Sapir Avrahami
23.6.2024

# SkillReal Home Assignment Answers

NOTE: The questions were answered in python.

1. For question 1, I was given 2 pictures that were taken from the same camera (which is positioned at the exact place and does not move) at different times. I created the ImageComp.py file inside the ImageCompariosn folder.
   For part 1, I was asked to find the differences in the two pictures given a specific threshold. I created a few functions that will help me with that.
   I created the function find_differences() that takes 2 images as arguments as well as a threshold.
   The function checks that the shape of the images is the same, or else it can't run.
   Then using the openCV library, it finds the absolute differences between the images.
   It uses the given threshold to create a mask for any difference that exceeds the threshold.
   Finally, it applies the mask to the original difference image and returns it.
   For part 2, I was told that some objects could have been moved, added, or removed in the time between the 2 images. I was asked to find the number of changes and for every pixel to check how many pixels changed.
   I created the function analyze_changes() that takes the outputed threshold image of part 1 as an argument.
   I use the connectedComponentsWithStats() from the OpenCV library to identify changes.
   I then filter small noises and exclude the background.
   Then I return the number of changes and a list of dictionaries with each changed pixel and the number of pixels changed.

   Since I wanted to run multiple test cases, I created a function process_test_case() that takes as arguments a folder path of the test case folder, a results folder path and a threshold (the case folder has the two images we wish to check).
   I look for the 2 images, and if there are no exactly 2 images I return an error.
   I use the os library to get the path to the images and the OpenCV library to read the image and turn it to grayscale.
   I then run find_differences() and analyze_changes() on the inputs and process to create a results folder for the test case (inside a results folder).

I then save the thresholded difference image in the results folder as well as the list of changed pixels (save_changes_to_csv() will be talked about in a second). Then I print out the test case name, the thresholded difference image path, number of changes, and path to changes csv.
The question asked me to list each pixel with the amount it changed. This can become a long and cumbersome list. So I created a save_changes_to_csv() function. Which takes a list of dictionaries and location path and saves the list into a csv.

SInce all of this runs and saves 1 test case, in the main() function I created an argument parser using Python's argparse library. The user can give a folder location path of where all the test case folders are located and a threshold to be used. If no arguments are given, default values are used.
Then the code runs on all test case folders.

2. For question 2, I was asked about Regions of Interest in a picture (or ROIs) and to check if they overlap. I created the Overlap.py file inside the SqaureOverlap folder.
   For part 1, I was asked to write a function that receives coordinates of 2 ROI squares and checks if they overlap.
   I created the squares_overlap() function.
   It receives 2 tuples that represent 2 squares to be checked. Each tuple consists of (x1, y1, x2, y2) where (x1, y1) are the coordinates of the top-left corner and (x2, y2) are the coordinates of the bottom-right corner of the square.
   If the coordinates of 1 rectangle overlap with the coordinates of the other then they overlap and we return false. Return true otherwise.
   For part 2, I was asked to create a function that checks and displays the function from part 1 by (1) creating N random ROIs, (2) draws and displays the ROIs to the screen, (3) any overlapping ROIs are colored differently.
   I created 2 functions: generate_and test_rois() and visualize_rois().
   generate_and test_rois() receives the number of ROIs to generate, the image size, and min and max size limit of ROIs.
   The function uses the random library to generate rois coordinates and saves them in a list.
   It then runs over the list with a nested for loop (O(n^2)) and calls square_overlap() on each pair of square coordinates.
   If they do overlap it adds them both to a set of overlapped ROIs. The reason I am using a set and not a list is because it is possible that 1 ROI overlaps with multiple different ROIs. If I use a list, it will add the same ROI to the list multiple times. A set does not allow duplicates so it is better in this case.
   It then returns both the list of all ROIs and set of overlapped ROIs.

After this, visualize_rois() takes the list of ROIs, set of overlapped ROIs and image size.
Using the matplotlib library it creates a graph in the dimensions of the given size. It then runs over the list of ROIs and plots them. If the ROI is also in the overlapped set, it paints it red. Blue otherwise.
Finally, it added all coordinates and visual text to the graph.
It displays the resulting graph and prints out the parameters used, number of generated ROIs, number of overlapping ROIs, and number of non-overlapping ROIs.

To run this program I enabled 2 possible options.

- If a user wishes to run the program only once it can use ‘python Overlap.py -n 50 -iw 1000 -ih 1000 -m 10 -M 100’
  Where:
  - -n Number of ROIs to generate.
  - -iw Width of the image.
  - -ih Height of the image.
  - -m Minimum size of ROI.
  - -M Maximum size of ROI.

  Parameters that were not changed will use default values.
  This will call run_single_test() which will run the test once.
- If a user wishes to run the program multiple times, they will need to use ‘python Overlap.py –c path/to/test_cases.csv’ and provide a path to a test_cases.csv file with all needed parameters (example file is provided in the code).
  This will call run_multiple_tests() which will run over all test cases in the given csv.