CS 470: Assignment 04

Programming Assignments (95%)

The goal of this assignment is to write code to **extract LBP features from a set of images**.

A04.py

You must provide code for the following functions in the file:

- **getOneLBPLabel(subimage, label_type)**
- **getLBPImage(image, label_type)**
- **getOneRegionLBPFeatures(subImage, label_type)**
- **getLBPFeatures(featureImage, regionSideCnt, label_type)**

You MUST implement the LBP algorithm yourself; however, you may use OpenCV and/or Numpy histogram functionality.

If you are not sure whether a library function is permitted, ASK ME!

getOneLBPLabel(subimage, label_type)

Given a (grayscale, uint8) 3x3 subimage (basically the neighborhood of a single pixel), return the correct LBP label for that pixel. **Please note that, for thresholding, value > center to be 1 (otherwise 0).**

In General_A04, there is an Enum, LBP_LABEL_TYPES:

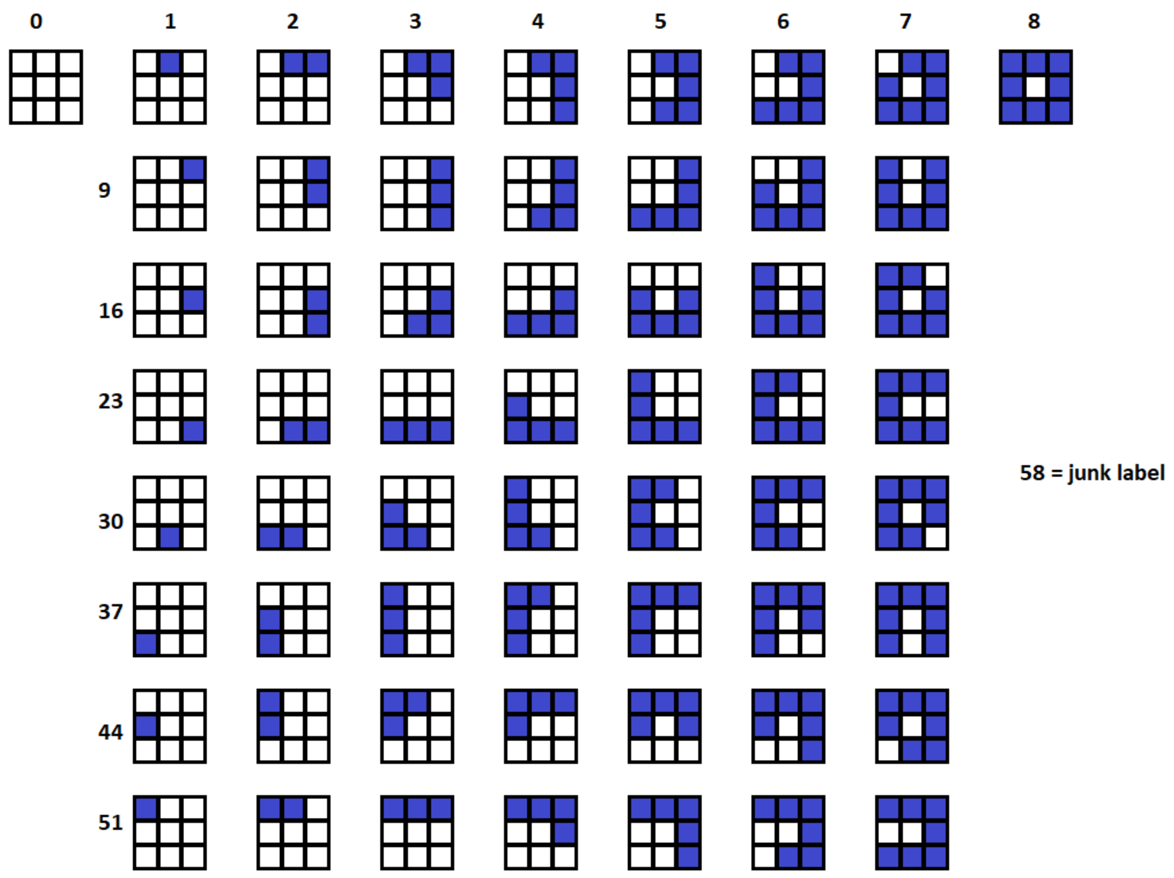
- **UNIFORM** = the uniform, rotation-invariant version of LBP (10 labels total)
- **FULL** = the full binary number version of LBP (256 labels total)
- **UNIFORM_ROT** = uniform LBP, BUT rotation-sensitive (59 labels total)

You must implement all of these approaches.

