Coding Assignment 3

Posting ID: 5582-527

1. Reflection

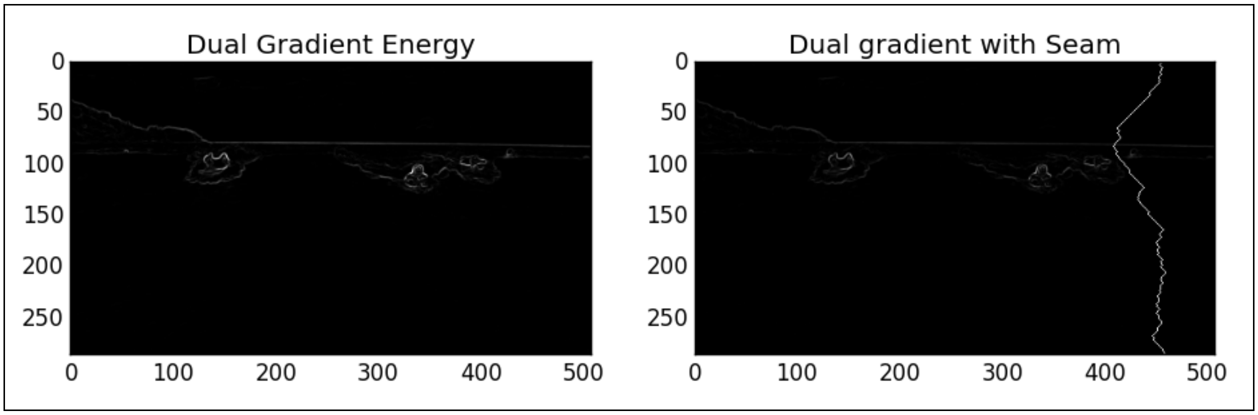
The objective of the assignment is to find least energy path of pixels (using dual gradient energy) in a given image so that the image can be reduced in width or height without losing significant information from the image. This could be achieved by using either a greedy approach or dynamic programming.
The approach taken in this submission is dynamic programming. The 4 functions implemented to achieve Seam Carving are as follows:

- The “dual_gradient_energy” function calculates and returns a 2-D array containing the energy at each pixel.
- The “find_seam” function adopts a bottom-up approach and finds the least energy pixel path at the last row. It then returns the column index of each row that lies on the identified path.
- The “plot_seam” function displays the image given as input, the energy function of the image and the seam identified by “find_seam” function.
- The “remove_seam” function returns the image by removing the pixel path in-place that was identified by “find_seam” function.

