Z_9_BCNN_2D_2D

October 12, 2022

1 Date: 7 2022

2 Method: 2D 2D BCNN

3 Data: Pavia

4 Results v.03

[ ]:
```
# Libraries
import pandas as pd
import numpy as np
import seaborn as sn

import keras
from keras.layers import Conv2D, Conv3D, Flatten, Dense, Reshape,␣
↪BatchNormalization, Lambda
from keras.layers import Dropout, Input
from keras.models import Model
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint
from keras.utils import np_utils

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 time

from plotly.offline import init_notebook_mode
import numpy as np

import matplotlib.pyplot as plt
import scipy.io as sio
import os
import spectral
```

```
import tensorflow as tf
import tensorflow_probability as tfp
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,␣
↪Dropout
from tensorflow.keras.layers import Input, Dense, Conv1D, MaxPooling1D,␣
↪Dropout, Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical

tfd = tfp.distributions
tfpl = tfp.layers
```

[ ]:
```
## VARIABLES

test_ratio = 0.3
test_val_ratio=0.6

train_ratio = 1-test_ratio
#train_val_ratio = 0.8


windowSize = 9  #  25
dimReduction = 80 # dimReduction

drop = 0.4
```

[ ]:
```
# Split Data

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
```

[ ]:
```
# PCA
def applyPCA(X, numComponents): # numComponents=64
    newX = np.reshape(X, (-1, X.shape[2]))
    print(newX.shape)
    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, pca, pca.explained_variance_ratio_
```

[ ]:
```
# pading With Zeros
def padWithZeros(X, margin=2):
    newX = np.zeros((X.shape[0] + 2 * margin, X.shape[1] + 2* margin, X.
↪shape[2]),dtype="float16")
    x_offset = margin
```

```
    y_offset = margin
    newX[x_offset:X.shape[0] + x_offset, y_offset:X.shape[1] + y_offset, :] = X
    return newX
```

[ ]:

```
# Split the hyperspectral image into patches of size windowSize-by-windowSize␣
↪pixels
def Patches_Creating(X, y, windowSize, removeZeroLabels = True):  #␣
↪windowSize=15, 25
    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]),dtype="float16")
    patchesLabels = np.zeros((X.shape[0] * X.shape[1]),dtype="float16")
    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
```

[ ]:

```
# channel_wise_shift
def channel_wise_shift(X,numComponents):
    X_copy = np.zeros((X.shape[0] , X.shape[1], X.shape[2]))
    half = int(numComponents/2)
    for i in range(0,half-1):
        X_copy[:,:,i] = X[:,:,(half-i)*2-1]
    for i in range(half,numComponents):
        X_copy[:,:,i] = X[:,:,(i-half)*2]
    X = X_copy
    return X
```

[ ]:

```
# Read data
from scipy.io import loadmat

def read_HSI():
  X = loadmat('PaviaU.mat')['paviaU']
  y = loadmat('PaviaU_gt.mat')['paviaU_gt']
  print(f"X shape: {X.shape}\ny shape: {y.shape}")
  return X, y
```

```
X, y = read_HSI()
```

```
X shape: (610, 340, 103)
y shape: (610, 340)
```

[ ]:
```
# Load and reshape data for training
X0, y0 = read_HSI()
#X=X0
#y=y0

InputShape=(windowSize, windowSize, dimReduction)

#X, y = loadData(dataset) channel_wise_shift
X1,pca,ratio = applyPCA(X0,numComponents=dimReduction)
X2_shifted = channel_wise_shift(X1,dimReduction) # channel-wise shift
#X2=X1

#print(f"X0 shape: {X0.shape}\ny0 shape: {y0.shape}")
#print(f"X1 shape: {X1.shape}\nX2 shape: {X2.shape}")

X3, y3 = Patches_Creating(X2_shifted, y0, windowSize=windowSize)
Xtrain, Xtest, ytrain, ytest = splitTrainTestSet(X3, y3, test_ratio)


print(f"Xtrain shape: {Xtrain.shape}\nytrain shape : {ytrain.shape}")
#print(f"Xtest shape: {Xtest.shape}\nytest shape : {ytest.shape}")
```

```
X shape: (610, 340, 103)
y shape: (610, 340)
(207400, 103)
Xtrain shape: (29943, 9, 9, 80)
ytrain shape : (29943,)
```

