CS7.404. Digital Image Processing
Monsoon-2022
Assignment-3
Posted on: 24/09/2022
Due on: 14/10/2022, 23:59 Hrs IST

- All your code should be in the `src` directory and images in `imgs` directory.
- Do not use any external library other than **`numpy`** for implementing any of the tasks. You can however use external libraries for I/O operations and plotting. If you are not sure if a library is allowed for a particular task, clarify with your TAs.
- You will be evaluated on correctness and how vectorized your code is - with correctness being the priority.
- Write modular code with relevant docstrings and comments for you to be able to use functions you have implemented in future assignments.
- All theory questions and observations must be written in a markdown cell of your jupyter notebook.
- All academic integrity policies apply. Check the course web page for more clarity.
- Start the assignment early, push your code regularly and enjoy learning!

---

1 Ambitious Jo

Seeing your expertise in Image Processing (from the last assignment), Sabre's CEO, Jo Bennett has now decided to add more software features to their scanners. After thorough market analysis they have decided that they want to sell an automatic correction system for OMR sheets - since she knew that such a system, if robust enough, will do well in a country like India - a country with majority of the large-scale competitive exams being objective. Develop a system for Sabre that will achieve this goal.

Assume that the questions had only single correct answers. Do the following:

1. Generate the answer key for the 45 questions given using `answerKey.png`
2. Compute the score of the student - whose answer sheet is `sampleStudentOMRSheet.png`. The exam had the following marking scheme:
   (a) +4 if the answer is correct.
   (b) -1 if the attempted answer is wrong.
   (c) 0 if a question un-attempted.

3. **(Bonus)** Modify the functions you have just written to output the roll number of the student.

Advice: This question can involve a fair amount of experimentation - so start early.

Expectations: We expect you to implement morphological operations for this question.

1.1 Automatic Grading

This question will be auto-graded using Github Classroom's Autograding Workflow. Read the instructions carefully to submit your solution successfully.

Problem Specifications:

1. Assume that all input test OMR sheets will **align exactly** with `answerKey.png` - and will only change in terms of the answers selected (or not selected).
2. Your task is to find out the choice filled out by the student for each question.

Input:

1. The first line contains a single integer T ($1 \leq T \leq 100$) — the number of test cases. Then T test cases follow.
2. Each test case consists of one line - the (relative) path to the test OMR image.

Output:

1. 45 lines for each test case - each line containing the option selected. They have to be one among $\{A, B, C, D, -1\}$
2. If the student has not answered a particular question, return -1.

Script Specifications:

1. This workflow requires that you write your functionality in the file having the following path `src/morOMR.py`
2. If you are new to automatic submissions of this kind or is confused about the input / output, please refer to `sample_morOMR.py` here - which can be used as a boiler plate.
3. Do not use any external library other than `opencv` for reading the image and `numpy` for the algorithms. Usage of any other (external) library will automatically fail the test case.

Passing the Test Cases:

To successfully pass all the test cases, you will have to successfully achieve **all** of the following:

1. You must place your script in the right path, with the right name.
2. You must properly process the input the system will give you.
3. Your output should be in the proper format.
4. And finally, your output must be correct.

Workflow Submission and Status:

1. Get to know your workflow/test status by visiting `Actions -> Github Classroom Workflow` (left panel) `->` Choose the commit of your interest `-> Autograding -> Run education/autograding@v1`.
2. Everytime you commit, github workflow will automatically re-run the tests.
3. Or you could manually re-run it by going to `Actions -> Github Classroom Workflow` (left panel) `->` Choose the commit of your interest `-> Re-run all Jobs ->` (top-right corner).

Note: To get points in this question, it is mandatory to successfully pass the test cases. If you are facing difficulty in submitting the test cases, do contact the TAs during the office hours.

Invalid Options:

As mentioned earlier, **you are supposed to detect the marked option using morphological operations** and not by any other naive method. Here an option is **defined as marked iff it is fully circled.** Any other style of marking is invalid and should be considered as un-attempted. The below figure shows examples of valid and invalid marking. The test cases will include invalid markings as well.

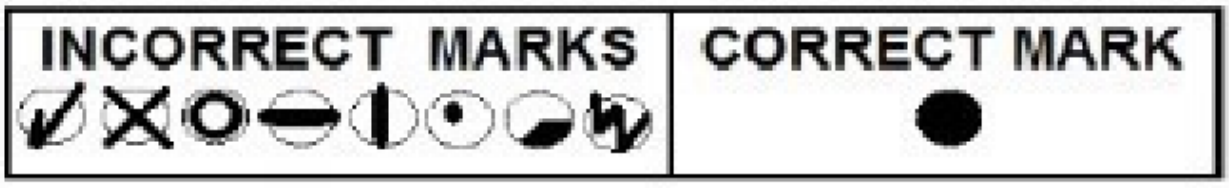

Figure 1: Incorrect vs Correct Option Marking

2 Creed's Side Gig

Creed runs a small fake ID company from his car with a laminating machine that he swiped from the Sheriff's station. He sees a bunch of his regular customers while out at the bar and decides to play a game with them. However, since Creed is a little old, he's having trouble seeing what the game exactly is. Can you modify the image of the game and find all the words in it? The initial image and expected output are given below.

