FastSLIC: Optimized SLIC Superpixel

Alchan Kim
Seoul National University
Dept. of Computer Science and Engineering
a9413@snu.ac.kr

Abstract

Superpixels are regions grouped by similarities such as color, texture and proximity, which are used in several computer vision tasks such as object detection, semantic segmentation, video tracking and selective search framework. Many superpixel algorithms have been developed, among which SLIC (Simple Linear Iterative Clustering) is one of the most commonly used algorithms because of its simplicity and low computational cost. However, the existing methods fail to be used in real-time environment, because they either have high latency or sacrifice accuracy for speed. In this paper, we present FastSLIC, an optimized variant of SLIC. We propose color quantization, integer-only arithmetic for clustering and row subsampling for computational efficiency. We also devise a tiling scheme for multicore parallelization. As a result, we observe it runs 10×-33× *faster than the conventional SLIC and outperforms the existing methods. Furthermore, we observe the performance of our method is comparable to that of a GPU implementation of SLIC. We evaluate it on BSDS500 dataset in three accuracy metrics and observe it retains the accuracy of the original SLIC.*

1 Introduction

Superpixels are regions grouped by similarities such as color, texture and proximity. They are used in object detection [33], semantic segmentation [7, 24], video tracking [8, 9, 31] and selective search framework [28]. Using superpixels can greatly reduce the computational cost by grouping hundreds of thousands of pixels to a few thousand superpixels, making it an attractive technique when low processing time is required.

Many superpixel extraction methods fall into two categories: the graph-based methods and the gradient-based methods. The graph-based methods represent each pixel as a vertex in a graph and similarity between neighboring pixels as an edge. Normalized Cut [26] divides a graph to multiple subgraphs by minimizing intra-group association cost and maximizing inter-group association cost. Felzenszwalb-Huttenlocher [5] starts from pixel-wise clusters and merge neighboring clusters in the bottom-up fashion. The gradient-based methods refine trainable parameters toward local maxima or local mimima, which is often performed iteratively until convergence. SLIC [3] is based on k-means clustering algorithm [17, 18] on 5-dimensional vectors on both color and position. LSC [15] performs clustering similar to SLIC, but in a high-dimensional feature space. SEEDS [29] uses iterative hill-climbing optimization for finding partitioning maximizing the energy function. Methods using other approaches were also developed, such as watershed transform [21, 30] and sampling from quantized region [12, 13]. Recently, several methods leveraging deep learning method were developed for more accurate superpixels [14, 27].

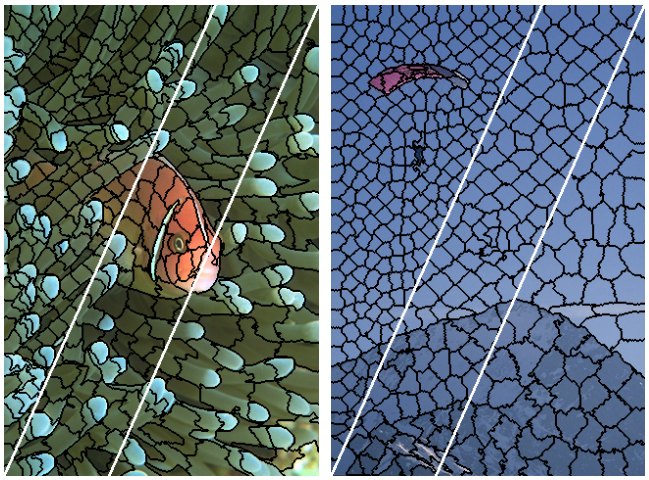


Figure 1. Examples of superpixel segmentation.

While superpixels can reduce the time taken by subsequent processes in pipeline, there is one drawback of using them: the extraction itself takes a significant amount of time. This makes it hard to utilize superpixels in real-time environments where extremely low latency is required. While several fast algorithms have been proposed for this purpose, they often come with low accuracy.

We propose **FastSLIC**, an optimized variant of SLIC, which uses quantization, subsampling and parallelization in order to reduce the latency. Our method runs 10 to 33 times faster than the conventional SLIC without any compromise of accuracy. To our best knowledge, the proposed method is the state-of-art superpixel extraction method in terms of running time and its performance is even comparable to a GPU implementation.

We make the following contributions:

- We provide a quantization scheme and a new distance metric based on 16-bit and 8-bit integer-only arithmetic, which allows for more efficient SIMD parallelization and less computational cost.
- We propose row subsampling scheme which substantially reduces latency without accuracy loss.
- We devise a tiling scheme to safely parallelize the cluster assignment, a bottleneck of our algorithm.

- We release the code as a python package so it can be easily used in prototyping. The code is published in github and publicly available. [1]

2 Related Work

Neubert et al. [20] shows that the running times of superpixel algorithms differ in at most five orders of magnitudes. Several fast methods have been proposed in literature.

Lightweight algorithms Watershed [30] is the one of fastest segmentation method generating segmentation by flooding an image surface. However, it has a drawback that shapes and sizes of the extracted superpixels tend to be irregular. Compact Watershed [21] addresses this problem by incorporating a spatial distance term to seed points, in the similar way to SLIC. USEQ [12] and USEAQ [13] achieve fast superpixel extraction via sampling from quantized region, which also support multi-core parallelization in their implementations.

SLIC-variants Preemptive-SLIC [21] is a novel extension of SLIC in which a cluster is not updated after it is stabilized and no more number of associated pixels than threshold are updated. SNIC [4] makes use of a priority queue to visit each pixel only once to achieve low latency and high accuracy.

Dedicated hardware There have been attempts to run SLIC on hardware other than CPUs. gSLICr [23] is a GPU implementation of SLIC running over 250hz. In addition, a hardware accelerator for SLIC is proposed in [11]. Despite their low latency, they often require expensive specialized hardware and are not suitable to general-purpose machines such as mobile devices.

3 Background

3.1 SLIC

SLIC (Simple Linear Iterative Clustering Algorithm) [3] is one of the most commonly used superpixel algorithms because of its simplicity and low computational cost. It is based on k-means clustering in 5-dimension feature space: 2 dimensions for y-x coordinates of a pixel and 3 dimensions for three color intensities of the pixel in CIELAB color space.

K-means clustering aims for minimizing the objective function, the sum of euclidean distances between each centroid and the points assigned to it. SLIC performs K-means clustering on pixels and centroids placed on a CIELAB image, as illustrated in Figure 2. It puts seed centroids on a regular grid initially and then it repeats to alternate cluster assignment and centroid update until clusters converge or max iteration reached. In cluster assignment phase, each pixel is assigned to the closest centroid. In cluster update phase, each centroid is moved to the center of feature vectors of pixels that belong to the centroid.

