中山大学计算机学院

人工智能

本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	202320346	专业（方向）	计算机科学与技术
学号	21312450	姓名	林隽哲

一、 实验题目

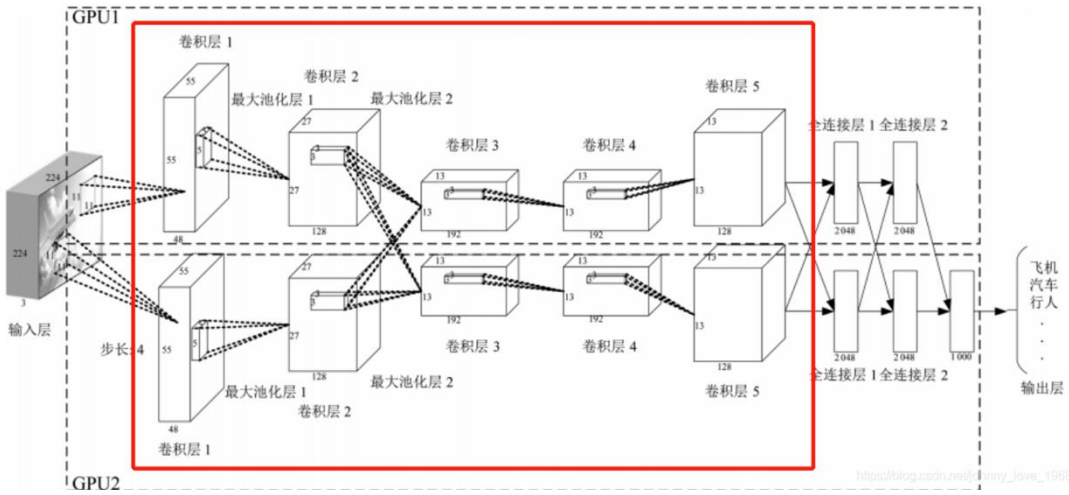
中药图片分类任务

利用pytorch框架搭建神经网络实现中药图片分类，其中中药图片数据分为训练集train和测试集test，训练集仅用于网络训练阶段，测试集仅用于模型的性能测试阶段。训练集和测试集均包含五种不同类型的中药图片：baihe、dangshen、gouqi、huaihua、jinyinhua。请合理设计神经网络架构，利用训练集完成网络训练，统计网络模型的训练准确率和测试准确率，画出模型的训练过程的loss曲线、准确率曲线。

二、 实验内容

1. 算法原理

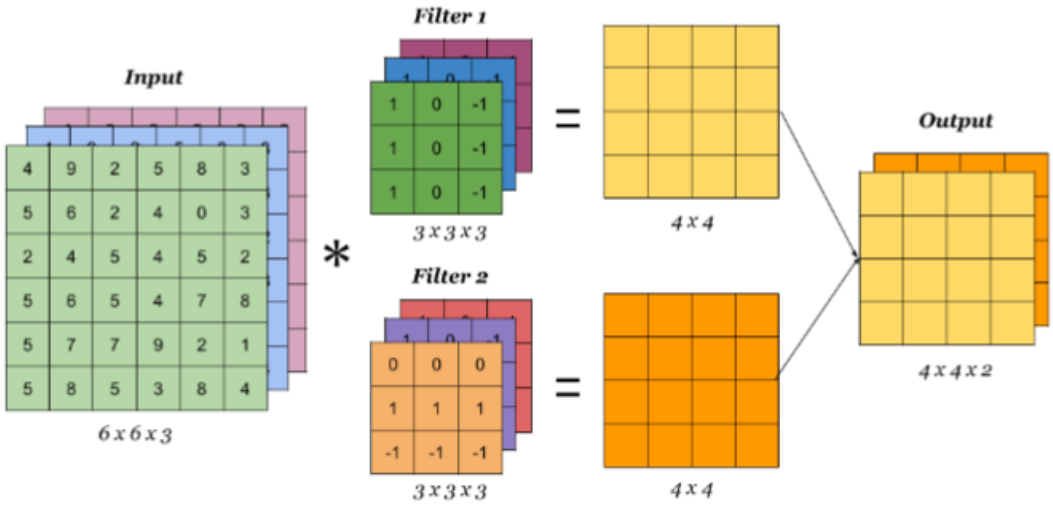
卷积神经网路（Convolution Neural Network, CNN）



（1） 输入层：

输入层接受原始图像数据。本次实验使用的训练数据为彩色的图像，均由三个颜色通道（RGB）组成，因此输入层的输入图像的厚度为 3。同时，为了方便计算，需要在数据预处理时统一每张图片的宽高。

