大作业-Dogs vs. Cats

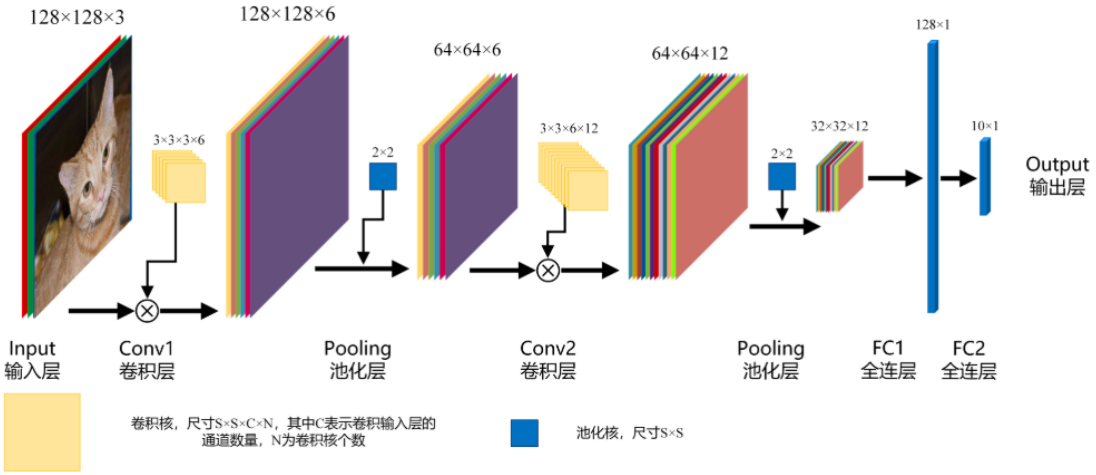
一、 选题原因

我对老师发的课程中的 CNN 网络的图像识别比较有兴趣，于是我在 github 上找到了猫狗大战（Dogs vs. Cats）的例子，猫狗大战是 Kaggle 大数据竞赛某一年的一道赛题，利用给定的数据集，用算法实现猫和狗的识别。我认为这个题目很有意思，而且比较简单。正好我学过一点 pytorch，所以我选择用 pytorch 实现猫狗图像识别。

二、 背景知识

1. CNN

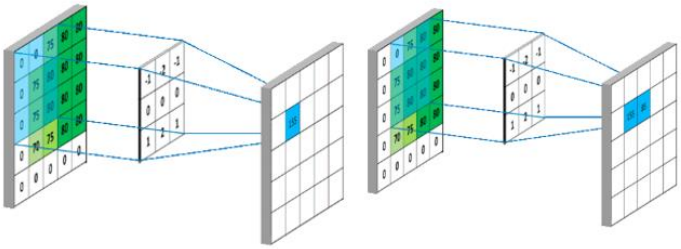
CNN(Convolutional Neural Networks)是一种神经网络，基本运算方式为卷积，因此叫卷积神经网络。一般由**输入层、卷积层、池化层、全连接层**组成，每层计算后有时还要用到激活函数。



图表 1 CNN

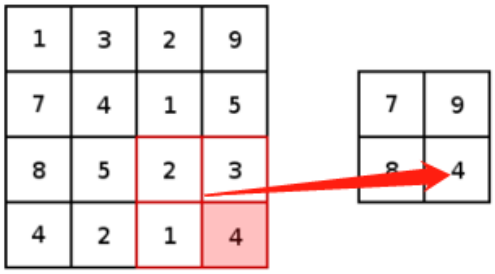
输入层(Input)：指输入的图像数据，如果 input 为 H×W×3，则代表尺寸为(H, W)的彩色图像。矩阵每个点数值范围为[0, 255]，3 表示 RGB 三个通道。

卷积层(Conv)：由卷积核对输入层图像进行卷积操作以提取图像特征。图像的卷积就是让卷积核(卷积模板)在原图像上依次滑动，在重叠的区域把对应位置的像素值和卷积模板值相乘，累加求和得到新图像(卷积结果)中的一个像素值，卷积核每滑动一次获得一个新值，当完成原图像的全部遍历，便完成原图像的一次卷积。下图示意了一次卷积计算过程。



