深度学习

Pytorch

Convolution

conv/bn/relu/pool/linear

```
1. torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
2. torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False)

# nn.con2d + F.relu + F.max_pool2d/nn.MaxPool2d + nn.conv2d + F.relu +
F.max_pool2d/nn.MaxPool2d
# + torch.flatten + nn.Linear + F.relu + nn.Dropout2d + nn.Linear +
F.log_softmax

# nn.Conv2d(i,o,k,s,p,d)  nn.BatchNorm2d(c)  nn.ReLU  nn.MaxPool2d(k,s)
nn.Linear(i,o)  nn.Dropout2d(0.5)
# F.relu()  F.max_pool2d(x, kernel_size=k)  F.log_softmax(x, dim=1)             #
torch.nn.functional as F
# MaxPool2d: stride, Default value is kernel_size
# tips: k=3,s=1,p=2 : 尺寸不变
```

$$H_{out} = \lfloor \frac{H_{in} + 2 * p - d * (k-1) - 1}{s} + 1 \rfloor$$
$$H_{out} = \lfloor H_{in} - k + 1 \rfloor \quad \text{if} \quad s = 1, p = 0, d = 1$$

Definition

Define the neural network 定义神经网络层

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


class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()

        self.conv1 = nn.Conv2d(3, 16, 5)
        self.conv2 = nn.Conv2d(16, 64, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.max_pool2d(F.relu(self.conv1(x)), 2)
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1)  # x = x.view(-1, 16 * 5 * 5) [don't forget
flatten!!!]
```

```
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
    #Net(
    #(conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
    #(conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
    #(fc1): Linear(in_features=400, out_features=120, bias=True)
    #(fc2): Linear(in_features=120, out_features=84, bias=True)
    #(fc3): Linear(in_features=84, out_features=10, bias=True)
    #)
    #self.conv = nn.Sequential(
    #     nn.Conv2d(3,16,5),
    #     nn.ReLU(),
    #     nn.MaxPool2d(2)
    # )
    # self.fc = torch.nn.Sequential(
    #     nn.Linear(16 * 5 * 5, 128),
    #     torch.nn.ReLU(),
    #     nn.Dropout(0.2),
    # )

# nn.con2d + F.relu + F.max_pool2d/nn.MaxPool2d + nn.conv2d + F.relu +
F.max_pool2d/nn.MaxPool2d
# + nn.Dropout2d + torch.flatten + nn.Linear + F.relu + nn.Dropout2d + nn.Linear
+ F.log_softmax
#  Dropout放在全连接层的激活函数层之后，防止过拟合，很少放在卷积层，卷积层一般用BN，一般放在激
活层前边
```

Parameter

网络参数

```
params = list(net.parameters())
print(len(params))
print(params[0].size())  # conv1's .weight
```

Dataset

Iterate over a dataset of inputs/Process input through the network

```
net = LeNet()
input = torch.randn(4, 3, 32, 32) # [b,c,h,w]
output = net(input)
print(output)
```

Loss/Propagate

Compute the loss, Propagate gradients back into the network's parameters

```
target = torch.randn(10)  #  [10] a dummy target, for example
target = target.view(1, -1)  # [1, 10] make it the same shape as output

criterion = nn.MSELoss()
loss = criterion(output, target)
```

```
net.zero_grad()     # zeroes the gradient buffers of all parameters
loss.backward()

#input -> conv2d -> relu -> maxpool2d -> conv2d -> relu -> maxpool2d
#      -> flatten -> linear -> relu -> linear -> relu -> linear
#      -> MSELoss
#      -> loss
print(loss.grad_fn)  # MSELoss
print(loss.grad_fn.next_functions[0][0])  # Linear
print(loss.grad_fn.next_functions[0][0].next_functions[0][0])  # ReLU
print('conv1.bias.grad before backward',net.conv1.bias.grad)
print('conv1.bias.grad after backward',net.conv1.bias.grad)

