KHOA CÔNG NGHỆ THÔNG TIN

BỘ MÔN TIN HỌC ĐỊA CHẤT

========

BÁO CÁO SINH HOẠT HỌC THUẬT

PHÂN TÍCH ẢNH VỆ TINH VỚI PYTHON

Người thực hiện: Phạm An Cương

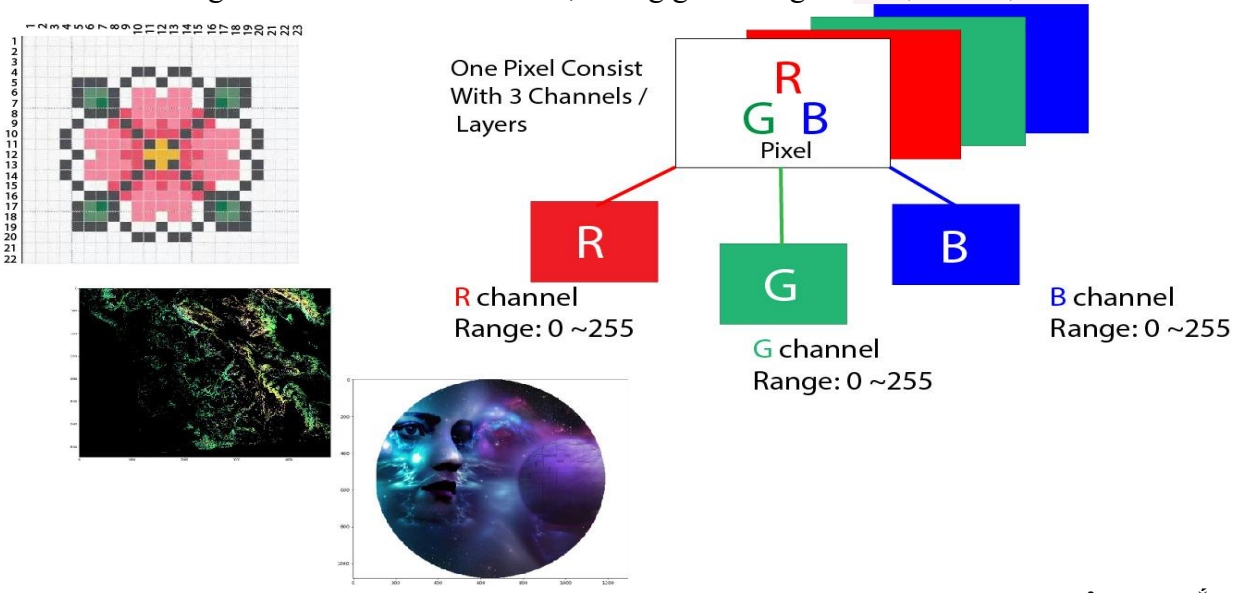
Hà Nội, tháng 12 năm 2021

MỤC LỤC

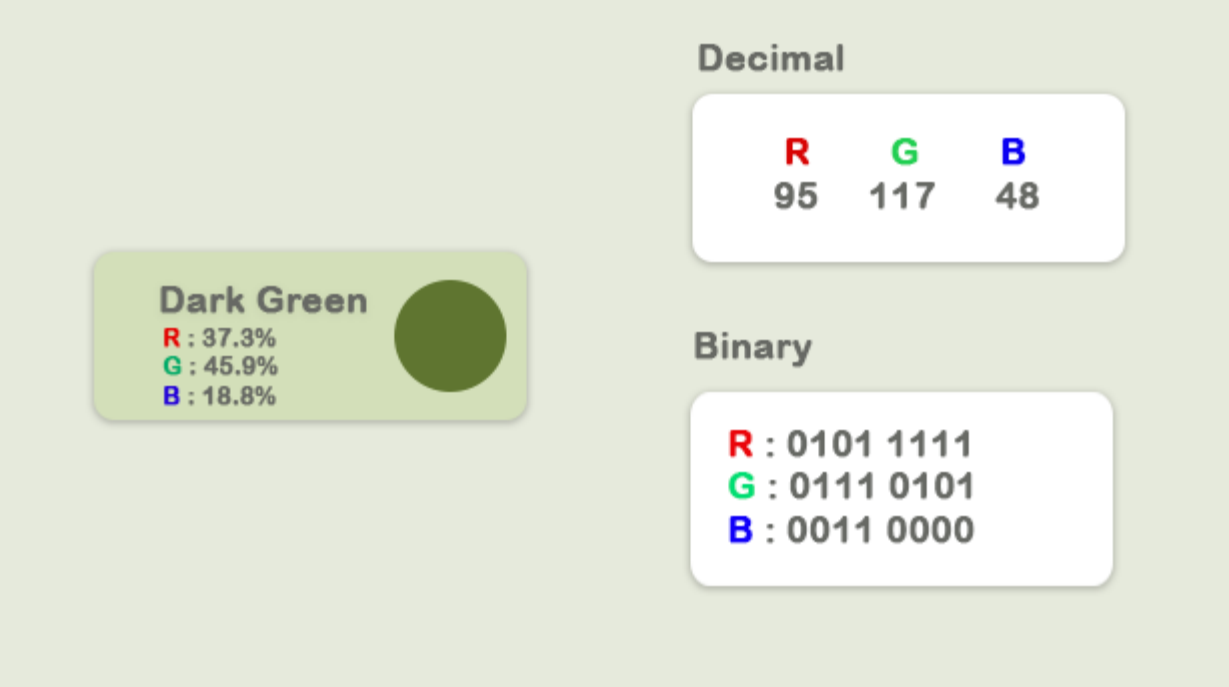
CHƯƠNG 1. PHÂN TÍCH DỮ LIỆU HÌNH ẢNH CƠ BẢN BẰNG PYTHON

1.1.Giới thiệu

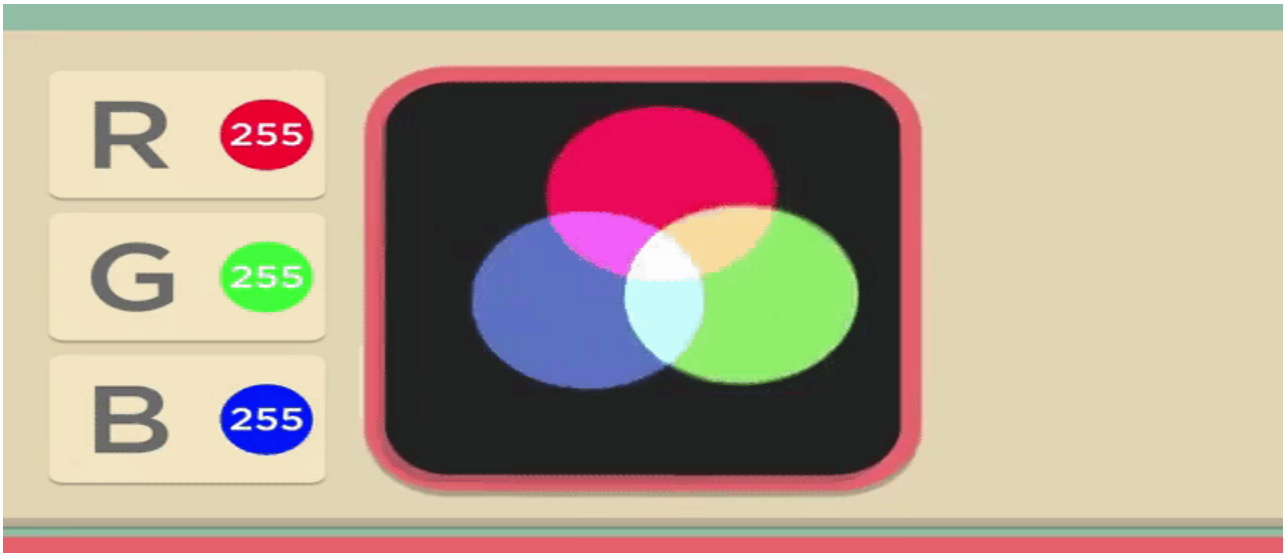
Máy tính lưu trữ hình ảnh như một bức tranh khảm của các ô vuông nhỏ. Điều này giống như hình thức nghệ thuật cổ xưa của gạch khảm, hoặc bộ dụng cụ hạt tan chảy mà trẻ em chơi với ngày nay. Bây giờ, nếu những ô vuông này quá lớn, thì thật khó để tạo ra các cạnh và đường cong mượt mà. Chúng ta càng sử dụng nhiều gạch nhỏ hơn, mịn hơn hoặc như chúng ta nói ít pixel hơn, hình ảnh sẽ có. Chúng đôi khi được gọi là độ phân giải của hình ảnh.
Đồ họa vector là một phương pháp lưu trữ hình ảnh hơi khác nhau nhằm tránh các vấn đề liên quan đến pixel. Nhưng ngay cả hình ảnh vector, cuối cùng, được hiển thị dưới dạng khảm các pixel. Pixel từ có nghĩa là một **yếu tố hình ảnh** . Một cách đơn giản để mô tả mỗi pixel là sử dụng kết hợp ba màu, cụ thể là **Red, Green, Blue**. Đây là những gì chúng ta gọi là một RGBhình ảnh.
Trong một hình ảnh RGB, mỗi pixel được biểu thị bằng ba 8 bitsố được liên kết với các giá trị **Red, Green, Blue**tương ứng. Cuối cùng, bằng cách sử dụng kính lúp, nếu chúng ta phóng to hình ảnh, chúng ta sẽ thấy hình ảnh được tạo thành từ những chấm nhỏ của ánh sáng nhỏ hoặc cụ thể hơn là các pixel. Điều thú vị hơn là thấy rằng những chấm nhỏ của ánh sáng nhỏ thực sự là nhiều chấm nhỏ của ánh sáng nhỏ có màu sắc khác nhau, không gì khác ngoài **Red, Green, Blue**các kênh.



Pixel cùng nhau từ xa tạo ra một hình ảnh và trả trước, chúng chỉ là những đèn nhỏ **BẬT và TẮT** . Sự kết hợp của những thứ tạo ra hình ảnh và về cơ bản những gì chúng ta thấy trên màn hình mỗi ngày.
Mỗi bức ảnh, ở dạng kỹ thuật số, được tạo thành từ các pixel. Chúng là đơn vị thông tin nhỏ nhất tạo nên một bức tranh. Thông thường hình tròn hoặc hình vuông, chúng thường được sắp xếp theo dạng lưới 2 chiều.
Bây giờ, nếu cả ba giá trị đều ở cường độ tối đa, điều đó có nghĩa là 255. Sau đó, nó hiển thị dưới dạng màu trắng và nếu cả ba màu bị tắt hoặc có giá trị 0, màu sẽ hiển thị là màu đen. Sự kết hợp của ba thứ này sẽ lần lượt cho chúng ta một sắc thái cụ thể của màu pixel. Vì mỗi số là một số **8 bit** , nên các giá trị nằm trong khoảng **0-255**.



Sự kết hợp của ba màu này có xu hướng giá trị cao nhất trong số chúng. Vì mỗi giá trị có thể có 256 giá trị cường độ hoặc độ sáng khác nhau, nó tạo ra **16,8** triệu sắc thái tổng.



