中山大学计算机学院人工智能本科生实验报告

课程：Artificial Intelligence

姓名： 学号：

一、实验题目

利用 pytorch 框架搭建神经网络实现中药图片分类。

二、实验内容

1.算法原理

卷积神经网络多用于图像处理。对于图像识别来说，局部特征影响大，重要位置常变化，采样压缩也没差，因此使用卷积来对图片进行特征分析。与普通神经网络稍有差别，为了更契合图像的特点，卷积神经网络中还存在卷积层和池化层。

卷积核用于提取图像中的特征，而卷积核中的参数是训练对象。通过将卷积核应用在图像上，可以将图像计算为一个更小的特征图。而计算机中的图像多用RGB，因此卷积核在每一个通道中都有作用，在定义卷积核时要注意将 in_channels 设置为3。

在卷积之后，则将新生成的特征图在加上偏置项之后输入到激活函数中。在这里，使用的激活函数是 ReLU ，定义如下：

$$ReLU(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$$

之后，基于“采样压缩也没差”的原理，对图像做池化操作，进一步压缩特征图大小。这里选用最大池化。

2.伪代码

首先给出类结构。完成任务的类命名为 MedicineClassify ，结构如下：

```
class MedicineClassify:
    def __init__(self) -> None:

    def Train(self) -> None:
    """训练函数"""

    def Test(self) -> None:
    """测试函数"""

    def Draw(self) -> None:
    """绘制函数"""

    def Solve(self) -> None:
    """对外的解决方法"""
```

其中， `Solve` 方法内部是很简单的，就是在迭代次数内循环训练与测试，最后输出结果：

```
def Solve(self) -> None:
    for 迭代次数:
        self.Train()
        self.Test()
    self.Draw()
```

训练函数则是计算预测误差之后、反向传播再优化器更新：

```
def Train(self) -> None:
    设置神经网络为训练模式
    for 训练数据集:
        计算预测误差
        反向传播
        优化器更新
```

测试函数是使用当前迭代得到的参数，在输入测试数据集之后，判断神经网络是否分类正确。在这一步中计算损失函数与准确率：

```
def Test(self) -> None:
    设置神经网络为评估模式
    with torch.no_grad():
        for 测试数据集:
            累加损失
            累加正确分类数
    记录损失
    记录准确率
```

3.关键代码展示

使用 Pytorch 构建CNN框架:

```
class NeuralNet(nn.Module):
    def __init__(self) -> None:
        super(NeuralNet, self).__init__()
        self.conv1 = nn.Conv2d(
            in_channels = 3,
            out_channels = 6,
            kernel_size = 5,
            stride = 1,
        )
        self.conv2 = nn.Conv2d(
            in_channels = 6,
            out_channels = 16,
            kernel_size = 5,
            stride = 1,
        )
        self.fc1 = nn.Linear(16 * 53 * 53, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 5)
        self.pool = nn.MaxPool2d(kernel_size = 2)

    def forward(self, x: torch.tensor) -> torch.tensor:
        x = F.relu(self.conv1(x)) # 卷积后激活
        x = self.pool(x) # 池化
        x = F.relu(self.conv2(x)) # 卷积后激活
        x = self.pool(x) # 池化
        x = x.view(-1, 16 * 53 * 53) # 转化为一维
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return self.fc3(x)
```

