```
! pip install earthpy gdal


%cd /content/drive/My Drive/Satellite_data

S_sentinel_bands = glob("/content/drive/MyDrive/Satellite_data/*B?*.tiff")
S_sentinel_bands.sort()
S_sentinel_bands


l = []
for i in S_sentinel_bands:
  with rio.open(i, 'r') as f:
    l.append(f.read(1))
    arr_st = np.stack(l)
    arr_st.shape

from glob import glob
import numpy as np
from scipy.io import loadmat
import rasterio as rio

S_sentinel_bands = glob("/content/drive/MyDrive/Satellite_data/*B?*.tiff")
S_sentinel_bands.sort()

l = []
for i in S_sentinel_bands:
  with rio.open(i, 'r') as f:
    l.append(f.read(1))
```

```
# Data
arr_st = np.stack(l)

# Ground Truth
y_data = loadmat('Sundarbands_gt.mat')['gt']



y_data.sum()

l = []
for i in S_sentinel_bands:
  with rio.open(i, 'r') as f:
    l.append(f.read(1))
    arr_st = np.stack(l)
    arr_st.shape
    from glob import glob

import earthpy as et
import earthpy.spatial as es
import earthpy.plot as ep



import rasterio as rio
from rasterio.plot import plotting_extent
from rasterio.plot import show
from rasterio.plot import reshape_as_raster, reshape_as_image

import matplotlib.pyplot as plt
```

```
import numpy as np
from matplotlib.colors import ListedColormap

import plotly.graph_objects as go

np.seterr(divide='ignore', invalid='ignore')



ep.plot_rgb(
    arr_st,
    rgb=(3, 2, 1),
    stretch=True,
    str_clip=0.02,
    figsize=(12, 16),
    # title="RGB Composite Image with Stretch Applied",
)

plt.show()



colors = ['tomato', 'navy', 'MediumSpringGreen', 'lightblue', 'orange', 'blue',
          'maroon', 'purple', 'yellow', 'olive', 'brown', 'cyan']

ep.hist(arr_st,
         colors = colors,
        title=[f'Band-{i}' for i in range(1, 13)],
        cols=3,
        alpha=0.5,
        figsize = (12, 10)
```

```
        )

plt.show()

# Visualize Groundtruth

ep.plot_bands(y_data,
            cmap=ListedColormap([‘darkgreen’, ‘green’, ‘black’,
                                 ‘#CA6F1E’, ‘navy’, ‘forestgreen’]))
plt.show()

ep.plot_bands(arr_st,
            cmap = ‘gist_earth’,
            figsize = (20, 12),
            cols = 6,
            cbar = False)
plt.show()



from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

x = np.moveaxis(arr_st, 0, -1)
X_data = x.reshape(-1, 12)

scaler = StandardScaler().fit(X_data)
X_scaled = scaler.transform(X_data)

pca = PCA(n_components = 4)
```

```
pca.fit(X_scaled)

data = pca.transform(X_scaled)
from sklearn.preprocessing import minmax_scale
from sklearn.model_selection import train_test_split
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D,Dropout,Conv3D
from tensorflow.keras.layers import Dense, Activation, Flatten,Reshape
from tensorflow.keras.utils import to_categorical
from tensorflow.keras import backend as K
from sklearn.model_selection import train_test_split
from tensorflow.keras import callbacks
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from datetime import datetime
from keras.callbacks import TensorBoard
from tensorflow.keras.models import Model
import tensorflow as tf
tf.keras.callbacks.TensorBoard
from sklearn.metrics import confusion_matrix
import seaborn as sns
from sklearn.metrics import classification_report

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

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 = False):
    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

def splitTrainTestSet(X, y, testRatio, randomState=42):
    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

## GLOBAL VARIABLES
dataset = 'SB'
test_size = 0.30
windowSize = 15
MODEL_NAME = 'Sundarbans'
path = '/content/drive/My Drive/Satellite_data'

X_data = np.moveaxis(arr_st, 0, -1)
y_data = loadmat('Sundarbands_gt.mat')['gt']

# Apply PCA
K = 5
X,pca = applyPCA(X_data,numComponents=K)

print(f'Data After PCA: {X.shape}')

# Create 3D Patches
X, y = createImageCubes(X, y_data, windowSize=windowSize)
print(f'Patch size: {X.shape}')

# Split train and test
```

```
X_train, X_test, y_train, y_test = splitTrainTestSet(X, y, testRatio = test_size)

X_train = X_train.reshape(-1, windowSize, windowSize, K, 1)
X_test = X_test.reshape(-1, windowSize, windowSize, K, 1)

# One Hot Encoding
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

print(f'Train: {X_train.shape}\nTest: {X_test.shape}\nTrain Labels:
{y_train.shape}\nTest Labels: {y_test.shape}')

S = windowSize
L = K
output_units = y_train.shape[1]

## input layer
input_layer = Input((S, S, L, 1))

## convolutional layers
conv_layer1 = Conv3D(filters=16, kernel_size=(2, 2, 3),
activation='relu')(input_layer)
