Seam Carving

Deadline: 11/4/2022 (23:55)

Introduction

Content-aware image resizing changes the image resolution while maintaining the aspect ratio of important regions. In this exercise, you will implement and explore image resizing using the seam carving algorithm and other basic image processing operations.

You are provided with a basic skeleton project which supports the following features:

- It resizes an input image using simple nearest-neighbor interpolation
- It outputs the image gradients

You need to extend the provided project and support the following features:

- Image resizing via basic seam carving algorithm with a given energy function
- Image resizing via seam carving algorithm with forward energy looking energy function

Your program should output the following:

- Resized image with the specified output dimension
- If seam carving method is chosen, you will also output visualization images with the chosen seams colored in red and black for horizontal and vertical seams, respectively. See next section for further details.

We will use python as our primary programming language. Please refer to the instructions on moodle on how to setup a python project using PyCharm and Anaconda environment.

Seam Carving - Overview:

In this exercise we will implement the seam carving algorithm. In the basic algorithm you need to define an energy function that specifies the “importance” of each image pixel. You can calculate the pixel importance using the image gradient. There are several ways you can calculate the image gradient, one option is given below:

Image Gradient:

To compute the image gradient, you can first convert the image into grayscale image (see utils.to_greysacale in the provided skeleton). Then the gradient magnitude $E(i,j)$ at pixel (i,j) can be defined as:

$$E(i,j) = \sqrt{\frac{\Delta_x^2 + \Delta_y^2}{2}}$$

where Δ_x^2 is the squared difference between the current and next horizontal pixel (in grayscale), and Δ_y^2 is the squared difference between the current and next vertical pixel (in grayscale). More specifically:

$$\Delta_x^2 = \left(I_{gs}[i,j] - I_{gs}[i,j+1]\right)^2$$
$$\Delta_y^2 = \left(I_{gs}[i,j] - I_{gs}[i+1,j]\right)^2$$

This function is already implemented for you (see utils.py). Go over it and understand the implementation.

Algorithm outline:

Reducing/Increasing the image width by k pixels:

It is recommended to set the grayscale image as your working image. This implementation can be relevant when implementing the forward energy function.

- Compute the image energy function (image gradient) E as defined in the previous section
- For 1…k:
  - Use dynamic programming to find the optimal vertical seam by calculating the cost matrix M.
  - Find the actual seam by finding the smallest cost in the bottom row, then start going up on a path of minimal costs.
  - Remove the seam from the grayscale image.
  - Store the order and pixels removed in each iteration.
- To reduce image size by k pixels, remove all chosen seams from the original image.
- To enlarge by k pixels, duplicate all chosen seams from the original image.

Reducing/Increasing the image height by k pixels:

To change the size in the height-dimension, you can rotate the image by 90 degrees counter-clockwise and apply the algorithm we outlined above on the rotated image. Once done, you can undo the rotation.

Reducing/Increasing the image width and height:

There are many ways to change the image size in both dimensions (height and width). For simplicity, we will first change the image width, and only then change the image height.

Forward Looking Energy Function:

Seam-carving can introduce artifacts to the resized images. To reduce these artifacts, you will need to implement the forward-looking energy function:

$$M(i,j) = \mathrm{E(i,j)} + \min \begin{cases} M(i-1,j-1) + C_L(i,j) \\ M(i-1,j) + C_V(i,j) \\ M(i-1,j+1) + C_R(i,j) \end{cases}$$

Where (recall I_{gs} is the grayscale image of the input):

$$C_L(i,j) = \left|I_{gs}(i,j+1) - I_{gs}(i,j-1)\right| + \left|I_{gs}(i-1,j) - I_{gs}(i,j-1)\right|$$
$$C_V(i,j) = \left|I_{gs}(i,j+1) - I_{gs}(i,j-1)\right|$$
$$C_R(i,j) = \left|I_{gs}(i,j+1) - I_{gs}(i,j-1)\right| + \left|I_{gs}(i-1,j) + I_{gs}(i,j+1)\right|$$

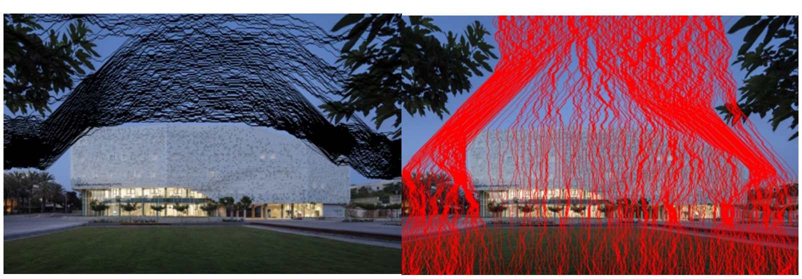
Again, use dynamic programming to find the cost matrix M. We will set E(i,j) to zero in the forward energy implementation.