Figure 2: Initial Image of the game

Figure 3: Expected Output

3 Color Operations

As a part of the new Dunder Mifflin Infinity project, Ryan is trying to add a new Digital Image Processing division to look good for the Stakeholders. He has already scammed a freelancer to implement Median Filtering, Edge Detection, Contrast Stretching, and Histogram Equalization for grayscale images. Now, however, the stakeholders are asking for these to be implemented for RGB images to look fancier for the customers. Since Ryan has been winging it since the beginning and has no idea what he's doing, you have to help him out so that he doesn't lose all hope and end up defrauding the company. You can use the previous implementations of the algorithms that you have already coded. Extend them to RGB images using the methods discussed in class.

1. Implement a function which takes in an image and filter size, and performs Median Filtering on the image. Use it for the image `salt.png`
2. Write a function which takes an image as input and returns a contrast-enhanced version of the image. Apply the function on `landscape.png` and display your result.
3. Perform Histogram Equalization on `landscape.png` and show the result.
4. Demonstrate "Vintage Effect", "Matrix Effect", Vignetting and duo-tone on any image of your choice.
   Note: You are allowed to use `cv2.LUT()` for duo-tone.

Figure 4: salt.png

Figure 5: landscape.png

4 Realistic Systems

In Question 1, you built a system that was defined for a fairly controlled setting - i.e. the test images were aligned to the answer key OMR (`sampleStudentOMRSheet.png`). However, this assumption is mostly not true in reality. While the size of the elements of interest remain same, since the answer sheets of the candidates are scanned they are often unaligned. Answer the following questions to get closer to building a more realistic *automatic omr sheet correction system.*

4.1 Image Rotation

The most common reason for nonalignment of a scanned OMR is due to translation and rotation of the image at the time of scanning. Take a look at images `rotatedOMR1.png` and `rotatedOMR2.png` to understand how realistic scans look like. One of the attributes of a realistic system would be its ability to rotate the image back to how it should have been ideally scanned. In order to do so, answer the following questions:

1. Implement a function `imrotate` which performs the rotation operation on an image about its center.
2. Your `imrotate` function should take in the following parameters:
   - Input Image
   - Angle of rotation (in degrees).
   - A boolean parameter `retainAllPixels`. If set to `True`, your function must return the rotated image containing all the pixels of the original image (this might require increasing the dimensions of the image). If set to `False`, your function should return the rotated image having the same dimensions as the original image - throwing away all pixels that do not fit in the original image's dimension.
3. Take an image of your choice, and test your `imrotate` function with angles 30°,45°,90° and any two angles of your choice in the range $[100°, 150°]$. For each of the angles, display the outputs when `retainAllPixels` is set to `True` and `False`.
4. Also test the reverse operation, i.e. for an image rotated by $x°$, rotate the resultant image by $-x°$ and display the outputs.

4.2 Hough Transform

It is evident that the scanned OMR sheets provided require rotation by some angle such that they become more aligned with an ideal OMR scanned image. Answer the following questions:

1. For rotating the image, you will need the angle of rotation. How would you go about finding the angle? Would simple edge detection technique work? Why or Why not, explain in detail?

2. Implement Hough Transform to detect lines in an image. Display the detected lines (by overlaying them with some color) on an image of your choice (your choice should **not** be the OMR sheet).

3. Use Hough Transform to find the angle by which the OMR sheets should be rotated - considering top-left corner as the origin. Now rotate the un-aligned OMR sheets (`rotatedOMR1.png` and `rotatedOMR2.png`) using `imrotate` (that you coded up in 4.1.2) and display your results. Also print the angles.

5 More Morphology

Answer the following questions by using appropriate structuring elements and morphological operations.

1. The image, `shapes.png` is a binary image containing various shapes with different properties. Answer the following questions (Use the already coded up connected components and morphological operation functions):

   - Find the number of objects having one or more hole(s). Plot the image containing only these objects.
   - Find the number of circular objects in the image. Plot the image containing only these objects.
   - Find the number of rectangular objects not containing any hole(s). Plot the image containing only these objects.
   - Find the number of circular objects containing one or more hole(s). Plot the image containing only these objects.

2. Read the binary image `circles.jpg`. Process the image appropriately to display image containing

   - only non-overlapping circles.
   - only circles which are touching the boundary.
   - only overlapping circles.

—— The End ——