B657 Assignment 1: Image Processing and Recognition Basics

Spring 2022
Due: Sunday February 20, 11:59PM
Late deadline: Tuesday February 22, 11:59PM (with 10% grade penalty)

In this assignment, you'll get experience with image operations and apply them to an object detection problem.

Coding guidelines. We strongly recommend using Python 3 for this assignment. (If you want to use another language, please contact us first; in any case, your must implement the image processing and computer vision operations yourself **and** your code runs on burrow.luddy.indiana.edu.) You can use any of the functions in the Pillow library (including basic image processing operations, convolutions, image I/O, image arithmetic, etc.) You may also use libraries for routines not related to computer vision (e.g. data structures, sorting algorithms, matrix operations, etc.). No matter which language and library you use, make sure that your program obeys the input and output requirements below (e.g., takes the right command line parameters in the right order, and creates the right output files) since we use testing scripts that automatically test your program. If you have any questions about any of this, please ask questions on Q&A Community.

Teams. As explained in the syllabus, for Assignment 1 we've randomly assigned you to a group of other students; you'll be able to choose your own groups for future assignments. See below for information about how to check your team assignment. You should only submit **one** copy of the assignment for your team, through GitHub, as described below. Part of the goal of this assignment is to successfully practice working in a team of new colleagues (which is something that is very common, whether you go to industry or academia). It's really critical that you make contact with each other as soon as possible, and that you discuss "ground rules" and expectations for how you will work together to complete the assignment. It is also really critical that you keep regular communication with the rest of your team, so that if teammates are having problems completing their work, there is enough time to develop an alternative plan. Everyone is expected to actively contribute to the assignment; please note that **the project report must include a section indicating how each person contributed to the assignment** (see Report section below). After the assignment, we will also collect confidential information from all teammates about everyone's contributions to the project.

Academic integrity. We take academic integrity very seriously. To maintain fairness to all students in the class and integrity of our grading system, we will prosecute any academic integrity violations that we discover. *Before beginning this assignment, make sure you are familiar with the Academic Integrity policy of the course, as stated in the Syllabus, and ask us about any doubts or questions you may have.* To briefly summarize, you may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials (e.g. in source code comments). We expect that you'll write your own code and not copy anything from anyone else, including online resources. *However, if you do copy something (e.g., a small bit of code that you think is particularly clever), you have to make it explicitly clear which parts were copied and which parts were your own. You can do this by putting a very detailed comment in your code, marking the line above which the copying began, and the line below which the copying ended, and a reference to the source.* Any code that is not marked in this way must be your own, which you personally designed and wrote. You may not share written answers or code with any other students, nor may you possess code written by another student, either in whole or in part, regardless of format.

The problem

In a major midwestern research university not far away, a mild-manner professor of computer science has a dream: a cheating-proof final exam that could be graded perfectly and effortlessly in has class of 100 students. He will first write 85 multiple-choice questions. Then he will randomly shuffle both the order of the questions and the order of the multiple-choice options within each question, creating 100 unique exam booklets. The first page of each exam booklet will be an answer sheet like that shown in Fig 1. The students will be asked to detach the answer sheet and turn it in after the exam. Unbeknownst to the students, each answer sheet will have a unique code printed on it that contains an encrypted copy of the correct answers for that particular answer sheet. After the exam, the instructor will scan all the answer sheets to produce 100 images, which he'll then run through a custom computer vision program to score the exams automatically. The grading program will simply read the code, decrypt the correct answers, recognize the student's marks on the answer sheet, and calculate how many questions have been answered correctly.

Your job is to help make the dream a reality, by writing an automatic grading program that will work as well as possible. Given a scan of the answer sheet like that shown in Fig 1, your program should identify the answers that the student marked as accurately and robustly as possible.

What to do

1. You can find your assigned teammate(s) by logging into IU Github, at `http://github.iu.edu/`. In the upper left hand corner of the screen, you should see a pull-down menu. Select `cs-b657-sp2022`. Then in the box below, you should see a repository called *userid1-userid2-userid3*-a1 or *userid1-userid2*-a1, where the other user ID(s) corresponds to your teammates.
2. While you may want to do your development work on a local machine (e.g. your laptop), remember that the code will be tested and thus must run on `burrow.luddy.indiana.edu`. After logging on to that machine via ssh, clone the github repository via one of these two commands:

   ```
   git clone https://github.iu.edu/cs-b657-sp2022/your-repo-name-a1
   git clone git@github.iu.edu:cs-b657-sp2022/your-repo-name-a1
   ```

   where *your-repo-name* is the one you found on the GitHub website above. (If this command doesn't work, you probably need to set up IU GitHub ssh keys. See Canvas A1 page for help.) This should fetch some test images and some sample code (see the Readme file in the repo for details).
3. Now write a program that accepts a scanned image file of a printed form like that in Fig 1, and produces an output file that contains the student's marked answers. The program should be run like this:

