姓名：刘瑜 学号：22020006076

姓名和学号?	**刘瑜，22020006076**
本实验属于哪门课程?	中国海洋大学24秋《软件工程原理与实践》
实验名称?	实验4: MobileNet & ShuffleNet
博客地址?	https://blog.csdn.net/yyds133/article/details/143064975?spm=1001.2014.3001.5501
Github仓库地址?	Yulia2233/software_project (github.com)

一、实验内容

【第一部分：代码练习】

网络部分的代码如下：

```
class_num = 16
windowSize = 25
K = 30
rate = 16

class HybridSN(nn.Module):
  #定义各个层的部分
  def __init__(self):
    super(HybridSN, self).__init__()
    self.S = windowSize
    self.L = K;

    self.conv1 = nn.Conv3d(in_channels=1, out_channels=8, kernel_size=(7, 3, 3))
    self.conv2 = nn.Conv3d(in_channels=8, out_channels=16, kernel_size=(5, 3, 3))
    self.conv3 = nn.Conv3d(in_channels=16, out_channels=32, kernel_size=(3, 3,
3))

    #不懂 inputX经过三重3d卷积的大小
    inputX = self.get2Dinput()
    inputConv4 = inputX.shape[1] * inputX.shape[2]
    # conv4 （24*24=576, 19, 19），64个 3x3 的卷积核 ==>（（64, 17, 17）
    self.conv4 = nn.Conv2d(inputConv4, 64, kernel_size=(3, 3))

    #self-attention
    self.sa1 = nn.Conv2d(64, 64//rate, kernel_size=1)
    self.sa2 = nn.Conv2d(64//rate, 64, kernel_size=1)

    # 全连接层（256个节点） # 64 * 17 * 17 = 18496
    self.dense1 = nn.Linear(18496, 256)
    # 全连接层（128个节点）
    self.dense2 = nn.Linear(256, 128)
    # 最终输出层(16个节点)
    self.dense3 = nn.Linear(128, class_num)
```

```
    #让某个神经元的激活值以一定的概率p，让其停止工作，这次训练过程中不更新权值，也不参加神经网络的计算。
    #self.drop = nn.Dropout(p = 0.4)
    #改成0.43试试
    self.drop = nn.Dropout(p = 0.43)
    self.soft = nn.Softmax(dim=1)
    pass

  #辅助函数，没怎么懂，求经历过三重卷积后二维的一个大小
  def get2Dinput(self):
    #torch.no_grad(): 做运算，但不计入梯度记录
    with torch.no_grad():
      x = torch.zeros((1, 1, self.L, self.S, self.S))
      x = self.conv1(x)
      x = self.conv2(x)
      x = self.conv3(x)
    return x
    pass

  #必须重载的部分，X代表输入
  def forward(self, x):
    #F在上文有定义torch.nn.functional，是已定义好的一组名称
    out = F.relu(self.conv1(x))
    out = F.relu(self.conv2(out))
    out = F.relu(self.conv3(out))

    # 进行二维卷积，因此把前面的 32*18 reshape 一下，得到 （576, 19, 19）
    out = out.view(-1, out.shape[1] * out.shape[2], out.shape[3], out.shape[4])
    out = F.relu(self.conv4(out))

    # Squeeze 第三维卷成1了
    weight = F.avg_pool2d(out, out.size(2))    #参数为输入，kernel

    # Excitation: sa（压缩到16分之一）--Relu--fc（激到之前维度）--Sigmoid（保证输出为0至1之间）
    weight = F.relu(self.sa1(weight))
    weight = F.sigmoid(self.sa2(weight))
    out = out * weight

    # flatten: 变为 18496 维的向量，
    out = out.view(out.size(0), -1)

    out = F.relu(self.dense1(out))
    out = self.drop(out)
    out = F.relu(self.dense2(out))
    out = self.drop(out)
    out = self.dense3(out)
    return out
    pass
```

