華東理工大學

模式识别大作业

题　　目　猫狗二分类

学　　院　信息科学与工程

专　　业　控制科学与工程

组　　员　林昊琪、郭奕杉、范玮

指导教师　赵海涛

完成日期：　**2019** 年 **11** 月 **29** 日

目录

猫狗二分类

一、实验目的

实验研究猫狗二分类问题。即给出一张猫或狗的图片，识别出这是猫还是狗。这是一个图像识别中的二分类问题，可以先采用一些图片尺寸放缩的方法，双线性插值或临近取样插值等，把不同尺寸的图片转换成同样格式的输入，把图片变成一样的分辨率，统一输入后可以采用卷积神经网络。卷积神经网络的细节架构参考一些经典的模型，Lenet 和 resnet，最后使用该 CNN 进行预测。

这种识别具有很重要的意义，比如：网络服务为了进行保护，会防止一些计算机进行恶意访问或信息爬取，进而设立一些验证问题，这些验证问题对于人来说很容易做，但是对于计算机这很困难。这样的方法称为全自动区分计算机和人类的图灵测试项目(Completely Automated Public Turing Test to Tell Computers and Humans Apart, CAPTCHA)或人类交互证明（Human Interactive Proof, HIP）。HIP 有很多用处，例如减少垃圾邮件，防止暴力破解密码等。

比较有名的用于限制访问的动物图像识别（Animal Species Image Recognition for Restricting Access, Asirra）就是一个 HIP，它会让用户识别图片信息，比如识别出图片中是猫还是狗。对于人来说这很容易，但是对于计算机很困难。以下是 Asirra 的一个例子：

寻找流浪宠物为其提供住所的网站——Petfinder.com，向微软研究院提供了超过三百万张猫和狗的图像，这些图片由美国各地成千上万的动物收容所手动分类。

对于要入侵的计算机，随机猜测一般是最简单的攻击方法。图片识别并不容易，因为图片之间不同的的背景，角度，姿势，亮度等都存在着巨大的差异，很难识别。

不过随着机器学习——尤其是神经网络的发展，这项工作精度可以达到 60%以上。而 60％分类器就已经能将 12 幅图像的猜测概率从 1/4096 提高到 1/459。

二、实验原理

1、卷积神经网络

卷积神经网络的出现被誉为计算机视觉和卷积神经网络（Convolutional

Neural Networks, CNN）发展十一座里程碑之一。

上世纪 60 年代，Hubel 等人通过对猫视觉皮层细胞的研究，提出了感受野这个概念，到 80 年代，Fukushima 在感受野概念的基础之上提出了神经认知机的概念，可以看作是卷积神经网络的第一个实现网络，神经认知机将一个视觉模式分解成许多子模式（特征），然后进入分层递阶式相连的特征平面进行处理，它试图将视觉系统模型化，使其能够在即使物体有位移或轻微变形的时候，也能完成识别。

卷积神经网络是多层感知机（Multilayer Perceptron, MLP）的变种，由生物学家休博尔和维瑟尔在早期关于猫视觉皮层的研究发展而来，视觉皮层的细胞存在一个复杂的构造，这些细胞对视觉输入空间的子区域非常敏感，称之为感受野。

CNN 由纽约大学的 Yann Lecun 于 1998 年提出，其本质是一个多层感知机，成功的原因在于其所采用的局部连接和权值共享的方式：一方面减少了权值的数量使得网络易于优化；另一方面降低了模型的复杂度，也就是减小了过拟合的风险。

该优点在网络的输入是图像时表现的更为明显，使得图像可以直接作为网络的输入，避免了传统识别算法中复杂的特征提取和数据重建的过程，在二维图像的处理过程中有很大的优势，如网络能够自行抽取图像的特征包括颜色、纹理、形状及图像的拓扑结构，在处理二维图像的问题上，特别是识别位移、缩放及其他形式扭曲不变性的应用上具有良好的鲁棒性和运算效率等。

