```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
pip install spectral
```

```
Requirement already satisfied: spectral in /usr/local/lib/python3.10/dist-packages (0.23.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from spectral) (1.23.5)
```

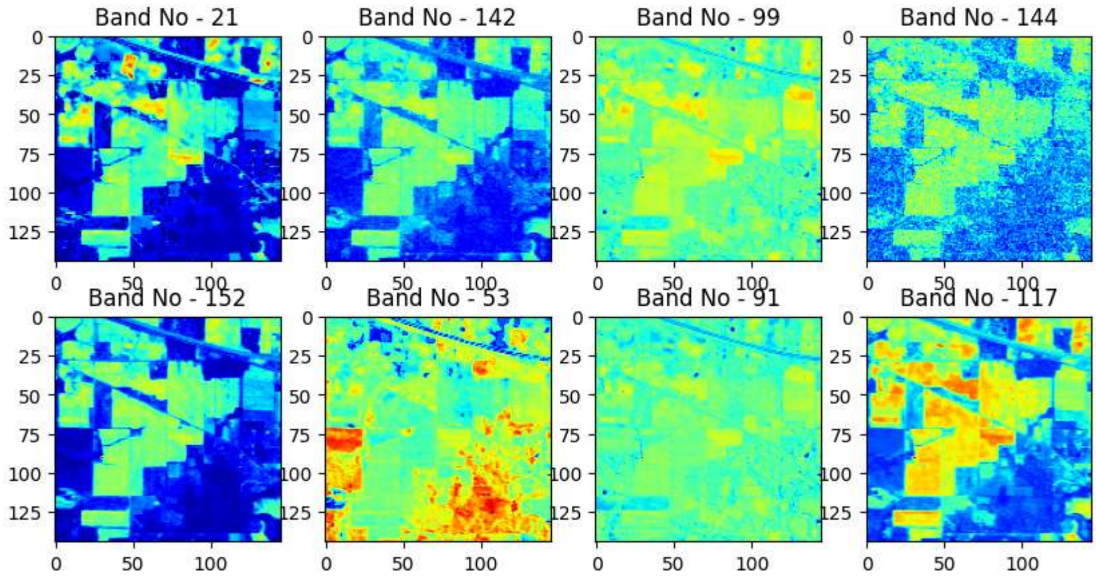
```
#importing libraries

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
import seaborn as sns
from sklearn.metrics import mean_squared_error, r2_score, classification_report, confusion_matrix, mean_absolute_error
from sklearn.decomposition import PCA
from scipy.io import loadmat
from PIL import Image
import spectral
from spectral import principal_components,open_image
from matplotlib.pyplot import imshow
from spectral import create_training_classes,GaussianClassifier

import math

import tensorflow as tf
from tensorflow.keras.optimizers import Adam
from tensorflow import keras
#from keras.utils import np_utils
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv3D, MaxPooling3D, Flatten, Dense
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, TensorBoard

from tqdm import tqdm
from numpy.random import seed
from time import time
```

```
#loading datasets
#/Indian_pines_corrected.mat
#/Indian_pines_gt.mat
X = loadmat("/Indian_pines_corrected.mat")["indian_pines_corrected"]
y = loadmat("/Indian_pines_gt.mat")["indian_pines_gt"]
```

```
#shape of the dataset
print("pinesdata_shape", X.shape)
print("groundtruth_shape", y.shape)
```

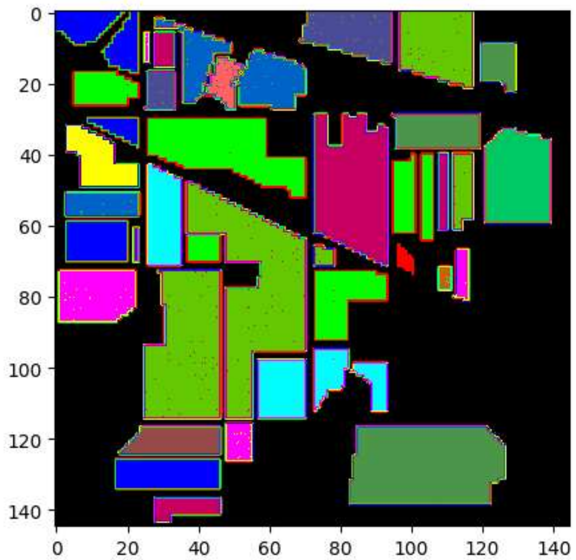
```
pinesdata_shape (145, 145, 200)
groundtruth_shape (145, 145)
```

```
#visualizing the bands
fig = plt.figure(figsize = (10, 5))

for i in range(1, 9):
    fig.add_subplot(2,4, i)
    band = np.random.randint(X.shape[2])
    plt.imshow(X[:,:,band], cmap = "jet")
    plt.title(f"Band No - {band}")
```



