5 X_BCNN_Indians

May 19, 2023

1 Date: 5 2023

2 Method: Bayesian CNN

3 Data: Indian Pines

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 = 7  #  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('Indian_pines.mat')['indian_pines']
  y = loadmat('Indian_pines_gt.mat')['indian_pines_gt']
  print(f"X shape: {X.shape}\ny shape: {y.shape}")
  return X, y
```

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

```
X shape: (145, 145, 220)
y shape: (145, 145)
```

[ ]:
```
# 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)  # 5 for PAvia␣
↪Center
Xtrain, Xtest, ytrain, ytest = splitTrainTestSet(X3, y3, test_ratio)

ytest0=ytest
Xtest0=Xtest

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

```
X shape: (145, 145, 220)
y shape: (145, 145)
(21025, 220)
Xtrain shape: (7174, 7, 7, 80)
ytrain shape : (7174,)
```

[ ]:
```
# 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=16,␣
 ↪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),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.2),
    # Statistical Dense-
    tfpl.DenseReparameterization(units=tfpl.OneHotCategorical.params_size(16),␣
↪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(16)

])
model_bayes.summary()
```

```
Model: "sequential"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================

c:\Users\kifah\AppData\Local\Programs\Python\Python310\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.
  loc = add_variable_fn(
c:\Users\kifah\AppData\Local\Programs\Python\Python310\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.
```

```
 untransformed_scale = add_variable_fn(
conv2d_reparameterization (  (None, 6, 6, 16)         10272
Conv2DReparameterization)

max_pooling2d (MaxPooling2D  (None, 5, 5, 16)         0
)

conv2d (Conv2D)             (None, 4, 4, 32)          2080

max_pooling2d_1 (MaxPooling  (None, 3, 3, 32)         0
2D)

flatten (Flatten)           (None, 288)               0

dense (Dense)               (None, 512)               147968

dropout (Dropout)           (None, 512)               0

dense_reparameterization (D  (None, 16)               16416
enseReparameterization)

one_hot_categorical (OneHot  ((None, 16),             0
Categorical)                 (None, 16))

Conv2DReparameterization)

max_pooling2d (MaxPooling2D  (None, 5, 5, 16)         0
)

conv2d (Conv2D)             (None, 4, 4, 32)          2080

max_pooling2d_1 (MaxPooling  (None, 3, 3, 32)         0
2D)

flatten (Flatten)           (None, 288)               0

dense (Dense)               (None, 512)               147968

dropout (Dropout)           (None, 512)               0

dense_reparameterization (D  (None, 16)               16416
enseReparameterization)

one_hot_categorical (OneHot  ((None, 16),             0
Categorical)                 (None, 16))

=================================================================
```

```
Total params: 176,736
Trainable params: 176,736
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)
```

