IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 12, 2021, accepted September 18, 2021, date of publication September 20, 2021,
date of current version September 30, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3114223

α-MeanShift++: Improving MeanShift++ for Image Segmentation

HANHOON PARK
Department of Electronic Engineering, Pukyong National University, Busan 43241, South Korea
e-mail: hanhoon.park@pknu.ac.kr

This work was supported by the National Research Foundation of Korea (NRF) Grant by the Korean Government through the MSIT under Grant 2021R1F1A1045749.

ABSTRACT MeanShift is one of the popular clustering algorithms and can be used to partition a digital image into semantically meaningful regions in an unsupervised manner. However, due to its prohibitively high computational complexity, a grid-based approach, called MeanShift++, has recently been proposed and succeeded to surprisingly reduce the computational complexity of MeanShift. Nevertheless, we found that MeanShift++ still has computational redundancy and there is room for improvement in terms of accuracy and runtime; thus, we propose an improvement to MeanShift++, named α-MeanShift++. We first attempt to minimize the computational redundancy by using an additional hash table. Then, we introduce a speedup factor (α) to reduce the number of iterations required until convergence, and we use more neighboring grid cells for the same bandwidth to improve accuracy. Through intensive experiments on image segmentation benchmark datasets, we demonstrate that α-MeanShift++ can run 4.1-4.6× faster on average (but up to 7×) than MeanShift++ and achieve better image segmentation quality.

INDEX TERMS Clustering, mean shift algorithm, MeanShift++, image segmentation.

I. INTRODUCTION

MeanShift is a classical clustering algorithm that assigns data points to the clusters iteratively by shifting points toward the mode (mode is the highest density of data points in a region). Hence, it is also known as the mode-seeking algorithm or the hill climbing algorithm [4]. MeanShift has been improved in various ways [17], [20], [22], [27]. Its representative applications in computer vision include object tracking [6] and image segmentation [18], [26], [37].

For unsupervised image segmentation, MeanShift can be preferred to simple linear iterative clustering (SLIC) [1] and Felzenszwalb [11] due to its non-parametric nature and independence on data probability distributions. However, one of the main drawbacks making it unpopular is its high computational complexity (impractical for high-resolution image segmentation). To tackle this problem, there have been several approaches to speed up the algorithm. Carreira-Perpinan [19] proposed four acceleration strategies and showed that a spatial discretization strategy can accelerate MeanShift image segmentation by up to two orders of magnitude while achieving almost the same segmentation. Wang *et al.* [34] proposed using a dual-tree to obtain a faster MeanShift approximation and Xiao and Liu [35] proposed using an adaptive Gaussian kd-tree. The tree-based approximation could significantly accelerate the neighborhood search, which is the main bottleneck of MeanShift. Duin *et al.* [9] proposed using a k-nearest neighbor (k-NN) density estimate rather than the kernel density estimate in MeanShift. In addition, the modes and gradient ascent path connected to the modes were relaxed to consist only of data points. As a result, the computational complexity of MeanShift could be significantly reduced. LeBourgeois *et al.* [16] proposed a fast algorithm that discretized the color space and used the integral image/volume. In addition, the algorithm memorized all discrete paths to mean values to avoid shifting colors along a similar path. Although the algorithm significantly reduced the runtime of MeanShift, it required a huge amount of memory.

Most recently, Jang and Jiang [13] proposed an extremely fast MeanShift algorithm, named MeanShift++, which first partitions the input space into a grid, assigns each point to its corresponding grid cell, and seeks the approximated mode from each point's and its neighboring grid cells. MeanShift++ is ideal in cases with a large number of data points but low dimensionality, such as image segmentation.

The associate editor coordinating the review of this manuscript and approving it for publication was Yizhang Jiang.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

The authors claim that MeanShift++ is more than 1,000× faster than MeanShift, but they do not provide sufficient details about the implementation.

Unfortunately, MeanShift++ is not sufficiently fast yet, without parallel processing or hardware acceleration. Through close investigation, we found that this is because MeanShift++ still has computational redundancy and there is room for further improvement in terms of accuracy and speed. For these reasons, we propose a variant of MeanShift++, named α-MeanShift++, which minimizes the computational redundancy of MeanShift++ using an additional hash table, introduces a speedup factor (α) to reduce the number of iterations required until convergence, and enables seeking more accurate modes using more numbers of neighboring grid cells while reducing the size of grid cells.

Focusing on improving MeanShift++, our contributions are as follows:

- A new clustering algorithm, α-MeanShift++, is proposed, which improves MeanShift++ in terms of runtime and image segmentation quality.
- A detailed description of Python implementation of α-MeanShift++ is provided.
- The performance of α-MeanShift++ is validated on image segmentation benchmark datasets in various aspects.

The remainder of this study is organized as follows: In Section II, related works are reviewed and the baseline algorithms, MeanShift and MeanShift++, are described. In Section III, our α-MeanShift++ algorithm is elaborated. In Section IV, the performance of α-MeanShift++ is validated on benchmark datasets and its limitations are discussed. Conclusion and future studies are presented in Section V.

II. RELATED WORK

A. UNSUPERVISED IMAGE SEGMENTATION

Image segmentation is the process of partitioning a digital image into multiple segments or labeling the pixels of objects of interest. "Unsupervised" means performing the task without prior knowledge of ground-truth. Prior to the emergence of convolutional neural networks as the main tool for image segmentation, unsupervised classical image segmentation algorithms, including k-means clustering, MeanShift [4], density-based spatial clustering of applications with noise (DBSCAN) [10], SLIC [1], fuzzy c-means clustering (FCM) [3], Felzenszwalb [11], and QuickShift [29] were used. These are typically called superpixel algorithms.

k-means is the most classical partition-based clustering algorithm [36]. It partitions data points into k clusters in which each point belongs to the cluster with the nearest mean (cluster centroid). Iteratively, the mean values of clusters are updated until convergence. Although the algorithm has been widely used for image segmentation due to its efficiency, value of k should be given in advance, and the selection of k value is very difficult to estimate.

MeanShift is a mode-seeking, density-based clustering algorithm, which is described in detail in the next section. It can partition an image into semantically meaningful regions by clustering the pixels in the image. MeanShift usually produces qualitatively good segmentations but is too slow and thus, impractical.

DBSCAN is also a density-based clustering algorithm. It defines core points, which are data points with neighbor points greater than *minPts* within a distance of *Eps*. Then, a neighborhood graph of the core points is constructed, and clusters are assigned on the basis of the connected components. DBSCAN has been effectively used for image segmentation [24]. It has better segmentation quality and time complexity than MeanShift for noisy images.

SLIC simply performs k-means clustering in the five-dimensional (5D) space of color information and pixel location, limiting the size of the search region for pixel distance computation. Due to its simplicity, it is very efficient and widely used for preprocessing of complicated algorithms for image segmentation. It is essential for this algorithm to work in Lab color space to obtain good results. The number of segments is determined by the number of centers for k-means.

FCM is similar to k-means clustering, but it allows pixels to have varying degrees of membership to multiple clusters, not assigning pixels exclusively to a single cluster. Compared to hard clustering such as k-means clustering, this soft clustering technique enables reliable image segmentation for real-world limitations such as noise, outliers, and other imaging artifacts. Nevertheless, FCM is still sensitive to noise or outliers; thus various improvements have been proposed in different areas of applications, including measuring patch-based local similarity using the structural similarity index [25], decomposing image pixels into feature spaces using tight wavelet frames [31], and integrating a residual regularization term [32].