图表 2 卷积流程

池化层(pooling)：降低图像分辨率，减少区域内图像的特征数。减少计算量，防止过拟合。用的方法有 Max pooling 和 average pooling，本次代码采用 Max pooling。



图表 3 池化

全连层(Fully connected)：将卷积图像映射至一个 n 维向量，通过设置多个全连接关系，起到从特征到分类的作用。 每个维度的值表示 CNN 结构输入图像属于该维目标的可能性。

激活函数（Activation Function）：激活函数主要引入非线性特性，在卷积过程中，所有的运算都是线性运算和线性叠加，没有激活函数，网络就无法拟合非线性特性的输入输出关系。卷积核的每一次卷积在累加模板内各个位置的乘积后，将累加值输入激活函数，然后将输出值作为卷积结果。常用的激活函数有 Sigmoid，Tanh，ReLU 等

2. Pytorch

Pytorch 由 Facebook 开源的神经网络框架，是当前难得的简洁优雅且高效快速的框架。比较简单易于上手，当前最流行的 TensorFlow 是比较不友好的，与 Python 等语言差距很大，并且在调试的时候比较复杂。所以我作为初学者选择了 pytorch。

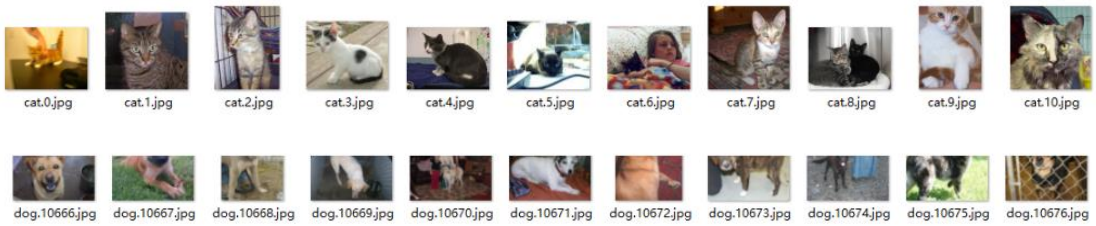
三、 思路方法

1. 数据集

1) 下载地址：https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data

2) 训练集：训练集由标记为 cat 和 dog 的猫狗图片组成，各 12500 张，总共 25000 张。图片为 24 位 jpg 格式，即 RGB 三通道图像，图片尺寸不一。

3) 测试集：测试集由 12500 张的 cat 或 dog 图片组成，未标记，图片也为 24 位 jpg 格式，RGB 三通道图像，图像尺寸不一。



图表 4 数据集图片

2. 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。把训练集中所有的数据整理成[输入， 给定输出]的形式，输入为一张张猫狗图片(input)，给定输出是对应的猫或者狗信息(label)。

