姓名和学号?	**张心顺，23120014029**
本实验属于哪门课程?	中国海洋大学25秋《软件工程原理与实践》
实验名称?	实验4：MobileNet & ShuffleNet
发布地址?	LoongGold/2025FALLSEPP

一、实验内容

阅读论文《HybridSN: Exploring 3-D–2-DCNN Feature Hierarchy for Hyperspectral Image Classification》，思考3D卷积和2D卷积的区别。并阅读HybridSN的代码。

把代码敲入 Colab 运行，网络部分需要自己完成。

代码与执行结果如下：

其中网络部分已经完成设计。

```
! 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
```

```
--2025-11-03 13:10:34--
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]
--2025-11-03 13:10:34--
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   510KB/s    in 9.9s

2025-11-03 13:10:44 (589 KB/s) - ‘Indian_pines_corrected.mat’ saved
[5953527/5953527]

URL transformed to HTTPS due to an HSTS policy
--2025-11-03 13:10:44--
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
```

```
2025-11-03 13:10:45 (524 MB/s) - ‘Indian_pines_gt.mat’ saved [1125/1125]

Collecting spectral
  Downloading spectral-0.24-py3-none-any.whl.metadata (1.3 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages
(from spectral) (2.0.2)
Downloading spectral-0.24-py3-none-any.whl (249 kB)
[2K   [90m━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━[0m [32m249.0/249.0 kB[0m [31m7.3
MB/s[0m eta [36m0:00:00[0m
[?25hInstalling collected packages: spectral
Successfully installed spectral-0.24
```

```python
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
```

```python
class_num = 16

# 网络部分设计
class HybridSN(nn.Module):
  def __init__(self):
    super(HybridSN, self).__init__()

    self.conv3d_1 = nn.Conv3d(1, 8, kernel_size=(7, 3, 3))
    self.conv3d_2 = nn.Conv3d(8, 16, kernel_size=(5, 3, 3))
    self.conv3d_3 = nn.Conv3d(16, 32, kernel_size=(3, 3, 3))

    self.conv2d_1 = nn.Conv2d(576, 64, kernel_size=(3, 3))

    self.fc1 = nn.Linear(18496, 256)
    self.fc2 = nn.Linear(256, 128)
    self.fc3 = nn.Linear(128, class_num)

    self.dropout = nn.Dropout(0.4)

  def forward(self, x):
    x = F.relu(self.conv3d_1(x))
    x = F.relu(self.conv3d_2(x))
    x = F.relu(self.conv3d_3(x))

    batch_size, channels, depth, height, width = x.shape
    x = x.view(batch_size, channels * depth, height, width)
```

```
    x = F.relu(self.conv2d_1(x))

    x = x.view(batch_size, -1)

    x = F.relu(self.fc1(x))
    x = self.dropout(x)
    x = F.relu(self.fc2(x))
    x = self.dropout(x)
    x = self.fc3(x)

    return x

# 随机输入，测试网络结构是否通
# x = torch.randn(1, 1, 30, 25, 25)
# net = HybridSN()
# y = net(x)
# print(y.shape)
```

```
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

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

def createImageCubes(X, y, windowSize=5, removeZeroLabels = True):

    margin = int((windowSize - 1) / 2)
    zeroPaddedX = padWithZeros(X, margin=margin)

    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)
    return X_train, X_test, y_train, y_test
```

```
class_num = 16
X = sio.loadmat('Indian_pines_corrected.mat')['indian_pines_corrected']
y = sio.loadmat('Indian_pines_gt.mat')['indian_pines_gt']

test_ratio = 0.90

patch_size = 25

pca_components = 30

print('Hyperspectral data shape: ', X.shape)
print('Label shape: ', y.shape)

print('\n... ... PCA tranformation ... ...')
X_pca = applyPCA(X, numComponents=pca_components)
print('Data shape after PCA: ', X_pca.shape)

print('\n... ... create data cubes ... ...')
X_pca, y = createImageCubes(X_pca, y, windowSize=patch_size)
print('Data cube X shape: ', X_pca.shape)
print('Data cube y shape: ', y.shape)

print('\n... ... create train & test data ... ...')
Xtrain, Xtest, ytrain, ytest = splitTrainTestSet(X_pca, y, test_ratio)
print('Xtrain shape: ', Xtrain.shape)
print('Xtest  shape: ', Xtest.shape)

Xtrain = Xtrain.reshape(-1, patch_size, patch_size, pca_components, 1)
Xtest  = Xtest.reshape(-1, patch_size, patch_size, pca_components, 1)
print('before transpose: Xtrain shape: ', Xtrain.shape)
print('before transpose: Xtest  shape: ', Xtest.shape)

Xtrain = Xtrain.transpose(0, 4, 3, 1, 2)
Xtest  = Xtest.transpose(0, 4, 3, 1, 2)
print('after transpose: Xtrain shape: ', Xtrain.shape)
print('after transpose: Xtest  shape: ', Xtest.shape)


""" Training dataset"""
class TrainDS(torch.utils.data.Dataset):
    def __init__(self):
        self.len = Xtrain.shape[0]
        self.x_data = torch.FloatTensor(Xtrain)
        self.y_data = torch.LongTensor(ytrain)
    def __getitem__(self, index):

        return self.x_data[index], self.y_data[index]
    def __len__(self):

        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):

        return self.x_data[index], self.y_data[index]
    def __len__(self):

        return self.len

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)
```