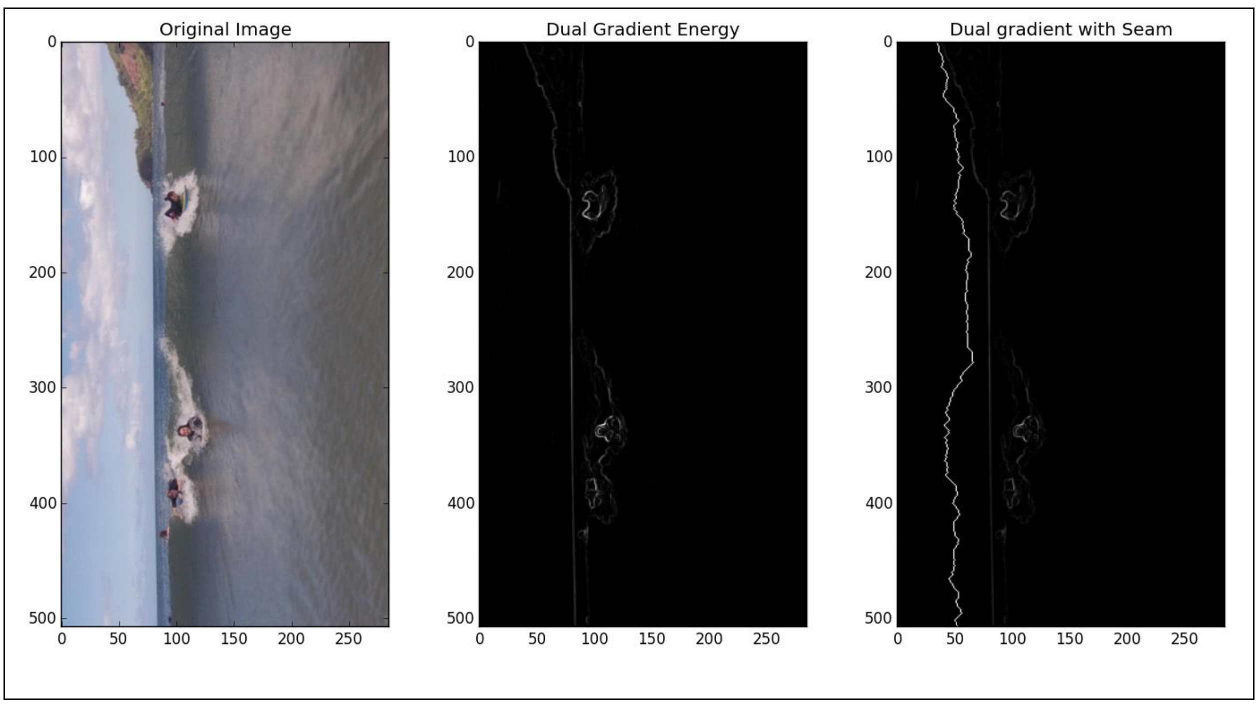
2. Testing Output

Below is the original image used as input.

The below screenshot shows the dual gradient energy of the input image (left) and the seam identified by the “find_seam” function (right). This seam is obtained as we start evaluating the least energy path from (0, 0) to (m, n) of the 2-D energy array.



The screenshot below shows the dual gradient energy of the transpose of the input image and the seam identified by the “find_seam” function:



The seams highlighted in the above 2 screenshots indicate the least energy paths with total seam energy equal to