（2） 卷积层：



卷积层由 **torch.nn.Conv2d** 实现。卷积层接受 5 个主要参数：

• **in_channels**: 输入维度，第一层的卷积层作为输入层，输入维度为 3。随后的卷积层的输入维度取决于上一个最大池化层的输出维度；

• **out_channels**: 输出维度，也即为卷积层使用的卷积核的个数；

• **kernel_size**: 卷积核的大小；

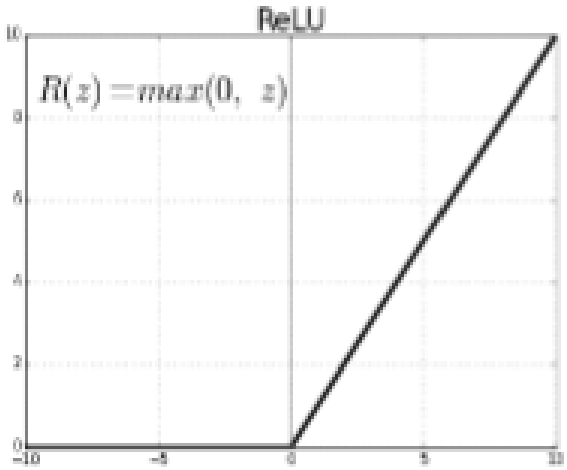
• **stride**: 卷积核移动的步长；

• **padding**：图像的边缘扩充。

由于我想要实现当一个图像在经过卷积变化后的大小与原来的一致，因此我选择了将每一个卷积层的 **stride** 都设为了 1，此时只要满足 $\boldsymbol{padding = \lfloor (kernelsize - 1)/2 \rfloor}$ 即可。

（3） 激活函数：

这里使用 **ReLU** 函数作为激活函数。通过 **torch.nn.ReLU** 实现。



（4） 最大池化层：

池化层通过减小特征图的大小来减少计算复杂性。它通过选择池化窗口内的最大值或平均值来实现。这有利于提取最重要的特征。最大池化层通过 **torch.nn.Maxpool2d** 实现。

（5） 全连接和输出：

最后，全连接层将提取的特征映射转化为网络的最终输出。这可以是一个分类标签、回归值或其他任务的结果。

2. 伪代码

CNN 结构的伪代码

```
conv_layers = ModuleList()

for 1 to conv_layer_number begin
    conv_layers.append(
        Conv2d(
            in_channels,
            out_channels,
            kernel_size,
            stride,
            padding
        ),
        ReLU(),
        MaxPool2d(pool_size)
    )
end

for i to conv_layer_number begin
    x = conv_layers[i](x)
end

return linear_layer(x)
```

训练流程

网络训练一般步骤

实例化网络 net = Net() 后，计算得到 Loss，并定义网络优化器

optim = nn.optim.Adam(net.parameters(), lr=lr)

在更新前，需清除上一步的梯度，即

optim.zero_grad()

然后 Loss 反向传播：

loss.backward()

最后优化器更新：

optim.step()

3. 关键代码展示

CNN 模型的实现

```
class CNN(nn.Module):
    def __init__(self, img_size=256, in_channels=3, out_channels=5,
conv_layers_params=None):
        super(CNN, self).__init__()

        if conv_layers_params is None:
            conv_layers_params = [
                {'kernel_num': 16, 'kernel_size': 5, 'pool_size': 2},
                {'kernel_num': 32, 'kernel_size': 5, 'pool_size': 2}
            ]

        self.conv_layers = nn.ModuleList()
        current_channels = in_channels

        # Create convolutional layers based on provided parameters
        for params in conv_layers_params:
            self.conv_layers.append(nn.Sequential(
                nn.Conv2d(
                    in_channels=current_channels,
                    out_channels=params['kernel_num'],
                    kernel_size=params['kernel_size'],
                    stride=1,
                    padding=(params['kernel_size'] - 1) // 2
                ),
                nn.ReLU(),
                nn.MaxPool2d(params['pool_size'])
            ))
            current_channels = params['kernel_num']

        # Calculate the size of the flattened features after all conv
and pool layers
        feature_size = self._calculate_feature_size(img_size,
conv_layers_params)

        # Fully connected layer
        self.out = nn.Linear(feature_size, out_channels)

    def _calculate_feature_size(self, img_size, conv_layers_params):
        # Calculate the size of the flattened features after all conv
and pool layers
        for params in conv_layers_params:
```

```
            padding = (params['kernel_size'] - 1) // 2
            img_size = (img_size - params['kernel_size'] + 2 * padding)
// params['pool_size'] + 1
        return img_size**2 * conv_layers_params[-1]['kernel_num']

    def forward(self, x):
        # Pass input through each convolutional layer
        for conv_layer in self.conv_layers:
            x = conv_layer(x)

        # Flatten the output for the fully connected layer
        x = x.view(x.size(0), -1)

        # Pass through the fully connected layer
        output = self.out(x)
        return output
```

