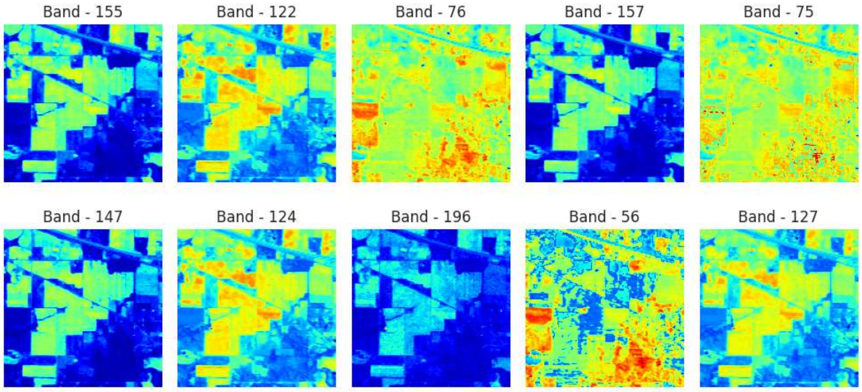
```
#pip install earthpy
```

```
#importing libraries

import numpy as np
import pandas as pd
import earthpy.plot as ep
import matplotlib.pyplot as plt
import seaborn as sns

from scipy.io import loadmat

from tqdm import tqdm

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import earthpy.spatial as es

import plotly.graph_objects as go
import plotly.express as px

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
#loading datasets
pines = loadmat("/content/Indian_pines_corrected.mat")["indian_pines_corrected"]
groundtruth = loadmat("/content/Indian_pines_gt.mat")["indian_pines_gt"]
```

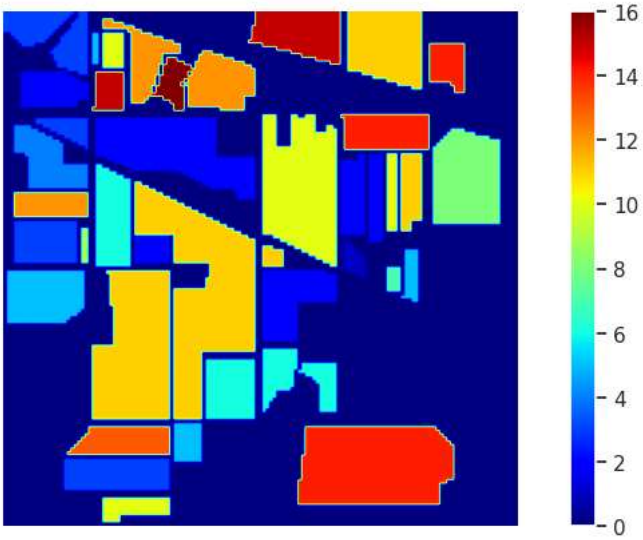
```
df = pd.DataFrame(pines.reshape(pines.shape[0]*pines.shape[1], -1))

df.columns = [f"band{i}" for i in range(1, df.shape[-1]+1)]

df["class"] = groundtruth.ravel()
```

```
#visualizing the bands

fig = plt.figure(figsize = (10, 5))

ax = fig.subplots(2, 5)

for i in range(2):
    for j in range(5):
        c = np.random.randint(200)
        ax[i][j].imshow(pines[:, :, c], cmap = "jet")
        ax[i][j].axis("off")
        ax[i][j].title.set_text(f"Band - {c}")
        c+=1

plt.tight_layout()

plt.show()
```




```
#visualizing rgb
mdata = np.moveaxis(pines, -1, 0)

ep.plot_rgb(mdata, (25, 10, 15), figsize =(10, 5))

plt.show()
```

```
#visualizing of groundtruth
def plot_data(pines):
    fig = plt.figure(figsize = (10, 5))
    plt.imshow(pines, cmap = "jet")
    plt.colorbar()
    plt.axis("off")
    plt.show()

plot_data(groundtruth)
```




```
X = df[df["class"]!=0].iloc[:, :-1].values

y = tf.keras.utils.to_categorical(df[df["class"]!=0].iloc[:, -1].values, num_classes = np.unique(groundtruth).shape[0], dtype = "float3
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.7, stratify = y)
print(f"Train Data: {X_train.shape}\nTest Data: {X_test.shape}")
```

```
Train Data: (7174, 200)
Test Data: (3075, 200)
```

```
model = Sequential(name = "Indian_Pines")

#Add input layer with the same shape as the training data
model.add(Input(shape = X_train[0].shape, name = "Input_Layer"))

#Add batch normalization layer to improve model stability and convergence
model.add(BatchNormalization(name = "BatchNormalization"))

#Add dense layers with 128 neurons and ReLU activation
model.add(Dense(units = 128, activation= "relu", name = "Layer1"))
model.add(Dense(units = 128, activation= "relu", name = "Layer2"))
model.add(Dense(units = 128, activation= "relu", name = "Layer3"))
model.add(Dense(units = 128, activation= "relu", name = "Layer4"))

#Add dropout layer to prevent overfitting
model.add(Dropout(rate = 0.2, name = "Dropout1",))

#Add dense layers with 64 neurons and ReLU activation
model.add(Dense(units = 64, activation= "relu", name = "Layer5"))
model.add(Dense(units = 64, activation= "relu", name = "Layer6"))
model.add(Dense(units = 64, activation= "relu", name = "Layer7"))
model.add(Dense(units = 64, activation= "relu", name = "Layer8"))

#Add another dropout layer
model.add(Dropout(rate = 0.2, name = "Dropout2"))

#Add dense layers with 32 neurons and ReLU activation
model.add(Dense(units = 32, activation= "relu", name = "Layer9"))
model.add(Dense(units = 32, activation= "relu", name = "Layer10"))
model.add(Dense(units = 32, activation= "relu", name = "Layer11"))
model.add(Dense(units = 32, activation= "relu", name = "Layer12"))

#Add output layer with softmax activation and the same number of units as the number of classes
model.add(Dense(units = y_train.shape[1], activation= "softmax", name = "Output_Layer"))
```

```
model.summary()
```