3. 搭建网络：

利用 PyTorch 提供的 API 搭建设计的网络。建立 CNN 中的各个计算层，包括卷积层，池化层，全连层和激活函数等，还有载入前向计算。

4. 训练网络

把准备好的数据送入搭建的网络中进行训练，获得网络各节点权值参数

5. 测试网络

导入训练好的网络中参数，然后输入图像完成猫狗分类任务，评估网络的输出结果。

四、 代码解析（具体每行代码用处在代码截图的注释中）

1. 准备数据

建立一个数据类，将每张图片的文件名进行分割，“cat”，“dog”分别映射不同标签，将图片与标签一一对应。本代码采用了 one-hot 编码方式，猫的编码为"[1,0]"，狗的编码为"[0, 1]"，网络输出两个值，分别为属于猫和狗的概率(两者相加恒等于 1)。因为训练阶段采用交叉熵作为损失函数 Loss，输入的 label 参数是 1 维 Tensor，其值表示样本 label 中出现 1 的索引。如猫的编码为"[1,0]"，则 1 所在索引为 0，所以制作的标签为一维。将不同尺寸的图片 resize 成统一大小将图片与标签进行一一对应。训练集将 image 和 label 转为 pytorch 的形式，对于测试集我们只读取 image。

```
class DogsVSCatsDataset(data.Dataset):      # 新建一个数据集类，并且需要继承PyTorch中的data.Dataset父类
    def __init__(self, mode, dir):          # 默认构造函数，传入数据集类别（训练或测试），以及数据集路径
        self.mode = mode
        self.list_img = []                   # 新建一个image list，用于存放图片路径，注意是图片路径
        self.list_label = []                 # 新建一个label list，用于存放图片对应猫或狗的标签，其中数值0表示猫，1表示狗
        self.data_size = 0                   # 记录数据集大小
        self.transform = dataTransform       # 转换关系
        if self.mode == 'train':             # 训练集模式下，需要提取图片的路径和标签
            dir = dir + '/train/'            # 训练集路径在"dir"/train/
            for file in os.listdir(dir):     # 遍历dir文件夹
                self.list_img.append(dir + file)        # 将图片路径和文件名添加至image list
                self.data_size += 1                     # 数据集增1
                name = file.split(sep='.')              # 分割文件名，"cat.0.jpg"将分割成"cat",".","jpg"3个元素
                # label采用one-hot编码，"1,0"表示猫，"0,1"表示狗，任何情况只有一个位置为"1"，在采用CrossEntropyLoss()计算Loss情况下，label只需要输入"1"的索
                if name[0] == 'cat':
                    self.list_label.append(0)         # 图片为猫，label为0
                else:
                    self.list_label.append(1)         # 图片为狗，label为1，注意，list_img和list_label中的内容是一一配对的
        elif self.mode == 'test':            # 测试集模式下，只需要提取图片路径就行
            dir = dir + '/test/'             # 测试集路径为"dir"/test/
            for file in os.listdir(dir):
                self.list_img.append(dir + file)    # 添加图片路径至image list
                self.data_size += 1
                self.list_label.append(2)       # 添加2作为label，实际未用到，也无意义
        else:
            print('Undefined Dataset!')
```

```
    def __getitem__(self, item):             # 重载data.Dataset父类方法，获取数据集中数据内容
        if self.mode == 'train':                                        # 训练集模式下需要读取数据集的image和label
            img = Image.open(self.list_img[item])                       # 打开图片
            label = self.list_label[item]                               # 获取image对应的label
            return self.transform(img), torch.LongTensor([label])      # 将image和label转换成PyTorch形式并返回
        elif self.mode == 'test':                                       # 测试集只需读取image
            img = Image.open(self.list_img[item])
            return self.transform(img)                                  # 只返回image
        else:
            print('None')
```

2. 搭建网络

网络继承 PyTorch 的 nn.Module 父类，在构造函数中新建 CNN 中的计算层，包括卷积层，池化层，全连层和激活函数等。还要重载父类的网络运算中的前向计算 forward()方法。该网络包含两个卷积层、三个全连接层，经过两次池化。第一次卷积结果经过 ReLU 激活函数处理，并且池化大小 2×2，方式 Max pooling，第二次卷积经过 ReLU 激活函数处理，池化大小 2×2，方式 Max pooling。然后将结果输入全连层，因为全连接层输入需要一维张量，因此需要对输入的格式数据排列成一维形式，再经过三次全连接层和两次 ReLU 激活。具体网络搭建如下：

