B657 Assignment 1: Image Processing and Recognition Basics

Spring 2019
Due: Tuesday February 12, 11:59PM
Late deadline: Thursday February 14, 11:59PM (with 10% grade penalty)

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

We've assigned you to a group of other students according to your stated preferences. You should only submit **one** copy of the assignment for your team, through GitHub, as described below. After the assignment, we will collect anonymous information from your teammates about everyone's contributions to the project. In general, however, all people on the team will receive the same grade on the assignment. Please read the instructions below carefully; we cannot accept any submissions that do not follow the instructions given here. Most importantly, **start early,** and ask questions on Piazza.

You may choose to use any general-purpose programming language, with the restriction that you must implement the image processing and computer vision operations yourself, **and** your code must run on the SICE Linux servers (e.g. tank.soic.indiana.edu). For example, you may use Python, but you should implement your own convolution and edge detection (and not use some existing Python libraries). You do not have to re-implement image I/O routines (i.e. you may use Python libraries for reading and writing images). You may also use libraries for routines not related to image processing (e.g. data structures, sorting algorithms, matrix operations, etc.). It is also acceptable to use multiple programming languages, as long as your code works as required below. All that said, we recommend using either Python with the Pillow library, or C/C++ with the CImg library. No matter what 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 the course staff.

Academic integrity. You must follow the academic integrity requirements described on the course syllabus. In particular, you and your partners may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about C/C++ or Python syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials in the documentation of your source code. However, the code that your group submits must be your own work, which you personally designed and wrote. You should not use code that you find online or in other resources; the point of this assignment is for you to write code, not for you to recycle the code from others. If you do "recycle" any code, you must explicitly indicate the source of the code (e.g. URL, book citation, etc.), and indicate exactly which part(s) of your program were borrowed from other sources (by mentioning in the project report, and then explicitly labeling the line(s) of code using comments in the source files). You may not share written code with any other students except your own partners, nor may you possess code written by another student who is not your partner, 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 write a Python program to 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 write 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-sp2019`. Then in the yellow box to the right, you should see a repository called *userid1-userid2-userid3-userid4*-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 the CS Linux machines, e.g. `tank.sice.indiana.edu`. After logging on to that machine via ssh, clone the github repository:

   `git clone https://github.iu.edu/cs-b657-sp2019/`*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 Piazza 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:

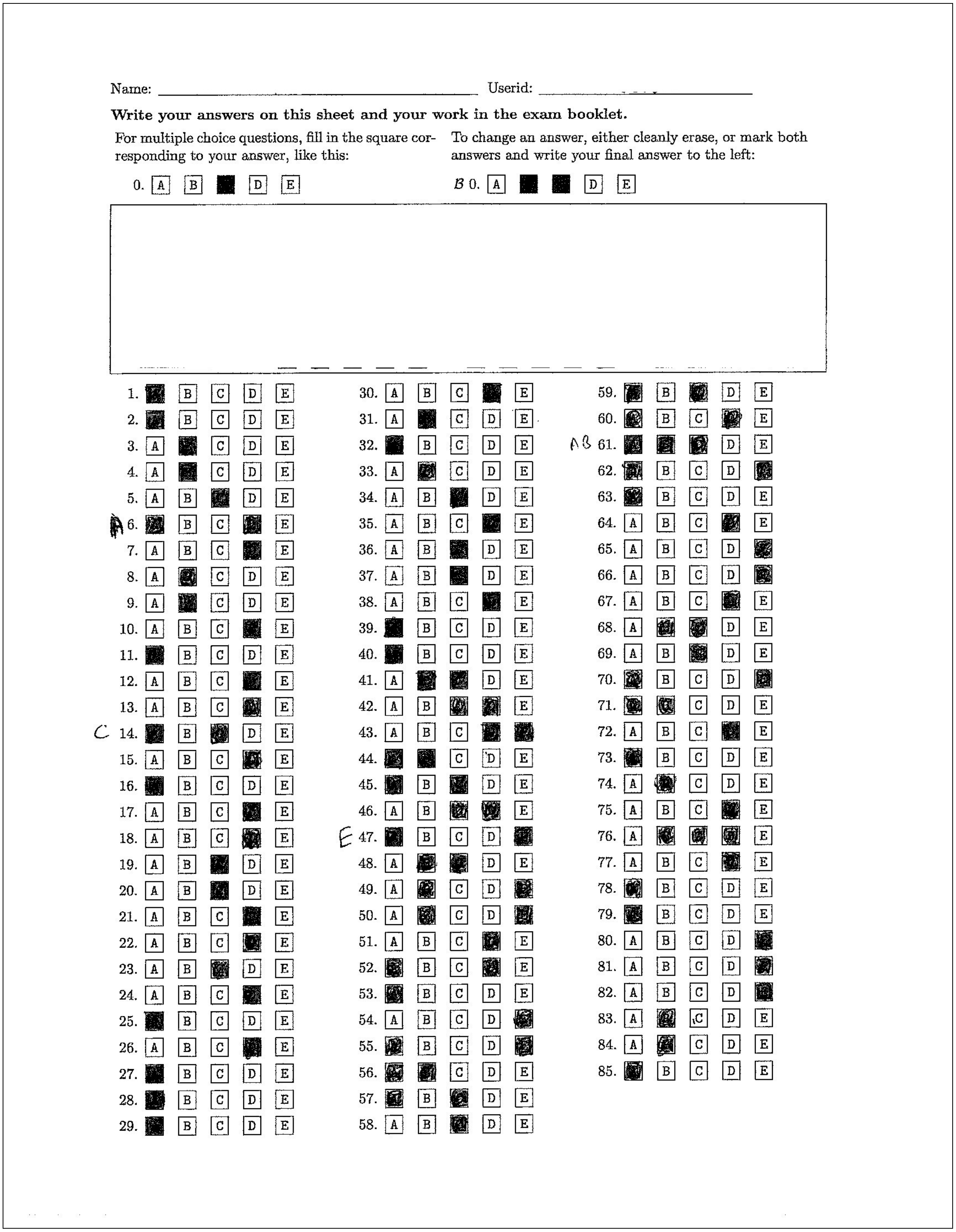
   ```
   ./grade.py form.jpg  output.jpg output.txt

   Recognizing form.jpg...
   ```

   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 (third 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). 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
   ...
   ```

   It should also create an output image (the second parameter on the command line above) that is the same as the input image, but with the location of the recognized marks displayed in some way (e.g., with a green rectangle outlining any letter that the system thinks was marked by the student).



Name: ______________________ Userid: ______________________

Write your answers on this sheet and your work in the exam booklet.

For multiple choice questions, fill in the square corresponding to your answer, like this:

To change an answer, either cleanly erase, or mark both answers and write your final answer to the left:

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 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:

   ```
   ./inject.py form.jpg answers.txt injected.jpg
   ./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.

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, i.e., for question 3 you can assume all the answer sheets were printed identically.

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. You can write a simple python program to count the number of lines that two files have in common, or you can use a Linux command like:

```
diff -y --suppress-common-lines output.txt groundtruth.txt | wc -l
```

which will report the number of lines that do *not* match (i.e., that are errors), from which you can easily calculate the accuracy of your program (i.e., percentage of answers that it recognizes correctly).

Your program will almost certainly not work perfectly all the time, 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 either as a PDF or 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 very 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 a bit like an argument for why you deserve a good grade on the assignment.

What to turn in

You should submit:

1. Your source code.
2. Your report, either as a PDF or 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.