完成任务的类命名为 `MedicineClassify` 。其中，构造函数需要配合预先定义好的CNN神经网络类。比较关键的步骤是创建训练集和测试集，并转化为可枚举类型；以及，定义损失函数和优化器，使用的损失函数是交叉熵函数：

```
def __init__(self) -> None:
    ......
    transformation = transforms.Compose([
        transforms.Resize([224, 224]), # 统一截取图片大小为224 * 224
        transforms.ToTensor(), # 转换为张量
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406],
            std=[0.229, 0.224, 0.225]) # 标准化
    ])
    self.train_set = ImageFolder("./train", transform = transformation)
    self.test_set = ImageFolder("./test", transform = transformation)
    self.train_loader = torch.utils.data.DataLoader(self.train_set, batch_size = 4, shuffle = Tr
    self.test_loader = torch.utils.data.DataLoader(self.test_set, batch_size = 4, shuffle = True
    # loader是可枚举类型，方便从数据集中取出数据
    self.NeuralNet = NeuralNet()
    self.Loss = nn.CrossEntropyLoss() # 使用交叉熵函数
    self.Optimizer = torch.optim.SGD(self.NeuralNet.parameters(), lr = 1e-3)
    # 定义优化器
```

训练函数:

```
def Train(self) -> None:
    """训练函数"""
    self.NeuralNet.train() # 设置为训练模式
    for batch, (X, y) in enumerate(self.train_loader):
        loss = self.Loss(self.NeuralNet(X), y) # 计算损失
        self.Optimizer.zero_grad() # 梯度清零
        loss.backward() # 反向传播
        self.Optimizer.step() # 优化器更新
```

测试函数:

```
def Test(self) -> None:
    """测试函数"""
    self.NeuralNet.eval() # 设置为评估模式
    size = len(self.test_loader.dataset)
    num_batches = len(self.test_loader)
    test_loss, correct = 0.0, 0.0
    with torch.no_grad():
        for X, y in self.test_loader:
            pred = self.NeuralNet(X)
            test_loss = test_loss + self.Loss(pred, y).item()
            correct = correct + (pred.argmax(1) == y).type(torch.float).sum().item()
    self.loss_per_iteration.append(test_loss / num_batches)
    self.accuracy_per_iteration.append(correct / size)
```