   ```
   python3 grade.py form.jpg output.txt
   ```

   There are 31 possible correct answers per question, because some questions might instruct the student to fill in multiple options in the same question (e.g. choices A and B might both be true so the student should mark both). The program should create an output file (second parameter in the command line above) that indicates the answers that it has recognized from the student's answer sheet. The output file should have one line per question, with the question number, followed by a space, followed by the letter(s) that were selected by the student. It should also output an `x` on a line for which it believes student has written in an answer to the left of the question (as the instructions on the answer form permit, but your program does not need to recognize which letter was written). For example, the first few lines of the output for Fig 1 should be:

   ```
   1 A
   2 A
   3 B
   4 B
   5 C
   6 AC x
   ...
   ```

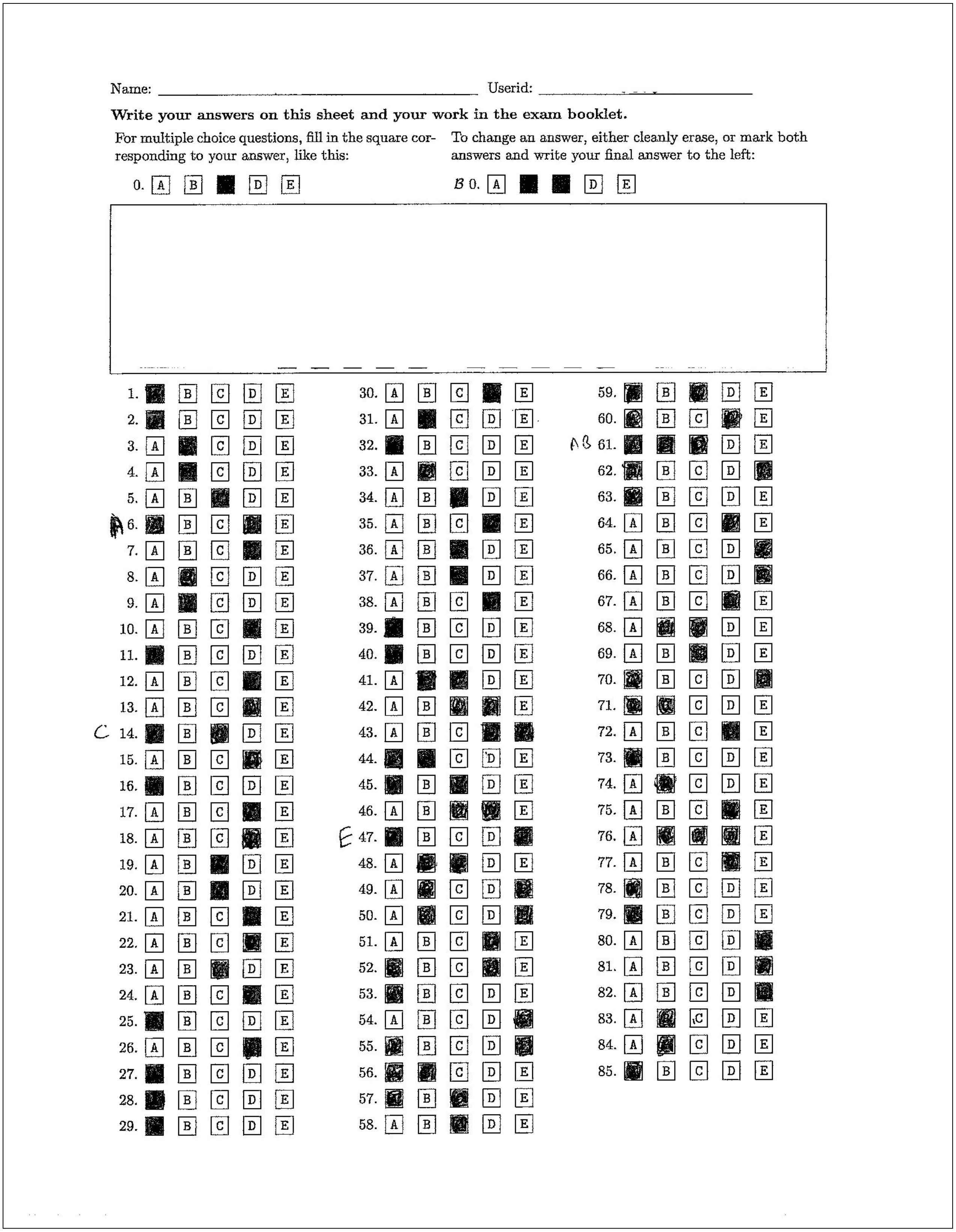

Figure 1: A sample scanned answer sheet.

4. Finally, devise a system for printing the correct answers on the answer sheet in some way so that your grading program can read them, even after the answer sheet is printed and scanned. There are many possible ways of doing this; you might add some textual annotations, or add some form of bar code, or a watermark, or some other pattern. Whatever system you adopt should not obviously reveal the correct answers to the students.