[1] https://github.com/Algy/fast-slic

Because the clustering does not guarantee pixel connectivity, a few stray pixels may remain [3]. To address this, Connected Component Labeling (CCL), a procedure to assign a unique label to each connected component in image [32], can be used to search for small connected components in a cluster label map. The original SLIC implementation incorporates a simple one-pass CCL for locating small isolated blobs and enforce connectivity by merging them into neighboring components.

The key to the computational efficiency of SLIC is that it only searches a limited area for candidate pixels of a centorid. Let K, H, W be the number of superpixels, height, and width of an image, respectively. The approximate width of a square superpixel is roughly given by eq. (1). Only pixels within $(2S + 1) \times (2S + 1)$ area around the center of cluster are considered.

$$S = \sqrt{\frac{W \times H}{K}} \tag{1}$$

In the original paper, a distance metric between a pixel i and a centroid k is a normalized euclidean distance, defined as eq. (2), where $[y_k, x_k, l_k^*, a_k^*, b_k^*]^T$ is the feature vector of centroid and $[y_i, x_i, l_i^*, a_i^*, b_i^*]^T$ is the feature vector of pixel i [2]. m is *compactness*, a normalizing factor for color distance. The higher the value of m is, the more compact a superpixel becomes.

$$\begin{aligned} D_{ik} = &\sqrt{(l_k^* - l_i^*)^2 + (a_k^* - a_k^*)^2 + (b_k^* - b_k^*)^2} \\ &+ \frac{m}{S}\sqrt{(y_k - y_i)^2 + (x_k - x_k)^2} \end{aligned} \tag{2}$$

The formula similar to eq. (3) is used instead in the other work of the authors of SLIC [3, 4]. It has a benefit on performance as expensive *sqrt* function is avoided.

$$\begin{aligned} D_{ik}^2 = &(l_k^* - l_i^*)^2 + (a_k^* - a_k^*)^2 + (b_k^* - b_k^*)^2 \\ &+ \frac{m^2}{S^2}((y_k - y_i)^2 + (x_k - x_k)^2) \end{aligned} \tag{3}$$

3.2 Parallelization

SIMD(Single Instruction, Multiple Data) is a parallelization paradigm where multiple elements are packed into a vector and the same operation on multiple elements can be performed at once. Depending on the actual architecture and the precision used, SIMD extensions can boost the potential peak performance by up to a factor of four in case of SSE [6].

AVX2 and NEON are used in this research, both of which are popular choices for SIMD parallelization in their corresponding architectures. AVX2 is a SIMD architecture for x64, which comes up with 256-bit vector instructions. ARM

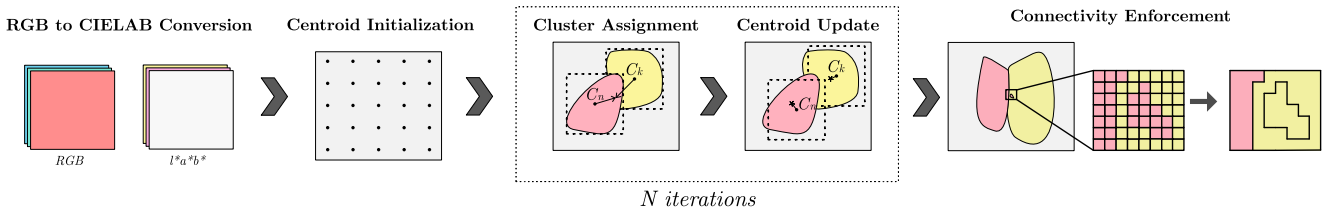


Figure 2. Overview of SLIC superpixel extraction.

NEON is a SIMD architecture with 128-bit vector instructions, which was introduced to ARMv7-A and ARM7-R profiles [1]. Not only does NEON yield 1.6x-2.5x speed gain on complex video codecs, it also has a benefit of saving power [22].

Distributing workloads to multiple cores has been a major strategy to further improve performance. While some problems such as matrix multiplication and monte-carlo simulation are embarrassingly parallel, meaning it is so easy to evenly distribute workload onto multiple cores by its nature, other problems are so inherently serial that it cannot be parallelized easily without alteration to its core mechanism. One of the problems with the existing superpixel algorithms is that they are either inherently serial or not optimized well to a multi-core cpu. We present a novel tiling method to address this issue in order to run SLIC on multiple cores.

4 FastSLIC

Our algorithm runs an order of magnitudes faster than the original SLIC by introducing several modifications. In this section, those improvements are described. In the rest of paper, it is assumed that an array index is zero-based and a two-dimensional array is row major.

We optimize the following bottlenecks: CIELAB conversion and quantization, cluster assignment, centroid update and connectivity enforcement.

We first convert a RGB image to a quantized CIELAB image represented with unsigned 8-bit integers. Then, we repeat to assign each pixel to a cluster and update each centroid to a mean vector of their member pixels. After max iteration reached, pixel connectivity of cluster label map is enforced. The brief description of our algorithm is shown in Algorithm 1.

4.1 Integer-only arithmetic and Quantization

Integer-only arithmetic with quantization is advantageous in regards to both SIMD vectorization and computational efficiency. First, when it comes to SIMD, the less element integer size, the more elements a vector contains, implying more elements can be computed at the same time. In the case, performance gain is easily achievable simply by using lower integer size. Second, accessing to small-sized array results in less memory footprint, contributing to less chance of cache miss. Third, integer-only arithmetic itself is efficient, especially on integer-only hardware.

In this section, we describe quantization method and how computation is carried out with integer-only arithmetic. CIELAB color space is a 3-dimensional color space where the same distance between any pair of points in the space corresponds to the same amount of visually perceived change. A point in the color space is represented as a vector of three real-valued elements: $[l^*, a^*, b^*]^T$, where l^* is for the lightness and both a^* and b^* are for the color information. The range of each component is: $l^* \in [0, 100]$, $a^* \in [-128, 127]$ and $b^* \in [-128, 127]$.

We adopt the following quantization scheme:

$$l = quantize\_to\_u8(l^* * 2^n) \tag{4}$$

$$a = quantize\_to\_u8((a^* + 128) * 2^n) \tag{5}$$

$$b = quantize\_to\_u8((b^* + 128) * 2^n) \tag{6}$$

where $quantize\_to\_u8$ is a function that clips a value into $[0, 255]$ and drops decimal places so the value can be represented with an 8-bit unsigned integer. While high n truncates high value into the maximum value (i.e. 255), low n cannot represent decimal places precisely. We set $n = 1$ in the rest of the paper because it is observed to have a negligible impact on accuracy.