代码运行与解析：

+ 代码 + 文本

```
[1] ! wget http://www.ehu.eus/ccwintco/uploads/6/67/Indian_pines_corrected.mat
    ! wget http://www.ehu.eus/ccwintco/uploads/c/c4/Indian_pines_gt.mat
    ! pip install spectral
```

```
--2024-10-17 10:22:45--  http://www.ehu.eus/ccwintco/uploads/6/67/Indian_pines_corrected.mat
Resolving www.ehu.eus (www.ehu.eus)... 158.227.0.65, 2001:720:1410::65
Connecting to www.ehu.eus (www.ehu.eus)|158.227.0.65|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://www.ehu.eus/ccwintco/uploads/6/67/Indian_pines_corrected.mat [following]
--2024-10-17 10:22:45--  https://www.ehu.eus/ccwintco/uploads/6/67/Indian_pines_corrected.mat
Connecting to www.ehu.eus (www.ehu.eus)|158.227.0.65|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 5953527 (5.7M)
Saving to: 'Indian_pines_corrected.mat'

Indian_pines_correc 100%[===================>]   5.68M   235KB/s    in 26s

2024-10-17 10:23:12 (222 KB/s) - 'Indian_pines_corrected.mat' saved [5953527/5953527]

URL transformed to HTTPS due to an HSTS policy
--2024-10-17 10:23:12--  https://www.ehu.eus/ccwintco/uploads/c/c4/Indian_pines_gt.mat
Resolving www.ehu.eus (www.ehu.eus)... 158.227.0.65, 2001:720:1410::65
Connecting to www.ehu.eus (www.ehu.eus)|158.227.0.65|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1125 (1.1K)
Saving to: 'Indian_pines_gt.mat'

Indian_pines_gt.mat 100%[===================>]   1.10K  --.-KB/s    in 0s

2024-10-17 10:23:12 (37.5 MB/s) - 'Indian_pines_gt.mat' saved [1125/1125]
```

个人解读：这段代码下载了 **Indian Pines** 高光谱数据集及其对应的地面真值（ground truth），并安装了用于处理高光谱数据的 `spectral` 库。

```
[2] import numpy as np
    import matplotlib.pyplot as plt
    import scipy.io as sio
    from sklearn.decomposition import PCA
    from sklearn.model_selection import train_test_split
    from sklearn.metrics import confusion_matrix, accuracy_score, classification_report, cohen_kappa_score
    import spectral
    import torch
    import torchvision
    import torch.nn as nn
    import torch.nn.functional as F
    import torch.optim as optim
```

个人解读：这段代码导入了用于数据处理和分析的多个库，包括 NumPy、Matplotlib、SciPy、scikit-learn 和 Spectral，以及用于构建和训练深度学习模型的 PyTorch 库，准备进行高光谱图像分类任务的相关处理和模型训练。

```
class_num = 16
windowSize = 25
K = 30              #参考Hybrid-Spectral-Net
rate = 16

class HybridSN(nn.Module):
    #定义各个层的部分
    def __init__(self):
        super(HybridSN, self).__init__()
        self.S = windowSize
        self.L = K;

        #self.conv_block = nn.Sequential()
        ## convolutional layers
        self.conv1 = nn.Conv3d(in_channels=1, out_channels=8, kernel_size=(7, 3, 3))
        self.conv2 = nn.Conv3d(in_channels=8, out_channels=16, kernel_size=(5, 3, 3))
        self.conv3 = nn.Conv3d(in_channels=16, out_channels=32, kernel_size=(3, 3, 3))

        #不懂 inputX经过三重3d卷积的大小
        inputX = self.get2Dinput()
        inputConv4 = inputX.shape[1] * inputX.shape[2]
        # conv4 (24*24=576, 19, 19)，64个 3x3 的卷积核 ==>(64, 17, 17)
        self.conv4 = nn.Conv2d(inputConv4, 64, kernel_size=(3, 3))

        #self-attention
        self.sa1 = nn.Conv2d(64, 64//rate, kernel_size=1)
        self.sa2 = nn.Conv2d(64//rate, 64, kernel_size=1)

        # 全连接层（256个节点） # 64 * 17 * 17 = 18496
        self.dense1 = nn.Linear(18496, 256)
        # 全连接层（128个节点）
        self.dense2 = nn.Linear(256, 128)
        # 最终输出层(16个节点)
        self.dense3 = nn.Linear(128, class_num)

        self.drop = nn.Dropout(p = 0.43)
        self.soft = nn.Softmax(dim=1)
        pass
```

```
        pass

    #辅助函数，求经历过三重卷积后二维的一个大小
    def get2Dinput(self):
        #torch.no_grad(): 做运算，但不计入梯度记录
        with torch.no_grad():
            x = torch.zeros((1, 1, self.L, self.S, self.S))
            x = self.conv1(x)
            x = self.conv2(x)
            x = self.conv3(x)
        return x
        pass
```

```
    #必须重载的部分，X代表输入
    def forward(self, x):
        #F在上文有定义torch.nn.functional，是已定义好的一组名称
        out = F.relu(self.conv1(x))
        out = F.relu(self.conv2(out))
        out = F.relu(self.conv3(out))

        # 进行二维卷积，因此把前面的 32*18 reshape 一下，得到 (576, 19, 19)
        out = out.view(-1, out.shape[1] * out.shape[2], out.shape[3], out.shape[4])
        out = F.relu(self.conv4(out))

        # Squeeze 第三维卷成1了
        weight = F.avg_pool2d(out, out.size(2))        #参数为输入，kernel
        # Excitation: sa（压缩到16分之一）--Relu--fc（激到之前维度）--Sigmoid（保证输出为0至1之间）
        weight = F.relu(self.sa1(weight))
        weight = F.sigmoid(self.sa2(weight))
        out = out * weight

        out = out.view(out.size(0), -1)

        out = F.relu(self.dense1(out))
        out = self.drop(out)
        out = F.relu(self.dense2(out))
        out = self.drop(out)
        out = self.dense3(out)
        return out
        pass
```

