姓名：管博宇
学号：202218013229060

中国科学院大学 NLP 课程

第一次作业

实验的主要代码我已经上传至我的 GitHub 仓库中。我采用的是 PyTorch 框架。

1 数据集处理

我去kaggle 网站上下载了所有的数据集，猫狗各 12500 张照片。为了便于训练与分类，我将训练集的每张照片按照类别分别装在子文件夹中。然后随机抽取其中的 20% 作为验证集进行训练。

因为数据集的照片大小比例并不一致，有些照片的长宽差距很大，直接 resize 成正方形的话照片失真会比较严重。所以以我对数据集进行进一步的剪裁，将所有照片全部在中心点裁切处理成正方形。新的正方形照片的边长等于原照片中长宽里的较小值。

接下来，对于训练集中的每一个图像我均作数据加强。包括调整到 (224, 224) 大小的正方形照片、随机水平翻转、随机图片色彩信息（饱和度、色相、曝光等）、Normalize 等。

所有图像处理的程序如下所示。

```
def splitTrainData(trainDataPath:str ):
    """将原始数据集按类别分割到不同文件夹

    Args:
        trainDataPath (str): 原始数据集路径
    """
    dataPath = './data'
    os.makedirs(os.path.join(dataPath, 'cat'), exist_ok=True)
    os.makedirs(os.path.join(dataPath, 'dog'), exist_ok=True)

    catCount, dogCount = 0, 0
    for imageName in os.listdir(trainDataPath):
        if imageName.startswith('cat'):
            catCount += 1
            os.rename(os.path.join(trainDataPath, imageName), os.path.join(dataPath, 'cat', f'cat_{catCount}.jpg'))
        elif imageName.startswith('dog'):
            dogCount += 1
            os.rename(os.path.join(trainDataPath, imageName), os.path.join(dataPath, 'dog', f'dog_{dogCount}.jpg'))

def makeDatasetToSquqre():
    """将原始数据集转换为方形
```

```
    """
    os.makedirs(os.path.join('./data1', 'cat'), exist_ok=True)
    os.makedirs(os.path.join('./data1', 'dog'), exist_ok=True)
    classNames = ['cat', 'dog']
    for className in classNames:
        for imageName in os.listdir(os.path.join('./data', className)):
            image = Image.open(os.path.join('./data', className, imageName))
            width, height = image.size
            newLen = min(width, height)
            image = image.crop((  int((width-newLen)/2), int((height-newLen)/2), int((width-
    newLen)/2) + newLen,  int((height-newLen)/2) + newLen))
            image.save(os.path.join('./data1', className, imageName))

    os.makedirs('./test', exist_ok=True)
    for imageName in os.listdir('./test1'):
        image = Image.open(os.path.join('./test1', imageName))
        width, height = image.size
        newLen = min(width, height)
        image = image.crop((  int((width-newLen)/2), int((height-newLen)/2), int((width-newLen
    )/2) + newLen,  int((height-newLen)/2) + newLen))
        image.save(os.path.join('./test', imageName))


def prepaingData(rootpath='./data', batchSize=32):
    """返回dataloader

    Args:
        rootpath (str, optional): 数据集地址. Defaults to './data'.
        batchSize (int, optional): 批大小. Defaults to 32.

    Returns:
        trainloader, testloader, classe: 训练和测试的dataloader以及类别
    """
    transform_train = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.RandomHorizontalFlip(),
        torchvision.transforms.ColorJitter(brightness=0.5, contrast=0.2, saturation=0.2, hue
    =0.2),
        transforms.ToTensor(),
        transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
    ])

    transform_test = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)),
    ])
    allDataset = torchvision.datasets.ImageFolder(root=rootpath)
```

```
    trainsetLen = int(len(allDataset) * 0.8 )
    trainset, testset = torch.utils.data.random_split(allDataset, [trainsetLen, len(allDataset) - trainsetLen])
    trainset.dataset.transform = transform_train
    testset.dataset.transform = transform_test
    trainloader = torch.utils.data.DataLoader(trainset,batch_size=batchSize, shuffle=True, num_workers=2)
    testloader = torch.utils.data.DataLoader(testset, batch_size=batchSize, shuffle=False, num_workers=2)

    classes = ('cat', 'dog')
    return trainloader, testloader, classes
```

Listing 1.1: 数据集处理程序

2 DNN 分类

2.1 网络设计

```
class DNN(nn.Module):
    def __init__(self):
        super(DNN, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(3*224*224, 1024),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 2)
        )
    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

Listing 2.1: DNN 模型设计

在模型中，我在第一步将其展开成一个有着 $3 \times 224 \times 224$ 个输入节点的线性输入，然后经过 DNN 网络。DNN 网络中采用 ReLU 作为激活函数，还用了 dropout 函数随机屏蔽一些神经元防止过拟合。最终用两个神经元作为输出，分别表示图片是两个类的概率。

2.2 模型训练

学习率选择为 10^{-4}，momentum 为 0.9，权重衰减为 5×10^{-4}。并训练 300 个 epoch。选择交叉熵函数做为损失函数，并在训练过程中保存验证集准确率最高的 epoch 对应的模型参数进行保存。

2.3 训练结果

对于验证集，最高的准确率为 68%，相较于随机分类的 50% 准确率提升有限。

3 CNN 分类

3.1 网络设计

