Satellite Image Data Analysis using numpy

Data Source: Satellite Image from WIFIRE Project

Name: ***Md. Abrar Jahin***

Roll:1811035

Department: Industrial Engineering and Management

Batch: 2018

University: Khulna University of Engineering and Technology, Khulna

Submission Date: 01 May 2020

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. The details about WIFIRE are mentioned at: [https://wifire.ucsd.edu/ (https://wifire.ucsd.edu/)](https://wifire.ucsd.edu/)

In this project, I analyzed a sample satellite image dataset from WIFIRE using the numpy Library.

Loading the libraries we need: numpy, scipy, matplotlib

In [1]:
```
%matplotlib inline
import numpy as np
from scipy import misc
import matplotlib.pyplot as plt
import imageio
```

Creating a numpy array from an image file:

Let's choose a WIFIRE satellite image file as an ndarray and display its type.

In [2]:
```
from skimage import data

photo_data = imageio.imread('./wifire/sd-3layers.jpg')

type(photo_data)
```

Out[2]:
```
imageio.core.util.Array
```

Now let's see what is in this image.

In [3]:
```
plt.figure(figsize=(15,15))
plt.imshow (photo_data)
```

Out[3]: <matplotlib.image.AxesImage at 0x1cd56ee5f08>

In [4]:
```
photo_data.shape

#print(photo_data)
```

Out[4]: (3725, 4797, 3)

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

</ul>

The higher values denote higher altitude, aspect and slope.

In [5]:
```
photo_data.size
```

Out[5]: 53606475

In [6]:
```
photo_data.min(), photo_data.max()
```

Out[6]: (0, 255)

In [7]:
```
photo_data.mean()
```

Out[7]: 75.8299354508947

Pixel on the 150th Row and 250th Column

In [8]:
```
photo_data[150, 250]#150=r,250=col,17, 35, 255=rgb
```

Out[8]:
```
Array([ 17,  35, 255], dtype=uint8)
```

In [9]:
```
photo_data[150, 250, 1]#0,1,2 means r,g,b
```

Out[9]:
```
35
```

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.

In [10]:
```
#photo_data = misc.imread('./wifire/sd-3layers.jpg')
photo_data[150, 250] = 0
plt.figure(figsize=(10,10))
plt.imshow(photo_data)
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x1cd5a29ba88>
```

Changing colors in a Range

In [11]:
```
photo_data = imageio.imread('./wifire/sd-3layers.jpg')

photo_data[200:800, : ,1] = 255
plt.figure(figsize=(10,10))
plt.imshow(photo_data)
```

Out[11]: <matplotlib.image.AxesImage at 0x1cd5a2da648>

In [12]:
```
photo_data = imageio.imread('./wifire/sd-3layers.jpg')

photo_data[200:800, :] = 255
plt.figure(figsize=(10,10))
plt.imshow(photo_data)
```

Out[12]: <matplotlib.image.AxesImage at 0x1cd5a33fcc8>

In [13]:
```
photo_data = imageio.imread('./wifire/sd-3layers.jpg')

photo_data[200:800, :] = 0
plt.figure(figsize=(10,10))
plt.imshow(photo_data)
```

Out[13]: <matplotlib.image.AxesImage at 0x1cd5a659688>

Pick all Pixels with Low Values

In [14]:
```
photo_data = imageio.imread('./wifire/sd-3layers.jpg')
print("Shape of photo_data:", photo_data.shape)
low_value_filter = photo_data < 200
print("Shape of low_value_filter:", low_value_filter.shape)
```

```
Shape of photo_data: (3725, 4797, 3)
Shape of low_value_filter: (3725, 4797, 3)
```

Filtering Out Low Values

In [28]:
```
import random
plt.figure(figsize=(10,10))
plt.imshow(photo_data)
photo_data[low_value_filter] = 0
plt.figure(figsize=(10,10))
plt.imshow(photo_data)
```

Out[28]: <matplotlib.image.AxesImage at 0x1cd039bd9c8>

More Row and Column Operations

In [16]:
```
rows_range = np.arange(len(photo_data))
cols_range = rows_range
print(type(rows_range))
print(rows_range)
```

```
<class 'numpy.ndarray'>
[   0    1    2 ... 3722 3723 3724]
```