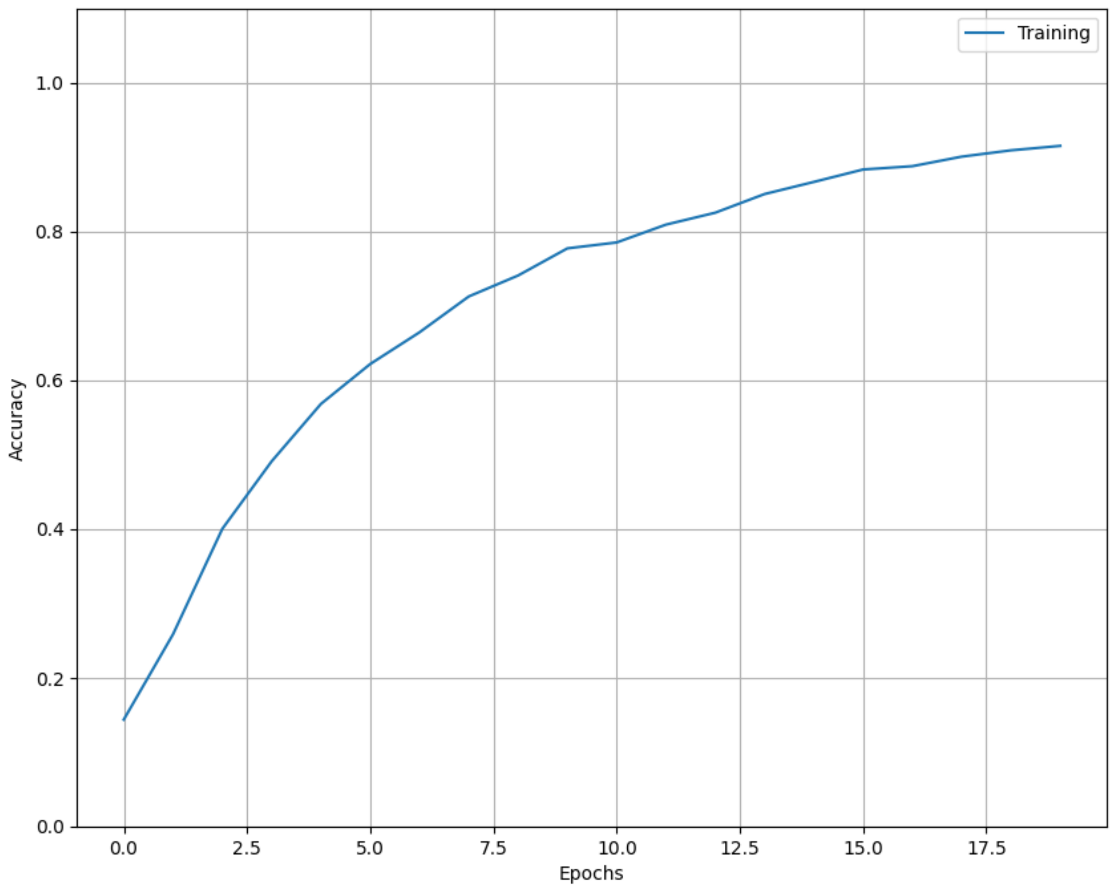
[ ]:
```
# Testing Model_ NO2
# Train
hist = model_bayes.fit(Xtrain,
                       ytrain,
                       epochs = 20,
                       batch_size = 512 ,
                       validation_data = (Xvalid, yvalid)    )
```

```
Epoch 1/20
15/15 [==============================] - 2s 56ms/step - loss: 7.9534 - accuracy:
0.1434 - val_loss: 6.8542 - val_accuracy: 0.1911
Epoch 2/20
15/15 [==============================] - 0s 29ms/step - loss: 6.6492 - accuracy:
0.2583 - val_loss: 6.2171 - val_accuracy: 0.3407
Epoch 3/20
15/15 [==============================] - 0s 28ms/step - loss: 6.1420 - accuracy:
0.4001 - val_loss: 5.9001 - val_accuracy: 0.4724
Epoch 4/20
15/15 [==============================] - 0s 29ms/step - loss: 5.8038 - accuracy:
0.4908 - val_loss: 5.5907 - val_accuracy: 0.5610
Epoch 5/20
15/15 [==============================] - 0s 28ms/step - loss: 5.6265 - accuracy:
0.5680 - val_loss: 5.4618 - val_accuracy: 0.6122
Epoch 6/20
15/15 [==============================] - 0s 31ms/step - loss: 5.4840 - accuracy:
0.6217 - val_loss: 5.3507 - val_accuracy: 0.6439
Epoch 7/20
15/15 [==============================] - 0s 30ms/step - loss: 5.3308 - accuracy:
0.6643 - val_loss: 5.2750 - val_accuracy: 0.7000
Epoch 8/20
15/15 [==============================] - 0s 31ms/step - loss: 5.2086 - accuracy:
0.7126 - val_loss: 5.1516 - val_accuracy: 0.7415
Epoch 9/20
15/15 [==============================] - 0s 30ms/step - loss: 5.1626 - accuracy:
0.7407 - val_loss: 5.1267 - val_accuracy: 0.7602
```

```