By applying quantization, the elements of the centroid k and pixel i, $[x_k, y_k, l_k, a_k, b_k]^T$ and $[x_i, y_i, l_i, a_i, b_i]^T$, are represented with integers. To keep this property, in the centroid update phases, each centroid moves to rounded integers of its center of mass.

A distance value is also represented with 16-bit unsigned integers instead of 32-bit floating-point numbers. While the conventional SLIC uses euclidean distance, when it comes to integer-only arithmetic, it is inefficient because it requires wider integers to store intermediate values. Computing euclidean distance involves multiplication of two 16-bit integers, a result of which should be represented with a 32-bit integer.

Due to the fact, we use manhattan distance to measure a distance between a centroid and a point, which is defined as the sum of absolute differences between elements of the two vectors. Not only does it keeps variable size same during all computation, it is also computationally more efficient because only addition, subtraction and conditional move are

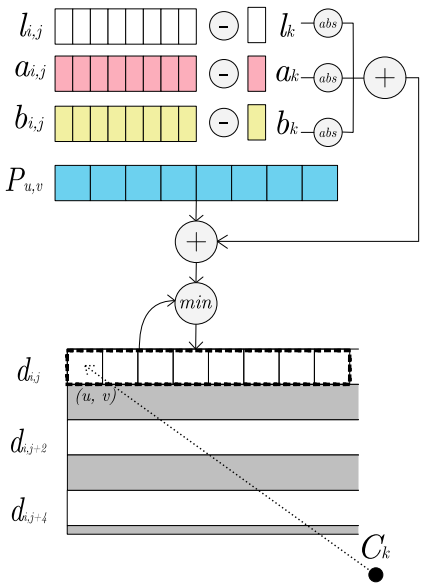


Figure 3. An illustration of vectorized cluster assignment with integer-only arithmetic. d_{ij}, the minimum distance of pixel at (i, j), is updated to distance to each centroid $C_k = \{x_k, y_k, l_k, a_k, b_k\}$ given it has less value. $P_{u,v}$ denotes a spatial distance at (u, v) from the centroid. Row subsampling is applied so that only a subset of rows is processed.

involved, comparing to euclidean distance where multiplications and wide variables are required.

We define the color distance term D^c_{ik} and the spatial distance term D^s_{ik} between the point i and the centroid k as follows:

$$D^c_{ik} = |l_i - l_k| + |a_i - a_k| + |b_i - b_k| \tag{7}$$

$$D^s_{ik} = \lfloor \frac{m}{S}(|x_i - x_k| + |y_i - y_k|) \rceil * 2^n \tag{8}$$

D^s is multiplied by 2^n in order to fit the scale to that of quantized color. The distance metric D_{ik} is a sum of those two distance terms.

$$D_{ik} = D^c_{ik} + D^s_{ik} \tag{9}$$

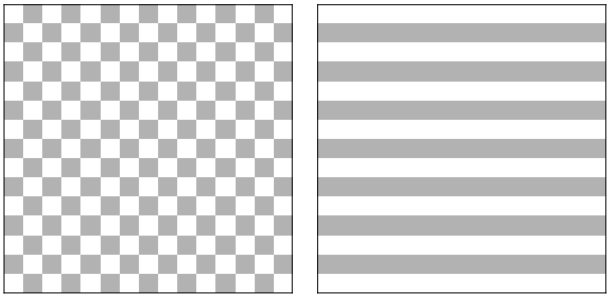
Since D^s_{ik} is a function of $x_i - x_k$ and $y_i - y_k$ whose values are constrained to the integer range of $[-S, S]$, the spatial distance term can be cached in advance in a 2D array P of size $(2S + 1) \times (2S + 1)$. By caching D^s into P, computing spatial distance, where floating-point multiplication and division are involved, can be replaced by one memory access.

The actual formulations used in our implementation are as follows:

$$P_{uv} = \lfloor \frac{m}{S}(|u| + |v|) \rceil * 2^n \tag{10}$$

$$D_{ik} = D^c_{ik} + P_{uv} \tag{11}$$

where $u = y_i - y_k$ and $v = x_i - x_k$.



(a) pixel subsampling **(b)** row subsampling

Figure 4. An illustration of two subsampling strategies. The shaded regions are discarded array elements by sampling. A cache line or a SIMD vector from (a) contains both sampled elements and discarded elements, which results in inefficient cache use and no decrease in the number of SIMD vector instructions. (b) does not have this drawback because a row is a long contiguous chunk of memory.

P_{uv} has a range of $[0, 2m * 2^n]$ and D^c has a range of $[0, 765]$. A distance fits in 16-bit integer unless $m \geq 16384$, as an unsigned 16-bit integer has the maximum value $2^{16} - 1$. Thus, there are no need to clip the value of P in practice considering m is actually set to a small value such as 10 or 20.

Figure 3 depicts the cluster assignment procedure using integer-only arithmetic. we maintain a minimum distance d_{ij} to its cluster during an iteration. For each cluster k, d_{ij} is updated into distance to the centroid if it has less value. For further optimization, we employ SIMD vectorization and row subsampling, which will be described in Section 4.2.

We represent cluster labels with unsigned 16-bit integers as well. Reserving $0xFFFF$ as the sentinel value indicating *Unassigned*, at most 65535 clusters can be allocated in our method.

4.2 Row subsampling

Subsampling is an effective way to speed up clustering by reducing memory access. For example, the running time of K-means clustering to large dataset is shown to be reduced by orders of magnitudes by sampling [25].

The running time of both cluster assignment and centroid update can be reduced by the subsampling as only a portion of pixels can be used for scattering cluster assignment and gathering pixels for computing new centroid position.

The natural way to sample pixels from an image is to treat each pixel as an independent unit and draw p proportion of pixels from the image. For example, when $p = 0.5$, one could randomly accept each pixel with probability $p = 0.5$. Otherwise, one could divide pixels in checkerboard pattern and alternate between white and black, as depicted in Figure 4a.

This method will be called *Pixel subsampling* in that each pixel represents a unit of sampling.

While running time is shown to be reduced by pixel subsampling [11], it should not be strictly inversely proportional to the the number of pixels. That is, performance gain is much less than 2× when $p = 0.5$. The main disadvantage of the method is that not all elements in memory cache line are utilized, resulting in enlarging the proportion of time taken by cache miss. Even worse is SIMD implementation of this method, because a vector packs contiguous multiple elements, vector instruction cannot be skipped out unless all elements in a vector are rejected by sampling.

To address this, *Row subsampling* is proposed in this paper. A sample is drawn from a set of image rows and then only the rows from the sample are used for cluster assignment and centroid update. Because a row of an image is a long contiguous line of memory, processing a row subsample results in less cache miss and is easy to be optimized with SIMD.

