Martin Mayr, Vincent Christlein, Mathias Seuret, Thomas Köhler, Prathmesh Madhu

Seam Carving

Deadline: May 17th 2021

Energy Function

For this exercise, you will have to compute the energy function of an image. This will be done in three steps: first on a simple 2D array, then on a small 3D array mimicking a color image, and finally on a real image.

Exercise 4.1: Removing columns and rows

For this first exercise, you will have to compute an energy function for an image, and remove the rows and columns which have the smallest energy to downscale the image. First implement the method without mask, and try it on the provided common kreskel image, and replicate the result shown in the lecture. Then, apply it either to the kingfishers image, or to another image of your choice. You will notice that important elements get distorded. As a final step, you will have to implement the mask approach to preserve some elements, and apply it to the kingfishers picture, or any other of your choice.

Energy function

To compute the energy function, you can use the Laplacian operator:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{1}$$

It is also possible to use other functions with larger kernels; do not hesitate to try things out!

You can convolve it on the image using `scipy.signal.convolve`. As the image has three channels, apply the filter on each of them, and make the sum of the results.

If the operator is applied only where it would fit on the image, then the result would have a smaller size[1]. A typical solution is to pad the input by appending rows and columns such that the output has the same size as the original input. You can either do it manually, or use `'same'` as `method` parameter of `convolve`.

Hint: you will compute energy functions many times – why don't you define a function for it? The same will apply to other parts of the code.

[1] It would not be possible to compute the operator on the border of the image.

Martin Mayr, Vincent Christlein, Mathias Seuret, Thomas Köhler, Prathmesh Madhu

Deleting rows/columns

You can compute the cost of a row or a column by using `numpy.sum`. Use 0 or 1 as `axis` parameter to indicate whether the sum has to be made horizontally or vertically.

Find the index of the row or column to delete using `numpy.argmin`.

Hint: you can have a single implementation (removing either rows or columns), and rotate your data with `numpy.rot90`.

Hint: remove rows and columns on both the RGB and grayscale images, and update the cost array.

Using a mask

Load the mask as a grayscale image, and add it element-wise to the cost array. You will also need to resize this mask along with the image.

Apply the resizing method to the common kestrel picture, using the provided mask.

Optional: some optimizations

For later exercises, it will be beneficial (but not mandatory) to store the different arrays as multiple channels of a single array, as described during the lecture.

Exercise 4.2: path finding

You will find the array shown in the slides in the file `ex2array.py`.

Cumulative cost

Compute the cumulative cost as shown in slide 18 of the lecture. You will have to process each row one after another, starting from the top.

You can compute the minimum filter either manually, or by using the function `scipy.ndimage.minimum_filt` Note that you will have to be careful with boundary issues: for the left-most column, for example, you will have to compute the filter on only two elements.

hint: if you use `minimum_filter1d`, then you can set the `mode` parameter to `'reflect'`. Padding the input with zeros would cause issues, as the cost sum along the padded area would always be null.

Computing the path

As seen in the lecture, compute the path from bottom to top. Note that you will then have to inverse the list of coordinates which you get – using a stack or a deque would be appropriate.

Martin Mayr, Vincent Christlein, Mathias Seuret, Thomas Köhler, Prathmesh Madhu

Exercise 4.3: use seam carving

Still working on the same array, apply seam carving – that is, compute paths, and remove cells which are along the minimum cost paths.

Hint: use ':' when indexing the arrays to avoid using too many loops.

Exercise 4.4: seam carving on images

For this exercise, you will have to merge together the codes which you wrote in the previous exercises.

Base your code on what you wrote for Ex. 4.1. Replace the search of a row/column by the computation of the lowest cost path, and the deletion of rows/columns by the deletion of cells along the path.

As before, do not forget to update the energy function and cumulative cost after each image update.

Hint: it will be slow; you will maybe want to work on smaller scales of the images until your code works properly. I would suggest to start with images which largest dimension is lower than 500 pixels.

Hint: save frequently the resulting image (with different filenames) so that you can check what is happening. It will be useful for debugging your code.

Keep track of seams

As seen in the part of the lecture about upscaling images, it might be beneficial to avoid having too many seams at the same place. Keep track of the seams by adding values to the mask where the seams are.

Compare the seam carving of an image with, and without this process.

Hint: make sure that you do not lose these values that you added when resizing the mask. Saving the mask as an image and checking how it looks like can be useful for debugging purpose.

Make the kingfishers get closer

The kingfishers are sitting far from each others. Use seam carving to bring them closer.

How close can you make them before distortions become annoyingly visible?

Make it faster

Seam carving for resizing images is unfortunately slow. The two main parts which cause this are computing the energy function and the cumulative cost. However, when a path is removed, then we can assume it would not be necessary to update these arrays for the whole image, but only for the neighborhood of the path.