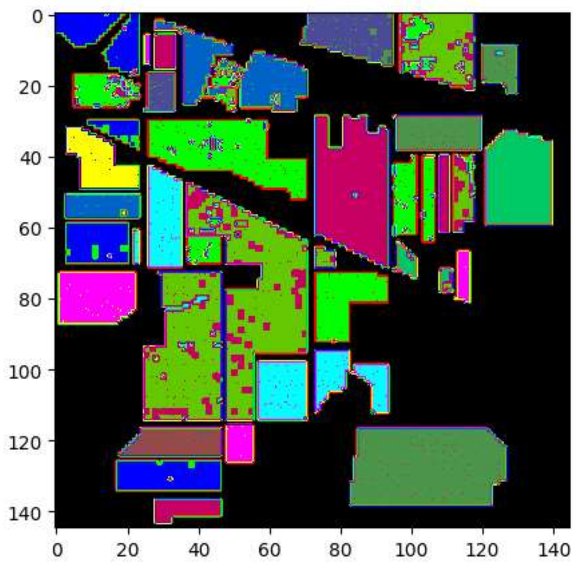

```
#visualizing the ground truth image
spectral.imshow(classes = y.astype(int), figsize = (5,5))
```

```
/usr/local/lib/python3.10/dist-packages/spectral/graphics/spypylab.py:796: UserWarnin
  warnings.warn(msg)
ImageView object:
  Interpolation       :  <default>
```



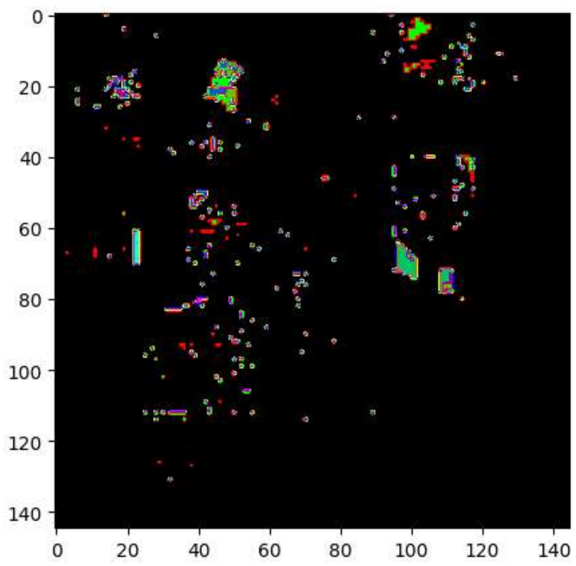
```
#visualization of classification map of classes after applying a Gaussian classifier
classes = create_training_classes(X,y)
guc = GaussianClassifier(classes)
output = guc.classify_image(X)
results = output*(y!= 0)
spectral.imshow(classes = results.astype(int), figsize = (10,5))
```

```
spectral:INFO: Setting min samples to 200
INFO:spectral:Setting min samples to 200
spectral:WARNING: Omitting class   1 : only 46 samples present
WARNING:spectral:Omitting class   1 : only 46 samples present
spectral:WARNING: Omitting class   7 : only 28 samples present
WARNING:spectral:Omitting class   7 : only 28 samples present
spectral:WARNING: Omitting class   9 : only 20 samples present
WARNING:spectral:Omitting class   9 : only 20 samples present
spectral:WARNING: Omitting class  16 : only 93 samples present
WARNING:spectral:Omitting class  16 : only 93 samples present
Processing...done
ImageView object:
  Interpolation       :  <default>
```



```
#visualizing the error between the classification map and the ground truth image
error = results*(results!= y)
spectral.imshow(classes = error.astype(int), figsize = (10,5))
```