Model: "Indian_Pines"

Layer (type)	Output Shape	Param #
BatchNormalization (BatchN ormalization)	(None, 200)	800
Layer1 (Dense)	(None, 128)	25728
Layer2 (Dense)	(None, 128)	16512
Layer3 (Dense)	(None, 128)	16512

```
 Layer4 (Dense)              (None, 128)               16512

 Dropout1 (Dropout)          (None, 128)               0

 Layer5 (Dense)              (None, 64)                8256

 Layer6 (Dense)              (None, 64)                4160

 Layer7 (Dense)              (None, 64)                4160

 Layer8 (Dense)              (None, 64)                4160

 Dropout2 (Dropout)          (None, 64)                0

 Layer9 (Dense)              (None, 32)                2080

 Layer10 (Dense)             (None, 32)                1056

 Layer11 (Dense)             (None, 32)                1056

 Layer12 (Dense)             (None, 32)                1056

 Output_Layer (Dense)        (None, 17)                561

=================================================================
Total params: 102609 (400.82 KB)
Trainable params: 102209 (399.25 KB)
Non-trainable params: 400 (1.56 KB)
_________________________________________________________________
```

```
model.compile(optimizer = "adam", loss = "categorical_crossentropy", metrics = ["accuracy"])
es = EarlyStopping(monitor = "val_loss",
                   min_delta = 0,
                   patience = 15,
                   verbose = 1,
                   restore_best_weights = True)
checkpoint = ModelCheckpoint(filepath = "Indian_Pines_CNNModel.h5",
                             monitor = "val_loss",
                             mode = "min",
                             save_best_only = True,
                             verbose = 1)
history = model.fit(x = X_train, y = y_train,
          validation_data = (X_test, y_test),
          epochs = 5,
          callbacks = [es, checkpoint])
```

```
Epoch 1/5
220/225 [============================>.] - ETA: 0s - loss: 1.8240 - accuracy: 0.3449
Epoch 1: val_loss improved from inf to 1.84814, saving model to Indian_Pines_CNNModel.h5
225/225 [==============================] - 4s 6ms/step - loss: 1.8178 - accuracy: 0.3475 - val_loss: 1.8481 - val_accuracy: 0.3148
Epoch 2/5
 33/225 [===>..........................] - ETA: 0s - loss: 1.5659 - accuracy: 0.4119/usr/local/lib/python3.10/dist-packages/keras/s
  saving_api.save_model(
218/225 [============================>.] - ETA: 0s - loss: 1.4447 - accuracy: 0.4857
Epoch 2: val_loss improved from 1.84814 to 1.42251, saving model to Indian_Pines_CNNModel.h5
225/225 [==============================] - 1s 6ms/step - loss: 1.4457 - accuracy: 0.4851 - val_loss: 1.4225 - val_accuracy: 0.4823
Epoch 3/5
224/225 [============================>.] - ETA: 0s - loss: 1.3070 - accuracy: 0.5212
Epoch 3: val_loss improved from 1.42251 to 1.14581, saving model to Indian_Pines_CNNModel.h5
225/225 [==============================] - 2s 8ms/step - loss: 1.3068 - accuracy: 0.5212 - val_loss: 1.1458 - val_accuracy: 0.5620
Epoch 4/5
215/225 [===========================>..] - ETA: 0s - loss: 1.2261 - accuracy: 0.5472
Epoch 4: val_loss did not improve from 1.14581
225/225 [==============================] - 1s 4ms/step - loss: 1.2260 - accuracy: 0.5464 - val_loss: 1.1612 - val_accuracy: 0.5519
Epoch 5/5
213/225 [===========================>..] - ETA: 0s - loss: 1.1620 - accuracy: 0.5621
Epoch 5: val_loss improved from 1.14581 to 1.06058, saving model to Indian_Pines_CNNModel.h5
225/225 [==============================] - 1s 5ms/step - loss: 1.1557 - accuracy: 0.5637 - val_loss: 1.0606 - val_accuracy: 0.5782
```