For FULL and UNIFORM_ROT, start in the UPPER-LEFT of the subimage and go CLOCKWISE around the center.

For FULL, the lower-order bits will be encountered FIRST (e.g., n[0] = 2^0, n[1] = 2^1, etc.)

For UNIFORM_ROT, the first 9 labels (0-8) are effectively just the sum of the thresholded neighbors. However, after that, you need to determine the position right before the first swap from 0 to 1. For the first row in the image below, first_swap would be 0. In the next row, it would be 1, and so on.



Thus, your algorithm for UNIFORM_ROT should:

1. FIRST check if the pattern is uniform at all. If it isn't, use label 58 (junk label).
2. Find the neighbor position right before the first (and only if it's uniform) swap from 0 to 1.
3. If the sum of the thresholded neighbors is 0 or 8, OR if the first_swap is at index 0, then the label is just equal to the sum of the thresholded neighbors (0-8, the top row above).
4. OTHERWISE, the label is given by: **first_swap * 7 + 1 + sum**

getLBPImage(image, label_type)

Given a (grayscale, uint8) image, generate and return the appropriate LBP label image.

- Radius will be 1 and the sample count is 8, so you may directly use the 8 pixel neighbors (strong and weak) for each pixel.
- The output label image will be the same size and type as the input image.
- If you extend out of bounds, assume zero padding. You may use cv2.copyMakeBorder (with a padding of 1 on all sides).
- Loop through each pixel in the image, cut out the appropriate subimage, and call getOneLBPLabel to get the correct output label per pixel.

getOneRegionLBPFeatures(subImage, label_type)

Given an **LBP label image** (NOT the original image!), compute and return the correct LBP histogram.

Do NOT forget to normalize the histogram by dividing by the total number of pixels!

Make sure that the histogram is the correct length:

- **UNIFORM** = 10 elements
- **FULL** = 256 elements
- **UNIFORM_ROT** = 59 elements

getLBPFeatures(featureImage, regionSideCnt, label_type)

Given an **LBP label image** (NOT the original image!) and the number of subregions on each side of the image, you will need to do the following:

- Compute the subregion width and height in pixels. **Use either integer division or floor(). This means that your subregions may not cover the entire image; that is acceptable.**
- Start with an empty list to hold all of the individual histograms.
- Loop through each possible subregion, ***going row by row and then column by column***.
    - Extract the subimage (using the precomputed subregion width and height as well as the starting point). **Subregions do NOT overlap!**
    - Call getOneRegionLBPFeatures() to get the one subimage's histogram.
    - Append this histogram to your list of all histograms.
- Convert your list of histograms to an np.array() and then reshape so that it is a flat array:
    - allHists = np.array(allHists)
    - allHists = np.reshape(allHists, (allHists.shape[0]*allHists.shape[1],))
- Return allHists

Testing Screenshot (5%)

I have provided several files:

- **Test_A04.py** – the test program
- **General_A04**.py – code shared by the test programs; you should also import this in A04.py using: *from General_A04 import **
- **assign04/**
    - **images/** – folder containing the input images
    - **ground/** – folder containing the ground truth LBP label images and the csv files

Once you have merged from my repo into yours, these files/folders should reside in your main project directory. Your source file (A04.py) should reside in the main project directory as well (NOT the assign04 folder).

You can either run the testing programs directly OR you can use the testing section of Visual Code.

You MUST run the tests and send a screenshot of the test results! Even if your program(s) do not pass all the tests, you MUST send this screenshot!

You may have to do “Command Palette” → “Python: Configure Tests” → unitttest → root directory → "test_*.py", and then modify .vscode/settings.json to use "Test_*.py

This screenshot should show clearly:

- The final result of the test run on the command line ("OK" for all passing, "FAILED (failures=N)" for some or all failing).
- OR
- The testing view in Visual Code

I'm really looking for the checkboxes (or failed overall markers) for Test_A04. I don't need to see ALL of the tests.

The screenshot should be copied to the screenshots/ folder.

Grading

Your OVERALL assignment grade is weighted as follows:

- 95% - Programming assignments
- 5% - Testing screenshot