AI-A-2023 Fall PJ1-Part2 report

柳世纯 20307130111

1. 实验原理

卷积神经网络的本质是局部参数共享.

上个实验中用到的全连接网络存在参数爆炸的问题; 同时, 将图片展开为一维向量很容易丢失空间信息. 由此, 使用CNN可以解决此问题.

CNN引入了若干新构件:

- 卷积层: 使用卷积核对于图像完成特征提取的操作; 不同层次的卷积核完成不同抽象等级的特征表示, 完成最终的分类操作;
- ReLU激活函数: 计算简单快捷, 增强神经元的稀疏性避免过拟合, 训练稳定收敛快;
- 池化层: 对输入数据进行滤波和降采样, 减少数据规模, 控制过拟合问题, 提高网络泛化能力; 本次实验用到的是最大池化;

CNN的设计天然适合于处理图像信息这种局部相关性强的数据类型. 一个卷积核与一种图像特征对应.

2. 实验过程

2.1 数据处理

数据增强

使用`torchvision.transforms`组件对于原始图片数据进行增强. 使用平移和旋转操作.

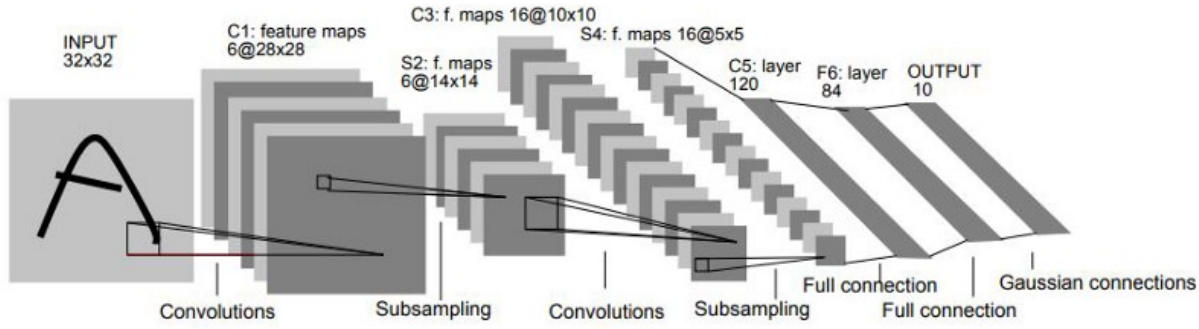
使用`PIL`的`Image`包保存处理图片数据, 使用`Dataset`包构造新数据集类`ChineseDataset`, 重载`__len__`方法与`__getitem__`方法;

数据集划分

训练集, 验证集, 测试集按照7:2:1进行划分, 并使用`DataLoader`加载数据(以`Tensor`格式), 方便后续将数据迁移到GPU进行并行加速.

2.2 网络结构

参考LeNet5网络结构, 包含 3 个卷积层, 2 个池化层, 1 个全连接层,其中所有卷积层的所有卷积核都为5x5, 步长`stride=1`, 池化方法都为`Max pooling`, 激活函数为`ReLu`.




```
self.cnn = nn.Sequential(
    nn.Conv2d(1, 20, 5),
    nn.BatchNorm2d(20),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2, 2), stride=2),

    nn.Conv2d(20, 50, 5),
    nn.BatchNorm2d(50),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=(2, 2), stride=2),
)
    self.fullyConnected = nn.Sequential(
    nn.Linear(50*4*4, 500),
    nn.ReLU(),
    nn.Linear(500, 12)
)
```

其中使用了`BatchNorm2d`解决过拟合问题, 对于数据进行归一化处理, 让训练更加平稳而不至于收到异常值干扰.

2.3 优化器

本次项目中使用`Adam`优化器, 使用了动量+动态调整学习率, 相较于最原始的SGD, 训练更加平稳, 避免过拟合.

2.4 训练

本项目在GPU(`NVIDIA GeForce GTX 3090`)上运行, 训练超参数为:

```
device = "cuda" if torch.cuda.is_available() else "cpu"
model = CNN().to(device)
batch_size = 16
epochs = 20
loss_function = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.0003, weight_decay=1e-5)
```

在验证集上的acc达到了`0.99261`

```
19 epoch: loss = 0.03185, acc = 0.98946
100%|██████████| 93/93 [00:00<00:00, 129.18it/s]
  2%||          | 8/326 [00:00<00:04, 73.43it/s]
19 epoch: loss = 0.02342, acc = 0.99261. Best acc on valid set.
save model
100%|██████████| 326/326 [00:03<00:00, 85.12it/s]
 15%|█         | 14/93 [00:00<00:00, 131.14it/s]
20 epoch: loss = 0.02851, acc = 0.99061
100%|██████████| 93/93 [00:00<00:00, 129.35it/s]
20 epoch: loss = 0.03571, acc = 0.98858.
done
```

2.5 测试

加载最优模型参数和数据, 获得测试集上的结果

```
test_loader = DataLoader(test_set, batch_size=batch_size,
                         shuffle=False, pin_memory=True)

model_best = CNN().to(device)
model_best.load_state_dict(torch.load("best_parameter.ckpt"))
model_best.eval()
test_accs = []
with torch.no_grad():
    for data, labels in tqdm(test_loader):
        data, labels = data.to(device), labels.to(device)
        logits = model_best(data)
        pred_labels = logits.argmax(dim=1)  # 获取预测的类别
        acc = (pred_labels == labels).sum().item() / len(labels)
        test_accs.append(acc)

    test_acc = sum(test_accs) / len(test_accs)
    print(f"testdata acc = {test_acc:.5f}")
```

✓ 0.3s

```
  0%|          | 0/47 [00:00<?, ?it/s]
100%|██████████| 47/47 [00:00<00:00, 130.17it/s]
testdata acc = 0.98670
```

`acc=98.67%`

2.6 调参

学习率

epochs = 10, batch size = 2, momentum=0.9, weight_decay=1e-5

学习率	测试集分类准确率（%）
1e-1	7.661
1e-2	91.129
1e-3	98.790
1e-4	97.581
1e-5	77.285

学习率过大会造成网络不能收敛; 学习率过小时参数更新幅度太小. 在本次实验条件下学习率设为1e-3~1e-4之间最合适

momentum

epochs = 10, batch size = 2, lr=0.001, weight_decay=1e-5

momentum	测试集分类准确率（%）
0.5	97.312
0.9	98.790
0.99	88.306

结果分析：momentum=0.5~0.9时准确率最高, 表示对此前1/(1-momentum)步的梯度进行更新, 当包含过去梯度过多时, 可能的导致当前梯度受到过去错误的梯度方向影响太大, 导致准确率降低.

网络结构

channel个数

channel1	channel2	测试集分类准确率（%）
4	12	96.809
20	50	99.069
100	200	97.872

channel太少时, 提取的特征数不足以代表整张图片; channel太多则会导致参数过多, 过拟合.

3. 实验总结

重新走了一遍Yann LeCun的老路!