|  |
| --- |
| Task/assessment brief: |
| * This task aims to use **Parallel Computation** on algorithms of **Facial Recognition** and **Object Recognition** that rely on **Image Classification**. * The first part of this task aims to identify the inside techniques used by classification algorithms on pixels while relying on **Parallel Processing**, whereas the second part aims the use of embedded algorithms of classification while relying on **Multithreading** and **OpenCV** libraries.   **Part 1) 40%**  You are required to implement an algorithm to compare two images while relying on multiple methods of parallel processing such as **Multithreading** along using **MPI**. Then, forwarding this algorithm to compare between one image and a dataset of images in part 2.   1. Create the code of a simple algorithm (Algorithm 1) to calculate the difference between the values of two pixels ***x*** and ***y***, where these pixels have integer values. The difference between these two pixels should be expressed as. 2. Create the code of a second algorithm (Algorithm 2) to calculate the sum of differences between the embedded pixels within two arrays (**X[ ]** and **Y[ ]**), where each array is having one dimension (1D). Each pixel from the array **X[ ]** is compared to an equivalent pixel from the array **Y[ ]** by calling the first algorithm (Algorithm 1) while using **Multithreads** and other techniques for distributed computation (**MPI, OpenMPI, etc**.) whereas the sum is a global variable accessed by all parallel processes. The sum of differences has a final value expressed as follows:   .   1. Create the code of a third algorithm (Algorithm 3) to calculate the sum of differences between the embedded pixels within two arrays (**X[ ][ ]** and **Y[ ][ ]**), where each array is having two dimensions (2D). Each pixel from the array **X[ ][ ]** is compared to an equivalent pixel from the array **Y[ ][ ]**. Therefore, compare each line of pixels from the array **X[ ][ ]** with an equivalent line of pixels from the array **Y[ ][ ]** by calling the Algorithm 2 in form of Multithreads or other technique of parallel computation whereas the sum is a global variable accessed by all parallel processes. The sum of differences now has a final value calculated by **Euclidian Distance Formula,** and it is expressed as follows:      1. Create a code for Algorithm 4 to open two images and return them into two arrays of type 2D, and then use the Algorithm 3 on them (You can use **OpenCV**). 2. Add code lines to return a **Percentage Distance Value** by using the given result in previous expression of **Ed2**. The **Percentage Distance Value** is based on the formula where **N** and **M** are the sizes of width and height of arrays respectively. The resulted code at this stage should be called as Algorithm 5 and it should return the value of calculated percentage.     **Part 2) 30%**  After familiarizing the use of **Parallel Computation** on images for comparison by using **Euclidian Distance** between them, you are requested to build on the used logic in previous part of this assessment to compare one image with a dataset of images (multiple images within a folder).  By using the developed algorithm in previous part on a dataset of multiple images, you are concretizing the bases for a **K-Nearest Neighbour** (KNN) algorithm from scratch while deploying the principles of **Parallel Computation** at multiple levels.  Following tasks should be completed during the process.   1. Identify a suitable dataset. You may use resources such as Kaggle, MNIST, or CIFAR websites for this purpose. Helpful links will be provided in Moodle. You should discuss the suitability of the dataset with the teaching team before using it for this task. Dataset should have sufficient number of images to split it into training and testing images. **It is recommended to choose a dataset of images with the same extensions of type, the same sizes of height and width and the same context (Facial Recognition, Object Recognition, Shape recognition, etc**.). 2. Investigate the methods that can be used to read and visualize the data using C++. You may have to use third party libraries such as OpenCV or follow customized binary data reading mechanisms that are provided in the documentation of the dataset. 3. Use appropriate data structures, variables and labels within the program. 4. Create a code to call the previous Algorithm 5 from part 1, and execute this algorithm to compare one image (X) with all images from the dataset in parallel by using **Multithreading** and other Distribution techniques **(MPI, OpenMPI, etc)**. You need to add code lines to compare between returned values of distance **(Percentage Distance Values)** from parallel processes in order to identify the minimum value among them, which will allow you to distinguish the closest training image from the dataset to the tested image X. You can add a formula to display the **Confidence of Result** in order to highlight the performance of algorithms. There are many formulas to highlight confidence of result, but you can use the following one:   : The minimum value among returned values of Percentage Distance.  : The sum of all returned values of Percentage Distance.  : How many elements (images) the algorithm has trained-on during the comparison between the testing image X and the dataset.  **Part 3): 20%**  You should generate a report of 2000 words composed of your codes (shank by shank) along with comments on each library, variable, instruction, condition statement, loop, function and returned value. You should highlight in your report each level where you deploy parallel computation and what is your chosen techniques for that along with used techniques for synchronization. The report should be well formatted and organized, and it should be uploaded on the dedicated Moodle submission point.  **Part 4): 10%**  A lab demonstration on each developed code for specified tasks is required. In addition, a screen recorded video demonstrating the full functionality of the part 1) and part 2) should be uploaded to the dedicated Moodle submission point. The recording should demonstrate the compilation steps, execution, and outputs of the program. |