Felzenszwalb is a graph-based image segmentation algorithm that oversegments of a multichannel (i.e., RGB) image using a fast, minimum spanning tree-based pairwise region comparison on the image grid. It runs in a time nearly linear to the number of image pixels and can run at video rates. The segment size can be controlled by a scale parameter, k [11].

QuickShift is based on an approximation of kernelized MeanShift, which is applied to the 5D space of color information and pixel location. It initializes the segmentation using MedoidShift [23], a modification of MeanShift. Then, it moves each point in the feature space to the nearest neighbor, increasing the Parzen density estimate. One of the benefits of QuickShift is that it simultaneously computes a hierarchical segmentation on multiple scales. However, it is slow and does not allow for explicit control over the size or number of segments.

QuickShift++ [15] is a recently developed algorithm, which is a modification of QuickShift. It provides initial seedings to QuickShift. The seedings are locally high-density regions that are obtained using the k-NN density estimator and the M-cores algorithm [14]. QuickShift++ produces more reasonable segmentations than QuickShift but has no improvement in terms of runtime.

DBSCAN++ [12] is a modification of DBSCAN that only requires computing point densities for a subset of data points. Compared to DBSCAN, DBSCAN++ can provide not only competitive performance but also added robustness in the bandwidth while requiring a fraction of the runtime.

MeanShift++ [13] is a fast version of MeanShift, its speed was improved by orders of magnitude and it is notably faster than QuickShift++. The algorithm is detailed in the next section.

To achieve hierarchical segmentation, agglomerative clustering can be employed, where an independent component analysis mixture model allows a better estimation of complex densities than k-NN [21].

As observed in this section, since unsupervised image segmentation algorithms have inherent weaknesses in terms of accuracy or runtime, regardless of differences in operating principles, many approaches have been continuously proposed to further improve the existing image segmentation algorithms in specific applications or by coupling one algorithm with the other.

B. MeanShift AND MeanShift++

We briefly introduce the procedure of baseline MeanShift and MeanShift++ to cluster data points $X := \{x_1, x_2, \ldots, x_n\}$, $x_i \in \mathbb{R}^d$ in Algorithms 1 and 2 [13]. At each iteration, MeanShift searches the neighbor points of each point within a radius of b (typically called bandwidth) and moves the point to the weighted mean of the neighbors until convergence. The weights w_j are typically set to 1 for simplicity [4]. However, the computational cost for the neighbor search is too high, i.e., $\mathcal{O}(n^2\ dt)$ where t represents the number of iterations of the algorithm, even if space-partitioning data structures (e.g., adaptive Gaussian kd-tree [35]) are used to speed up the search.

MeanShift++ first partitions data points into grid cells, which are hypercubes of side length h. Then, at each iteration, MeanShift++ computes the mean by moving each point using the neighbor points of the grid cell to which the point belongs and its 1-neighboring grid cells. To have the same bandwidth as MeanShift, h needs to be equal to $2b/3$. The neighbor search can be much faster because determining which cell each point belongs to is as simple as dividing the point values by h and taking the element-wise floor function, which gives a d-dimensional integer index of the grid cells. In Algorithm 2, two hash tables $\mathcal{C}$ and $\mathcal{S}$ store the count of data points belonging to each grid cell and their sum, respectively, simplifying the mean computation using neighbor points. As a result, computational complexity is reduced from $\mathcal{O}(n^2\ dt)$ to $\mathcal{O}(n3^d t)$.

III. α-MeanShift++

In this section, we present α-MeanShift++, which aims to improve MeanShift++.

Algorithm 1 MeanShift

```
Inputs: bandwidth $b$, tolerance $\eta$, $X$.
Initialize $y_{0,i} := x_i$ for $i \in [1, n]$, $t = 1$.
do
    For all $i \in [1, n]$:
        Get a set $\mathcal{N}_{t-1,i}$ which contains the neighbors of $y_{t-1,i}$ as elements.
        $y_{t,i} \leftarrow \frac{\sum_{y_{t-1,j} \in \mathcal{N}_{t-1,i}} w_j y_{t-1,j}}{\sum_{y_{t-1,j} \in \mathcal{N}_{t-1,i}} w_j}$.
    $t \leftarrow t + 1$.
while $\sum_{i=1}^{n} \left\| y_{t,i} - y_{t-1,i} \right\| \geq \eta$;
```

Algorithm 2 MeanShift++

```
Inputs: cells' side length $h$, tolerance $\eta$, $X$.
Initialize $y_{0,i} := x_i$ for $i \in [1, n]$, $t = 1$.
do
    Initialize empty hash tables $\mathcal{C}$ (stores cell counts) and $\mathcal{S}$ (stores cell sum).
    $\mathcal{C}\left(\lfloor y_{t-1,i}/h \rfloor\right) \leftarrow \mathcal{C}\left(\lfloor y_{t-1,i}/h \rfloor\right) + 1$ for $i \in [1, n]$.
    $\mathcal{S}\left(\lfloor y_{t-1,i}/h \rfloor\right) \leftarrow \mathcal{S}\left(\lfloor y_{t-1,i}/h \rfloor\right) + y_{t-1,i}$ for $i \in [1, n]$.
    Next, for all $i \in [1, n]$:
        $y_{t,i} \leftarrow \frac{\sum_{v \in \{-1,0,1\}^d} \mathcal{S}\left(\lfloor y_{t-1,i}/h \rfloor + v\right)}{\sum_{v \in \{-1,0,1\}^d} \mathcal{C}\left(\lfloor y_{t-1,i}/h \rfloor + v\right)}$.
    $t \leftarrow t + 1$.
while $\sum_{i=1}^{n} \left\| y_{t,i} - y_{t-1,i} \right\| \geq \eta$;
```

MeanShift++ still has computational redundancy. In Algorithm 2, y_t is computed for all points. However, points belonging to the same grid cell yield the same y_t value. Therefore, if y_t for a point in a grid cell has already been computed, y_ts for the other points in the same grid cell need not be computed again. Considering this, we create an additional hash table $\mathcal{U}$ in Algorithm 3. When y_t is computed for a point, we add the index of the grid cell that the point belongs to and y_t to $\mathcal{U}$. Then, if points belong to the grid cells added to $\mathcal{U}$, their y_ts are brought from $\mathcal{U}$ without re-computation.

In Algorithm 2, MeanShift++ only considers 1-neighboring grid cells to compute y_t, i.e., $v \in \{-1, 0, 1\}^d$. However, we extend to the r-neighboring grid cells, i.e., $v \in [-r, r]^d, r \geq 1$. In practice, the searching radius (=bandwidth) for neighboring points in MeanShift could be determined by h and r in MeanShift++, but r was fixed to 1. This may be because MeanShift++ works similar to MeanShift as r increases (see Fig. 1); thus, computational complexity increases in proportion to r. In α-MeanShift++, however, due to $\mathcal{U}$ and fast indexing (NumPy slicing in Sect. IV-A), using r (>1) barely increases the runtime while improving image segmentation quality. This will be detailed in the experimental results.