Epoch 10/20
15/15 [==============================] - 0s 31ms/step - loss: 5.0762 - accuracy:
0.7773 - val_loss: 5.0134 - val_accuracy: 0.7902
Epoch 11/20
15/15 [==============================] - 0s 30ms/step - loss: 5.0389 - accuracy:
0.7852 - val_loss: 4.9742 - val_accuracy: 0.8146
Epoch 12/20
15/15 [==============================] - 0s 29ms/step - loss: 4.9854 - accuracy:
0.8092 - val_loss: 4.9355 - val_accuracy: 0.8195
Epoch 13/20
15/15 [==============================] - 0s 29ms/step - loss: 4.9318 - accuracy:
0.8252 - val_loss: 4.9485 - val_accuracy: 0.8301
Epoch 14/20
15/15 [==============================] - 0s 29ms/step - loss: 4.8715 - accuracy:
0.8503 - val_loss: 4.8772 - val_accuracy: 0.8553
Epoch 15/20
15/15 [==============================] - 0s 31ms/step - loss: 4.8166 - accuracy:
0.8666 - val_loss: 4.8017 - val_accuracy: 0.8724
Epoch 16/20
15/15 [==============================] - 0s 33ms/step - loss: 4.7689 - accuracy:
0.8833 - val_loss: 4.8388 - val_accuracy: 0.8577
Epoch 17/20
15/15 [==============================] - 0s 32ms/step - loss: 4.7681 - accuracy:
0.8878 - val_loss: 4.7881 - val_accuracy: 0.8732
Epoch 18/20
15/15 [==============================] - 0s 29ms/step - loss: 4.7183 - accuracy:
0.9006 - val_loss: 4.7203 - val_accuracy: 0.8943
Epoch 19/20
15/15 [==============================] - 0s 27ms/step - loss: 4.7003 - accuracy:
0.9091 - val_loss: 4.7027 - val_accuracy: 0.9008
Epoch 20/20
15/15 [==============================] - 0s 24ms/step - loss: 4.6768 - accuracy:
0.9151 - val_loss: 4.6877 - val_accuracy: 0.9049
```

