像  
  
 for i in range(6): # 画出前6个样本的预测结果  
 plt.subplot(2, 3, i + 1) # 2行3列，第 i+1 个子图  
 plt.tight\_layout() # 自动适配子图参数，使之填充整个图像区域  
 index = random.randint(0, len(example\_data) - 1)  
 plt.imshow(example\_data[index][0], interpolation='none') # 画出第 i 个样本  
 plt.title(  
 f"Predict Label: {labelList[pred[index]]}") # 标题为预测结果  
 plt.xticks([]) # 不显示 x 轴刻度  
 plt.yticks([]) # 不显示 y 轴刻度  
 plt.show() # 显示图像  
  
  
showDemo()

展示结果如下图：

