|  |
| --- |
| package org.hipi.tools.covar; |
|  |  |
|  | import static org.bytedeco.javacpp.opencv\_imgproc.CV\_RGB2GRAY; |
|  |  |
|  | import org.hipi.image.HipiImageHeader.HipiColorSpace; |
|  | //import opencv2\_cookbook.OpenCVUtils.\*; |
|  | import org.bytedeco.javacpp.helper.opencv\_core.\*; |
|  | import org.hipi.opencv.OpenCVUtils; |
|  | //import org.opencv.core.Core; |
|  | import org.bytedeco.javacpp.opencv\_imgproc.\*; |
|  |  |
|  | import java.lang.Object; |
|  | import org.bytedeco.javacpp.Pointer; |
|  | import org.bytedeco.javacpp.opencv\_core.CvType; |
|  |  |
|  | import org.hipi.image.FloatImage;  import org.hipi.image.filter;  import org.hipi.image.threshold; |
|  | import org.hipi.image.HipiImageHeader; |
|  | import org.hipi.imagebundle.mapreduce.HibInputFormat; |
|  |  |
|  | import org.apache.hadoop.conf.Configured; |
|  | import org.apache.hadoop.util.Tool; |
|  | import org.apache.hadoop.util.ToolRunner; |
|  | import org.apache.hadoop.fs.Path; |
|  | import org.apache.hadoop.io.IntWritable; |
|  | import org.apache.hadoop.io.Text; |
|  | import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; |
|  | import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; |
|  | import org.apache.hadoop.mapreduce.Job; |
|  | import org.apache.hadoop.mapreduce.Mapper; |
|  | import org.bytedeco.javacpp.opencv\_imgproc.\*; |
|  |  |
|  | import org.bytedeco.javacpp.opencv\_highgui.\*; |
|  |  |
|  | import org.bytedeco.javacpp.opencv\_core.Mat; |
|  |  |
|  | import org.apache.hadoop.mapreduce.Reducer; |
|  | import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; |
|  | import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; |
|  | /////////////////////////////////////////////////////////////// |
|  | import org.bytedeco.javacpp.opencv\_core.\*; |
|  |  |
|  | import org.bytedeco.javacpp.opencv\_core.Rect; |
|  | import org.bytedeco.javacpp.opencv\_core.Scalar; |
|  | import org.apache.hadoop.conf.Configuration; |
|  | import java.io.File; |
|  | import javax.imageio.ImageIO; |
|  | import java.awt.image.BufferedImage; |
|  | import java.util.Random; |
|  | import javax.swing.JFrame; |
|  | import java.io.IOException; |
|  | import java.awt.Component; |
|  | import java.io.BufferedReader; |
|  | import java.io.CharArrayReader; |
|  | import java.lang.reflect.Field; |
|  | import java.util.Arrays; |
|  | import static java.lang.System.out; |
|  | import java.lang.reflect.Array; |
|  | import static java.lang.System.out; |
|  | import java.lang.reflect.Constructor; |
|  | import java.lang.reflect.Type; |
|  | import java.util.ArrayList; |
|  | import java.util.Arrays; |
|  | import java.util.List; |
|  |  |
|  | import org.bytedeco.javacpp.\*; |
|  | import static org.bytedeco.javacpp.opencv\_core.\*; |
|  | import static org.bytedeco.javacpp.opencv\_imgcodecs.\*; |
|  | import static org.bytedeco.javacpp.opencv\_stitching.\*; |
|  | //////////////////////////////////////////////////////////////// |
|  |  |
|  |  |
|  | import java.io.IOException; |
|  |  |
|  | public class proj extends Configured implements Tool { |
|  |  |
|  | public static class projMapper extends Mapper<HipiImageHeader, FloatImage, IntWritable, IntWritable> { |
|  |  |
|  | public void map(HipiImageHeader key, FloatImage value, Context context) |
|  | throws IOException, InterruptedException { |
|  |  |
|  | // Verify that image was properly decoded, is of sufficient size, and has three color channels (RGB) |
|  | if (value != null && value.getWidth() > 1 && value.getHeight() > 1 && value.getNumBands() == 3) { |
|  |  |
|  | // Get dimensions of image |
|  | int w = value.getWidth(); |
|  | int h = value.getHeight(); |
|  |  |
|  |  |
|  | ///////////////////////////////////////////////////////WORK AREA STARTS |
|  | FloatImage image1; |
|  | // image1 = value; |
|  |  |
|  | Mat cvImage = new Mat(value.getHeight(), value.getWidth(), opencv\_core.CV\_32FC1); |
|  |  |
|  | Mat cvImage1 = new Mat(value.getHeight(), value.getWidth(), opencv\_core.CV\_8UC1); |
|  | Mat cvImage2=new Mat(); |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | Mat cvImage3 = new Mat(); |