conv_layer2 = Conv3D(filters=32, kernel_size=(2, 2, 3),
activation='relu')(conv_layer1)
conv2d_shape = conv_layer2.shape
conv_layer3 = Reshape((conv2d_shape[1], conv2d_shape[2],
conv2d_shape[3]*conv2d_shape[4]))(conv_layer2)
conv_layer4 = Conv2D(filters=64, kernel_size=(2,2),
activation='relu')(conv_layer3)
```

```
flatten_layer = Flatten()(conv_layer4)

## fully connected layers
dense_layer1 = Dense(128, activation='relu')(flatten_layer)
dense_layer1 = Dropout(0.4)(dense_layer1)
dense_layer2 = Dense(64, activation='relu')(dense_layer1)
dense_layer2 = Dropout(0.4)(dense_layer2)
dense_layer3 = Dense(20, activation='relu')(dense_layer2)
dense_layer3 = Dropout(0.4)(dense_layer3)
output_layer = Dense(units=output_units, activation='softmax')(dense_layer3)
# define the model with input layer and output layer
model = Model(name = dataset+'_Model' , inputs=input_layer,
outputs=output_layer)

model.summary()



# Compile
model.compile(optimizer = 'adam', loss = 'categorical_crossentropy',
            metrics = ['accuracy'])

# Callbacks
logdir = path+"logs/" +model.name+'_'+datetime.now().strftime("%d:%m:%Y-
%H:%M:%S")

tensorboard_callback = TensorBoard(log_dir=logdir)

es = EarlyStopping(monitor = 'val_loss',
```

```
                min_delta = 0,
                patience = 1,
                verbose = 1,
                restore_best_weights = True)

checkpoint = ModelCheckpoint(filepath = ‘Pavia_University_Model.h5’,
                        monitor = ‘val_loss’,
                        mode =’min’,
                        save_best_only = True,
                        verbose = 1)
# Fit
history = model.fit(x=X_train, y=y_train,
                batch_size=1024*6, epochs=6,
                validation_data=(X_test, y_test), callbacks =
[tensorboard_callback, es, checkpoint])

import pandas as pd
history = pd.DataFrame(history.history)

plt.figure(figsize = (12, 6))
plt.plot(range(len(history[‘accuracy’].values.tolist())),
history[‘accuracy’].values.tolist(), label = ‘Train_Accuracy’)
plt.plot(range(len(history[‘loss’].values.tolist())), history[‘loss’].values.tolist(),
label = ‘Train_Loss’)
plt.plot(range(len(history[‘val_accuracy’].values.tolist())),
history[‘val_accuracy’].values.tolist(), label = ‘Test_Accuracy’)
plt.plot(range(len(history[‘val_loss’].values.tolist())),
history[‘val_loss’].values.tolist(), label = ‘Test_Loss’)
plt.xlabel(‘Epochs’)
```

```
plt.ylabel('Value')
plt.legend()
plt.show()


pred = model.predict(X_test, batch_size=1204*6, verbose=1)

plt.figure(figsize = (10,7))

classes = [f'Class-{i}' for i in range(1, 7)]

mat = confusion_matrix(np.argmax(y_test, 1),
                       np.argmax(pred, 1))

df_cm = pd.DataFrame(mat, index = classes, columns = classes)

sns.heatmap(df_cm, annot=True, fmt='d')

plt.show()

# Classification Report
print(classification_report(np.argmax(y_test, 1),
                       np.argmax(pred, 1),
     target_names = [f'Class-{i}' for i in range(1, 7)]))

pred_t = model.predict(X.reshape(-1, windowSize, windowSize, K, 1),
                    batch_size=1204*6, verbose=1)
# Visualize Groundtruth
```

```
ep.plot_bands(np.argmax(pred_t, axis=1).reshape(954, 298),
        cmap=ListedColormap([‘darkgreen’, ‘green’, ‘black’,
                ‘#CA6F1E’, ‘navy’, ‘forestgreen’]))
plt.show()


import earthpy.spatial as es

ndvi = es.normalized_diff(arr_st[7], arr_st[3])

ep.plot_bands(ndvi, cmap=”RdYlGn”, cols=1, vmin=-1, vmax=1, figsize=(10,
14))
ndvi.mean()
plt.show()


L = 0.5
savi = ((arr_st[7] – arr_st[3]) / (arr_st[7] + arr_st[3] + L)) * (1 + L)
ep.plot_bands(savi, cmap=”RdYlGn”, cols=1, vmin=-1, vmax=1, figsize=(10,
14))
plt.show()

vari = (arr_st[2] – arr_st[3])/ (arr_st[2] + arr_st[3] – arr_st[1])
ep.plot_bands(vari, cmap=”RdYlGn”, cols=1, vmin=-1, vmax=1, figsize=(10,
14))
plt.show()

ep.hist(np.stack([ndvi, savi, vari]),
    alpha=0.5,
```

```
        cols=3,
        figsize=(20, 5),
        title = ['NDVI', 'SAVI', 'VARI'],
        colors = ['mediumspringgreen', 'tomato', 'navy'])
plt.show()

mndwi = es.normalized_diff(arr_st[2], arr_st[10])
ep.plot_bands(mndwi, cmap="RdYlGn", cols=1, vmin=-1, vmax=1,
figsize=(10, 14))
plt.show()


ndmi = es.normalized_diff(arr_st[7], arr_st[10])
ep.plot_bands(ndmi, cmap="RdYlGn", cols=1, vmin=-1, vmax=1, figsize=(10,
14))
plt.show()

cmr = np.divide(arr_st[10], arr_st[11])
ep.plot_bands(cmr, cmap="RdYlGn", cols=1, vmin=-1, vmax=1, figsize=(10,
14))
plt.show()

fmr = np.divide(arr_st[10], arr_st[7])
ep.plot_bands(fmr, cmap="RdYlGn", cols=1, vmin=-1, vmax=1, figsize=(10,
14))
plt.show()
```