```python
class DNN(nn.Module):
    def __init__(self):
        super(DNN, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(3*224*224, 1024),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.5),
            nn.Linear(512, 2)
        )
    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

Listing 3.1: DNN 模型设计

我采用 *ResNet*50 模型 [1]，其架构如图3.1所示。

3.2 模型训练

学习率选择为 0.005，momentum 为 0.9，权重衰减为 5×10^{-4}。并训练 300 个 epoch。选择交叉熵函数做为损失函数，并在训练过程中保存验证集准确率最高的 epoch 对应的模型参数进行保存。

3.3 训练结果

对于验证集，最高的准确率为 94%。相较于有着 DNN 模型大幅度提高，且模型收敛性好。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3.1: ResNet 架构

4 RNN 分类

4.1 网络设计

经过调研发现，RNN 模型进行图像分类任务主要有如图4.1和图4.2所示的两种结构。图4.1所示的方式与 DNN 模型完全一致，故没有重复实验的必要，所以我们选择第二种分类方式。

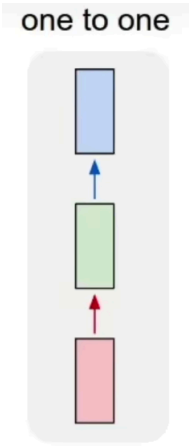

Figure 4.1: 第一种 RNN 模型

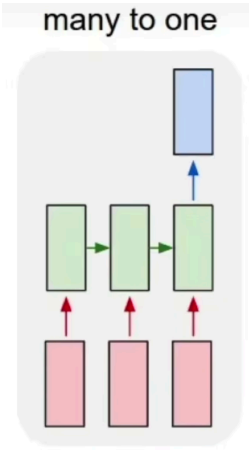

Figure 4.2: 第二种 RNN 模型

我们每张照片预处理完后的纬度为 $(3, 224, 224)$，但对于 RNN 模型的分类任务来说 $3 \times 224 \times 224$ 长的步数有些过长，容易出现梯度消失或者梯度爆炸的问题，我们将原始图片中每一列像素的 RGB 三个分量合在一起形成一个 3×224 长的向量作为每一步的输入。在每次输入完一整张相片之后只考虑最后一个输出，将最后一个输出放入一个全连接层得到最终分类结果。表4.1展示了 RNN 模型的搭建代码与正向传播过程。

```
class RNN(nn.Module):
    def __init__(self, in_feature=3*224, hidden_feature=1000, num_class=2, num_layers=2):
        super(RNN, self).__init__()
```

```
        self.rnn = nn.GRU(in_feature, hidden_feature, num_layers) # 使用两层 GRU
        self.classifier = nn.Linear(hidden_feature, num_class)

    def forward(self, x):
        x = x.permute(0,2,3,1) # 此步与下一步是为了将每一列3个通道的像素值放在一起
        x = x.reshape(x.shape[0], x.shape[1], -1)
        x = x.permute(1, 0, 2) # 将最后一维放到第一维，变成每次输入的维度为3*224，输入224次
        out, _ = self.rnn(x)
        out = out[-1] # 取序列中的最后一个输出
        out = self.classifier(out) # 将其导入全连接层分类
        return out
```

Listing 4.1: RNN 模型代码

4.2 模型训练

学习率选择为 0.0001，momentum 为 0.9，权重衰减为 5×10^{-4}。并训练 300 个 epoch。选择交叉熵函数做为损失函数，并在训练过程中保存验证集准确率最高的 epoch 对应的模型参数进行保存。

4.3 训练结果

对于验证集，最高准确率仅有 63%，为所有模型中的最低。其中的原因可能为：

1. RNN 模型本身不适合做图片分类任务.
2. 224 步的输入还是有些过长，发生了梯度爆炸、梯度消失等问题。
3. 按照 GRU 模型的输入很没有道理，一张照片按列从左到右输入到模型中，各个列的初始权重应该是一样重要的，而 RNN 模型的特点就决定了图片靠近右侧的列要比图片靠近左侧的列有更大影响。

5 一些发现

1. 在一开始训练时，我直接用 transom 中的 resize 函数将其放缩成一个正方形。后来研究图片时发现个别图片的长宽比例与正方形差距较大，比如长宽比大于 3 或者小于 0.3。对于这种照片，倘若对其直接 resize 成正方形图片的话图片失真会比较严重，连人眼都难以辨别。所以我采取了长宽中较小的数字作为正方形的边长，然后将图片在中间按照边长裁剪成新的正方形图片。无论是训练集还是测试集都用新的图片进行验证。
2. DNN 的模型参数量大、容易过拟合，而且训练的学习率也要小一些。
3. 图片预处理的正则化参数很重要，我选择了之前训练 cifar 数据集 [2] 使用的正则化参数，这个正则化参数对于训练有一定影响。因为主要是学习深度学习框架的基础实用，所以我并未进行进一步的调参。

4. 用正确的模型干正确的事儿。对于图像分类问题，卷积神经网络不仅能很好的表示图像的空间关系、提取分类维度特征，而且运算量小、收敛快。而 DNN 模型与 RNN 模型则因为很难表示出矩阵空间特征，而且运用全连接层，导致效果差、收敛慢。

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.