# 实验二 猫狗分类

## 一、 实验目的

1. 进一步理解和掌握卷积神经网络中卷积层、卷积步长、卷积核、池化层、池化核、微调(Fine-tune)等概念。
2. 进一步掌握使用深度学习框架进行图像分类任务的具体流程：如读取数据、构造网络、训练和测试模型等等。

## 二、 实验要求

1. 基于 Python 语言和任意一种深度学习框架（实验指导书中使用 Pytorch 框架进行介绍），从零开始一步步完成数据读取、网络构建、模型训练和模型测试等过程，最终实现一个可以进行猫狗图像分类的分类器。
2. 考虑到同学们机器性能的差异，该实验不强制要求使用 Kaggle 猫狗竞赛的原始数据集，大家可以根据自己的实际情况将原始数据集中训练集里的猫狗图像人为重新划分训练集和测试集。原则上要求人为划分的数据集中，训练集图像总数不少于 2000 张，测试集图像总数不少于大于 500，最终模型的准确率要求不低于 75%。鼓励在机器性能满足条件的情况下，使用大的数据集提高猫狗分类的准确率。
3. 按规定时间在课程网站提交实验报告、代码以及 PPT。

## 三、 实验原理

在构建图像分类网络时，主要使用到卷积层、池化层以及全连接层。在 pytorch 中，他们的定义如下：

1. Class torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)

   参数：

   in_channels(int) // 输入信号的通道

   out_channels(int) // 卷积产生的通道

   kerner_size(int or tuple) // 卷积核的尺寸

   stride(int or tuple, optional) // 卷积步长

   padding(int or tuple, optional) // 输入的每一条边补充 0 的层数

dilation(int or tuple, optional) // 卷积核元素之间的间距

groups(int, optional) // 从输入通道到输出通道的阻塞连接数

bias(bool, optional) // 如果 bias=True，添加偏置

2. Class torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False)

参数：

kernel_size(int or tuple) // max pooling 窗口大小

stride(int or tuple, optional) // max pooling 窗口移动的步长，默认值是 kernel_size

padding(int or tuple, optional) // 输入的每一条边补充 0 的层数

dilation(int or tuple, optional) //一个控制窗口中元素步幅的参数

return_indices // 如果等于 True，会返回输出最大值的序号，对于上采样操作会有帮助

ceil_mode // 如果等于 True，计算输出信号大小的时候，会使用向上取整，代替默认的向下取整的操作

3. Class torch.nn.Linear(in_features, out_features, bias=True)

参数：

in_features // 每个输入样本的大小

out_features // 每个输出样本的大小

bias // 若设置为 False，这层不会学习偏置。默认值：True

四、 实验所有数据集及工具

1. 数据集

本实验使用实验数据基于 kaggle Dogs vs. Cats 竞赛提供的官方数据集，数据集可在百度网盘中进行下载：

链接：https://pan.baidu.com/s/13hw4LK8ihR6-6-8mpjLKDA 密码：dmp4。

数据集的目录划分如下：

```
dataset
    |— train
        |— dogs
        |— cats
```

|— validation
|— dogs
|— cats

将数据集划分为训练集（training dataset）和验证集（validation dataset），均包含 dogs 和 cats 两个目录，且每个目录下包含与目录名类别相同的 RGB 图。数据集共 25000 张照片，其中训练集猫狗照片各 10000 张，验证集猫狗照片各 2500 张。（注：可根据计算资源情况自己调整训练集和验证集的大小，但最好按比例调整）

