COMS W4735y: Visual Interfaces to Computers
Spring, 2023
Assignment 3: Due April 6, 2023

0 Overview

0.1 Goal

The goal of this assignment is to explore how well a visual image can be described in human language terms. You are to write a program that takes as input a two-dimensional image of the Columbia campus, and then produces as output for each "target" building a non-numeric description of this objects (the "what"), a non-numeric description of its location (the "where"), and a non-numeric relationship to a "source" building (the "how"). The assignment also asks you to evaluate the effectiveness of these natural language outputs.

For example, your system could print out for one of the buildings: "This medium C-shaped rightmost building is near to the building which is large, square, and centrally located".

Note that this is a description of Kent with respect to Low, but neither "Kent" nor "Low" have been mentioned. This is because a user—especially a visitor—probably doesn't know any symbolic place names. Instead, the descriptions of the buildings come in three major parts: the "what" ("medium C-shaped" / "large"), the "where" ("rightmost" / "centrally located") and the "how" ("near").

Your job is to write a system that uses computer vision tecniques plus a primitive form of filtering and inference. First, it describes the buildings' shapes in a minimally possible way. Then it describes their spatial locations in a minimally possible way. Then it describes their relationships to each other in the least confusing way.

To help structure the assignment, it is broken down into five steps, with the full assignment worth the 20 points toward your final course score. As a general rule, whatever you do in code or in write-up, style counts. It is your obligation to write it all up so that the instructor and/or the TA can understand it on the very first try.

0.2 Files

The required images and other aids to the construction of the program are found on the class web pages, in standard portable formats (.pgm, .txt). The following three data files were created for the assignment:

1. The first file "Campus.pgm" is a binary image of the main campus as seen from above, where a large number within the image represents a part of a named building, and zeros represent the space between buildings.

2. The second file "Labeled.pgm" is an integer-valued image based on the first, in which each building is given an encoded integer, and all the pixels belonging to the same building are

encoded with the same integer; zero still means empty space. Therefore, in order to get the what/where/how features for Building N, one only has to scan the image for the occurrences of the encoded integer N.

3. The third file "Table.txt" translates the building's encoded integer into a semantically meaningful string. Some of your assignment's answers will require these strings. There are only 26 buildings in all, since some of them (like the Chemistry buildings, or the southeastern dorms) have been combined into one non-compact building. This will sometimes create challenges..

1 Step 1 (3 points): Raw data

1.1 Computation

This step is preliminary to the following steps, and uses standard computer vision computations, some of which you saw in Ass. 1.

1.2 Deliverables

For this step, have your system print out, *for each building*, the following table of building raw image properties:

Raw data:

1. Building number and name.
2. Building (x,y) coordinates of its center of mass, and total area in pixels.
3. Building Minimum Bounding Rectangle (MBR): (x,y) coordinates of upper left and lower right MBR corners, and length of MBR diagonal.
4. Any other buildings whose MBRs intersect this building's MBR.

You also need to submit a well-documented code listing that justifies *in comments* the algorithms and the thresholds that you used.

2 Step 2 (7 points): "What": describing shape

2.1 Computation

This step describes each building in terms of a vocabulary of shapes that are intrinsic to the building, without any reference to its location. These can include sizes, geometric shapes, and other identifying adjectives such as extrema. For example, people might use "tiny", "gigantic" as size descriptions, with maybe "longest" as an extremum. And people could use "triangular", "tubby", "lumpy", or "fractal" as geometric descriptions. As an example, Low Library can be described as "sizable highly-symmetric squarish".

Note that anything you produce as a description must be automatically computed from the given data. For example, sizes must be computed based on a measurement, not by using the numbers in the hard-coded Table.txt. This means that, except for the extrema, more than one building may end up with the same adjectives. So, there could be a number of "I-shaped" buildings, a number of "small" ones, etc. You should design, document, code, and test this by having a method or function for each shape description that evaluates a shape, and returns a boolean value for that presence of that shape.

These methods *can use* magic numbers within them, as long as they are visually or geometrically determined.. For example, for "small", you can compare areas or diagonal lengths against some constants, as long as those magic numbers are justified with a comment that explains how or why you set them at the values you did, given the visual context of this fixed picture. But it is not fair to *hard code* your descriptions in a manually-generated table that uses building identities. For example, you can't encode a shape description by saying "isCentral(buildingNumber)" means "return (buildingNumber == 113)". You must generate a descriptor based solely on visual and geometric properties, not on semantic symbols.like building names or building numbers.

You should implement the following descriptions.

1. Size: smallest / small / medium-size / large / largest
2. Aspect ratio: narrow / medium-width / wide
3. Geometry: square / rectangular / I-shaped / C-shaped / L-shaped / asymmetric

The algorithms are up to you do determine, and likewise any thresholds. Note that there are therefore 5 x 3 x 6 possible different descriptions for a building, but that each one is a triple of small integers Note also that all of the buildings already turn out to be aligned with the horizontal and vertical axes of the image (the campus is very Manhattan!), which makes some of the computations very easy.

The more complicated shapes ("I", "C", "L") can be done similarly to the "folding" done in the symmetry computation of Ass. 2. They are left up to you to intuit, since some of this is not exact. (Is Journalism&Furnald L-shaped? What about Chandler&Havemeyer? Is Ham&Hart&Wall&Jay C-shaped? What about Schermerhorn&Fairchild?)

One way to approach this:

- An L-shaped building can be folded on its diagonal so that its legs fully or mostly overlap, resulting in a rectangular building
- A C-shaped building can be folded top to bottom, so the upper and lower arms of the "C" overlap; resulting in an L-shaped building
- An I-shaped building can folded on its major axis, resulting in a C-shaped building.