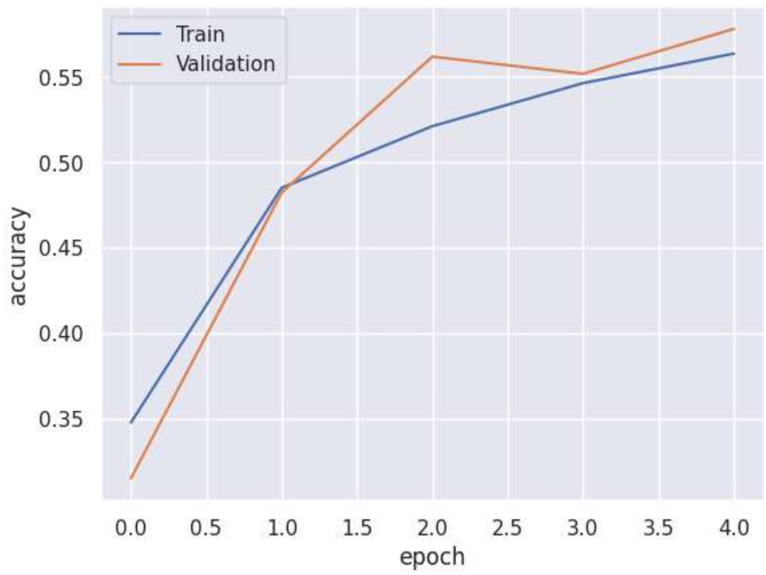
```
y_pred = model.predict(X_test)
```

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

```
score = model.evaluate(X_test, y_test, verbose = 0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

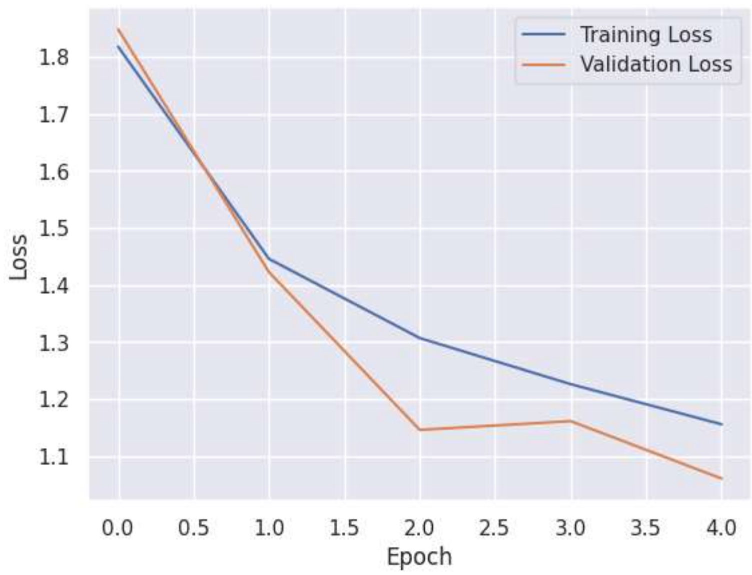
```
Test loss: 1.0605764389038086
Test accuracy: 0.5782113671302795
```

```
sns.set()
plt.plot(history.history["accuracy"])
plt.plot(history.history["val_accuracy"])
plt.xlabel("epoch")
plt.ylabel("accuracy")
plt.legend(["Train", "Validation"])
plt.show()
```