Ở đây, chúng tôi sẽ quan sát một số điều sau đây, đó là phân tích dữ liệu hình ảnh cơ bản với Numpyvà một số mối quan tâm về Python, như imageio, matplotlibv.v.

- Nhập hình ảnh và quan sát thuộc tính của nó
- Tách các lớp
- Tòa nhà màu xám
- Sử dụng toán tử logic trên các giá trị pixel
- Tạo mặt nạ bằng Toán tử logic
- Phân tích dữ liệu ảnh vệ tinh

1.2. Tải ảnh

Bây giờ hãy tải một hình ảnh và quan sát các thuộc tính khác nhau của nó nói chung.

```
if __name__ == '__main__':
import imageio
import matplotlib.pyplot as plt
%matplotlib inline
pic = imageio.imread('F:/demo_2.jpg')
plt.figure(figsize = (15,15))
plt.imshow(pic)
```

Quan sát các thuộc tính cơ bản của hình ảnh

```
print('Type of the image : ' , type(pic))
print('Shape of the image : {}'.format(pic.shape))
print('Image Hight {}'.format(pic.shape[0]))
print('Image Width {}'.format(pic.shape[1]))
```

```
print('Dimension of Image {}'.format(pic.ndim))
```

```
Type of the image : <class 'imageio.core.util.Image'>

Shape of the image : (562, 960, 3)

Image Hight 562

Image Width 960

Dimension of Image 3
```

Hình dạng của ndarray cho thấy nó là một ma trận ba lớp. Hai số đầu tiên ở đây là chiều dài và chiều rộng và số thứ ba (tức là 3) dành cho ba lớp : **Red, Green, Blue**. Vì vậy, nếu chúng ta tính kích thước của hình ảnh RGB, tổng kích thước sẽ được tính là**height x width x 3**

```
print('Image size {}'.format(pic.size))

print('Maximum RGB value in this image {}'.format(pic.max()))

print('Minimum RGB value in this image {}'.format(pic.min()))
```

```
Image size 1618560

Maximum RGB value in this image 255

Minimum RGB value in this image 0
```

Các giá trị này rất quan trọng để xác minh vì cường độ màu tám bit không thể nằm ngoài phạm vi từ 0 đến 255.

Bây giờ, bằng cách sử dụng biến được gán hình ảnh, chúng tôi cũng có thể truy cập bất kỳ giá trị pixel cụ thể nào của hình ảnh và có thể truy cập thêm từng kênh **RGB** riêng biệt.

```
'''

Let's pick a specific pixel located at 100 th Rows and 50 th Column.

And view the RGB value gradually.

'''

pic[ 100, 50 ]
```

```
Image([109, 143,  46], dtype=uint8)
```

Trong trường hợp này: R = 109; G = 143; B = 46 và chúng ta có thể nhận ra rằng pixel đặc biệt này có rất nhiều màu XANH LÁ trong đó. Bây giờ, chúng tôi cũng có thể đã chọn một trong những số này một cách cụ thể bằng cách đưa ra giá trị chỉ mục của ba kênh này. Bây giờ chúng tôi biết điều này:

- 0giá trị chỉ mục cho kênh **Đỏ**
- 1giá trị chỉ mục cho kênh **Xanh**
- 2giá trị chỉ mục cho kênh **Blue**

