实验二 猫狗识别实验报告

202228013329025 张喜玥

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 MindSpore 或者任意一种深度学习框架(该部分实验优先推荐使用Mindspore)，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. 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。

三、实验数据集及环境

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。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。原始数据集共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。

本实验所用数据集数量：训练集猫狗图像共4000张，测试集图像总数1000张

2.实验框架：`Python 3.8`,`Pytorch 2.0.0`

3.实验环境：Macbook Air M2，使用GPU加速

四、网络架构及实验参数

网络架构

本实验采用ResNet18进行猫狗图像分类，并在该网络上进行适当的修改，因为是二分类网络，所以将最后一层全连接层输出改成2，并在全连接层前面添加了Dropout层防止过拟合

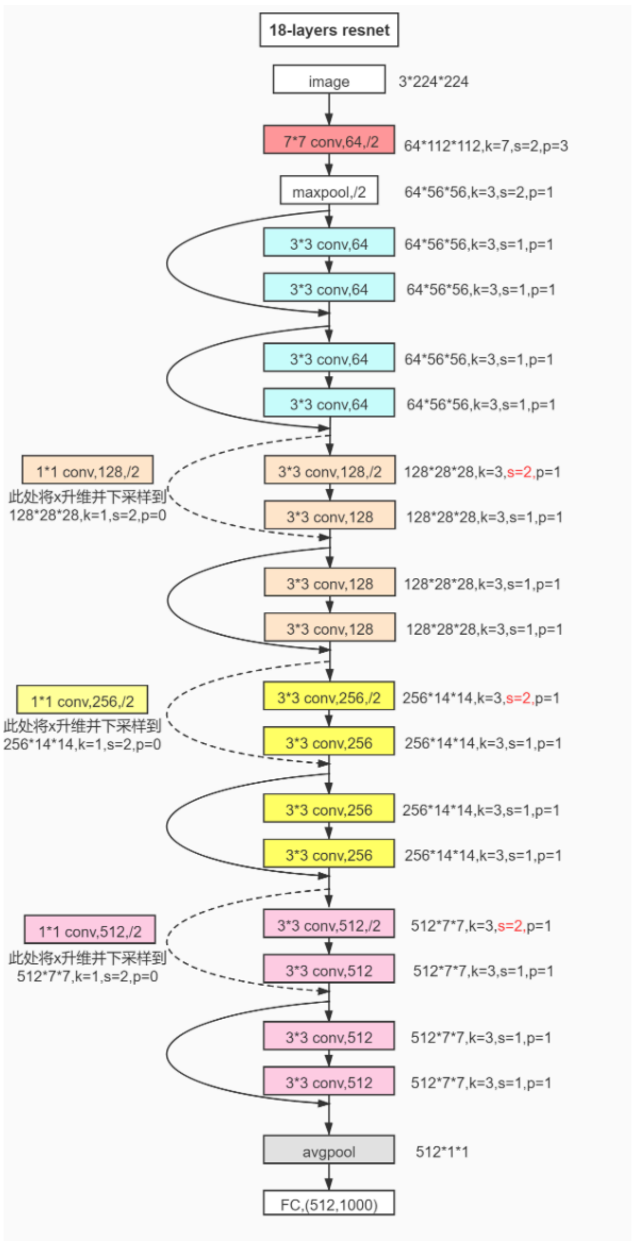


图1 网络架构

实验参数

`batch_size=20`，`epoch=40`。

初始学习率`learning_rate=0.001`。

采用`Adam`优化器和交叉熵损失函数`CrossEntropyLoss`。

五、代码说明

1.数据集预处理

（1）首先加载数据集：训练集共4000张图像，测试集共1000张图像

```
train_dir = os.path.join(root_path, 'train')
test_dir = os.path.join(root_path, 'test')

train_imgs = [os.path.join(train_dir, img) for img in os.listdir(train_dir)]
test_imgs = [os.path.join(test_dir, img) for img in os.listdir(test_dir)]
```

（2）对图像进行处理：`CatDog_Dataset`类用于生成`DataLoader`中的`Dataset`对象，根据图像文件名将猫标记为0，狗标记为1。

```
class CatDog_Dataset(Dataset):
    def __init__(self, imgs, transform=None):
        self.imgs = imgs
        self.len = len(imgs)
        self.transform = transform

    def __getitem__(self, index):
        imgs = np.random.permutation(self.imgs)
        img_path = self.imgs[index]
        data = Image.open(img_path)
        data = self.transform(data)
        label = 0 if 'cat' in img_path.split('/')[-1] else 1
        return data, label

    def __len__(self):
        return len(self.imgs)
```