# nn.MSELoss()  nn.L1Loss  nn.BCELoss  nn.CrossEntropyLoss
```

Optimizer

Update the weights of the network

```
import torch.optim as optim

net = Net()
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.01)

optimizer.zero_grad()   # zero the gradient buffers
output = net(input)
loss = criterion(output, target)
net.zero_grad()
loss.backward()
optimizer.step()    # Does the update

# optim.SGD  optim.Adagrad torch.optim.Adam(AMSGrad)
```

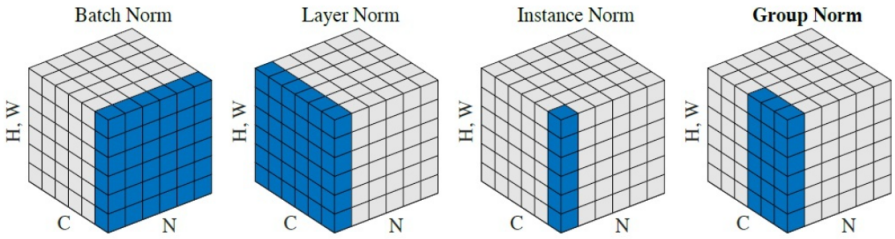
Test

```
net = Net()
net.load_state_dict(torch.load(PATH))
```

Gpu

```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
net.to(device)
inputs, labels = data[0].to(device), data[1].to(device)
```

BN/LN



> BN: [N, C, H, W]，标准的Batch Normalization就是在通道Channel这个维度上进行移动，对所有样本的每一个通道求均值和方差，得到C个均值和方差
>
> LN: 在一个样本的所有通道上标准化
>
> IN: 在一个样本的每个通道上标准化
>
> GN: 在一个样本的几个通道上标准化

卷积/卷积核/通道

- 尺寸问题
  - 输入RGB三通道的彩色图像，输出n个特征，需要n个卷积核，每一个卷积核包含3个矩阵。
  - 3个矩阵与3个通道分别相乘，然后相加再加上偏置，作为一个卷积核的输出，有几个卷积核，就输出几个特征图。
  - 特征图的尺寸计算，如上所示。
- 卷积核的理解

> CNN中的卷积本质上就是利用一个共享参数的过滤器（kernel），通过计算中心像素点以及相邻像素点的加权和来构成feature map实现空间特征的提取，当然加权系数就是卷积核的权重系数。

> 那么卷积核的系数如何确定的呢？是随机化初值，然后根据误差函数通过反向传播梯度下降进行迭代优化。这是一个关键点，卷积核的参数通过优化求出才能实现特征提取的作用，GCN的理论很大一部分工作就是为了引入可以优化的卷积参数。

Loss

优化器

推荐算法

机器学习

交叉熵

交叉熵损失函数 nn.CrossEntropyLoss()，结合了 nn.LogSoftmax() 和 nn.NLLLoss() 两个函数。

$$H(p,q) = -\sum_{x}(p(x)logq(x)$$

提升树

CART

CART是一个二叉树，也是回归树，同时也是分类树，CART的构成简单明了。

CART用GINI指数来决定如何分裂，GINI指数：总体内包含的类别越杂乱，GINI指数就越大（跟熵的概念很相似），比较a和b，发现b的凌乱程度比a要小，即GINI指数b比a小，所以选择b的方案。以此为例，将所有条件列出来，选择GINI指数最小的方案，这个和熵的概念很类似。

CART还是一个回归树，回归解析用来决定分布是否终止。CART对每个叶节点里的数据分析其均值方差，当方差小于一定值可以终止分裂，以换取计算成本的降低。

CART和ID3一样，存在偏向细小分割，即过度学习（过度拟合的问题），为了解决这一问题，对特别长的树进行剪枝处理，直接剪掉。

RF

用随机的方式建立一个森林。RF 算法由很多决策树组成，每一棵决策树之间没有关联。建立完森林后，当有新样本进入时，每棵决策树都会分别进行判断，然后基于投票法给出分类结果。