Tuy nhiên, thật tốt khi biết rằng trong OpenCV, Hình ảnh không phải là RGB mà là BGR. imageio.imread tải hình ảnh dưới dạng RGB (hoặc RGBA), nhưng OpenCV giả định hình ảnh là BGR hoặc BGRA (BGR là định dạng màu OpenCV mặc định).

```
# A specific pixel located at Row : 100 ; Column : 50
# Each channel's value of it, gradually R , G , B
print('Value of only R channel {}'.format(pic[ 100, 50, 0]))
print('Value of only G channel {}'.format(pic[ 100, 50, 1]))
print('Value of only B channel {}'.format(pic[ 100, 50, 2]))
```

```
Value of only R channel 109
Value of only G channel 143
Value of only B channel 46
```

bây giờ hãy xem nhanh từng kênh trong toàn bộ hình ảnh.

```
plt.title('R channel')
plt.ylabel('Height {}'.format(pic.shape[0]))
plt.xlabel('Width {}'.format(pic.shape[1]))
plt.imshow(pic[ : , : , 0])
plt.show()
```

```
plt.title('G channel')
plt.ylabel('Height {}'.format(pic.shape[0]))
plt.xlabel('Width {}'.format(pic.shape[1]))
plt.imshow(pic[ : , : , 1])
plt.show()
```

```
plt.title('B channel')
plt.ylabel('Height {}'.format(pic.shape[0]))
plt.xlabel('Width {}'.format(pic.shape[1]))
plt.imshow(pic[ : , : , 2])
plt.show()
```

Bây giờ, chúng ta cũng có thể thay đổi số lượng giá trị RGB. Ví dụ: hãy đặt lớp Đỏ, Xanh lục, Xanh lam để theo các giá trị Hàng thành cường độ đầy đủ.

- Kênh R: Hàng - 100 đến 110
- Kênh G: Hàng - 200 đến 210
- Kênh B: Hàng - 300 đến 310

Chúng tôi sẽ tải hình ảnh một lần để chúng tôi có thể hình dung từng thay đổi cùng một lúc.

```
pic = imageio.imread('F:/demo_2.jpg')


pic[50:150 , : , 0] = 255 # full intensity to those pixel's R channel

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

plt.imshow(pic)

plt.show()
```

```
pic[200:300 , : , 1] = 255 # full intensity to those pixel's G channel
plt.figure( figsize = (10,10))
plt.imshow(pic)
plt.show()
```

```
pic[350:450 , : , 2] = 255 # full intensity to those pixel's B channel

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

plt.imshow(pic)

plt.show()
```

Để làm rõ hơn, chúng ta cũng hãy thay đổi phần cột và lần này chúng ta sẽ thay đổi kênh RGB cùng một lúc.

```
# set value 200 of all channels to those pixels which turns them to white

pic[ 50:450 , 400:600 , [0,1,2] ] = 200

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

plt.imshow(pic)

plt.show()
```

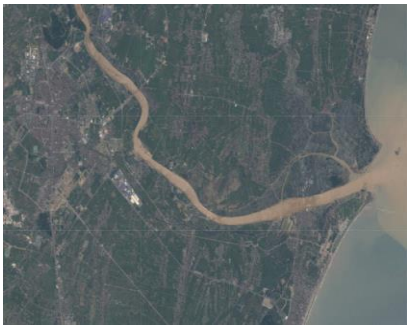
1.3.Chia lớp

Bây giờ, chúng ta biết rằng mỗi pixel của hình ảnh được đại diện bởi ba số nguyên. Việc tách hình ảnh thành các thành phần màu riêng biệt chỉ là vấn đề kéo ra lát cắt chính xác của mảng hình ảnh.

```
import numpy as np
```

```
pic = imageio.imread('F:/demo_2.jpg')

fig, ax = plt.subplots(nrows = 1, ncols=3, figsize=(15,5))

for c, ax in zip(range(3), ax):

# create zero matrix

split_img = np.zeros(pic.shape, dtype="uint8") # 'dtype' by default: 'numpy.float64'

# assing each channel

split_img[ :, :, c] = pic[ :, :, c]

# display each channel

ax.imshow(split_img)
```





Hình ảnh đen trắng được lưu trữ trong mảng 2 chiều. Có hai loại hình ảnh đen trắng:

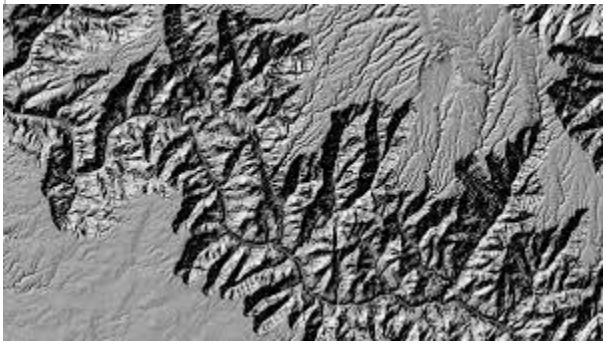
- Greyscale: Phạm vi sắc thái của màu xám: 0~255
- Nhị phân: Pixel có màu đen hoặc trắng: 0hoặc255

Bây giờ, Gre chọc trời là một quá trình mà một hình ảnh được chuyển đổi từ một màu đầy đủ sang các sắc thái của màu xám. Ví dụ, trong các công cụ xử lý hình ảnh: trong OpenCV, nhiều chức năng sử dụng hình ảnh thang độ xám trước khi xử lý và điều này được thực hiện vì nó đơn giản hóa hình ảnh, hoạt động gần như giảm nhiễu và tăng thời gian xử lý vì có ít thông tin hơn trong hình ảnh.

Có một số cách để thực hiện điều này trong python để chuyển đổi hình ảnh thành thang độ xám , nhưng cách sử dụng matplotlib đơn giản là lấy giá trị trung bình của giá trị RGB của hình ảnh gốc bằng công thức này .

```
Y' = 0.299 R + 0.587 G + 0.114 B
pic = imageio.imread('F:/demo_2.jpg')

gray = lambda rgb : np.dot(rgb[... , :3] , [0.299 , 0.587, 0.114])

gray = gray(pic)

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

```
plt.imshow(gray, cmap = plt.get_cmap(name = 'gray'))
plt.show()
```



Tuy nhiên, phần mềm chuyển đổi màu GIMP sang màu xám có ba thuật toán để thực hiện nhiệm vụ.

Độ sáng Graylevel sẽ được tính là

Lightness = ½ × (max(R,G,B) + min(R,G,B))

Độ sáng Độ xám sẽ được tính như

Luminosity = 0.21 × R + 0.72 × G + 0.07 × B

Trung bình Graylevel sẽ được tính là

Average Brightness = (R + G + B) ÷ 3

Hãy thử một trong những thuật toán của họ. Làm thế nào về độ sáng?

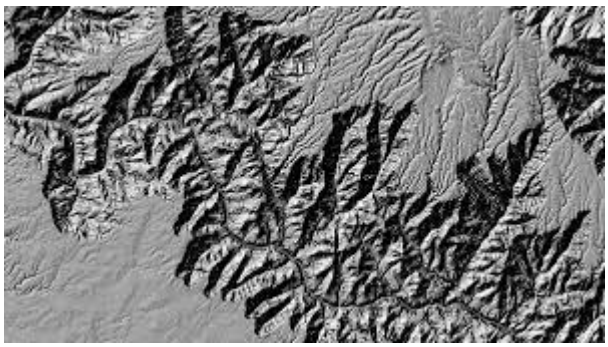
```
pic = imageio.imread('F:/demo_2.jpg')
gray = lambda rgb : np.dot(rgb[... , :3] , [0.21 , 0.72, 0.07])
gray = gray(pic)
plt.figure( figsize = (10,10))
plt.imshow(gray, cmap = plt.get_cmap(name = 'gray'))
plt.show()
'''
```

Let's take a quick overview some the changed properties now the color image.

Like we observe some properties of color image, same statements are applying

now for gray scaled image.