[ ]:
```
# split data for Training and Testing
Xtrain = Xtrain.reshape(-1, windowSize,windowSize, dimReduction)
ytrain = np_utils.to_categorical(ytrain)

#Xvalid, Xtest, yvalid, ytest = splitTrainTestSet(Xtest, ytest,␣
 ↪(test_ratio-train_ratio/train_val_ratio)/test_ratio)
Xvalid, Xtest, yvalid, ytest = splitTrainTestSet(Xtest, ytest, test_val_ratio)

Xvalid = Xvalid.reshape(-1, windowSize,windowSize, dimReduction)
yvalid = np_utils.to_categorical(yvalid)
```

[ ]:
```
# Function to define the spike and slab distribution
# => To be used in prior
```

```
def spike_and_slab(event_shape, dtype):
    distribution = tfd.Mixture(
        cat=tfd.Categorical(probs=[0.5, 0.5]),
        components=[
            tfd.Independent(tfd.Normal(
                loc=tf.zeros(event_shape, dtype=dtype),
                scale=1.0*tf.ones(event_shape, dtype=dtype)),
                            reinterpreted_batch_ndims=1),
            tfd.Independent(tfd.Normal(
                loc=tf.zeros(event_shape, dtype=dtype),
                scale=10.0*tf.ones(event_shape, dtype=dtype)),
                            reinterpreted_batch_ndims=1)],
    name='spike_and_slab')
    return distribution
```

[ ]:
```
# Testing Model_ N01
from tensorflow.keras.optimizers import RMSprop

def nll(y_true, y_pred):
    return -y_pred.log_prob(y_true)
```

[ ]:
```
#Testing Model_ N01
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,␣
 ↪TensorBoard

def negative_log_likelihood(y_true, y_pred):
    return -y_pred.log_prob(y_true)
```

Model_ N02

[ ]:
```
# Testing Model_ N02
# Basian deep neural network (BCNN)
divergence_fn = lambda q,p,_:tfd.kl_divergence(q,p)/len(Xtrain)    #3457

# BCNN model
#
model_bayes = Sequential([
    # Statistical 2D conv
    tfpl.Convolution2DReparameterization(input_shape=InputShape, filters=8,␣
 ↪kernel_size=2, activation='relu',
                                           kernel_prior_fn = tfpl.
 ↪default_multivariate_normal_fn,
                                           kernel_posterior_fn=tfpl.
 ↪default_mean_field_normal_fn(is_singular=False),
                                           kernel_divergence_fn = divergence_fn,
```

```
                                                bias_prior_fn = tfpl.
↪default_multivariate_normal_fn,
                                                bias_posterior_fn=tfpl.
↪default_mean_field_normal_fn(is_singular=False),
                                                bias_divergence_fn = divergence_fn),
    MaxPooling2D(2,1),
    Conv2D(32, (2,2), activation='relu'),
    MaxPooling2D(2,1),
    Conv2D(16, (2,2), activation='relu'),
    MaxPooling2D(2,1),
    Conv2D(16, (2,2), activation='relu'),
    MaxPooling2D(1,1),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.2),
    # Statistical Dense-
    tfpl.DenseReparameterization(units=tfpl.OneHotCategorical.params_size(9),␣
↪activation=None,
                                        kernel_prior_fn = tfpl.
↪default_multivariate_normal_fn,
                                        kernel_posterior_fn=tfpl.
↪default_mean_field_normal_fn(is_singular=False),
                                        kernel_divergence_fn = divergence_fn,
                                        bias_prior_fn = tfpl.
↪default_multivariate_normal_fn,
                                        bias_posterior_fn=tfpl.
↪default_mean_field_normal_fn(is_singular=False),
                                        bias_divergence_fn = divergence_fn
                                    ),
    # output-
    tfpl.OneHotCategorical(9)

])
model_bayes.summary()
```

```
Model: "sequential_15"
```

```
c:\Users\kifah\anaconda3\lib\site-
packages\tensorflow_probability\python\layers\util.py:95: UserWarning:

`layer.add_variable` is deprecated and will be removed in a future version.
Please use the `layer.add_weight()` method instead.

c:\Users\kifah\anaconda3\lib\site-
packages\tensorflow_probability\python\layers\util.py:105: UserWarning:

`layer.add_variable` is deprecated and will be removed in a future version.
Please use the `layer.add_weight()` method instead.
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_reparameterization_1  (None, 8, 8, 8)          5136
 5 (Conv2DReparameterization
 )

 max_pooling2d_60 (MaxPoolin  (None, 7, 7, 8)          0
 g2D)

 conv2d_45 (Conv2D)          (None, 6, 6, 32)          1056

 max_pooling2d_61 (MaxPoolin  (None, 5, 5, 32)         0
 g2D)

 conv2d_46 (Conv2D)          (None, 4, 4, 16)          2064

 max_pooling2d_62 (MaxPoolin  (None, 3, 3, 16)         0
 g2D)

 conv2d_47 (Conv2D)          (None, 2, 2, 16)          1040

 max_pooling2d_63 (MaxPoolin  (None, 2, 2, 16)         0
 g2D)

 flatten_15 (Flatten)        (None, 64)                0

 dense_15 (Dense)            (None, 512)               33280

 dropout_15 (Dropout)        (None, 512)               0

 dense_reparameterization_15  (None, 9)                9234
  (DenseReparameterization)

 one_hot_categorical_15 (One  ((None, 9),              0
 HotCategorical)              (None, 9))

=================================================================
Total params: 51,810
Trainable params: 51,810
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
# Testing Model_ NO2
# Comiple
```