2、灰度图像

一幅完整的图像，是由红色、绿色、蓝色三个通道组成的。红色、绿色、蓝色三个通道的缩览图都是以灰度显示的。用不同的灰度色阶来表示“红，绿，蓝”在图像中的比重。通道中的纯白，代表了该色光在此处为最高亮度，亮度级别是255。在计算机领域中，这类图像通常显示为从最暗黑色到最亮的白色的灰度，尽管理论上这个采样可以任何颜色的不同深浅，甚至可以是不同亮度上的不同颜色。灰度图像与黑白图像不同，在计算机图像领域中黑白图像只有黑色与白色两种颜色；灰度图像在黑色与白色之间还有许多级的颜色深度。我们可以通过下面几种方法，将其转换为灰度：

1）浮点算法：Gray=R*0.3+G*0.59+B*0.11

2）整数方法：Gray=(R*30+G*59+B*11)/100

3）移位方法：Gray=(R*76+G*151+B*28)>>8;

4）平均值法：Gray=（R+G+B）/3;

5）仅取绿色：Gray=G；

Gray 表示灰度，RGB 表示颜色值，R 表示红色，G 表示绿色，B 表示蓝色。通过上述任一种方法求得 Gray 后，将原来的 RGB(R,G,B)中的 R,G,B 统一用 Gray 替换，形成新的颜色 RGB(Gray,Gray,Gray)，用它替换原来的 RGB(R,G,B)就是灰度图了。

3、双线性差值

双线性插值，又称为双线性内插。在数学上，双线性插值是有两个变量的插值函数的线性插值扩展，其核心思想是在两个方向分别进行一次线性插值。双线性插值作为数值分析中的一种插值算法，广泛应用在信号处理，数字图像和视频处理等方面。

假如我们想得到未知函数 f 在点 P=(x,y)的值，假设我们已知函数 f 在 $Q_{11}=(x_1,y_1)$,$Q_{12}=(x_1,y_2)$,$Q_{21}=(x_2,y_1)$,及 $Q_{22}=(x_2,y_2)$四个点的值。

首先在 x 方向进行线性插值，得到

$$f(R_1)\approx\frac{x_2-x}{x_2-x_1}f(Q_{11})+\frac{x-x_1}{x_2-x_1}f(Q_{21})\quad Where\quad R_1=(x,y_1)$$

$$f(R_2)\approx\frac{x_2-x}{x_2-x_1}f(Q_{12})+\frac{x-x_1}{x_2-x_1}f(Q_{22})\quad Where\quad R_2=(x,y_2)$$

然后在 y 方向进行线性插值，得到

$$f(P)\approx\frac{y_2-y}{y_2-y_1}f(R_1)+\frac{y-y_1}{y_2-y_1}f(R_2)$$

这样就得到所要的结果 $f(x,y)$，即

$$f(x,y)\approx\frac{f(Q_{11})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y_2-y)+\frac{f(Q_{21})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y_2-y)+$$

$$\frac{f(Q_{12})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y_2-y)+\frac{f(Q_{22})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y_2-y)$$

如果选择一个坐标系统使得 f 的四个已知点坐标分别为(0,0)、(0,1)、(1,0)和(1,1)，那么插值公式就可以化简为

$$f(x,y)\approx f(0,0)(1-x)(1-y)+f(1,0)x(1-y)+f(0,1)(1-x)y+f(1,1)xy$$

或者用矩阵运算表示为

$$f(x,y)\approx\begin{bmatrix}1-x & x\end{bmatrix}\begin{bmatrix}f(0,0) & f(0,1)\\ f(1,0) & f(1,1)\end{bmatrix}\begin{bmatrix}1-y\\ y\end{bmatrix}$$

这种插值方法的结果通常不是线性的，线性插值的结果与插值的顺序无关。

首先进行 y 方向的插值，然后进行 x 方向的插值，但是所得到的结果是一样的。

三、实验步骤

3.1 数据预处理部分

对于从 lintcode 上下载到的猫狗图片，大致如下面几张图所示。

图 3-1 猫狗二分类数据集示意

其中狗的图片名为 dog 开头，猫的图片名为 cat 开头，由于图片在原始文件夹中是了混乱的，所以在 train 文件夹下建立了 dog 和 cat 文件夹，将所对应的猫狗类别的图片分别放进去。整理过后，cat 文件夹底下有 10000 张图片，dog 文件夹底下也有 10000 张图片。接着手动分割训练集和验证集。本次大作业中，选取每个类别 8000 张图片为训练集，2000 张图片为验证集，并且为了保证及时停止训练，每一个 epoch 进行一次验证。