```
'''
print('Type of the image : ' , type(gray))
print()
print('Shape of the image : {}'.format(gray.shape))
print('Image Hight {}'.format(gray.shape[0]))
print('Image Width {}'.format(gray.shape[1]))
print('Dimension of Image {}'.format(gray.ndim))
print()
print('Image size {}'.format(gray.size))
print('Maximum RGB value in this image {}'.format(gray.max()))
print('Minimum RGB value in this image {}'.format(gray.min()))
print('Random indexes [X,Y] : {}'.format(gray[100, 50]))
```



```
Type of the image : <class 'imageio.core.util.Image'>
Shape of the image : (562,960)
Image Height 562
Image Widht 960
Dimension of Image 2
Image size 539520
Maximum RGB value in this image 254.9999999997
Minimum RGB value in this image 0.0
Random indexes [X,Y] : 129.07
```

1.4. Sử dụng toán tử logic để xử lý các giá trị pixel

Chúng ta có thể tạo một ndarray bullion trong cùng kích thước bằng cách sử dụng một **toán tử logic**. Tuy nhiên, điều này sẽ không tạo ra bất kỳ mảng mới nào, nhưng nó chỉ đơn giản trở về True. Ví dụ: hãy xem xét chúng tôi muốn lọc ra một số pixel có giá trị thấp hoặc giá trị cao hoặc (bất kỳ điều kiện nào) trong hình ảnh RGB, và vâng, thật tuyệt vời khi chuyển đổi RGB sang thang độ xám, nhưng bây giờ, chúng tôi sẽ không đi cho điều đó hơn là đối phó với một hình ảnh màu.

Trước tiên hãy tải một hình ảnh và hiển thị nó trên màn hình.

```
pic = imageio.imread('F:/demo_1.jpg')

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

plt.imshow(pic)

plt.show()
```

Trong mọi trường hợp, chúng tôi muốn lọc ra tất cả các giá trị pixel, thấp hơn, giả sử, 20. Đối với điều này, chúng tôi sẽ sử dụng một toán tử logic để thực hiện nhiệm vụ này, chúng tôi sẽ trả về làm giá trị của Truecho tất cả các chỉ số.

```
low_pixel = pic < 20

# to ensure of it let's check if all values in low_pixel are True or not

if low_pixel.any() == True:

print(low_pixel.shape)
```

(1079, 1293, 3)

Như chúng ta đã nói, một biến chủ không được sử dụng theo truyền thống, nhưng tôi đề cập đến nó bởi vì nó hoạt động. Nó chỉ giữ giá trị thật và không có gì khác. Vì vậy, nếu chúng ta thấy shapecả hai low_pixelvà pic, chúng ta sẽ thấy rằng cả hai đều giống nhau shape.

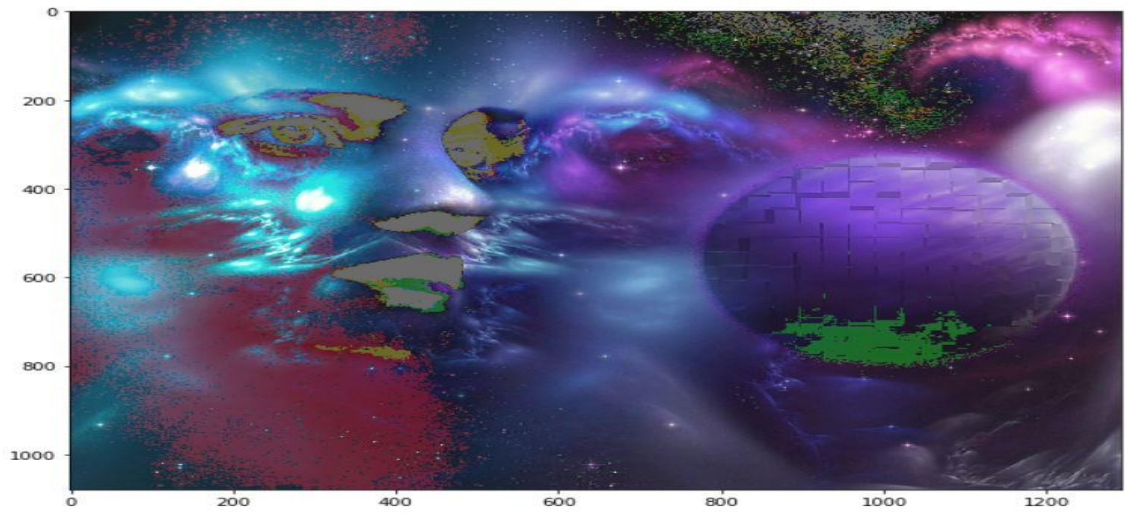
```
print(pic.shape)

print(low_pixel.shape)
```

(1079, 1293, 3)

```
(1079, 1293, 3)
```

Chúng tôi đã tạo bộ lọc giá trị thấp đó bằng cách sử dụng toán tử so sánh toàn cục cho tất cả các giá trị nhỏ hơn 200. Tuy nhiên, chúng tôi có thể sử dụng low_pixelmảng này làm chỉ mục để đặt các giá trị thấp đó thành một số giá trị cụ thể, có thể cao hơn hoặc thấp hơn trước đó giá trị pixel.

```
# randomly choose a value

import random

# load the orginal image

pic = imageio.imread('F:/demo_1.jpg')

# set value randomly range from 25 to 225 - these value also randomly choosen

pic[low_pixel] = random.randint(25,225)

# display the image

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

plt.imshow(pic)