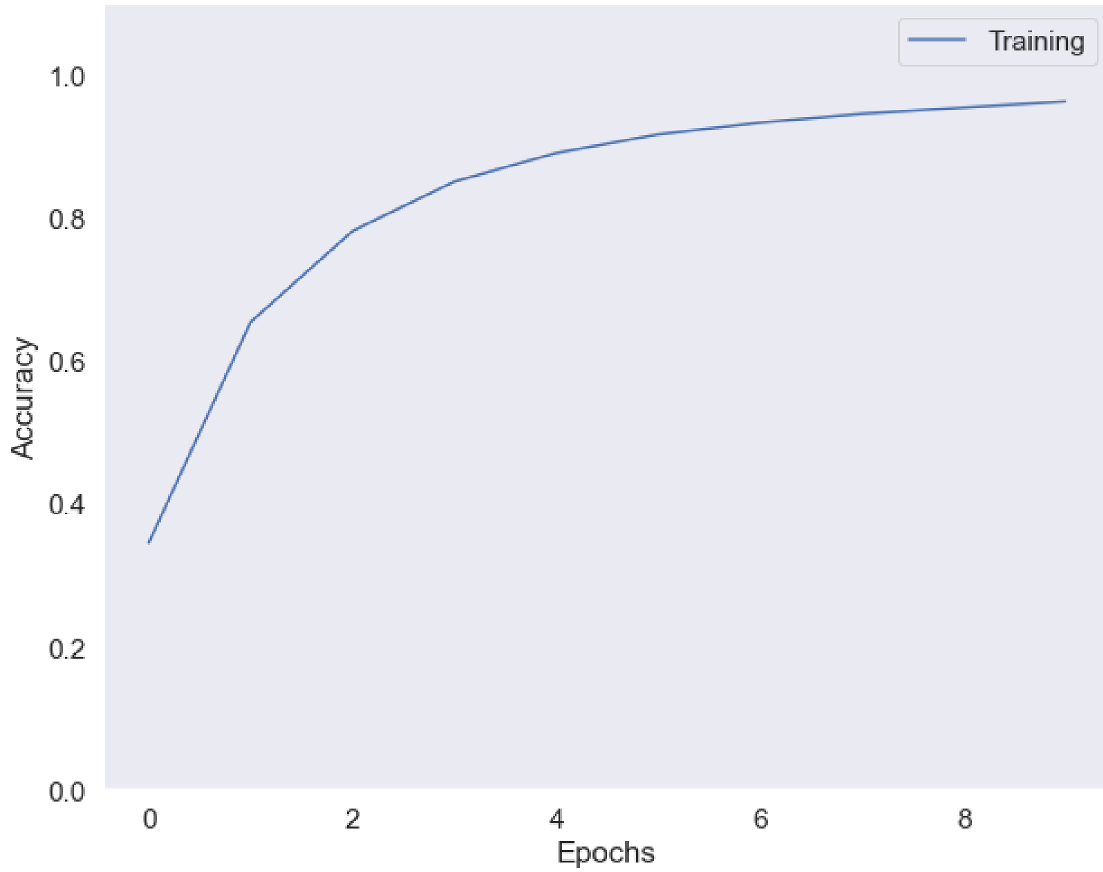
```
model_bayes.compile(loss = negative_log_likelihood,
              optimizer = Adam(learning_rate=0.001), #0.005
              metrics = ['accuracy'],
              experimental_run_tf_function = False)
```

