CS 426 – Parallel Computing

Fall 2017

Project 3

Due: December 7th, 2017 at 11:59 PM

In this project, you will implement a parallel face recognition algorithm using idea of Local Binary Patterns.

Local Binary Patterns is a simple but effective idea in texture analysis. It is also used for face recognition and gesture recognition applications.

LBP idea & implementation:

You can find the data set in the following link. It is highly recommended to read the Readme.txt file first.

https://drive.google.com/file/d/0B3nmpRSZ28SXbkhweFQ3ajNncXM/view?usp=sharing

This link also have additional util.c and util.h files, they are explained further below. If you can't reach this link, contact me ASAP.

The LBP approach uses two separate subsets of a given dataset. The first subset will be training set whereas the second one will be test set.

- There are 18 different persons' pictures in our dataset with 20 pictures for each person that are taken under different light condition and with different gestures.
- First k pictures of each person will be training images and the remaining 20-k pictures of each person will be used as test images.
  - Pictures are named as the following format : person_id.photo_id.txt
  - Each file contains 2D matrices.
  - Ex for k=10:
    - training set for person 1: 1.1.txt, 1.2.txt, 1.3.txt, … , 1.10.txt
    - test set for person 1: 1.11.txt, 1.12.txt, 1.13.txt, … , 1.20.txt
- All pictures in the dataset are gray scale images. Each pixel value will be an integer between 0 and 255.
- Each image is composed of 180x200 pixels.

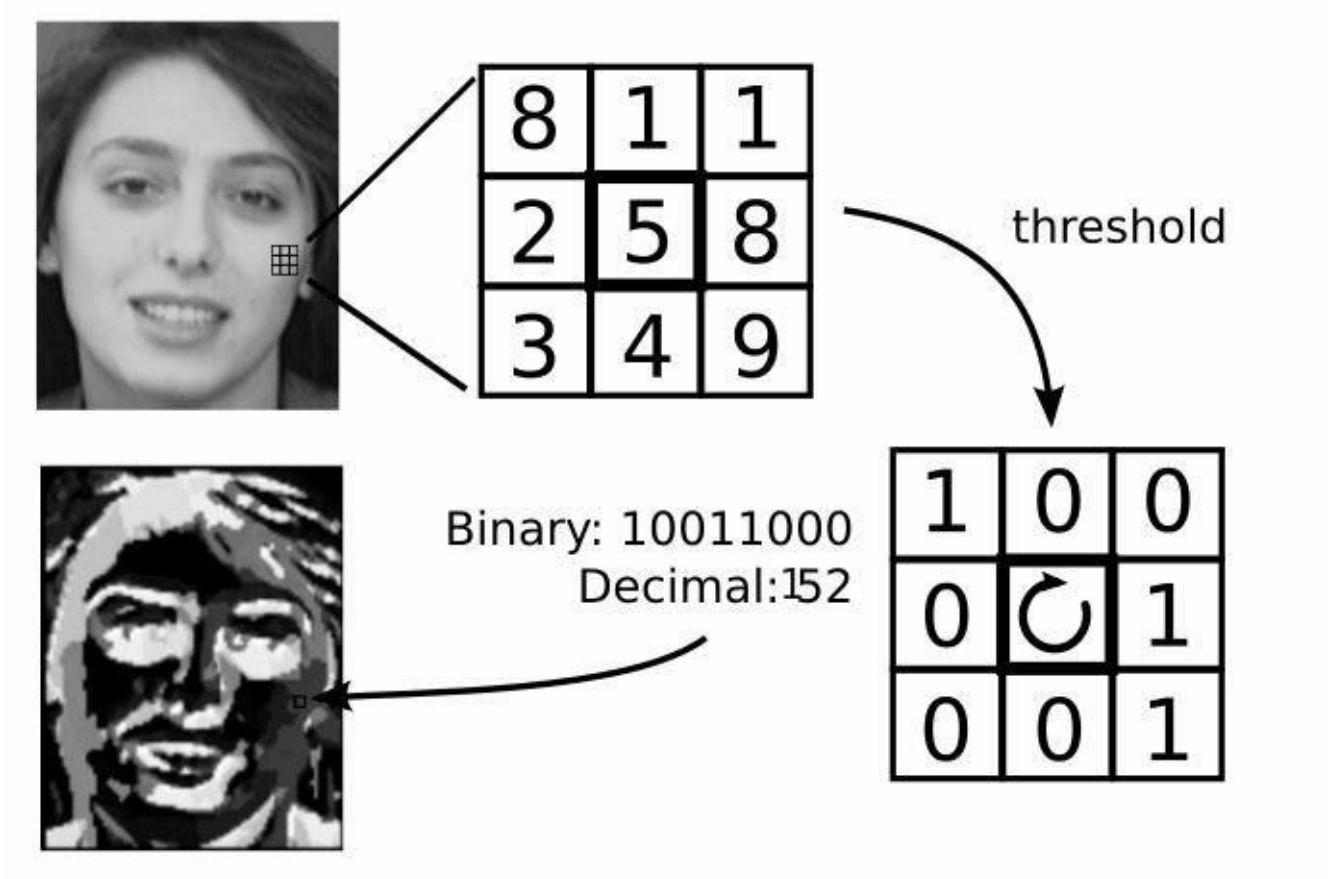
Following algorithm is adapted from the algorithm given in Wikipedia:

http://en.0wikipedia.org/wiki/Local_binary_patterns

Training Step:

For each image in the dataset, you will apply the following steps:

- For each pixel in an image, compare the pixel with its 8 neighbors (on its left-top, left-middle, left-bottom, right-top, etc.). Follow the pixels along a circle, i.e. clockwise.
- Where the center pixel's value is less than the neighbor's value, write "1". Otherwise, write "0". This gives an 8-digit binary number.
- Convert this value into a decimal value.
- Compute the histogram, over the pixels, of the frequency of each "number" occurring (i.e., each combination of which pixels are smaller and which are greater than the center).
  - You can read about histograms from the following link:
    - http://en.0wikipedia.org/wiki/Histogram



- As a result you will have 18x10=180 histograms, using the above example k=10.