3.2 以 pytorch 为框架编辑神经网络架构训练以及测试。

3.2.1dataset 数据读取部分

首先使用 torchvision.datasets 里的 ImageFolder 包读取数据。使用该包需要将不同类别的图片放入相应类别的文件夹内，在数据预处理的时候已经做过处理，因此可以读取。由于下载下来的数据集图片为 24 位 jpg 格式，即 RGB 三通道图像，**图片尺寸不一**，我们把每一张图片都进行预处理，主要是改变图片大小、数据类型、数据形式，故对于训练集的数据采取数据转换，具体来说，采取 resize 操作，resize 到 84*84 的尺寸，再转化成 tensor 并且进行归一化操作。同样的，这些操作也同样用于验证集和测试集。

3.2.2 网络部分

1998 年 Yann LeCun 在 IEEE 上发表了一篇 42 页题为《Gradient-based learning applied to document recognition》 的长文，文中首次提出卷积-池化-全连接的神经网络结构，由 LeCun 提出的七层网络命名为 LeNet-5。

LeNet-5 的简略网络结构具体如下所示：

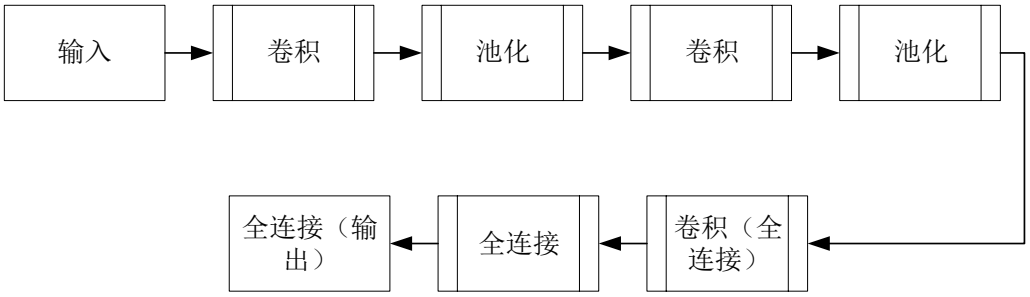

图 3-2　LeNet-5 的简略网络结构

计入层数的每层都有一定的训练参数，其中三个卷积层的训练参数较多，每层都有多个滤波器，也叫特征图，每个滤波器都对上一层的输出提取不同的像素特征。

然而 LeNet-5 提出后的十几年里，由于神经网络的可解释性问题和计算资源的限制，神经网络的发展一直处于低谷。直至 2012 年，Alex Krizhevsky 率先提出了 AlexNet，相较于 LeNet-5 的六万个参数，AlexNet 包含了 6 亿三千万条连接，6000 万个参数和 65 万个神经元，其网络结构包括 5 层卷积，其中第一、第二和第五层卷积后面连接了最大池化层，然后是 3 个全连接层。并在当年度的 ILSVRC（ImageNet 大规模视觉挑战赛）以显著的优势获得当届冠军，计算机视觉也开始逐渐进入深度学习主导的时代。此后不断有新的网络被提出，这些都极大的繁荣了深度学习的理论和实践，致使深度学习逐渐发展兴盛起来，CNN 结构演化的历史可以概括如下图所示。

