dataset

May 16, 2018

In [1]:
```
import numpy as np
from sklearn.decomposition import PCA
import scipy.io as sio
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import os
import random
from random import shuffle
from skimage.transform import rotate
import scipy.ndimage
from spectral import *
```

In [2]:
```
def loadIndianPinesData():
    data_path = os.path.join(os.getcwd(), 'data')
    data = sio.loadmat(os.path.join(data_path, 'Indian_pines.mat'))['indian_pines']
    labels = sio.loadmat(os.path.join(data_path, 'Indian_pines_gt.mat'))['indian_pines_

    return data, labels
```

In [3]:
```
def loadHSIData():
    data_path = os.path.join(os.getcwd(), 'HSI_data')
    data = open_image(os.path.join(data_path, '92AV3C.lan')).load()
    data = np.array(data).astype(np.int32)
    labels = open_image(os.path.join(data_path, '92AV3GT.GIS')).load()
    labels = np.array(labels).astype(np.uint8)
    labels.shape = (145, 145)
    return data, labels
```

In [4]:
```
def splitTrainTestSet(X, y, testRatio=0.10):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=testRatio, rand
                                                        stratify=y)
    return X_train, X_test, y_train, y_test
```

In [5]:
```
def oversampleWeakClasses(X, y):
    uniqueLabels, labelCounts = np.unique(y, return_counts=True)
    maxCount = np.max(labelCounts)
    labelInverseRatios = maxCount / labelCounts
    # repeat for every label and concat
```

```
            newX = X[y == uniqueLabels[0], :, :, :].repeat(round(labelInverseRatios[0]), axis=0
            newY = y[y == uniqueLabels[0]].repeat(round(labelInverseRatios[0]), axis=0)
            for label, labelInverseRatio in zip(uniqueLabels[1:], labelInverseRatios[1:]):
                cX = X[y== label,:,:,:].repeat(round(labelInverseRatio), axis=0)
                cY = y[y == label].repeat(round(labelInverseRatio), axis=0)
                newX = np.concatenate((newX, cX))
                newY = np.concatenate((newY, cY))
            np.random.seed(seed=42)
            rand_perm = np.random.permutation(newY.shape[0])
            newX = newX[rand_perm, :, :, :]
            newY = newY[rand_perm]
            return newX, newY
```

```
In [6]: def standartizeData(X):
            newX = np.reshape(X, (-1, X.shape[2]))
            scaler = preprocessing.StandardScaler().fit(newX)
            newX = scaler.transform(newX)
            newX = np.reshape(newX, (X.shape[0],X.shape[1],X.shape[2]))
            return newX, scaler
```

```
In [7]: def applyPCA(X, numComponents=75):
            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, pca
```

```
In [8]: 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
```

```
In [9]: def createPatches(X, y, windowSize=5, removeZeroLabels = True):
            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
```

In [10]:
```
def AugmentData(X_train):
    for i in range(int(X_train.shape[0]/2)):
        patch = X_train[i,:,:,:]
        num = random.randint(0,2)
        if (num == 0):

            flipped_patch = np.flipud(patch)
        if (num == 1):

            flipped_patch = np.fliplr(patch)
        if (num == 2):

            no = random.randrange(-180,180,30)
            flipped_patch = scipy.ndimage.interpolation.rotate(patch, no,axes=(1, 0),
                                                               reshape=False, output=


    patch2 = flipped_patch
    X_train[i,:,:,:] = patch2

    return X_train
```

In [11]:
```
def savePreprocessedData(path, X_trainPatches, X_testPatches, y_trainPatches, y_testPa
    data_path = os.path.join(os.getcwd(), path)

    if wasPCAapplied:
        with open(os.path.join(data_path, "XtrainWindowSize") + str(windowSize) + "PC
            np.save(outfile, X_trainPatches)
        with open(os.path.join(data_path, "XtestWindowSize") + str(windowSize) + "PCA
            np.save(outfile, X_testPatches)
        with open(os.path.join(data_path, "ytrainWindowSize") + str(windowSize) + "PC
            np.save(outfile, y_trainPatches)
        with open(os.path.join(data_path, "ytestWindowSize") + str(windowSize) + "PCA
            np.save(outfile, y_testPatches)
    else:
        with open(os.path.join(data_path, "preXtrainWindowSize") + str(windowSize) +
            np.save(outfile, X_trainPatches)
        with open(os.path.join(data_path, "preXtestWindowSize") + str(windowSize) + "
            np.save(outfile, X_testPatches)
        with open(os.path.join(data_path, "preytrainWindowSize") + str(windowSize) +
            np.save(outfile, y_trainPatches)
        with open(os.path.join(data_path, "preytestWindowSize") + str(windowSize) + "
            np.save(outfile, y_testPatches)
```

In [12]:
```
# Global Variables
```

```
        numComponents = 30
        windowSize = 5
        testRatio = 0.25
```

In [13]:
```
# X, y = loadIndianPinesData()
X, y = loadHSIData()
```

```
/home/danquxunhuan/software/anaconda3/envs/tf/lib/python3.6/site-packages/ipykernel_launcher.py
  This is separate from the ipykernel package so we can avoid doing imports until
/home/danquxunhuan/software/anaconda3/envs/tf/lib/python3.6/site-packages/ipykernel_launcher.py
  """
```

In [14]:
```
X, pca = applyPCA(X, numComponents=numComponents)
```

In [15]:
```
XPatches, yPatches = createPatches(X, y, windowSize=windowSize)
```

In [16]:
```
X_train, X_test, y_train, y_test = splitTrainTestSet(XPatches, yPatches, testRatio)
```

In [17]:
```
X_train, y_train = oversampleWeakClasses(X_train, y_train)
```

In [18]:
```
X_train = AugmentData(X_train)
```

In [19]:
```
savePreprocessedData('predata', X_train, X_test, y_train, y_test, windowSize = windowS
                            wasPCAapplied=True, numPCAComponents = numComponents,testRatio =
```