[ ]:

```
# Plot accuracy
plt.figure(figsize=(10,8))
plt.ylim(0,1.1)
plt.grid()
plt.plot(hist.history['accuracy'])
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['Training','Validation'])
plt.savefig("acc_curve.pdf")
plt.show()
```

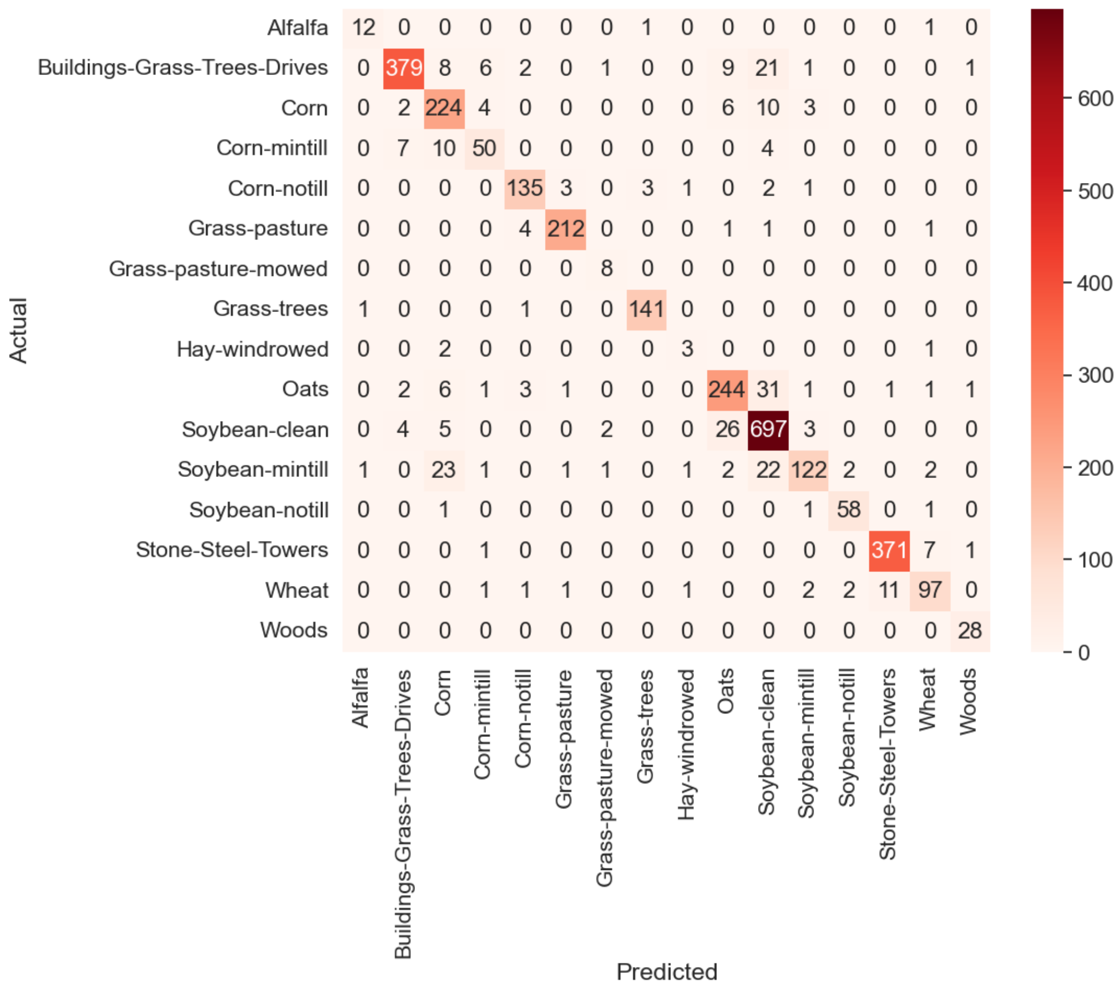


[ ]:
```
# 16 classes


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']
```

[ ]:
```
# confusion_matrix
Y_pred = model_bayes.predict(Xtest0)
y_pred = np.argmax(Y_pred, axis=1)

confusion = confusion_matrix(ytest0, 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'
```

```python
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)
```

```
97/97 [==============================] - 0s 2ms/step
```



[ ]:
```python
# 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
```

[ ]:
```python
# average_acc

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

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

```
accuracy for each:
[0.85714286 0.88551402 0.89959839 0.70422535 0.93103448 0.96803653
 1.         0.98601399 0.5        0.83561644 0.94572592 0.68539326
 0.95081967 0.97631579 0.8362069  1.        ]
OA accuracy:
0.872602724424328
```

[ ]:
```python
# classification_report
print(classification_report(ytest0, y_pred, target_names = names, digits = 3))
```

```
                              precision    recall  f1-score   support

                     Alfalfa      0.857     0.857     0.857        14
                 Corn-notill      0.962     0.886     0.922       428
                Corn-mintill      0.803     0.900     0.848       249
                        Corn      0.781     0.704     0.741        71
               Grass-pasture      0.925     0.931     0.928       145
                 Grass-trees      0.972     0.968     0.970       219
         Grass-pasture-mowed      0.667     1.000     0.800         8
               Hay-windrowed      0.972     0.986     0.979       143
                        Oats      0.500     0.500     0.500         6
              Soybean-notill      0.847     0.836     0.841       292
             Soybean-mintill      0.885     0.946     0.914       737
               Soybean-clean      0.910     0.685     0.782       178
                       Wheat      0.935     0.951     0.943        61
                       Woods      0.969     0.976     0.972       380
Buildings-Grass-Trees-Drives      0.874     0.836     0.855       116
          Stone-Steel-Towers      0.903     1.000     0.949        28

                    accuracy                          0.904      3075
                   macro avg      0.860     0.873     0.863      3075
                weighted avg      0.906     0.904     0.904      3075
```