个人解读：这段代码定义了一个名为 **HybridSN** 的深度学习模型类，继承自 `nn.Module`，用于处理高光谱图像分类任务。该模型包含多层 3D 卷积和 2D 卷积，以及自注意力机制，以提取图像特征。模型的初始化方法定义了各个卷积层、全连接层和自注意力层，并计算经过三重卷积后的特征图尺寸。`forward` 方法实现了前向传播过程，包括激活函数应用、特征重塑、自注意力加权、全连接层处理及最终分类输出。

```
[4]  #  对高光谱数据  X  应用  PCA  变换
     def  applyPCA(X,  numComponents):
            newX  =  np.reshape(X,  (-1,  X.shape[2]))
            pca  =  PCA(n_components=numComponents,  whiten=True)
            newX  =  pca.fit_transform(newX)
            newX  =  np.reshape(newX,  (X.shape[0],  X.shape[1],  numComponents))
            return  newX

     #  对单个像素周围提取  patch  时，边缘像素就无法取了，因此，给这部分像素进行  padding  操作
     def  padWithZeros(X,  margin=2):
            newX  =  np.zeros((X.shape[0]  +  2  *  margin,  X.shape[1]  +  2*  margin,  X.shape[2]))
            x_offset  =  margin
            y_offset  =  margin
            newX[x_offset:X.shape[0]  +  x_offset,  y_offset:X.shape[1]  +  y_offset,  :]  =  X
            return  newX

     #  在每个像素周围提取  patch  ，然后创建成符合  keras  处理的格式
     def  createImageCubes(X,  y,  windowSize=5,  removeZeroLabels  =  True):
            #  给  X  做  padding
            margin  =  int((windowSize  -  1)  /  2)
            zeroPaddedX  =  padWithZeros(X,  margin=margin)
            #  split  patches
            patchesData  =  np.zeros((X.shape[0]  *  X.shape[1],  windowSize,  windowSize,  X.shape[2]))
            patchesLabels  =  np.zeros((X.shape[0]  *  X.shape[1]))
            patchIndex  =  0
            for  r  in  range(margin,  zeroPaddedX.shape[0]  -  margin):
                   for  c  in  range(margin,  zeroPaddedX.shape[1]  -  margin):
                          patch  =  zeroPaddedX[r  -  margin:r  +  margin  +  1,  c  -  margin:c  +  margin  +  1]
                          patchesData[patchIndex,  :,  :,  :]  =  patch
                          patchesLabels[patchIndex]  =  y[r-margin,  c-margin]
                          patchIndex  =  patchIndex  +  1
            if  removeZeroLabels:
                   patchesData  =  patchesData[patchesLabels>0,:,:,:]
                   patchesLabels  =  patchesLabels[patchesLabels>0]
                   patchesLabels  -=  1
            return  patchesData,  patchesLabels

     def  splitTrainTestSet(X,  y,  testRatio,  randomState=345):
            X_train,  X_test,  y_train,  y_test  =  train_test_split(X,  y,  test_size=testRatio,  random_state=randomState,  stratify=y)
```