In MeanShift++, at iteration t, y_{t-1} is replaced by the newly-computed mean y_t, i.e., $y_t = mean(y_{t-1})$. This can

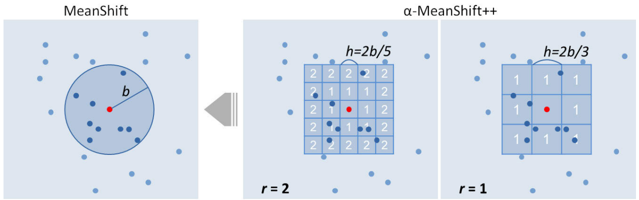

FIGURE 1. Bandwidth in MeanShift and α-MeanShift++. The red circle is the point we want to shift to the mean of its neighbors.

Algorithm 3 α-MeanShift++

```
Inputs: cells' side length $h$, tolerance $\eta$, $X$.
Initialize $y_{0,i} := x_i$ for $i \in [1, n]$, $t = 1$.
do
    Initialize empty hash tables $\mathcal{C}$ (stores cell counts), $\mathcal{S}$ (stores cell sum), and $\mathcal{U}$ (stores update values).
    $\mathcal{C}\left(\lfloor y_{t-1,i}/h \rfloor\right) \leftarrow \mathcal{C}\left(\lfloor y_{t-1,i}/h \rfloor\right) + 1$ for $i \in [1, n]$.
    $\mathcal{S}\left(\lfloor y_{t-1,i}/h \rfloor\right) \leftarrow \mathcal{S}\left(\lfloor y_{t-1,i}/h \rfloor\right) + y_{t-1,i}$ for $i \in [1, n]$.
    Next, for all $i \in [1, n]$:
        if $\lfloor y_{t-1,i}/h \rfloor \in \mathcal{U}.keys$ then
            $y_{t,i} \leftarrow \mathcal{U}\left(\lfloor y_{t-1,i}/h \rfloor\right)$.
        else
            $\hat{y}_{t,i} = \dfrac{\sum_{v \in [-r,r]^d} \mathcal{S}\left(\lfloor y_{t-1,i}/h \rfloor + v\right)}{\sum_{v \in [-r,r]^d} \mathcal{C}\left(\lfloor y_{t-1,i}/h \rfloor + v\right)}$.
            $\alpha \leftarrow 1 + \kappa e^{\gamma(1-t)}$.
            $y_{t,i} \leftarrow \alpha \hat{y}_{t,i} + (1-\alpha) y_{t-1,i}$.
            $\mathcal{U}\left(\lfloor y_{t-1,i}/h \rfloor\right) \leftarrow y_{t,i}$.
        end
    $t \leftarrow t + 1$.
while $\sum_{i=1}^{n} \left\| y_{t,i} - y_{t-1,i} \right\| \geq \eta$;
```

be rewritten as $y_t = y_{t-1} + \{mean(y_{t-1}) - y_{t-1}\}$. To reduce the number of iterations, we add a speedup factor α, which is a positive value and exponentially decays. y_t is forced to move to the direction of $mean(y_{t-1}) - y_{t-1}$, i.e., $y_t = y_{t-1} + \alpha\{mean(y_{t-1}) - y_{t-1}\}$.

As a result, the computational complexity of α-MeanShift++ is reduced to $\mathcal{O}(c(2r+1)^d t)$, where c represents the number of grid cells, i.e., $c = n/h^d$. In Algorithm 3, the difference between α-MeanShift++ and MeanShift++ is highlighted in blue.

IV. IMPLEMENTATION AND EVALUATION

A. IMPLEMENTATION DETAILS

All algorithms (MeanShift, MeanShift++, and α-MeanShift) were implemented in Python (unlike using Cython in [13]). Parallel processing was not used. Only the NumPy module [38] was used, except that the Scikit-Learn module [40] was used to accelerate searches for the nearest neighbors in MeanShift. In MeanShift++ and α-MeanShift++, C and S were created as NumPy arrays, and $\mathcal{U}$ was created as a Python dictionary. NumPy slicing was used to access a grid cell of C or S and its r-neighboring grid cells.

For image segmentation, input images were converted into three-dimensional NumPy arrays, which accept (R, G, B) color. Given that, adding spatial coordinates makes no difference in segmentation quality [13], the spatial coordinates of each pixel were not considered. For each algorithm, the returned clusters are taken as the segments. The label of each cluster depends on the color of the pixels in the cluster, which may differ from the ground-truth label.

B. EXPERIMENTAL SETUP

To evaluate the performance of α-MeanShift++, we mainly compared α-MeanShift++ with baseline algorithms, MeanShift and MeanShift++, on image segmentation. In addition, we included three popular algorithms, SLIC, Felzenszwalb, and QuickShift, from the scikit-image module [30] in the comparison. The comparison of MeanShift++ with other algorithms can also be found in a previous study [13], but not in detail. To measure the quality of segmentation results, we used the adjusted Rand index (ARI), the adjusted mutual information (AMI), the Fowlkes-Mallows index (FMI), and the mean absolute error (MAE), which are metrics provided in the Scikit-Learn module [40]. Since each segment in the segmentation results may not have the same label as the ground-truth, we did not use metrics such as the dice score for measuring classification accuracy. Instead, we employed the MAE to indirectly measure the accuracy. To compute the MAE, we colored the ground-truth images, by replacing each labeled segment with its average color (computed from the corresponding color images) using the *color.label2rgb* function in the scikit-image module [30]. We conducted experiments using the Berkeley segmentation dataset and benchmarks (BSDS500) [2] comprising 500 images, the Leeds Butterfly dataset (LBDS832) [33] comprising 832 images, and the PASCAL Visual Object Classes Challenge 2012 (VOC2012) comprising 2,913 images with ground-truth segmentations each (see Fig. 2). The LBDS832 images were rescaled to 320 × 240 or 240 × 320. We ran the algorithms on a desktop computer equipped with an i7-11700 2.5GHz CPU and 128GB RAM.

We set κ, γ, and η to 1, 1, and 20, respectively. Larger κs or smaller γs are expected to further reduce the runtime of α-MeanShift++ but may increase the runtime for some images (this will be demonstrated later). b in MeanShift was heuristically set to 30; thus, h in MeanShift++ was set to 20 (=$2b/3$). In α-MeanShift++, h and r were adjusted so that the bandwidth for mean computation was the same (=30) as in MeanShift and MeanShift++. A wider bandwidth could reduce the runtime of these algorithms, but the images were undersegmented, as shown in Fig. 3. For the optimal selection of the bandwidth, the bandwidth could be updated adaptively by computing the covariance matrix of data points [27] or the reciprocal of the local density of each point [5]; however, this is not considered in this study.

MeanShift++ originally fixed r to 1 but was modified to work with different r in our experiments to show the effect of r.