训练部分的实现

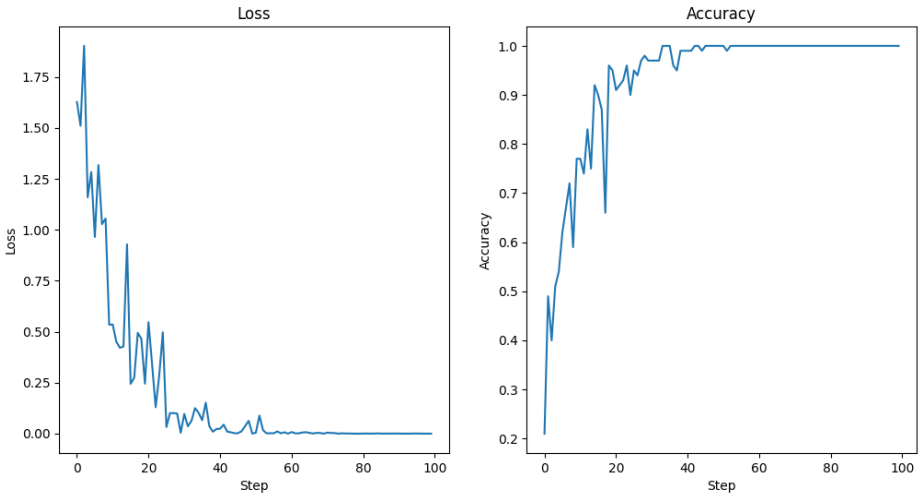
```
for epoch in range(EPOCH):
    for step, (x, y) in enumerate(train_loader):
        x = x.to(device=device)
        y = y.to(device=device)
        # forward
        output = cnn(x)
        # calculate loss
        loss = loss_func(output, y)
        # clear gradients for this training step before backward
        optimizer.zero_grad()
        # backpropagation, compute gradients
        loss.backward()
        # apply gradients
        optimizer.step()
        # Test
        if step and step % 1 == 0:
            pred_y, accuracy = test_model()
            print('Epoch: ', epoch, '| train loss: %.4f' % loss.item(),
'| test accuracy: %.2f' % accuracy)

            loss_record.append(loss.item())
            accuracy_record.append(accuracy)
else:
    print(f'Prediction:\t {pred_y}')
    print(f'Ground Truth:\t {test_y.cpu().data.numpy()}')
```

三、 实验结果及分析

本次实验中我使用了两个测试集，一个为实验数据中给出的 10 个数据集，另一个为我在训练数据中随机抽选出的 100 个数据集。在计算准确率时与画图时，我将会使用较大的数据集。更小的数据集的测试我将会单独给出。

训练过程的 loss 曲线、准确率曲线



使用大测试集最终的 loss 与 accuracy

```
Epoch:  9 | train loss: 0.0002 | test accuracy: 1.00
Epoch:  9 | train loss: 0.0003 | test accuracy: 1.00
Prediction:      [3 1 4 4 4 1 3 2 0 1 4 4 1 4 4 0 0 0 4 2 2 0 0 2 1 0 4 0 0 0 3 3 0 3 2 4 1
 0 1 1 1 1 2 2 0 0 1 3 1 3 0 4 3 2 4 3 2 3 2 4 0 0 4 0 2 2 0 2 0 1 4 3 4 2
 1 1 2 2 4 1 3 1 1 1 4 1 0 2 2 1 2 1 1 2 1 1 1 0 4 2]
Ground Truth:    [3 1 4 4 4 1 3 2 0 1 4 4 1 4 4 0 0 0 4 2 2 0 0 2 1 0 4 0 0 0 3 3 0 3 2 4 1
 0 1 1 1 1 2 2 0 0 1 3 1 3 0 4 3 2 4 3 2 3 2 4 0 0 4 0 2 2 0 2 0 1 4 3 4 2
 1 1 2 2 4 1 3 1 1 1 4 1 0 2 2 1 2 1 1 2 1 1 1 0 4 2]
```

使用小测试集最终的 loss 与 accuracy

```
Epoch:  4 | train loss: 0.1440 | test accuracy: 0.80
Epoch:  4 | train loss: 0.1752 | test accuracy: 0.80
Epoch:  4 | train loss: 0.2628 | test accuracy: 0.90
Epoch:  4 | train loss: 0.0798 | test accuracy: 0.90
Epoch:  4 | train loss: 0.0194 | test accuracy: 1.00
Prediction:      [0 0 1 1 2 2 3 3 4 4]
Ground Truth:    [0 0 1 1 2 2 3 3 4 4]
```

四、 参考资料

- 理论课课件