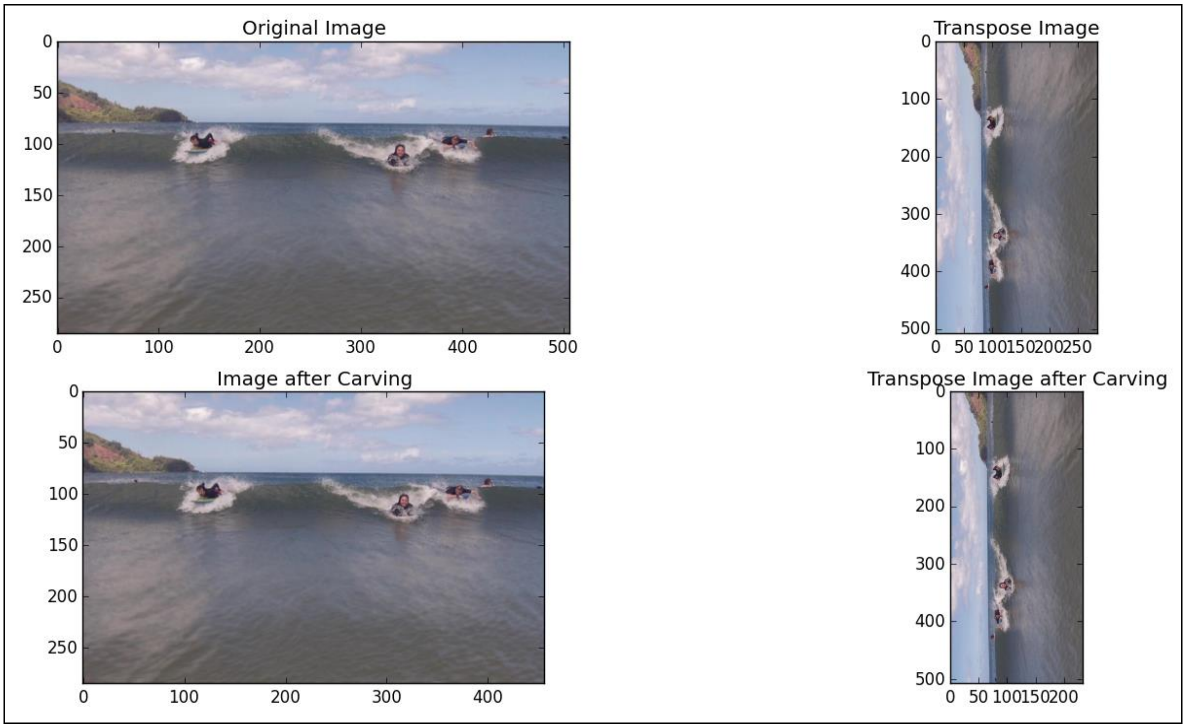
0.488050766739 → for original image

0.038767777001 → for Transpose image

The total seam energy of the original image is shown in the doctest as well:

```
    print x[1]
Expecting:
    0.488050766739
ok
```

After removing the least energy seam for 50 times, we obtain the images shown in the below screenshot:



As we can see, the significant information in the image is not lost even after 50 iterations. Hence, this implementation is suitable for seam carving of any image.

3. Static Analysis / Compilation Output

Doctest output:

```
6 items had no tests:
    seamcarver
    seamcarver.dual_gradient_energy
    seamcarver.main
    seamcarver.plot_seam
    seamcarver.remove_seam
    seamcarver.seam_path_tracking
2 items passed all tests:
   3 tests in seamcarver.find_seam
   9 tests in seamcarver.seam_cost
12 tests in 8 items.
12 passed and 0 failed.
Test passed.
```

Flake8 output: Complexity of the implementation is below 9.

```
C:\Users\MadhulikaBushi\Desktop\501\Coding Assignments\Coding Assignment 3>flake8 --max-complexity 9 seamcarver.py

C:\Users\MadhulikaBushi\Desktop\501\Coding Assignments\Coding Assignment 3>
```

4. Source Code

```python
"""
This file provides implementation of Seam Carving.
"""
import pylab
from skimage import filters
from skimage import img_as_float
import numpy


def dual_gradient_energy(img):
    """
    Dual gradient energy is the sum of the square of a horizontal gradient and a vertical gradient.
    Use skimage.filter.hsobel and vsobel to calculate the gradients of each channel independently.
    The energy is the sum of the square the horizontal and vertical gradients over all channels.
    :param img: input image
    :return: dual gradient energy of input image
    """
    red_channel = img[:, :, 0]      # red channel of the image
    green_channel = img[:, :, 1]    # green channel of the image
    blue_channel = img[:, :, 2]     # blue channel of the image

    horizontal_gradient_red = filters.sobel_h(red_channel)     # horizontal gradient of the red channel
    vertical_gradient_red = filters.sobel_v(red_channel)       # vertical gradient of the red channel

    horizontal_gradient_green = filters.sobel_h(green_channel)  # horizontal gradient of the green channel
    vertical_gradient_green = filters.sobel_v(green_channel)   # vertical gradient of the green channel

    horizontal_gradient_blue = filters.sobel_h(blue_channel)  # horizontal gradient of the blue channel
    vertical_gradient_blue = filters.sobel_v(blue_channel)    # vertical gradient of the blue channel

