The Journal of Supercomputing (2022) 78:4330–4357
https://doi.org/10.1007/s11227-021-04027-5

A parallel and accurate method for large-scale image segmentation on a cloud environment

Gangmin Park[1] · Yong Seok Heo[2] · Kisung Lee[3] · Hyuk-Yoon Kwon[4]

Accepted: 13 August 2021 / Published online: 30 August 2021

Abstract
In this paper, we present a parallel algorithm for SLIC on Apache Spark, which we call *PSLIC-on-Spark*. To this purpose, we have extended the original SLIC algorithm to use the operations in Apache Spark, supporting its parallel processing on multiple executors in the Apache Spark cluster. Then, we analyze the trade-off relationship of PSLIC-on-Spark between its processing speed and accuracy due to partitioning of the original image datasets. Through experiments, we verify the trade-off relationship. Specifically, we show that PSLIC-on-Spark using 8 CPU cores reduces the processing time of SLIC by 2.24–2.93 times while it reduces the boundary recall (BR) of SLIC by 1.54–6.32% and increases under-segmentation error (UE) by 1.79–6.2%. Then, we propose an improved algorithm of PSLIC-on-Spark that improves the accuracy of PSLIC-on-Spark, which we call *PASLIC-on-Spark*. We employ two important features for PASLIC-on-Spark. It contains two main features: (1) image partitioning considering the shape and position of the clusters rather than a evenly partitioning method and (2) controllable duplication for the boundary between image partitions. Through experiments, we show the accuracy and efficiency of PASLIC-on-Spark on an actual cloud environment configured with 8 worker nodes using Amazon AWS. The experimental results indicate that PASLIC-on-Spark improves the accuracy of PSLIC-on-Spark by 3.66–3.77% of BR and 1.39–1.96% of UE. PASLIC-on-Spark still decreases that of processing time SLIC significantly 1.5–1.67 times on a single-node configuring using 8 CPU cores and 1.18–1.26 times on a cloud environment using 8 computing nodes.

Keywords SLIC · Image segmentation · Apache spark · Parallel processing · Accuracy

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021R1F1A1064050).

✉ Hyuk-Yoon Kwon
hyukyoon.kwon@seoultech.ac.kr

Extended author information available on the last page of the article

1 Introduction

We are witnessing a huge amount of image data generated from various devices such as smartphones, medical devices, satellites, and surveillance cameras. Such image data are being stored and processed for many types of applications and services including facial recognition, autonomous driving, and precision medicine. Since an image can include multiple objects with their semantic information, we often use image segmentation techniques to detect the objects by partitioning the image into several segments [1]. For image segmentation, we can utilize several features of image data such as color, location, texture, and intensity. Superpixel segmentation is the representative image segmentation technique that produces clusters (called superpixels) of similar and spatially close pixels [2]. Since superpixels can reduce the overhead of downstream tasks by providing summarized image data [3], they have been widely used for not only image segmentation [4] but also centerline extraction [5], object localization [6], and depth estimation [7]. Simple linear iterative clustering (SLIC) [2] is an efficient technique to find superpixels with the time complexity of $O(N)$. SLIC has been applied to many kinds of computer vision tasks and applications such as segmentation [8–10], depth estimation [11], optical flow estimation [12], saliency estimation [13], hyperspectral image classification [14], object detection and classification in unmanned aerial vehicle (UAV) image data [15, 16], object detection for glaucoma in medical image data [17], fault detection of solar energy system in thermal image data [18], and a breast cancer classification in ultrasound image data [19].

One critical challenge for image segmentation is the increasing usage of high-resolution images thanks to the advancement of imaging equipment. For example, one satellite image of DigitalGlobe has 256 million ($16,000^2$) pixels covering an area greater than $64km^2$ [20]. SLIC requires about 1600 times more time to process a satellite image with 256 million pixels than to process an image with 0.15 million pixels, which is the typical image size for evaluating image segmentation [21]. To efficiently process such high-resolution images for image segmentation, it is imperative that we extend SLIC for parallel and distributed processing using multiple cores and machines. Even though there are a few proposed techniques [22, 23] for parallel processing of SLIC, it is challenging to run SLIC in a parallel manner for a single image because we need to effectively partition the image into smaller blocks for parallelism. In other words, one image block may not include all necessary pixels to generate optimal superpixels.

In this paper, we propose and develop a parallel and distributed version of SLIC on top of Apache Spark, a state-of-the-art big data framework that provides the MapReduce programming model [24] for processing large-scale data on a cluster of machines. Spark has been widely used for various big data applications such as cloud-based log file analysis [25], mobile big data analysis [26], and bioinformatics data analysis [27]. We present a parallel algorithm for SLIC on Apache Spark, which we call *PSLIC-on-Spark*. To this purpose, we extend the original SLIC algorithm to use the operations in Apache Spark, supporting its parallel processing on multiple executors in the Apache Spark cluster. Then, we propose an improved algorithm of PSLIC-on-Spark that improves the accuracy of PSLIC-on-Spark, which we call *PASLIC-on-Spark*. We employ two important features for PASLIC-on-Spark. It contains two main features: (1) image partitioning considering the shape and position of the clusters rather than a evenly partitioning method and (2) controllable duplication for the boundary between image partitions.

The contributions of this paper are summarized as follows:

1. We analyze the trade-off relationship of PSLIC-on-Spark between its processing speed and accuracy when performing segmentation for the original image datasets. Especially, we identify two limitations in PSLIC-on-Spark, which degrade the accuracy of the original SLIC.
2. We verify the trade-off relationship between the processing speed and the accuracy of PSLIC-on-Spark. Specifically, we show that PSLIC-on-Spark using 8 CPU cores significantly reduces the processing time of SLIC by 2.24 ~ 2.93 times while it reduces the boundary recall (BR) of SLIC by 1.54–6.32% and increases under-segmentation error (UE) by 1.79–6.2%. In contrast, PSLIC-on-Spark using 2 CPU cores reduces the processing time of SLIC only by 1.38–1.45 times while it reduces the recall of SLIC only by 0.28–1.5%, and increases UE by 0.25 –1.77%.
3. We show the accuracy and efficiency of PASLIC-on-Spark on an actual cloud environment configured with 8 worker nodes using Amazon AWS. The experimental results indicate that PASLIC-on-Spark improves the accuracy of PSLIC-on-Spark by 3.66–3.77% of BR and 1.39–1.96% of UE. PASLIC-on-Spark still decreases that of processing time SLIC significantly 1.5–1.67 times on a single-node configuring using 8 CPU cores and 1.18–1.26 times on a cloud environment using 8 computing nodes.

The paper is organized as follows. In Sect. 2, we explain the related work. In Sect. 3, we explain the background. In Sect. 4, we present the PSLIC-on-Spark algorithm and analyze the result in terms of the parallelism and the accuracy. In Sect. 5, we describe PASLIC-on-Spark algorithm to overcome the limitations of PSLIC-on-Spark. In Section 6, we describe the experimental results to show the trade-off between the processing speed and the accuracy and the effectiveness of PASLIC-on-Spark. In Sect. 7, we conclude the paper.

2 Related work

In this section, we summarize the related works of this paper. It consists of three parts: (1) superpixel segmentation, (2) the benchmark for superpixel segmentation, and (3) parallel processing of superpixel segmentation algorithms.

2.1 Superpixel segmentation

Various algorithms for superpixel segmentation have been proposed. We can classify them into two groups of algorithms: (1) graph-based algorithms and (2) gradient-ascent-based algorithms. In the graph-based algorithms, Wu et al. [28] proposed the Minstpixel algorithm based on the minimum spanning tree. The algorithm applies the greedy optimization strategy to the adjacent pixels from the first minimal gradient seed pixels. Liu et al. [29] proposed a superpixel segmentation algorithm using entropy rate. Here, each pixel is mapped to a node of the graph and the edge is defined between similar pixels. The entropy rate calculation takes into account the similarity between nodes and forms a cluster. Shi et al. [30] proposed a superpixel segmentation method that models the image pixels as a graph. That is, they mapped each pixel as the node and the similarity between pixels as the edge. Then, to generate superpixels, they partitioned the graph so as to minimize the dissimilarity between groups and maximize the similarity within groups. In the gradient-ascent-based algorithms, Wang et al. [31] proposed an adaptive non-local random walk algorithm (ANRW). ANRW improves the accuracy of the superpixel by creating the initial cluster seeds based on the gradient of the pixels and by locating them to be not at the object boundary. Li et al. [32] proposed a linear spectral clustering (LSC) algorithm to map pixels of the image to a feature space in 10 dimensions. LSC preserves the global properties of the image by managing those high-dimensional features. Levinshtein et al. [33] proposed a superpixel segmentation method, named turbo pixel, which calculates the gradient of the surrounding pixels for each initial seed pixel and expands the seed pixel to the gradient flow to form a superpixel until all pixels are assigned into one of superpixels.

Meanwhile, there have been research efforts that apply the superpixel segmentation algorithms to a specific dataset. Wang et al. [34] applied the SLIC algorithm to the polarimetric synthetic aperture radar (PolSAR) image data. Specifically, the gradient information in the PolSAR image used to reduce the number of repetitions in SLIC and to improve the accuracy. Sun et al. [35] proposed SLFTIC that modifies SLIC for the classification of coal images. SLFTIC algorithm produces segments by combining color, space location, and texture information during the clustering phase, enabling to detect the segment of minerals that is similar with the background.

There have been studies using superpixel to solve problems in certain applications. Sharma and Biswas [36] used superpixel to classify hyper spectral images. In particular, for the classification process of hyper spectral images, graph-based superpixel segmentation, which uses the entropy rate of the path in the graph to segment pixels, is used to generate segments of HSI and to group extra homogeneous. Zhao et al. [37] proposed an improved image semantic segmentation method using superpixels and conditional random fields (CRFs). In this work, superpixel-level semantic features, which are created through SLIC superpixel, and pixel-level semantic features, which are created through fully convolutional networks (FCN), are combined to improve the segmentation results.

Recently, due to the development of deep learning techniques, there have also been many research efforts to produce superpixels through deep neural networks.