plt.show()
```



1.5. Mặt nạ

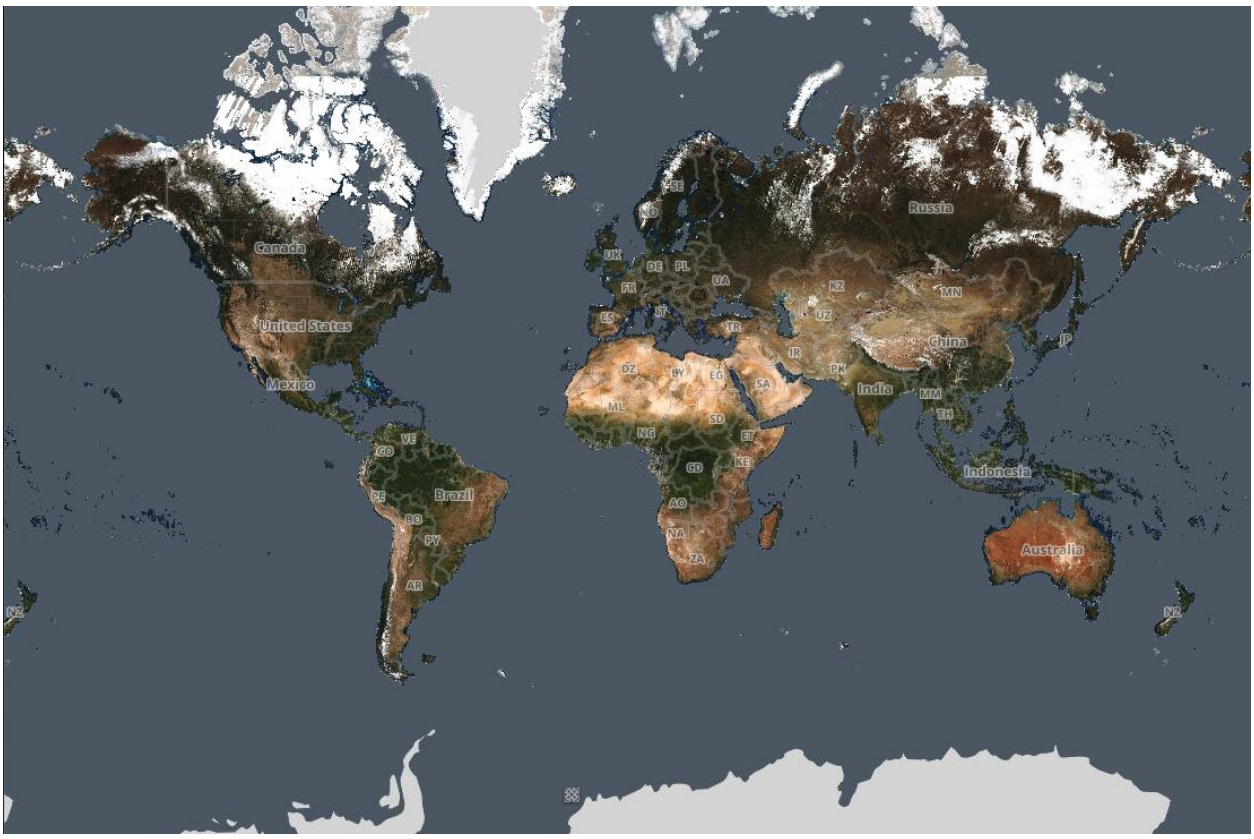
Mặt nạ hình ảnh là một kỹ thuật xử lý hình ảnh được sử dụng để loại bỏ nền mà từ đó các bức ảnh có các cạnh mờ, trong suốt hoặc các phần tóc.

Bây giờ, chúng ta sẽ tạo một mặt nạ có hình đĩa tròn. Trước tiên, chúng tôi sẽ đo khoảng cách từ trung tâm của hình ảnh đến mọi giá trị pixel viền. Và chúng tôi lấy một giá trị bán kính thuận tiện, sau đó sử dụng toán tử logic, chúng tôi sẽ tạo một đĩa tròn. Nó khá đơn giản, hãy xem mã.

```
if __name__ == '__main__':
# load the image
pic = imageio.imread('F:/demo_1.jpg')
# seperate the row and column values
total_row , total_col , layers = pic.shape
'''
Create vector.
Ogrid is a compact method of creating a multidimensional-
ndarray operations in single lines.
for ex:
>>> ogrid[0:5,0:5]
output: [array([[0],
[1],
[2],
[3],
[4]]),
array([[0, 1, 2, 3, 4]])]
'''
x , y = np.ogrid[:total_row , :total_col]
# get the center values of the image
cen_x , cen_y = total_row/2 , total_col/2
'''
Measure distance value from center to each border pixel.
To make it easy, we can think it's like, we draw a line from center-
to each edge pixel value --> s**2 = (Y-y)**2 + (X-x)**2
'''
distance_from_the_center = np.sqrt((x-cen_x)**2 + (y-cen_y)**2)
# Select convenient radius value
```

```
radius = (total_row/2)


# Using logical operator '>'
'''
logical operator to do this task which will return as a value
of True for all the index according to the given condition
'''
circular_pic = distance_from_the_center > radius
'''
let assign value zero for all pixel value that outside the cirular disc.
All the pixel value outside the circular disc, will be black now.
'''
pic[circular_pic] = 0
plt.figure(figsize = (10,10))
plt.imshow(pic)
plt.show()
```

CHƯƠNG 2. PHÂN TÍCH ẢNH VỆ TINH VỚI PYTHON



2.1. Hình ảnh vệ tinh: Tổng quan

Hình ảnh vệ tinh là hình ảnh của Trái đất (hoặc các hành tinh khác) được thu thập bởi các vệ tinh hình ảnh. Chính phủ hoặc các công ty tư nhân có thể sở hữu các vệ tinh này. Các công ty hình ảnh vệ tinh bán hình ảnh bằng cách cấp phép cho các chính phủ và doanh nghiệp như Apple Maps và Google Maps.

Thiết lập hệ thống

Các thư viện sau đây được yêu cầu để chạy dự án này:

Planet's Python Client

Rasterio: Các hệ thống thông tin địa lý sử dụng GeoTIFF và các định dạng khác để tổ chức và lưu trữ các bộ dữ liệu raster có lưới như hình ảnh vệ tinh và mô hình địa hình. Rasterio là một thư viện Python đọc và ghi các định dạng này và cung cấp API Python dựa trên mảng Numpy N-chiều và GeoJSON.

numpy

matplotlib

requests

2.2. Lấy dữ liệu

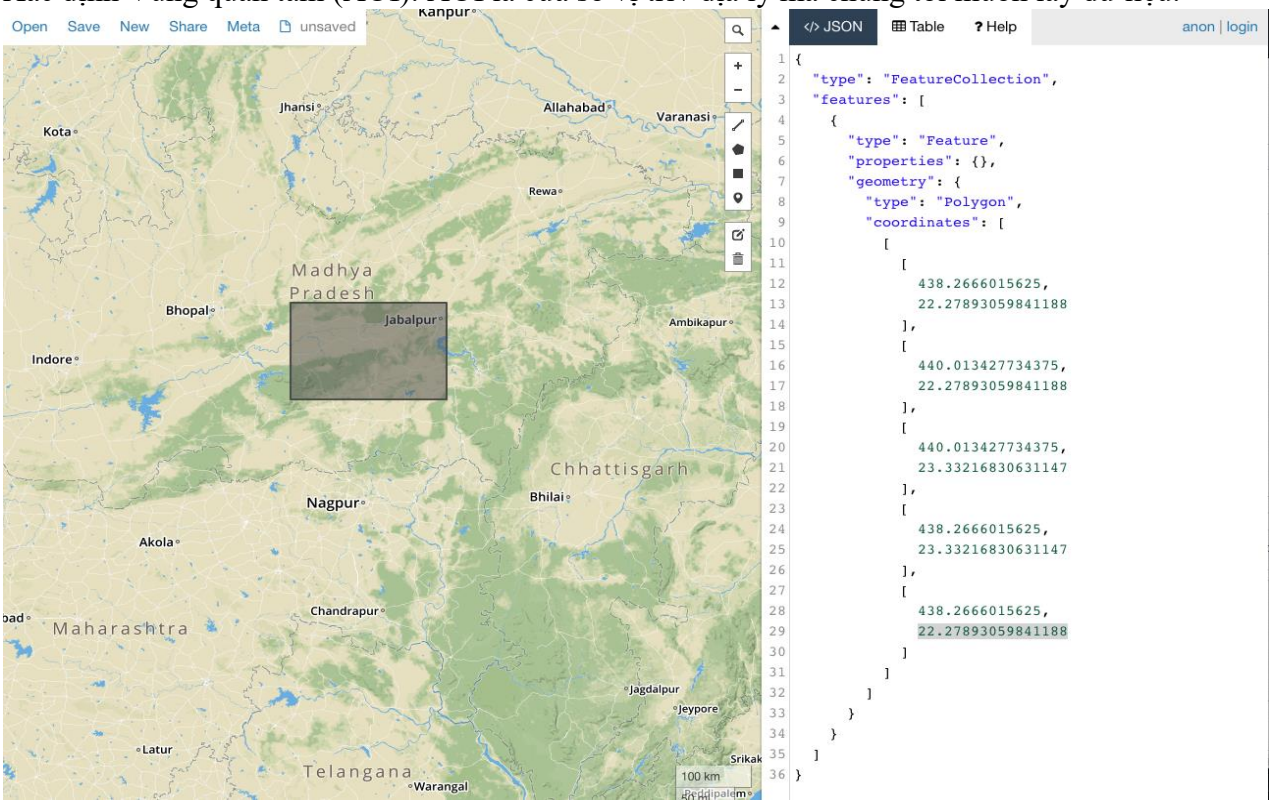
Đối với trường hợp cụ thể này, chúng tôi sẽ làm việc với Dữ liệu Phản xạ Bề mặt (SR). Nói một cách đơn giản, dữ liệu SR là dữ liệu vệ tinh đã được sửa chữa bằng thuật toán để loại bỏ bất kỳ sự can thiệp nào khỏi bầu khí quyển.

Dữ liệu được sử dụng trong bài tập này đã được tải xuống từ Planet Explorer. Planet Explorer là một sản phẩm của phòng thí nghiệm sản phẩm và được sử dụng để khám phá hình ảnh hàng ngày ngay trong trình duyệt của chúng tôi. Các phòng thí nghiệm trên hành tinh vận hành đội vệ tinh chụp ảnh Trái đất lớn nhất và dữ liệu do họ cung cấp được sử dụng để theo dõi thảm thực vật để đo lường sản lượng nông nghiệp.

Các bước cần thiết để tải xuống dữ liệu hình ảnh.

Mở liên kết: Geojson.io. Đây là trình chỉnh sửa thời gian nhanh cho dữ liệu bản đồ

Xác định Vùng quan tâm (AOI): AOI là cửa sổ vị trí / địa lý mà chúng tôi muốn lấy dữ liệu.



Lưu tọa độ AOI được tạo ở định dạng GeoJSON trong sổ ghi chép Jupyter