原始数据集如图 1、2、3、4 所示。

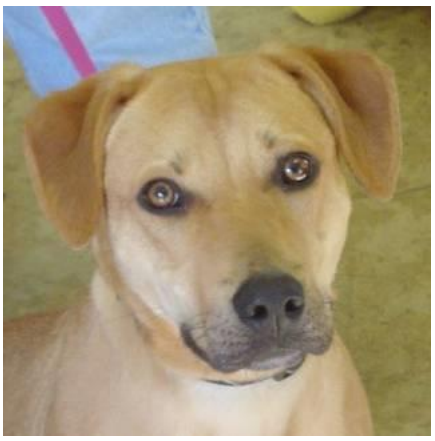

图 1

图 2

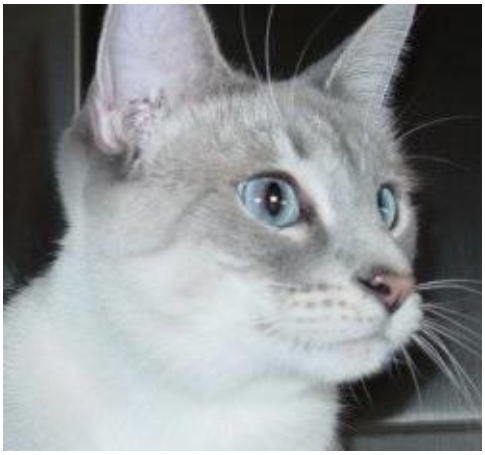

图 3

图 4

2. Pytorch 深度学习框架

PyTorch 是一个开源的 Python 机器学习库，由 Facebook 人工智能研究院

（FAIR）于 2017 年 1 月基于 Torch 推出。它是一个基于 Python 的可续计算包，提供两个高级功能：（1）具有强大的 GPU 加速的张量计算（如 NumPy）；（2）包含自动求导系统的深度神经网络。同学们可以自行查找资料进行一步一步学习。

五、实验步骤与方法

对猫狗照片识别分类的卷积神经网络模型可以自行设计，本实验指导书给出的方法是自定义简单的卷积网络结构以实现猫狗图像分类。

实验可简单地划分为数据准备、模型训练和模型验证三个步骤。

1. 数据准备

首先要将自己的数据集划分为训练集和验证集，根据资源情况可以自行调整训练集和验证集大小。

```
split_ratio = [0.6, 0.2, 0.2]
split_names = ['train', 'eval', 'test']
split_cnt = [0, 0, 0]
```

其次可以定义图像的预处理操作，在这里可以使用更多图像增强方法以提升模型的性能，根据自己的实际情况进行设置。

```
transform = transforms.Compose([
    transforms.RandomResizedCrop(150),
    transforms.ToTensor(),
    transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
])
```

然后使用 Pytorch 自带的 ImageFolder 进行数据读取，使用 DataLoader 进行数据导入。

```
dataset_train = datasets.ImageFolder(root + '/train', transform)
dataset_test = datasets.ImageFolder(root + '/val', transform)
train_loader = torch.utils.data.DataLoader(dataset_train,
batch_size=BATCH_SIZE, shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset_test,
batch_size=BATCH_SIZE, shuffle=True)
```

2. 模型构建

模型构建时，需要创建一个继承自 nn.Module 的类，在这个类_init_( ) 中定义网络结构，在 forward 中定义前向传播过程。

示例中定义了一个仅仅由两层卷积层和池化层构成的网络结构。在实验中大家可以自己定义更加复杂的网络结构以取得更好的效果。

示例：

```
class ConvNet(nn.Module):
    def _init_ (self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3)
        self.max_pool1 = nn.MaxPool2d(2)
        self.conv2 = nn.Conv2d(32, 64, 3)
        self.max_pool2 = nn.MaxPool2d(2)
        self.fc1 = nn.Linear(6272, 512)
        self.fc2 = nn.Linear(512, 1)

    def forward(self, x):
        in_size = x.size(0)
        x = self.conv1(x)
        x = F.relu(x)
        x = self.max_pool1(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.max_pool2(x)
        x = x.view(in_size, -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
```

```
        x = torch.sigmoid(x)
        return x
```

3. 模型训练

为了让模型正常训练，需要首先设置优化器

```
optimizer = optim.Adam(model.parameters(), lr=1e-4)
```

之后让模型逐条读取训练集中的数据，进行训练：

```
def train(model, train_loader, optimizer, epoch):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        optimizer.zero_grad()
        output = model(data)
        loss = F.binary_cross_entropy(output, target)
        loss.backward()
        optimizer.step()
```

4. 模型验证

模型测试过程与训练过程类似，不用进行反向传播

```
def test(model, device, test_loader):
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        for data, target in tqdm(test_loader):
            data, target = data.to(device),
target.to(device).float().reshape(50, 1)
            output = model(data)
            test_loss += F.binary_cross_entropy(output, target,
reduction='sum').item() # 将一批的损失相加
            pred = torch.tensor([[1] if num[0] >= 0.5 else [0] for num in
```

output]).to(device)