Jampani et al. [38] applied a convolutional neural network (CNN) model to extract the features for obtaining superpixels, enabling the extraction of features tailored to given superpixels, instead of fixed features. Yang et al. [39] proposed a superpixel segmentation using a CNN model. They regarded the initial regular grids of the image as the clusters and classified each pixel into one of the surrounding clusters using a CNN model, resulting in each cluster becoming a superpixel.

2.2 The benchmark for superpixel segmentation

The performance of superpixel segmentation has been evaluated by various metrics. Neubert et al. [40] evaluated the performance of eight superpixel segmentation algorithms by using two metrics: the boundary recall and the under-segmentation error, which are typical metrics to evaluate the error in superpixel. Radhakrishna et al. [2] compared 5 superpixel algorithms using boundary recall, under-segmentation error, segmentation speed, and segmentation accuracy.

2.3 Parallel processing of superpixel segmentation algorithms

There have been only a few existing methods to deal with parallel processing in superpixel algorithms. Prajapati and Vij [41] classified parallel processing types into three categories: (1) data parallelization, (2) task parallelization, and (3) pipeline parallelization. Ren and Reid [22] proposed a gSLIC algorithm that expands the SLIC algorithm to be parallelized for the Nvidia CUDA framework based on GPU. Similarly, Quesada-Barriuso et al. [42] proposed a waterpixel algorithm to be parallelized for the Nvidia CUDA framework for hyperspectral remote sensing images. Derkson [23] partitioned the entire image with a tile-wise framework in SLIC, which tries to preserve the original superpixel in each partitioned image block to reduce the accuracy degradation by allowing overlapping between partitioned images.

A few studies [22, 23] have been proposed to extend SLIC for parallel algorithms. However, they still have the limitation in applying them in a distributed environment. Derksen et al. [23] proposed their own parallel algorithm for SLIC; however, there is a limitation in scalability and availability because the additional implementation for them is required. Ren and Reid [22] proposed a parallel superpixel algorithm so as to leverage multiple GPUs. However, its scalability is limited to a possible maximum number of GPUs equipped in a single machine and it requires much effort to extend it to a distributed environment.

Processing methods based on Apache Spark, including our methods, can naturally support scalability and availability in dealing with large-scale images [43]. Without Apache Spark, we need to additionally develop those functionalities. Moreover, Apache Spark allows us to easily support parallel processing in a scalable distributed environment by constructing the algorithms with map and reduce operations. Without Apache Spark, we need to directly implement the parallelism of the algorithms considering an underlying distributed configuration. As a result, we do not directly compare the method proposed in this paper with the previous parallel algorithms for SLIC [22, 42].

It has been shown that a method based on Apache Spark is a feasible approach in the image processing as well. Liu et al. [44] proposed distributed processing of the fuzzy *c*-means algorithm based on Apache Spark to segment pixels in agricultural image data. That is, they extended the existing fuzzy *c*-means algorithm to the Apache Spark framework. However, to the best of our knowledge, there have been no attempts to produce superpixel on Apache Spark, and this paper is the first research effort to study a parallel and distributed processing method for SLIC on Apache Spark.

A preliminary version of this work appeared in *Proc. 2021 IEEE International Conference on Big Data and Smart Computing*, pp. January 5–12, 2021 [45]. This is a fully rewritten and extended version of the preliminary version. The major extensions include (1) a completely new algorithm, PASLIC-on-Spark, to improve the accuracy than the algorithm in the preliminary version, (2) new experiments on an actual distributed environment using Amazon AWS, and (3) the detailed and extended study on related work.

3 Background

In this section, we provide a brief background about SLIC and Apache Spark.

3.1 SLIC

SLIC utilizes two types of pixel information to generate superpixels: (1) pixel color information and (2) pixel location in the given image [2]. It represents the pixel color information as a three-dimensional vector (L, a, b) in the CIELAB color space. For the pixel location, it uses a two-dimensional position (x, y) on the Euclidean space. SLIC uses the vectors to calculate the distance between a pixel and a cluster center during pixel clustering.

Given that there are N pixels in an input image, SLIC groups them into k (an input parameter for running SLIC) clusters (or segments). Assuming roughly equally sized clusters, the distance S between two cluster centers is approximately $\sqrt{N/k}$, and the size of a cluster is approximately $S \times S$. Therefore, to find a cluster, SLIC assumes that pixels related to this cluster are located within the range of $2S \times 2S$ around the cluster center, reducing the overhead of clustering. Then, SLIC calculates the distance between a pixel i, which is represented in $[l_i, a_i, b_i, x_i, y_i]$, and a k-th superpixel cluster center, which is represented as $[l_k, a_k, b_k, x_k, y_k]$. To calculate the distance, SLIC uses two distance values: (1) color value distance d_{lab} and (2) Euclidean distance d_{xy}, as defined in Eq. 1. The final distance is the sum of the color value distance and the normalized Euclidean distance, as defined in Eq. 1.

$$\begin{aligned} d_{lab} &= \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \\ d_{xy} &= \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \\ D_\mathrm{s} &= d_{lab} + \tfrac{m}{S} d_{xy} \end{aligned} \tag{1}$$

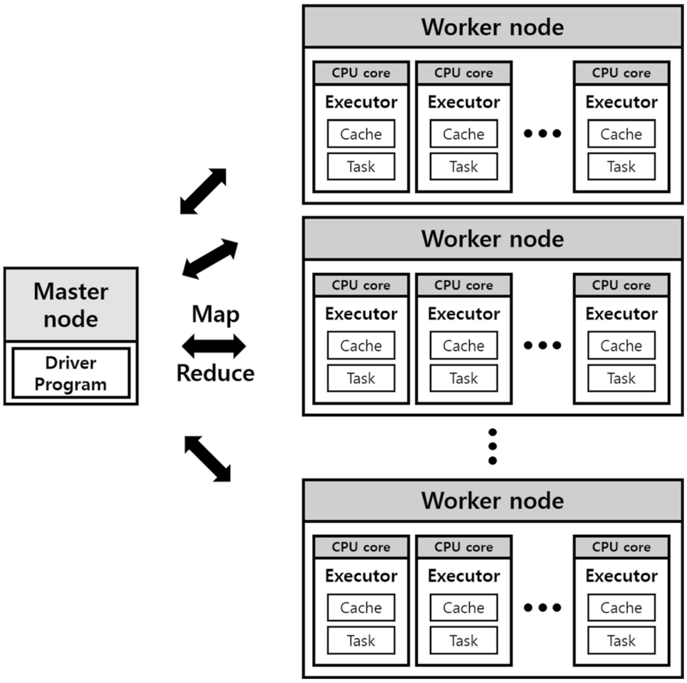


Fig. 1 The architecture of Apache Spark cluster

To adjust the weight of each distance value, SLIC uses a parameter called m. By increasing m, we can give higher weights to d_{xy}. When we deal with the large size of superpixels, d_{xy} is considered much more importantly than d_{lab} [2]. Initially, SLIC splits the entire image into regular grids with a constant size where each grid is mapped to a target segment. Then, it iteratively updates the segmentation label of each pixel by computing D_s between the pixel and the cluster center C_k and finding the nearest cluster.

3.2 Apache Spark

Apache Spark is a state-of-the-art open-source big data framework that supports MapReduce-like operations on a cluster of machines and provides efficient large-scale data processing through in-memory persistence and execution optimizations [43]. Spark, as a unified processing engine, can be configured with one of several resource management frameworks such as YARN and Kubernetes. A typical Spark cluster consists of a master node (and potentially one standby node for high availability) and a set of worker nodes. Figure 1 shows working of a typical MapReduce job on a Spark cluster. When the job is submitted to the master node, Spark distributes the job into multiple executors on multiple worker nodes for parallel processing. There is no data shuffling among the executors in the map phase. Spark also

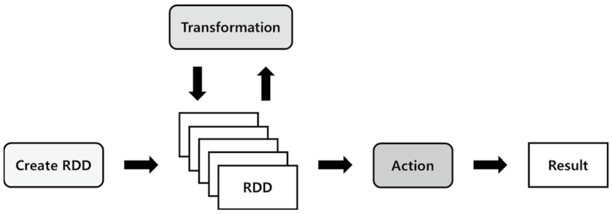


Fig. 2 Resilient distributed datasets on Apache Spark

supports various storage engines such as HDFS (Hadoop Distributed File System), MongoDB, and Amazon S3 to store input and output data.

Resilient Distributed Datasets (RDDs) are the fundamental unit of data in Apache Spark. RDDs are partitioned across multiple machines on a cluster for parallel processing of large-scale data. Since RDDs are immutable, we can only create new ones based on existing ones, instead of modifying existing RDD data. Spark supports two types of operations: (1) transformations and (2) actions. We can create a new RDD from an existing one using a transformation. For example, the `map` and `filter` operations are frequently used transformations. On the other hand, an action triggers actual data processing and returns a value to the driver program (or results are written to the storage system). For example, the `reduce` and `take` operations are actions. Figure 2 shows the working of a typical Spark job that consists of multiple transformations followed by one action. After we create an input RDD, we process the data and generate new RDDs through transformations. Because of the lazy evaluation in Spark, the transformations will not start actual data processing. The action will trigger the actual processing and return a value.

4 Parallelization of SLIC on Apache Spark

In this section, we describe our parallel algorithm that can run SLIC on the top of Apache Spark.