[ ]:
```
# Testing Model_ NO2
# Train
hist = model_bayes.fit(Xtrain,
                       ytrain,
                       epochs = 10,
                       batch_size = 512 ,
                       validation_data = (Xvalid, yvalid)    )
```

```
Epoch 1/10
59/59 [==============================] - 5s 61ms/step - loss: 2.1180 - accuracy:
0.3452 - val_loss: 1.6012 - val_accuracy: 0.5371
Epoch 2/10
59/59 [==============================] - 3s 54ms/step - loss: 1.3280 - accuracy:
0.6545 - val_loss: 1.0661 - val_accuracy: 0.7551
Epoch 3/10
59/59 [==============================] - 3s 48ms/step - loss: 1.0212 - accuracy:
0.7820 - val_loss: 0.8994 - val_accuracy: 0.8188
Epoch 4/10
59/59 [==============================] - 3s 45ms/step - loss: 0.8800 - accuracy:
0.8514 - val_loss: 0.8271 - val_accuracy: 0.8798
Epoch 5/10
59/59 [==============================] - 3s 44ms/step - loss: 0.7989 - accuracy:
0.8913 - val_loss: 0.7493 - val_accuracy: 0.9110
Epoch 6/10
59/59 [==============================] - 3s 47ms/step - loss: 0.7469 - accuracy:
0.9174 - val_loss: 0.7043 - val_accuracy: 0.9330
Epoch 7/10
59/59 [==============================] - 3s 51ms/step - loss: 0.7099 - accuracy:
0.9340 - val_loss: 0.6895 - val_accuracy: 0.9427
Epoch 8/10
59/59 [==============================] - 3s 50ms/step - loss: 0.6874 - accuracy:
0.9462 - val_loss: 0.6758 - val_accuracy: 0.9527
Epoch 9/10
59/59 [==============================] - 3s 53ms/step - loss: 0.6636 - accuracy:
0.9549 - val_loss: 0.6554 - val_accuracy: 0.9532
Epoch 10/10
59/59 [==============================] - 3s 53ms/step - loss: 0.6416 - accuracy:
0.9636 - val_loss: 0.6659 - val_accuracy: 0.9511
```

[ ]:
```
# Plot accuracy
plt.figure(figsize=(10,8))
plt.ylim(0,1.1)
plt.grid()
```

```python
plt.plot(hist.history['accuracy'])
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Training','Validation'])
plt.savefig("acc_curve.pdf")
plt.show()
```