（3）将图像转化成224*224大小，进行归一化处理，并对训练集图像做数据增强，从而减少过拟合。

```
normalize = transforms.Normalize(mean=[0.485,0.456,0.406],
                                 std=[0.229,0.224,0.225])
train_transform = transforms.Compose([transforms.Resize((224, 224)),
                                      transforms.RandomHorizontalFlip(),
                                      transforms.RandomRotation(20),
                                      transforms.ToTensor(),
                                      normalize])
test_transform = transforms.Compose([transforms.Resize((224, 224)),
                                     transforms.ToTensor(),
                                     normalize])

train_data = CatDog_Dataset(train_imgs, train_transform)
test_data = CatDog_Dataset(test_imgs, test_transform)

train_loader = DataLoader(train_data, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_data, batch_size=batch_size, shuffle=True)
```

3.构建网络模型，使用ResNet18网络结构

```python
import torch
from torch import nn
from torch.nn import functional as F


class ResidualBlock(nn.Module):
    """
    残差块
    """
    def __init__(self, in_channel, out_channel, stride=1, extra=None):
        super(ResidualBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channel, out_channel, kernel_size=3,
stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channel)
        self.relu = nn.ReLU()

        self.conv2 = nn.Conv2d(out_channel, out_channel, kernel_size=3, stride=1,
padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channel)

        self.extra = extra

    def forward(self, x):
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        residual = x #虚线输出值
        if self.extra is not None:
            residual = self.extra(x)
        out += residual
        out = F.relu(out)
        return out


class ResNet18(nn.Module):
    """
    ResNet18网络
    _make_layer函数实现残差块的重复
    """

    def __init__(self):
        super(ResNet18, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3, bias=False), #7x7,64,stride 2
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True)) #output 112x112x64

        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1) #output 56x56x64

        # 重复的layer，分别有2个residual block
        self.layer1 = self._make_layer(64, 64, 2) #conv2_x 56x56x64
        self.layer2 = self._make_layer(64, 128, 2, stride=2) #conv3_x
        self.layer3 = self._make_layer(128, 256, 2, stride=2) #conv4_x
        self.layer4 = self._make_layer(256, 512, 2, stride=2) #conv5_x

        self.avgpool = nn.AvgPool2d(7, stride=1)
        self.fc = nn.Linear(512, 2)


    def _make_layer(self, in_channel, out_channel, block_num, stride=1):
        extra = None
        if stride != 1 :
            extra = nn.Sequential( #右边分支1x1卷积层
                nn.Conv2d(in_channel, out_channel, kernel_size=1, stride=stride, bias=False),
                nn.BatchNorm2d(out_channel))

        layers = []
        layers.append(ResidualBlock(in_channel, out_channel, stride, extra))

        for _ in range(1, block_num):
            layers.append(ResidualBlock(out_channel, out_channel))
        return nn.Sequential(*layers)

    def forward(self, x):
        x = self.conv1(x)
        x = self.maxpool(x)

        x = self.layer1(x)
        x = self.layer2(x)
        x = self.layer3(x)
        x = self.layer4(x)

        x = self.avgpool(x)
        x = x.view(x.size(0), -1)
        x = F.dropout(x)
        x = self.fc(x)
        return x
```

4.设置损失函数、优化器等相关参数

```python
batch_size = 20
epochs = 40
device = torch.device("mps")
learning_rate = 0.001

model = ResNet18().to(device)
criterion = nn.CrossEntropyLoss().to(device)  # 设置损失函数为交叉熵函数
optimizer = optim.Adam(model.parameters(), lr=learning_rate)  # 设置优化器
```

5.训练模型

```python
# 训练模型
for epoch in range(epochs):
  model.train()
  train_correct = 0
  test_correct = 0
  batch_num = 0
  train_loss = 0
  test_loss = 0
  for i,(input, label) in enumerate(train_loader):
    batch_num += 1
    input, label = input.to(device), label.to(device)
    output = model(input)
    loss = criterion(output, label)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    pred = torch.max(output, 1)[1].cpu().numpy()
    label = label.cpu().numpy()
    correct = (pred == label).sum()
    train_correct += correct
    train_loss += loss.item()
  train_loss_list.append(train_loss/batch_size)
```