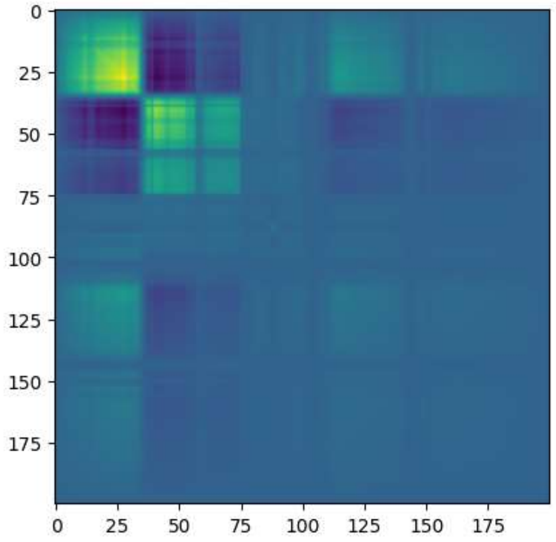
```
ImageView object:
  Interpolation       :  <default>
```



```
#shapes of the classification map and the output of the Gaussian classifier
print(results.shape,output.shape)
```

```
(145, 145) (145, 145)
```

```
#applying principal component analysis to the dataset and visualizing the covariance matrix of the principal components
pca = principal_components(X)

v = imshow(pca.cov)
```

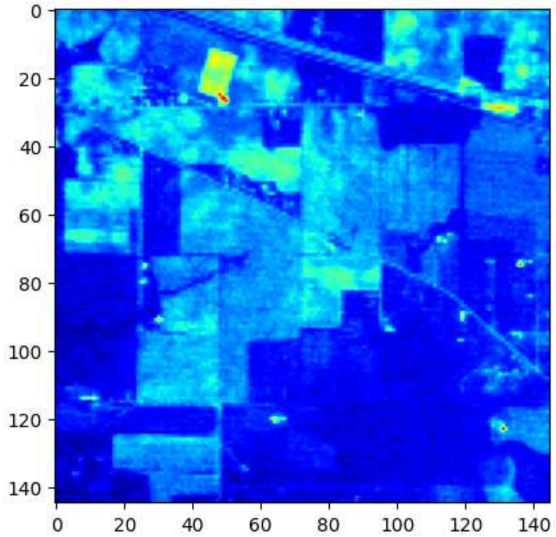


```
#reducing the dimensionality of the dataset by keeping 98.9% of the variance using PCA
pca_0989 = pca.reduce(fraction = 0.989)
len(pca_0989.eigenvalues)
```

```
24
```

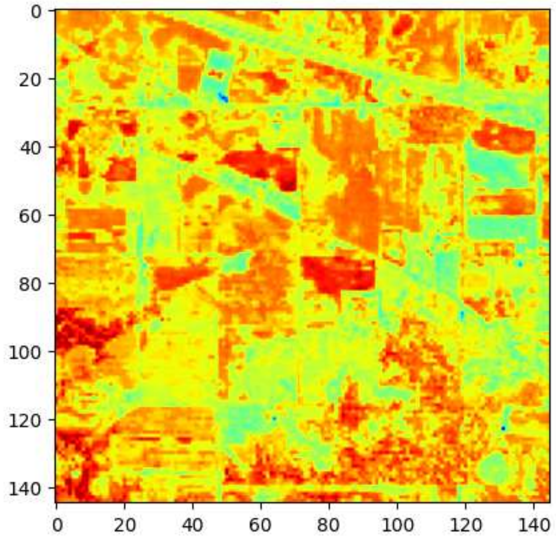
```
#visualizing a band of the original dataset
plt.imshow(X[:,:,4], cmap = "jet")
```

```
<matplotlib.image.AxesImage at 0x7bd48d841db0>
```



```
#transforming the original dataset using the reduced number of principal components and visualizing one of the bands of the transformed
img_pc = pca_0989.transform(X)
plt.imshow(img_pc[:,:,4], cmap = "jet")
```

```
<matplotlib.image.AxesImage at 0x7bd48d6cf0a0>
```

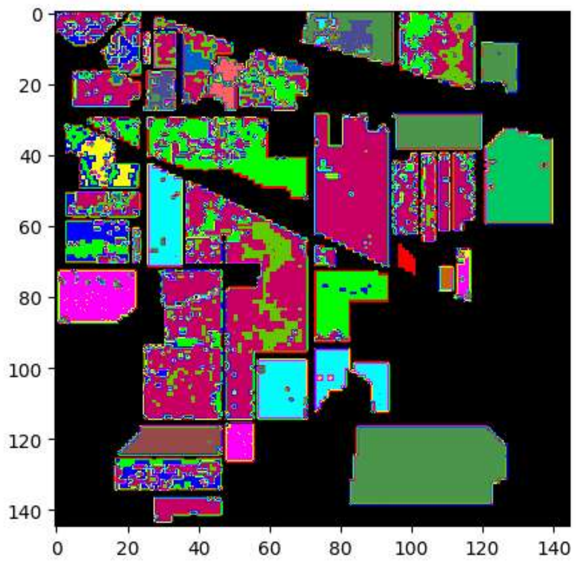



```
img_pc.shape
```

```
(145, 145, 24)
```

