Project Name: Satellite Image Data Analysis Using Numpy

Objective

The objective of this project is to analyze a satellite image dataset from the WIFIRE Project using the numpy library. The primary goals are:

1. To understand and manipulate the image data represented in RGB format.
2. To visualize and modify different aspects of the image, such as altitude, aspect, and slope.
3. To apply image processing techniques like masking, filtering, and color manipulation to extract meaningful information.

Summary

The WIFIRE Project provides satellite imagery that captures various environmental factors. In this project, RGB color mapping is used to represent different attributes:

- **RED** pixels indicate altitude.
- **BLUE** pixels indicate aspect.
- **GREEN** pixels indicate slope.

Using numpy and other Python libraries, this project demonstrates how to:

- Load and display satellite images.
- Perform basic image manipulations and analyses.
- Apply masking techniques to highlight specific features in the images.

Results

1. **Image Loading and Display**:
    - The satellite image was successfully loaded and displayed using imageio and matplotlib.
    - The image data was represented as a three-dimensional numpy array with dimensions corresponding to the image's height, width, and color layers.
2. **Pixel Value Analysis**:
    - Pixel values were analyzed to understand the intensity of each color channel (Red, Green, Blue).
    - Examples of specific pixel manipulations were demonstrated, including setting pixel values to zero and adjusting color intensities.
3. **Color Range Manipulation**:
    - Modified specific color layers within defined ranges to observe changes in the image.
    - Demonstrated the impact of setting color intensities to maximum and minimum values.

4. **Filtering and Masking**:
    - Created and applied boolean masks to filter out low pixel values.
    - Implemented more complex patterns, such as diagonal lines and circular masks, to manipulate image sections.
    - Applied masks to highlight high-altitude areas by isolating high-intensity red pixels.
5. **Composite Masking**:
    - Developed a composite mask to filter pixels based on thresholds across all three color layers (Red, Green, Blue).
    - Highlighted specific features by combining multiple masks.

Conclusion

The analysis of satellite imagery using numpy provides valuable insights into image data manipulation and visualization. By applying various techniques such as pixel manipulation, color intensity adjustment, and masking, it is possible to extract meaningful information from satellite images. This project demonstrated the versatility of numpy in handling image data and illustrated practical methods to visualize and analyze environmental factors captured in satellite imagery.