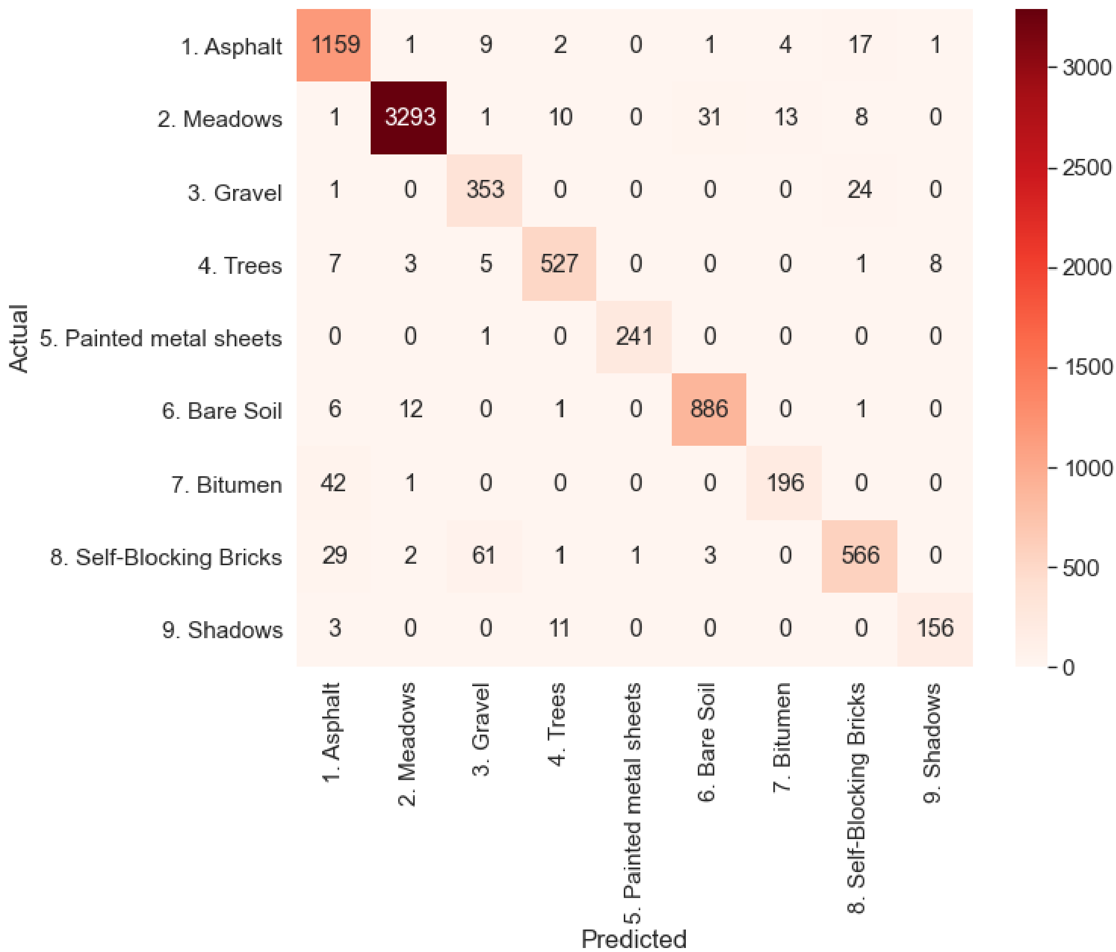
[ ]:
```python
# 9 classes
names = ['1. Asphalt', '2. Meadows', '3. Gravel', '4. Trees',
                 '5. Painted metal sheets','6. Bare Soil', '7. Bitumen',
                 '8. Self-Blocking Bricks', '9. Shadows']
```

[ ]:
```python
# confusion_matrix
Y_pred = model_bayes.predict(Xtest)
y_pred = np.argmax(Y_pred, axis=1)

confusion = confusion_matrix(ytest, y_pred)
df_cm = pd.DataFrame(confusion, columns=np.unique(names), index = np.
 ↪unique(names))
```

```
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'
plt.figure(figsize = (10,8))
sn.set(font_scale=1.4)#for label size
sn.heatmap(df_cm, cmap="Reds", annot=True,annot_kws={"size": 16}, fmt='d')
plt.savefig('cmap.png', dpi=300)
```

```
241/241 [==============================] - 1s 2ms/step
```