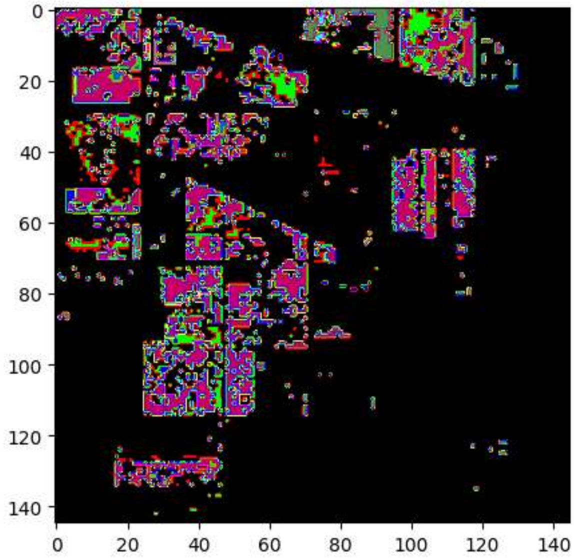
```
#creating a training dataset and a Gaussian classifier using the transformed dataset and visualizing the classification map of classes a
classes = create_training_classes(img_pc,y)
guc = GaussianClassifier(classes)
output = guc.classify_image(img_pc)
results = output*(y!= 0)
spectral.imshow(classes = results.astype(int), figsize = (10,5))
```

```
spectral:INFO: Setting min samples to 24
INFO:spectral:Setting min samples to 24
spectral:WARNING: Omitting class   9 : only 20 samples present
WARNING:spectral:Omitting class   9 : only 20 samples present
Processing...done
ImageView object:
  Interpolation       :  <default>
```




```
#visualizing the error between the classification map and the ground truth image
error = results*(results!= y)
spectral.imshow(classes = error.astype(int), figsize = (10,5))
```

```
ImageView object:
  Interpolation       :  <default>
```




```
#two utility functions are defined for creating windows of a specified size from the dataset and padding the dataset with zeros.

def padding_with_zeros(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


#defining a function imagewindows() to split the image into patches of a specified size and create corresponding labels for each patch.

def imagewindows(X, y, windowSize = 5, removeZeroLabels = True):

    margin = int((windowSize - 1) / 2)
    zeroPaddedX = padding_with_zeros(X, margin=margin)
    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
```

```
X_win, y = imagewindows(img_pc, y, windowSize = 25)
X_win.shape, y.shape
```

```
((10249, 25, 25, 24), (10249,))
```

```
X_train,X_test,y_train,y_test = train_test_split(X_win, y, test_size = 0.50, random_state = 42)
```

```
X_train.shape, y_train.shape
```

```
((5124, 25, 25, 24), (5124,))
```

```
#reshaping the training data to have a 5D shape
X_train = X_train.reshape(-1, 25, 25, 24, 1)
X_train.shape
```

```
(5124, 25, 25, 24, 1)
```

```
y_train = tf.keras.utils.to_categorical(y_train)
y_train.shape

#from tensorflow.keras.utils import to_categorical
```

```
(5124, 16)
```

```
model = tf.keras.models.Sequential([
        keras.layers.Conv3D(16, (3, 3, 3), input_shape = (25, 25, 24, 1)),
        keras.layers.Conv3D(32, (3, 3, 5)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(256, activation = "relu"),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(512, activation = "relu"),
        tf.keras.layers.Dropout(0.5),
        tf.keras.layers.Dense(16, activation = "softmax")
])
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv3d (Conv3D)              (None, 23, 23, 22, 16)    448

 conv3d_1 (Conv3D)            (None, 21, 21, 18, 32)    23072

 flatten (Flatten)            (None, 254016)            0

 dense (Dense)                (None, 256)               65028352

 dropout (Dropout)            (None, 256)               0

 dense_1 (Dense)              (None, 512)               131584

 dropout_1 (Dropout)          (None, 512)               0

 dense_2 (Dense)              (None, 16)                8208

=================================================================
Total params: 65191664 (248.69 MB)
Trainable params: 65191664 (248.69 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(loss = "categorical_crossentropy", optimizer = "adam", metrics = ["accuracy"])
```

```
early_stop = EarlyStopping(monitor = "val_loss",
                           min_delta = 0,
                           patience = 10,
                           mode = "min",
                           verbose = 1,
                           restore_best_weights = True)
```

```
checkpoint = ModelCheckpoint(filepath = "Indian_Pines_3DCNN.h5",
                             monitor = "val_loss",
                             mode = "min",
                             verbose = 1,
                             save_best_only = True)
```

```
tensorboard = TensorBoard(log_dir = "SA_logs/{}".format(time()))
```

```
history = model.fit(X_train, y_train,
                    epochs = 3,
                    batch_size = 256,
                    callbacks = [early_stop, checkpoint, tensorboard])
```

