Nice work! Tested on my machine and ran fine.
Good job with the writeup too.

Grade: A

Trinh Nguyen, Jordan Ando
E27

Final Project Report

I. Introduction:

For the final project, we chose to implement seam carving (Avidar and Shamir, 2009). Seam carving is an algorithm that searches for horizontal/ vertical lines of pixels that have the lowest energy. This is a method of content-aware resizing, and it's more useful than traditional cropping in cases where we want to preserve the more important parts of the image.

We have completed the main goal we set out in our proposal: We implemented the algorithm from scratch using cv2 and numpy libraries. We didn't implement seam carving an image into cv2 display window size. Instead, users can input their desired width and height, and the program will remove all the seams necessary. The former option is interesting, but would be incredibly slow to appear.

Both Trinh and Jordan took turns researching and writing the code, meeting up several times during reading week/exam period in order to accomplish this..

II. The Algorithm:

The basic algorithm for seam carving is as follows.

1. Find energy map:

   First, we need to find the energy of each pixel in the image by calculating their gradient in the x and y direction. Then we add the absolute value of the gradient of the former to that of the latter. We then sum all the energies of each color channel in a pixel, creating a 2D energy map. We calculate this using cv2.sobel(). There are other energy maps, although this one is successful--each accomplishes a similar task.

2. Find the accumulated cost matrix (forward energy):

   We create an accumulated cost matrix (M) that has the same shape as the energy map. We assume that we are looking for vertical seams in the default case. If we want to look for horizontal seams, the matrix is constructed from left to right rather than vertically, but the steps below are otherwise the same.

The very top row of M is equal to its value in the energy map. Starting from the second row, we iterate through each row and calculate each pixel's value according to this rule: a pixel value is equal to the sum of its value in the energy map and the minimum of it's three top neighbors (top-left, top-center, top-right). A pixel on edges will be calculated according to whatever top neighbors it has: for example, a pixel on the left edge doesn't have a top-left neighbor, so its calculation only involves the top-right and top-center neighbor.

3. Finding a seam:

   We find the minimum seam by finding the minimum pixel on the bottom row, then backtracking from that pixel. We will find the coordinates of the minimum top neighbor pixel, save that coordinates, and keep going until we are done with the top row. This is essentially, with the previous step, dynamic programming, similar to Dijkstra's algorithm which may be familiar.

4. Removing a seam:

   To remove a vertical seam, we go through all its coordinates and remove them. Then we stitch the rest of the image together. For example, if a coordinate in a seam is (i,j), we stitch image[i, 0:j, :] with image[i, j+1:, :], leaving out point (i,j). We can do something similar to remove a horizontal seam by stitching image[0:i, j, :] with image[i+1:, j, :] instead.

5. Main function:

   In the main function, we take out seams in alternating order (remove the lowest horizontal seam, then vertical) until the image gets to the user's desired size. Every time we want to take out a seam, we will have to recalculate M, find the minimum seam, remove it and decrement the number of seams we need to take out. This is a slow process, but fairly effective.

This is implemented as a python command line script, with input format "python seamcarving.py IMAGE NEWHEIGHT NEWWIDTH" with the latter two in pixels.

III. Example outputs:

We ran seam carving on some landscape photos we found, attached below. (https://treesunlimitednj.com/growing-cherry-trees-in-new-jersey/)

Original image (resized from 1600x1200 to 800x600 for testing purposes)

Output image after seam carving (size 500x700)

From the example above, we can clearly see that a lot of the clouds, mountains and grass patches are removed, however the cherry tree as well as the dark grass on the left edge are fully preserved.

Here's another example of Rio Tinto, an acid-mine drainage system in Spain (source: https://blog.fuertehoteles.com/en/activities/river-tinto-optical-illusions/), originally 420x640.

Resized to 300 by 500, we get

This looks pretty good. The trees manage to obscure any major artifacts, and the larger trees and piece of something in the river remain distinct.

Let's see what happens to a landscape without distinct features. Here's the Windows XP background (421x590 to 300x500) (source: https://slate.com/technology/2014/04/charles-o-rear-is-the-photographer-who-took-the-windows-xp-wallpaper-photo-in-napa-valley.html)

It’s a little weird on the edge of some of the hills, but mostly still looks like the good old friend we know and love.

Finally, here's Parrish Beach from Swarthmore's website, originally 430x720.

Resized to 350 by 650 we get:

We've finally found a pathological example. As there are a lot of edges in the foreground and background, we end up removing seams from the right chair, which leads to it looking all smushed. Also, the more we reduce, the harder it is to make sure that we are removing seams without foreground objects.

IV. Limits:

Because we have to recalculate the energy map and M to find the minimum seam to remove in each iteration, seam carving can be excruciatingly slow on large photos. It is also slow if we want to remove too many seams. There are also methods to find the optimal way to remove seams, using a transport map, but this requires essentially calculating every possible way one can remove seams in order to find the one with the lowest energy. This is extremely computationally intensive, and not practical. Moreover, our results look good enough with an alternating order that it is likely that having an optimal order would not end up having a significant advantage in image quality.

The other main computational limit is that the algorithm is significantly slower on larger images. This is a bit of a practical issue, since one reason one might want to use this particular algorithm is to make larger images smaller, rather than smaller ones smaller. For this reason faster implementations of seam carving would be better. One person (https://github.com/li-plus/seam-carving) has made a "super fast" implementation of this, which they say takes 2~3 seconds to remove 200 rows or columns of pixels, from an image that is 600X407, so even that is quite slow. This may just be a limitation of the method as it currently exists.

A limit with the method itself is that we must restrict ourselves to taking away only one pixel from each row (if a vertical seam) or column (if horizontal) in order to keep images rectangular. This means that the lowest energy seam may not be the lowest energy path through the image, and limits the possible slope that a seam may have to being 1 pixel/pixel. This is part of why we end up taking seams through the chair--the area above the chair has low energy relative to much of the image, but going through the chair is necessary to form a seam there.

If we could work on this project for two more weeks, we would want to expand it to try and make it faster anyways, perhaps with numba or cython as recommended. As we have no familiarity with these tools, it would take some time to examine them. It would also be interesting to see if there are any other types of content-aware resizing algorithms that have been developed that don't rely so heavily on the slow process that is dynamic programming. It also might be interesting to try the transport map, which we have coded up, on the Strelka cluster and see what the results of applying it are. We could also compare to the version already implemented in scikit, to see how it compares in quality and speed.

Finally, a similar process which might be interesting is image enlargement that is content aware. Theoretically, this can be accomplished by adding more pixels near low-energy seams that are the averages of their surroundings. This would mostly require the addition of two functions to add seams instead of remove them, as the energy map finding and accumulated cost finding functions would not change. We anticipate that this would be slower, though, because rather than calculating continually smaller images (corresponding to faster finding of M), we continually increase the size of the image which we then need the properties of.

V. Conclusion:

Overall, we successfully implemented seam carving from scratch. While some of our stretch goals did not turn out to be feasible due to computation time, the general process was successful, and our main project goal was thus satisfied. While it is inherently a slow algorithm, it is indeed effective at performing content aware resizing of images.