```

```
    # dual gradient energy at each pixel
    energy = (horizontal_gradient_red * horizontal_gradient_red)\
        + (vertical_gradient_red * vertical_gradient_red)\
        + (horizontal_gradient_green * horizontal_gradient_green)\
        + (vertical_gradient_green * vertical_gradient_green)\
        + (horizontal_gradient_blue * horizontal_gradient_blue)\
        + (vertical_gradient_blue * vertical_gradient_blue)

    return energy


def find_seam(img):
    """
   An array of H (number of rows in the image) integers, for each row return the column of the seam.
   :param img: input image
   :return: least energy seam to be removed
   >>> img = pylab.imread('someimage.png')
   >>> img = img_as_float(img)
   >>> print find_seam(img)
   [ 456.  453.  454.  453.  452.  453.  454.  455.  454.  454.  453.  453.
     453.  454.  453.  452.  451.  451.  452.  452.  451.  452.  451.  450.
     449.  448.  447.  446.  445.  444.  443.  442.  442.  443.  442.  441.
     440.  439.  438.  437.  436.  435.  434.  433.  432.  431.  430.  429.
     428.  427.  426.  425.  424.  423.  422.  421.  420.  419.  418.  417.
     416.  415.  414.  413.  412.  411.  411.  412.  413.  412.  411.  412.
     412.  413.  414.  414.  413.  412.  411.  410.  409.  408.  408.  407.
     408.  408.  409.  410.  411.  410.  410.  410.  411.  412.  412.  413.
     414.  415.  416.  417.  418.  418.  419.  420.  421.  422.  423.  423.
     423.  423.  424.  425.  426.  427.  428.  429.  430.  431.  432.  433.
     434.  435.  436.  437.  436.  435.  434.  433.  432.  433.  432.  432.
     431.  431.  432.  431.  432.  433.  433.  434.  435.  436.  437.  438.
     439.  440.  441.  442.  441.  441.  442.  443.  444.  445.  446.  447.
     448.  449.  450.  451.  452.  453.  454.  455.  456.  455.  454.  453.
     452.  452.  453.  454.  453.  452.  451.  450.  449.  449.  450.  450.
     451.  452.  451.  450.  450.  450.  451.  451.  450.  451.  452.  453.
     454.  455.  454.  453.  454.  454.  455.  454.  453.  454.  455.  456.
     457.  458.  458.  457.  456.  455.  454.  454.  453.  454.  454.  455.
     455.  454.  453.  453.  453.  453.  453.  452.  453.  453.  452.  451.
     450.  450.  451.  451.  451.  451.  451.  450.  449.  450.  449.  450.
     451.  452.  451.  450.  450.  449.  450.  451.  451.  451.  451.  452.
```

```
     453.  454.  454.  454.  453.  452.  451.  450.  450.  450.  450.  449.
     448.  447.  446.  445.  446.  447.  446.  446.  447.  448.  449.  450.
     451.  452.  453.  454.  455.  455.  456.  456.  457.]
    """
    h, w = img.shape[:2]                             # h - rows, w - columns
    dg_energy = dual_gradient_energy(img)            # get the dual gradient energy of the image
    seam_calc_energy = numpy.zeros(shape=(h, w))     # holding sum of the energies till that row for each pixel
    seam = numpy.zeros(shape=h)                      # actual seam that can be removed
    numpy.copyto(seam_calc_energy, dg_energy)        # initializing with dual gradient energy as default

    seam_path, seam_calc_energy = seam_path_tracking(h, w, seam_calc_energy)
    index_min_energy = seam_cost(h, w, seam_calc_energy)

    index = index_min_energy[0]
    seam[0] = index

    for i in range(h - 1, 0, -1):                    # gathering the least energy pixel path
        index = seam_path[i][index]
        seam[i] = index

    return seam


def seam_path_tracking(h, w, seam_calc_energy):
    """
    Getting the path chosen by each pixel from the first row
    :param h: Height (number of rows)
    :param w: Width (number of columns)
    :param seam_calc_energy: sum of the energies till the selected row for each pixel
        :return: seam_path for each pixel
        """
        seam_path = numpy.zeros(shape=(h, w))            # tracking the choice
        for j in range(0, w):                            # initializing the seam path
            seam_path[0][j] = j

        for i in range(1, h):                            # computing the least energy path
            for j in range(1, w - 1):
                center_seam = seam_calc_energy[i - 1][j]
                if j == 1:                                   # boundary case
```

```
                left_seam = float('inf')
                right_seam = seam_calc_energy[i - 1][j + 1]
            elif j == (w - 2):                            # boundary case
                left_seam = seam_calc_energy[i - 1][j - 1]
                right_seam = float('inf')