Specifically, in each iteration i, rows whose index y satisfies $y \mod Stride \equiv i$ are sampled. For instance, if the value of *Stride* is 2, it first samples even-numbered rows and then odd-numbered rows in the alternate fashion. Centroid updates are performed in the same way as the standard k-means clustering, without weighted update or gradient step.

Algorithm 1: FastSlic

Input: $W \times H$ quantized CIELAB Image
$I_{ij} = \{l_{ij}, a_{ij}, b_{ij}\}$, initial K centroids
$C_k = \{x_k, y_k, l_k, a_k, b_k\}$, grid size S,
compactness m, color quantization scale n,
max iteration N, Subsampling stride *Stride*

Output: superpixel assignment A_{ij}

1 **repeat**
  `/* The assignment procedure is further described in Algorithm 2 */`
2 Assign clusters and store the assignment into A_{ij} with $Offset$ and $Stride$.
3 Update the centroids from assignment A_{ij} in parallel.
4 $Offset \leftarrow (Offset + 1) \mod Stride$;
5 **until** iterated N times;
6 Assign clusters and store into A_{ij} with $Stride = 1$.
7 Enforce connectivity of A_{ij}.

4.3 Parallelization

Multi-core parallelization is able to be applied to the four parts. CIELAB conversion and quantization can be parallelized on per-pixel basis. Similarly, the centroid update is parallelizable by accumulating 5-dimensional pixel vectors for each cluster in each thread and gathering from threads.

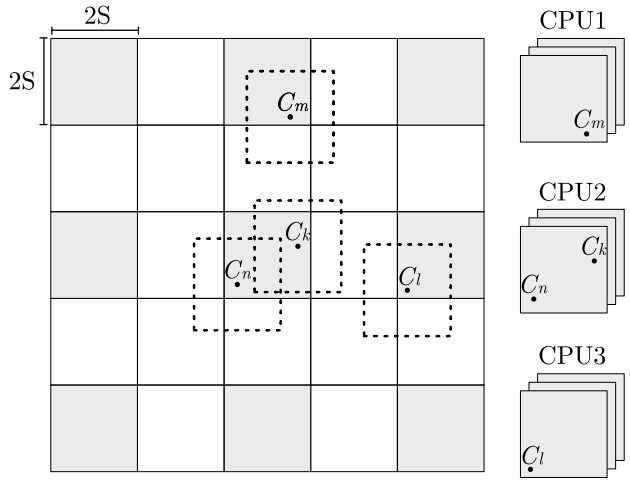


Figure 5. The tiling scheme for parallel cluster assignment. The shaded regions represent tiles of $\{T_{2i+1,2j+1}\}$. The regions of clusters C_k, C_l, C_m, drawn from different tiles of $\{T_{2i+1,2j+1}\}$, do not overlap. However, C_k and C_n in the same tile have an overlapping region. From these observations, we can treat a tile as a unit to distribute to multiple threads.

In the next sections, we describe how to perform cluster assignment and connectivity enforcement in parallel.

4.3.1 Parallel cluster assignment

We parallelize the outermost loop, where $(2S+1)\times(2S+1)$ region are scanned for each cluster. Due to overlapping regions to which multiple clusters have access, naive parallelization results in race condition bringing about a nondeterministic behavior.

In this section, we present a tiling scheme for avoiding the race condition. The key observation is that assignment of cluster k and cluster l can run in parallel if two regions of interest do not overlap, that is, $|y_k - y_l| > 2S \vee |x_k - x_l| > 2S$. We define two clusters are *independent* if their regions do not overlap.

We place the image on a 2D uniform grid where the size of a tile is $2S \times 2S$. Tile $T_{i,j}$ has row index i and column index j. A tile includes clusters whose centroid is within its area. Two tiles are independent if and only if all pairs of clusters drawn from each tile are independent.

The following inequalities hold because of the size of tile.

$$\forall_{C_k \in T_{i,j}} \forall_{C_l \in T_{i,j+2}} |x_k - x_l| > 2S \tag{12}$$

$$\forall_{C_k \in T_{i,j}} \forall_{C_l \in T_{i+2,j}} |y_k - y_l| > 2S \tag{13}$$

We can deduce from these that tiles with odd-numbered row index and those with even-numbered row index in a specific grid column are independent one another. That is, each tile of $\{T_{2i,x_0}\}$ or $\{T_{2i+1,x_0}\}$ are independent one another for all i in column x_0. The same applies to $\{T_{y_0,2j}\}$ and $\{T_{y_0,2j+1}\}$

for all j in row y_0. By putting these together, each tile of any tile set of $\{T_{2i,2j}\}$, $\{T_{2i,2j+1}\}$, $\{T_{2i+1,2j}\}$ and $\{T_{2i+1,2j+1}\}$, is independent one another for all i and j, as illustrated in Section 4.3.1.

Algorithm 2: Parallel cluster assignment

Input: K centroids $C_k = \{x_k, y_k\}$, grid size S, image width W, image height H, K centroids $C_k = \{x_k, y_k, l_k, a_k, b_k\}$, grid size S, compactness m, color quantization scale n, Subsampling offset *Offset*, Subsampling stride *Stride*

Output: distance map d_{ij}, superpixel assignment A_{ij}

Initialize d_{ij} with $0xFFFF$;
Initialize A_{ij} with $0xFFFF$;
`/* Prepare spatial distance cache P */`
Initialize P as u16 array of size $(2S+1) \times (2S+1)$;
for $u = -S$ to S **do**
    for $v = -S$ to S **do**
        $P_{uv} \leftarrow \lfloor \frac{m}{S}(|u| + |v|) \rceil * 2^n$
Initialize T as an 2D array with size $\lceil \frac{H}{2S} \rceil \times \lceil \frac{W}{2S} \rceil$;
for $C_k = \{x_k, y_k\}$ **do**
    $i \leftarrow \lfloor \frac{y}{2S} \rfloor$, $j \leftarrow \lfloor \frac{x}{2S} \rfloor$;
    $T_{i,j} \leftarrow T_{i,j} \cup \{C_k\}$;
for $W \in \{\{T_{2i,2j}\}, \{T_{2i,2j+1}\}, \{T_{2i+1,2j}\}, \{T_{2i+1,2j+1}\}\}$ **do**
    Split W into W_t for each thread t ;
    Create new threads and run the following loop in each thread t ;
    for $T_c \in W_t$ **do**
        for $C_k = \{x_k, y_k, l_k, a_k, b_k\} \in T_c$ **do**
            for $u = -S$ to S **do**
                $i \leftarrow y_k + u$;
                if $i \mod Stride \not\equiv Offset$ **then**
                    continue
                for $v = -S$ to S **do**
                    $j \leftarrow x_k + v$;
                    $d_s \leftarrow |l_{ij} - l_k| + |a_{ij} - a_k| + |b_{ij} - b_k|$;
                    $d \leftarrow P_{uv} + d_s$;
                    if $d_{ij} > d$ **then**
                        $d_{ij} \leftarrow d$;
                        $A_{ij} \leftarrow k$;
    Join threads ;

Algorithm 2 shows how to distribute clusters onto multiple threads. We first choose one of the four tile sets. Then tiles of the set are evenly distributed to multiple threads. Each thread performs cluster assignment against the clusters included in its own tiles. By this way, cluster assignment can be safely parallelized without race condition.