个人解读：这段代码包含多个函数，用于处理高光谱图像数据。具体功能包括：

1. `applyPCA(X, numComponents)`：对高光谱数据 `X` 应用主成分分析（PCA）变换，以减少维度到 `numComponents`，并返回变换后的数据。
2. `padWithZeros(X, margin=2)`：对输入数据 `X` 进行零填充，以处理边缘像素时的特征提取问题，填充的边缘宽度由 `margin` 参数指定。
3. `createImageCubes(X, y, windowSize=5, removeZeroLabels=True)`：在每个像素周围提取 `windowSize` x `windowSize` 的小块（patch），并将其格式化为适合 Keras 处理的数据格式，同时提取对应的标签。可以选择去除标签为零的样本。
4. `splitTrainTestSet(X, y, testRatio, randomState=345)`：将数据集 `X` 和标签 `y` 分割为训练集和测试集，按照指定的测试比例 `testRatio`，并确保数据分割的随机性和标签的分层分布。

```
        def __len__(self):
            # 返回文件数据的数目
            return self.len

""" Testing dataset"""
class TestDS(torch.utils.data.Dataset):
    def __init__(self):
        self.len = Xtest.shape[0]
        self.x_data = torch.FloatTensor(Xtest)
        self.y_data = torch.LongTensor(ytest)
    def __getitem__(self, index):
        # 根据索引返回数据和对应的标签
        return self.x_data[index], self.y_data[index]
    def __len__(self):
        # 返回文件数据的数目
        return self.len

# 创建 trainloader 和 testloader
trainset = TrainDS()
testset  = TestDS()
train_loader = torch.utils.data.DataLoader(dataset=trainset, batch_size=128, shuffle=True, num_workers=2)
test_loader  = torch.utils.data.DataLoader(dataset=testset,  batch_size=128, shuffle=False, num_workers=2)
```

```
Hyperspectral data shape:  (145, 145, 200)
Label shape:  (145, 145)

... ... PCA tranformation ... ...
Data shape after PCA:  (145, 145, 30)

... ... create data cubes ... ...
Data cube X shape:  (10249, 25, 25, 30)
Data cube y shape:  (10249,)

... ... create train & test data ... ...
Xtrain shape:  (1024, 25, 25, 30)
Xtest  shape:  (9225, 25, 25, 30)
before transpose: Xtrain shape:  (1024, 25, 25, 30, 1)
before transpose: Xtest  shape:  (9225, 25, 25, 30, 1)
after transpose: Xtrain shape:  (1024, 1, 30, 25, 25)
after transpose: Xtest  shape:  (9225, 1, 30, 25, 25)
```

个人解读：实现了高光谱图像数据的处理和准备工作，具体步骤如下：

1. **加载数据**：从 `.mat` 文件中加载 Indian Pines 高光谱数据及其对应的地面真值（标签）。
2. **设置参数**：定义地物类别数、测试样本比例、提取的小块（patch）尺寸以及用于 PCA 降维的主成分数量。
3. **PCA 降维**：对高光谱数据进行 PCA 变换以减少特征维度，并打印变换后的数据形状。
4. **提取图像小块**：通过 `createImageCubes` 函数在每个像素周围提取指定尺寸的小块，并创建适合模型处理的数据格式。
5. **分割数据集**：使用 `splitTrainTestSet` 将数据集分为训练集和测试集。
6. **调整数据形状**：将训练和测试数据调整为符合 Keras 要求的形状，并转置为 PyTorch 需要的格式。
7. **定义数据集类**：定义 `TrainDS` 和 `TestDS` 类以创建自定义数据集，包含数据和标签的获取方法。
8. **创建数据加载器**：通过 `DataLoader` 为训练集和测试集创建数据加载器，以便在训练期间批量加载数据。

```
# 使用GPU训练，可以在菜单 "代码执行工具" -> "更改运行时类型" 里进行设置
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# 网络放到GPU上
net = HybridSN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)

# 开始训练
total_loss = 0
for epoch in range(100):
    for i, (inputs, labels) in enumerate(train_loader):
        inputs = inputs.to(device)
        labels = labels.to(device)
        # 优化器梯度归零
        optimizer.zero_grad()
        # 正向传播 + 反向传播 + 优化
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    print('[Epoch: %d]     [loss avg: %.4f]     [current loss: %.4f]' %(epoch + 1, total_loss/(epoch+1), loss.item()))

print('Finished Training')
```

```
[Epoch: 84]   [loss avg: 2.6096]   [current loss: 0.0080]
[Epoch: 85]   [loss avg: 2.5806]   [current loss: 0.0298]
[Epoch: 86]   [loss avg: 2.5526]   [current loss: 0.0117]
[Epoch: 87]   [loss avg: 2.5273]   [current loss: 0.0058]
[Epoch: 88]   [loss avg: 2.5023]   [current loss: 0.0401]
[Epoch: 89]   [loss avg: 2.4758]   [current loss: 0.0201]
[Epoch: 90]   [loss avg: 2.4511]   [current loss: 0.0496]
[Epoch: 91]   [loss avg: 2.4260]   [current loss: 0.0088]
[Epoch: 92]   [loss avg: 2.4011]   [current loss: 0.0605]
[Epoch: 93]   [loss avg: 2.3768]   [current loss: 0.0496]
[Epoch: 94]   [loss avg: 2.3530]   [current loss: 0.0009]
[Epoch: 95]   [loss avg: 2.3293]   [current loss: 0.0057]
[Epoch: 96]   [loss avg: 2.3056]   [current loss: 0.0047]
[Epoch: 97]   [loss avg: 2.2832]   [current loss: 0.0607]
[Epoch: 98]   [loss avg: 2.2607]   [current loss: 0.0020]
[Epoch: 99]   [loss avg: 2.2385]   [current loss: 0.0146]
[Epoch: 100]   [loss avg: 2.2169]   [current loss: 0.0010]
Finished Training
```