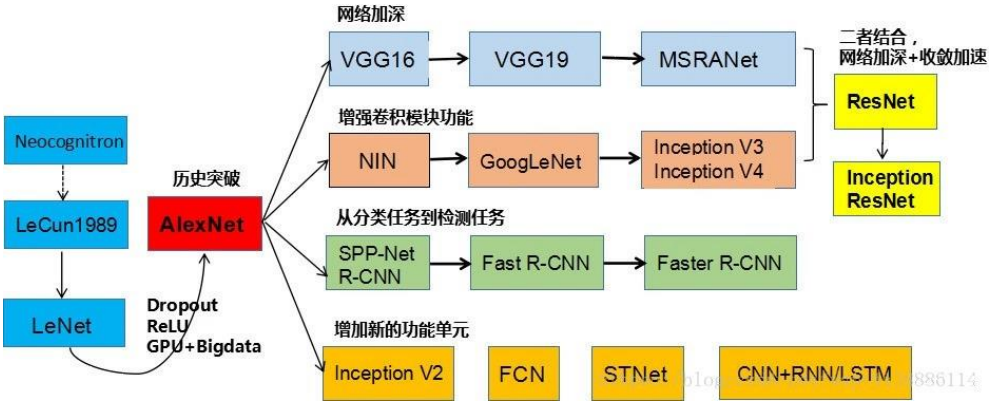

图 3-3 CNN 结构演化的历史

在发展的过程中，深度神经网络也不断面临新的问题，最主要的问题是梯度消失和梯度爆炸。其间通过设置 relu 和归一化激活函数层等手段可以解决这些问题，但当网络层数加到更深时却发现训练的准确率在逐渐降低，即出现了退化现象。因此，何恺明等学者借鉴了 Highway Network 思想在 2015 年提出了 ResNet 网络，而且影响了 2016 年深度学习在学术界和工业界的发展方向。

先前的卷积网络结构通常就是卷积池化再卷积池化，中间的卷积池化操作可以很多层。何恺明给出的创新在于给网络之间添加一个捷径，可以让捷径之间的网络能够学习一个恒等函数，使得在加深网络的情形下训练效果至少不会变差，

称为残差块。其基本结构如下：

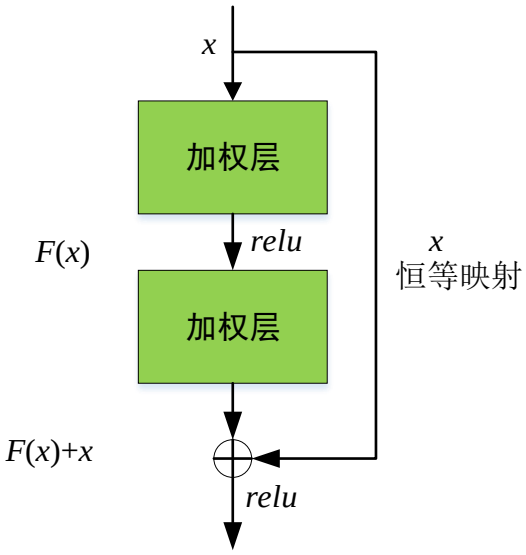

图 3-4 残差块的基本结构

从上图可以看出，输入 x 经过两层的加权和激活得到 $F(x)$的输出，这是典型的普通卷积网络结构。但残差块的区别在于添加了一个从输入 x 到两层网络输出单元的捷径，这使得输入节点的信息单元直接获得了与输出节点的信息单元通信的能力 ，这时候在进行 relu 激活之前的输出是 $F(x)+x$，其中残差函数 $F(x) = H(x) -x$。当很多个具备类似结构的这样的残差块组建到一起时，ResNet 就顺利形成了。ResNet 能够顺利训练很深层的卷积网络，其中能够很好的解决网络的退化问题。基于上述介绍分析，考虑到 CNN 能够提取 Low/mid/high-level 的特征，网络的层数越多，意味着提取到的不同等级的特征越丰富，但是单纯增加网络层数会有梯度消失和弥散。除此之外，越深的网络提取的特征越抽象，越具有语义信息，故对于猫狗二分类问题，采用两个网络来进行实验：LeNet 和 ResNet。整个网络的设计思想就是给网络一个输入即一张张猫狗图片，并给定输出是对应的猫或者狗信息(label)，然后网络按照特定的规则去调整自己的内部参数，使得自己实际计算的输出不断逼近给定的输出。

3.2.3 训练部分

我们采用 PyTorch 架构来编写训练的部分。PyTorch，是一种深度学习框架，简单的说就是 python 编程环境下的一个函数库，通过调用函数接口可以方便的搭建各种网络，进行训练，测试和分析。PyTorch 中数据以张量形式存在，可以理解为一个 N 维数组。