In [17]:
```
photo_data[rows_range, cols_range] = 255
```

In [18]:
```
plt.figure(figsize=(15,15))
plt.imshow(photo_data)
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x1cd5a6a7188>
```

Masking Images

In [19]:
```
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)
print(Y)
```

```
photo_data =  (3725, 4797, 3)
X =  (3725, 1)  and Y =  (1, 4797)
[[   0    1    2 ... 4794 4795 4796]]
```

In [20]:
```
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])
```

```
center_row =  1862.5 AND center_col =  2398.5
[[-1862.5]
 [-1861.5]
 [-1860.5]
 ...
 [ 1859.5]
 [ 1860.5]
 [ 1861.5]]
[[-2398.5 -2397.5 -2396.5 ...  2395.5  2396.5  2397.5]]
[[9221708.5 9216912.5 9212118.5 ... 9207326.5 9212118.5 9216912.5]
 [9217984.5 9213188.5 9208394.5 ... 9203602.5 9208394.5 9213188.5]
 [9214262.5 9209466.5 9204672.5 ... 9199880.5 9204672.5 9209466.5]
 ...
 [9210542.5 9205746.5 9200952.5 ... 9196160.5 9200952.5 9205746.5]
 [9214262.5 9209466.5 9204672.5 ... 9199880.5 9204672.5 9209466.5]
 [9217984.5 9213188.5 9208394.5 ... 9203602.5 9208394.5 9213188.5]]
Radius =  3468906.25
[[ True  True  True ...  True  True  True]
 [ True  True  True ...  True  True  True]
 [ True  True  True ...  True  True  True]
 ...
 [ True  True  True ...  True  True  True]
 [ True  True  True ...  True  True  True]
 [ True  True  True ...  True  True  True]]
[[False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False False]
 ...
 [False False False ... False False False]
 [False False False ... False False False]
 [False False False ... False False False]]
```

In [21]:
```
photo_data = imageio.imread('./wifire/sd-3layers.jpg')
photo_data[circular_mask] = 0
plt.figure(figsize=(15,15))
plt.imshow(photo_data)
```

Out[21]: <matplotlib.image.AxesImage at 0x1cd00148d48>

Further Masking

Half Upper Masking

In [22]:
```
import numpy as np
X, Y = np.ogrid[:total_rows, :total_cols]
half_upper = X < center_row # this line generates a mask for all rows above th
e center

half_upper_mask = np.logical_and(half_upper, circular_mask)
```

In [29]:
```
photo_data = imageio.imread('./wifire/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)
```

Out[29]: <matplotlib.image.AxesImage at 0x1cd0392b808>

Further Processing of our Satellite Imagery

Processing of RED Pixels

We know, red pixels tell us about the height. I highlighted all the high altitude areas. I did this by detecting high intensity RED Pixels and muting down other areas.

In [24]:
```
photo_data = imageio.imread('./wifire/sd-3layers.jpg')
red_mask   = photo_data[:, : ,0] < 150

photo_data[red_mask] = 0
plt.figure(figsize=(15,15))
plt.imshow(photo_data)
```

Out[24]: <matplotlib.image.AxesImage at 0x1cd03587d48>

Detecting High-GREEN Pixels

In [25]:
```
photo_data = imageio.imread('./wifire/sd-3layers.jpg')
green_mask = photo_data[:, : ,1] < 150

photo_data[green_mask] = 0
plt.figure(figsize=(15,15))
plt.imshow(photo_data)
```

Out[25]: <matplotlib.image.AxesImage at 0x1cd035ef748>

Detecting Highly-BLUE Pixels

In [26]:
```
photo_data = imageio.imread('./wifire/sd-3layers.jpg')
blue_mask  = photo_data[:, : ,2] < 150

photo_data[blue_mask] = 0
plt.figure(figsize=(15,15))
plt.imshow(photo_data)
```

Out[26]: <matplotlib.image.AxesImage at 0x1cd038b3fc8>

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

In [27]:
```
photo_data = imageio.imread('./wifire/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)
```

Out[27]: <matplotlib.image.AxesImage at 0x1cd03920548>

In [ ]: