1) How to execute the code?

python seamer.py <no_of_times_the_verticle_seam_should_be_removed>

Note: the image should be named ‘sample.png’ and should be present in the same location as the python code.

2) Reflection

Seam Carving is a context aware algorithm which removes less important pixels in the image thereby reducing the size of the image without removing the important features of the image.

The algorithm is as follow, We calculate the energy gradient of each pixel using the formula

Energy of pixel $(x, y) = \Delta x2(x, y) + \Delta y2(x, y)$, where,

$\Delta x2(x, y) = R_x(x, y)2 + G_x(x, y)2 + B_x(x, y)2$ and
$\Delta y2(x, y) = R_y(x, y)2 + G_y(x, y)2 + B_y(x, y)2$

Once the energy is calculated, we calculate the seam matrix which is used to find out the vertical seam of least importance. The seam matrix is calculated as follows,

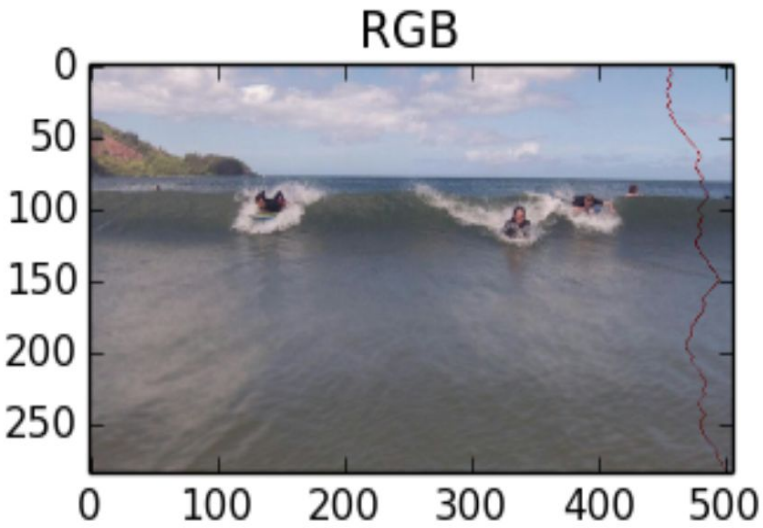
seam[i][j] = min (seam[i-1][j-1] , seam[i-1][j], seam[i-1][j+1]) + energy_matrix(i,j)

Once seam matrix is calculated we proceed to the last row and find the minimum element in that row and backtrack to find a minimum path from last row to first row. This gives us the vertical seam.

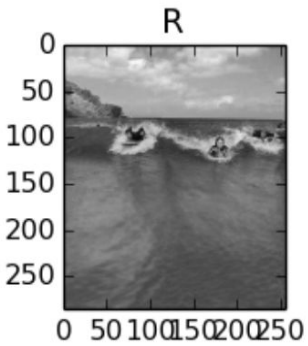
We can delete this column. We can do this iteratively till the size of the image is reduced significantly.

3) Testing Output

Below is the screenshot after running the code for 1 time. It shows the original and the output image.