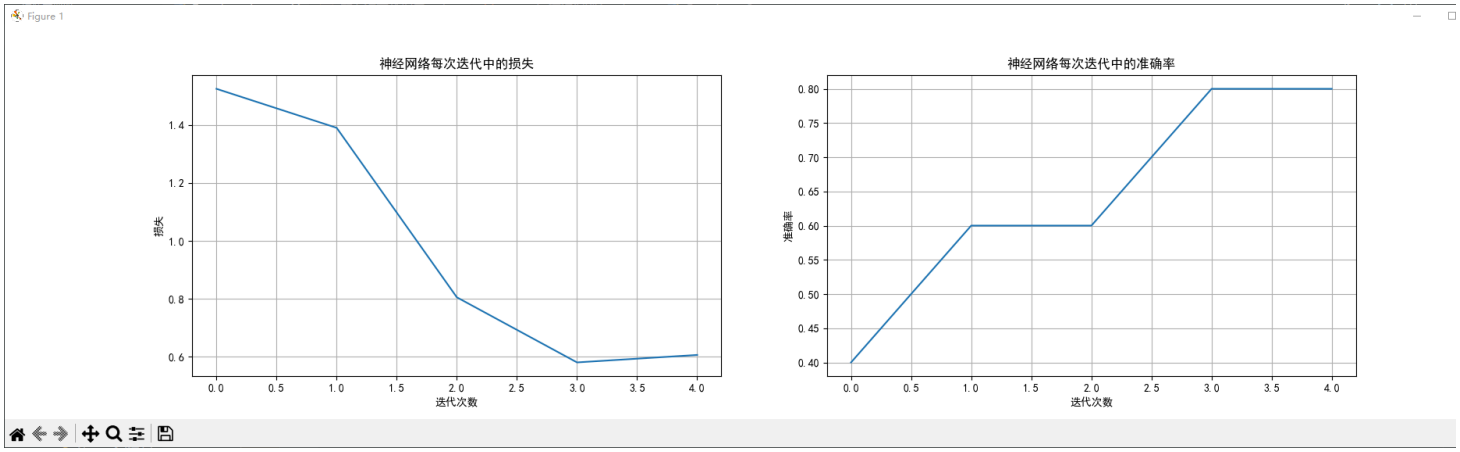
4.创新优化

无。

三、实验结果分析

1.实验结果展示

以下展示几个不同迭代次数情况下的损失与准确率图。当迭代次数为5时：



实际上曲线是有很大可能出现波动的，但是这次运行恰好出现了准确率单调递增的情况。接下来分别是迭代次数为$20, 100, 500, 1000, 3000$的情况：