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

net = HybridSN().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)

total_loss = 0
for epoch in range(100):
    for i, (inputs, labels) in enumerate(train_loader):
        inputs = inputs.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
```

```
        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: 1]   [loss avg: 21.3324]   [current loss: 2.6479]
[Epoch: 2]   [loss avg: 20.5882]   [current loss: 2.5331]
[Epoch: 3]   [loss avg: 20.0198]   [current loss: 2.3684]
[Epoch: 4]   [loss avg: 19.4399]   [current loss: 2.2042]
[Epoch: 5]   [loss avg: 18.8794]   [current loss: 1.8831]
[Epoch: 6]   [loss avg: 18.2142]   [current loss: 1.7925]
[Epoch: 7]   [loss avg: 17.4348]   [current loss: 1.4763]
[Epoch: 8]   [loss avg: 16.5592]   [current loss: 1.1821]
[Epoch: 9]   [loss avg: 15.6723]   [current loss: 1.0568]
[Epoch: 10]   [loss avg: 14.7469]   [current loss: 0.6681]
[Epoch: 11]   [loss avg: 13.8813]   [current loss: 0.6663]
[Epoch: 12]   [loss avg: 13.0805]   [current loss: 0.4532]
[Epoch: 13]   [loss avg: 12.3133]   [current loss: 0.3692]
[Epoch: 14]   [loss avg: 11.6070]   [current loss: 0.2495]
[Epoch: 15]   [loss avg: 10.9611]   [current loss: 0.1959]
[Epoch: 16]   [loss avg: 10.3543]   [current loss: 0.0862]
[Epoch: 17]   [loss avg: 9.8000]   [current loss: 0.1600]
[Epoch: 18]   [loss avg: 9.3025]   [current loss: 0.0743]
[Epoch: 19]   [loss avg: 8.8419]   [current loss: 0.0486]
[Epoch: 20]   [loss avg: 8.4205]   [current loss: 0.0175]
[Epoch: 21]   [loss avg: 8.0460]   [current loss: 0.0629]
[Epoch: 22]   [loss avg: 7.6981]   [current loss: 0.0392]
[Epoch: 23]   [loss avg: 7.3823]   [current loss: 0.0368]
[Epoch: 24]   [loss avg: 7.0910]   [current loss: 0.0408]
[Epoch: 25]   [loss avg: 6.8183]   [current loss: 0.0228]
[Epoch: 26]   [loss avg: 6.5650]   [current loss: 0.0484]
[Epoch: 27]   [loss avg: 6.3411]   [current loss: 0.1151]
[Epoch: 28]   [loss avg: 6.1253]   [current loss: 0.0614]
[Epoch: 29]   [loss avg: 5.9247]   [current loss: 0.0564]
[Epoch: 30]   [loss avg: 5.7326]   [current loss: 0.0130]
[Epoch: 31]   [loss avg: 5.5580]   [current loss: 0.0641]
[Epoch: 32]   [loss avg: 5.3909]   [current loss: 0.0363]
[Epoch: 33]   [loss avg: 5.2378]   [current loss: 0.0382]
[Epoch: 34]   [loss avg: 5.0915]   [current loss: 0.0160]
[Epoch: 35]   [loss avg: 4.9525]   [current loss: 0.0148]
[Epoch: 36]   [loss avg: 4.8187]   [current loss: 0.0166]
[Epoch: 37]   [loss avg: 4.6970]   [current loss: 0.1800]
[Epoch: 38]   [loss avg: 4.5807]   [current loss: 0.0902]
[Epoch: 39]   [loss avg: 4.4722]   [current loss: 0.0132]
[Epoch: 40]   [loss avg: 4.3647]   [current loss: 0.0423]
[Epoch: 41]   [loss avg: 4.2622]   [current loss: 0.0180]
[Epoch: 42]   [loss avg: 4.1658]   [current loss: 0.0369]
[Epoch: 43]   [loss avg: 4.0728]   [current loss: 0.0218]
[Epoch: 44]   [loss avg: 3.9865]   [current loss: 0.0614]
[Epoch: 45]   [loss avg: 3.9013]   [current loss: 0.0116]
```