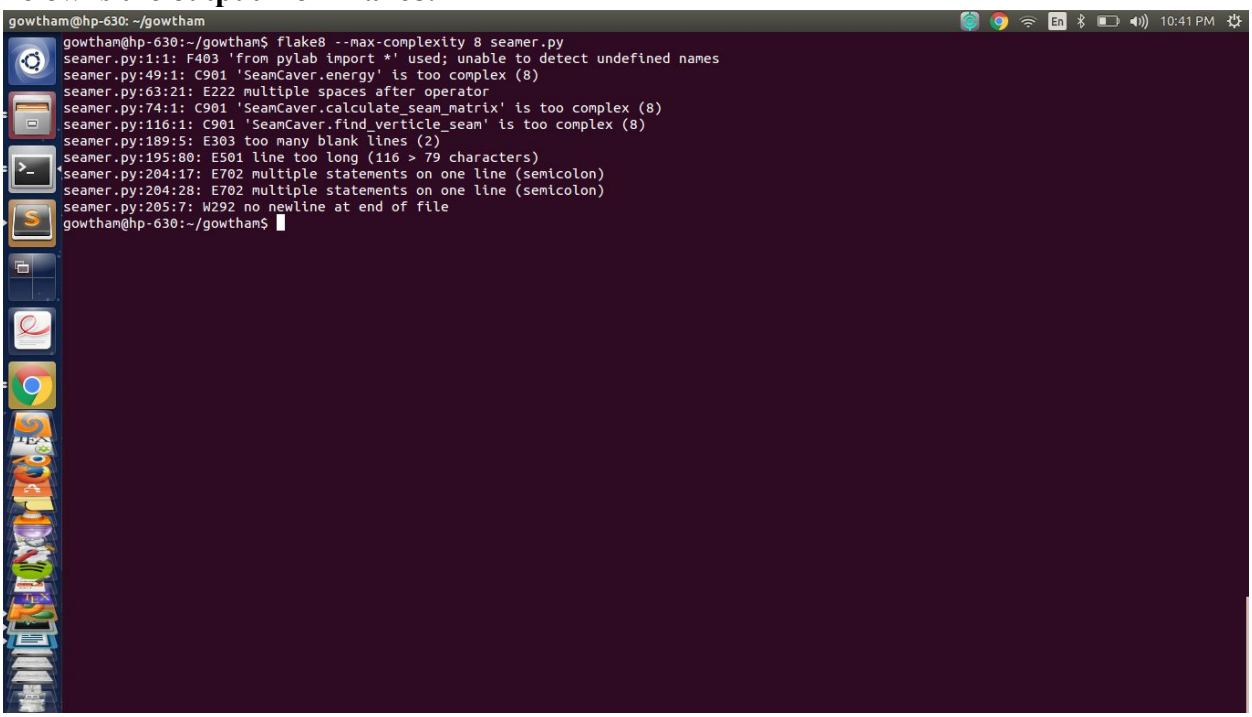
After running the code for 250 time, I.e, 50% reduction of image size we get,



As can be seen from the image x-axis width of the image has reduced from 500 to 250

4) Static Analysis / Compilation Output

Below is the output from flake8.