6.测试模型

```python
  model.eval()
  with torch.no_grad():
    for ii, (input, target) in enumerate(test_loader):
      input = input.to(device)
      target = target.to(device)
      output = model(input)
      loss = criterion(output, target)

      pred = torch.max(output, 1)[1].cpu().numpy()
      target = target.cpu().numpy()
      correct = (pred == target).sum()
      test_correct += correct
      test_loss += loss.item()

    test_loss_list.append(test_loss/batch_size)
    train_acc = train_correct / len(train_data)
    test_acc = test_correct / len(test_data)
    train_acc_list.append(train_acc)
    test_acc_list.append(test_acc)
    print("Epoch:", epoch+1, "train_loss:{:.5f}, test_loss:{:.5f}, train_acc:{:.2f}%, test_acc:{:.2f}%".format(train_loss/(i+1), test_loss/(ii+1), train_acc*100, test_acc*100))
```

6.绘制评估曲线

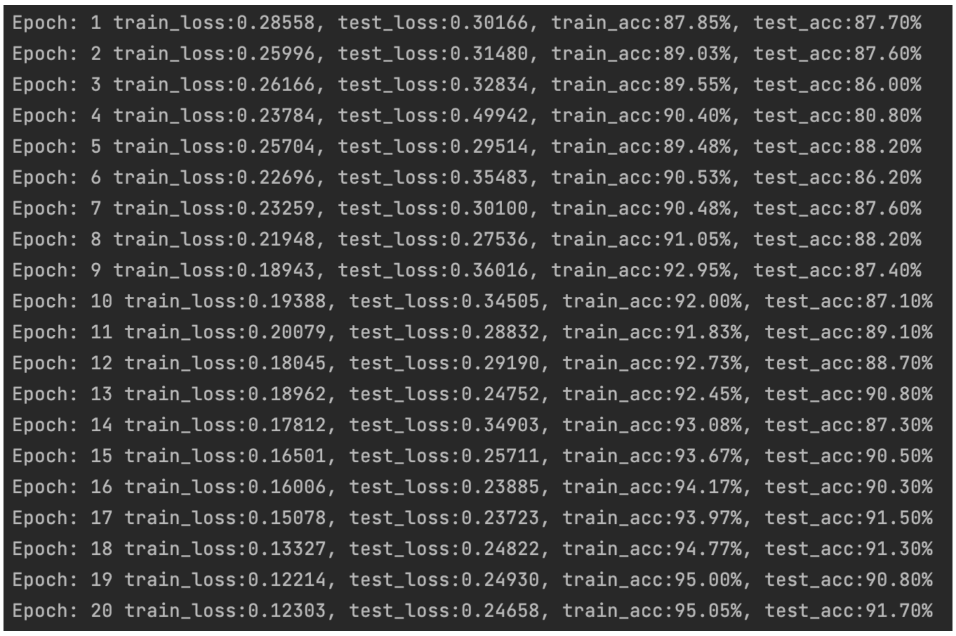
```python
def matplot_loss(train_loss, test_loss):
    plt.plot(train_loss, label="train_loss")
    plt.plot(test_loss, label="val_loss")
    plt.legend(loc='best')
    plt.ylabel('loss')
    plt.xlabel("epoch")
    plt.title("loss")
    plt.show()

def matplot_acc(train_acc, test_acc):
    plt.plot(train_acc, label="train_acc")
    plt.plot(test_acc, label="val_acc")
    plt.legend(loc='best')
    plt.ylabel('acc')
    plt.xlabel("epoch")
    plt.title("acc")
    plt.show()
```