Once the matrix is found, you need find the optimal seam. Be careful when retrieving the seam from the final cost matrix. You need to "**reverse**" back the decisions that led to the final optimal seam cost. This can be found by:

- Find the index of the optimal cost in the bottom row.
    - For every row from bottom to up:
        - Chose the direction that led to the current pixel cost
        - Go up in that direction, this will be the next seam pixel

Seam Visualization:

In addition to the resized image, you will need to return **two additional** images which visualize the chosen seams in each direction (vertical and horizontal). To do so, you will need to handle vertical seams and only then do horizontal seams. You need to create a copy of the original image (before resizing) and color the vertical seams with red. Then, handle the vertical seams and resize the image width. Finally, create a copy of the partially resized image and similarly color the horizontal seams; use black instead. See the illustration below (in this example, we didn't perform resizing in both directions simultaneously):



Requirements:

- Your solution should be implemented using Python (use python version 3.8)
- You may use the python packages (Pillow, numpy) – other libraries are prohibited.
- Your program should be a command-line application with the following options:
    - --image_path (str)– An absolute/relative path to the image you want to process
    - --output_dir (str)– The output directory where you will save your outputs.
    - --height (int) – the output image height
    - --width (int) – the output image width
    - --resize_method (str) – a string representing the resize method. Could be one of the following: [‘nearest_neighbor’,
      ‘seam_carving’]
    - --use_forward_implementation – a boolean flag indicates if forward looking energy function is used or not.
    - --output_prefix (str) – an optional string which will be used as a prefix to the output files. If set, the output files names will start with the given prefix. For seam carving, we will output two images, the resized image, and visualization of the chosen seems. So if --output_prefix is set to “my_prefix” then the output will be my_prefix_resized.png and my_prefix_horizontal _seams.png, my_prefix_vertical_seams.png. If the prefix is not set, then we will chose “img” as a default prefix.

Tips and Guidelines:

- When applying seam-carving, especially in the forward-looking implementation, make sure you work on a grayscale image. Once you find the seams, you can then handle them by removing/duplicating their corresponding pixels in the original image.
- When you remove a seam, all pixels to the right of it are shifted left by one. You will have to remember their original position. Use a helper array for that, e.g. an array with the size of the image, with the original column indices in each row:

```
0 1 2 3 4 5 6
0 1 2 3 4 5 6
0 1 2 3 4 5 6
```

becomes

```
0 1 3 4 5 6
0 1 2 4 5 6
0 1 2 4 5 6
```

- At the sides (left/right), you don't have all three options for the costs. What you should do, is using the options you can.
  For example, at a leftmost pixel ($x = 0$), the cost would be:
  $$M_{i,0} = E(i,0) + \min \begin{cases} M_{i-1,0} + C_V(i,0) \\ M_{i-1,1} + C_R(i,0) \end{cases}$$
  The case of the rightmost pixels is equivalent.
- At the top row ($y = 0$), the cost would be:
  $$M_{0,j} = E(0,j)$$

* Make a small synthetic test image that would make it easy for you to debug (e.g. 5x5).
* Useful functions: numpy.rot90, numpy.zeros_like, numpy.zeros, numpy.copy
* You may use utility functions given in utils.py

Submission Guidelines:

1. Submission is in pairs.
2. Submit your zip files through the Moodle site of the course.
3. Submit **.zip** file titled ex1_<id1>_<id2>.zip, where <id1> and <id2> are your ids
   a. Make sure your main.py file is in the top-directory inside of the zip file. This means the zip file structure should be something like:
```
/
|   main.py
|   utils.py
|   seam_carving.py
|   nearest_neighbor.py
|__ /directory1
|     |   file1.py
|     |   file2.py
|     |__________
|
|__ /directory2
      |   file1.py
      |   file2.py
      |__________
```
   b. You don't need to add new files or directories if you don't want to.
4. Your code should run in a reasonable time: for example, scaling an image of size 600x800 by 50% in each dimension should take no more than few seconds.

Good Luck!