在深度学习框架下，搭建一个网络进行训练和测试十分方便，主要包括三个步骤：

1) 准备数据，将数据集中的数据整理成程序代码可识别读取的形式。

2) 搭建网络，利用 PyTorch 提供的 API 搭建设计的网络。

3) 训练网络，把 1 中准备好的数据送入 2 中搭建的网络中进行训练，获得网络各节点权值参数。

4) 测试网络，导入 3 中获取的参数，并输入网络一个数据，然后评估网络的输出结果。

本次大作业中采用 batch_size 为 8，训练的轮数为 100 轮，并且采用 tensor boardX 绘制训练，以此来验证集合的损失和准确率曲线。由于是分类问题，故选择的损失函数为交叉熵损失函数 CrossEntropyLoss()，采用 SGD(随机梯度下降)的方法来进行梯度的更新。每一轮计算训练集的准确率损失以及验证集合的准确率和损失。当验证集的损失不再下降同时准确率不再上升时，手动停止训练。

3.2.4 测试部分

使用 PIL 来读取图片，将训练完成保存的模型结构加载进来，将图片同样做数据预处理后输入网络，完成猫狗分类任务，最终得到预测结果。

四、实验结果分析与总结

LeNet 网络上训练以及验证曲线如下图所示：

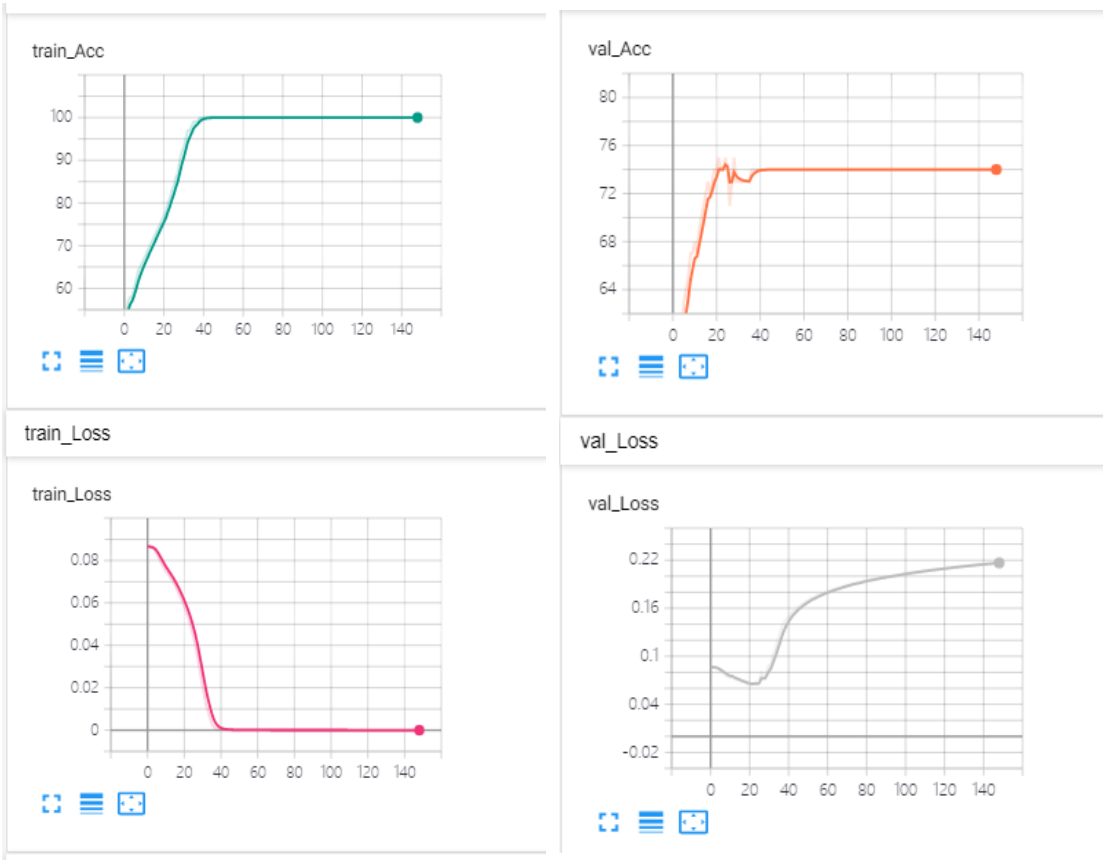

图 4-1 LeNet 网络训练及验证曲线

可以看出训练集上的损失不断下降，准确率不断上升。在第 40 轮时趋向平缓，说明在训练集上已经收敛。在验证集上，准确率在 25 轮左右达到最高，损失最低。之后验证集损失有所上升，是由于出现了过拟合的原因导致验证集的损失上升。在训练时应在验证集损失达到最小也就是第 25 轮停止训练。