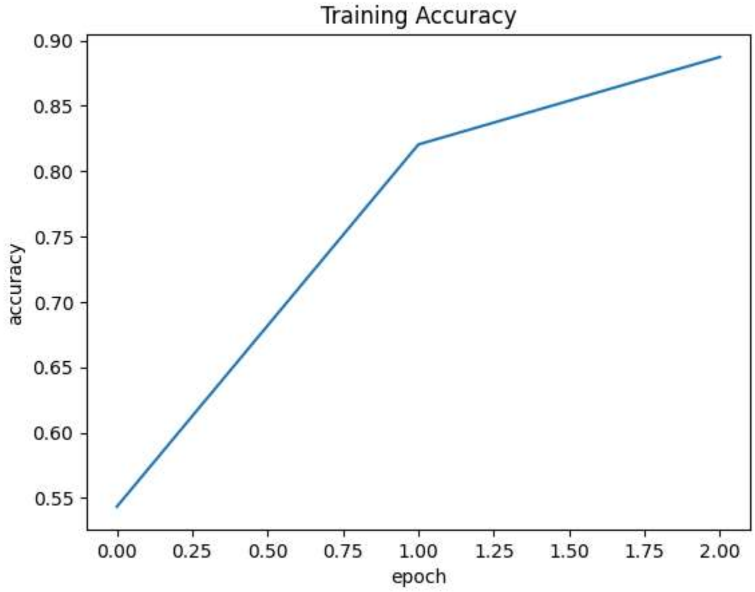
```
Epoch 1/3
21/21 [==============================] - ETA: 0s - loss: 552.0843 - accuracy: 0.5433 WARNING:tensorflow:Early stopping conditioned o
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
21/21 [==============================] - 426s 20s/step - loss: 552.0843 - accuracy: 0.5433
Epoch 2/3
21/21 [==============================] - ETA: 0s - loss: 259.4223 - accuracy: 0.8205 WARNING:tensorflow:Early stopping conditioned o
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
21/21 [==============================] - 379s 18s/step - loss: 259.4223 - accuracy: 0.8205
Epoch 3/3
21/21 [==============================] - ETA: 0s - loss: 151.7419 - accuracy: 0.8874 WARNING:tensorflow:Early stopping conditioned o
WARNING:tensorflow:Can save best model only with val_loss available, skipping.
21/21 [==============================] - 375s 18s/step - loss: 151.7419 - accuracy: 0.8874
```

```
y_pred = model.predict(X_test)
```

```
161/161 [==============================] - 117s 729ms/step
```

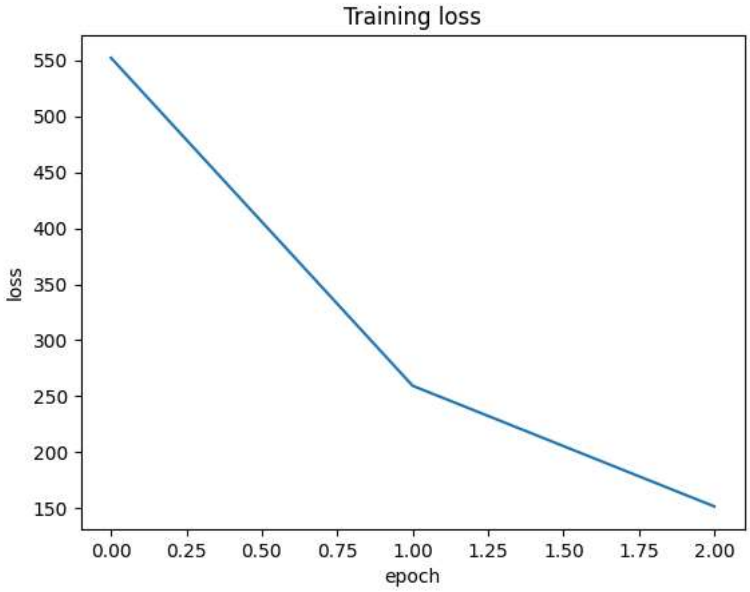
```
plt.plot(history.history["accuracy"])
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.title("Training Accuracy")
```

```
Text(0.5, 1.0, 'Training Accuracy')
```




```
plt.plot(history.history["loss"])
plt.xlabel("epoch")
plt.ylabel("loss")
plt.title("Training loss")
```

```
Text(0.5, 1.0, 'Training loss')
```




```
model.save("/content/indianapines_3DCNNmodel.h5")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file vi
  saving_api.save_model(
```

```
X_test_reshaped = X_test.reshape(-1, 25, 25, 24, 1)
X_test_reshaped.shape
```

```
(5125, 25, 25, 24, 1)
```

```
y_test_reshaped = tf.keras.utils.to_categorical(y_test )
y_test_reshaped.shape
```

```
(5125, 16)
```