4.1 PSLIC-on-Spark: A parallel algorithm for SLIC

Given an input image, our algorithm partitions the image into smaller image blocks to utilize the parallel processing capability of Spark. Our algorithm consists of two core parts: (1) Algorithm 1 for the cluster master and (2) Algorithm 2 for the executors. As a first step, we determine how to partition the image and distribute the partitioned image blocks to the executors. We assign one image block to executor i with (1) pixel ranges in the image to be processed in i (PR_i) and (2) the number of target segments assigned to i (TS_i). To calculate PR_i, we horizontally partition the input image to generate equally sized image blocks. To calculate TS_i, we divide the total number of segments by the number of executors to evenly distribute segments among the executors. In the case of PR_i, we assign the remaining pixels divided by

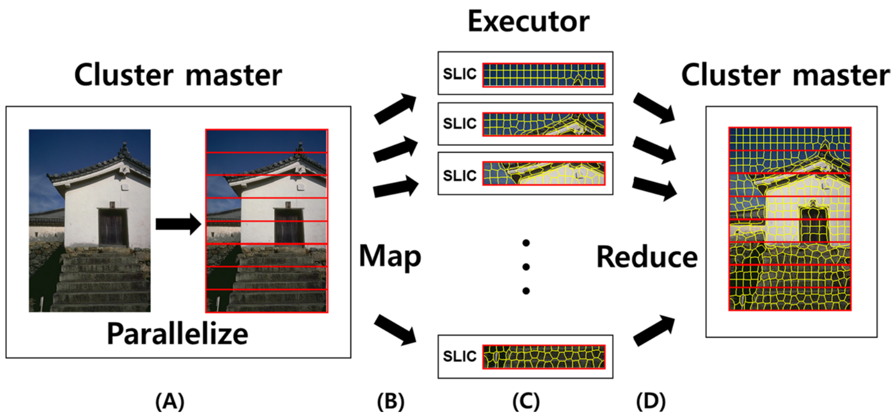


Fig. 3 The overall process of PSLIC-on-Spark

p, which is the target number of partitions, to some of the executors; in the case of TS_i, we simply cut the remaining segments off.

```
Algorithm 1 The algorithm for the cluster master
Input: an array $PR_i (1 \leq i \leq p), TS_i, (1 \leq i \leq p)$, and the number of partitions, $p$
Output : $\sum_{i=1}^{p} S_\mathrm{i}$
1: sc = spark.SparkContext
2: rdd = sc.parallelize((array of $PR_i, TS_i$), $p$)
3: rdd.map(SLIC)
4: rdd.reduce()
```

```
Algorithm 2 The algorithm for the executor
Input : $PR_i$, $TS_i$
Output : $S_\mathrm{i}$
1: Load an image block, $IB_i$, mapped to $PR_i$ in the entire image file
2: $S_i$ = SLIC($IB_i, TS_i$)
3: return $S_i$
```

Algorithm 1 takes the inputs: (1) an array $PR_i(1 \leq i \leq p)$, (2) TS_i, $(1 \leq i \leq p)$, and (3) the target number of partitions, p, which is equal to the number of executors. The algorithm runs three Spark operations: (1) `parallelize` for assigning PR_i and TS_i to each executor among p executors, (2) `map`(*SLIC*) for transferring function *SLIC* (the original SLIC algorithm) to each executor, and (3) `reduce` for integrating the results of all executors. The final results are represented in $\sum_{i=1}^{p} S_\mathrm{i}$. Algorithm 2 for an executor i takes the inputs: 1) PR_i, (2) TS_i. Each executor i runs the given SLIC algorithm on its assigned image block to generate superpixels S_i.

Figure 3 shows the overall process of PSLIC-on-Spark using an example. In step (A), PSLIC-on-Spark horizontally partitions the input image and assigns one image

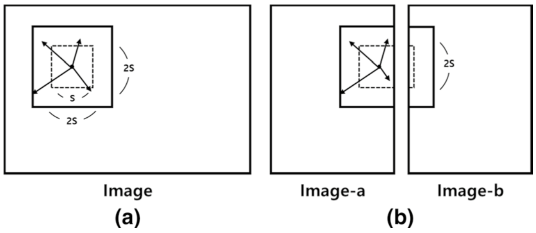


Fig. 4 The loss of some pixels required in calculating one superpixel in PSLIC-on-Spark

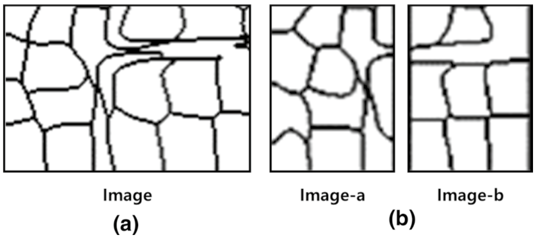


Fig. 5 The boundary mismatch in PSLIC-on-Spark

block to each executor using the `parallelize` operation. In step (B), PSLIC-on-Spark transfers the original SLIC algorithm to all executors for parallel processing using the `map` operation. In step (C), each executor runs the SLIC algorithm on its assigned image block to generate sub-superpixels. In step (D), PSLIC-on-Spark aggregates all sub-superpixels generated by all executors using the `reduce` operation to get final superpixels.

4.2 Limitations of PSLIC-on-Spark

Even though PSLIC-on-Spark can improve the processing speed of SLIC using multiple executors, it may generate different segments, compared to those of SLIC, because a partitioned image block is independently processed without having access to other image blocks. In this section, we analyze the limitations of PSLIC-on-Spark in terms of its accuracy by comparing it to that of SLIC. Specifically, we find two limitations regarding the accuracy of PSLIC-on-Spark: (1) loss of some pixels required in calculating one superpixel and (2) boundary mismatch between sub-superpixels in adjacent image blocks.

Limitation 1. As described in Sect. 3.1, SLIC assumes that pixels related to a cluster are located within the range of $2S \times 2S$ around its cluster center. Figure 4a shows a cluster and its $2S \times 2S$ range. However, since PSLIC-on-Spark partitions the input image into smaller image blocks for parallel processing, one image block cannot have all pixels located within the $2S \times 2S$ range in the original input image, as depicted in Fig. 4b. Therefore, the accuracy of PSLIC-on-Spark can degrade compared to that of the original SLIC algorithm.

Limitation 2. Since each executor processes one image block independently without having access to other image blocks in PSLIC-on-Spark, there can be boundary mismatches between adjacent image blocks when PSLIC-on-Spark integrates sub-superpixels generated by all executors. For example, Fig. 5a shows

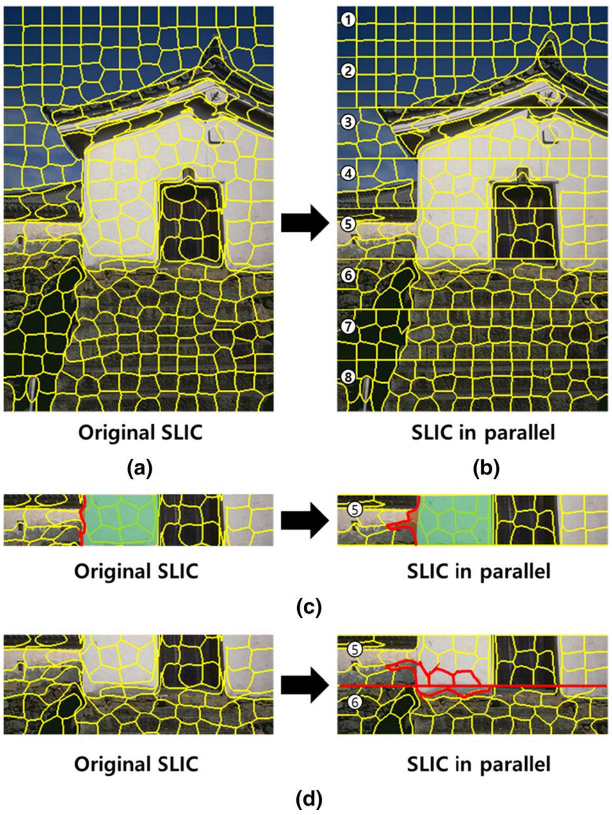


Fig. 6 The comparison of the results by SLIC and PSLIC-on-Spark

the segmentation result of SLIC on an input image while Fig. 5b shows the result of PSLIC-on-Spark on the partitioned image blocks with boundary mismatches between the sub-superpixels.

We explain the limitations using a full image in detail. Figure 6a shows the result of SLIC when the number of target segments is 300. Figure 6b shows the integrated result of PSLIC-on-Spark when we run eight executors (i.e., eight image blocks). Figure 6c highlights the segmentation difference between SLIC and PSLIC-on-Spark for image block 5 in detail. The green area represents the portion of a superpixels covering the object wall, while the red area shows the region not overlapping with the object wall. This shows the effectiveness of SLIC in detecting the object. It also shows that PSLIC-on-Spark preserves most parts of the original superpixel but loses some parts because of **Limitation 1**. Figure 6d shows that the superpixels generated by PSLIC-on-Spark at the boundary between image blocks 5 and 6 do not match smoothly because of **Limitation 2**.

The number of partitions in PSLIC-on-Spark determines the size of each partitioned image block when the number of target segments is fixed. As the size of each image block becomes small, the effects of **Limitation 1** and **Limitation 2** will increase. This implies that as the number of partitions increases, PSLIC-on-Spark will generate smaller partitioned image blocks, degrading the segmentation accuracy while increasing the parallelism of the algorithm. We verify this implication through the extensive experimental results in Sect. 6.

5 Improving segmentation accuracy

Section 4 describes the limitations of PSLIC-on-Spark in terms of segmentation accuracy. In this section, we propose an improved algorithm, called *Parallel-and-Accurate SLIC-on-Spark* (*PASLIC-on-Spark* in short), that maintains the benefit of parallel processing of PSLIC-on-Spark while reducing the accuracy degradation incurred by image partitioning. It contains two main features: (1) effective image partitioning (in Sect. 5.1) and (2) duplication for the boundary between image partitions (in Sect. 5.2).

5.1 Effective image partitioning

In this section, we devise a new image partitioning and allocation method for solving Limitation 2 described in Sect. 4.2 (i.e., boundary mismatch between sub-superpixels in adjacent image blocks). To reduce accuracy loss incurred by partitioning, it is necessary to minimize the inconsistency between sub-superpixels. Since we run SLIC on each partitioned image block, we can improve accuracy of segmentation if we effectively partition the input image such that partitioned image blocks preserve the superpixels in the entire input image. As described in Sect. 3.1, SLIC divides the entire image into regular grids of the constant size, which is determined by a target number of segments, k, along with the initial cluster center $C_i(1 \leq i \leq k)$.

Then, the clusters are iteratively updated by computing the distance between cluster centers C_k and the surrounding pixels. Here, the important observation is that the final shape and position of the superpixel become different depending on the initial cluster center C_k. Thus, if we split the image without considering a target number of segment k and regular grids in the original image, such as PSLIC-on-Spark, the position of cluster center C_k in each image block would be significantly different from the result of the original SLIC.

In PASLIC-on-Spark, to resolve the superpixel mismatch problem, we propose a method for partitioning the input image while preserving the regular grids in the original image. In other words, by partitioning the image along the boundaries of the regular grid, we can preserve the number and shape of superpixels as similar as possible to the SLIC results from the original image. Specifically, while PSLIC-on-Spark evenly distributes all pixels across all executors, PASLIC-on-Spark uses the regular grids, instead of pixels, as the unit for partitioning the input image. For example, for the input image in Fig.7, assuming that there are 27 rows of grids and 8 executors, PASLIC-on-Spark assigns three or four grids to each executor. Even though the pixels are not evenly distributed unlike PSLIC-on-Spark, PASLIC-on-Spark can preserve the original regular grids in the input image.

Table 1 The notation for PASLIC-on-Spark

Variables	Definition
S	The initial distance between two adjacent cluster centers
p	The number of target partitions
i	The i-th image block out of partitioned image blocks in the height of the image ($1 \leq i \leq p$)
$pixel_h$	The number of pixels in the height of the image
$pixel_w$	The number of pixels in the width of the image
$pixel\text{-}partition(pixel_h, p, i)$	i-th image block partitioned by p consisting of pixels ranged from $(\frac{pixel_h}{p} \times i)$ to $(\frac{pixel_h}{p} \times (i+1) - 1)$ where $\frac{pixel_h}{p}$ is rounded down to the first decimal point
$grid_h$	The number of grids in the height of the image
$grid_h(i)$	The number of grids in the height of the i-th image block
$grid_w$	The number of grids in the width of the image
$grid\text{-}partition(grid_h, p, i)$	i-th image block partitioned by p consisting of grids ranged from $(\frac{grid_h}{p} \times i)$ to $(\frac{grid_h}{p} \times (i+1) - 1)$ where the remainder grids of $\frac{grid_h}{p}$ are distributed one by one from the top image blocks
$overlap_h$	The number of grids overlapped from one side of the image block

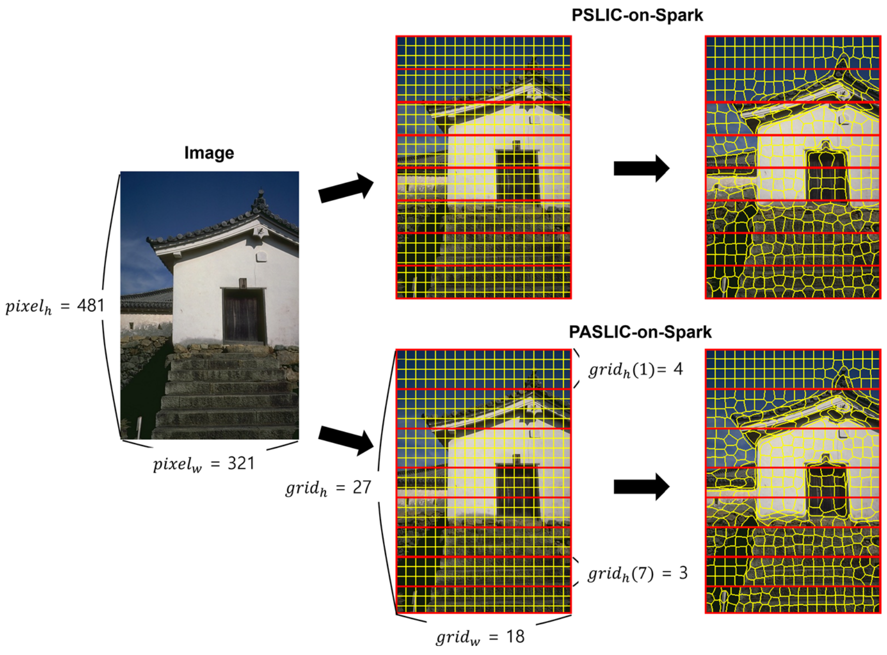


Fig. 7 The partitioned result comparison between PSLIC-on-Spark and PASLIC-on-Spark

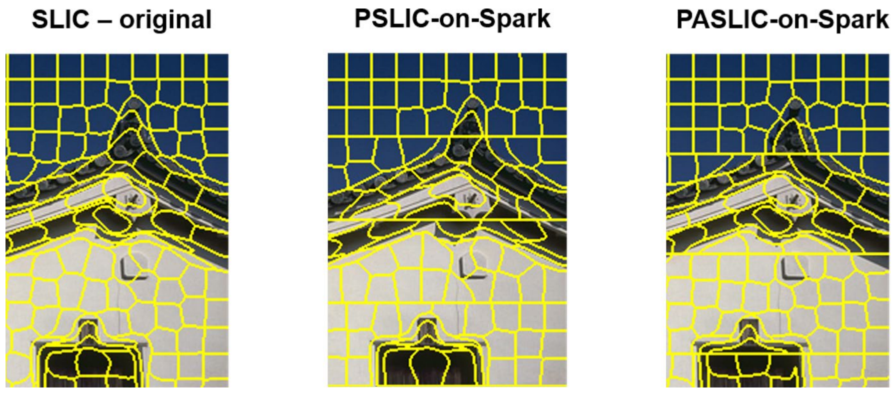


Fig. 8 The final result comparison of SLIC, PSLIC-on-Spark, and PASLIC-on-Spark

Table 1 shows the notations needed to explain how to partition images for PASLIC-on-Spark. SLIC partitions the entire image in the unit of regular grids where each one is $S \times S$ in size. Then, the image consists of the number of grids calculated by $grid_h \times grid_w$. In PSLIC-on-Spark and PASLIC-on-Spark, we define $grid_h(i)$ for the i-th image block partitioned by p to differently represent the number of grids for each image block. $grid_w$ is the same for all of SLIC, PSLIC-on-Spark, and PASLIC-on-Spark because we partition the image in the height. SLIC uses the number of grids, i.e., $grid_h \times grid_w$, for a target number of segments k. In PSLIC-on-Spark and PASLIC-on-Spark, we use $\sum_{i=1}^{p} grid_h(i) \times grid_w$ for k. $grid_h(i)$ for PSLIC-on-Spark is calculated by $pixel\text{-}partition(pixel_h, p, i)$ in the unit of the pixel and that for PASLIC-on-Spark by $grid\text{-}partition(grid_h, p, i)$ in the unit of the grid.

Figure7 shows the result of segmenting images of 321×481 pixels according to PSLIC-on-Spark and that according to PASLIC-on-Spark. We can verify the effect of preserving the initial regular grids in PASLIC-on-Spark compared to PSLIC-on-Spark. In the example as in Fig.7, we have the number of target segments, k, is 500 and the number of partitions p is 8. First, let us consider the application of SLIC to the example image. In SLIC, $S = 18, grid_h = 27$ and $grid_w = 18$ are obtained by the definition for a given image based on the SLIC algorithm, resulting that a target k is redefined as $grid_h \times grid_w = 486$. Then, let us compare this result with those of PSLIC-on-Spark and PASLIC-on-Spark. In PSLIC-on-Spark, we partition the entire image by p evenly in the unit of the pixel. Because $grid_h(i)$ of every image block i is 3, each image block consists of 54 grids and k of the image becomes 432. This implies that, PSLIC-on-Spark produces less superpixels than SLIC roughly 11.2% (i.e., 1-$\frac{432}{486}$), which degrades the overall accuracy. However, in PASLIC-on-Spark, k is preserved as 486 as in SLIC because each image block can contain different number of grids (i.e., 3 or 4 in this example). Furthermore, we can also preserve the initial cluster centers by preserving the initial regular grids, which affects the improvement of the accuracy of PASLIC-on-Spark.

As shown in Fig. 8, PSLIC-on-Spark has quite different segmentation results compared to SLIC because the number of target segments and the position of cluster

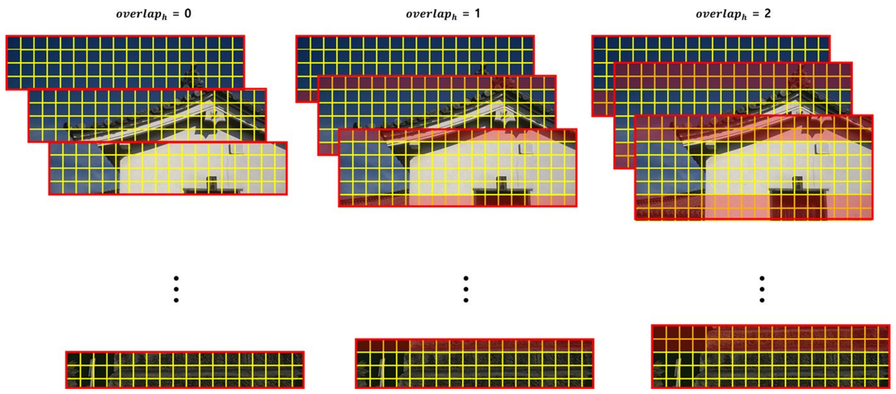


Fig. 9 The area expanded by the overlapping in PASLIC-on-Spark

centers are different. In particular, even superpixels located in the middle of the image block, which are less affected by partitioning, also differ from the superpixels of the original SLIC. On the other hand, PASLIC-on-Spark produces superpixels of relatively similar numbers and shapes compared to the original SLIC results. However, we can observe that the superpixels still have a different shape at the boundary between image blocks.