Compute the standard deviation σ of the path. Then, we can assume that, in most cases, neighbor paths would have similar standard deviations, so it is unlikely that erasing a path will have an impact on the cumulative cost array further than a few σ.

Thus, instead of updating the whole energy and cumulative cost arrays, you can resize them in the same way as the image, and update only the cells which are at a distance to the path inferior to $2 \cdot \sigma_r + 1$, where σ_r is σ rounded to the upper value.

Enjoy your new cruise speed.

Exercise 5: upscaling images

Upscaling images is very similar to downscaling them. Instead of deleting the cells along the path, insert some on the left of the path. This can be done by increasing the width of the arrays, and offsetting to the right by one pixel the values of all cells starting from the path. This leaves a gap to fill – use the average of the values on its left and right.

First, try to upscale the picture of Vincent Christlein on a cliff to make it twice larger. You should get a result similar to the one shown in the slides. Do not forget to make a mask so that you do not distort your teacher!

Then, try it on a few other images of your choice – either photos you took, or photos found on Internet.

Compare nature landscales and cityscapes

Compare results obtained on these two kinds of images. Some parts of the images are rescaled without it being visible, while it's clear in other ones. Can you identify them?

Optional exercise 6: share an image

Apply heavy modifications to an image of your choice using seam carving, and share the results in the forum :-)

Optional exercise 7: share a video

Select a portrait photo, either of yourself or of a star you like/dislike. Apply heavy seam carving on it, saving all different scales. Then, resize all images to their original size (with a "normal" method, e.g., mogrify), and make a video out of them, as in the one given in the additional material.

Share the result in the forum so that everybody can enjoy it.

Exercise 8: text line segmentation

Do either exercise 8, or exercise 9. Or both if you feel motivated :-)

For this exercise, you will have to apply seam carving to at least one of the provided text images:

- e-codices_acv-P-Antitus_027v_large.jpg, the easiest one, with very regular text and almost no overlapping
- e-codices_kba-0016-2_006v_large.jpg, a more difficult one with large ascenders and descenders, and more overlapping

Exercise 8.1: computing projection profiles

Either binarize the images, or increase their contrast. Then compute projection profiles by summing the cost of pixels along rows. Smooth the result (with a moving average or median), and plot it.

Exercise 8.2: searching for line starts

Now, apply the projection only to the left side of the image (e.g., the first 25% of the image's columns). Look for local minimas and consider them as line start. Display them on the text image. You might have to tune the smoothing step, or the amount of considered columns, if you do not get good results.

Exercise 8.3: compute seams

Now that you have the line starts, you can compute seams to segment the text lines. For each cut, proceed as follows:

1. Compute the cost function on the image,
2. Store the largest row sum as m,
3. Manually set the value of all cells along the left border to m, excepted for the considered line start, which you set at a low cost (e.g., 0 or $-m$)
4. Compute the cumulative costs and the seam

Finally, display the text image with the seams in red, as in slide 21 of the lecture.

Martin Mayr, Vincent Christlein, Mathias Seuret, Thomas Köhler, Prathmesh Madhu

Exercise 9: variants of the optimal path

Do either exercise 8, or exercise 9.

In Slide 14 of the lecture, it is said that all minimum cost paths are equivalent. From a mathematical point of view, they are, however they lead to different results. In this exercise, you have to investiage slightly different strategies for dealing with multiple path finding variants.

Please, apply this to at least four pictures taken by you, in addition to the one of the kingfishers, and of Vincent on a cliff.

Exercise 9.1: take four photos

Take two photos taken in a city (containing mostly buildings), and two photos taken in a park or in the nature (with no or few buildings visible). Downscale them with seam carving, and use the results for comparison purpose in the following exercises.

Exercise 9.1: prefer vertical seams

Previously, it did not matter whether seams move much horizontally. In this exercise, try to privilege verticality in the seams by modifying the cost function, such that if there would be only a slihtly higher cost when moving vertically than moving in a diagonal, then the method will prefer to move vertically. This can be achieved by adding a small extra-cost, when computin the cumulative sum, to the pixels which are on the side. Compare downscaled images with this approach, and with the standard approach - can you observe anything?

Exercise 9.2: larger strides

So far, paths can move one pixel to the left or one pixel to the right at a time. Modify your code to allow larger strides (e.g., ± 3). How does this impact the resulting images?

Exercise 9.3: prefer linear seams

For this final exercise, you have to modify the path finding algorithm such that it aims at preserving the linearity of the seams. This means that when computing a seam, if the selected pixel is on the left side, for example, then on the next step the algorithm should prefer continuing toward the left, even if it leads to a cost *slightly*[2] higher. The same applies for the right, and upwards. Try this as well with larger strides.

What can you observe when applying this approach to your photos?

[2] How much is up to you – maybe you can experiment with this value as well.