|  | Covariance.convertFloatImageToGrayscaleMat(value, cvImage); |
|  |  |
|  | cvImage.convertTo(cvImage3, opencv\_core.CV\_8UC1); |
|  |  |
|  |  |
|  |  |
|  |  |
|  | double nish = 6.75; //constant sigma value used to calculate MedianBlur and to filter up the noise |
|  |  |
|  | Mat img = new Mat(cvImage1); |
|  |  |
|  |  |
|  | Size size = new Size(3, 3); |
|  | opencv\_imgproc.MedianBlur(cvImage3, cvImage3, size, nish); |
|  | // opencv\_imgproc.MedianBlur(convertedTo8UC1, convertedTo8UC1, size, nish); |
|  |  |
|  | opencv\_imgproc.threshold(cvImage3, cvImage3, 0, 255, opencv\_imgproc.THRESH\_OTSU); |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | //dilation operation for extracting the background |
|  |  |
|  | // opencv\_imgproc.watershed(cvImage, cvImage1); |
|  |  |
|  | int avg = countNonZero(cvImage3); //VALUE |
|  | // int avg = countNonZero(convertedTo8UC1); |
|  |  |
|  | // imwrite("1234567.png",cvImage); |
|  |  |
|  |  |
|  | // Emit record to reducer |
|  | context.write(new IntWritable(1),new IntWritable (avg)); |
|  |  |
|  | } // If (value != null... |
|  |  |
|  | } // map() |
|  |  |
|  | } // projMapper |
|  |  |
|  | ////////////////REDUCER CRIGINAL COPY//////////////////////////// |
|  |  |
|  | public static class projReducer extends Reducer<IntWritable, IntWritable, IntWritable, Text> { |
|  |  |
|  | //Change all float to FloatImage to make it to previous version |
|  | public void reduce(IntWritable key, Iterable<IntWritable> values1, Context context) |
|  | throws IOException, InterruptedException { |
|  |  |
|  | // Create FloatImage object to hold final result |
|  | // FloatImage avg = new FloatImage(1, 1, 3); |
|  | double avg=0.0; //Remove |
|  |  |
|  | // ArrayList<Integer> valList = new ArrayList<Integer>(); |
|  | // Initialize a counter and iterate over IntWritable/int records from mapper |
|  | int sum = 0; |
|  | int total = 0; |
|  | for (IntWritable val : values1) { //float to FloatImage |
|  | sum += val.get(); |
|  | // valList.add(val.get()); |
|  | total++; |
|  | } |
|  |  |
|  | if (total > 0) { |
|  | avg = sum / (double)total; |
|  | String result = String.format("Average pixel value: %f ", avg); |
|  | // Emit output of job which will be written to HDFS |
|  | context.write(key, new Text(result)); |
|  | } |
|  |  |
|  | } // reduce() |
|  |  |
|  | } // projReducer |
|  |  |
|  | ////////////////////////REDUCER ORIGINAL COPY ENDS////////////////////////// |
|  |  |
|  |  |
|  | public int run(String[] args) throws Exception { |
|  | // Check input arguments |
|  | /\* if (args.length != 2) { |
|  | System.out.println("Usage: helloWorld <input HIB> <output directory>"); |
|  | System.exit(0); |
|  | }\*/ |
|  | Configuration conf = getConf(); |
|  |  |
|  | conf.set("mapreduce.input.fileinputformat.split.minsize", "656000"); |
|  | // Initialize and configure MapReduce job |
|  | Job job = Job.getInstance(conf,"Increase Split Size"); |
|  | // Set input format class which parses the input HIB and spawns map tasks |
|  | job.setInputFormatClass(HibInputFormat.class); |
|  | // Set the driver, mapper, and reducer classes which express the computation |
|  | job.setJarByClass(proj.class); |
|  | job.setMapperClass(projMapper.class); |
|  | job.setReducerClass(projReducer.class); |
|  | // Set the types for the key/value pairs passed to/from map and reduce layers |
|  | job.setMapOutputKeyClass(IntWritable.class); |
|  | job.setMapOutputValueClass(IntWritable.class); |
|  | job.setOutputKeyClass(IntWritable.class); |
|  | job.setOutputValueClass(Text.class); |
|  |  |
|  | // Set the input and output paths on the HDFS |
|  | FileInputFormat.setInputPaths(job, new Path(args[0])); |
|  | FileOutputFormat.setOutputPath(job, new Path(args[1])); |
|  |  |
|  | // Execute the MapReduce job and block until it complets |
|  | boolean success = job.waitForCompletion(true); |
|  |  |
|  | // Return success or failure |
|  | return success ? 0 : 1; |
|  | } |
|  |  |
|  | public static void main(String[] args) throws Exception { |
|  | ToolRunner.run(new proj(), args); |
|  | System.exit(0); |
|  | } |
|  |  |
|  | } |