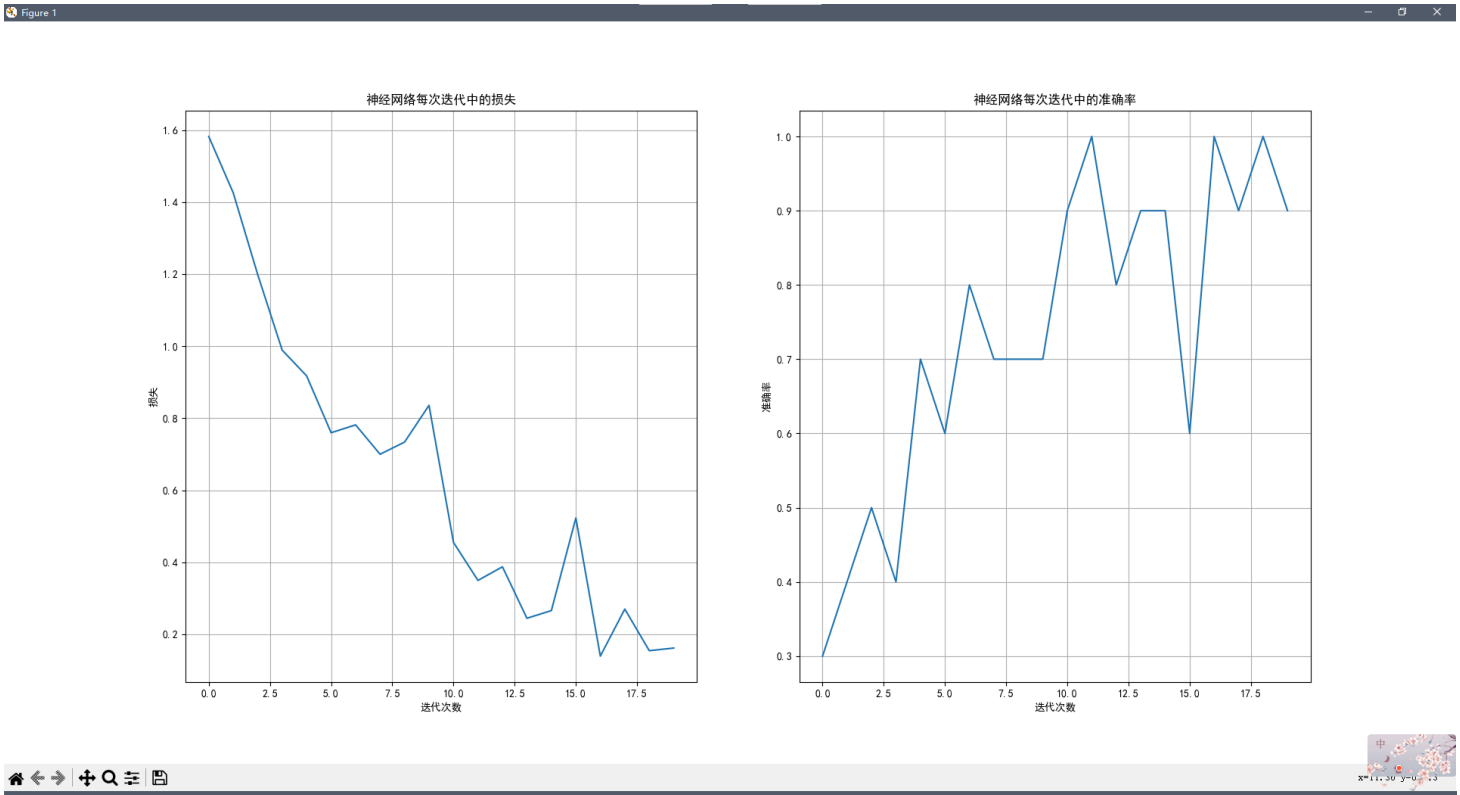


Figure 1
神经网络每次迭代中的损失
损失
迭代次数
神经网络每次迭代中的准确率
准确率
迭代次数

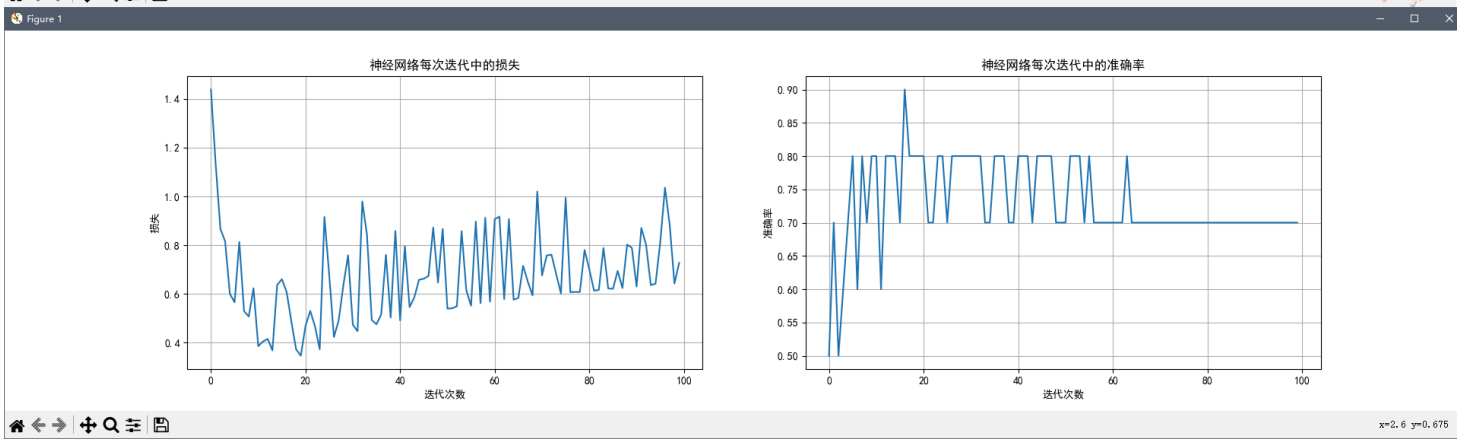


Figure 1
神经网络每次迭代中的损失
损失
迭代次数
神经网络每次迭代中的准确率
准确率
迭代次数
x=2.6 y=0.675

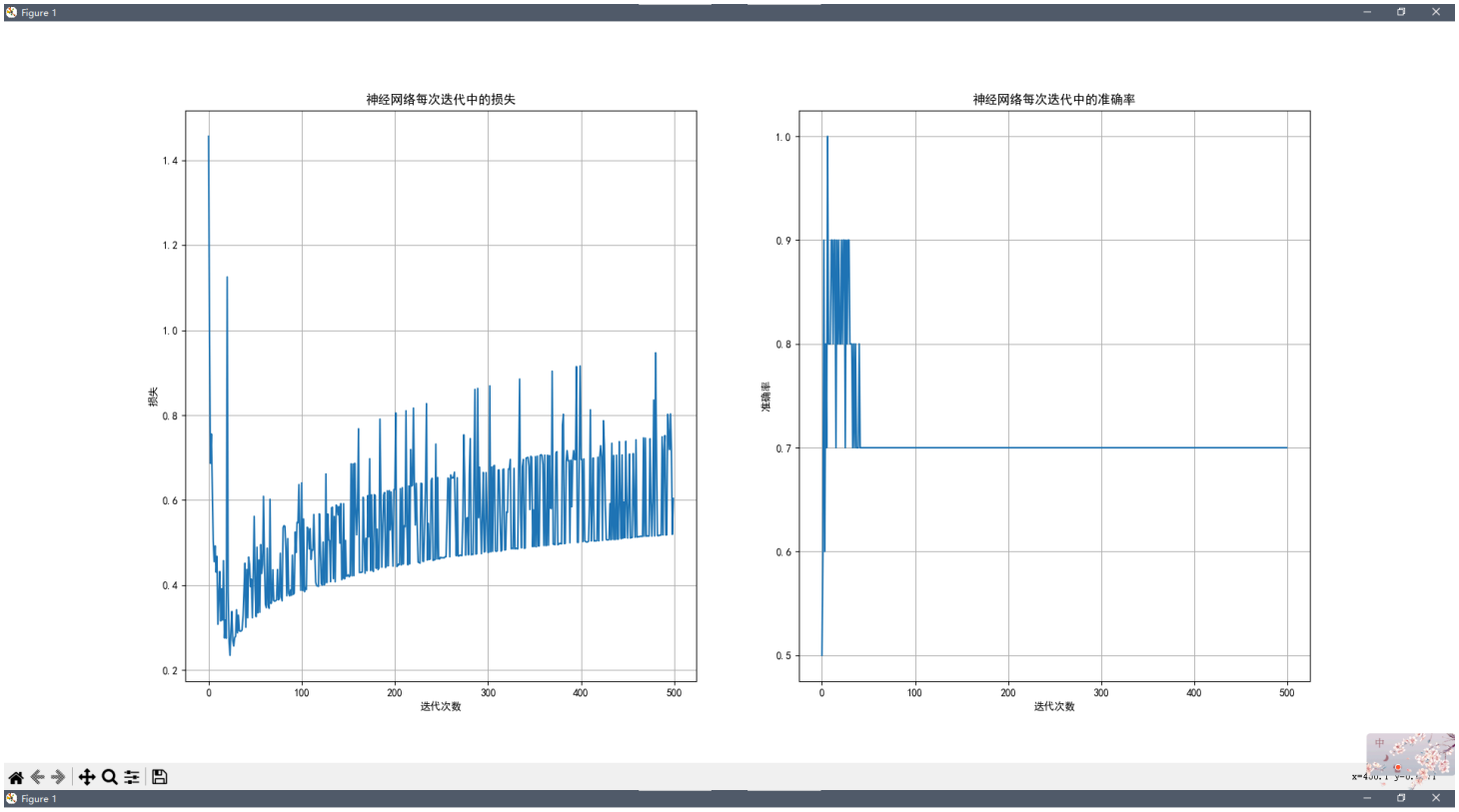