个人解读：用于在 GPU 上训练 **HybridSN** 模型，通过交叉熵损失函数和 Adam 优化器优化模型参数，并在每个 epoch 打印训练损失信息。

```
count = 0
# 模型测试
for inputs, _ in test_loader:
        inputs = inputs.to(device)
        outputs = net(inputs)
        outputs = np.argmax(outputs.detach().cpu().numpy(), axis=1)
        if count == 0:
                y_pred_test =    outputs
                count = 1
        else:
                y_pred_test = np.concatenate( (y_pred_test, outputs) )

# 生成分类报告
classification = classification_report(ytest, y_pred_test, digits=4)
print(classification)
```

```
              precision    recall  f1-score   support

         0.0     0.7955    0.8537    0.8235        41
         1.0     0.9707    0.9541    0.9623      1285
         2.0     0.9573    0.9893    0.9730       747
         3.0     0.9648    0.9014    0.9320       213
         4.0     0.9766    0.9609    0.9687       435
         5.0     0.9802    0.9772    0.9787       657
         6.0     1.0000    0.9200    0.9583        25
         7.0     0.9840    1.0000    0.9919       430
         8.0     0.7778    0.7778    0.7778        18
         9.0     0.9633    0.9886    0.9757       875
        10.0     0.9757    0.9792    0.9774      2210
        11.0     0.9665    0.9719    0.9692       534
        12.0     0.9781    0.9676    0.9728       185
        13.0     0.9938    0.9877    0.9908      1139
        14.0     0.9881    0.9568    0.9722       347
        15.0     0.9024    0.8810    0.8916        84

    accuracy                         0.9731      9225
   macro avg     0.9484    0.9417    0.9447      9225
weighted avg     0.9732    0.9731    0.9731      9225
```

个人解读：用于对训练好的 **HybridSN** 模型进行测试，计算并输出分类报告。具体而言，它遍历测试数据加载器，获取模型的预测结果，使用 `np.argmax` 获取每个样本的类别标签，然后将所有预测结果合并，并最终生成并打印分类报告，以评估模型在测试集上的分类性能。

```python
from operator import truediv
def AA_andEachClassAccuracy(confusion_matrix):
    counter = confusion_matrix.shape[0]
    list_diag = np.diag(confusion_matrix)
    list_raw_sum = np.sum(confusion_matrix, axis=1)
    each_acc = np.nan_to_num(truediv(list_diag, list_raw_sum))
    average_acc = np.mean(each_acc)
    return each_acc, average_acc
def reports (test_loader, y_test, name):
    count = 0
    # 模型测试
    for inputs, _ in test_loader:
        inputs = inputs.to(device)
        outputs = net(inputs)
        outputs = np.argmax(outputs.detach().cpu().numpy(), axis=1)
        if count == 0:
            y_pred = outputs
            count = 1
        else:
            y_pred = np.concatenate( (y_pred, outputs) )

    if name == 'IP':
        target_names = ['Alfalfa', 'Corn-notill', 'Corn-mintill', 'Corn'
                        ,'Grass-pasture', 'Grass-trees', 'Grass-pasture-mowed',
                        'Hay-windrowed', 'Oats', 'Soybean-notill', 'Soybean-mintill',
                        'Soybean-clean', 'Wheat', 'Woods', 'Buildings-Grass-Trees-Drives',
                        'Stone-Steel-Towers']
    elif name == 'SA':
        target_names = ['Brocoli_green_weeds_1','Brocoli_green_weeds_2','Fallow','Fallow_rough_plow','Fallow_smooth',
                        'Stubble','Celery','Grapes_untrained','Soil_vinyard_develop','Corn_senesced_green_weeds',
                        'Lettuce_romaine_4wk','Lettuce_romaine_5wk','Lettuce_romaine_6wk','Lettuce_romaine_7wk',
                        'Vinyard_untrained','Vinyard_vertical_trellis']
    elif name == 'PU':
        target_names = ['Asphalt','Meadows','Gravel','Trees', 'Painted metal sheets','Bare Soil','Bitumen',
                        'Self-Blocking Bricks','Shadows']
    classification = classification_report(y_test, y_pred, target_names=target_names)
    oa = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    each_acc, aa = AA_andEachClassAccuracy(confusion)
    kappa = cohen_kappa_score(y_test, y_pred)

    return classification, confusion, oa*100, each_acc*100, aa*100, kappa*100
```

个人解读：这段代码实现了高光谱图像分类模型的评估功能。具体而言，定义了两个函数：

1. `AA_andEachClassAccuracy(confusion_matrix)`：计算每个类别的准确率（每类准确率）和平均准确率（AA），输入为混淆矩阵。
2. `reports(test_loader, y_test, name)`：在给定的测试集上评估模型的性能：
    - 遍历测试数据加载器，获取模型预测结果，并将其合并为一个数组。
    - 根据传入的 `name` 参数确定目标类别名称。
    - 生成分类报告，计算整体准确率、混淆矩阵、每类准确率、平均准确率和 Cohen's Kappa 系数，最终返回这些评估指标。

```python
[11] classification, confusion, oa, each_acc, aa, kappa = reports(test_loader, ytest, 'IP')
     classification = str(classification)
     confusion = str(confusion)
     file_name = "classification_report.txt"

     with open(file_name, 'w') as x_file:
         x_file.write('\n')
         x_file.write('{} Kappa accuracy (%)'.format(kappa))
         x_file.write('\n')
         x_file.write('{} Overall accuracy (%)'.format(oa))
         x_file.write('\n')
         x_file.write('{} Average accuracy (%)'.format(aa))
         x_file.write('\n')
         x_file.write('\n')
         x_file.write('{}'.format(classification))
         x_file.write('\n')
         x_file.write('{}'.format(confusion))
```

个人解读：这段代码用于将分类报告和评估指标写入文本文件 `classification_report.txt`。具体流程如下：

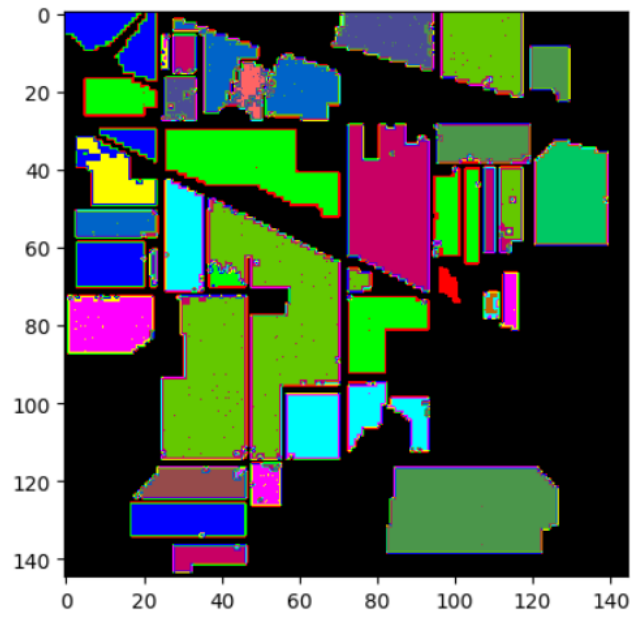
1. 调用 `reports` 函数，获取分类报告、混淆矩阵、总体准确率、每类准确率、平均准确率和 Cohen's Kappa 系数。
2. 将这些结果转换为字符串格式，以便于写入文件。
3. 使用 `with open` 语句打开或创建 `classification_report.txt` 文件，并将结果写入文件，包括 Kappa 准确率、总体准确率、平均准确率、分类报告和混淆矩阵。

```
[12] # load the original image
     X = sio.loadmat('Indian_pines_corrected.mat')['indian_pines_corrected']
     y = sio.loadmat('Indian_pines_gt.mat')['indian_pines_gt']

     height = y.shape[0]
     width = y.shape[1]

     X = applyPCA(X, numComponents= pca_components)
     X = padWithZeros(X, patch_size//2)

     # 逐像素预测类别
     outputs = np.zeros((height,width))
     for i in range(height):
            for j in range(width):
                   if int(y[i,j]) == 0:
                          continue
                   else :
                          image_patch = X[i:i+patch_size, j:j+patch_size, :]
                          image_patch = image_patch.reshape(1,image_patch.shape[0],image_patch.shape[1], image_patch.shape[2], 1)
                          X_test_image = torch.FloatTensor(image_patch.transpose(0, 4, 3, 1, 2)).to(device)
                          prediction = net(X_test_image)
                          prediction = np.argmax(prediction.detach().cpu().numpy(), axis=1)
                          outputs[i][j] = prediction+1
            if i % 20 == 0:
                   print('... ... row ', i, ' handling ... ...')
```

```
<ipython-input-12-935d004c0cb5>:23: DeprecationWarning: Conversion of an array with ndim > 0 to a scalar is deprecated, and will error in future. Ensure
  outputs[i][j] = prediction+1
... ... row  0  handling ... ...
... ... row  20  handling ... ...
... ... row  40  handling ... ...
... ... row  60  handling ... ...
... ... row  80  handling ... ...
... ... row  100  handling ... ...
```

个人解读：用于对高光谱图像进行逐像素分类预测。实现了对整个高光谱图像的像素级分类，生成的`outputs`数组包含了每个像素的预测类别标签。

```
predict_image = spectral.imshow(classes = outputs.astype(int),figsize =(5,5))
```



个人解读：用于可视化逐像素分类的结果。生成一个图像窗口，展示分类结果，其中不同的颜色代表不同的地物类别，使用户能够直观地查看模型的分类效果。

【第二部分：问题总结】

- **训练HybridSN，然后多测试几次，会发现每次分类的结果都不一样，请思考为什么？**

Dropout 是一种正则化技术，通常在训练阶段使用，它通过随机忽略一部分神经元的输出，防止模型过拟合。在某些情况下，测试阶段也可能不正确地应用了 Dropout，而在测试阶段正确的做法是关闭 Dropout 或使用其推断模式（即：使用所有神经元并乘以一个缩放因子）。在测试阶段，如果不关闭 Dropout，会导致每次分类结果不一致。这是因为Dropout在每次前向传播时随机丢弃一部分神经元，增加了模型的不确定性。

- **如果想要进一步提升高光谱图像的分类性能，可以如何改进？**

混合不同类型的网络模型（如结合卷积神经网络和循环神经网络）可以增强对时序特征或更复杂的光谱-空间关系的捕捉。例如，**3D CNN + LSTM** 结构可以有效捕捉光谱数据的局部与全局特征。

高光谱图像中不同尺度的空间信息对分类至关重要。引入**多尺度卷积层**（Multi-scale CNN）能够在不同尺度上提取空间和光谱信息，增强模型的特征提取能力；高光谱图像的像素之间有复杂的空间关系，GNN可以通过建模这些空间邻域关系来提升分类效果。

- **depth-wise conv 和 分组卷积有什么区别与联系？**

分组卷积通过将输入特征图的通道分为若干组，每组通道独立执行卷积操作。这意味着每组通道都有其独立的卷积核，卷积计算在各组内部进行，而不会跨组进行。这种设计能够有效减少参数量和计算量。分组卷积最著名的应用是 AlexNet 中的设计，它将输入通道分为两个组，以适应同时使用两块 GPU 进行计算。

举例说明，假设输入有 6 个通道，分为 2 组，每组有 3 个通道，那么每组只与其对应的卷积核进行操作。如果每组使用 3 个卷积核，则每组的卷积计算会独立完成，并最终得到相应的输出。这样，分组卷积减少了卷积核的数量，使得计算更高效。

深度卷积则是一种极限情况，它将每个输入通道单独看作一个组，即每个通道只使用一个独立的卷积核进行卷积操作。这种方式显著减少了参数量，因为卷积不再像标准卷积那样涉及所有输入通道，而是通道之间独立进行。深度卷积通常与**逐点卷积（Point-wise Convolution）*结合使用，形成*深度可分离卷积（Depthwise Separable Convolution）**。深度可分离卷积通过先对每个通道进行深度卷积，再使用 1x1 卷积进行跨通道融合，大幅降低了计算复杂度，同时保留了对特征的提取能力。、

- **SENet 的注意力是不是可以加在空间位置上？**

SENet（Squeeze-and-Excitation Networks） 的注意力机制原本是设计用来增强特征图的通道维度上的表示能力，通过对每个通道的重要性进行自适应权重调整，从而提升网络的表示能力。但 SENet 的机制是基于通道注意力的，即它对每个通道进行全局池化，计算每个通道的权重，最终对通道维度进行加权，而**不直接关注空间位置上的注意力。**