            else:                                         # all other cases
                left_seam = seam_calc_energy[i - 1][j - 1]
                right_seam = seam_calc_energy[i - 1][j + 1]

            # tracking the pixel position to identify the choice from the previous row
            if left_seam <= right_seam and left_seam <= center_seam:
                seam_calc_energy[i][j] = seam_calc_energy[i][j] + left_seam
                seam_path[i][j] = j - 1
            elif center_seam <= left_seam and center_seam <= right_seam:
                seam_calc_energy[i][j] = seam_calc_energy[i][j] + center_seam
                seam_path[i][j] = j
            elif right_seam <= center_seam and right_seam <= left_seam:
                seam_calc_energy[i][j] = seam_calc_energy[i][j] + right_seam
                seam_path[i][j] = j + 1
    return seam_path, seam_calc_energy


def seam_cost(h, w, seam_calc_energy):
    """
    Getting the index at the top row for which the energy path is least
    :param h: height of image
    :param w: width of image
    :param seam_calc_energy: sum of the energies till the selected row for each pixel
    :return: index at the top row of the least energy path
    >>> img = pylab.imread('someimage.png')
    >>> img = img_as_float(img)
    >>> h, w = img.shape[:2]
    >>> seam_calc_energy = numpy.zeros(shape=(h, w))
    >>> dg_energy = dual_gradient_energy(img)
    >>> numpy.copyto(seam_calc_energy, dg_energy)
    >>> seam_path, seam_calc_energy = seam_path_tracking(h, w, seam_calc_energy)
    >>> x = seam_cost(h, w, seam_calc_energy)
    >>> print x[1]
    0.488050766739
    """
    minimum_energy = float('inf')
```

```python
    for i in range(1, w - 1):                       # checking the last row to identify least energy pixel path
        if seam_calc_energy[h - 1][i] < minimum_energy:
            minimum_energy = seam_calc_energy[h - 1][i]
            index = i
    return index, minimum_energy


def plot_seam(img, seam):
    """
   Visualization of the seam, img, and energy func.
   :param img: input image
   :param seam: seam identified for the image
   :return: NA
   """
    h, w = img.shape[:2]                        # h - rows, w - columns
    dg_energy1 = dual_gradient_energy(img)      # get the dual gradient energy of the image
    dg_energy2 = numpy.zeros(shape=(h, w))
    numpy.copyto(dg_energy2, dg_energy1)
    pylab.figure()
    pylab.gray()
    pylab.subplot(1, 3, 1)
    pylab.imshow(img)                           # plot original image
    pylab.title("Original Image")
    pylab.subplot(1, 3, 2)
    pylab.imshow(dg_energy1)
    pylab.title("Dual Gradient Energy")         # plot dual gradient energy

    for i in range(0, h):                       # highlighting the seam
        dg_energy2[i][seam[i]] = 2
    pylab.subplot(1, 3, 3)
    pylab.imshow(dg_energy2)      # plot dual gradient energy with the identified seam
    pylab.title("Dual gradient with Seam")

    pylab.show()


def remove_seam(img, seam):
    """
   Modify img in-place and return a W-1 x H x 3 slice
```

```
    :param img: input image
    :param seam: seam identified for the image
    :return: image after removing the seam
    """
    h, w = img.shape[:2]                    # h - rows, w - columns

    for i in range(0, h):                   # moving all the columns to the right
        width_position = seam[i]
        img[i, 1:width_position + 1, :] = img[i, 0:width_position, :]

    return numpy.delete(img, 0, 1)          # deleting the empty column after shifting


def main():
    img = pylab.imread('someimage.png')          # getting the image
    transpose_img = img.transpose(1, 0, 2)       # getting the transpose image
    img = img_as_float(img)
    transpose_img = img_as_float(transpose_img)

    seam = find_seam(img)                        # find seam
    transpose_seam = find_seam(transpose_img)    # find transpose seam

    plot_seam(img, seam)                         # plot seam
    plot_seam(transpose_img, transpose_seam)     # plot transpose seam

    pylab.figure()

    pylab.subplot(2, 2, 1)
    pylab.imshow(img)                       # plot original image
    pylab.title("Original Image")

    h, w = img.shape[:2]
    print 'original image dimensions: W = ' + str(w) + ' H = ' + str(h)

    pylab.subplot(2, 2, 2)
    pylab.imshow(transpose_img)             # plot transpose of original image
    pylab.title("Transpose Image")

    h, w = transpose_img.shape[:2]
    print 'Transpose image dimensions: W = ' + str(w) + ' H = ' + str(h)

