In [2]:

```
import numpy as np
from matplotlib import pyplot as plt
import scipy.io
import pandas as pd
```

In [3]:

```
paviaU = scipy.io.loadmat('./PaviaU.mat')['paviaU']
paviaU_gt = scipy.io.loadmat('./PaviaU_gt.mat')['paviaU_gt']
```

In [4]:

```
paviaU.shape
```

Out[4]:

```
(610, 340, 103)
```

In [6]:

```
from sklearn.decomposition import PCA
```

In [7]:

```
paviaU.max()
```

Out[7]:

```
8000
```

In [113]:

```
Nmax = paviaU.max()
paviaU = paviaU/Nmax
```

In [114]:

```
paviaU = paviaU.reshape(610*340, 103)
```

In [115]:

```
pca = PCA(3)
paviaU_PCA = pca.fit_transform(paviaU)
```

In [116]:

```
paviaU_PCA = paviaU_PCA.reshape((610,340,3))
```

In [117]:

```
paviaU.shape
```

Out[117]:

```
(207400, 103)
```

In [118]:

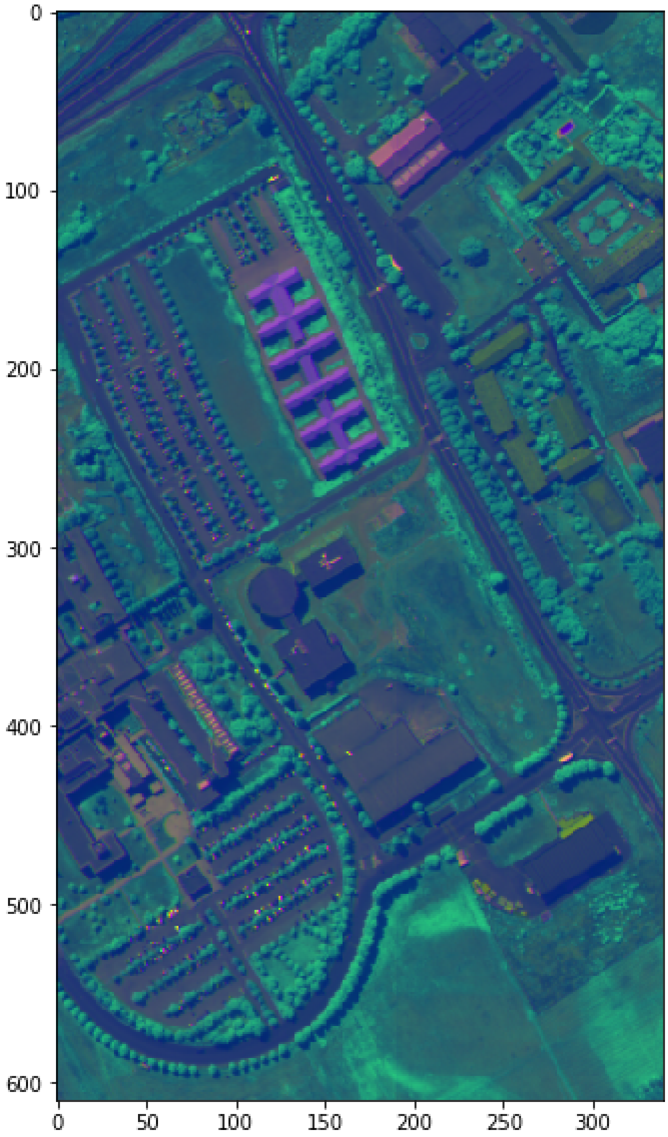
```
for i in range(3):
    paviaU_PCA[:,:,i] = (paviaU_PCA[:,:,i]-paviaU_PCA[:,:,i].min())/(paviaU_PCA[:,:,i].max()-paviaU_PCA[:,:,i
].min())
    paviaU_PCA[:,:,i] = paviaU_PCA[:,:,i]*255
paviaU_PCA = paviaU_PCA.astype(np.uint8)
```

In [121]:

```
plt.figure(figsize=(20,10))
plt.imshow(paviaU_PCA)
```

Out[121]:

```
<matplotlib.image.AxesImage at 0x2343571e548>
```



In [123]:

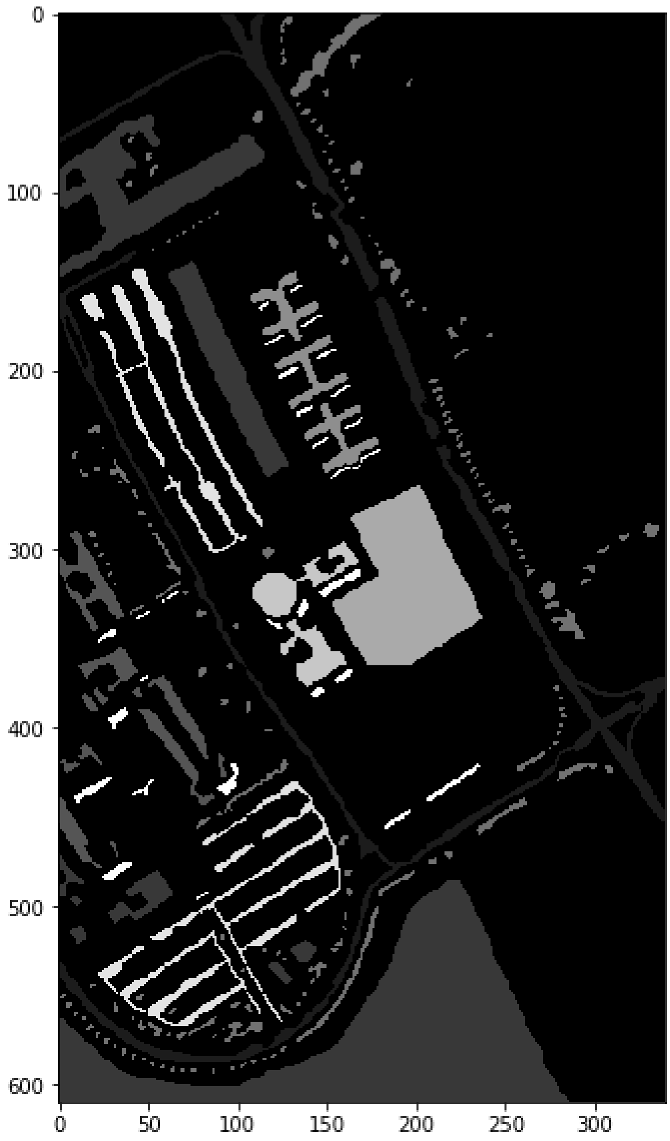
```
from PIL import Image
im = Image.fromarray(paviaU_PCA.astype(np.uint8))
im.save('./final5-3bandMin.png')
```

In [98]:

```
plt.figure(figsize=(20,10))
plt.imshow(paviaU_gt)
```

Out[98]:

```
<matplotlib.image.AxesImage at 0x23432a4cbc8>
```



In [15]:

```
n_class,num_pre_class = np.unique(paviaU_gt.reshape((610*340)), return_counts=True)
```

In [16]:

```
pca2 = PCA(50)
paviaU_PCA2 = pca2.fit_transform(paviaU)
```

In [17]:

```
df = pd.DataFrame(paviaU_PCA2,paviaU_gt.reshape(610*340))
df = df.sample(frac=1)
df = df.sort_index()
df
```

Out[17]:

```
            0          1          2          3          4          5          6          7
0   -0.419919   0.883765   0.162589   0.033725  -0.016419  -0.073402  -0.031792   0.009819  -0.
0    0.757739   0.766179  -0.473082   0.058154   0.015834  -0.028421   0.024878  -0.010561  -0.
0   -0.151969  -0.517426   0.042015   0.011967   0.006387   0.005717   0.030648   0.000831   0.
0   -1.179693  -0.964610   0.039623  -0.041884   0.005324  -0.014784   0.011408   0.004623  -0.
0   -0.787296  -0.850028   0.090382  -0.023381   0.005680  -0.018487   0.000813   0.034183   0.
...       ...        ...        ...        ...        ...        ...        ...        ...
9   -1.265273  -1.056977   0.075191   0.012749  -0.046316  -0.003674   0.009551  -0.035741  -0.
9   -1.244744  -1.052016   0.057734   0.003594  -0.027191  -0.001870  -0.013330   0.000384  -0.
9   -1.175072  -0.924598   0.111936  -0.010237  -0.017361  -0.004807   0.008021  -0.030619  -0.
9   -1.179533  -1.007106   0.085686   0.020339  -0.046996   0.000844  -0.031397  -0.006171  -0.
9   -1.270090  -0.968028   0.055700  -0.030246  -0.015856   0.002175   0.021756  -0.019950  -0.

207400 rows × 50 columns
```

In [18]:

```
n_class,num_pre_class
```

Out[18]:

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], dtype=uint8),
 array([164624,   6631,  18649,   2099,   3064,   1345,   5029,   1330,
          3682,    947], dtype=int64))
```

In [23]:

```
ttc = np.floor(num_pre_class/10).astype(int)
```

In [24]:

```
X_train = np.concatenate((df.iloc[0:ttc[0]],
                          df.iloc[num_pre_class[:1].sum():num_pre_class[:1].sum()+ttc[1]],
                          df.iloc[num_pre_class[:2].sum():num_pre_class[:2].sum()+ttc[2]],
                         df.iloc[num_pre_class[:3].sum():num_pre_class[:3].sum()+ttc[3]],
                         df.iloc[num_pre_class[:4].sum():num_pre_class[:4].sum()+ttc[4]],
                         df.iloc[num_pre_class[:5].sum():num_pre_class[:5].sum()+ttc[5]],
                         df.iloc[num_pre_class[:6].sum():num_pre_class[:6].sum()+ttc[6]],
                         df.iloc[num_pre_class[:7].sum():num_pre_class[:7].sum()+ttc[7]],
                         df.iloc[num_pre_class[:8].sum():num_pre_class[:8].sum()+ttc[8]],
                         df.iloc[num_pre_class[:9].sum():num_pre_class[:9].sum()+ttc[9]]))

y_train = np.concatenate(( np.zeros(ttc[0]) ,
                        np.ones(ttc[1]),
                        np.ones(ttc[2])*2,
                        np.ones(ttc[3])*3,
                        np.ones(ttc[4])*4,
                        np.ones(ttc[5])*5,
                        np.ones(ttc[6])*6,
                        np.ones(ttc[7])*7,
                        np.ones(ttc[8])*8,
                        np.ones(ttc[9])*9,))

X_test = np.concatenate((df.iloc[ttc[0]:num_pre_class[:1].sum()],
                        df.iloc[num_pre_class[:1].sum()+ttc[1]:num_pre_class[:2].sum()],
                        df.iloc[num_pre_class[:2].sum()+ttc[2]:num_pre_class[:3].sum()],
                        df.iloc[num_pre_class[:3].sum()+ttc[3]:num_pre_class[:4].sum()],
                        df.iloc[num_pre_class[:4].sum()+ttc[4]:num_pre_class[:5].sum()],
                        df.iloc[num_pre_class[:5].sum()+ttc[5]:num_pre_class[:6].sum()],
                        df.iloc[num_pre_class[:6].sum()+ttc[6]:num_pre_class[:7].sum()],
                        df.iloc[num_pre_class[:7].sum()+ttc[7]:num_pre_class[:8].sum()],
                        df.iloc[num_pre_class[:8].sum()+ttc[8]:num_pre_class[:9].sum()],
                        df.iloc[num_pre_class[:9].sum()+ttc[9]:]))
y_test = np.concatenate(( np.zeros(num_pre_class[0]-ttc[0]) ,
                        np.ones(num_pre_class[1]-ttc[1]),
                        np.ones(num_pre_class[2]-ttc[2])*2,
                        np.ones(num_pre_class[3]-ttc[3])*3,
                        np.ones(num_pre_class[4]-ttc[4])*4,
                        np.ones(num_pre_class[5]-ttc[5])*5,
                        np.ones(num_pre_class[6]-ttc[6])*6,
                        np.ones(num_pre_class[7]-ttc[7])*7,
                        np.ones(num_pre_class[8]-ttc[8])*8,
                        np.ones(num_pre_class[9]-ttc[9])*9,))
```

In [25]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(20735, 50)
(20735,)
(186665, 50)
(186665,)
```

In [26]:

```
idx = np.random.permutation(len(X_train))
X_train = X_train[idx]
y_train = y_train[idx]
idx = np.random.permutation(len(X_test))
X_test = X_test[idx]
y_test = y_test[idx]
```

In [27]:

```
from tensorflow import keras
```

```
C:\Users\Lay\Anaconda3\envs\ai\lib\site-packages\tensorflow\python\framework\dtypes.py:
526: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a fu
ture version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
C:\Users\Lay\Anaconda3\envs\ai\lib\site-packages\tensorflow\python\framework\dtypes.py:
527: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a fu
ture version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
C:\Users\Lay\Anaconda3\envs\ai\lib\site-packages\tensorflow\python\framework\dtypes.py:
528: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a fu
ture version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
C:\Users\Lay\Anaconda3\envs\ai\lib\site-packages\tensorflow\python\framework\dtypes.py:
529: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a fu
ture version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
C:\Users\Lay\Anaconda3\envs\ai\lib\site-packages\tensorflow\python\framework\dtypes.py:
530: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a fu
ture version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
C:\Users\Lay\Anaconda3\envs\ai\lib\site-packages\tensorflow\python\framework\dtypes.py:
535: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a fu
ture version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

In [28]:

```
y_train_encode = keras.utils.to_categorical(y_train)
y_test_encode = keras.utils.to_categorical(y_test)
```

In [55]:

```
def nnmodel(input_shape):
    X_input = keras.layers.Input((input_shape))
    #X = keras.layers.Dense(1024,activation='relu')(X_input)
    X = keras.layers.Dense(2048)(X_input)
    X = keras.layers.LeakyReLU(alpha=0.3)(X)
    X = keras.layers.Dense(512)(X)
    X = keras.layers.LeakyReLU(alpha=0.3)(X)
    X = keras.layers.Dense(256)(X)
    X = keras.layers.LeakyReLU(alpha=0.3)(X)
    X = keras.layers.Dense(10,activation='softmax')(X)
    model = keras.models.Model(inputs=X_input, outputs=X, name='model')
    return model
```

In [56]:

```
mymodel = nnmodel(X_train[0].shape)
mymodel.compile(optimizer="adam", loss="categorical_crossentropy", metrics=['accuracy'])
mymodel.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_5 (InputLayer)         (None, 50)                0         
_________________________________________________________________
dense_16 (Dense)             (None, 2048)              104448    
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 2048)              0         
_________________________________________________________________
dense_17 (Dense)             (None, 512)               1049088   
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 512)               0         
_________________________________________________________________
dense_18 (Dense)             (None, 256)               131328    
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 256)               0         
_________________________________________________________________
dense_19 (Dense)             (None, 10)                2570      
=================================================================
Total params: 1,287,434
Trainable params: 1,287,434
Non-trainable params: 0
_________________________________________________________________
```

In [57]:

```
mymodel.fit(X_train,y_train_encode,batch_size=32,epochs=20,validation_data=(X_test[:200],y_test_encode[:200]))
```

```
Train on 20735 samples, validate on 200 samples
Epoch 1/20
20735/20735 [==============================] - 4s 195us/sample - loss:
0.6195 - acc: 0.7925 - val_loss: 0.5955 - val_acc: 0.7650
Epoch 2/20
20735/20735 [==============================] - 4s 182us/sample - loss:
0.4877 - acc: 0.8084 - val_loss: 0.4805 - val_acc: 0.8350
Epoch 3/20
20735/20735 [==============================] - 4s 183us/sample - loss:
0.4420 - acc: 0.8204 - val_loss: 0.5074 - val_acc: 0.7950
Epoch 4/20
20735/20735 [==============================] - 4s 186us/sample - loss:
0.4213 - acc: 0.8258 - val_loss: 0.5085 - val_acc: 0.7950
Epoch 5/20
20735/20735 [==============================] - 4s 186us/sample - loss:
0.3984 - acc: 0.8312 - val_loss: 0.4464 - val_acc: 0.8300
Epoch 6/20
20735/20735 [==============================] - 4s 187us/sample - loss:
0.3856 - acc: 0.8367 - val_loss: 0.4616 - val_acc: 0.8200
Epoch 7/20
20735/20735 [==============================] - 4s 188us/sample - loss:
0.3729 - acc: 0.8405 - val_loss: 0.4463 - val_acc: 0.8050
Epoch 8/20
20735/20735 [==============================] - 4s 189us/sample - loss:
0.3650 - acc: 0.8447 - val_loss: 0.4407 - val_acc: 0.8150
Epoch 9/20
20735/20735 [==============================] - 4s 184us/sample - loss:
0.3603 - acc: 0.8469 - val_loss: 0.4570 - val_acc: 0.8250
Epoch 10/20
20735/20735 [==============================] - 4s 187us/sample - loss:
0.3562 - acc: 0.8497 - val_loss: 0.4687 - val_acc: 0.8300
Epoch 11/20
20735/20735 [==============================] - 4s 192us/sample - loss:
0.3519 - acc: 0.8492 - val_loss: 0.4341 - val_acc: 0.8250
Epoch 12/20
20735/20735 [==============================] - 4s 187us/sample - loss:
0.3412 - acc: 0.8545 - val_loss: 0.4049 - val_acc: 0.8200
Epoch 13/20
20735/20735 [==============================] - 4s 187us/sample - loss:
0.3350 - acc: 0.8552 - val_loss: 0.4104 - val_acc: 0.8350
Epoch 14/20
20735/20735 [==============================] - 4s 190us/sample - loss:
0.3329 - acc: 0.8594 - val_loss: 0.4492 - val_acc: 0.8150
Epoch 15/20
20735/20735 [==============================] - 4s 185us/sample - loss:
0.3290 - acc: 0.8613 - val_loss: 0.4507 - val_acc: 0.8000
Epoch 16/20
20735/20735 [==============================] - 4s 185us/sample - loss:
0.3254 - acc: 0.8597 - val_loss: 0.4170 - val_acc: 0.8100
Epoch 17/20
20735/20735 [==============================] - 4s 181us/sample - loss:
0.3224 - acc: 0.8629 - val_loss: 0.4346 - val_acc: 0.8300
Epoch 18/20
20735/20735 [==============================] - 4s 182us/sample - loss:
0.3191 - acc: 0.8629 - val_loss: 0.5035 - val_acc: 0.8100
Epoch 19/20
20735/20735 [==============================] - 4s 182us/sample - loss:
0.3173 - acc: 0.8628 - val_loss: 0.5096 - val_acc: 0.8000
Epoch 20/20
20735/20735 [==============================] - 4s 182us/sample - loss:
0.3099 - acc: 0.8680 - val_loss: 0.4606 - val_acc: 0.8250
```

Out[57]:

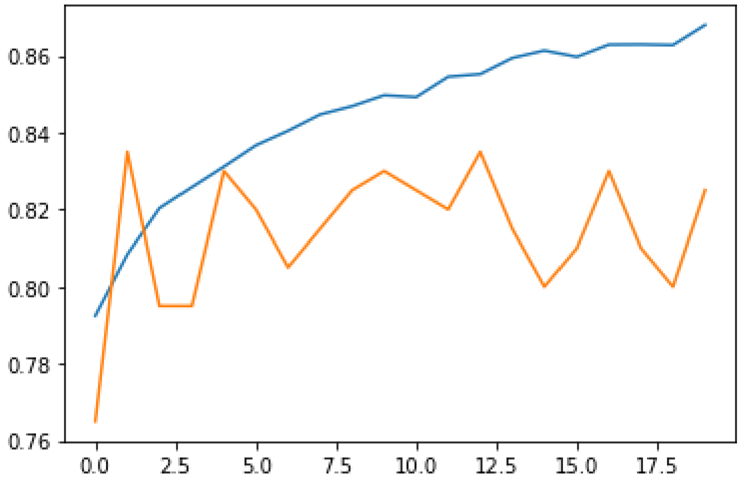
```
<tensorflow.python.keras.callbacks.History at 0x2342dd47ac8>
```

In [58]:

```
his = mymodel.history.history
plt.plot(his['acc'])
plt.plot(his['val_acc'])
```

Out[58]:

```
[<matplotlib.lines.Line2D at 0x2342e350f08>]
```



In [59]:

```
from sklearn.metrics import confusion_matrix,cohen_kappa_score,classification_report
```

In [60]:

```
print(mymodel.evaluate(X_test,y_test_encode))
```

```
186665/186665 [==============================] - 9s 47us/sample - loss:
0.3708 - acc: 0.8472
[0.37077086235417545, 0.84718615]
```

In [61]:

```
pred = mymodel.predict(X_test)
```

In [62]:

```
print(classification_report(y_test,np.argmax(pred,axis=1)))
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.94      0.91    148162
         1.0       0.51      0.48      0.49      5968
         2.0       0.75      0.62      0.68     16785
         3.0       0.69      0.09      0.15      1890
         4.0       0.61      0.25      0.35      2758
         5.0       0.76      0.97      0.85      1211
         6.0       0.78      0.22      0.34      4527
         7.0       0.66      0.78      0.71      1197
         8.0       0.59      0.42      0.49      3314
         9.0       0.00      0.00      0.00       853

    accuracy                           0.85    186665
   macro avg       0.62      0.48      0.50    186665
weighted avg       0.83      0.85      0.83    186665
```

In [126]:

```
conMat = confusion_matrix(y_test,np.argmax(pred,axis=1))
print(conMat)
```

```
[[139560   2712   3395     65    436    372    271    451    900      0]
 [  3073   2860      1      0      0      0      0     28      6      0]
 [  6372      0  10405      0      3      0      5      0      0      0]
 [  1662      1      0    164      0      0      0      0     63      0]
 [  2066      0      5      0    687      0      0      0      0      0]
 [    39      0      0      0      0   1172      0      0      0      0]
 [  3447      0     95      0      0      0    985      0      0      0]
 [   193     74      0      0      0      0      0    930      0      0]
 [  1925      1      1      8      0      0      0      2   1377      0]
 [   853      0      0      0      0      0      0      0      0      0]]
```

In [133]:

```
overall_acc = (conMat*np.eye(len(conMat))).sum()/conMat.sum()
overall_acc
```

Out[133]:

```
0.8471861355904964
```

In [63]:

```
cohen_kappa_score(y_test,np.argmax(pred,axis=1))
```

Out[63]:

```
0.5146317094258179
```