5) Source Code

```
from pylab import *
from skimage import img_as_float
import sys

loop_number = sys.argv[1]
INFINITY = float('inf')
img = imread('sample.png')
img = img_as_float(img)
w, h = img.shape[:2]
TR = img[:, :, 0]
TG = img[:, :, 1]
TB = img[:, :, 2]

R = TR.tolist()
G = TG.tolist()
B = TB.tolist()


class SeamCaver:
    def __init__(self):
        self.height = len(R)
        self.width = len(R[1])
        self.seam = []
        self.energy_matrix = []
        self.path = []

    def calculate_energy(self, x, y, fx, bx, fy, by):
        Rx = R[x][fy] - R[x][by]
        Gx = G[x][fy] - G[x][by]
        Bx = B[x][fy] - B[x][by]

        Ry = R[fx][y] - R[bx][y]
        Gy = G[fx][y] - G[bx][y]
        By = B[fx][y] - B[bx][y]

        Rx2 = math.pow(Rx, 2)
        Gx2 = math.pow(Gx, 2)
        Bx2 = math.pow(Bx, 2)

        Ry2 = math.pow(Ry, 2)
        Gy2 = math.pow(Gy, 2)
        By2 = math.pow(By, 2)

        delta_x_square = Rx2 + Gx2 + Bx2
        delta_y_square = Ry2 + Gy2 + By2

        pixal_energy = delta_x_square + delta_y_square

        return pixal_energy

    def energy(self, x, y):
        if x == (self.height - 1) or x == 0:
            if x == (self.height - 1):
                fx = 0
                bx = x - 1
            elif x == 0:
                fx = x + 1
                bx = self.height - 1
        else:
            fx = x + 1
            bx = x - 1
```

```
    if y == (self.width - 1) or y == 0:
        if y == 0:
            fy =  y + 1
            by = self.width - 1
        elif y == self.width - 1:
            fy = 0
            by = y - 1
    else:
        fy = y + 1
        by = y - 1

    return self.calculate_energy(x, y, fx, bx, fy, by)

def calculate_seam_matrix(self):
    self.seam = []
    temp_array = []
    for i in range(0, self.height):
            self.seam.append([0 for col in range(self.width)])

    for j in range(0, self.width):
        temp_array.append(self.energy_matrix[0][j])

    self.seam[0] = temp_array

    for i in range(1, self.height):
        for j in range(0, self.width):
            seam_i_minus_1 = INFINITY
            seam_i_plus_1 = INFINITY
            if j > 0:
                seam_i_minus_1 = self.seam[i-1][j-1]
            else:
                seam_i_minus_1 = INFINITY
            seam_i = self.seam[i-1][j]

            if j < self.width-1:
                seam_i_plus_1 = self.seam[i-1][j+1]
            else:
                seam_i_plus_1 = INFINITY
            minimum = min(seam_i_minus_1, seam_i, seam_i_plus_1)
            self.seam[i][j] = minimum + self.energy_matrix[i][j]


def displaySeamMatrix(self):
    for i in range(self.height):
        print self.seam[i]

def find_min_j_the_row(self):
    last_row = self.height-1
    min_val = self.seam[last_row][0]
    min_j = 0
    for j in range(1, self.width):
        if min_val > self.seam[last_row][j]:
            min_val = self.seam[last_row][j]
            min_j = j
    return min_j

def find_verticle_seam(self, i, j):
    self.path.append((i, j))
    if i == 0:
        return

    if j > 0:
        x = self.seam[i-1][j-1]
    else:
        x = INFINITY

    y = self.seam[i-1][j]
```

```
    if j < self.width-1:
        z = self.seam[i-1][j+1]
    else:
        z = INFINITY

    minimum = self.min_index(x, y, z)

    if minimum == 0:
        col = j-1
    elif minimum == 1:
        col = j
    elif minimum == 2:
        col = j + 1

    self.find_verticle_seam(i-1, col)

def min_index(self, x, y, z):
    min_indexer = min(x, y, z)
    index = 0
    if min_indexer == x:
        index = 0
    elif min_indexer == y:
        index = 1
    elif min_indexer == z:
        index = 2
    return index

def print_min_path(self):
    for i in range(self.height):
        print self.path[i]

def calculate_energy_matrix(self):
    self.energy_matrix = []
    for i in range(0, self.height):
        temp_matrix = []
        for j in range(0, self.width):
            temp_matrix.append(A.energy(i, j))
        self.energy_matrix.append(temp_matrix)

def printEnergyMatrix(self):
    for i in range(0, self.height):
        print self.energy_matrix[i]

def findAndRemoveVerticleSeam(self):
    self.calculate_energy_matrix()
    self.calculate_seam_matrix()
    min_j = self.find_min_j_the_row()
    self.path = []
    self.find_verticle_seam(self.height-1, min_j)
    self.removeVerticleSeam()

def removeVerticleSeam(self):
    for i in range(self.height):
        (row, cloumn) = self.path[i]
        R[row].pop(cloumn)
        G[row].pop(cloumn)
        B[row].pop(cloumn)
    self.width = self.width - 1


def calculateHeightAndWidth(self):
    self.height = len(R)
    self.width = len(R[1])
    print self.height
    print self.width
```

```
#-------------------------------------------------------------------------------------------
-------------------------

A = SeamCaver()

for i in range(0, loop_number):
    A.findAndRemoveVerticleSeam()

figure()
gray()
subplot(1, 4, 1); imshow(R); title("R")
show()
```