1) Input：图像尺寸为 200×200 像素，由于训练集和测试集中的图片大小尺寸多样，因此在送入网络前，须将图片调整至 200×200 像素

2) conv1：卷积核的规模为[3×3×3×16]，size 大小 3×3，深度 3，数量 16

第一次卷积结果：16 个卷积图像(feature map)，200×200 像素

3) Pooling：第一次池化，size 大小 2×2，Max pooling
第一次池化结果：图像缩小为 100×100 像素
4) conv2：卷积核的规模为[3×3×16×16]，size 大小 3×3，深度 16，数量 16
第二次卷积结果：16 个卷积图像(feature map)，100×100 像素
5) Pooling：第二次池化，size 大小 2×2，Max pooling
第二次池化结果：图像缩小为 50×50 像素
6) FC1：第一次全连接，输入节点数为 50×50×16= 40000，输出节点数为 128，输出数据为[128×1]
7) FC2：第二次全连接，输入节点数为 128，输出节点数为 64，输出数据为[64×1]
8) FC3：第三次全连接，输入节点数为 64，输出节点数为 2，即两个数值，分别表示猫和狗的概率(后面要通过 softmax 方法进行转换将数值转换为概率形式)。

```
class Net(nn.Module):                                   # 新建一个网络类，就是需要搭建的网络，必须继承PyTorch的nn.Module父类
    def __init__(self):                                 # 构造函数，用于设定网络层
        super(Net, self).__init__()                     # 标准语句
        self.conv1 = torch.nn.Conv2d(3, 16, 3, padding=1)   # 第一个卷积层，输入通道数3，输出通道数16，卷积核大小3×3，padding大小1，其他参数默认
        self.conv2 = torch.nn.Conv2d(16, 16, 3, padding=1)  # 第二个卷积层，输入通道数16，输出通道数16，卷积核大小3×3，padding大小1，其他参数默认

        self.fc1 = nn.Linear(50*50*16, 128)             # 第一个全连层，线性连接，输入节点数50×50×16，输出节点数128
        self.fc2 = nn.Linear(128, 64)                   # 第二个全连层，线性连接，输入节点数128，输出节点数64
        self.fc3 = nn.Linear(64, 2)                     # 第三个全连层，线性连接，输入节点数64，输出节点数2

    def forward(self, x):                   # 重写父类forward方法，即前向计算，通过该方法获取网络输入数据后的输出值
        x = self.conv1(x)                   # 第一次卷积
        x = F.relu(x)                       # 第一次卷积结果经过ReLU激活函数处理
        x = F.max_pool2d(x, 2)              # 第一次池化，池化大小2×2，方式Max pooling

        x = self.conv2(x)                   # 第二次卷积
        x = F.relu(x)                       # 第二次卷积结果经过ReLU激活函数处理
        x = F.max_pool2d(x, 2)              # 第二次池化，池化大小2×2，方式Max pooling

        x = x.view(x.size()[0], -1)         # 由于全连层输入的是一维张量，因此需要对输入的[50×50×16]格式数据排列成[40000×1]形式
        x = F.relu(self.fc1(x))             # 第一次全连，ReLU激活
        x = F.relu(self.fc2(x))             # 第二次全连，ReLU激活
        y = self.fc3(x)                     # 第三次激活，ReLU激活

        return y
```

3. 训练网络

Dogs vs. Cats 训练集中共有 25000 个图片， epoch 为 10，即全部数据做了 10 次训练。batch size 为 16。那么网络参数总共调整的参数次数即 iteration 为 25000/16= 1,562 次。DataLoader()的作用是对定义好的数据集类做一次封装，可以完成打乱数据集分布、设定多线程数据读取、一次获得 batch size 大小的数据。原来代码采用 GPU 进行运算，如果没有 GPU 做计算的话，可以去掉代码中的.cuda()。

采用训练模式，选定好 adm 优化器和交叉熵计算 loss，开始训练每次住区 16 组数据，并放置在 Variable 中，然后计算网络输出值并于 label 计算误差 loss，并将误差反向传播，再用优化器对节点各个参数进行优化，最后清楚优化器中梯度以便下次运算。将全部数据集训练十次，最终将训练好的网络保存成 pkl 文件。