5.2 Duplication for the boundary between image partitions

In this section, we devise a method to improve the segmentation accuracy by addressing Limitation 1 (i.e., incomplete pixel information between partitioned image blocks) discussed in Sect. 4.2. Although transferring the required pixels via network communication between different nodes can preserve the original accuracy of SLIC, network overhead can dramatically degrade processing efficiency in a distributed environment. Therefore, in PASLIC-on-Spark, we aim to minimize the accuracy degradation while maintaining the efficiency of distributed processing by storing partially redundant data on each node. Specifically, we store additional pixels that are required to construct superpixels in adjacent executors redundantly, removing the network overhead and improving the accuracy. Basically, SLIC can be influenced by pixels in the entire image because cluster centers are repeatedly updated. Thus, the closer a image block in each executor is to the size of the original image, the more accurate results can be generated. Thus, a trade-off relationship is established between the degree to which image blocks are stored in duplicate and accuracy as discussed in Sect. 4.2. $overlap_h$, defined in Table 1, refers to the number of grids expanding to one side of the image block. We have the following three cases for $overlap_h$. (1) If overlapping does not occur between the image blocks as in PSLIC-on-Spark, $overlap_h$ of the image block is zero. (2) For the first and last image blocks, the grid can be expanded to only one side, and the number of expanded grid in the image block increases by $overlap_h$. (3) For the other image blocks, the grid can be expanded in both sides, and the number of expanded grid in image blocks increases by $2 \times overlap_h$. We constraint a possible maximum $overlap_h$ of an image block i as $grid_h(i) - 1$ to prevent the overlapped area to be larger than the image block. Figure 9 shows the expanded area obtained by PASLIC-on-Spark when the number of target segments $K = 500$ and the number of partition $p = 8$ are given. In the image block of Fig.9, the red area represents an expanded grid. For the first and last image blocks, $grid_h(i)$ increases by $overlap_h$. For the other image blocks, $grid_h(i)$ increases by $2 \times overlap_h$.

When the image blocks obtained in the worker nodes are aggregated in the cluster master, a method of the superpixel integration for the overlapped area between image blocks needs to be carefully designed because it affects the accuracy of superpixels. Algorithm 3 shows the integration process of overlapped regions between image blocks. Algorithm 3 takes the inputs: (1) an array of image blocks, IB_i, (2) an array of cluster label of $IB_i(L_i)$. First, we initialize SB_i as the split boundary of IB_i. Then, we initialize an array for a set of clusters that cross with SB_i, *boundarySeg*, as one of the overlapped image blocks (i.e., we choose the block above in the implementation). For the pixels in a region S in SB_i, we update the label of the pixel as the cluster of the image block above if it belongs to *boundarySeg*, otherwise, we label it as the cluster of the other image block (i.e., the block below). The algorithm repeats for each image block i. Then, the entire label of the image, L, is updated by each cluster label L_i.

Algorithm 3 The algorithm for the overlap processing

```
Input: an array $IB_i (1 \le i \le p)$, an array $L_i (1 \le i \le p)$
Output: $L$
1: for $i$ from 1 to $p$
2: Initialize $SB_i$ as the split boundary of $IB_i$
3: Initialize an array for a set of clusters that cross with $SB_i$ $boundarySeg$ as the image
   block above
4: for $j$ from $SB_i$ to $SB_i + S$
5: if $L_i[j]$ is in $boundarySeg$
6: $L[j] = L_i[j]$
7: else
8: $L[j] = L_{i+1}[j]$
```

Figure 10 shows the process to integrate the area overlapped between two image blocks $IB_3 and IB_4$ when k=500 and p=8 are given. The dotted line means the split boundary between image blocks. In Fig. 10a, the red area under the dotted line means the expanded area of IB_3. In Fig. 10b, we select a superpixel generated from IB_3 cross the split boundary. In Fig. 10c, the selected clusters are preserved as the complete superpixels obtained in IB_3. In Fig. 10d, the remaining pixels that are not included by superpixels of IB_3 are selected from IB_4.

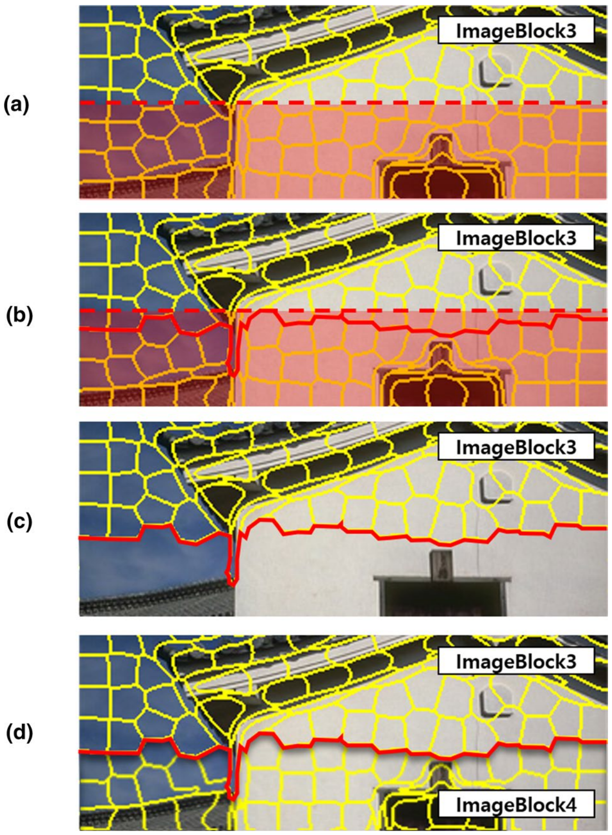


Fig. 10 The process of integration of overlapped area between image blocks in PASLIC-on-Spark

5.3 Algorithm

PASLIC-on-Spark consists of two core parts: (1) the algorithm for the cluster master, Algorithm 4, and (2) the algorithm for the executors, Algorithm 5. Unlike PSLIC-on-Spark, PASLIC-on-Spark needs to deal with expanded and overlapped pixels and the number of target segments are required for each image block. We define the pixels for each worker i, PR_i^*, the number of target segments for i, TS_i. PR_i^* consists of (1) a image block partitioned by p considering the regular grids explained in Section 5.1, PR_i, and (2) overlapped areas between image blocks explained in Section 5.2, O_i.

Algorithm 4 takes the inputs: (1) an array $PR_i^*(1 \leq i \leq p)$, (2) an array $TS_i(1 \leq i \leq p)$, and (3) the number of partition, p. The algorithm executes three Spark operations: (1) `parallelize` for assigning PR_i^* and TS_i for each executor i, (2) `map` for transferring a variation of the SLIC algorithm, $SLIC^*$, which is the same as SLIC only removing the postprocessing (i.e., enforce connectivity), to each worker, and (3) `reduce` for integrating the results of all executors. Then, for all the results transferred from all executors, we apply Algorithm 3 to integrate the sub-results into one final result. Finally, we enforce the connectivity, which is the same postprocessing as in SLIC.

Algorithm 4 The algorithm for the cluster master

```
Input: an array $PR_i (1 \leq i \leq p)$, $TS_i (1 \leq i \leq p)$, $O_i$, $p$
Output : $L$
1: $PR_i^* = PR_i + O_i$
2: sc = spark.SparkContext
3: rdd = sc.parallelize(($PR_i^*$, $TS_i$), p)
4: rdd.map($SLIC^*$)
5: reduce()
6: Algorithm 3 in Section 5.2
7: Enforce connectivity of SLIC
```

Algorithm 5 for each executor *i* takes the inputs from the cluster master: 1) PR_i^*, 2) TS_i. The executor *i* runs the $SLIC^*$ algorithm transferred from the cluster master on its assigned image block to generate superpixels L_i. The outputs of Algorithm 5, $\sum_{i=1}^{p} L_\mathrm{i}$, are integrated as one output by the cluster master.

Algorithm 5 The algorithm for the executor

```
Input: $PR_i^*$, $TS_i$
Output : $\sum_{i=1}^{p} L_\mathrm{i}$
1: Load an image block, $IB_i$, mapped to $PR_i^*$ with $o$ in the entire image file
2: $L_i$=SLIC*($IB_i$, $TSi$)
3: return $L_i$
```

Figure 11 shows the final results of SLIC, PSLIC-on-Spark, and PASLIC-on-Spark for the quantitative evaluation. The results are obtained when k=500 and p=8 are given. (1) is the result of SLIC. (2) is the result of PSLIC-on-Spark. Compared to (1), we observe that the superpixel is cut according to the split boundary of the image block in (2), which degrades the accuracy of SLIC. (3)~(6) is the results of PASLIC-on-Spark, varying $overlap_h$ from 0 to 3. In (3), i.e., $overlap_h = 0$, PASLIC-on-Spark generates similar shapes and positions of superpixels with SLIC in (1) due to the strategy proposed in Section 5.1, but superpixel is still cut along the split boundary of the image. We can observe that PASLIC-on-Spark forms a superpixel that becomes similar with SLIC in (1) as $overlap_h$ increases from (4) to (6) due to the strategy proposed in Section 5.2.

6 Performance evaluation

In this section, we experimentally evaluate our proposed methods using various configurations.

6.1 Experimental environments and data

To evaluate our proposed method, we measure the processing time and accuracy of SLIC, PSLIC-on-Spark, and PASLIC-on-Spark.

For measuring the segmentation accuracy, we use the error metrics defined in a superpixel benchmark [40]: boundary recall (BR) and under-segmentation error (UE). BR follows the precision-recall concept [46] and calculates the ratio of the obtained superpixel contained in the ground truth with respect to the ground truth segments. Therefore, the larger the BR, the higher the accuracy. UE calculates the ratio of the obtained superpixels out of the ground truth with respect to the ground truth segments. Therefore, the smaller the UE, the higher the accuracy.

We use the Berkeley benchmark dataset [21] for our evaluation. It provides the images for segmentation evaluation and the ground truth on detected objects. It contains two image types according to the size: 480×320 and 320×480 pixels, consisting of a total of 500 images. We additionally use various sizes of images of 1032×682, 1426×951, 1738×1159, and 2002×1335 pixels, which are collected from the web, to measure the performance with a variety of image sizes. For all the experiments, we use a value of 10 for m, which is the preference between color proximity and space proximity, and a value of 10 for the maximum number of iterations that stop if no pixels are changed their clusters, which is the same condition as in SLIC.

We conduct our experiments using two different environments on AWS: (1) a single-node configuration using an AWS EC2 instance equipped with multiple processors and (2) a cluster configuration using AWS EMR with YARN and Spark 2.4.5 in which each node has 8 vCPU cores. For the cluster configuration, we use one vCPU core and one executor per node to focus on the actual network overhead of distributed computing while we focus on the effect of parallel processing in a single-node configuration with multiple cores. As a data repository for AWS, we use S3 services working with AWS EMR. We use the source code for SLIC in Python package provided in Anaconda 2.4.5 and the source code for superpixel benchmark.[1]

6.2 Experimental results of PSLIC-on-spark

6.2.1 Processing time

Figure 12 shows the average processing time of SLIC and PSLIC-on-Spark when we process the 500 images using the single-node configuration. The *original* in the x-axis shows the processing time of SLIC, and from *partition*2 to *partition*8 show that of PSLIC-on-Spark by varying the number of partitions from 2 to 8 in which each partition is assigned to an executor (i.e., single CPU core). We also vary the number of the target segments, k, from 500 to 2000. The result indicates that PSLIC-on-Spark significantly improves the processing speed of SLIC-based image

[1] https://github.com/davidstutz/superpixel-benchmark

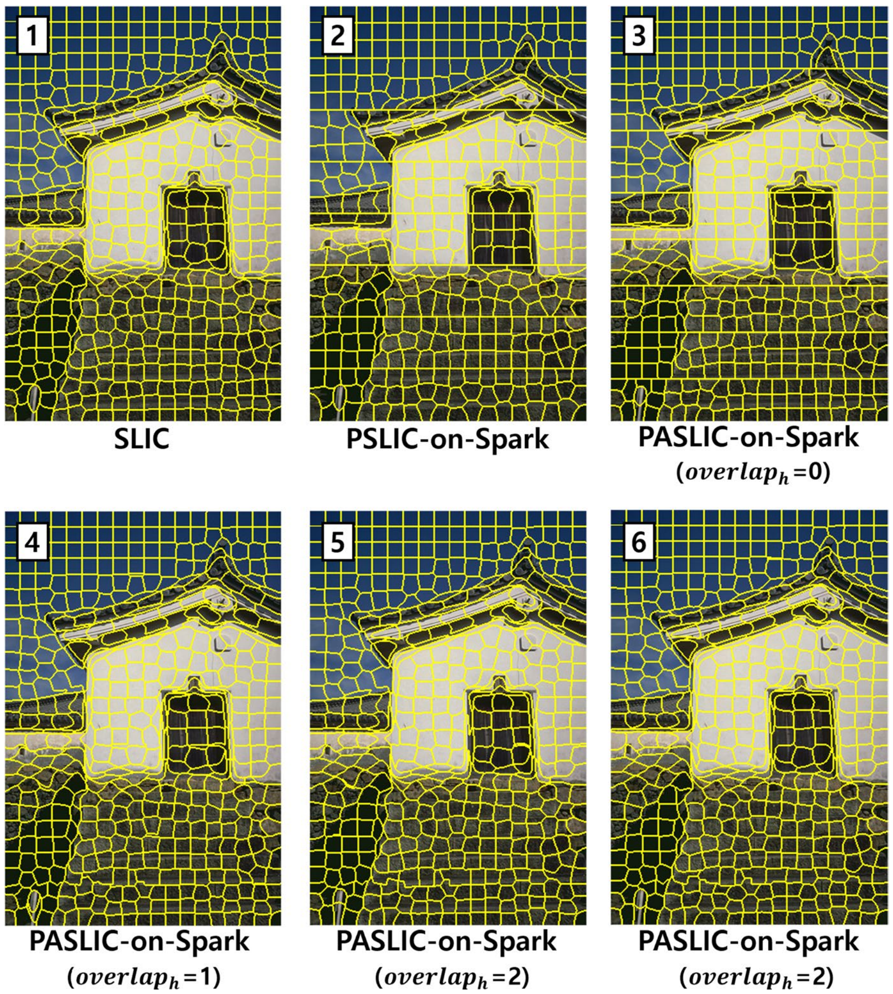


Fig. 11 The final results of original SLIC, PSLIC-on-Spark, and PASLIC-on-Spark

segmentation as we increase the number of partitions because of the parallel processing in Spark. Specifically, *partition*8 reduces the processing time of SLIC by approximately 2.23–2.92 times.

Figure 13 shows the processing time of SLIC and PSLIC-on-Spark for different image sizes. We calculate the average of the processing times for different numbers of target segments k (500, 1000, 1500, and 2000) for each image size. In this experiment, We evaluate five image sizes: 480×320, 1032×682, 1426×951, 1738×1159, and 2002×1335 pixels. We note that the effect of parallel processing becomes more significant as the image size increases, which shows the effectiveness of PSLIC-on-Spark when dealing with a large-scale image. Specifically, PSLIC-on-Spark with 8 partitions reduces the processing time of SLIC only by about 2.23

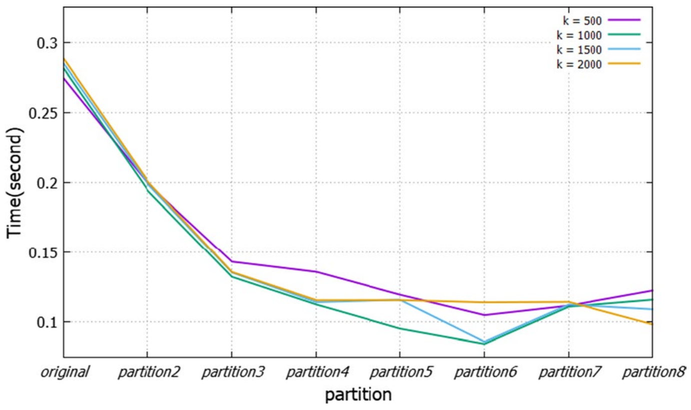


Fig. 12 Processing time of SLIC and PSLIC-on-Spark as the number of partitions is varied

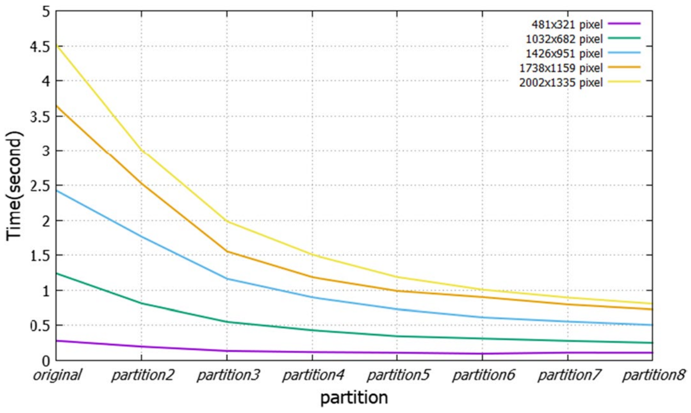


Fig. 13 Processing time of SLIC and PSLIC-on-Spark as the image size is varied

times for the image of 480×320 pixels while it reduces that of SLIC by about 5.59 times for the image of 2002×1335 pixels. We note that the effect of the parallelism becomes more significant when the image size becomes larger. This result implies the initial constant cost to prepare for the task between the cluster master and executors.

Figure 14 shows the processing time of SLIC and PSLIC-on-Spark using our distributed cluster configuration on AWS. *original* in the x-axis shows the processing time of SLIC, and from *partition*2 to *partition*8 show that of PSLIC-on-Spark by varying the number of partitions from 2 to 8 in which each partition is assigned to a computing node. Here, we fix the target segments k as 1000 to focus on the performance change due to the different numbers of nodes in the cloud environment. We use two types of images: 1) a pixel size of 321×481, and 2) that of 481×321. We use 348 images for the former and 152 images for the latter and obtain the average time for the entire 500 images. The result shows that PSLIC-on-Spark can

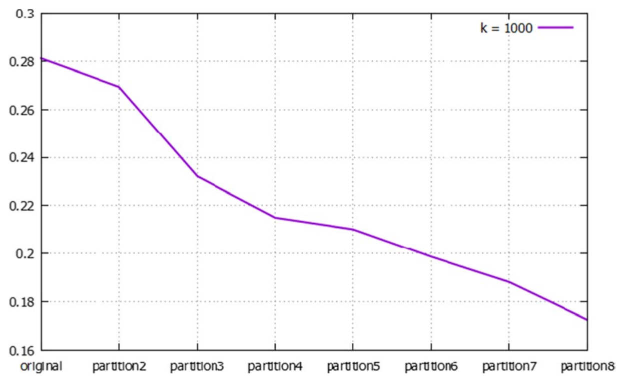


Fig. 14 Processing time of PSLIC-on-Spark as the number of partitions is varied on a distributed environment

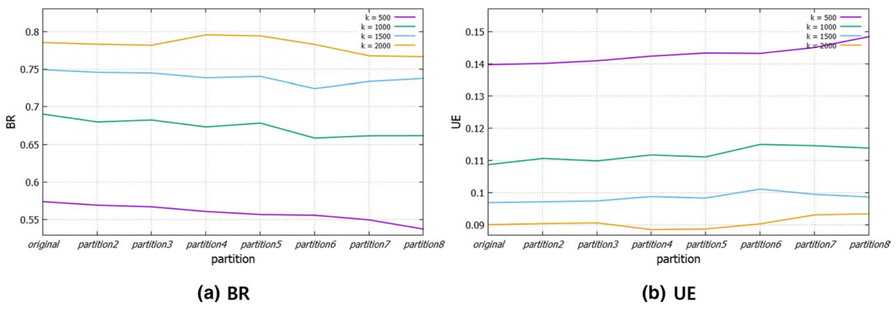
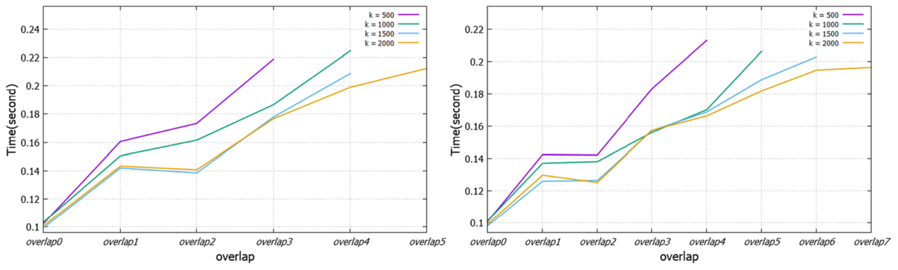


Fig. 15 Accuracy of SLIC and PSLIC-on-Spark as the number of partitions increases

significantly improve the processing speed of SLIC as the number of partitions increases because of the distributed processing in Spark using multiple computing nodes. Specifically, *partition*8 reduces the processing time of SLIC by approximately 1.63 times. We can observe more efficient processing of PSLIC-on-Spark as the number of partitions increases on a cloud environment as well. However, the improvement is rather limited compared to the case of in a single-node configuration, which stems from the network overhead occurred in a distributed environment.

6.2.2 Segmentation accuracy

Figure 15 shows the segmentation accuracy of SLIC and PSLIC-on-Spark. Figure 15a and b shows the average BR and UE of SLIC and PSLIC-on-Spark, respectively, by varying the number of the target segments. Overall, the accuracy of PSLIC-on-Spark is degraded as the number of partitions increases because it splits each image more finely, maximizing the effects of **Limitation 1** and **Limitation 2** explained in Sect. 4.2. In addition, we observe that the degree of accuracy degradation becomes different according to the number of target segments *k*. Specifically,



(a) 481 x 321 images **(b) 321 x 481 images**

Fig. 16 Processing time of PASLIC-on-Spark as the degree of overlapping is varied on a single-node configuration

as the number of segments becomes smaller (i.e., larger superpixels), the accuracy degradation becomes more significant. Because, by Eq. 1, the greater S, which is the size of the superpixel, the less the effect of d_{xy}, the superpixel forms an irregular shape that adheres to the boundary of an object. Here, when an image is partitioned by PSLIC-on-Spark, the loss of information in superpixels becomes more significant. Specifically, when k is 500, BR of PSLIC-on-Spark decreases that of SLIC by about 6.3%, and UE increases by about 6.2%. In contrast, when k is 2000, BR of PSLIC-on-Spark decreases that of SLIC only by about 2.3%, and UE increases only by about 3.7%.

6.3 Experimental results of PASLIC-on-Spark

6.3.1 Processing time

Figure 16 compares the processing time of PASLIC-on-Spark with $p = 8$ in a single-node configuration using two image types from the Berkeley benchmark dataset [21]: 481×321 pixels and 321×481 pixels. We also vary the number of the target segments, k, for the same image size, that are represented in different lines in the figure. Because the degree of overlapping in PASLIC-on-Spark depends on the size of the regular grid, a maximum possible degree of overlapping could be affected by the number of target segments, k and image size. As a result, we have a different maximum degree of overlapping for each case of PASLIC-on-Spark. Figure 16a shows the average processing time of PASLIC-on-Spark for 481×321 pixels as the degree of overlapping (i.e., $overlap_h$) is varied from 0 to 5; Fig. 16b that for 321×481 pixels images as $overlap_h$ is varied from 0 to 7. $overlap0$ in the x-axis shows the processing time when $overlap_h$ is 0, which is approximately similar with that of PSLIC-on-Spark. Varying from $overlap1$ to the maximum degree of overlapping (i.e., $overlap5$ for 481×321 pixels images and $overlap7$ for 321×481 pixels images), we measure the processing time of PASLIC-on-Spark. The experimental results show that PASLIC-on-Spark increases the processing time due to the increase in image block size as the overlapping of the image block increases. We note that PASLIC-on-Spark is obviously slower than PSLIC-on-Spark roughly 1.68–1.81 times due to the

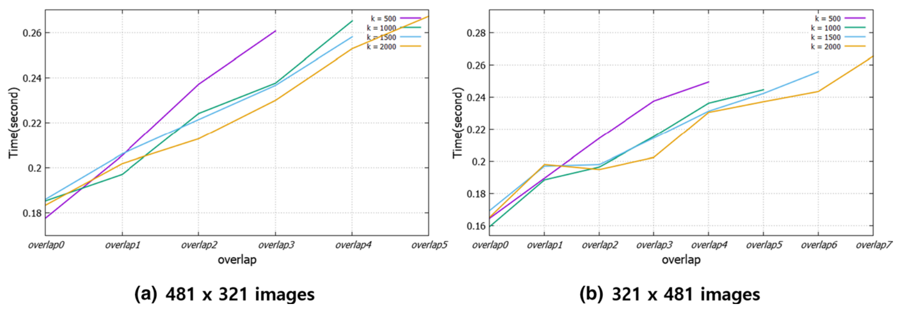


Fig. 17 Processing time of PASLIC-on-Spark as the degree of overlapping is varied on a distributed environment

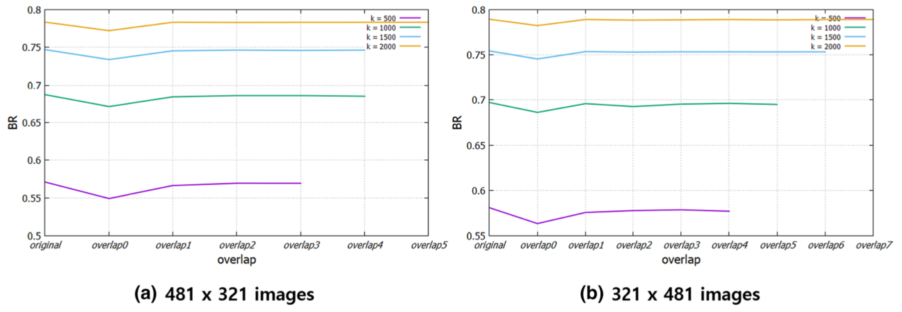


Fig. 18 Boundary Recall of SLIC and PASLIC-on-Spark as the degree of overlapping is varied

overlapping to improve the accuracy, but it is still faster than SLIC roughly 1.5–1.67 times in the case of using the degree of overlapping is 1 or more.

Figure 17 compares the processing time of PASLIC-on-Spark with $p = 8$ in a distributed environment with multiple nodes using two image types from the Berkeley benchmark dataset [21]: 481×321 pixels and 321×481 pixels. Figure 17a shows the average processing time of PASLIC-on-Spark for 481×321 pixels images of $overlap_h$ from 0 to 5; Fig. 17b for 321×481 pixels images of $overlap_h$ from 0 to 7. The experimental results show that PASLIC-on-Spark also increases the processing time as the degree of overlapping increases on a distributed environment. PASLIC-on-Spark is slower than PSLIC-on-Spark roughly 2.22–2.29 times, but it is still faster than SLIC roughly 1.18–1.26 times in the case of using the degree of overlapping is 1 or more.

6.3.2 Accuracy measurement

Figure 18 shows the segmentation accuracy of SLIC and PASLIC-on-Spark. Figure 18a and b shows the average BR of SLIC and PASLIC-on-Spark for 481×321 pixels and 321×481 pixels, respectively, by varying the number of the target

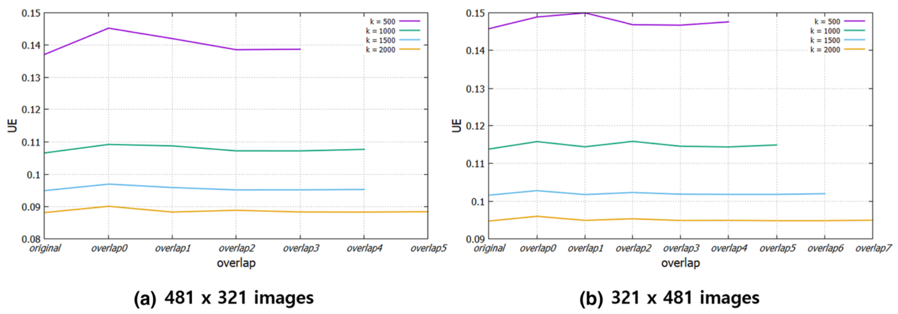


Fig. 19 Under segmentation error of SLIC and PASLIC-on-Spark as the degree of overlapping is varied

segments. Overall, the BR of PASLIC-on-Spark becomes similar to that of SLIC when the degree of overlapping increases after decreasing it at *overlap*0 significantly. Specifically, the difference of BR between PASLIC-on-Spark and SLIC is only 0.01–0.1%. We note that PASLIC-on-Spark significantly improves the BR of PSLIC-on-Spark in the case of $p = 8$ by 3.66–3.77% when $overlap_h$ from 1 to *max*.

Figure 19 shows the segmentation accuracy of SLIC and PASLIC-on-Spark. Figure 19a and b shows the average UE of SLIC and PASLIC-on-Spark for 481×321 pixels and 321×481 pixels, respectively, by varying the number of the target segments. The overall patterns for UE are similar to them for BR. Specifically, UE of PASLIC-on-Spark becomes similar to SLIC when the degree of overlapping increases after increasing at *overlap*0 compared to SLIC. The difference of UE between PASLIC-on-Spark and SLIC becomes less than that of UE between PSLIC-on-Spark and SLIC in the case of $p = 8$, from 6.29% to 4.19% when $overlap_h$ is maximum.

7 Conclusions

In this paper, we have presented a parallel algorithm for SLIC on Apache Spark, which we call *PSLIC-on-Spark*. To this purpose, we have extended the original SLIC algorithm to use the operations in Apache Spark, supporting its parallel processing on multiple executors in the Apache Spark cluster. Then, we have analyzed the trade-off relationship of PSLIC-on-Spark between its processing speed and accuracy due to partitioning of the original image datasets. Especially, we have identified two limitations in PSLIC-on-Spark, which degrade the accuracy of the original SLIC. Through experiments, we have verified the trade-off relationship. Specifically, we have shown that PSLIC-on-Spark using 8 CPU cores significantly reduces the processing time of SLIC by 2.24–2.93 times while it reduces the boundary recall (BR) of SLIC by 1.54–6.32% and increases under-segmentation error (UE) by 1.79–6.2%.

Then, we have proposed an improved algorithm of PSLIC-on-Spark that improves the accuracy of PSLIC-on-Spark, which we call *PASLIC-on-Spark*. We have employed two important features for PASLIC-on-Spark. It contains two main features: (1) image partitioning considering the shape and position of the clusters rather than a evenly partitioning method and (2) controllable duplication for the boundary between image partitions. Through experiments, we have shown the accuracy and efficiency of PASLIC-on-Spark on an actual cloud environment configured with 8 worker nodes using Amazon AWS. The experimental results indicate that PASLIC-on-Spark improves the accuracy of PSLIC-on-Spark by 3.66–3.77% of BR and 1.39–1.96% of UE. PASLIC-on-Spark still decreases that of processing time SLIC significantly 1.5–1.67 times on a single-node configuring using 8 CPU cores and 1.18–1.26 times on a cloud environment using 8 computing nodes.