Test Step:

In test you will follow the following steps:

- Create a histogram for the test image as described previously.
- Find the distance values between test image's histogram and training histograms.
- Select the closest training histogram to determine the person.
- In this homework you will use the following distance function
  - a, b are two vectors of size d.
  - distance(a,b) = $\sum_{i=0}^{d} \frac{1}{2} \frac{(a_i - b_i)^2}{a_i + b_i}$
  - If $a_i + b_i = 0$, you can assume $\frac{1}{2} \frac{(a_i - b_i)^2}{a_i + b_i}$ to be 0 as well

Implementation Details

- First, you will implement the sequential version of LBP Face Recognition
  - 3 functions will be implemented
    - void create_histogram(int * hist, int ** img, int num_rows, int num_cols)
      - Creates a histogram for image given by int **img and returns histogram as int * hist
    - double distance(int * a, int *b, int size)
      - Finds the distance between two vectors
    - int find_closest(int ***training_set, int num_persons, int num_training, int size, int * test_image)
      - Finds the closest histogram for test image's histogram from training set histograms
      - Returns person id of the closest histogram
- You will profile your code using gprof and try to find the functions that are needed to be parallelized.
  - For gprof details, there are many online tutorials.
- Second, you will implement parallel version of LBP Face Recognition
  - Basically, you will insert OpenMP pragmas to parallelize your code.
- You will profile your code again using gprof.
- You can download and use the util.h and util.c files, they are in the link provided at the beginning of the file. In these files, 2D array allocate and free functions, file reading functions are implemented.

- Your program will take a single command line input k which will show the number of images that will be used as training images for each person.

Output Format

You will print test results of your program in the following format.

- file_name test_result_person_id correct_person_id
- You will also print parallel execution time and sequential execution time as in previous homework.
- You will also print out the number of correct answers.
- Ex:

1.11.txt 1 1

1.12.txt 1 1

1.13.txt 1 1

2.11.txt 2 2

2.12.txt 3 2

Accuracy: T correct answers for Z tests

Parallel time: XX.XX ms

Sequential time: YY.YY ms

Notice that we are still using k=10 example. So for every person, first 10 images make up training data. Then we take a test image, let's say 2.12.txt. We will create histogram for this image file. Than we will compare it with the training histograms of all people. Hopefully, result of the comparison will tell us that this image belongs to person 2. Closest histogram to 2.12.txt is expected to come from images between 2.1.txt and 2.10.txt. You are going to output both the correct result and your own prediction.

Submission

- Please put everything under same directory, do not structure your project under directories like parallel, sequential etc. Put all of them in the same directory, one called name_surname_bilkentid_p3. You will zip this directory and send it to me. When I unzip it, I should get that directory and files inside.
- Your code:
  - lbp_seq.c, lbp_omp.c file and any other file that you have implemented
    - If you used given files util.h util.c, also include them in your submission
  - lbp_seq.c will include your sequential implementation and lbp_omp.c will include your parallel implementation.
  - A compile script that can compile your code
    - Name this script as compile.sh
    - It will produce 2 executables, lbp_seq and lbp_omp

- An example run script run.sh
  - That will run your code and print the output of your code to a file named as surname_name.output (not surname_name.output.txt) for all specifications below.
    This script will run your code with several configurations
    - You should run your sequential code with various k numbers
      - k=1, k=2, k=5, k=7, k=10
    - You should include runs with different number of threads
      - num_of_threads=1, num_of_threads=2, num_of_threads=4, num_of_threads=6, num_of_threads=8, num_of_threads=16
  - Your code also should be run with the following commands:
    export OMP_NUM_THREADS= num_of_threads && ./lbp_omp k
    ./lbp_seq k

- Profiling results
  - You will submit gprof profiling results
    - prof_sequential.txt file for sequential implementation's profiling results
    - prof_omp.txt file for parallel implementation's profiling results
      - It will include profiling results for several runs with different number of threads
- Your report
  - Reports should be in .pdf format.
    - Submissions with wrong format will get 0.
  - Detailed description of gprof's profiling outputs.
  - Detailed description of your implementation details
    - Explain pragmas that you have used
      - Why did you insert this pragma to this specific region?
      - What does this pragma do?
      - What are the possible options that can be used with this pragma?
      - What are the options that you have used?
  - Plot for accuracy results.
  - Plot for execution times with different threads.
  - Discussion of your results
    - Don't forget to use gprof's profiling outputs in your discussion.
- Email to: kaan.akyol@bilkent.edu.tr
  - Email subject: CS426_HW3
  - You should submit a single .zip file named as name_surname_bilkentid_p3.zip

Hints

In our implementation, we had the following accuracy results:

- If k=1
  - 16 errors out of 342 test images.
- If k=10
  - 0 error out of 180 test images.