CAI 4841: Computer Vision, Fall 2025

Problem Set 1

Instructor: Utkarsh Ojha (uojha@usf.edu)
TA: Mahit Venkat Gautam Kumpatla (mahitvenkatgautam@usf.edu)

Due: Monday, September 22nd, 11:59 PM

Instructions

1. Your main workspace will be Google Colab. Write all your code that implements the problem in one of the python notebooks.
2. Submissions must be uploaded on Canvas and include (i) a single ipython notebook uploaded on Canvas, and (ii) a corresponding .py file (click on File -> Download -> Download .py).
3. The name of your files should be *FirstName_LastName_PS1.ipynb* and *FirstName_LastName_PS1.py*.
4. The notebook should contain all the code and all the outputs from each cell, matching the requirements as mentioned below.
5. Please do not create subdirectories within the main directory, and make sure that all the functions are described together in the main file.
6. You are to work alone on this assignment. We will be using different forms of plagiarism checking. So, if you have used the help of any resource (e.g., internet, classmate), make sure to mention that in the last cell of the notebook.
7. Make sure your code is documented, is bug-free, and works out of the box. Please be sure to submit all main and helper functions. Be sure to not include absolute paths. Points will be deducted if your code does not run out of the box.
8. If plots are required, you must include them in your notebook and your code must display them when run. Points will be deducted for not following this protocol.
9. Late penalty: after the submission due date, you will get three additional late days with increasing penalty. If you submit
   a. <24 hours after the due date, penalty is -10 points.
   b. >24 hours and <48 hours after the due date, penalty is -25 points.
   c. >48 hours and <72 hours after the due date, penalty is -50 points.
   d. after 72 hours post the due date, you get 0 for the assignment.

Content-aware image resizing: Seam Carving [Total: 100 points]

You will implement a version of the content-aware image resizing technique described in Shai Avidan and Ariel Shamir's SIGGRAPH 2007 paper, "Seam Carving for Content-Aware Image Resizing". The paper is available on Canvas under Week 3 module. The goal is to implement the method, and then examine and explain its performance on different kinds of input images.

First read through the paper, with emphasis on sections 3, 4.1, and 4.3. Note: choosing the next pixel to add one at a time in a greedy manner will give sub-optimal seams; the dynamic programming solution ensures the best seam (constrained by 8-connectedness) is computed. Use the dynamic programming solution as given in the paper and explained in class.

Write **Python** code with functions that can do the following tasks. Save each of the below functions in the same python-notebook file described above.

- `energyImg = energy_img(im)` - to compute the energy at each pixel using the magnitude of the x and y gradients. The gradients should be computed with the grayscale converted image. To convert the RBG image into a single channel grayscale image, use the following weighing scheme (R:0.299, G: 0.587 and G: 0.114). You do not need to smooth the image before computing the gradients. The input `im` should be an MxNx3 matrix of datatype uint8. For this specific function, you can use `np.gradient` to compute the gradient. The output energyImg should be a 2D matrix of shape MxN of datatype double.
- `cumulativeEnergyMap = cumulative_min_energy_map(energyImg,seamDirection)` - to compute minimum cumulative energy. The input energyImg should be a 2D matrix of datatype double (the output of `energy_img` function defined above). The input `seamDirection` must be the strings 'HORIZONTAL' or 'VERTICAL'. The output `cumulativeEnergyMap` must be a 2D matrix of datatype double.
- `verticalSeam = find_vertical_seam(cumulativeEnergyMap)` - to compute the optimal vertical seam. The input `cumulativeEnergyMap` should be a 2D matrix of datatype double (the output of the `cumulative_min_energy_map` function defined above). The output `verticalSeam` must be a vector containing the column indices of the pixels which form the seam for each row.
- `horizontalSeam = find_horizontal_seam(cumulativeEnergyMap)` - to compute the optimal horizontal seam. The input `cumulativeEnergyMap` should be a 2D matrix of datatype double (the output of the `cumulative_min_energy_map` function defined above). The output `horizontalSeam` must be a vector containing the row indices of the pixels which form the seam for each column.
- `view_seam(im,seam,seamDirection)` - to display the selected type of seam on top of an image. The input `im` should be an MxNx3 matrix of datatype uint8. `seam` can be the output of `find_vertical_seam` or `find_horizontal_seam`. `seamDirection` can be the strings 'HORIZONTAL' or 'VERTICAL'. The function should display the input image and plot the seam on top of it.
- Functions with the following interface:

`[reducedColorImg,reducedEnergyImg] = decrease_width(im,energyImg)`
`[reducedColorImg,reducedEnergyImg] = decrease_height(im,energyImg)` These functions should take as inputs (a) an MxNx3 matrix `im` of datatype uint8 and (b) a 2D matrix `energyImg` of datatype double. The input `energyImg` can be the output of the `energy_img` function. The output must return 2 variables: (a) a 3D matrix `reducedColorImg` same as the input image but with its width or height reduced by **one** pixel; (b) a 2D matrix `reducedEnergyImg` of datatype double same as the input `energyImg`, but with its width or height reduced by **one** pixel.

Now, after implementing each of the functions described above, write code to make use of the functions in subsequent cells, one cell having the code for each task.

1. [15 points] Load a color input image called `inputSeamCarvingPrague.jpg` from Problem set 1 section in Canvas. Reduce the width of the image by 100 pixels using the above functions. Saves the resulting image as `outputReduceWidthPrague.png` and display this output. Repeat the steps for an input image called `inputSeamCarvingMall.jpg` (available in the same folder as previous image). Display the output for this image as well.
2. [15 points] Repeat the above steps for both input images but reduce the height by 50 pixels. Save outputs as `outputReduceHeightPrague.png` and `outputReduceHeightMall.png`, respectively.
3. [15 points] Display in your notebook: (a) the energy function output for the provided image `inputSeamCarvingPrague.jpg`, and (b) the two corresponding cumulative minimum energy maps for the seams in each direction (use `matplotlib.pyplot.imshow` function). Explain why these outputs (cumulative energy maps) look the way they do given the original image's content.
4. [15 points] For the same image `inputSeamCarvingPrague.jpg`, display the original image together with (a) the first selected horizontal seam and (b) the first selected vertical seam. Explain why these are the optimal seams for this image.
5. [20 points] Make some change to the way the energy function is computed (i.e., filter used, its parameters, or incorporating some other prior knowledge). Display the first horizontal and vertical seams for `inputSeamCarvingPrague.jpg` and explain the impact. You will need to also include the code for new filtering operation. Implement the 2D filtering operation yourself using one of the edge detection filters explained in lecture 3, i.e., writing the sliding window utility without using any pre-existing library.
6. [20 points] Now, for the real results! Use your system with different kinds of images and seam combinations, and see what kind of interesting results it can produce. The goal is to form some perceptually pleasing outputs where the resizing better preserves content than a blind resizing would, as well as some examples where the output looks unrealistic or has artifacts. Include results for at least three images of your own choosing. Include an example of a "bad" outcome. Be creative in the images you choose, and in the amount of combined vertical and horizontal carvings you apply. Try to predict types of images where you might see something interesting happen. It's ok to fiddle with the parameters (seam sequence, number of seams, etc.) to look for interesting and explainable outcomes. For each result, include the following things, clearly labeled (see title function) and using the subplot function for displaying the images:
    a. the original input image
    b. your algorithm's resized image
    c. the result using the more standard resizing, e.g., `cv2.resize`

d. an explanation of what we are seeing in the output.