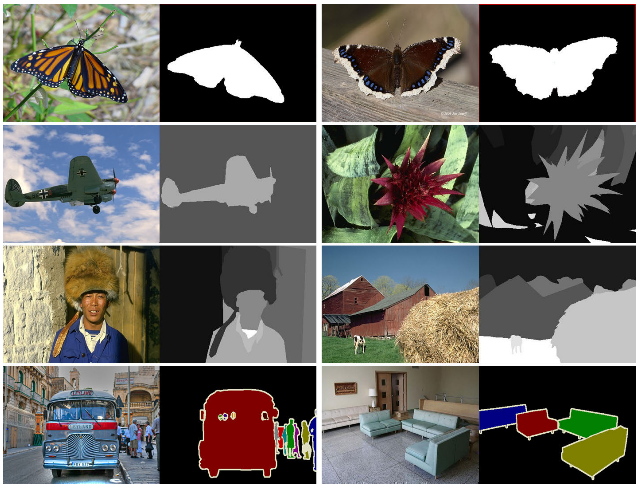

FIGURE 2. **A part of images used in our experiments and their ground-truth segmentations, where each segment has a binary, gray, or color label.**

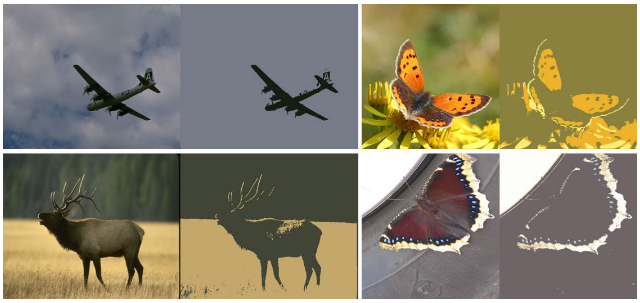

FIGURE 3. **Undersegmentation by MeanShift++ with $h = 30$. Left: original and right: segmentation results.**

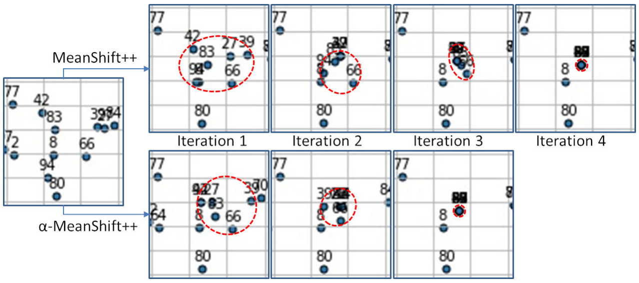

FIGURE 4. **Clustering of random points using MeanShift++ and α-MeanShift++ ($r = 1$). The red ellipses represent the distribution range of the points that will move to the same mean point. The points are clustered in fewer iterations in α-MeanShift++, i.e., 3, compared to 4 in MeanShift++. The number above each point represents its index.**

C. EXPERIMENTAL RESULTS

Before performing image segmentation, we attempted to cluster randomly-generated two-dimensional points based on their spatial coordinates, which are integer numbers ranging from 0 to 255. In Fig. 4, 100 points are generated and clustered using MeanShift++ and α-MeanShift++, where it was verified that the introduction of α can reduce the number of iterations.

Figure 5 shows the image segmentation results of several images. MeanShift achieved results visually closer to the ground-truth, whereas MeanShift++ achieved similar results in an extremely shorter time. However, runtime reduction was not as substantial as was mentioned in [13] (This may be in part because the algorithms were implemented in Cython in [13]). The proposed α-MeanShift++ achieved results identical to MeanShift++ but up to 7× faster. For most images, the segmentation quality could be improved when r was greater than 1, unlike r being fixed to 1, as in MeanShift++, as shown in Fig. 6 and Tables 1 and 2. The last column of Fig. 5 shows the segmentation results with tuned rs in α-MeanShift++, where the quality metrics were the best. SLIC and Felzenszwalb ran in just tens or hundreds of milliseconds, and QuickShift ran in a couple of seconds. However, their results were either too oversegmented or undersegmented.

As shown in Table 1, MeanShift++ was 342× faster than MeanShift and its ARI and FMI values were 0.034 and 0.019 higher, respectively. Its MAE was 0.11 lower as well. However, the runtime was still long. α-MeanShift++ was 4.6× faster than MeanShift++, with a little improvement in all quality metrics. Besides, the quality metrics notably improved with tuned rs. Note that this tuning does not cause a loss of runtime, as shown in Fig. 7. It was the same for MeanShift++, as the segmentation quality improved when r was tuned. The performance difference between the two datasets (BSDS500 and LBDS832) was unnoticeable. There was nothing to mention in the standard deviation values. There was little difference in performance variability for each algorithm. The ARI, AMI, and FMI values of SLIC, Felzenszwalb, and QuickShift were too low. In addition, their MAE values were greater than MeanShift++ or α-MeanShift++, which means that the segmentation accuracy was also worse than MeanShift++ or α-MeanShift++.

As shown in Table 2, the segmentation quality for the VOC2012 dataset was much worse than that for the other two datasets. However, the overall tendency in the performance of each algorithm was similar. MeanShift++ was 251× faster than MeanShift, and α-MeanShift++ was 4.1× faster than MeanShift++. In contrast, the segmentation quality of α-MeanShift++ was slightly worse than that of MeanShift and MeanShift++. When r was tuned, the segmentation quality of MeanShift++ and α-MeanShift++ was improved, and α-MeanShift++ had the best performance in terms of all metrics.

The quality metrics were not correlated with r when the bandwidth was the same (Fig. 6). However, all or some metrics were better when r was greater than 1. As shown in Tables 3 and 4, setting r to 1 achieved the worse results in 75%-80% of the images in terms of segmentation quality for both MeanShift++ and α-MeanShift++. The MAE values were not always correlated with the other metrics. In other words, although MeanShift++ and α-MeanShift reduced MAE values of MeanShift for some images, their ARI, AMI, and FMI values were lower than those of MeanShift, and

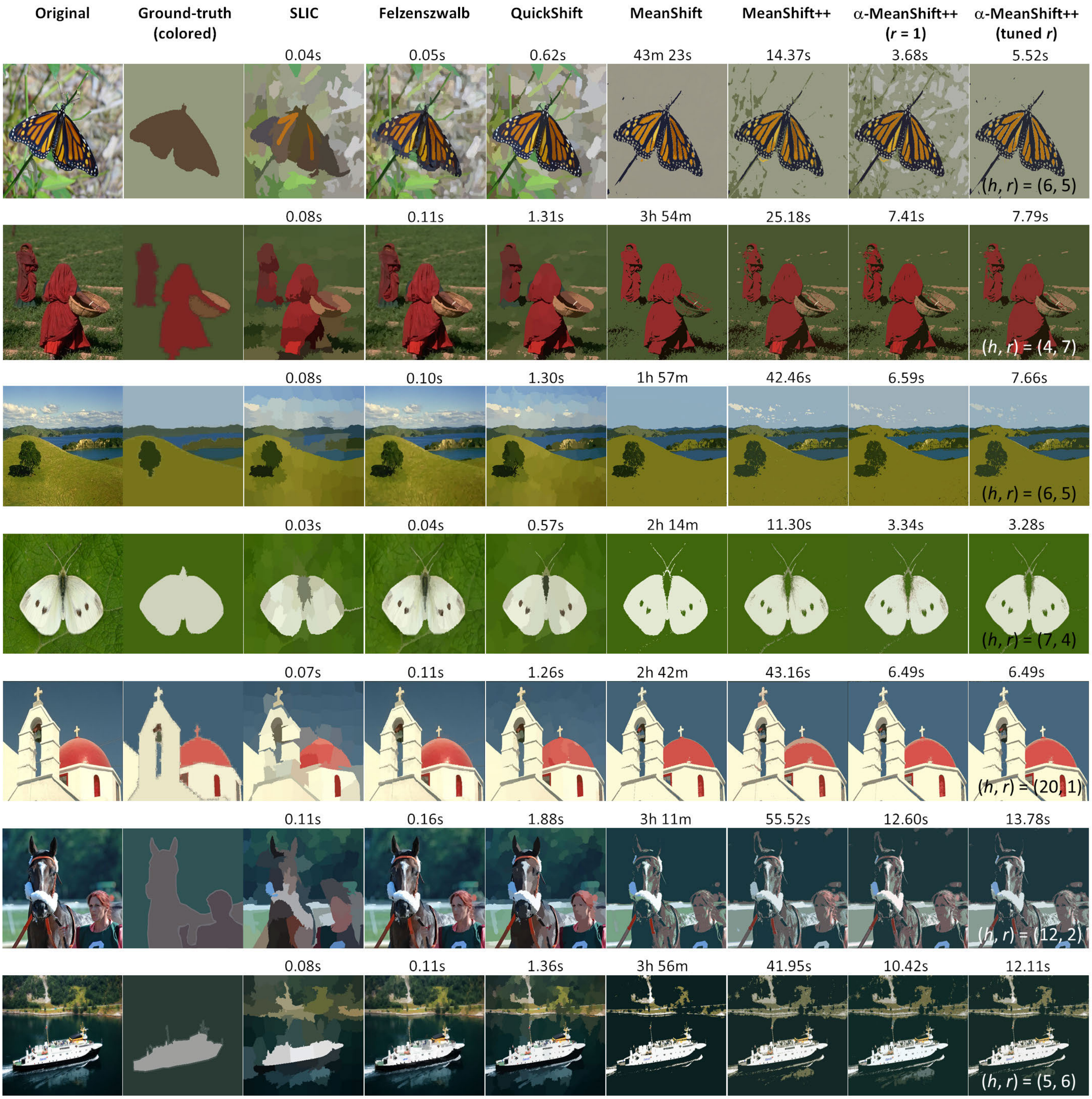

FIGURE 5. **Segmentation results of MeanShift, MeanShift++, α-MeanShift++, and the scikit-image algorithms. MeanShift yields results closer to the ground-truth but takes too long to run. MeanShift++ and α-MeanShift++ return segmentations that are similar to MeanShift with up to 600× and 2,000× speedup, respectively. In α-MeanShift++, the segmentation quality could be improved when using r greater than 1 for most images. The other algorithms run very fast but yield too over or under-segmented results.**

vice versa. Overall, MAE values were inversely correlated with the other metrics (Table 1 and Table 2).

As described in Section III, α-MeanShift++ has two speedup factors ($\mathcal{U}$ and α) to be faster than MeanShift++. As demonstrated in Fig. 7 and Tables 5 and 6, the use of $\mathcal{U}$ minimized the computational redundancy of MeanShift++ and reduced the runtime of MeanShift++ by 69.87% on the BSDS500 and LBDS832 datasets and 69.46% on the VOC2012 dataset on average when $r = 1$. When the bandwidth was the same, the runtime of MeanShift++ tended to increase in proportion to r, but the runtime with $\mathcal{U}$ remained almost constant according to r. Then, the introduction of α further reduced the runtime, although the reduction was not evident for some images. The runtime was reduced

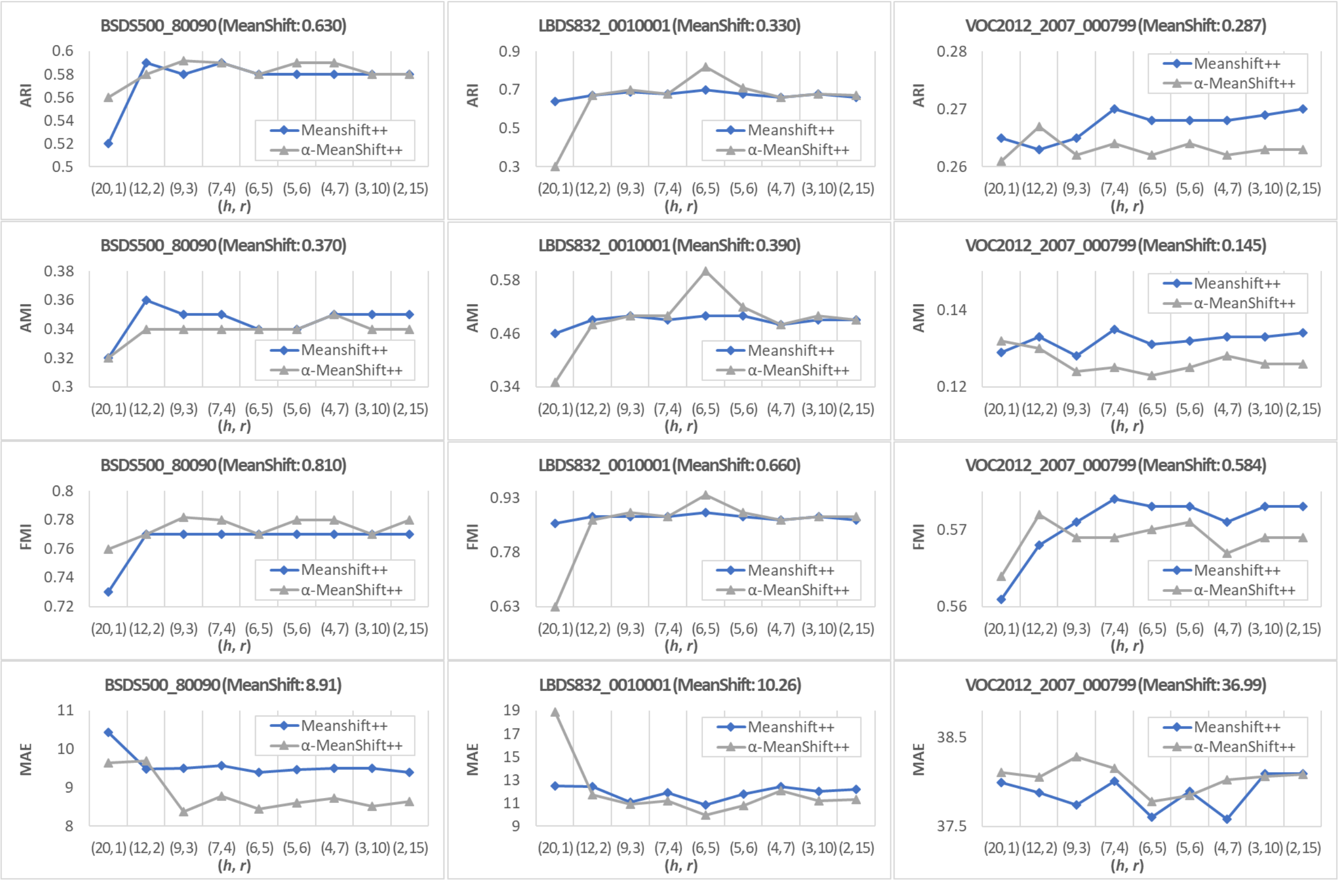

FIGURE 6. Change in the quality metrics when using different pairs of h and r with the same bandwidth. In MeanShift++, the quality metrics are not the best when $r = 1$, which is why r greater than 1 is used to improve the segmentation quality in α-MeanShift++.

TABLE 1. Runtime and segmentation quality averaged across 200 randomly selected images from the BSDS500 and LBDS832 datasets. The values to the left and right of "/" represent the mean and standard deviation, respectively.

	Runtime (s)	ARI	AMI	FMI	MAE
SLIC	0.06 / 0.02	0.069 / 0.054	0.340 / 0.111	0.210 / 0.026	15.64 / 5.66
Felzenszwalb	0.09 / 0.03	0.026 / 0.015	0.223 / 0.084	0.131 / 0.035	17.96 / 5.71
QuickShift	1.06 / 0.36	0.072 / 0.044	0.326 / 0.121	0.220 / 0.054	16.24 / 6.58
MeanShift	9,815.76 / 5265.88	0.561 / 0.252	0.499 / 0.143	0.724 / 0.191	13.57 / 9.07
MeanShift++ ($r = 1$)	28.71 / 13.92	0.595 / 0.261	0.483 / 0.143	0.743 / 0.222	13.46 / 8.11
MeanShift++ (tuned r)	29.08 / 9.83	0.620 / 0.264	0.506 / 0.145	0.758 / 0.227	12.90 / 8.30
α-MeanShift++ ($r = 1$)	6.18 / 2.34	0.604 / 0.250	0.484 / 0.137	0.745 / 0.228	13.30 / 8.48
α-MeanShift++ (tuned r)	6.50 / 2.09	0.637 / 0.267	0.517 / 0.148	0.765 / 0.231	12.72 / 8.44

TABLE 2. Runtime and segmentation quality averaged across 100 randomly selected images from the VOC2012 dataset. The values to the left and right of "/" represent the mean and standard deviation, respectively.

	Runtime (s)	ARI	AMI	FMI	MAE
SLIC	0.09 / 0.01	0.041 / 0.036	0.240 / 0.127	0.214 / 0.063	29.14 / 13.14
Felzenszwalb	0.12 / 0.02	0.011 / 0.011	0.112 / 0.063	0.131 / 0.098	36.65 / 16.03
QuickShift	1.49 / 0.17	0.027 / 0.026	0.192 / 0.104	0.178 / 0.087	36.12 / 16.04
MeanShift	10,016.07 / 3,348.28	0.277 / 0.228	0.215 / 0.188	0.605 / 0.227	39.03 / 19.89
MeanShift++ ($r = 1$)	39.81 / 10.29	0.258 / 0.205	0.229 / 0.160	0.588 / 0.218	34.53 / 16.30
MeanShift++ (tuned r)	54.26 / 19.94	0.284 / 0.212	0.238 / 0.174	0.621 / 0.205	33.91 / 17.10
α-MeanShift++ ($r = 1$)	9.63 / 2.01	0.252 / 0.222	0.226 / 0.173	0.579 / 0.225	34.98 / 17.10
α-MeanShift++ (tuned r)	10.12 / 2.39	0.283 / 0.212	0.234 / 0.175	0.620 / 0.205	33.68 / 17.72

by 22.00% on average and up to 47.91% on the BSDS500 and LBDS832 datasets and by 19.14% on average and up to 34.54% on the VOC2012 dataset, compared to before introducing α. The advantage of α is more evident in Table 7, where the number of iterations was reduced by 0.54 on average in α-MeanShift++ compared to MeanShift++.

TABLE 3. Numbers (ratios) of images with the best segmentation quality (the highest in ARI, AMI, and FMI) per r, for 200 randomly selected images from the BSDS500 and LBDS832 datasets.

r	1	2	3	4	5	6	7	10	15
MeanShift++	56 (28%)	32 (16%)	20 (10%)	20 (10%)	10 (5%)	12 (6%)	28 (14%)	10 (5%)	12 (6%)
α-MeanShift++	48 (24%)	50 (25%)	16 (8%)	8 (4%)	28 (14%)	4 (2%)	30 (15%)	6 (3%)	10 (5%)

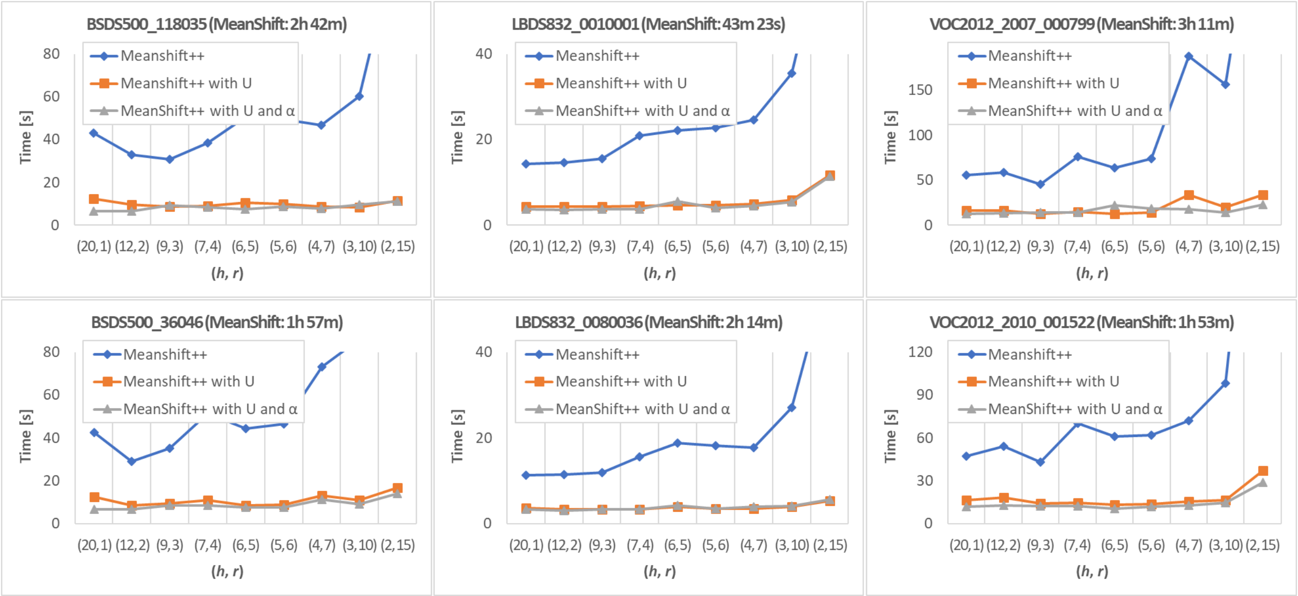

FIGURE 7. Change in the runtimes when using different pairs of h and r. The employment of $\mathcal{U}$ and α contributes to the runtime reduction in α-MeanShift++. With $\mathcal{U}$ and α, the runtime does not change much with increase in r.

TABLE 4. Numbers (ratios) of images with the best segmentation quality (the highest in ARI, AMI, and FMI) per r, for 100 randomly selected images from the VOC2012 dataset.

r	1	2	3	4	5	6	7	10	15
MeanShift++	19 (19%)	11 (11%)	11 (11%)	9 (9%)	9 (9%)	11 (11%)	19 (19%)	7 (7%)	4 (4%)
α-MeanShift++	17 (17%)	14 (14%)	10 (10%)	5 (5%)	19 (19%)	11 (11%)	16 (16%)	4 (4%)	4 (4%)

TABLE 5. Contribution of $\mathcal{U}$ and α to the runtime on the BSDS500 and LBDS832 datasets.

MeanShift++	α-MeanShift++	
	MeanShift++ with $\mathcal{U}$	MeanShift++ with $\mathcal{U}$ and α
28.71 s	8.57 s	6.18 s

TABLE 6. Contribution of $\mathcal{U}$ and α to the runtime on the VOC2012 dataset.

MeanShift++	α-MeanShift++	
	MeanShift++ with $\mathcal{U}$	MeanShift++ with $\mathcal{U}$ and α
39.81 s	12.19 s	9.63 s

The reduction was different depending on r but up to 1.58. Table 7 also shows the results for when κ was set to 0.2, 0.5, and 1.5. When κ was set to values less than 1, the number of iterations was less reduced. When κ was set to values greater than 1, α-MeanShift++ could not always reduce the number of iterations (see the results when $r = 3$). This is why we set κ to 1 in our experiments.

D. LIMITATION

For images with a wide and flat color distribution, α-MeanShift++ may require more iterations until convergence, making it slower than MeanShift++. This is an inherent problem in hill-climbing algorithms. For those images, we can make α-MeanShift++ operate similarly to MeanShift++ by setting κ to a small value and γ to a large value. Fig. 8 shows the channel-wise histograms of some images. The upper images have wide and flat histograms in all channels, whereas the lower images have sharp and split histograms. α-MeanShift++ performed well for the lower images, but it took 11.01 s (1.18 s longer than fixing α to 1, as in MeanShift++) for the upper-left image when $h = 20$ and $r = 1$, and 11.66 s (2.14 s longer than fixing α to 1, as in MeanShift++) for the upper-right image when $h = 6$ and $r = 5$. However, by setting κ to 0.5, the runtimes were reduced to 8.62 s and 8.66 s, respectively, which were 1.21 s and 0.86 s shorter than fixing α to 1. To investigate how many images in each dataset have a wide, flat color distribution, we calculated the entropy of the color histograms of images using the *stats.entropy* function in the SciPy module [41]. In our experiments, wide, flat color histograms corresponded to entropy values greater than 5.3. On the BSDS500, LBDS832, and VOC2012 datasets, 13, 63, and 462 images had such wide, flat histograms in all color channels, respectively.

TABLE 7. Average iterations until convergence per r for 200 randomly selected images from the BSDS500 and LBDS832 datasets.

	r	1	2	3	4	5	6	7	10	15
κ=1.5	MeanShift++	8.32	8.55	8.23	8.80	8.34	8.67	9.01	9.03	9.77
	α-MeanShift++	8.28	8.09	8.74	8.24	8.29	8.55	8.72	8.49	8.87
κ=1	MeanShift++	8.40	8.51	7.96	8.51	8.26	8.3	9.61	8.45	8.59
	α-MeanShift++	8.19	7.68	7.66	7.98	8.09	7.68	8.03	7.98	8.48
κ=0.5	MeanShift++	8.36	8.20	8.08	8.34	8.10	8.42	8.61	8.85	9.59
	α-MeanShift++	7.85	7.94	7.65	7.67	7.96	8.12	8.38	8.52	9.13
κ=0.2	MeanShift++	8.69	8.65	8.22	8.70	8.52	8.53	9.37	8.89	9.14
	α-MeanShift++	8.65	8.42	8.20	8.28	7.98	8.18	8.72	8.66	9.14

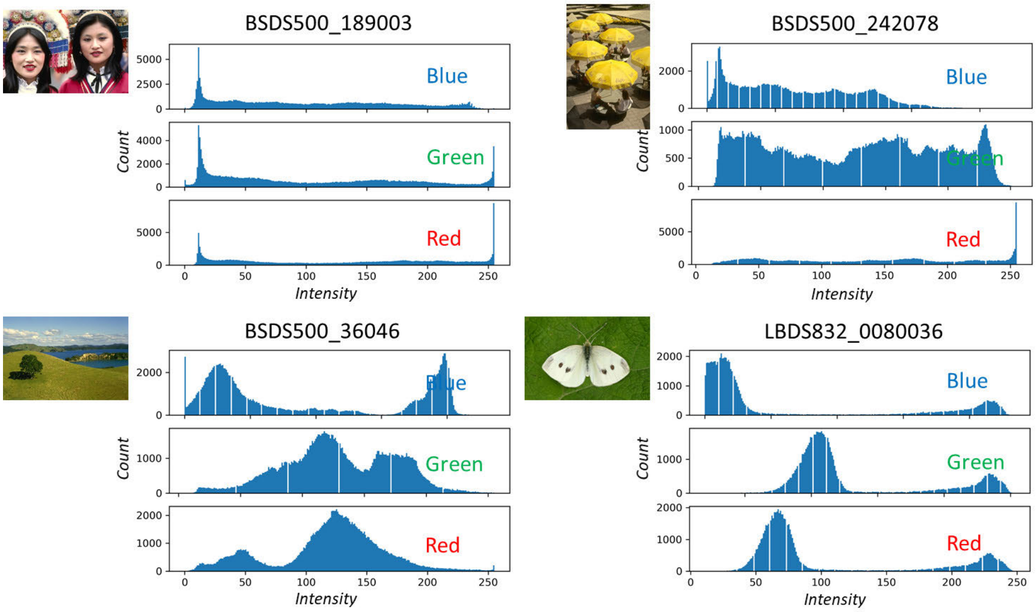

FIGURE 8. Channel-wise histograms of images. α-MeanShift++ may work incorrectly for the upper images with wide and flat histograms.

In Fig. 8, the upper-left image had entropy values of 5.35, 5.33, and 5.29 in blue, green, and red channels, respectively. In contrast, the lower-right image had entropy values of 4.41, 4.56, and 4.56 in blue, green, and red channels, respectively.

V. CONCLUSION

This study proposed a new clustering algorithm, α-MeanShift++, which is a modification of MeanShift++. It reduced the computational redundancy of MeanShift++ and introduced a factor α to accelerate convergence. α-MeanShift++ was 4.1-4.6× faster than MeanShift++, depending on image segmentation benchmark datasets, without parallel processing on CPU. In addition, α-MeanShift++ allowed using more neighbor grid cells of smaller sizes with the same bandwidth. With the number of neighbor grid cells tuned, α-MeanShift++ significantly improved the segmentation quality of MeanShift++.

Future studies will focus on optimizing the proposed algorithm in terms of speed to make it more practical (e.g., implementing it in Cython or C/C++ [8] and parallelizing it to run on graphics processing units [28] or field-programmable gate arrays [7]. In addition, to overcome the limitations mentioned in Section IV-D, we plan to adjust α more precisely by analyzing the color distribution of images (e.g., using histogram entropy). In this study, we showed that using r greater than 1 improves the performance of MeanShift++, but we did not provide a method for determining the optimal value per image. Thus, we are planning a future study to determine the optimal value of r.

REFERENCES

[1] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "SLIC superpixels compared to state-of-the-art superpixel methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012.

[2] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011.

[3] J. C. Bezdek, W. Full, and R. Ehrlich, "FCM: The fuzzy *c*-means clustering algorithm," *Comput. Geosci.*, vol. 10, nos. 2–3, pp. 191–203, 1984.

[4] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 8, pp. 790–799, Aug. 1995.

[5] D. Comaniciu, V. Ramesh, and P. Meer, "The variable bandwidth mean shift and data-driven scale selection," in *Proc. 8th IEEE Int. Conf. Comput. Vis. (ICCV)*, Jul. 2001, pp. 438–445.

[6] D. Comaniciu, V. Ramesh, and P. Meer, "Kernel-based object tracking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 5, pp. 564–577, May 2003.

[7] S. Craciun, R. Kirchgessner, A. D. George, H. Lam, and J. C. Principe, ''A real-time, power-efficient architecture for mean-shift image segmentation,'' *J. Real-Time Image Process.*, vol. 14, no. 2, pp. 379–394, Feb. 2018.

[8] D. Demirović, ''An implementation of the mean shift algorithm,'' *Image Process. Line*, vol. 9, pp. 251–268, Sep. 2019.