2.2 Deliverables

For this step, have your system print out, *for each building*, the following table of building "what" properties:

Shape Description:

1. Geometry: The three geometric "what" features, in the order size, aspect ratio, shape.

2. Confusion: A list of which other buildings have the same three "what" features

3. Minimization: A shorter "what" description, if any, that would introduce no more confusion, strictly on the basis of its "what". For example, AlmaMater could be minimized to "smallest square", or even just "smallest".

You also need to submit a well-documented code listing that justifies *in comments* the algorithms and the thresholds that you used.

3 Step 3 (4 points): "Where": describing *absolute* space

3.1 Computation

Now, augment your descriptions of the buildings with information that assumes an external coordinate system and the bounded universe of the image.

You should implement the following descriptions, with respect to the absolute framework:

1. Verticality: uppermost / upper / mid-height / lower / lowermost

2. Horizontality: leftmost / left / mid-width / right / rightmost

3. Orientation: vertically-oriented / non-oriented / horizontally-oriented

Again, the algorithms are up to you to determine, and likewise any thresholds. Note that there are therefore 5 x 5 x 3 possible different descriptions for a building's absolute location, but that each one is a triple of small integers

3.2 Deliverables

For this step, have your system print out, *for each building*, the following table of building "where" properties:

Absolute location description:

1. Geometry: The three geometric "where" features, in order verticality, horizontality, orientation.

2. Confusion: A lsit of which other buildings have the same three "where" features

3. Minimization: A shorter "where" description, if any, that would introduce no more confusion, strictly on the basis of its "where". For example, NorthwestCorner could be minimized to "vertically-oriented uppermost".

You also need to submit a well-documented code listing that justifies *in comments* the algorithms and the thresholds that you used.

4 Step 4 (4 points): "How": describing *relative* space

4.1 Computation

Now, design and code the boolean-valued function for building pair: nearTo(S, T). This is read, "Near to S is T". It takes the raw data of the source building S and the raw data of the target building T, and returns true if they are in the "near" relationship. Note that nearTo *must* be affected by the dimensions of the S building. For example, it should be harder to be near to AlmaMater than to be near to Low.

To do this, write an algorithm to create this binary spatial "how" relationship for *every* building pair, according to your definition of nearTo. This part should generate 26 x 26 relationships, as every building pair either does or does not have a "near" relationship established by the source for the target. You can print these out if you wish to check them, but most of them should be false. Also, note that like the ground truth in Ass. 2, this relationship need not be symmetric: nearTo(S, T) may not automatically imply nearTo(T, S), depending on how you define it.

You should implement the following description, with respect to the relative frameowrk:

1. Relativity: near

Again, the algorithms are up to you to determine, and likewise any thresholds. Note that there are therefore 25 possible different descriptions for a building's relative location but they can be represented as a short list of small integers.

4.2 Deliverables

Even though your internal data strutures continue to be based on integer indicies, For this output you should use building *names* for this output rather than numbers, as it makes it easier to understand.

Relative location description:

1. Nearness: *For each building* as a source S, list all target buildings T that are nearTo(S, T). Conversely, *for each building* as a target T, list all source bulidings S that are nearo(S, T).

2. Confusion: Find which source S is nearTo(S, T) for the most targets T, then find the S for the least targets T. Conversely, find which target T is nearTo(S, T) for the most sources S, then find the T for the least.sources S.

3. Minimization: *For each building* as a target, indicate which of its possible sources is the source most often used as a source for other buildings also. (These are "landmark" sources.).

You also need to submit a well-documented code listing that justifies *in comments* the algorithms and the thresholds that you used.

5 Step 5 (2 points): Total descriptions

5.1 Computation

At this point, for each building you can form from the previous steps an internal representation consisting of four parts like the following (since the "near" is implicit):

what$_T$ where$_T$ what$_S$ where$_S$

You can now express it in natural language by using a simple fill-in-the-blanks template, like the example given at the start of the assignment. But even though this sentence is composed of separately optimized phrases, more minimization of the full sentence may be possible, as long as again it does not introduce any additional confusion.

The above expression can have as many as 12 words (3 for what and 3 for where, for both source and target), and as few as 2. For example "The small building near the largest building" could describe OldComputerCenter with respect to Uris..

So, using further filtering, compact these sentences further.

5.2 Deliverables

For this step, have your system print out, the following table:

Total description:

1. Total description: *For each building*, using a natural language sentence.
2. Confusion: Find any total descriptions that are still confused.
3. Minimization: Which is the shortest unambiguous description? Which is the longest?

You also need to submit a well-documented code listing that justifies *in comments* the algorithms and the thresholds you used.

6 Checklist of deliverables

Your assignment consists of a write-up with examples, then a listing of all the code used. Any code that you did not write yourself has to be documented with a statement about its source and an explanation of why you have permission to use it.

For all steps, your write-up explains: what design choices you made and why, what algorithms you used, what you observed in the output, and how well you believe your design worked.

Here is a short summary of what is expected:

1. For step 1, you must show and justify your computer vision processing of buildings into their raw properties.
2. For step 2, you must show and justify your computer vision processing of buildings into their shape descriptions, and their confusions and minimizations..
3. For step 3, you must show and justify your computer vision processing of buildings into their absolute location descriptions, and their confusions and minimizations..

4. For step 4, you must show and justify your computer vision processing of buildings into their relative location descriptions, and their confusions and minimizations..

5. For step 5, you must show and justify your compaction of the total descriptions, and their confusions and minimizations..