```
[Epoch: 46]   [loss avg: 3.8219]   [current loss: 0.0237]
[Epoch: 47]   [loss avg: 3.7439]   [current loss: 0.0231]
[Epoch: 48]   [loss avg: 3.6701]   [current loss: 0.0219]
[Epoch: 49]   [loss avg: 3.6013]   [current loss: 0.0090]
[Epoch: 50]   [loss avg: 3.5315]   [current loss: 0.0179]
[Epoch: 51]   [loss avg: 3.4649]   [current loss: 0.0092]
[Epoch: 52]   [loss avg: 3.4009]   [current loss: 0.0031]
[Epoch: 53]   [loss avg: 3.3405]   [current loss: 0.0576]
[Epoch: 54]   [loss avg: 3.2825]   [current loss: 0.0208]
[Epoch: 55]   [loss avg: 3.2247]   [current loss: 0.0118]
[Epoch: 56]   [loss avg: 3.1691]   [current loss: 0.0138]
[Epoch: 57]   [loss avg: 3.1153]   [current loss: 0.0253]
[Epoch: 58]   [loss avg: 3.0635]   [current loss: 0.0389]
[Epoch: 59]   [loss avg: 3.0134]   [current loss: 0.0196]
[Epoch: 60]   [loss avg: 2.9654]   [current loss: 0.0236]
[Epoch: 61]   [loss avg: 2.9187]   [current loss: 0.0028]
[Epoch: 62]   [loss avg: 2.8753]   [current loss: 0.0400]
[Epoch: 63]   [loss avg: 2.8323]   [current loss: 0.0363]
[Epoch: 64]   [loss avg: 2.7910]   [current loss: 0.0155]
[Epoch: 65]   [loss avg: 2.7504]   [current loss: 0.0159]
[Epoch: 66]   [loss avg: 2.7098]   [current loss: 0.0113]
[Epoch: 67]   [loss avg: 2.6706]   [current loss: 0.0013]
[Epoch: 68]   [loss avg: 2.6324]   [current loss: 0.0243]
[Epoch: 69]   [loss avg: 2.5951]   [current loss: 0.0005]
[Epoch: 70]   [loss avg: 2.5610]   [current loss: 0.0030]
[Epoch: 71]   [loss avg: 2.5275]   [current loss: 0.0934]
[Epoch: 72]   [loss avg: 2.4933]   [current loss: 0.0072]
[Epoch: 73]   [loss avg: 2.4602]   [current loss: 0.0267]
[Epoch: 74]   [loss avg: 2.4277]   [current loss: 0.0060]
[Epoch: 75]   [loss avg: 2.3959]   [current loss: 0.0008]
[Epoch: 76]   [loss avg: 2.3659]   [current loss: 0.0001]
[Epoch: 77]   [loss avg: 2.3360]   [current loss: 0.0013]
[Epoch: 78]   [loss avg: 2.3079]   [current loss: 0.0070]
[Epoch: 79]   [loss avg: 2.2798]   [current loss: 0.0010]
[Epoch: 80]   [loss avg: 2.2514]   [current loss: 0.0008]
[Epoch: 81]   [loss avg: 2.2241]   [current loss: 0.0022]
[Epoch: 82]   [loss avg: 2.1974]   [current loss: 0.0001]
[Epoch: 83]   [loss avg: 2.1725]   [current loss: 0.0017]
[Epoch: 84]   [loss avg: 2.1482]   [current loss: 0.0032]
[Epoch: 85]   [loss avg: 2.1240]   [current loss: 0.0055]
[Epoch: 86]   [loss avg: 2.1001]   [current loss: 0.0214]
[Epoch: 87]   [loss avg: 2.0763]   [current loss: 0.0019]
[Epoch: 88]   [loss avg: 2.0538]   [current loss: 0.0234]
[Epoch: 89]   [loss avg: 2.0314]   [current loss: 0.0009]
[Epoch: 90]   [loss avg: 2.0091]   [current loss: 0.0107]
[Epoch: 91]   [loss avg: 1.9876]   [current loss: 0.0015]
[Epoch: 92]   [loss avg: 1.9671]   [current loss: 0.0101]
[Epoch: 93]   [loss avg: 1.9475]   [current loss: 0.0011]
[Epoch: 94]   [loss avg: 1.9277]   [current loss: 0.0048]
[Epoch: 95]   [loss avg: 1.9097]   [current loss: 0.0579]
[Epoch: 96]   [loss avg: 1.8915]   [current loss: 0.0735]
[Epoch: 97]   [loss avg: 1.8747]   [current loss: 0.0022]
[Epoch: 98]   [loss avg: 1.8563]   [current loss: 0.0003]
[Epoch: 99]   [loss avg: 1.8378]   [current loss: 0.0015]
[Epoch: 100]   [loss avg: 1.8199]   [current loss: 0.0029]
Finished Training
```