[ ]:
```
# average_acc
from operator import itemgetter
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((list_diag/ list_raw_sum))
    average_acc = np.mean(each_acc)
```

```
    return each_acc, average_acc
```

[ ]:
```
# average_acc

each_acc, aa = AA_andEachClassAccuracy(confusion)
print("accuracy for each:")
print (each_acc)

print("OA accuracy:")
print(aa)
```

```
accuracy for each:
[0.97068677 0.98093536 0.93386243 0.95644283 0.99586777 0.97792494
 0.82008368 0.85369532 0.91764706]
OA accuracy:
0.934127352191321
```

[ ]:
```
# classification_report
print(classification_report(ytest, y_pred, target_names = names, digits = 3))
```

```
                        precision    recall  f1-score   support

            1. Asphalt      0.929     0.971     0.949      1194
            2. Meadows      0.994     0.981     0.988      3357
             3. Gravel      0.821     0.934     0.874       378
              4. Trees      0.955     0.956     0.956       551
5. Painted metal sheets      0.996     0.996     0.996       242
          6. Bare Soil      0.962     0.978     0.970       906
            7. Bitumen      0.920     0.820     0.867       239
8. Self-Blocking Bricks      0.917     0.854     0.884       663
            9. Shadows      0.945     0.918     0.931       170

              accuracy                          0.958      7700
             macro avg      0.938     0.934     0.935      7700
          weighted avg      0.959     0.958     0.958      7700
```

[ ]:
```
# Calculation  the predicted image
def Patch(data,height_index,width_index):
    height_slice = slice(height_index, height_index+PATCH_SIZE)
    width_slice = slice(width_index, width_index+PATCH_SIZE)
    patch = data[height_slice, width_slice, :]
    return patch
```

[ ]:
```
# Calculation  the predicted image
PATCH_SIZE = windowSize
#X2_shifted, y0
```

```
#X,pca,ratio = applyPCA(X0,numComponents=40)

X = padWithZeros(X2_shifted, PATCH_SIZE//2) # PATCH_SIZE=15,  PATCH_SIZE//2=7
height = y0.shape[0]
width = y0.shape[1]
```

[ ]:
```
# the predicted image

outputs = np.zeros((height,width),dtype="float16")
outputs2 = np.zeros((height,width),dtype="float16")
for i in range(0,height,1):
    for j in range(0,width,1):
        target = int(y0[i,j])
        if target == 0 :
            image_patch=Patch(X,i,j)
            X_test_image = image_patch.reshape(1,image_patch.
 ↪shape[0],image_patch.shape[1], image_patch.shape[2]).astype('float32')
            prediction2 = (model_bayes.predict(X_test_image))
            prediction2 = np.argmax(prediction2, axis=1)
            outputs2[i][j] = prediction2+1
            print(i); print(j)
            #print(outputs2[i][j])
        else :
            image_patch=Patch(X,i,j)
            X_test_image = image_patch.reshape(1,image_patch.
 ↪shape[0],image_patch.shape[1], image_patch.shape[2]).astype('float32')
            prediction = (model_bayes.predict(X_test_image))
            prediction = np.argmax(prediction, axis=1)
            outputs[i][j] = prediction+1
            outputs2[i][j] = prediction+1
            #print("target=1")
            #print(outputs2[i][j])
```

[ ]:
```
import spectral
ground_truth = spectral.imshow(classes = y0,figsize␣
 ↪=(10,8),cmap='nipy_spectral'); plt.colorbar()



predict_image = spectral.imshow(classes = outputs.astype(int),figsize␣
 ↪=(7,7),cmap='nipy_spectral')
predict_image2 = spectral.imshow(classes = outputs2.astype(int),figsize =(7,7))

#spectral.save_rgb("predictions.png", outputs.astype(int), colors=spectral.
 ↪spy_colors)
#spectral.save_rgb("predictions2.png", outputs2.astype(int), colors=spectral.
 ↪spy_colors)
```

[ ]:

```

```