[9] R. P. Duin, A. L. Fred, M. Loog, and E. Pekalska, ''Mode seeking clustering by kNN and mean shift evaluated,'' in *Proc. Joint Int. Workshops Stat. Techn. Pattern Recognit. Struct. Syntactic Pattern Recognit.*, 2012, pp. 51–59.

[10] M. Ester, H. Kriegel, J. Sander, and X. Xu, ''A density-based algorithm for discovering clusters in large spatial databases with noise,'' in *Proc. 2nd Int. Conf. Knowl. Discov. Data Mining*, 1996, pp. 226–231.

[11] P. F. Felzenszwalb and D. P. Huttenlocher, ''Efficient graph-based image segmentation,'' *Int. J. Comput. Vis.*, vol. 59, no. 2, pp. 167–181, Sep. 2004.

[12] J. Jang and H. Jiang, ''DBSCAN++: Towards fast and scalable density clustering,'' in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3019–3029.

[13] J. Jang and H. Jiang, ''MeanShift++: Extremely fast mode-seeking with applications to segmentation and object tracking,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2021, pp. 4102–4113.

[14] H. Jiang and S. Kpotufe, ''Modal-set estimation with an application to clustering,'' in *Proc. Int. Conf. Artif. Intell. Statist.*, 2017, pp. 1197–1206.

[15] H. Jiang, J. Jang, and S. Kpotufe, ''Quickshift++: Provably good initializations for sample-based mean shift,'' in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2299–2308.

[16] F. LeBourgeois, F. Drira, D. Gaceb, and J. Duong, ''Fast integral Mean-Shift: Application to color segmentation of document images,'' in *Proc. Int. Conf. Document Anal. Recognit.*, Aug. 2013, pp. 52–56.

[17] J. N. Myhre, K. O. Mikalsen, S. Løkse, and R. Jenssen, ''Robust clustering using a knn mode seeking ensemble,'' *Pattern Recognit.*, vol. 76, pp. 491–505, Apr. 2018.

[18] J. H. Park, G. S. Lee, and S. Y. Park, ''Color image segmentation using adaptive mean shift and statistical model-based methods,'' *Comput. Math. with Appl.*, vol. 57, no. 6, pp. 970–980, Mar. 2009.

[19] M. A. Carreira-Perpinan, ''Acceleration strategies for Gaussian mean-shift image segmentation,'' in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, 2006, pp. 1160–1167.

[20] Y. Ren, U. Kamath, C. Domeniconi, and G. Zhang, ''Boosted mean shift clustering,'' in *Proc. Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, 2014, pp. 646–661.

[21] A. Salazar, J. Igual, G. Safont, L. Vergara, and A. Vidal, ''Image applications of agglomerative clustering using mixtures of non-Gaussian distributions,'' in *Proc. Int. Conf. Comput. Sci. Comput. Intell.*, 2015, pp. 459–463.

[22] H. Sasaki, A. Hyvärinen, and M. Sugiyama, ''Clustering via mode seeking by direct estimation of the gradient of a log-density,'' in *Proc. Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, vol. 2014, pp. 19–34.

[23] Y. A. Sheikh, E. A. Khan, and T. Kanade, ''Mode-seeking by medoidshifts,'' in *Proc. IEEE 11th Int. Conf. Comput. Vis.*, Oct. 2007, pp. 1–8.

[24] J. Shen, X. Hao, Z. Liang, Y. Liu, W. Wang, and L. Shao, ''Real-time superpixel segmentation by DBSCAN clustering algorithm,'' *IEEE Trans. Image Process.*, vol. 25, no. 12, pp. 5933–5942, Dec. 2016.

[25] Y. Tang, F. Ren, and W. Pedrycz, ''Fuzzy C-means clustering through SSIM and patch for image segmentation,'' *Appl. Soft Comput.*, vol. 87, Feb. 2020, Art. no. 105928.

[26] W. Tao, H. Jin, and Y. Zhang, ''Color image segmentation based on mean shift and normalized cuts,'' *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 37, no. 5, pp. 1382–1389, Oct. 2007.

[27] Y. Tian and Y. Yokota, ''Estimating the major cluster by mean-shift with updating kernel,'' *Mathematics*, vol. 7, no. 9, p. 771, Aug. 2019.

[28] B. Varga and K. Karacs, ''High-resolution image segmentation using fully parallel mean shift,'' *EURASIP J. Adv. Signal Process.*, vol. 2011, no. 1, p. 111, Dec. 2011.

[29] A. Vedaldi and S. Soatto, ''Quick shift and kernel methods for mode seeking,'' in *Proc. Eur. Conf. Comput. Vis.*, 2008, pp. 705–718.

[30] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, and T. Yu, ''Scikit-image: Image processing in Python,'' *PeerJ*, vol. 2, p. e453, Jun. 2014.

[31] C. Wang, W. Pedrycz, J. Yang, M. Zhou, and Z. Li, ''Wavelet frame-based fuzzy C-means clustering for segmenting images on graphs,'' *IEEE Trans. Cybern.*, vol. 50, no. 9, pp. 3938–3949, Sep. 2020.

[32] C. Wang, W. Pedrycz, Z. Li, and M. Zhou, ''Residual-driven fuzzy C-means clustering for image segmentation,'' *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 4, pp. 876–889, Apr. 2021.

[33] J. Wang, K. Markert, and M. Everingham, ''Learning models for object recognition from natural language descriptions,'' in *Proc. Brit. Mach. Vis. Conf.*, 2009, pp. 2.1–2.11.

[34] P. Wang, D. Lee, A. Gray, and J. M. Rehg, ''Fast mean shift with accurate and stable convergence,'' in *Proc. Int. Conf. Artif. Intell. Statist.*, 2007, pp. 604–611.

[35] C. Xiao and M. Liu, ''Efficient mean-shift clustering using Gaussian KD-tree,'' *Comput. Graph. Forum*, vol. 29, no. 7, pp. 2065–2073, Sep. 2010.

[36] X. Zheng, Q. Lei, R. Yao, Y. Gong, and Q. Yin, ''Image segmentation based on adaptive K-means algorithm,'' *EURASIP J. Image Video Process.*, vol. 2018, no. 1, pp. 1–10, Dec. 2018.

[37] H. Zhou, X. Wang, and G. Schaefer, ''Mean shift and its application in image segmentation,'' in *Innovations in Intelligent Image Analysis* (Studies in Computational Intelligence), vol. 339. Berlin, Germany: Springer, 2011, pp. 291–312.

[38] *NumPy*. Accessed: Aug. 9, 2021. [Online]. Available: https://numpy.org/doc/stable/index.html

[39] *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. Accessed: Aug. 20, 2021. [Online], Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2012

[40] *Scikit-Learn*. Accessed: Aug. 9, 2021. [Online]. Available: https://scikit-learn.org/stable/

[41] *SciPy*. Accessed on: Aug. 20, 2021. [Online]. Available: https://www.scipy.org/scipylib/index.html

HANHOON PARK received the B.S., M.S., and Ph.D. degrees in electrical and computer engineering from Hanyang University, Seoul, South Korea, in 2000, 2002, and 2007, respectively. From 2008 to 2011, he was a Postdoctoral Researcher with NHK Science and Technology Research Laboratories, Tokyo, Japan. In 2012, he joined the Department of Electronic Engineering, Pukyong National University, Busan, South Korea, where he is currently a Professor. His current research interests include augmented reality, human–computer interaction, and deep learning application.