概念讲解：

Epoch：表示整个数据集进了多少次重复训练。训练时，将所有数据迭代训练一次是不够的，需要反复多次才能拟合收敛。

batch size：一次训练所选取的样本数。为了在内存效率和内存容量之间寻求最佳平衡，batch size 应该精心设置，从而最优化网络模型的性能及速度。

iteration：一个 epoch 中网络参数调整(迭代)的次数。

```python
def train():
    datafile = DVCD('train', dataset_dir)                                                    # 实例化一个数据集
    dataloader = DataLoader(datafile, batch_size=batch_size, shuffle=True, num_workers=workers, drop_last=True)     # 用PyTorch的DataLoader类封装
    print('Dataset loaded! length of train set is {0}'.format(len(datafile)))
    model = Net()                       # 实例化一个网络
    # model = model.cuda()                # 网络送入GPU，即采用GPU计算，如果没有GPU加速，可以去掉".cuda()"
    model = nn.DataParallel(model)
    model.train()                       # 网络设定为训练模式，有两种模式可选，.train()和.eval()，训练模式和评估模式，区别就是训练模式采用了dropout策略，可
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)         # 实例化一个优化器，即调整网络参数，优化方式为adam方法
    criterion = torch.nn.CrossEntropyLoss()                         # 定义loss计算方法，cross entropy，交叉熵，可以理解为两者数值越接近其值越小
    cnt = 0             # 训练图片数量
    for epoch in range(nepoch):
        # 读取数据集中数据进行训练，因为dataloader的batch_size设置为16，所以每次读取的数据量为16，即img包含了16个图像，label有16个
        for img, label in dataloader:                                       # 循环读取封装后的数据集，其实就是调用了数据集中的 __getitem__()方法，
            # img, label = Variable(img).cuda(), Variable(label)#.cuda()
            img, label = Variable(img), Variable(label)                     # 将数据放置在PyTorch的Variable节点中，并送入GPU中作为网络计算起点
            out = model(img)                                                # 计算网络输出值，就是输入网络一个图像数据，输出猫和狗的概率，调用了网络
            loss = criterion(out, label.squeeze())      # 计算损失，也就是网络输出值和实际label的差异，显然差异越小说明网络拟合效果越好，此处需要注意的是
            loss.backward()                             # 误差反向传播，采用求导的方式，计算网络中每个节点参数的梯度，显然梯度越大说明参数设置不合理，需要
            optimizer.step()                            # 优化采用设定的优化方法对网络中的各个参数进行调整
            optimizer.zero_grad()                       # 清除优化器中的梯度以便下一次计算，因为优化器默认会保留，不清除的话，每次计算梯度都回累加
            cnt += 1

            print('Epoch:{0},Frame:{1}, train_loss {2}'.format(epoch, cnt*batch_size, loss/batch_size))          # 打印一个batch_size的训练结果

    torch.save(model.state_dict(), '{0}/model.pth'.format(model_cp))            # 训练所有数据后，保存网络的参数
```

4. 测试网络

将训练好的网络参数导入新定义的网络中，然后输入图像完成猫狗分类任务。并验证结果的准确性

```python
def test():
    model = Net()                                   # 实例化一个网络
    model = nn.DataParallel(model)
    model.load_state_dict(torch.load(model_file))       # 加载训练好的模型参数
    model.eval()                                    # 设定为评估模式，即计算过程中不要dropout
    # get data
    files = random.sample(os.listdir(dataset_dir), N)   # 随机获取N个测试图像
    imgs = []           # img
    imgs_data = []      # img data
    for file in files:
        img = Image.open(dataset_dir + file)            # 打开图像
        img_data = getdata.dataTransform(img)           # 转换成torch tensor数据

        imgs.append(img)                                # 图像list
        imgs_data.append(img_data)                      # tensor list
    imgs_data = torch.stack(imgs_data)                  # tensor list合成一个4D tensor
    # calculation
    out = model(imgs_data)                              # 对每个图像进行网络计算
    out = F.softmax(out, dim=1)                         # 输出概率化
    out = out.data.cpu().numpy()                        # 转成numpy数据
    # pring results         显示结果
    for idx in range(N):
        plt.figure()
        if out[idx, 0] > out[idx, 1]:
            plt.suptitle('cat:{:.1%},dog:{:.1%}'.format(out[idx, 0], out[idx, 1]))
        else:
            plt.suptitle('dog:{:.1%},cat:{:.1%}'.format(out[idx, 1], out[idx, 0]))
        plt.imshow(imgs[idx])
    plt.show()
```

五、 结果展示

1. 训练结果（loss）

看一看出来损失值 loss 下降了很多。

```
Epoch:0,Frame:16, train_loss 0.043813023716211132
Epoch:0,Frame:32, train_loss 0.043063025921583176
Epoch:0,Frame:48, train_loss 0.04457520693540573
Epoch:0,Frame:64, train_loss 0.04382330924272537
Epoch:0,Frame:80, train_loss 0.042920563369989395
```

```
Epoch:9,Frame:249856, train_loss 0.02563639171421528
Epoch:9,Frame:249872, train_loss 0.008000656962394714
Epoch:9,Frame:249888, train_loss 0.013245390728116035
Epoch:9,Frame:249904, train_loss 0.013621248304843903
Epoch:9,Frame:249920, train_loss 0.012428155168890953

Process finished with exit code 0
```

2. 测试结果

可以看出测试结果准确性很高，训练的网络效果很好。