```
# AOI co-ordinates (created via geojson.io)
geojson_geometry = {
"type": "Polygon",
"coordinates": [
    [
      [438.2666015625,22.27893059841188],
      [440.013427734375,22.27893059841188],
      [440.013427734375,23.33216830631147],
      [438.2666015625,23.33216830631147],
      [438.2666015625,22.27893059841188]
    ]
  ]
}
```

Tạo các bộ lọc cho phạm vi ngày, vùng phủ sóng và hình học. Điều này sẽ cho phép chúng tôi tiếp tục hạn chế tìm kiếm API dữ liệu của mình.

```
# get images that overlap with our AOI
geometry_filter = {
  "type": "GeometryFilter",
 "field_name": "geometry",
```

```
  "config": geojson_geometry
}

# get images acquired within a date range
date_range_filter = {
  "type": "DateRangeFilter",
  "field_name": "acquired",
  "config": {
    "gte": "2016-08-31T00:00:00.000Z",
    "lte": "2016-09-01T00:00:00.000Z"
  }
}

# only get images which have <50% cloud coverage
cloud_cover_filter = {
  "type": "RangeFilter",
  "field_name": "cloud_cover",
  "config": {
    "lte": 0.5
  }
}

# combine our geo, date, cloud filters
combined_filter = {
  "type": "AndFilter",
  "config": [geometry_filter, date_range_filter, cloud_cover_filter]
}
```

2.3. Khóa API Planet

Để sử dụng API planet, bạn sẽ cần một khóa API. Tạo một tài khoản (dùng thử 14 ngày) tại Planet Explorer và truy cập khóa API từ đây.

Tìm kiếm: **Items and Assets**

Các hình ảnh được chụp bởi các vệ tinh có thể được phân loại thành **Items and Assets**. Trong khi các mục đề cập đến một quan sát duy nhất được chụp bởi vệ tinh, các tài sản mô tả một sản phẩm có thể được lấy từ dữ liệu nguồn của vật phẩm và có thể được sử dụng cho các mục đích phân tích, trực quan hoặc khác

Trong trường hợp của chúng tôi, chúng tôi sẽ thử và có được một hình ảnh về các hoạt động phân tích có thể được tiến hành. Do đó, chúng tôi muốn có một hình ảnh 4 băng tần với dữ liệu phổ cho các giá trị Đỏ, Xanh lục, Xanh lam và Gần hồng ngoại. Để có được hình ảnh mà chúng tôi muốn, chúng tôi sẽ chỉ định một loại mặt hàng của PSScene4Band và loại tài sản.analytic

```
import os
import json
import requests
from requests.auth import HTTPBasicAuth

# API Key stored as an env variable
PLANET_API_KEY = os.getenv('PL_API_KEY') # replace PL_API_KEY with Planet API
key in quotes

item_type = "PSScene4Band"

# API request object
search_request = {
  "interval": "day",
  "item_types": [item_type],
```

```
  "filter": combined_filter
}

# fire off the POST request
search_result = \
  requests.post(
    'https://api.planet.com/data/v1/quick-search',
    auth=HTTPBasicAuth(PLANET_API_KEY, ''),
    json=search_request)

print(json.dumps(search_result.json(), indent=1))
```

2.4. Kích hoạt và tải xuống hình ảnh

Để tải hình ảnh, chúng ta cần kích hoạt nó. Khi trạng thái kích hoạt trở thành kích hoạt, thì chúng tôi có thể tải xuống hình ảnh quan tâm.

Khi trạng thái kích hoạt thay đổi thành Chế độ hoạt động của người dùng từ người dùng không hoạt động, bạn có thể tải xuống hình ảnh ở định dạng .tiff.

```
links = result.json()[u"analytic"]["_links"]
self_link = links["_self"]
activation_link = links["activate"]

# Request activation of the 'analytic' asset:
activate_result = \
  requests.get(
    activation_link,
    auth=HTTPBasicAuth(PLANET_API_KEY, '')
  )

  activation_status_result = \
  requests.get(
    self_link,
    auth=HTTPBasicAuth(PLANET_API_KEY, '')
  )

print(activation_status_result.json()["status"])
```

2.5.Khám phá hình ảnh vệ tinh

Thư viện python sườn Rasterio giúp bạn dễ dàng khám phá ảnh vệ tinh. Hình ảnh vệ tinh không là gì ngoài lưới các giá trị pixel và do đó có thể được hiểu là mảng đa chiều.

Importing the image

In [2]:

```
import math
import rasterio
import matplotlib.pyplot as plt

image_file = "image.tif"
sat_data = rasterio.open(image_file)
```

Calculating the dimensions of the image on earth in metres

In [2]:

```
width_in_projected_units = sat_data.bounds.right - sat_data.bounds.left
```

```
height_in_projected_units = sat_data.bounds.top - sat_data.bounds.bottom

print("Width: {}, Height: {}".format(width_in_projected_units, height_in_proj
ected_units))
```
```
Width: 19347.0, Height: 17406.0
```
Rows and Columns

In [3]:

```
print("Rows: {}, Columns: {}".format(sat_data.height, sat_data.width))
```
```
Rows: 5802, Columns: 6449
```
Converting the pixel co-ordinates to longitudes and latitudes

In [4]:

```
# Upper left pixel
row_min = 0
col_min = 0

# Lower right pixel.  Rows and columns are zero indexing.
row_max = sat_data.height - 1
col_max = sat_data.width - 1

# Transform coordinates with the dataset's affine transformation.
topleft = sat_data.transform * (row_min, col_min)
botright = sat_data.transform * (row_max, col_max)

print("Top left corner coordinates: {}".format(topleft))
print("Bottom right corner coordinates: {}".format(botright))
```
```
Top left corner coordinates: (505761.0, 2180130.0)
Bottom right corner coordinates: (523164.0, 2160786.0)
```
Bands

The image that we are inspecting is a multispectral image consisting of 4 bands int he order B,G,R,N where N stands for near infrared.each band is stored as a numpy array.

In [5]:

```
print(sat_data.count)

# sequence of band indexes
print(sat_data.indexes)
```
```
4
(1, 2, 3, 4)
```
Visualising the Satellite Imagery

We will use matplotlib to visualise the image since it essentially consists of arrays.

In [4]:

```
# Load the 4 bands into 2d arrays - recall that we previously learned PlanetS
cope band order is BGRN.
b, g, r, n = sat_data.read()
```

In [7]:

```
# Displaying the blue band.

fig = plt.imshow(b)
plt.show()
```

In [8]:

```
# Displaying the green band.

fig = plt.imshow(g)
fig.set_cmap('gist_earth')
plt.show()
```

In [5]:

```
# Displaying the red band.

fig = plt.imshow(r)
fig.set_cmap('inferno')
plt.colorbar()
plt.show()
```

In [9]:

```
# Displaying the infrared band.

fig = plt.imshow(n)
fig.set_cmap('winter')
plt.colorbar()
plt.show()
```

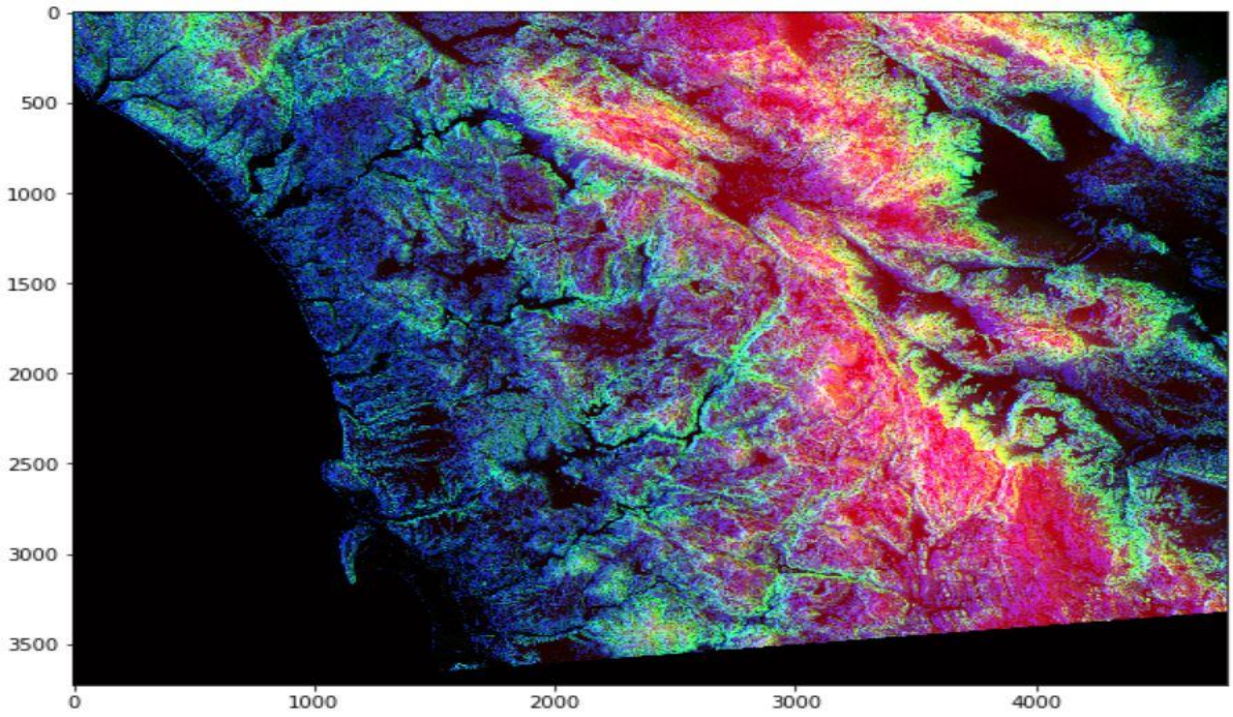
In [ ]:

2.6.Xử lý ảnh vệ tinh

Tải ảnh vệ tinh.

```
# load the image
```

```
pic = imageio.imread('F:\satimg.jpg')
plt.figure(figsize = (10,10))
plt.imshow(pic)
plt.show()
```



Chúng ta hãy xem một số thông tin cơ bản:

```
print(f'Shape of the image {pic.shape}')
print(f'hieght {pic.shape[0]} pixels')
print(f'width {pic.shape[1]} pixels')
```