```

```python
    removed_img = remove_seam(img, seam)
    for i in range(0, 49):                 # image after removing 50 seams
        seam = find_seam(removed_img)
        removed_img = remove_seam(removed_img, seam)
    pylab.subplot(2, 2, 3)
    pylab.imshow(removed_img)              # plot original image after carving 50 times
    pylab.title("Image after Carving")

    h, w = removed_img.shape[:2]
    print 'After carving image dimensions: W = ' + str(w) + ' H = ' + str(h)

    removed_transpose_img = remove_seam(transpose_img, transpose_seam)
    for i in range(0, 49):                 # transpose image after removing 50 seams
        transpose_seam = find_seam(removed_transpose_img)
        removed_transpose_img = remove_seam(removed_transpose_img, transpose_seam)
    pylab.subplot(2, 2, 4)
    pylab.imshow(removed_transpose_img)    # plot original image after carving 50 times
    pylab.title("Transpose Image after Carving")

    h, w = removed_transpose_img.shape[:2]
    print 'After carving transpose image dimensions: W = ' + str(w) + ' H = ' + str(h)

    pylab.show()


if __name__ == '__main__':
    main()
```

Revised rubric for coding assignments.

This is a 5-point rubric for coding projects. Graders should refrain from using fractional points (they are a pain to defend), choose the one one number that best reflects the assignment.   For assignments with multiple parts, choose the lowest scoring part.

This rubric is based on the idea that students submit PDF write-ups with their coding assignment. Write-ups *must* be PDF's with the source code so that graders can quickly view them annotate them using blackboard. The rubric does not address specific learning objectives — the assumption is that by completing the assignment the student has implicitly demonstrated some set objectives in addition to coding.

0 points — Student does not submit **all** parts of the assignment, meaning *both* a **PDF** writeup (all sections) that includes source code and output of testing, as well as a .**zip** file with source code.

2 points — The code does not run or does not *appear* to be able to run. The code it much longer than it should be, or does not appear to follow the scaffolding provided. The grader can but **does not have to verify that it does not run**, it is the student's responsibility to provide a writeup that is sufficiently convincing. Student may not appeal by coming after the fact and showing that code runs on their machine.

When grading, the grader should indicate portions of the code by annotating the writeup that are suspicious.

3 points — The code runs or looks like it would run, but the student has not shown via their writeup that it produces the correct result on reasonable inputs.   Or, the student has implemented algorithms using approaches other than the ones indicated in the assignment, or the implementation has the wrong asymptotic complexity or that demonstrates a lack of understanding of the assignments objectives. The grader can, but **does not have to run the code to verify correctness** — it is the student's responsibility to make a convincing case that the output and the algorithm is correct.

When grading, the grader should indicate by annotating the write-up where results

4 points — The code runs or appears to run correctly, but has readability or style issues. The student has not demonstrated that their code has passed style guidelines, or the student's implementation appears to be unnecessarily complex (even though it looks like it works).

When grading, the grader should indicate the style problems.

5 points — No issues that we can spot.