六、实验结果

本实验在训练40个epoch后对测试集的准确率达到86.20%，继续训练20个epoch，测试集准确率可以达到91%左右。实验结果如下图所示

```
Epoch: 22 train_loss:0.52424, test_loss:0.54491, train_acc:74.
25%, test_acc:73.90%
Epoch: 23 train_loss:0.51574, test_loss:0.52139, train_acc:75.
22%, test_acc:74.20%
Epoch: 24 train_loss:0.50860, test_loss:0.58534, train_acc:75.
00%, test_acc:72.20%
Epoch: 25 train_loss:0.50256, test_loss:0.53571, train_acc:75.
75%, test_acc:72.60%
Epoch: 26 train_loss:0.48196, test_loss:0.54576, train_acc:77.
25%, test_acc:74.80%
Epoch: 27 train_loss:0.47237, test_loss:0.47771, train_acc:78.
05%, test_acc:78.00%
Epoch: 28 train_loss:0.45146, test_loss:0.48287, train_acc:79.
62%, test_acc:76.80%
Epoch: 29 train_loss:0.43544, test_loss:0.49717, train_acc:80.
15%, test_acc:77.60%
Epoch: 30 train_loss:0.42876, test_loss:0.45312, train_acc:80.
42%, test_acc:77.60%
Epoch: 31 train_loss:0.40768, test_loss:0.43113, train_acc:81.
95%, test_acc:82.00%
Epoch: 32 train_loss:0.40855, test_loss:0.41267, train_acc:81.
77%, test_acc:81.70%
Epoch: 33 train_loss:0.38823, test_loss:0.37210, train_acc:82.
58%, test_acc:84.90%
Epoch: 34 train_loss:0.38238, test_loss:0.38678, train_acc:83.
45%, test_acc:82.90%
Epoch: 35 train_loss:0.37456, test_loss:0.37627, train_acc:83.
67%, test_acc:83.00%
Epoch: 36 train_loss:0.34722, test_loss:0.38412, train_acc:84.
78%, test_acc:82.80%
Epoch: 37 train_loss:0.33811, test_loss:0.42833, train_acc:85.
32%, test_acc:82.40%
Epoch: 38 train_loss:0.33056, test_loss:0.38047, train_acc:85.
50%, test_acc:82.70%
Epoch: 39 train_loss:0.30558, test_loss:0.32572, train_acc:86.
58%, test_acc:86.50%
Epoch: 40 train_loss:0.29383, test_loss:0.33168, train_acc:87.
48%, test_acc:86.20%
```



```
Epoch: 1 train_loss:0.28558, test_loss:0.30166, train_acc:87.85%, test_acc:87.70%
Epoch: 2 train_loss:0.25996, test_loss:0.31480, train_acc:89.03%, test_acc:87.60%
Epoch: 3 train_loss:0.26166, test_loss:0.32834, train_acc:89.55%, test_acc:86.00%
Epoch: 4 train_loss:0.23784, test_loss:0.49942, train_acc:90.40%, test_acc:80.80%
Epoch: 5 train_loss:0.25704, test_loss:0.29514, train_acc:89.48%, test_acc:88.20%
Epoch: 6 train_loss:0.22696, test_loss:0.35483, train_acc:90.53%, test_acc:86.20%
Epoch: 7 train_loss:0.23259, test_loss:0.30100, train_acc:90.48%, test_acc:87.60%
Epoch: 8 train_loss:0.21948, test_loss:0.27536, train_acc:91.05%, test_acc:88.20%
Epoch: 9 train_loss:0.18943, test_loss:0.36016, train_acc:92.95%, test_acc:87.40%
Epoch: 10 train_loss:0.19388, test_loss:0.34505, train_acc:92.00%, test_acc:87.10%
Epoch: 11 train_loss:0.20079, test_loss:0.28832, train_acc:91.83%, test_acc:89.10%
Epoch: 12 train_loss:0.18045, test_loss:0.29190, train_acc:92.73%, test_acc:88.70%
Epoch: 13 train_loss:0.18962, test_loss:0.24752, train_acc:92.45%, test_acc:90.80%
Epoch: 14 train_loss:0.17812, test_loss:0.34903, train_acc:93.08%, test_acc:87.30%
Epoch: 15 train_loss:0.16501, test_loss:0.25711, train_acc:93.67%, test_acc:90.50%
Epoch: 16 train_loss:0.16006, test_loss:0.23885, train_acc:94.17%, test_acc:90.30%
Epoch: 17 train_loss:0.15078, test_loss:0.23723, train_acc:93.97%, test_acc:91.50%
Epoch: 18 train_loss:0.13327, test_loss:0.24822, train_acc:94.77%, test_acc:91.30%
Epoch: 19 train_loss:0.12214, test_loss:0.24930, train_acc:95.00%, test_acc:90.80%
Epoch: 20 train_loss:0.12303, test_loss:0.24658, train_acc:95.05%, test_acc:91.70%
```