```
Shape of the image (3725, 4797, 3)
hieght 3725 pixels
width 4797 pixels
```

Có một cái gì đó thú vị về hình ảnh này. Giống như nhiều hình ảnh trực quan khác, màu sắc trong mỗi lớp RGB có ý nghĩa gì đó. Ví dụ: cường độ của màu đỏ sẽ là một dấu hiệu cho thấy độ cao của điểm dữ liệu địa lý trong pixel. Cường độ của màu xanh sẽ biểu thị thước đo của khía cạnh và màu xanh lá cây sẽ biểu thị độ dốc. Những màu này sẽ giúp truyền đạt thông tin này một cách nhanh chóng và hiệu quả hơn là hiển thị số.

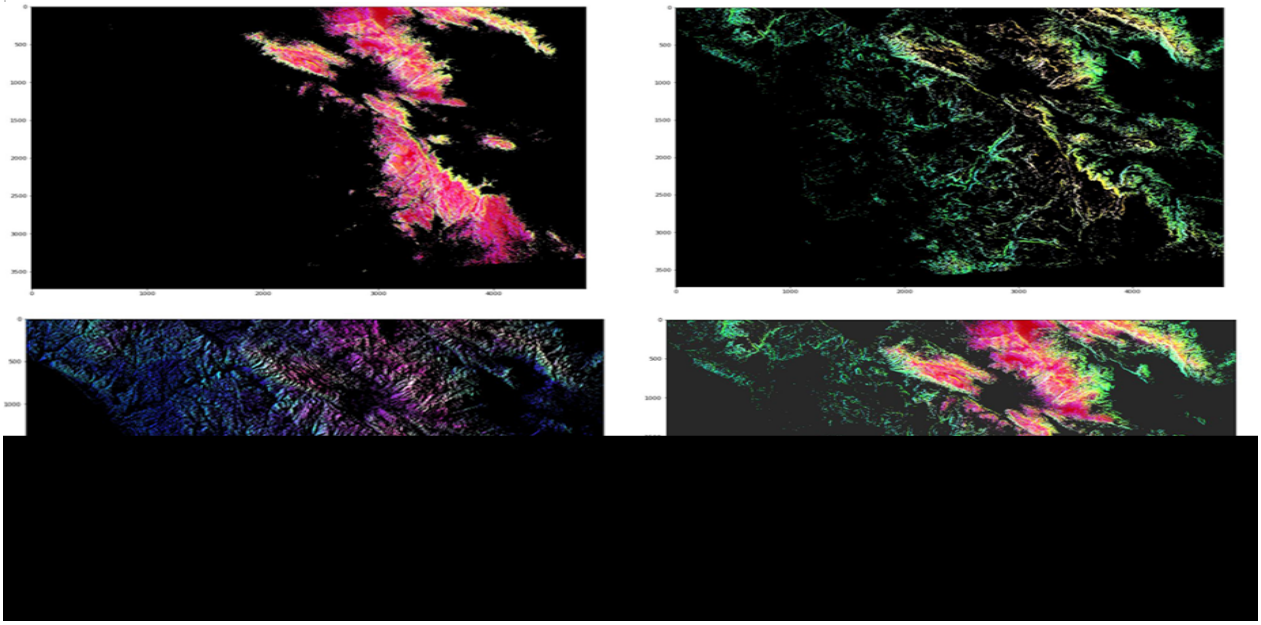
- Pixel đỏ cho biết: **Độ cao**
- Pixel màu xanh biểu thị: **Aspect**
- Pixel màu xanh lá cây biểu thị: **Độ dốc**

Có điều, chỉ cần nhìn vào hình ảnh đầy màu sắc này, một con mắt được đào tạo có thể biết được độ cao là gì, độ dốc là gì và khía cạnh là gì. Vì vậy, đó là ý tưởng tải một số ý nghĩa hơn cho các màu này để chỉ ra một cái gì đó khoa học hơn.

Phát hiện pixel cao của mỗi kênh

```
# Only Red Pixel value , higher than 180

pic = imageio.imread('F:\satimg.jpg')

red_mask = pic[:, :, 0] < 180

pic[red_mask] = 0

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

plt.imshow(pic)

# Only Green Pixel value , higher than 180

pic = imageio.imread('F:\satimg.jpg')

green_mask = pic[:, :, 1] < 180

pic[green_mask] = 0

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

plt.imshow(pic)

# Only Blue Pixel value , higher than 180

pic = imageio.imread('F:\satimg.jpg')

blue_mask = pic[:, :, 2] < 180

pic[blue_mask] = 0

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

plt.imshow(pic)

# Composite mask using logical_and

pic = imageio.imread('F:\satimg.jpg')

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

pic[final_mask] = 40
```

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



2.7. Tính toán chỉ số thực vật từ hình ảnh vệ tinh

2.7.1. Chỉ số thực vật

Chỉ số thực vật là một chỉ số về màu xanh của bất kỳ khu vực nào. Đó là một biện pháp để theo dõi sức khỏe của thảm thực vật. Một loạt dữ liệu được thu thập bởi các cảm biến vệ tinh và một loại dữ liệu như vậy đặc biệt đo các bước sóng ánh sáng được hấp thụ và phản xạ bởi các cây xanh.

Thảm thực vật dày đặc phản chiếu rất nhiều ánh sáng cận hồng ngoại (không nhìn thấy được bằng mắt người) so với ánh sáng đỏ nhìn thấy được. Điều ngược lại xảy ra trong trường hợp thảm thực vật thưa thớt. Do đó, khi một tán cây thay đổi từ tăng trưởng đầu mùa xuân sang trưởng thành vào cuối mùa và tuổi già, các tính chất phản xạ này cũng thay đổi.

2.7.2. NDVI

Một trong những chỉ số được sử dụng rộng rãi nhất để đo thực vật là Chỉ số thực vật khác biệt bình thường hóa (NDVI). các giá trị NDVI nằm trong khoảng từ +1.0 đến -1.0. Nó được phát triển bởi

nhà khoa học NASA Compton Tucker vào năm 1977 và có nguồn gốc từ hình ảnh vệ tinh. Nó có thể được thể hiện như sau.

$$NDVI = \frac{\rho_{NIR} - \rho_{Red}}{\rho_{NIR} + \rho_{Red}}$$

NDVI so sánh ánh sáng cận hồng ngoại phản xạ với ánh sáng đỏ có thể nhìn thấy được.

<=0.1	0.2 to 0.5	0.6 to 0.9
Areas of barren rock, sand, or snow	Sparse vegetation such as shrubs and grasslands or senescing crops	dense vegetation such as that found in temperate and tropical forests or crops at their peak growth stage.

Lợi ích của NDVI

Các giá trị NDVI đưa ra ước tính sơ bộ về loại, số lượng và tình trạng của thảm thực vật tại một nơi rất hữu ích cho các nhà nghiên cứu.

Các giá trị NDVI cũng có thể được tính trung bình theo thời gian để thiết lập các điều kiện phát triển trên một khu vực trong một thời gian nhất định trong năm. Điều này chủ yếu giúp xác định các khu vực có sự thay đổi trong thảm thực vật do các hoạt động của con người như phá rừng, xáo trộn tự nhiên như cháy rừng hoặc thay đổi trong giai đoạn hiện tượng của nhà máy.

Tính toán NDVI cho Khu vực quan tâm của chúng tôi

Chúng tôi đã có dữ liệu được tải xuống dưới dạng hình ảnh .tiff. Trong phần này, chúng ta sẽ tính toán và chỉ số NDVI và phân tích nó.

Toàn bộ mã cũng có sẵn trên Github tại https://github.com/parulnith/Satocate-Imagery-Analysis-with-Python

Dữ liệu hình ảnh vệ tinh có thể được phân tích trong một khoảng thời gian để tìm hiểu nguyên nhân của sự suy giảm thảm thực vật cho một khu vực. Tương tự, phân tích cũng có thể cho phép chúng tôi chỉ ra nếu có nạn phá rừng nghiêm trọng ở bất kỳ khu vực nào có thể dẫn đến ảnh hưởng của sự nóng lên toàn cầu. Dự đoán về bão, hạn hán và lũ lụt là các lĩnh vực khác, nơi phân tích hình ảnh vệ tinh đang được áp dụng rộng rãi. Không có cách nào tốt hơn để sử dụng công nghệ hơn là làm việc trên một số vấn đề thực sự đe dọa hành tinh và có thể sử dụng dữ liệu từ các vệ tinh là một bước theo hướng đó.

Tài liệu tham khảo:

[1]. This article is an adaptation of the wonderful talk given by Sara on Satellite Imagery analysis in Scipy 2018 — Satellite Image analysis with Python,
[2]. https://earthobservatory.nasa.gov/features/MeasuringVegetation/measuring_vegetation_3.php https://phenology.cr.usgs.gov/ndvi_foundation.php

[3]. https://developers.arcgis.com/python/guide/working-with-feature-layers-and-features/
[4].https://developers.arcgis.com/python/sample-notebooks/predict-floods-with-unit-hydrographs/