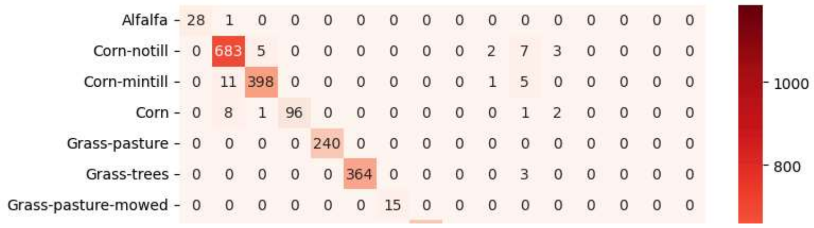
```
y_prediction_test = model.predict(X_test_reshaped)
y_pred_test = np.argmax(y_prediction_test, axis = 1)
y_test_uncategorical = np.argmax(y_test_reshaped, axis = 1)
```

```
161/161 [==============================] - 118s 736ms/step
```

```
# Classification Report
report = classification_report(y_test_uncategorical, y_pred_test, target_names=["Alfalfa", "Corn-notill", "Corn-mintill", "Corn", "Grass-

print(report)
```

```
                              precision    recall  f1-score   support

                     Alfalfa       1.00      0.97      0.98        29
                 Corn-notill       0.96      0.98      0.97       700
                Corn-mintill       0.99      0.96      0.97       415
                        Corn       1.00      0.89      0.94       108
               Grass-pasture       1.00      1.00      1.00       240
                 Grass-trees       0.99      0.99      0.99       367
         Grass-pasture-mowed       1.00      1.00      1.00        15
               Hay-windrowed       1.00      1.00      1.00       238
                        Oats       1.00      1.00      1.00         8
              Soybean-notill       0.99      0.99      0.99       494
             Soybean-mintill       0.98      0.99      0.99      1200
               Soybean-clean       0.97      1.00      0.98       308
                       Wheat       1.00      1.00      1.00        97
                       Woods       1.00      1.00      1.00       634
Buildings-Grass-Trees-Drives       0.97      0.99      0.98       225
          Stone-Steel-Towers       1.00      0.94      0.97        47

                    accuracy                           0.98      5125
                   macro avg       0.99      0.98      0.98      5125
                weighted avg       0.98      0.98      0.98      5125
```

```
plt.figure(figsize = (8,6))

#confusion matrix
c_matrix = confusion_matrix(y_test_uncategorical, y_pred_test)

#heatmap of the confusion matrix
sns.heatmap(c_matrix, annot = True, annot_kws = {"size": 10}, fmt= "d", cmap = "Reds",
            xticklabels=["Alfalfa", "Corn-notill", "Corn-mintill", "Corn", "Grass-pasture", "Grass-trees", "Grass-pasture-mowed", "Hay-w
            yticklabels=["Alfalfa", "Corn-notill", "Corn-mintill", "Corn", "Grass-pasture", "Grass-trees", "Grass-pasture-mowed", "Hay-w

plt.show()
```




```
MSE = np.square(np.subtract(y_test_uncategorical, y_pred_test)).mean()
RMSE = math.sqrt(MSE)
print("Root Mean Square Error:\n", RMSE)
```

```
Root Mean Square Error:
 0.7671295880606149
```

```
print("Mean Absolute Error (MAE)", mean_absolute_error(y_test_uncategorical, y_pred_test))
```

```
Mean Absolute Error (MAE) 0.0784390243902439
```