Coding Assignment-3:Seam Carving

A) Reflection:

- I have completed the coding assignment on seam carving. Seam Carving is an algorithm for image resizing. It works by establishing a number of seams which are paths of lowest energy from one end to other in the image. Each iteration of polt_seam function gives one seam in the image. These seams are then removed using the remove_seam function. With each iteration of remove_seam, the width of image reduces by 1.
- Dynamic Programming is used in seam carving. Since we are working for calculation of vertical seams(paths), for each pixel in a row, we see the three possible pixels above it(except for boundary condition when we see 2 possible pixels). The assignment gave me a deep understanding into image processing. Working on the project gave me a very deep insight into dynamic programming. I became familiar with the table population and backtracking in a very good way. Before the assignment, I had no idea about seam carving. I read about the algorithm in the book and implemented it and now I have a very nice understanding of it.
- My biggest challenge in the assignment still is reducing the flake8 complexity. I do not have any warnings for complexity 9, and I am trying to reduce it to 8(the complexity of plot_seam() function is not coming to 8, it is 9). This is probably due to the use of intense looping in the function needed to back-track the array. A complexity of 9 is acceptable, but I am still figuring out whether an even more efficient way exists or not.

Also previously I had no idea on how to plot images in Python. I used the functions and came to know of the concepts. Additionally I came to know of python functions like hsobel and vsobel.

On the whole, the assignment was very interesting, with each line of code, we had a different output. Writing the code for manipulating the image and carrying out seam carving really interested me and I would love to do such assignments in the future.

Coreman Question

15-8 a) Show that the number of such possible seams grows at-least exponentially in m, assuming that n>1.

Solution) We can conclude this when we say that for each pixel, there are either 2(boundary condition) or 3 options to compute the energy or seam value using Dynamic Programming.

Thus the number of possible seams will be at-least 2n and thus they grow exponentially.

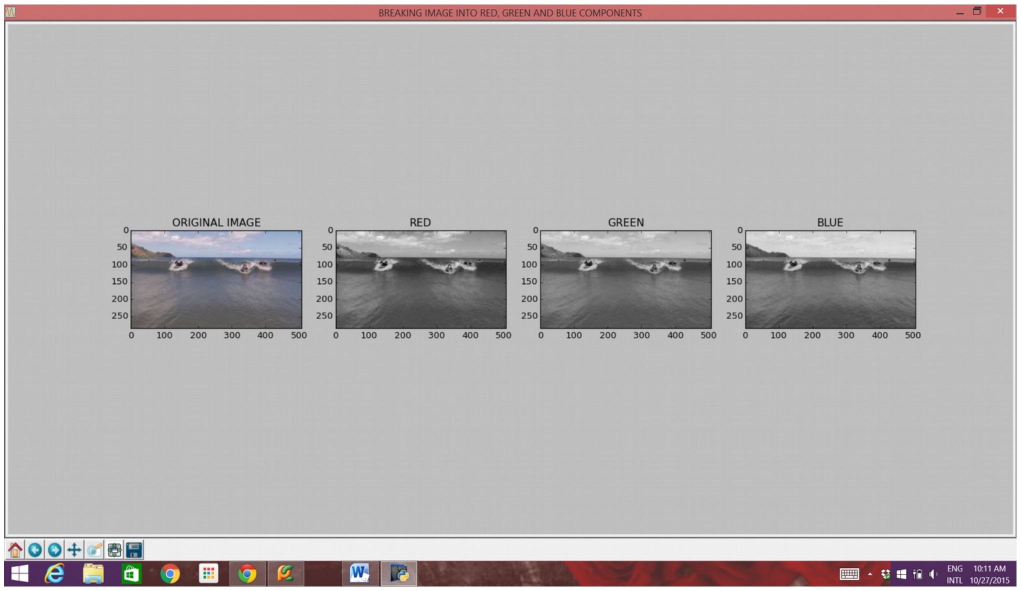
15.8-b) Give an algorithm to find a seam with the lowest disruption measure. How efficient is your algorithm?

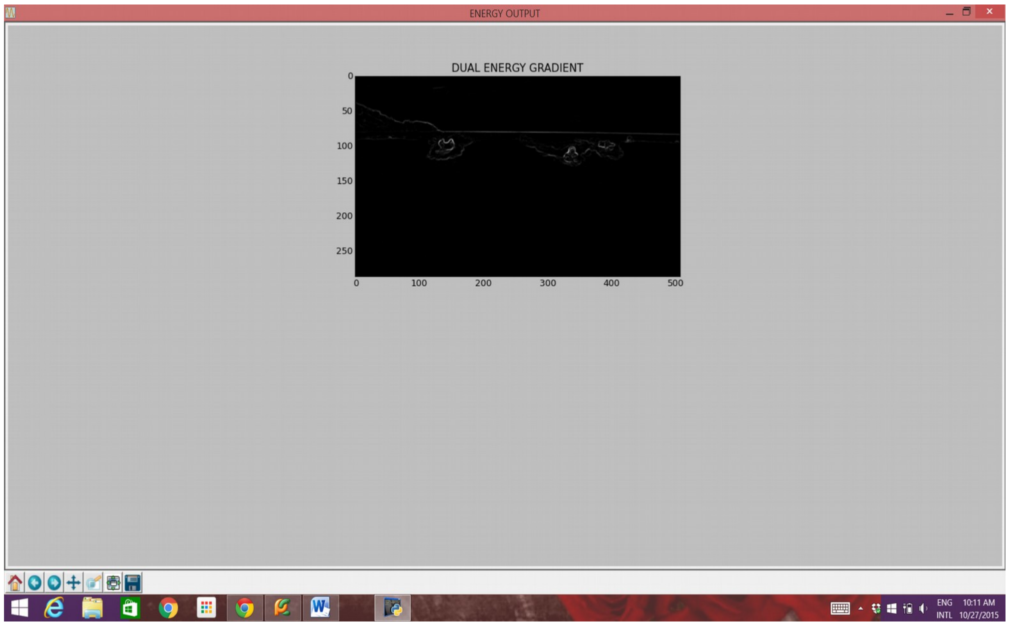
Solution) For each row, except the first row(it does not have any rows above it, so the value will be same for all pixels in that row), we need to find out which pixel among the three ([i-1,j-1],[i-1,j] and [i-1,j+1] whichever are reachable) have the lowest disruption measure. This measure will be added to the current pixels's disruption measure and the index of the pixel from where the value was taken is stored.

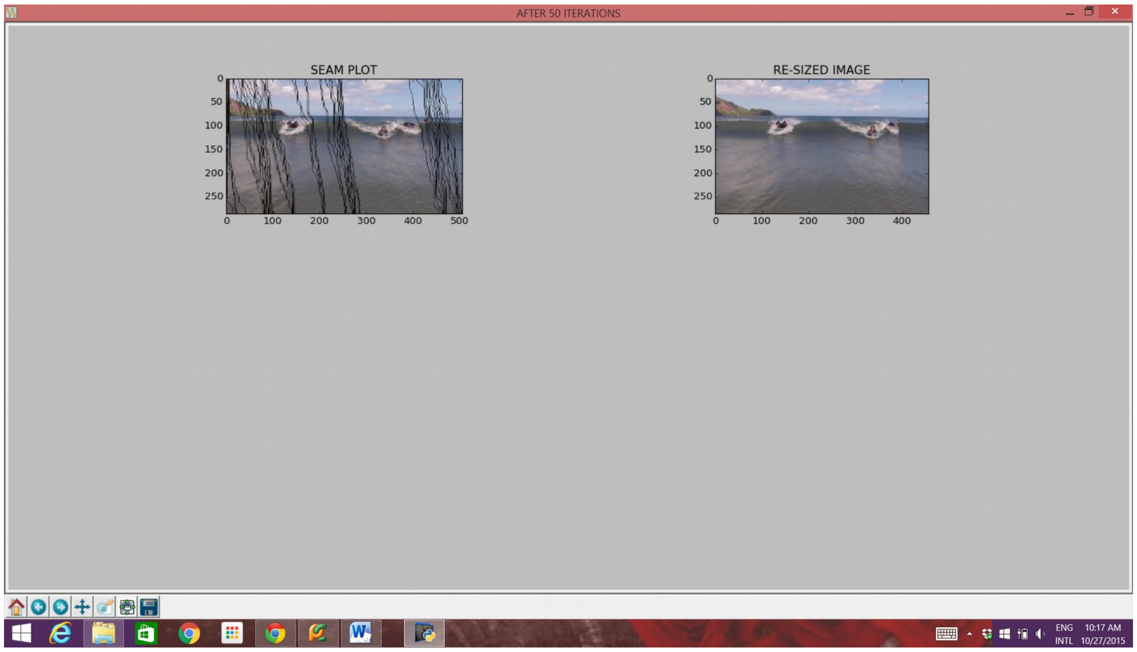
This is dynamic programming. After the entire table has been populated, we traverse back the matrix. In the last row, we find the pixel with least energy and note its index. The procedure is repeated for all rows until we reach the first. This will give you the value of seam.

Complexity: O(n).

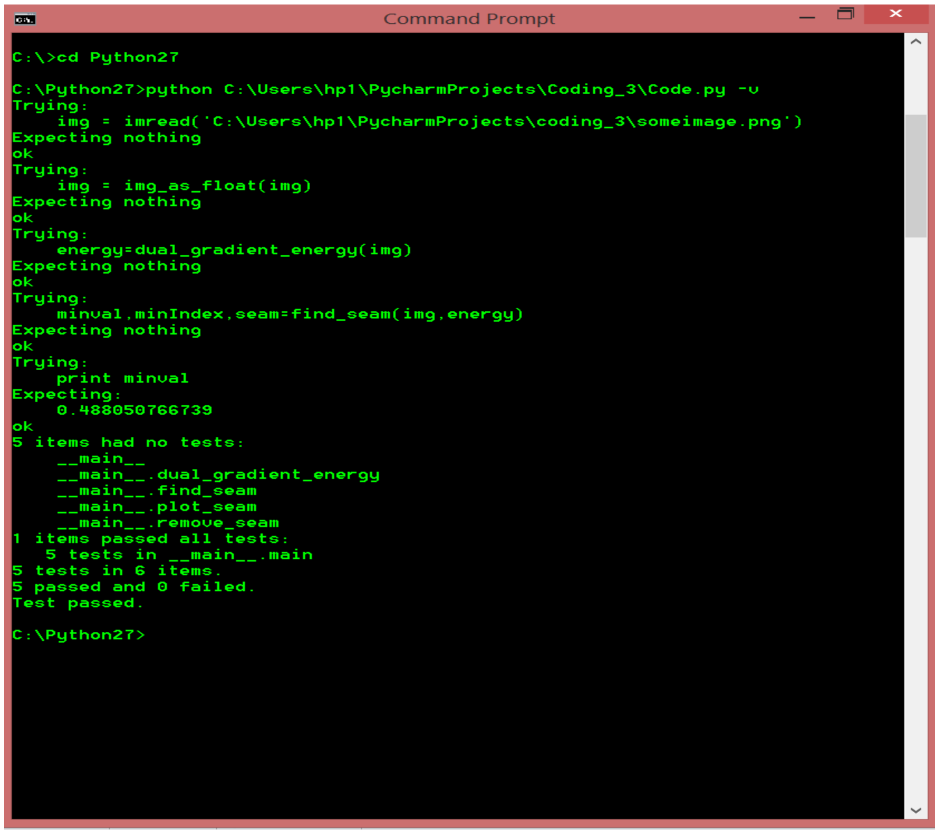
B) Testing Output:
1) Screenshot of output:


BREAKING IMAGE INTO RED, GREEN AND BLUE COMPONENTS
ORIGINAL IMAGE
RED
GREEN
BLUE
ENG INTL 10:11 AM 10/27/2015





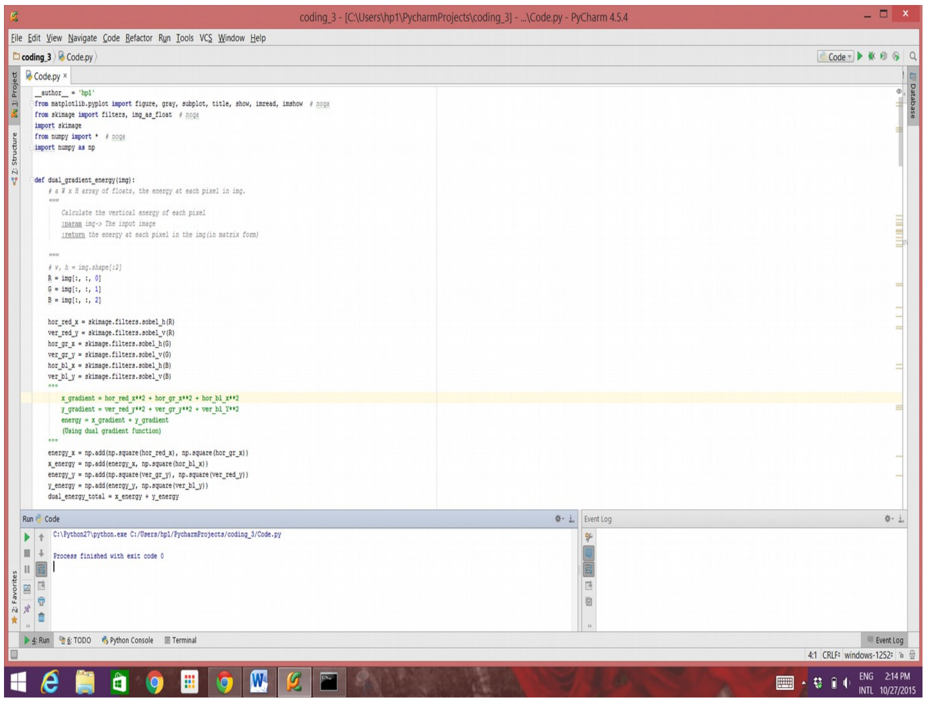
2) **<u>Doc-test</u>**(This clearly indicates my code is running in proper format as specified.)



```
C:\>cd Python27

C:\Python27>python C:\Users\hp1\PycharmProjects\Coding_3\Code.py -v
Trying:
    img = imread('C:\Users\hp1\PycharmProjects\coding_3\someimage.png')
Expecting nothing
ok
Trying:
    img = img_as_float(img)
Expecting nothing
ok
Trying:
    energy=dual_gradient_energy(img)
Expecting nothing
ok
Trying:
    minval,minIndex,seam=find_seam(img,energy)
Expecting nothing
ok
Trying:
    print minval
Expecting:
    0.488050766739
ok
5 items had no tests:
    __main__
    __main__.dual_gradient_energy
    __main__.find_seam
    __main__.plot_seam
    __main__.remove_seam
1 items passed all tests:
   5 tests in __main__.main
5 tests in 6 items.
5 passed and 0 failed.
Test passed.

C:\Python27>
```

C) Static Analysis/Compilation Output

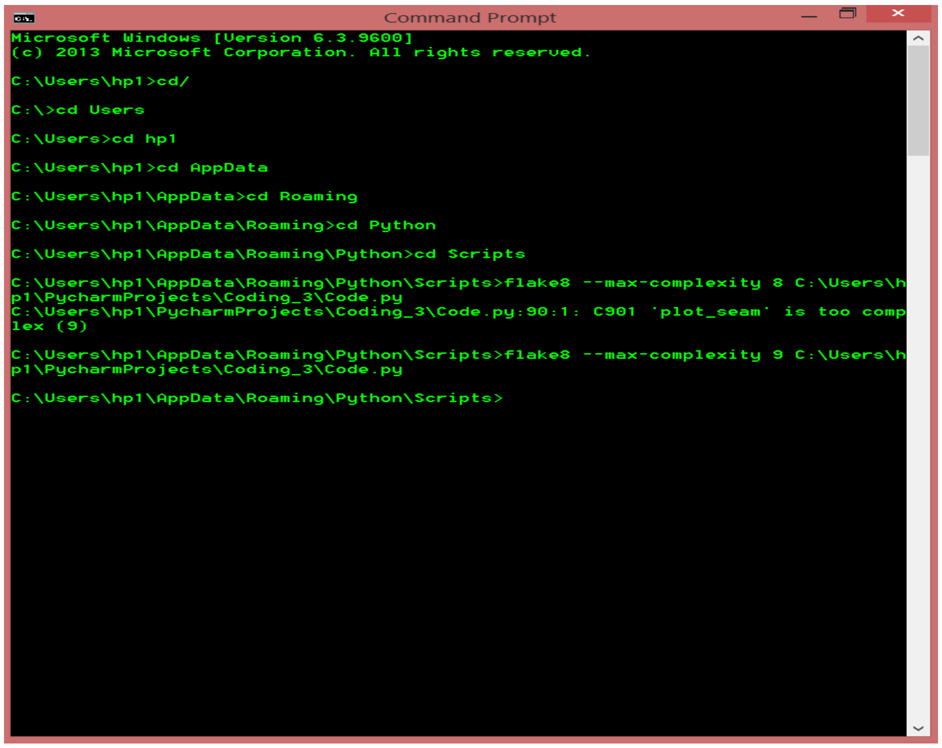
1) When code run is completed



```
__author__ = 'hp1'
from matplotlib.pyplot import figure, gray, subplot, title, show, imread, imshow  # noqa
from skimage import filters, img_as_float  # noqa
import skimage
from numpy import *  # noqa
import numpy as np


def dual_gradient_energy(img):
    # a W x H array of floats, the energy at each pixel in img.
    """
        Calculate the vertical energy of each pixel
        :param img-> The input image
        :return the energy at each pixel in the img(in matrix form)

    """
    # v, h = img.shape[:2]
    R = img[:, :, 0]
    G = img[:, :, 1]
    B = img[:, :, 2]

    hor_red_x = skimage.filters.sobel_h(R)
    ver_red_y = skimage.filters.sobel_v(R)
    hor_gr_x = skimage.filters.sobel_h(G)
    ver_gr_y = skimage.filters.sobel_v(G)
    hor_bl_x = skimage.filters.sobel_h(B)
    ver_bl_y = skimage.filters.sobel_v(B)
    """
        x_gradient = hor_red_x**2 + hor_gr_x**2 + hor_bl_x**2
        y_gradient = ver_red_y**2 + ver_gr_y**2 + ver_bl_Y**2
        energy = x_gradient + y_gradient
        (Using dual gradient function)
    """
    energy_x = np.add(np.square(hor_red_x), np.square(hor_gr_x))
    x_energy = np.add(energy_x, np.square(hor_bl_x))
    energy_y = np.add(np.square(ver_gr_y), np.square(ver_red_y))
    y_energy = np.add(energy_y, np.square(ver_bl_y))
    dual_energy_total = x_energy + y_energy
```

```
C:\Python27\python.exe C:/Users/hp1/PycharmProjects/coding_3/Code.py

Process finished with exit code 0
```

2) Flake8 complexity



```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\hp1>cd/

C:\>cd Users

C:\Users>cd hp1

C:\Users\hp1>cd AppData

C:\Users\hp1\AppData>cd Roaming

C:\Users\hp1\AppData\Roaming>cd Python

C:\Users\hp1\AppData\Roaming\Python>cd Scripts

C:\Users\hp1\AppData\Roaming\Python\Scripts>flake8 --max-complexity 8 C:\Users\h
p1\PycharmProjects\Coding_3\Code.py
C:\Users\hp1\PycharmProjects\Coding_3\Code.py:90:1: C901 'plot_seam' is too comp
lex (9)

C:\Users\hp1\AppData\Roaming\Python\Scripts>flake8 --max-complexity 9 C:\Users\h
p1\PycharmProjects\Coding_3\Code.py

C:\Users\hp1\AppData\Roaming\Python\Scripts>
```

D) Source Code:

```
__author__ = 'hp1'
from matplotlib.pyplot import figure, gray, subplot, title, show, imread, imshow  # noqa
from skimage import filters, img_as_float  # noqa
import skimage
from numpy import *  # noqa
import numpy as np


def dual_gradient_energy(img):
    # a W x H array of floats, the energy at each pixel in img.
    """
       Calculate the energy of each pixel
       :param img-> The input image
       :return the energy at each pixel in the img(in matrix form)

   """
    # w, h = img.shape[:2]
    R = img[:, :, 0]
    G = img[:, :, 1]
    B = img[:, :, 2]

    hor_red_x = skimage.filters.sobel_h(R)
    ver_red_y = skimage.filters.sobel_v(R)
    hor_gr_x = skimage.filters.sobel_h(G)
    ver_gr_y = skimage.filters.sobel_v(G)
    hor_bl_x = skimage.filters.sobel_h(B)
    ver_bl_y = skimage.filters.sobel_v(B)
    """
      x_gradient = hor_red_x**2 + hor_gr_x**2 + hor_bl_x**2
      y_gradient = ver_red_y**2 + ver_gr_y**2 + ver_bl_Y**2
      energy = x_gradient + y_gradient
      (Using dual gradient function)
    """
    energy_x = np.add(np.square(hor_red_x), np.square(hor_gr_x))
    x_energy = np.add(energy_x, np.square(hor_bl_x))
    energy_y = np.add(np.square(ver_gr_y), np.square(ver_red_y))
    y_energy = np.add(energy_y, np.square(ver_bl_y))
    dual_energy_total = x_energy + y_energy

    return dual_energy_total
    pass

```

```

def find_seam(img, energy_total):
    # an array of H integers, for each row returns the column of the seam.
    """
      To find the minimum intensity seam
       which is needed to be removed
      :param img-> The input image
      :param energy_total-> Represents the energy calculated
                           from the dual_gradient_energy()
      :return val-> minimum value of energy in last row
      :return: minIndex-> minimum index(column) in last row
                           which contains the minimum value of energy
      :return: the seam to remove
   """
    h = img.shape[0]
    w = img.shape[1]
    seam = np.zeros(shape=(h, w), dtype=float)  # initializing seam to zeroes.
    np.copyto(seam, energy_total)
    for i in range(1, h):
        """
          Populating the table using dynamic programming.
          Each time for a block [i][j] we consider
```

```
                    the three blocks([i-1,j-1],[i-1,j]and [i,j+1].
        """
        for j in range(1, w-1):
            if j == 1:
                seam[i, j] = \
                    seam[i, j] + min(seam[i - 1, j], seam[i - 1, j + 1])
            if j == w-2:
                seam[i, j] = \
                    seam[i, j] + min(seam[i - 1, j - 1], seam[i - 1, j])
            else:
                seam[i, j] = \
                    seam[i, j] + min(seam[i-1, j-1],
                                     seam[i-1, j], seam[i-1, j+1])

    val = inf
    index = -1
    for i in range(1, w-1):
        # computing minimum value of energy and its index(column) in last row
        if seam[h-1, i] < val:
            val = seam[h-1, i]
            index = i
    return val, index, seam
    pass


def plot_seam(img, index, seam):
    # your own visualization of the seam, img, and energy func.
    """
        Plot the seam to be removed
        :param img -> The input image to be read
        :param index -> The index(column) of minimum energy value in last row
        :param seam -> The minimum energy array
                        computed using Dynamic Programming in find_seam()
        :return img -> The image with the seam plotted
        :return path -> The array containing the path of traversal
                        in the energy array using backtracking in DP.
    """

    h = img.shape[0]
    w = img.shape[1]
    path = np.zeros(shape=(h), dtype=int)
    for i in range(h-1, -1, -1):
        # backtracking the matrix; traverse the matrix bottom to top
        img[i, index] = [255, 0, 0]
```

```python
        path[i] = index
        if i != 0:
            # row cannot be 0 (first row),
            # as it does not have values to compare with.
            # boundary conditions
            if index == 1:
                # The column comparison starts with second column,
                # since first column has all zeros, so
                # comparing it to first column will cause everything to be 0.
                if seam[i-1, index + 1] < seam[i - 1, index]:
                    index += 1
            elif index == w - 2:
                # col is second last, not comparing
                # with last col since it will make all 0s
                if seam[i - 1, index - 1] < seam[i-1, index]:
                    index -= 1
            else:
                if seam[i - 1, index - 1] < seam[i - 1, index]\
                        and seam[i - 1, index - 1] < seam[i - 1, index + 1]:
                    index -= 1
                elif seam[i-1, index+1] < seam[i-1, index] \
                        and seam[i-1, index+1] < seam[i-1, index+1]:
```

```
                index += 1
    # print path
    return img, path
    pass


def remove_seam(img, path):
    # modify img in-place and returns a W-1 x H x 3 slice.
    """
        Remove seam from picture and reduce width of image by 1 in each call
        :param img -> The image which has the seam plotted from find_seam()
        :param path -> The array containing the path of
                        traversal in the energy array using backtracking in DP.
        :return resize_img -> The re-sized image with the seam removed.

    """

    h = img.shape[0]
    w = img.shape[1]
    resize_img = np.zeros(shape=(h, w-1, 3))
    # (w-1) to remove one seam only, 3 refers to rgb component
    for i in range(h-1, -1, -1):
        b = img[i, :, :]
        resize_img[i, :, :] = np.delete(b, path[i], axis=0)
        # delete position mentioned by path[i] array so as to
        # remove seam

    return resize_img
    pass


def main():
    """
   >>> img = imread('C:\Users\hp1\PycharmProjects\coding_3\someimage.png')
   >>> img = img_as_float(img)
   >>> energy=dual_gradient_energy(img)
   >>> minval,minIndex,seam=find_seam(img,energy)
   >>> print minval
   0.488050766739
   """
    img = imread('C:\Users\hp1\PycharmProjects\coding_3\someimage.png')
    img = img_as_float(img)
    R = img[:, :, 0]
    G = img[:, :, 1]
```

```
B = img[:, :, 2]
figure("BREAKING IMAGE INTO RED, GREEN AND BLUE COMPONENTS")
gray()

subplot(1, 4, 1)
imshow(img)
title("ORIGINAL IMAGE")
subplot(1, 4, 2)
imshow(R)
title("RED")
subplot(1, 4, 3)
imshow(G)
title("GREEN")
subplot(1, 4, 4)
imshow(B)
title("BLUE")
show()
energy = dual_gradient_energy(img)
figure("ENERGY OUTPUT")
subplot(2, 1, 1)
imshow(energy)
title("DUAL ENERGY GRADIENT")
```

```python
show()

for i in range(50):
    energy = dual_gradient_energy(img)
    val, index, seam = find_seam(img, energy)
    img, path = plot_seam(img, index, seam)
figure("AFTER 50 ITERATIONS")
subplot(3, 2, 1)
imshow(img)
title("SEAM PLOT")

img = imread('C:\Users\hp1\PycharmProjects\coding_3\someimage.png')
img = img_as_float(img)
for i in range(50):
    energy = dual_gradient_energy(img)
    val, index, seam = find_seam(img, energy)
    img, path1 = plot_seam(img, index, seam)
    img = remove_seam(img, path1)
subplot(3, 2, 2)
imshow(img)
title("RE-SIZED IMAGE")
show()
```

```python


if __name__ == '__main__':
    main()
    import doctest
    doctest.testmod()
```