- **在 ShuffleNet 中，通道的 shuffle 如何用代码实现？**

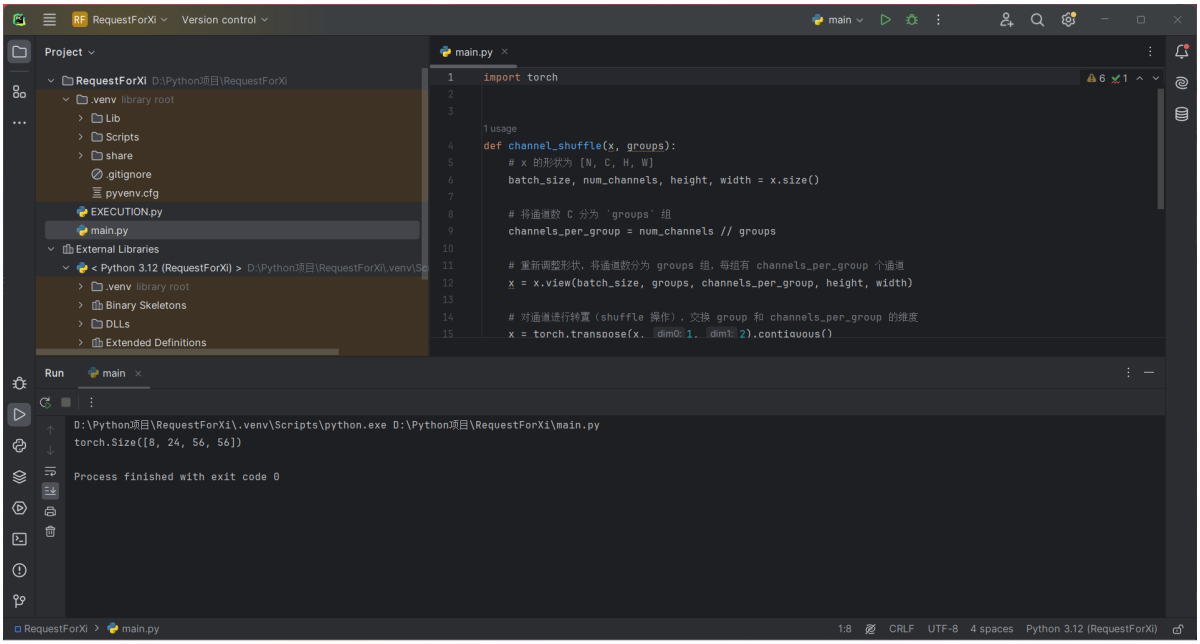
具体步骤如下：

1. 将通道划分为若干组。
2. 将每组通道的顺序打乱，进行跨组混合。
3. 按照打乱后的顺序重新排列通道。

假设我们有一个形状为 `[N, C, H, W]` 的张量，其中 `N` 是批量大小，`C` 是通道数，`H` 和 `W` 分别是特征图的高度和宽度。如果我们将通道数 `C` 分为 `g` 组，Shuffle 操作可以通过以下代码实现：

```
import torch

def channel_shuffle(x, groups):
    # x 的形状为 [N, C, H, W]
    batch_size, num_channels, height, width = x.size()

    # 将通道数 C 分为 `groups` 组
    channels_per_group = num_channels // groups

    # 重新调整形状，将通道数分为 groups 组，每组有 channels_per_group 个通道
    x = x.view(batch_size, groups, channels_per_group, height, width)

    # 对通道进行转置（shuffle 操作），交换 group 和 channels_per_group 的维度
    x = torch.transpose(x, 1, 2).contiguous()

    # 将调整后的张量再恢复回原来的形状 [N, C, H, W]
    x = x.view(batch_size, num_channels, height, width)

    return x

x = torch.randn(8, 24, 56, 56)  # 假设批量大小为 8，通道数为 24，特征图大小为 56x56
groups = 4
output = channel_shuffle(x, groups)
print(output.shape)  # 输出形状仍然为 [8, 24, 56, 56]
```

1. **调整形状**：首先通过 `view` 方法将输入张量重新调整形状，将通道分为 `groups` 组，每组 `channels_per_group` 个通道。
2. **转置通道维度**：通过 `torch.transpose` 交换组的维度和每组通道的维度，达到打乱通道顺序的效果。
3. **恢复形状**：最后使用 `view` 将张量恢复到原来的形状 `[N, C, H, W]`。



二、问题总结与体会

问题总结：

在训练模型阶段运行了2个多小时，中途中断一次。原因：数据集较大，训练轮数多。

体会:

在完成这次实验的过程中，我深刻体会到了深度学习模型设计的精妙之处，尤其是在处理高光谱图像分类这类复杂任务时。通过实验，我不仅加深了对MobileNet和ShuffleNet这两种高效网络架构的理解，还学会了如何通过调整网络结构和参数来优化模型性能。MobileNet的深度可分离卷积和ShuffleNet的通道混洗技术，让我认识到在保持计算效率的同时，也能实现出色的特征提取能力。这些体会不仅提升了我的技术实践能力，也为我未来在深度学习领域的研究和应用打下了坚实的基础。