ResNet 的训练验证曲线则如下图所示：

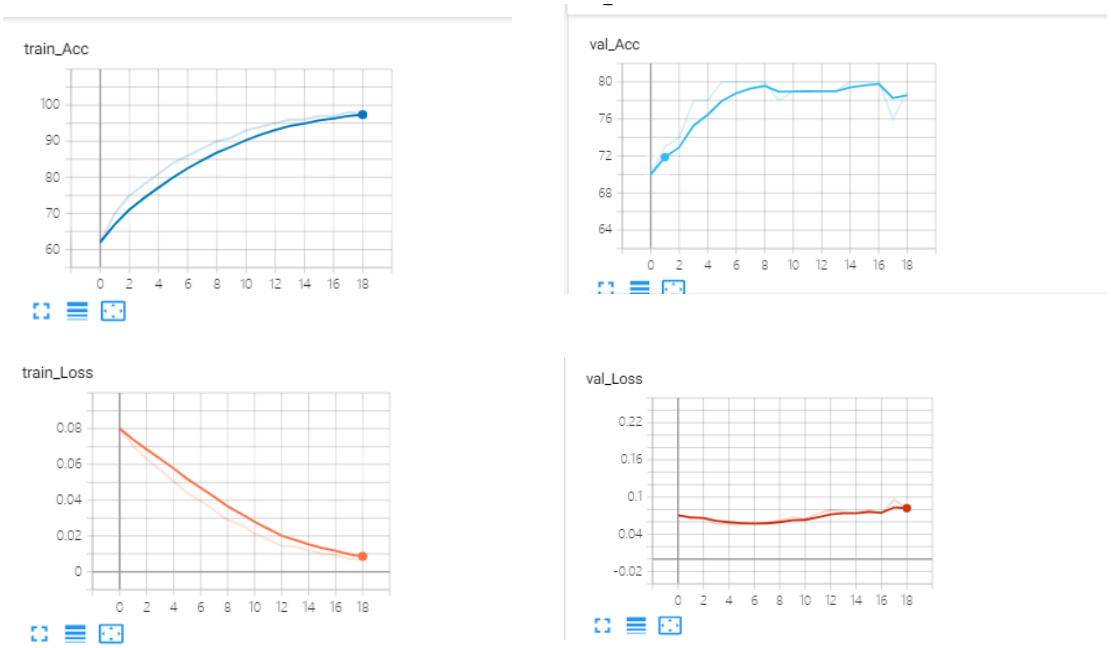

图 4-2 ResNet 网络训练及验证曲线

可以从图中看出，resnet 在训练的第 18 轮左右就已经收敛，验证集和训练集的曲线和 lenet 相似。但是可以可出 resnet 在验证集上的过拟合现象不严重，符合该网络特点。

以下是提交在 lintcode 上的排名以及得分。

图 4-3 最终排名与得分

五、心得体会

程序编写：范玮、郭奕杉

程序调试：林昊琪

实验报告：林昊琪、范玮、郭奕杉

通过此次大作业，使我们更加扎实的掌握了有关模式识别方面的知识，在程序实现过程中虽然遇到了一些问题，但经过一次又一次的思考，一遍又一遍的检查，终于找出了原因所在。我们在调试程序的过程中也明白一个道理，要不断反

思、总结，避免重复同样的错误，同时也要磨练自己的耐心，山重水复疑无路，柳暗花明又一村。在调试程序的过程中，我们不断发现错误，不断改正，不断领悟，不断获取。在此过程中，我们通过查找大量资料，请教老师和同学，不仅加深了我们对理论知识的理解，还加强了我们写程序该程序的动手实践能力。

附录

1.数据预处理

```
import os, shutil
from PIL import Image

 src_dir = os.path.abspath(r"D:\\lhq\\catdog\\train\\")
 dst1_dir = os.path.abspath(r"D:\\lhq\\catdog\\train\\dog")
 dst2_dir = os.path.abspath(r"D:\\lhq\\catdog\\train\\cat")
 for root,dirs,files in os.walk(src_dir):
            for file in files:
                if file[0:3] == 'dog':
                    src_file = os.path.join(root, file)
                    shutil.copy(src_file, dst1_dir)
                else:
                    src_file = os.path.join(root, file)
                    shutil.copy(src_file, dst2_dir)
```

2、数据读取

```
import torch.utils.data as data
from PIL import Image
import os
import os.path
from torchvision.datasets import ImageFolder
from torchvision import models, transforms
data_transform = transforms.Compose([
    transforms.Resize((84,84)),
    transforms.ToTensor(),
    transforms.Normalize(mean = [0.5,0.5,0.5],std = [0.5,0.5,0.5])
])
class train_data(data.Dataset):
```

```
    def __init__(self, root):
        img_array = ImageFolder(root,transform=data_transform)
        self.data=img_array
    def __getitem__(self, index):
        img,label = self.data[index]
        return img, label
    def __len__(self):
        # 返回图像的数量
        return len(self.data)
class val_data(data.Dataset):
    def __init__(self, root):
        img_array = ImageFolder(root,transform=data_transform)
        self.data=img_array
    def __getitem__(self, index):
        img,label = self.data[index]
        return img, label
    def __len__(self):
        # 返回图像的数量
        return len(self.data)
class test_data(data.Dataset):
    def __init__(self, root):
        img_array = ImageFolder(root,transform=data_transform)
        self.data=img_array
    def __getitem__(self, index):
        img, label= self.data[index]
        return img, label
    def __len__(self):
        # 返回图像的数量
return len(self.data)
```

3、network 代码

```
import numpy as np
import torch
from torch import nn
from torch.autograd import Variable
```

```
import torch.nn.functional as F
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 18 * 18, 800)
        self.fc2 = nn.Linear(800, 120)
        self.fc3 = nn.Linear(120, 2)
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 18 * 18)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
def conv3x3(in_channel, out_channel, stride=1):
    return nn.Conv2d(in_channel, out_channel, 3, stride=stride, padding=1,
bias=False)
class residual_block(nn.Module):
    def __init__(self, in_channel, out_channel, same_shape=True):
        super(residual_block, self).__init__()
        self.same_shape = same_shape
        stride = 1 if self.same_shape else 2
        self.conv1 = conv3x3(in_channel, out_channel, stride=stride)
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.conv2 = conv3x3(out_channel, out_channel)
        self.bn2 = nn.BatchNorm2d(out_channel)
        if not self.same_shape:
            self.conv3 = nn.Conv2d(in_channel, out_channel, 1, stride=stride)
    def forward(self, x):
        out = self.conv1(x)
        out = F.relu(self.bn1(out), True)
```

```
            out = self.conv2(out)
            out = F.relu(self.bn2(out), True)
            if not self.same_shape:
                x = self.conv3(x)
            return F.relu(x + out, True)
class resnet(nn.Module):
    def __init__(self, in_channel, num_classes, verbose=False):
        super(resnet, self).__init__()
        self.verbose = verbose
        self.block1 = nn.Conv2d(in_channel, 64, 7, 2)
        self.block2 = nn.Sequential(
            nn.MaxPool2d(3, 2),
            residual_block(64, 64),
            residual_block(64, 64)
        )
        self.block3 = nn.Sequential(
            residual_block(64, 128, False),
            residual_block(128, 128)
        )
        self.block4 = nn.Sequential(
            residual_block(128, 256, False),
            residual_block(256, 256)
        )
        self.block5 = nn.Sequential(
            residual_block(256, 512, False),
            residual_block(512, 512),
            nn.AvgPool2d(3)
        )
        self.classifier = nn.Linear(512, num_classes)
    def forward(self, x):
        x = self.block1(x)
        if self.verbose:
            print('block 1 output: {}'.format(x.shape))
        x = self.block2(x)
        if self.verbose:
```

```
            print('block 2 output: {}'.format(x.shape))
        x = self.block3(x)
        if self.verbose:
            print('block 3 output: {}'.format(x.shape))
        x = self.block4(x)
        if self.verbose:
            print('block 4 output: {}'.format(x.shape))
        x = self.block5(x)
        if self.verbose:
            print('block 5 output: {}'.format(x.shape))
        x = x.view(-1,512)
        #print(x.shape)
        x = self.classifier(x)
        return x
```

4、训练

```
from network import Net,resnet
from dataset import train_data,val_data
import os
import time
import torch
from torchvision import models, transforms
from torch import optim, nn
from torch.autograd import Variable
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
from tensorboardX import SummaryWriter
batch_size=8
epochs =100
train_root='D:\\lhq\\catdog\\train\\'
val_root='D:\\lhq\\catdog\\val\\'
train_dataset=train_data(train_root)
val_dataset=val_data(val_root)
print()
train_loader=DataLoader(train_dataset,batch_size=batch_size,
```

```
shuffle=True,num_workers=0)
val_loader=DataLoader(val_dataset,batch_size=batch_size,
shuffle=False,num_workers=0)
if torch.cuda.is_available()==True:
    #net=resnet(3, 2, False).cuda()
    net = Net().cuda()
cirterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.0001, momentum=0.9)
writer = SummaryWriter()
for epoch in range(epochs):
    running_loss = 0.0
    train_correct = 0
    train_total = 0
    for i, data in enumerate(train_loader):
        net.train()
        inputs, train_labels = data
        inputs, labels = Variable(inputs.cuda()), Variable(train_labels.cuda())
        optimizer.zero_grad()
        outputs = net(inputs)
        _, train_predicted = torch.max(outputs.data, 1)#输出每一行最大的元素
        train_correct += (train_predicted == labels.data).sum()
        loss = cirterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        train_total += train_labels.size(0)
    val_loss = 0.0
    val_correct = 0
    val_total = 0
    for i, data in enumerate(val_loader):
        net.eval()
        inputs, val_labels = data
        inputs, labels = Variable(inputs.cuda()), Variable(val_labels.cuda())
        outputs = net(inputs)
        _, val_predicted = torch.max(outputs.data, 1)  # 输出每一行最大的元素
```

```
            val_correct += (val_predicted == labels.data).sum()
            loss = cirterion(outputs, labels)
            val_loss += loss.item()
            val_total += val_labels.size(0)
        print('train %d epoch loss: %.3f   acc: %.3f val loss: %.3f   val acc: %.3f ' % (
            epoch + 1, running_loss / train_total, 100 * train_correct / train_total,val_loss /val_total,100 * val_correct / val_total))
        with open("log_Lenet.txt", "a") as f:
            f.write('train %d epoch loss: %.3f   acc: %.3f val loss: %.3f   val acc: %.3f \n' % (
            epoch + 1, running_loss / train_total, 100 * train_correct / train_total,val_loss /val_total,100 * val_correct / val_total))
        torch.save(net, 'model_'+str(epoch)+'.pt')
        # ====================== 使用 tensorboard ==================
        writer.add_scalars('train_Loss', {'train': running_loss / train_total},epoch)
        writer.add_scalars('train_Acc', {'train': 100*train_correct / train_total}, epoch)
        writer.add_scalars('val_Loss', {'val': val_loss /val_total}, epoch)
        writer.add_scalars('val_Acc', {'val': 100 * val_correct / val_total}, epoch)
        #
==========================================================
torch.save(net, 'model.pt')
```

5、测试

```
import os
import torch
from torchvision import models, transforms
from torch.autograd import Variable
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
from dataset import test_data
import pandas as pd
from PIL import Image
mysub = pd.read_csv('mysub.csv')
model = torch.load('model_122.pt')
model.eval()
```

```
data_transform = transforms.Compose([
        transforms.Resize((84,84)),
        transforms.ToTensor(),
        transforms.Normalize(mean = [0.5,0.5,0.5],std = [0.5,0.5,0.5])
])
my_predict=[]
for i in range(5000):
        image = Image.open('D:\\lhq\\catdog\\test\\pic\\'+str(i)+'.jpg')
        image_transformed = data_transform(image)
        images = image_transformed.unsqueeze(0)
        images= Variable(images.cuda())
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        predicted=predicted.cpu().numpy()
        my_predict.extend(predicted)
print(my_predict)
mysub.label=my_predict
mysub.to_csv('mysub.csv')
```