Code:

```
'''
Data Source: Satellite Image from WIFIRE Project
WIFIRE is an integrated system for wildfire analysis, with specific regard to changing urban dynamics
and
climate. The system integrates networked observations such as heterogeneous satellite data and
real-time
remote sensor data, with computational techniques in signal processing, visualization, modeling, and
data
assimilation to provide a scalable method to monitor such phenomena as weather patterns that can
help
predict a wildfire's rate of spread.
RGB Color Mapping in the Photo:
 RED pixel indicates Altitude
 BLUE pixel indicates Aspect
 GREEN pixel indicates Slope
The higher values denote higher altitude, aspect and slope.
In this example, we will analyze a sample satellite image dataset from WIFIRE using the numpy
Library.
'''


#Loading the libraries we need: numpy, scipy, matplotlib
import numpy as np
import scipy
import imageio
import matplotlib.pyplot as plt
import skimage #install scikit-image
import warnings
warnings.filterwarnings("ignore")


#Creating a numpy array from an image file:
#Lets choose a WIFIRE satellite image file as an ndarray and display its type
from skimage import data
```

```
photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

type(photo_data)


#Let us see what is in this image.

plt.figure(figsize=(15,15))

plt.imshow(photo_data)

plt.show()

#Let us see how does the photo_data looks like:

print(photo_data)

#Let us print the shape of photo_data

print(photo_data.shape)

#The shape of the ndarray show that it is a three layered matrix.

# The first two numbers here are length and width, and

# the third number (i.e. 3) is for three layers: Red, Green and Blue.



#RGB Color Mapping in the Photo:

#RED pixel indicates Altitude

#BLUE pixel indicates Aspect

#GREEN pixel indicates Slope

#The higher values denote higher altitude, aspect and slope.

photo_data.size # Data Size

photo_data.min(), photo_data.max() # Maximum and Minimum of the pixel value

photo_data.mean() # Average Pixel Value


#Pixel on the 150th Row and 250th Column:

photo_data[150, 250] # It will print the Red, Green , Blue Value in order

photo_data[150, 250, 1] # It will print out the Green Value
```

```
#Pixel on the 2nd Row and 2nd Column:

photo_data[1, 1] # It will print the Red, Green, Blue Value in order

#Set a Pixel to All Zeros

#We can set all three layer in a pixel as once by assigning zero globally to that

# (row,column) pairing. However, setting one pixel to zero is not noticeable.

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

photo_data[150, 250] = 0 # We set all three layers of RGB of this

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

plt.imshow(photo_data)

plt.show()


#Changing colors in a Range

#We can also use a range to change the pixel values. As an example,

# let us set the green layer for rows 200 to 800 to full intensity.

#We will set the value of Green layer to full intensity for rows 200 (inclusive)

# to 800 (exclusive) for all the columns.

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

photo_data[200:800, : ,1] = 255

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

plt.imshow(photo_data)

plt.show()


#We will set the value of Red, Green and Blue layer to full intensity

# (we will get a white block) for rows 200 (inclusive) to 800 (exclusive) for all the columns.

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

photo_data[200:800, :] = 255

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

plt.imshow(photo_data)

plt.show()
```

```
#We will set the value of Red, Green and Blue layer to least intensity

# (we will get a black block) for rows 200 (inclusive) to 800 (exclusive) for all the columns.

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

photo_data[200:800, :] = 0

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

plt.imshow(photo_data)

plt.show()



#Pick all Pixels with Low Values

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

print("Shape of photo_data:", photo_data.shape)

low_value_filter = photo_data < 100 #Create a boolen array with the same shape as of phoyo_data

print("Shape of low_value_filter:", low_value_filter.shape)



#Filtering Out Low Values:

#Whenever the low_value_filter is True, set value to 0.

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

plt.imshow(photo_data)

photo_data[low_value_filter] = 0 #set low values to 0

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

plt.imshow(photo_data)

plt.show()


#More Row and Column Operations

#We can design complex patterns by making columns a function

# of rows or vice-versa. Here we try a linear relationship between rows and columns.
```

```
rows_range = np.arange(len(photo_data)) #Create a range array

print(rows_range)

cols_range = rows_range #Create a range array

print(cols_range)

print(type(rows_range))


#We are setting the selected rows and columns to the maximum value of 255

photo_data[rows_range, cols_range] = 255

print(photo_data)

plt.figure(figsize=(15,15))

plt.imshow(photo_data)

plt.show()

#We see a diagonal white line that is a result of our operation.



#Masking Images

#Now let us try to mask the image in shape of a circular disc.

total_rows, total_cols, total_layers = photo_data.shape

print("photo_data = ", photo_data.shape)

X, Y = np.ogrid[:total_rows, :total_cols]

print("X = ", X.shape, " and Y = ", Y.shape)

from IPython.display import Image

Image("Images/figure.png")

center_row, center_col = total_rows / 2, total_cols / 2

print("center_row = ", center_row, "AND center_col = ", center_col)

print(X - center_row)

print(Y - center_col)

dist_from_center = (X - center_row)**2 + (Y - center_col)**2

print(dist_from_center)

radius = (total_rows / 2)**2

print("Radius = ", radius)
```

```
circular_mask = (dist_from_center > radius)

print(circular_mask)

print(circular_mask[1500:1700,2000:2200])

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

photo_data[circular_mask] = 0

plt.figure(figsize=(15,15))

plt.imshow(photo_data)

#Further Masking

#We can further improve the mask, for example just get upper half disc.

X, Y = np.ogrid[:total_rows, :total_cols]

half_upper = X < center_row # this line generates a mask for all rows above the center

half_upper_mask = np.logical_and(half_upper, circular_mask)

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

photo_data[half_upper_mask] = 255

#photo_data[half_upper_mask] = random.randint(200,255)

plt.figure(figsize=(15,15))

plt.imshow(photo_data)

plt.show()


'''

Further Processing of our Satellite Imagery

* Processing of RED Pixels


* Remember that red pixels tell us about the height. Let us try to highlight all the high altitude areas.

We will do this by detecting high intensity RED Pixels and muting down other areas.
'''


photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

red_mask = photo_data[:, : ,0] < 150
```

```
photo_data[red_mask] = 0

plt.figure(figsize=(15,15))

plt.imshow(photo_data)

plt.show()

#Detecting Highl-GREEN Pixels

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

green_mask = photo_data[:, : ,1] < 150

photo_data[green_mask] = 0

plt.figure(figsize=(15,15))

plt.imshow(photo_data)

plt.show()

#Detecting Highly-BLUE Pixels

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

blue_mask = photo_data[:, : ,2] < 150

photo_data[blue_mask] = 0

plt.figure(figsize=(15,15))

plt.imshow(photo_data)

plt.show()

#Composite mask that takes thresholds on all three layers: RED, GREEN, BLUE

photo_data = imageio.imread("D:\\Aditya's Notes\\Aditya's Data Science Notes\\Projects and Other Datasets\\ML PROJECTS\\data\\sd-3layers.jpg")

red_mask = photo_data[:, : ,0] < 150

green_mask = photo_data[:, : ,1] > 100

blue_mask = photo_data[:, : ,2] < 100

final_mask = np.logical_and(red_mask, green_mask, blue_mask)

photo_data[final_mask] = 0

plt.figure(figsize=(15,15))

plt.imshow(photo_data)

plt.show()
```