4.3.2 Parallel connectivity enforcement

As the main procedure gets faster by parallelization, a serial connectivity enforcement eventually becomes the bottleneck of the entire process. We use the parallel CCL procedure proposed in [10] with several alterations for connectivity enforcement.

There are a few differences in our implementation from the original one. We alter the original algorithm to treat adjacent pixels as neighbors only if they have the same cluster number. Plus, we use 4-connectivity instead of 8-connectivity for the sake of performance, in which only north, east, south and west pixel of each pixel are considered adjacent.

5 Experiment

In this section, we compare the accuracy and running time of our approach against those of the state-of-the-art methods: SLIC [3], SNIC [4], preSLIC (Preemptive SLIC) [21], CWS (Compact Watershed) [21], SEEDS [29], USEQ [12] and USEAQ [13], on the BSDS500 dataset [19]. We also carry out experiments with several values of *Stride* and see its effect on accuracy and running time. Then, we further examine how much each optimization technique presented in this paper contributes to its speed gain.

For the experiments, we used Ryzen 2600x, ARM Cortex-A72 and GTX 1070Ti for experiments on x64, ARM and GPU. The running time in benchmark include both colorspace conversion time and post-processing time. We use N of 10, *Stride* of 3 and m of 10, unless specified otherwise.

5.1 Evaluation of accuracy

The accuracy of the superpixel methods is evaluated in the three metrics: Boundary Recall (BR), Undersegmentation error(USE), and Achievable Segmentation Accuracy (ASA).

Boundary recall Boundary recall is a metric to measure boundary adherence. The higher boundary recall is, the more superpixels are adherent to boundaries. It is defined as the ratio of true positive boundary pixels to ground truth boundary pixels [24]. A ground-truth boundary pixel is considered *positive* if at least one boundary pixel on superpixels is within r pixels of it. We choose $r = 1$ for our experiments.

$$BR(B^G, B^S) = \frac{\sum_{p \in B^G} [\bigvee_{q \in B^S} max(|x_p - x_q|, |y_p - y_q|) \leq r]}{|B^G|} \tag{14}$$

where B^G is a set of pixels on ground-truth boundaries, B^S is a set of pixels on superpixel boundaries and $[\cdot]$ is the indicator function.

Undersegmentation error Undersegmentation error is a metric to measure a fraction of leakage of superpixel around ground truth boundary. There is a variation in its definition in literature. We define it as follows, as proposed in [20], because it does not suffer from error overestimation on large superpixels having a small overlap with ground truth.

$$USE(G,S) = \frac{1}{N}\sum_i \sum_{S_k \cap G_i \neq \emptyset} min(|S_k \cap G_i|, |S_k - G_i|) \quad (15)$$

where G_i is a ground-truth segmentation, S_k is a superpixel segmentation and N is the total number of pixels in image.

Achievable segmentation accuracy Achievable segmentation accuracy is the maximum accuracy achievable when classification of each superpixel is perfectly correct, as defined in [16]:

$$ASA(G,S) = \frac{\sum_k max_i |S_k \cap G_i|}{\sum_i G_i} \quad (16)$$

Figure 6a, 6c and 6e plot accuracy of each superpixel method assessed by BR, USE and ASA with respect to the number of superpixels (K). From these results, it appears accuracy of FastSLIC is comparable to that of the conventional SLIC, or even better. It implies quantization and subsampling with *Stride* of 3, described in Section 4, have little effect on accuracy.

5.2 Comparison of computational efficiency of superpixel methods

In this section, we show running time comparison of the superpixel methods. For fair comparison, the experiments to be performed are separated into serial runs and parallel runs. In the parallel runs, 6 threads and 4 threads are used on x64 and ARM respectively.

Figure 7 depicts the benchmark results of the superpixel methods on a single core and multiple cores. Only the methods supporting multi-core are shown in parallel runs. We observe our method significantly outperforms other methods running on cpus in terms of latency.

Table 1 shows that it is 9.85 times faster than the original SLIC implementation when executed in serial and even 32.8 times faster when multiple cores used, on the x64 processor. We observe it is 9.48 times faster in serial and 17.35 times faster in parallel on the ARM processor. The running time of our method is comparable to that of gSLIC which takes advantage of GPU.

Method	x64 (ms)	ARM (ms)
SLIC	394	1840
SEEDS	311	1503
USEAQ	224	928
USEQ	95	345
preSLIC	85	443
CWS	82	454
FastSLIC	40	194
FastSLIC (2 cores)	23	124
FastSLIC (3 cores)	19	**106**
FastSLIC (4 cores)	16	107
FastSLIC (6 cores)	**12**	
gSLICr	12	

Table 1. Running times of fast superpixel methods for 1280× 720 image on the x64 and the ARM processor. The last line describes the running time of gSLICr, a GPU implementation of SLIC.

	x64		ARM	
Stride	Time (ms)	Speed up	Time (ms)	Speed up
1	76.1	1.00	581.9	1.00
3	38.9	1.96	305.1	1.91
5	32.1	2.37	246.3	2.36
8	28.8	2.64	209.1	2.78
10	27.4	2.78	197.1	2.95

Table 2. Running time with respect to *Stride* for 1280×720 at K of 1500.

5.3 Analysis of effect of subsampling

Figure 6b, 6d and 6f demonstrates how each value of *Stride* affects the accuracy of outputs. We observe the results generated with *Stride* of 3 show almost the same accuracy as those with *Stride* of 1. As *Stride* gets higher than 3, accuracy starts to degrade in all three metrics.

5.4 Analysis of running time improvement

FastSLIC owes its computational efficiency the four techniques proposed in this paper: distance caching (DC), 16-bit distance quantization and integer-only-arithmetic (DQ), SIMD and row subsampling (RSS). Table 3 shows the effects of each of these components on running time improvement, tested on a single core.