图2 实验结果

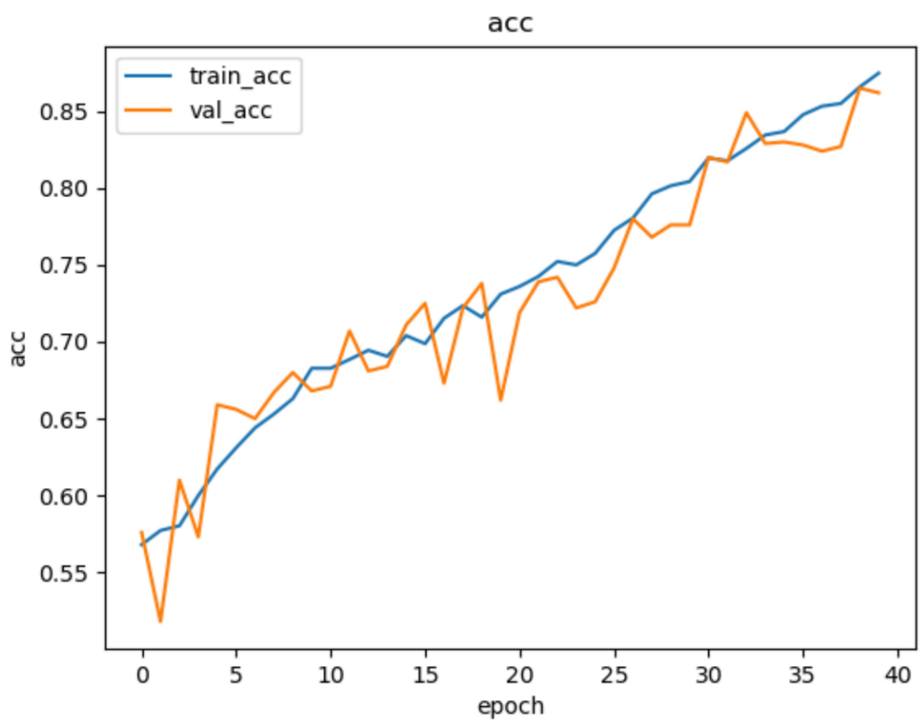


图3 准确率曲线图

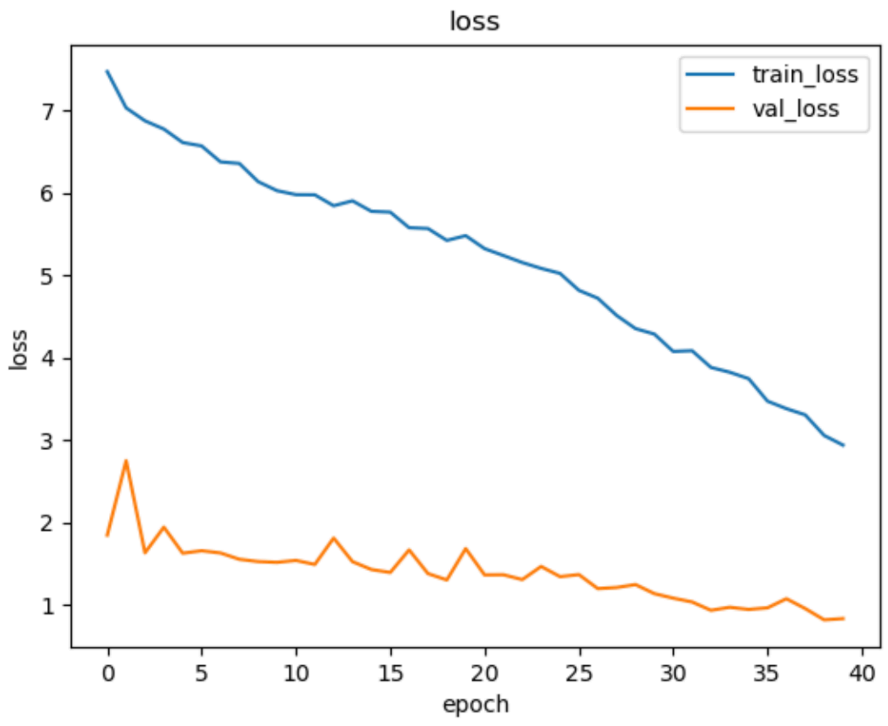


图4 损失率曲线图