# Satellite Image Data Analysis using numpy

## 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. You can read more about WIFIRE at: https://wifire.ucsd.edu/ (https://wifire.ucsd.edu/)

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

In [ ]:

```
%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:

Lets choose a WIFIRE satellite image file as an ndarray and display its type.

In [ ]:

```
from skimage import data

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

# Note: In the video, this is mentioned as an ndarray. Don't worry about the type f
# still work as usual.
type(photo_data)
```

Let's see what is in this image.

In [ ]:

```
plt.figure(figsize=(15,15))
plt.imshow(photo_data)
```

In [ ]:

```
photo_data.shape

#print(photo_data)
```

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.

In [ ]:

```
photo_data.size
```

In [ ]:

```
photo_data.min(), photo_data.max()
```

In [ ]:

```
photo_data.mean()
```

Pixel on the 150th Row and 250th Column

In [ ]:

```
photo_data[150, 250]
```

In [ ]:

```
photo_data[150, 250, 1]
```

## 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 [ ]:

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

## Changing Colors in a Range

We can also use a range to change the pixel values. As an example, let's set the green layer for rows 200 t0 800 to full intensity.

In [ ]:

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

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

In [ ]:

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

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

In [ ]:

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

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

## Pick all Pixels with Low Values

In [ ]:

```
photo_data = imageio.imread('./wifire/sd-3layers.jpg')
print("Shape of photo_data:", photo_data.shape)
low_value_filter = photo_data < 50
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.

In [ ]:

```
# 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)
```

## More Row and Column Operations

You can design complex patters by making cols a function of rows or vice-versa. Here we try a linear relationship between rows and columns.

In [ ]:

```
rows_range = np.arange(len(photo_data))
cols_range = rows_range
print(type(rows_range))
```

In [ ]:

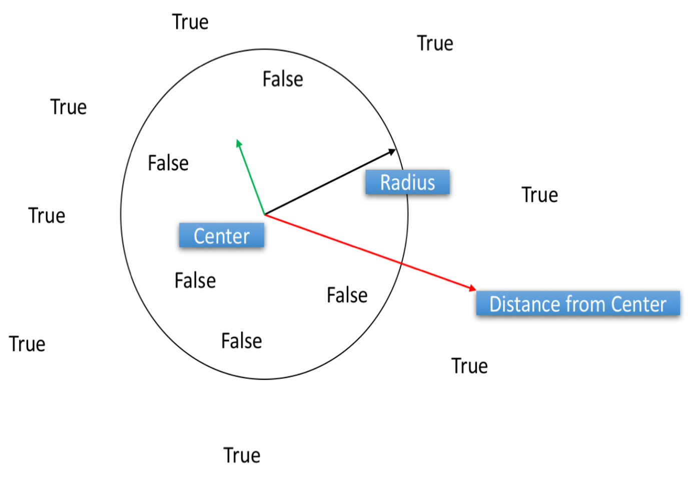
```
photo_data[rows_range, cols_range] = 255
```

In [ ]:

```
plt.figure(figsize=(15,15))
plt.imshow(photo_data)
```

## Masking Images

Now let us try something even cooler...a mask that is in shape of a circular disc.



In [ ]:

```
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)
```

In [ ]:

```
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])
```

In [ ]:

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

## Further Masking

You can further improve the mask. For example, just get the upper half of the disc.

In [ ]:

```
X, Y = np.ogrid[:total_rows, :total_cols]
half_upper = X < center_row # this line generates a mask for all rows above the cen

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

In [ ]:

```
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)
```

# 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.

In [ ]:

```
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)
```

## Detecting Highly-GREEN Pixels

In [ ]:

```
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)
```

## Detecting Highly-BLUE Pixels

In [ ]:

```
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)
```

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

In [ ]:

```
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)
```

In [ ]:

```

```