Random Forest（随机森林）是 Bagging 的扩展变体，它在以决策树为基学习器构建 Bagging 集成的基础上，进一步在决策树的训练过程中引入了随机特征选择，因此可以概括 RF 包括四个部分：

- 随机选择样本（放回抽样）；
- 随机选择特征；
- 构建决策树；
- 随机森林投票（平均）。

Adaboost

AdaBoost（Adaptive Boosting，自适应增强），其自适应在于：

前一个基本分类器分错的样本会得到加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。

Adaboost 迭代算法有三步：

- 初始化训练样本的权值分布，每个样本具有相同权重；
- 训练弱分类器，如果样本分类正确，则在构造下一个训练集中，它的权值就会被降低；反之提高。用更新过的样本集去训练下一个分类器；
- 将所有弱分类组合成强分类器，各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，降低分类误差率大的弱分类器的权重。

GBDT

SVM

LR

简历

老师好，我叫赵书光，来自哈尔滨工业大学深圳研究生院的计算机学院，我的实验室是生物计算研究中心，我的研究课题是图像恢复和图像去噪的领域，之前也做过迁移学习域适应，哈希检索相关的课题研究。

去年发表了一篇关于哈希检索的ccf-c类会议的论文，现在正在写关一篇关于图像去噪的论文。在实验室做过一个关于中药材性状图像识别的项目，提出了基于多视图特征融合的中药材性状识别模型，引入基于类别中心的损失函数，提高了模型识别的性能。

论文

对于传统的离散监督哈希问题提出一个简单而有效的方法。

1. 采用Encoder-Decoder的结构，将监督标签投影到隐空间，在隐空间的表示认这里加一个离散的约束，隐空间就是哈希编码的汉明空间。再从隐空间投影到特征空间。
2. 目标函数是一个量化误差，加入一个松弛变量去优化。对于离散二值化的求解问题，采用DCC循环坐标下降法和迭代优化，得到问题的解析解。
3. 提出的方法更好的利用了标签信息，减少了信息损失，提高了模型的判别性和准确性，在Caltech-256，CIFAR-10和MNIST三个数据集上超过了很多SOTA的基于传统方法的离散监督哈希方法。

项目

项目主要分为三部分

1. 第一部分就是采集数据，我们制作了简易的封闭箱子，控制一定的环境条件，进行数据的采集。
2. 第二部分，是数据的预处理，采集的数据中，一幅图像包含很多个药材，我们首先要进行检测和分割，这里主要采用的是传统的图像处理的方法，首先设置一定的像素阈值和相应的mask矩阵，进行去背景操作，然后检测物体的轮廓并框出来，最后进行切割输出每一个药材的图像。
3. 第三部分，也是最核心的部分就是设计药材识别的模型，我们设计了基于多视图特征融合的图像识别模型，这个多视图指的是药材不同层次的特征信息，我们模型含有三个分支，一个分支用来提取高层的语义信息，用resnet做backbone网络；另外两个分支用来提取颜色形状等细节信息和边缘纹理等高频信息，采用含有跳跃连接的U-Net做backbone网络，其中对于上下采样、BN层有所改变。最后，对于三个分支的特征信息，各自学习一个注意力权重，然后对三个输出进行加权求和，经过softmax层输出最后的预测结果。对于预测的结果采用交叉熵损失函数，在softmax层之前提出了一个基于类别中心的损失函数，目的是为了让不同类别的特征要相互远离，相同类别的特征相互靠近。现在模型的识别准确率可以达到95%以上。

$$L_c = \max\left\{0, M - \frac{1}{|C|(|C|-1)}\sum_{\substack{i\neq j \\ i,j\in C}}\left\|c_i - c_j\right\|\right\} + \frac{1}{|C|}\sum_{k=1}^{|C|}\frac{1}{n_k}\sum_{y_i=k}\left\|g_i - c_{y_i}\right\| \qquad (3\text{-}1)$$

面经