In this paper, we have proposed parallel algorithms of SLIC based on the Apache Spark framework with a purpose for effectively operating them on a distributed environment. Recently, there have been research efforts to incorporate deep learning models to generate superpixels as described in Sect. 2. Deep learning models have advantages in terms of increasing accuracy for a target purpose because it allows to extract features suitable for given superpixels. Therefore, we plan to investigate deep learning-based superpixel segmentation algorithms based on the Apache Spark framework. It is a challenging issue because the deep learning models require high-performance processing power using GPUs in a single node while the Apache Spark framework requires scalable computing nodes in a distributed environment. Here, we aim to provide the high and robust accuracy obtained by deep learning models while maintaining the high scalability for large-scale images supported by the Apache Spark framework.

References

1. Dass P, Rajeshwar Devi S (2012) Image segmentation techniques 1. Int J Electron Commun Technol 3(1):66–70
2. Achanta R, Shaji A, Smith K, Lucchi A, Fua P, Süsstrunk S (2012) Slic superpixels compared to state-of-the-art superpixel methods. IEEE Trans Pattern Anal Mach Intell 34(11):2274–2282
3. Wang M, Liu X, Gao Y, Ma X, Soomro NQ (2017) Superpixel segmentation: a benchmark. Signal Process Image Commun 56:28–39
4. Xie X, Xie G, Xu X, Cui L, Ren J (2019) Automatic image segmentation with superpixels and image-level labels. IEEE Access 7:10–11
5. Shen Y, Ai T, Yang M (2019) Extracting centerlines from dual-line roads using superpixel segmentation. IEEE Access 7:967–979
6. Yang A, Hurt JA, Veal CT, Scott GJ (2019) "Remote sensing object localization with deep heterogeneous superpixel features," in *2019 IEEE International Conference on Big Data (Big Data)*. IEEE, pp. 5453–5461
7. Chuchvara A, Barsi A, Gotchev A (2019) Fast and accurate depth estimation from sparse light fields. IEEE Trans Image Process 29:2492–2506
8. Hossain MD, Chen D (2019) Segmentation for object-based image analysis (obia): a review of algorithms and challenges from remote sensing perspective. ISPRS J Photogramm Remote Sens 150:115–134
9. Xie X, Xie G, Xu X (2018) High precision image segmentation algorithm using slic and neighborhood rough set. Multimedia Tools Appl 77(24):525–543
10. Boemer F, Ratner E, Lendasse A (2018) Parameter-free image segmentation with slic. Neurocomputing 277:228–236
11. Qiao Y, Jiao L, Hou B (2018) High-quality depth up-sampling based on multi-scale slic. Electron Lett 54(8):494–496

12. Donné S, Aelterman J, Goossens B, Philips W (2015) “Fast and robust variational optical flow for high-resolution images using slic superpixels,” in *International Conference on Advanced Concepts for Intelligent Vision Systems*. Springer, pp. 205–216
13. Zhang K, Li T, Liu B, Liu Q (2019) Co-saliency detection via mask-guided fully convolutional networks with multi-scale label smoothing, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3095–3104
14. Zhang Y, Liu K, Dong Y, Wu K, Hu X (2019) Semisupervised classification based on slic segmentation for hyperspectral image. IEEE Geosci Remote Sens Lett. https://doi.org/10.1109/LGRS.2019.2945546
15. Crommelinck S, Bennett R, Gerke M, Koeva M, Yang M, Vosselman G (2017) Slic superpixels for object delineation from uav data. ISPRS Annal Photogram Remote Sens Spatial Inform Sci 4:9
16. Martins J, Junior JM, Menezes G, Pistori H, Sant D, Gonçalves W et al (2019) Image segmentation and classification with slic superpixel and convolutional neural network in forest context, in IGARSS 2019–2019 IEEE International Geoscience and Remote Sensing Symposium. IEEE 6543–6546
17. Vimal S, Robinson YH, Kaliappan M, Vijayalakshmi K, Seo S (2021) A method of progression detection for glaucoma using k-means and the glcm algorithm toward smart medical prediction. J Supercomput. https://doi.org/10.1007/s11227-021-03757-w
18. Alsafasfeh M, Abdel-Qader I, Bazuin B (2017) “Fault detection in photovoltaic system using slic and thermal images,” in *2017 8th International Conference on Information Technology (ICIT)*. IEEE, pp. 672–676
19. Fang Z, Zhang W, Ma H (2019) “Breast cancer classification with ultrasound images based on slic,” in *International Conference on Frontier Computing*. Springer, pp. 235–248
20. Van Etten A (2019) “Satellite imagery multiscale rapid detection with windowed networks,” in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, pp. 735–743
21. Martin D, Fowlkes C, Tal D, Malik J (2001) “A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics,” in Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001, vol. 2. IEEE, pp. 416–423
22. Ren CY, Reid I (2011) gslic: a real-time implementation of slic superpixel segmentation, University of Oxford. Department of Engineering, Technical Report, pp 1–6
23. Derksen D, Inglada J, Michel J (2019) Scaling up slic superpixels using a tile-based approach. IEEE Trans Geosci Remote Sens 57(5):3073–3085
24. Dean J, Ghemawat S (2008) Mapreduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
25. Mavridis I, Karatza H (2017) Performance evaluation of cloud-based log file analysis with apache hadoop and apache spark. J Syst Softw 125:133–151
26. Alsheikh MA, Niyato D, Lin S, Tan H-P, Han Z (2016) Mobile big data analytics using deep learning and apache spark. IEEE Netw 30(3):22–29
27. Guo R, Zhao Y, Zou Q, Fang X, Peng S (2018) Bioinformatics applications on apache spark. GigaScience 7, no. 8:giy098
28. Wu X, Liu X, Chen Y, Shen J, Zhao W (2018) A graph based superpixel generation algorithm. Appl Intell 48(11):4485–4496
29. Liu M-Y, Tuzel O, Ramalingam S, Chellappa R (2011) Entropy rate superpixel segmentation, in CVPR. IEEE 2011:2097–2104
30. Shi J, Malik J (2000) Normalized cuts and image segmentation. IEEE Trans Pattern Anal Mach Intell 22(8):888–905
31. Wang H, Shen J, Yin J, Dong X, Sun H, Shao L (2019) Adaptive nonlocal random walks for image superpixel segmentation. IEEE Trans Circ Syst Video Technol 30(3):822–834
32. Li Z, Chen J (2015) Superpixel segmentation using linear spectral clustering, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1356–1363
33. Levinshtein A, Stere A, Kutulakos KN, Fleet DJ, Dickinson SJ, Siddiqi K (2009) Turbopixels: fast superpixels using geometric flows. IEEE Trans Pattern Anal Mach Intell 31(12):2290–2297
34. Wang T, Yin J, Yang J, Liu X (2019) Image gradient-based fast superpixel segmentation algorithm for polsar images, in *2019 6th Asia-Pacific Conference on Synthetic Aperture Radar (APSAR)*. IEEE, pp. 1–6

35. Sun Z, Xuan P, Song Z, Li H, Jia R (2019) A texture fused superpixel algorithm for coal mine waste rock image segmentation. Int J Coal Preparation Util. https://doi.org/10.1080/19392699.2019.1699546
36. Sharma M, Biswas M (2021) Classification of hyperspectral remote sensing image via rotation-invariant local binary pattern-based weighted generalized closest neighbor. J Supercomput 77(7):1–34. https://doi.org/10.1007/s11227-020-03474-w
37. Zhao W, Fu Y, Wei X, Wang H (2018) An improved image semantic segmentation method based on superpixels and conditional random fields. Appl Sci 8(5):837
38. Jampani V, Sun D, Liu M-Y, Yang M-H, Kautz J (2018) "Superpixel sampling networks," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 352–368
39. Yang F, Sun Q, Jin H, Zhou Z (2020) Superpixel segmentation with fully convolutional networks, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 13 964–13 973
40. Neubert P, Protzel P (2012) "Superpixel benchmark and comparison," in *Proc. Forum Bildverarbeitung*, vol. 6, pp. 1–12
41. Prajapati HB, Vij SK (2011) "Analytical study of parallel and distributed image processing," in *2011 International Conference on Image Information Processing*. IEEE, pp. 1–6
42. Quesada-Barriuso P, Heras DB, Argüello F (2021) Gpu accelerated waterpixel algorithm for superpixel segmentation of hyperspectral images. J Supercomput 1–13
43. Salloum S, Dautov R, Chen X, Peng PX, Huang JZ (2016) Big data analytics on apache spark. Int J Data Sci Analyt 1(3–4):145–164
44. Liu B, He S, He D, Zhang Y, Guizani M (2019) A spark-based parallel fuzzy *c*-means segmentation algorithm for agricultural image big data. IEEE Access 7:169–180
45. Park G-M, Heo YS, Kwon H-Y (2021) Trade-off analysis between parallelism and accuracy of slic on apache spark, in *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE, pp. 5–12
46. Martin DR, Fowlkes CC, Malik J (2004) Learning to detect natural image boundaries using local brightness, color, and texture cues. IEEE Trans Pattern Anal Mach Intell 26(5):530–549

Authors and Affiliations

Gangmin Park[1] · Yong Seok Heo[2] · Kisung Lee[3] · Hyuk-Yoon Kwon[4]

Gangmin Park
wjdrmf314@seoultech.ac.kr

Yong Seok Heo
ysheo@ajou.ac.kr

Kisung Lee
lee@csc.lsu.edu

[1] Department of Industrial Engineering, Seoul National University of Science and Technology, Seoul, Republic of Korea

[2] Department of Electrical and Computer Engineering, Department of Artificial Intelligence, Ajou University, Suwon, Republic of Korea

[3] Division of Computer Science and Engineering, Louisiana State University, Baton Rouge, Louisiana, US

[4] Department of Industrial Engineering, Seoul National University of Science and Technology, Seoul, Republic of Korea