The baseline is SLIC incorporating the color quantization and connectivity enforcement procedure described in Section 4. DC was effective on x64 making 18% faster than the baseline while there was little improvement on ARM. In contrast, DQ resulted in a 24% performance gain while it had little impact on x64. SIMD and RSS were the most effective ways to reduce running time. SIMD gave speed up of 104% and 53% and RSS gave additional performance gain of 96%

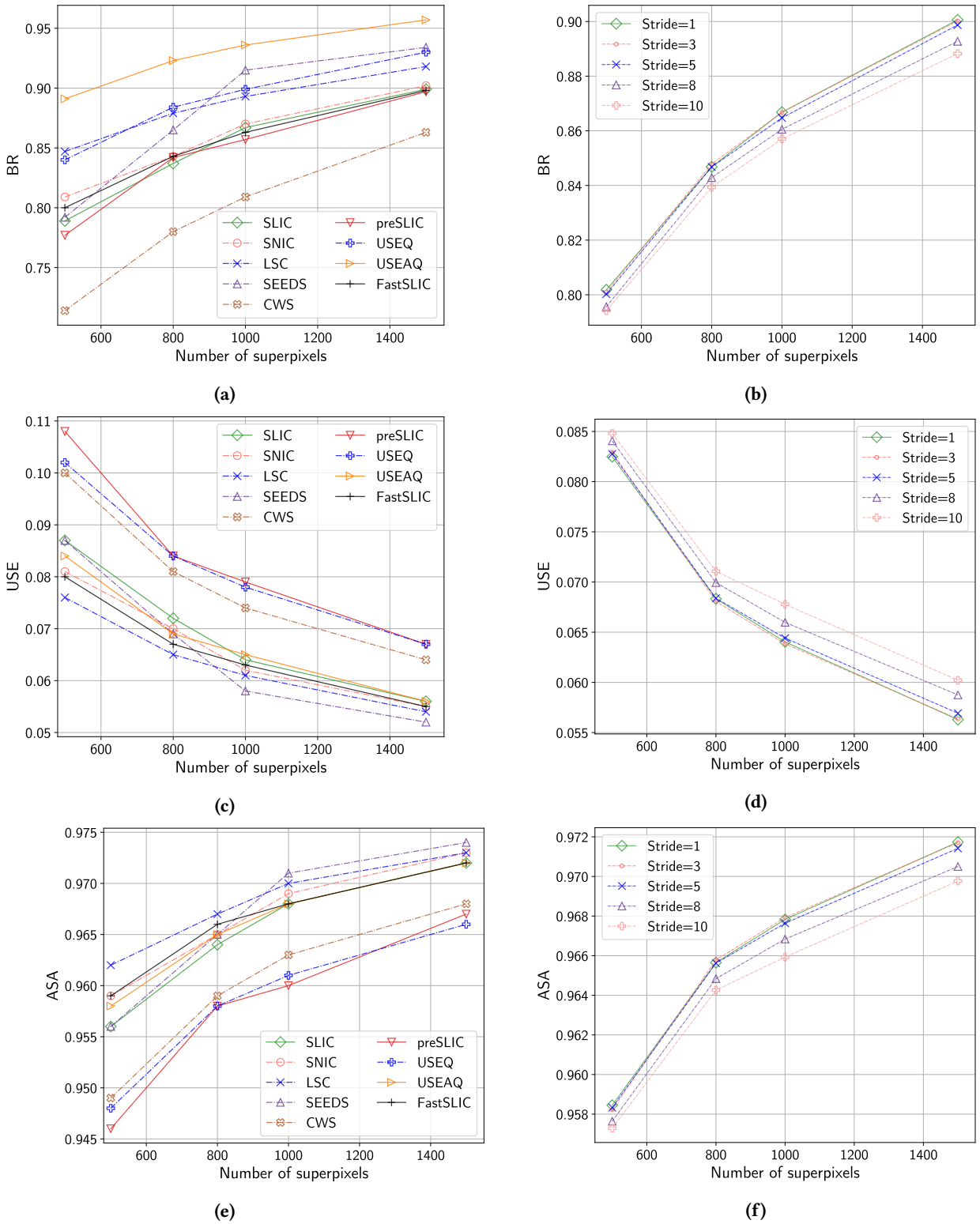
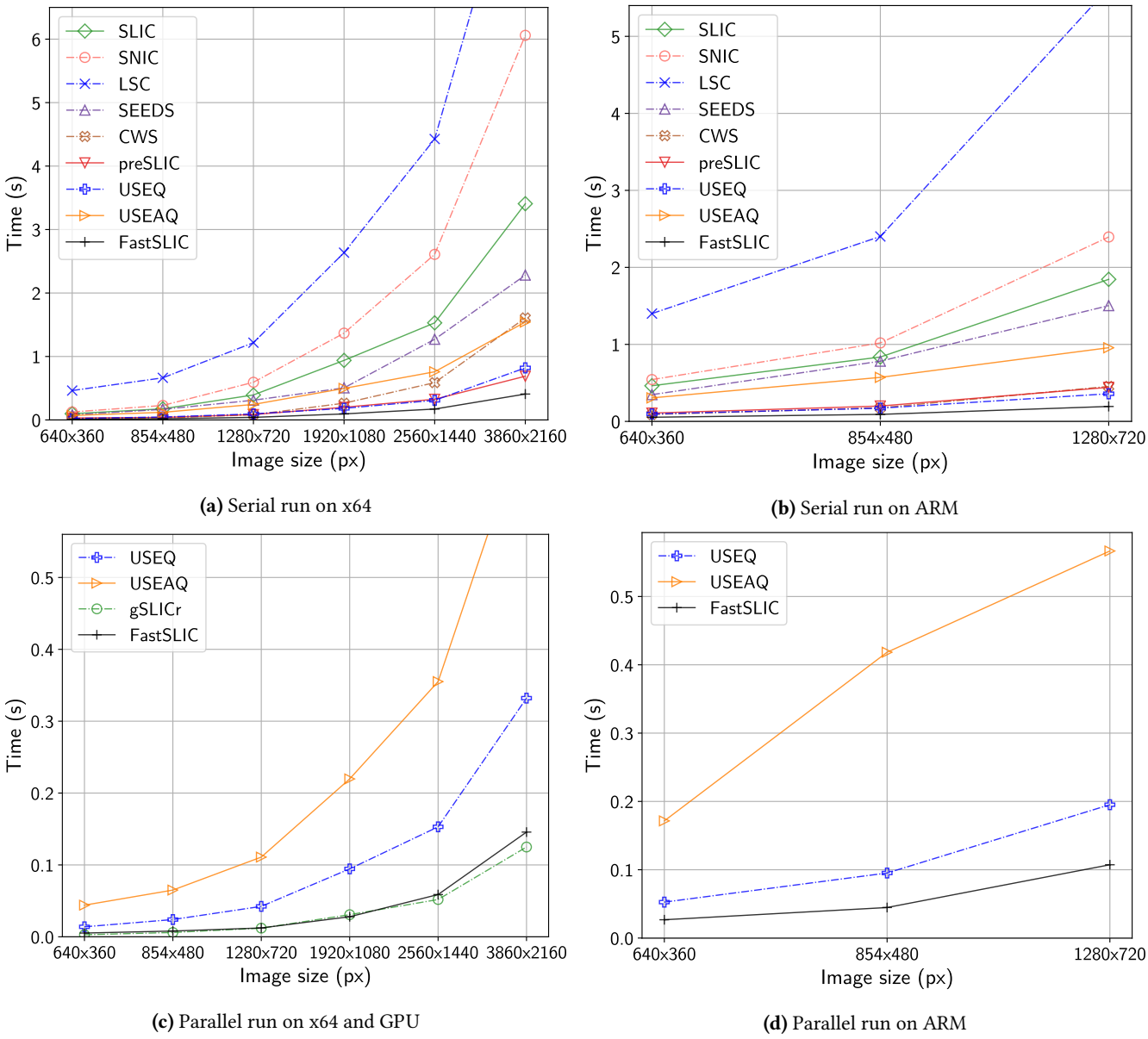


Figure 6. The first column corresponds to accuracy of superpixel algorithms evaluated in the three metrics: (a) boundary recall (higher is better). (c) under-segmentation error (lower is better). (e) achievable segmentation accuracy (higher is better). the second column demonstrates accuracy loss incurred by *Stride* of 1, 3, 5, 8 and 10.



(a) Serial run on x64

(b) Serial run on ARM

(c) Parallel run on x64 and GPU

(d) Parallel run on ARM

Figure 7. Benchmark on latency with respect to image size at K of 1500.

	x64		ARM	
Method	Time (ms)	Speed up	Time (ms)	Speed up
baseline	185.2	1.00	1146.2	1.00
DC	156.3	1.18	1112.5	1.03
DC + DQ	155.9	1.19	894.6	1.24
DC + DQ + SIMD	76.1	2.43	581.9	1.91
DC + DQ + SIMD + RSS	38.9	4.76	305.1	3.65

Table 3. Running time of FastSLIC applying different optimizations for 1280×720 image on the x64 and the ARM processor at K of 1500. RSS is performed with *Stride* of 3.

	x64		ARM	
# Threads	Time (ms)	Speed up	Time (ms)	Speed up
1	40.4	1.00	194.0	1.00
2	22.5	1.80	124.2	1.56
3	19.0	2.13	106.2	1.83
4	15.9	2.54	107.3	1.81
5	12.6	3.21		
6	12.3	3.28		

Table 4. Running time with respect to the number of threads for 1280×720 at K of 1500.

and 91% on x64 and ARM, respectively. Overall, FastSLIC ran 4.76 times faster on x64 and 3.65 times faster on ARM than its baseline.

As shown in Table 4, we gained speed up of 3.28 and 1.83 when 6 cores and 3 cores are used on x64 and ARM respectively. The speed up did not grow further when more than 3 threads are used on ARM. We conjecture it is due to saturated memory bandwidth.

6 Conclusion

Superpixel extraction is widely used in pre-processing phases of computer vision tasks such as object detection and video tracking. SLIC is one of the most commonly used method due to its simplicity and high quality superpixels. Nevertheless, it is not capable of processing at real-time large-sized images without resorting to a powerful-but-expensive GPU. We present a fast SLIC-variant algorithm that is an order of magnitude faster than SLIC and other state-of-art algorithms. We show color quantization, integer distance metric, SIMD and row subsampling are effective techniques for reducing computational cost. Also, clusters are distributed on per-tile basis to multiple threads for multi-core parallelization. Benchmark shows that our method is 10 times faster than SLIC in serial and is 33 times faster when parallelized on 6 cores. FastSLIC does not degrade accuracy at the cost of speed, achieving the same or better quality comparing to SLIC.