   This should consist of two separate programs, one to inject the answers into the answer sheet and one to recognize them, like this:

   ```
   python3 inject.py form.jpg answers.txt injected.jpg
   python3 extract.py injected.jpg output.txt
   ```

   where the first command takes the blank form and injects the answers (in the same file format as the output text file described above) into the image to create injected.jpg, and the second takes an image that has been injected and extracts the correct answers (writing them out in again the same text file format as above).

   Explain your technique in detail in your report (see below). Try to make your technique robust enough to be detectable even after the image has been printed, filled in by a student, and then scanned into an image file. We've provided a blank form called blank_form.jpg for you to do some experimentation along these lines, and please report about these experiments in your report.

This assignment is purposely open-ended. How should you go about it? It's up to you, but here are a few ideas. You could use edge detection and Hough transforms to try to find the alignment of the form within the page. You could use segmentation to find blobs of ink. You could use differences in color to separate ink from the printed form. You could use a cross correlation to find local image regions of interest – filled-in squares, or empty squares, or letters. It's probably much easier to figure question numbers by their position on the page as opposed to trying to actually recognize the question numbers through optical digit recognition, although you could try this if you really want to.

Evaluation. To help you evaluate your code, we've provided some test images. These are actual scans of completed test sheets, which means that they have some natural variation – slightly different positions and orientations within the image caused by the imperfect paper feeder of the scanner, variations in the ink and style across different students, etc. You can assume that these images are quite representative of the types of variations that would occur in real life; i.e., your program should try to handle these types of variations as much as possible, but we don't expect it to handle extreme cases (like answer sheets that were scanned upside down, etc). Your program only has to work for this one particular format of answer sheet.

Please use these images to evaluate the accuracy of your program and present these results in your report (see below). We've included two text files that have the expected (ground truth) output for two of the test images (and you can create your own ground truth files for the others). This provides an easy way to quantitatively evaluate your program, by simply comparing your output to the ground truth output file. Your program will almost certainly not work perfectly, and that's okay! To make things fun, we will hold a competition in which we will evaluate the programs on a separate test dataset of unseen exam sheets. A small portion of your grade will be based on how well your system works compared to the systems developed by others in the class. We may also give extra credit for additional work beyond that required by the assignment.

Report. An important part of developing as a graduate student is to practice explaining your work clearly and concisely, and to learn how to conduct experiments and present results. Thus an important part of this assignment is a report, to be submitted as a Readme.md file in GitHub (which allows you to embed images and other formatting using MarkDown). Your report should explain how to run your code and any design decisions or other assumptions you made. Report on the accuracy of your program on the test images both quantitatively and qualitatively. When does it work well, and when does it fail? Give credit to any source of assistance (students with whom you discussed your assignments, instructors, books, online sources, etc.).

How could it be improved in the future? Note that even if your code performs poorly, you can still write an interesting report that explains what you tried, what the advantages and disadvantages of that approach are, why you think it didn't work, etc. You can think of the report as an argument for why you deserve a good grade on the assignment. **Important:** As the very last section of the report, please include a section called "Contributions of the Authors" which explains how each person in your team contributed to the project (formulating an approach, writing code, conducting experiments, writing the report, etc.). Please be as specific as possible. We will not grade submissions that do not include this section.

What to turn in

You should submit: (1) Your source code, and (2) Your report, either as a Readme.md file in Github. To submit, simply put the finished version in your GitHub repository (remember to `add`, `commit`, `push`) — we'll grade whatever version you've put there as of 11:59PM on the due date. To make sure that the latest version of your work has been accepted by GitHub, you can log into the github.iu.edu website and browse the code online.

Grading

Unfortunately (or fortunately, depending on how you look at it), in computer vision there is almost never a single "right" approach that one can prove to be optimal. This means that there are many possible ways to approach this assignment and many possible paths to a good grade. Here are some basic principles for how to get a good grade: (1) Make sure that your code runs on the SICE Linux machines; if your code crashes or if we can't get it to work at all, it's very hard for us to evaluate what you've done and you will probably not get a good grade. (2) Use your report to showcase what you have done. If you tried many techniques before settling on the one you submit, talk about these failed techniques in your report! Otherwise we have no idea whether you hacked something together at the last minute, or whether you went through methodical experimentation to arrive at your solution. Be as specific and concrete as possible. If you made assumptions or design decisions, state them clearly, and discuss how they informed your eventual submission. You can think of your report as sort of an argument for why you deserve a good grade. A good report is substantive but need not be long. (3) Write clean, clear, correct, reasonably efficient code. We expect you to use good programming practices, e.g. meaningful variable and function names, comments when appropriate, etc. This is mostly for your benefit; the graders can be much more generous with partial credit if they understand what you were trying to do. (4) Your code should execute in a reasonable amount of time – under about 10 minutes. (5) The accuracy of your program is a good indication of your code's quality, since a good implementation of a well-planned approach is likely to perform better than a haphazardly implemented version of a poorly implemented one. However, if we had the choice between a submission that took an unusually creative or risky approach that was carefully planned, implemented, tested, and reported, but did not give very good results, or a submission that took a boring approach that you found in a book that gave good results but was obviously haphazardly executed, we would prefer the former.