```
count = 0

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) )

classification = classification_report(ytest, y_pred_test, digits=4)
print(classification)
```

```
              precision    recall  f1-score   support

         0.0     0.8810    0.9024    0.8916        41
         1.0     0.9807    0.9486    0.9644      1285
         2.0     0.9535    0.9880    0.9704       747
         3.0     0.9643    0.8873    0.9242       213
         4.0     0.9758    0.9264    0.9505       435
         5.0     0.9574    0.9909    0.9738       657
         6.0     0.9259    1.0000    0.9615        25
         7.0     1.0000    0.9837    0.9918       430
         8.0     0.6400    0.8889    0.7442        18
         9.0     0.9869    0.9474    0.9668       875
        10.0     0.9547    0.9919    0.9729      2210
        11.0     0.9430    0.9607    0.9518       534
        12.0     0.9670    0.9514    0.9591       185
        13.0     0.9913    0.9965    0.9939      1139
        14.0     0.9938    0.9164    0.9535       347
        15.0     0.9333    0.8333    0.8805        84

    accuracy                         0.9685      9225
   macro avg     0.9405    0.9446    0.9407      9225
weighted avg     0.9691    0.9685    0.9684      9225
```

```
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
```

```
    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','Fa
llow_smooth',

 'Stubble','Celery','Grapes_untrained','Soil_vinyard_develop','Corn_senesced_gree
n_weeds',

 'Lettuce_romaine_4wk','Lettuce_romaine_5wk','Lettuce_romaine_6wk','Lettuce_romai
ne_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
```