```
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7c97b24a6e30>
```

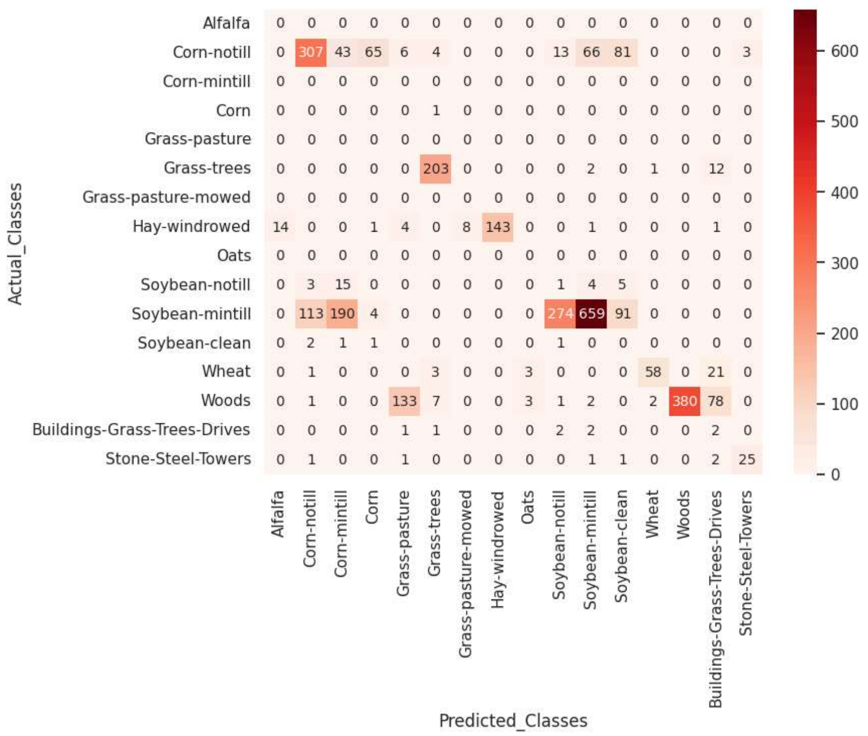



```
pred = np.argmax(model.predict(X_test), axis=1)

# Classification Report
print(classification_report(pred, np.argmax(y_test, 1),
      target_names = ["Alfalfa", "Corn-notill", "Corn-mintill", "Corn", "Grass-pasture", "Grass-trees", "Grass-pasture-mowed", "Hay-winc
```

```
97/97 [==============================] - 0s 2ms/step
                              precision    recall  f1-score   support

                     Alfalfa       0.00      0.00      0.00         0
                 Corn-notill       0.72      0.52      0.60       588
                Corn-mintill       0.00      0.00      0.00         0
                        Corn       0.00      0.00      0.00         1
               Grass-pasture       0.00      0.00      0.00         0
                 Grass-trees       0.93      0.93      0.93       218
         Grass-pasture-mowed       0.00      0.00      0.00         0
               Hay-windrowed       1.00      0.83      0.91       172
```

```
                         Oats       0.00      0.00      0.00         0
               Soybean-notill       0.00      0.04      0.01        28
              Soybean-mintill       0.89      0.50      0.64      1331
                Soybean-clean       0.00      0.00      0.00         5
                        Wheat       0.95      0.67      0.79        86
                        Woods       1.00      0.63      0.77       607
 Buildings-Grass-Trees-Drives       0.02      0.25      0.03         8
           Stone-Steel-Towers       0.89      0.81      0.85        31

                     accuracy                           0.58      3075
                    macro avg       0.40      0.32      0.35      3075
                 weighted avg       0.88      0.58      0.69      3075

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Recall and F-score are ill
  _warn_prf(average, modifier, msg_start, len(result))
```

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

classes = ["Alfalfa", "Corn-notill", "Corn-mintill", "Corn", "Grass-pasture", "Grass-trees", "Grass-pasture-mowed", "Hay-windrowed", "Oa

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

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

df_cm.index.name = "Actual_Classes"
df_cm.columns.name = "Predicted_Classes"

sns.heatmap(df_cm, annot = True, annot_kws = {"size": 10}, fmt= "d", cmap = "Reds")

plt.show()
```




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

```
Root Mean Square Error:
```

```
from sklearn.metrics import mean_absolute_error
print("Mean Absolute Error (MAE)", mean_absolute_error(y_test,y_pred))
```

```
Mean Absolute Error (MAE) 0.058708027
```

```
'''l = []

for i in tqdm(range(df.shape[0])):
    if df.iloc[i, -1] == 0:
        l.append(0)
    else:
        l.append(np.argmax(model.predict(df.iloc[i, :-1].values.reshape(-1, 200)), 1))

q = np.array(l).reshape(groundtruth.shape).astype('float')

plot_data(q)'''
```

```
'l = []\n\nfor i in tqdm(range(df.shape[0])):\n    if df.iloc[i, -1] == 0:\n        l.append(0)\n    else:\n        l.append(np.arg
max(model.predict(df.iloc[i, :-1].values.reshape(-1, 200)), 1))\n\nq = np.array(l).reshape(groundtruth.shape).astype('float')\n\npl
ot_data(q)'
```

+ Code + Text

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.