Figure 1
神经网络每次迭代中的损失
损失
迭代次数
神经网络每次迭代中的准确率
准确率
迭代次数

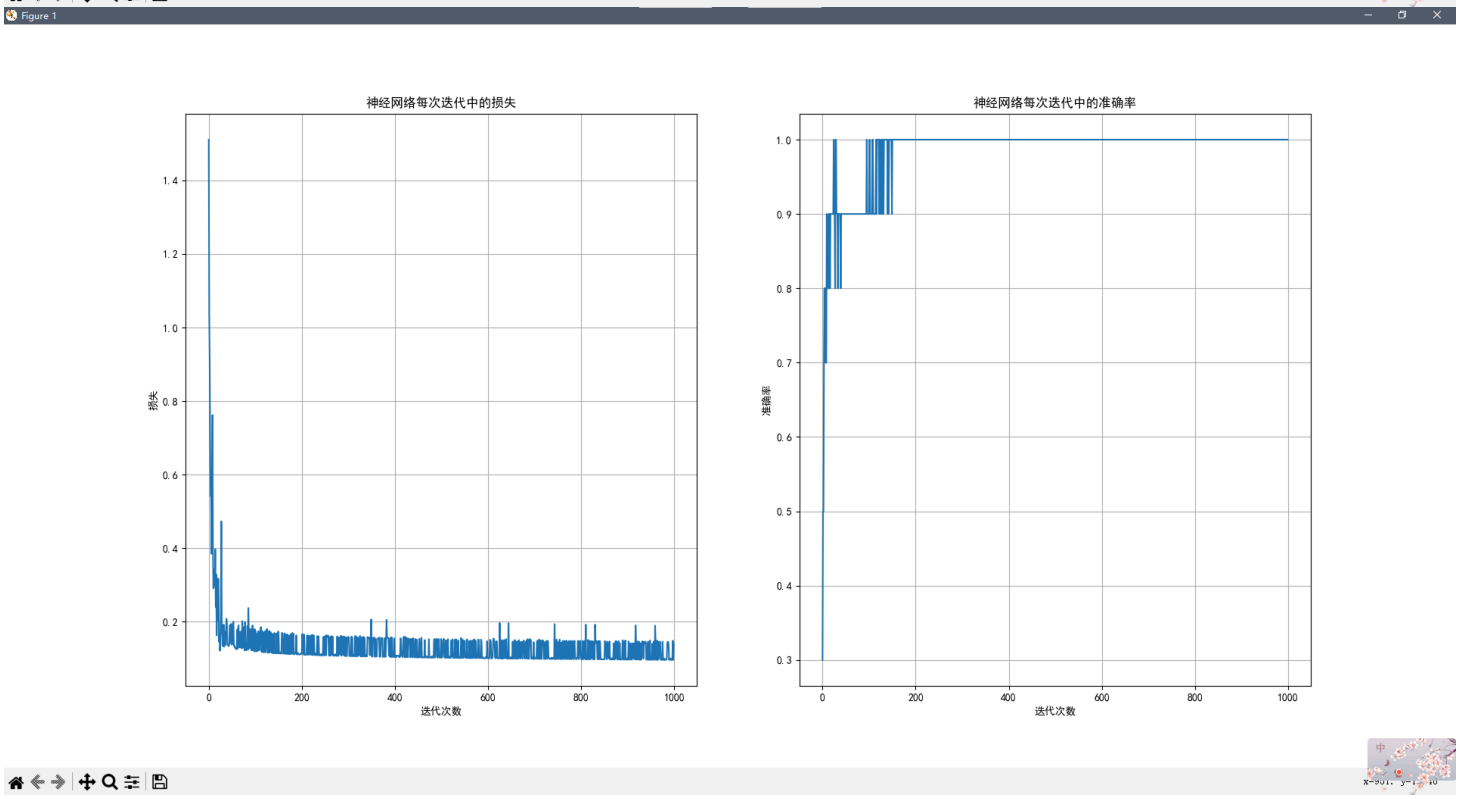
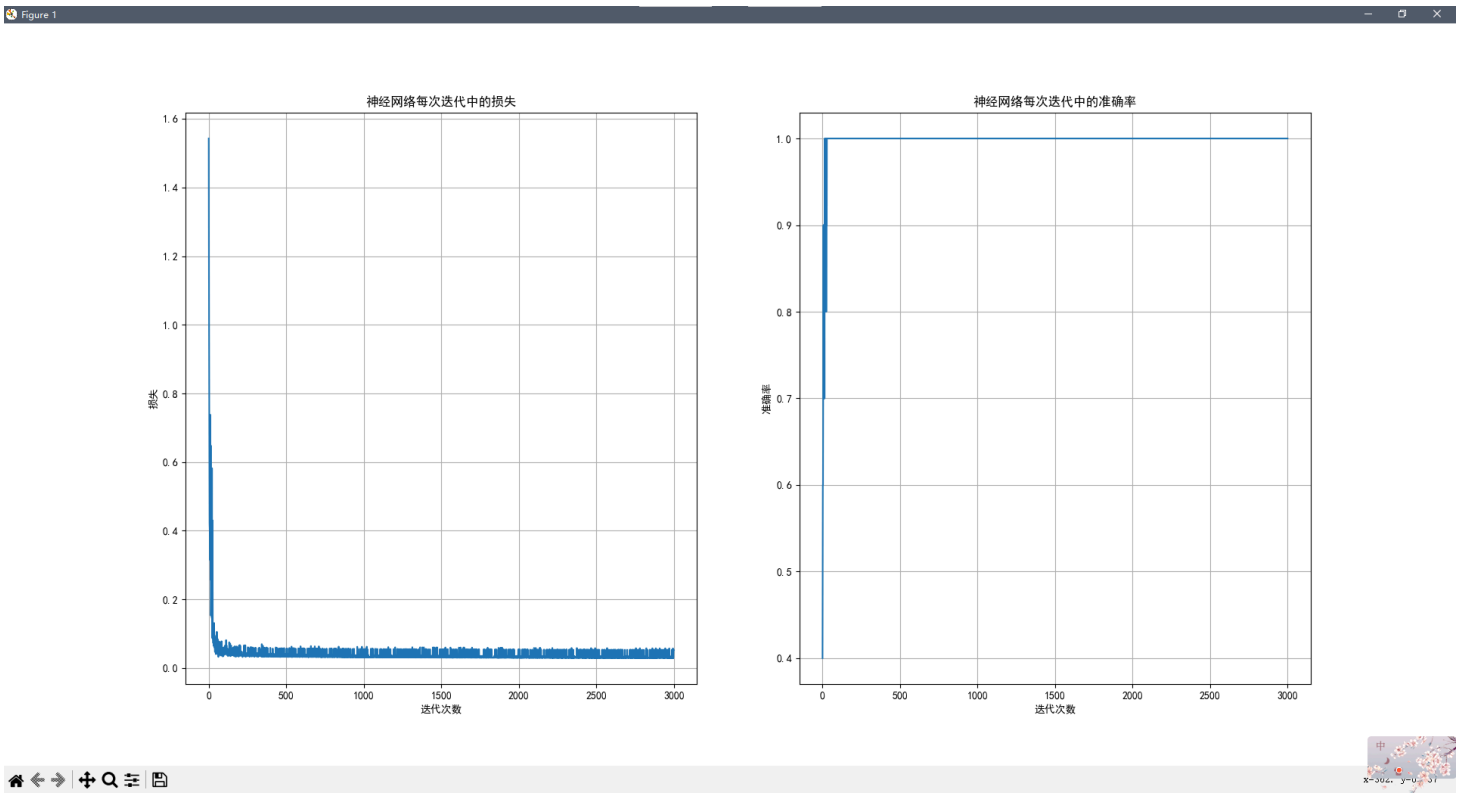


Figure 1
神经网络每次迭代中的损失
损失
迭代次数
神经网络每次迭代中的准确率
准确率
迭代次数



可以看到，有时候结果并不是最优的，损失函数也可能会在达到小值之后波动上升，准确率也可能在达到最大值之后下降。这是神经网络的随机性。

在迭代次数为$1000, 3000$的实验中，准确率非常幸运地稳定驻留在100%了。因此对于这种较小规模的分类任务，较少的迭代次数即可实现非常好的效果。

2.评测指标展示分析

以下展示各迭代次数情况下的运行时长(未使用 `multiprocessing`)(网络结构没有变化)(但可能会受后台程序占用资源影响):

迭代次数	运行时间	最终准确率
5	$1min$	80%
20	$5min$	90%
100	$25min$	70%
500	$2h$	70%
1000	$4h$	100%
3000	$12h$	100%

四、思考题

无。

五、参考资料

深度学习入门-卷积神经网络（一）卷积层 - 严忻恺的文章 - 知乎
https://zhuanlan.zhihu.com/p/259751387