```
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')
```

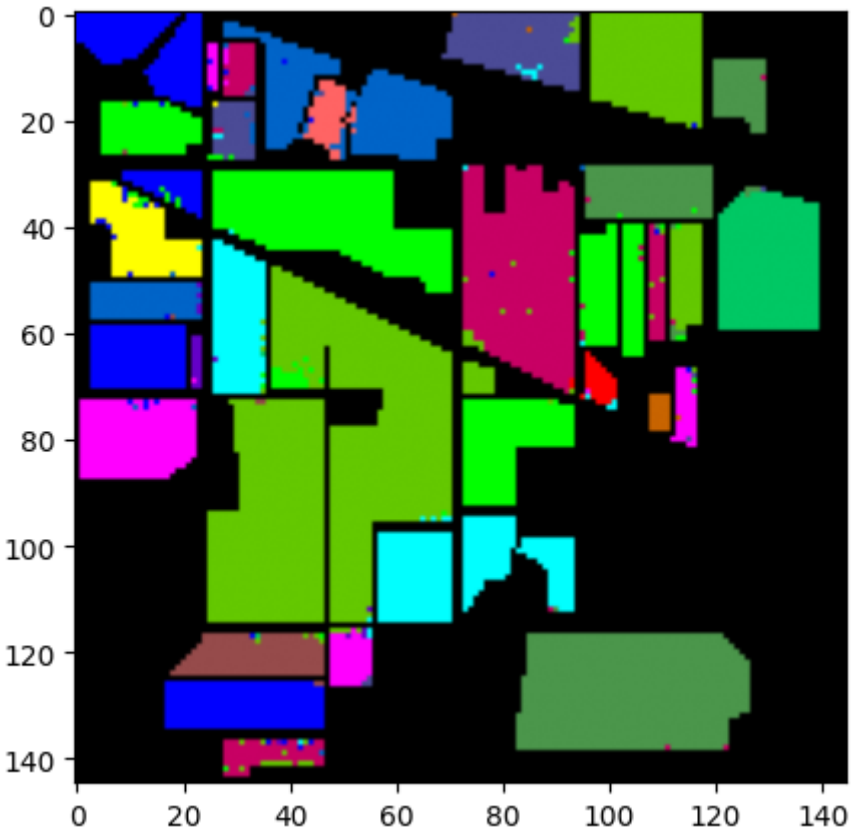
```
    x_file.write('{} Average accuracy (%)'.format(aa))
    x_file.write('\n')
    x_file.write('\n')
    x_file.write('{}'.format(classification))
    x_file.write('\n')
    x_file.write('{}'.format(confusion))
```

```
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)

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 ... ...')
```

```
/tmp/ipython-input-4282682281.py:21: DeprecationWarning: Conversion of an array
with ndim > 0 to a scalar is deprecated, and will error in future. Ensure you
extract a single element from your array before performing this operation.
(Deprecated NumPy 1.25.)
  outputs[i][j] = prediction+1
```

```
... ... row  0  handling ... ...
... ... row  20  handling ... ...
... ... row  40  handling ... ...
... ... row  60  handling ... ...
... ... row  80  handling ... ...
... ... row  100  handling ... ...
... ... row  120  handling ... ...
... ... row  140  handling ... ...
```

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



二、问题总结与体会

● 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?

思考的原因：

1. 神经网络的参数初始值随机生成时，不同的初始化会导致结果不同。
2. 训练时，shuffle的顺序不同，模型每次训练接收的数据不同，使得结果不同。
3. Dropout层在训练时会随机丢弃神经元，导致结果不同。

● 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?

改进方式：

1. 对数据进行旋转，缩放，色彩调整等操作。
2. 在模型架构中引入注意力机制，使用残差连接。或尝试Transformer架构。
3. 设计更好的学习率调整机制。

● depth-wise conv 和 分组卷积有什么区别与联系?

联系：

1. 都是标准卷积的一种分解形式，都能减少计算量和参数量。
2. 后者是前者的泛化形式。

区别：

1. 前者的每个输入通道独立计算卷积，产生相同数量的输出通道。
2. 后者将输入通道分为几组，每组内部进行卷积计算，输出通道数量等于组数乘每组的输出通道数。

● SENet 的注意力是不是可以加在空间位置上?

可以。

● 在 ShuffleNet 中，通道的 shuffle 如何用代码实现?

代码如下，代码解释为，将通道的维数拆分成组数和每组通道数，变换组和通道维度的位置，传递组间信息。恢复原始形状，完成通道的重新排列。

```
def channel_shuffle(x, groups):

    batch_size, channels, height, width = x.size()

    channels_per_group = channels // groups
    x = x.view(batch_size, groups, channels_per_group, height, width)

    x = x.transpose(1, 2).contiguous()

    x = x.view(batch_size, -1, height, width)

    return x
```