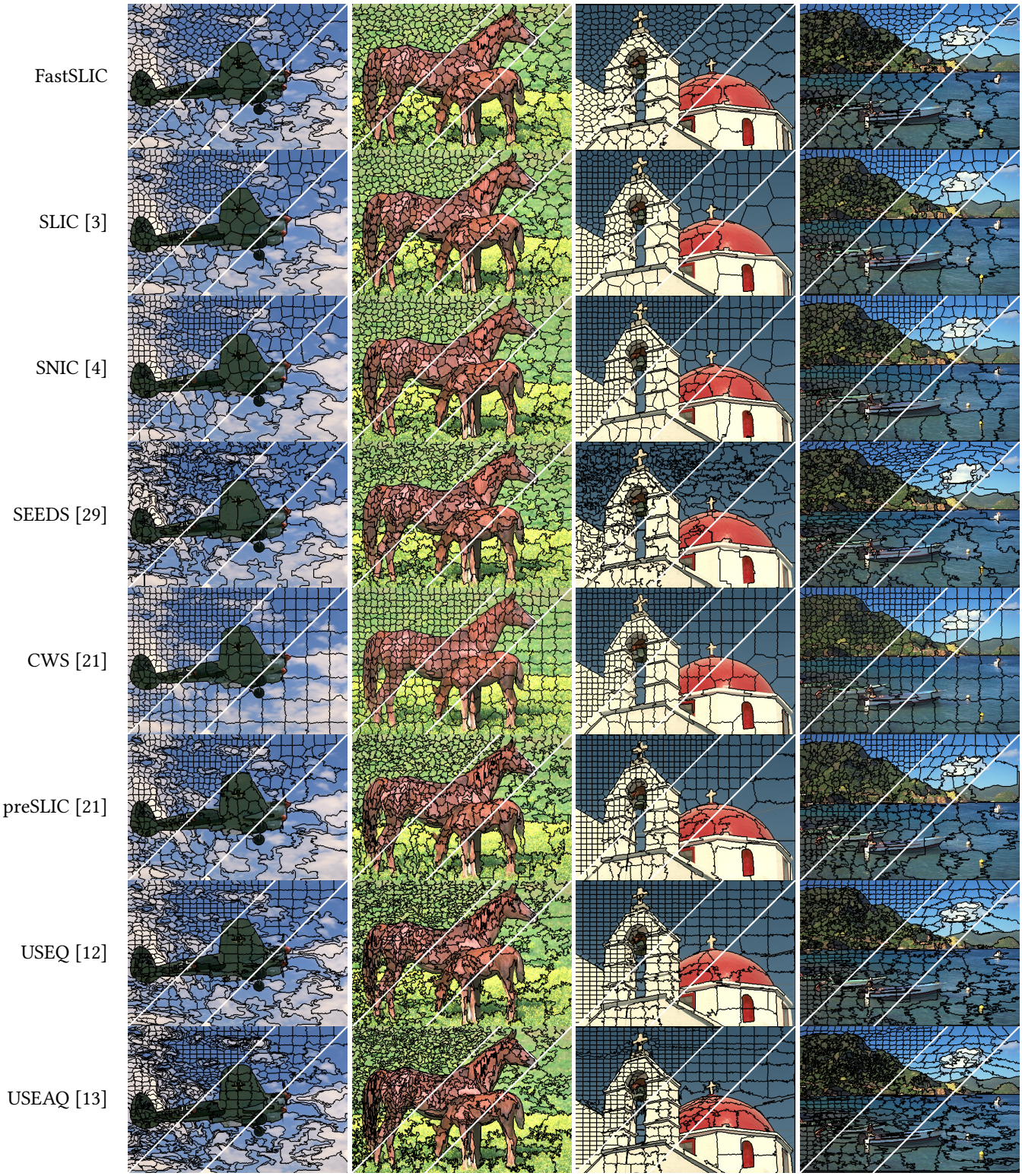


Figure 8. Visual comparison of superpixel algorithms. Image is segmented into superpixels of size 1024 (left), 256 (center) and 64 (right).

References

[1] SIMD ISAs Neon. https://developer.arm.com/architectures/instruction-sets/simd-isas/neon Accessed: 2019-05-20.

[2] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. 2010. *Slic superpixels*. Technical Report.

[3] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. 2012. SLIC superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence* 34, 11 (2012), 2274–2282.

[4] Radhakrishna Achanta and Sabine Susstrunk. 2017. Superpixels and polygons using simple non-iterative clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4651–4660.

[5] Pedro F Felzenszwalb and Daniel P Huttenlocher. 2004. Efficient graph-based image segmentation. *International journal of computer vision* 59, 2 (2004), 167–181.

[6] Franz Franchetti, Stefan Kral, Juergen Lorenz, and Christoph W Ueberhuber. 2005. Efficient utilization of SIMD extensions. *Proc. IEEE* 93, 2 (2005), 409–425.

[7] Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. 2009. Class segmentation and object localization with superpixel neighborhoods. In *2009 IEEE 12th international conference on computer vision*. IEEE, 670–677.

[8] Fabio Galasso, Roberto Cipolla, and Bernt Schiele. 2012. Video segmentation with superpixels. In *Asian Conference on Computer Vision*. Springer, 760–774.

[9] Daniela Giordano, Francesca Murabito, Simone Palazzo, and Concetto Spampinato. 2015. Superpixel-based video object segmentation using perceptual organization and location prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4814–4822.

[10] Siddharth Gupta, Diana Palsetia, Md Mostofa Ali Patwary, Ankit Agrawal, and Alok Choudhary. 2014. A new parallel algorithm for two-pass connected component labeling. In *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 1355–1362.

[11] Injoon Hong, Iuri Frosio, Jason Clemons, Brucek Khailany, Rangharajan Venkatesan, and Stephen W Keckler. 2016. A real-time energy-efficient superpixel hardware accelerator for mobile computer vision applications. In *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[12] Chun-Rong Huang, Wei-An Wang, Szu-Yu Lin, and Yen-Yu Lin. 2016. USEQ: Ultra-fast superpixel extraction via quantization. In *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 1965–1970.

[13] Chun-Rong Huang, Wei-Cheng Wang, Wei-An Wang, Szu-Yu Lin, and Yen-Yu Lin. 2018. USEAQ: Ultra-Fast Superpixel Extraction via Adaptive Sampling From Quantized Regions. *IEEE Transactions on Image Processing* 27, 10 (2018), 4916–4931.

[14] Varun Jampani, Deqing Sun, Ming-Yu Liu, Ming-Hsuan Yang, and Jan Kautz. 2018. Superpixel sampling networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 352–368.

[15] Zhengqin Li and Jiansheng Chen. 2015. Superpixel segmentation using linear spectral clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 1356–1363.

[16] Ming-Yu Liu, Oncel Tuzel, Srikumar Ramalingam, and Rama Chellappa. 2011. Entropy rate superpixel segmentation. In *CVPR 2011*. IEEE, 2097–2104.

[17] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[18] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[19] David Martin, Charless Fowlkes, Doron Tal, Jitendra Malik, et al. 2001. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Iccv Vancouver:.

[20] Peer Neubert and Peter Protzel. 2012. Superpixel benchmark and comparison. In *Proc. Forum Bildverarbeitung*, Vol. 6.

[21] Peer Neubert and Peter Protzel. 2014. Compact watershed and preemptive slic: On improving trade-offs of superpixel segmentation algorithms. In *2014 22nd International Conference on Pattern Recognition*. IEEE, 996–1001.

[22] Venu Gopal Reddy. 2008. Neon technology introduction. *ARM Corporation* 4 (2008), 1.

[23] Carl Yuheng Ren, Victor Adrian Prisacariu, and Ian D Reid. 2015. gSLICr: SLIC superpixels at over 250Hz. *arXiv preprint arXiv:1509.04232* (2015).

[24] Xiaofeng Ren and Jitendra Malik. 2003. Learning a classification model for segmentation. In *null*. IEEE, 10.

[25] David Sculley. 2010. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*. ACM, 1177–1178.

[26] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *Departmental Papers (CIS)* (2000), 107.

[27] Wei-Chih Tu, Ming-Yu Liu, Varun Jampani, Deqing Sun, Shao-Yi Chien, Ming-Hsuan Yang, and Jan Kautz. 2018. Learning superpixels with segmentation-aware affinity loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 568–576.

[28] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. 2013. Selective search for object recognition. *International journal of computer vision* 104, 2 (2013), 154–171.

[29] Michael Van den Bergh, Xavier Boix, Gemma Roig, Benjamin de Capitani, and Luc Van Gool. 2012. Seeds: Superpixels extracted via energy-driven sampling. In *European conference on computer vision*. Springer, 13–26.

[30] Luc Vincent and Pierre Soille. 1991. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 6 (1991), 583–598.

[31] Shu Wang, Huchuan Lu, Fan Yang, and Ming-Hsuan Yang. 2011. Superpixel tracking. In *2011 International Conference on Computer Vision*. IEEE, 1323–1330.

[32] Kesheng Wu, Ekow Otoo, and Kenji Suzuki. 2005. *Two strategies to speed up connected component labelingalgorithms*. Technical Report. Ernest Orlando Lawrence Berkeley NationalLaboratory, Berkeley, CA (US).

[33] Junjie Yan, Yinan Yu, Xiangyu Zhu, Zhen Lei, and Stan Z Li. 2015. Object detection by labeling superpixels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5107–5116.