Neurocomputing 277 (2018) 228–236

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Parameter-free image segmentation with SLIC

Fabian Boemer[a,*], Edward Ratner[a], Amaury Lendasse[b,c,d]

[a] Lyrical Labs, 14.5 Clinton St., Iowa City, IA, USA
[b] Department of Mechanical and Industrial Engineering, The University of Iowa, Iowa City, USA
[c] The Iowa Informatics Initiative, The University of Iowa, Iowa City, USA
[d] Risklab, Arcada University of Applied Sciences, Helsinki, Finland

ARTICLE INFO

Article history:
Received 1 October 2016
Revised 27 April 2017
Accepted 9 May 2017
Available online 24 August 2017

Keywords:
SLIC
ELM
Image segmentation
Superpixel
Streaming

ABSTRACT

In this paper, we develop a parameter-free image segmentation framework using Simple Linear Iterative Clustering (SLIC) and Extreme Learning Machines (ELM). SLIC requires a single parameter, the number of centroids *k*. Our framework, called PF-SLIC (Parameter-Free SLIC) uses an ELM to predict the optimal *k*, generating a parameter-free framework. PF-SLIC and its streaming variant SPF-SLIC (Streaming PF-SLIC) achieve performance comparable to other models on ultra-high-definition (4K) images and streams, with runtimes orders of magnitude lower.

1. Introduction

The goal of image segmentation is to partition an image with N pixels into disjoint sets of pixels called clusters. Image segmentation has been widely applied in machine vision [1], medical applications [2], and video compression [3]. Existing image segmentation algorithms are based on artificial neural networks, partial differential equations, edge-detection, fuzzy theory, region-detection, and thresholds [4]. While many of these approaches yield reasonable segmentations, they are often slow [5] and therefore computationally intractable for large images.

One way to counteract computational infeasibility is through a pre-processing step called superpixel segmentation. Superpixel segmentation is a sub-problem of image segmentation with the goal of representing an image accurately using a smaller number of pixels, called superpixels. Desired properties of superpixels depend on specific application, but may include adherence to image boundaries, computational efficiency, and improvement of subsequent image processing [6,7]. Efficient superpixel segmentation is important in processing large images, especially in the realm of ultra-high-definition ($N = 3840 \times 2160$) resolution.

A downside to many superpixel segmentation algorithms is the need for parameter tuning, which can greatly reduce the practical efficiency of the algorithm. Turbopixels [8] and SLIC [6], for instance, have a single parameter to tune, while an energy-minimization algorithm [9] and FH [10] have three parameters to tune. One of the fastest superpixel algorithms, SLIC [6], is a clustering approach with $O(N)$ runtime, and has state-of-the-art edge adherence. However, SLIC requires two input parameters, a compactness factor m and the number of clusters k. In practice m can be fixed to a constant. A variation, Adaptive-SLIC (ASLIC) [6], chooses m adaptively and therefore requires only a single parameter k.

We propose a machine learning framework to solve the problem of parameter tuning by learning the optimal SLIC parameter k. Since superpixel segmentation should be efficient, we also desire efficiency in the machine learning approach. Extreme Learning Machines (ELM) [11] are single-hidden-layer, feed-forward neural networks with extremely fast training time and performance comparable to other machine learning models, hence a promising candidate for computationally efficient learning.

We formulate the problem of parameter tuning as a multi-class prediction problem, where the classes are a set of values for the parameter to be tuned. Our approach, Parameter-Free SLIC (PFSLIC), and its streaming variant, Streaming Parameter-Free SLIC (SPF-SLIC), predict optimal SLIC parameters using an ELM. PF-SLIC and SPF-SLICE achieve prediction performance comparable to other methods, but with only a fraction of the runtime on both 4K images and 4K video streams.

* Corresponding author.
E-mail addresses: fabian.boemer@lyricallabs.com (F. Boemer), ed.ratner@lyricallabs.com (E. Ratner), amaury-lendasse@uiowa.edu (A. Lendasse).

http://dx.doi.org/10.1016/j.neucom.2017.05.096
0925-2312/

2. Previous work

We discuss recent developments in both image segmentation and machine learning.

2.1. Image segmentation

Currently-existing image segmentation algorithms fall into several broad categories: graph-based methods, gradient-ascent methods, and clustering methods. We provide a brief summary of existing superpixel methods:

- The watershed algorithm [12] is a gradient-ascent method which runs in $O(N\log N)$ time. The algorithm generates segments, bounded by *watershed* lines, bottom-up by expanding regions from local minima. Achanta et al. find the watershed algorithm has poor boundary adherence and produces superpixels irregular in size and shape.
- The Turbopixel approach [8] is a geometric-flow based algorithm, which runs in nearly $O(N)$ time. Achanta et al. find Turbopixels exhibit poor boundary adherence and are slow in practice.
- SLIC (Simple Linear Iterative Clustering) [6] is a superpixel segmentation algorithm which runs in $O(N)$ time. SLIC adapts k-means clustering by searching for cluster centers only over a constant-sized region. SLIC is very fast and has good boundary adherence. Achanta et al. find SLIC is the fastest superpixel algorithm in practice.
- Felzenszwalb and Huttenlocher [10] proposed a graph-based image segmentation algorithm (FH) that runs in $O(N\log N)$ time. The algorithm builds superpixels bottom-up with a method similar to Kruskal's algorithm. FH requires tuning a single parameter which sets the preference for larger or smaller superpixels, though the threshold function τ can be used to tune desired superpixel shapes. Achanta et al. find this algorithm has very good boundary adherence and runs nearly as fast as SLIC.

Several other superpixel segmentation algorithms are superlinearithmic and likely too slow for large N. For ultra-high-definition images, such as 4K (3840×2160) images, N approaches 10^7. As such, possible performance gains in considering the Normalized Cuts [13] $O(N^{3/2})$, Quickshift [14] $O(N^{3/2}\log N)$, Superpixel Lattices [15], ($O(N^{3/2}\log N)$, and an energy-minimization algorithm [9] ($O(N^2)$) are outweighed by intractable runtimes.

We conclude SLIC and FH are the only feasible algorithms for our setting. FH, however, does not provide direct control over either superpixel size or compactness. We therefore consider SLIC as the most viable superpixel candidate to develop a fast parameter-tuning framework.

2.1.1. SLIC

SLIC (Simple Linear Iterative Clustering) [6] is a superpixel segmentation algorithm with very good boundary adherence. SLIC adapts k-means clustering by searching for cluster centers only over a $2S\times2S$ region, where $S\approx\sqrt{N/k}$. In practice, SLIC converges within 10 iterations of recomputing centroids, yielding a $O(N)$ runtime. SLIC also requires a post-processing step to ensure superpixels are not disjoint. SLIC takes two parameters: the number of clusters k, and a compactness parameter m. We refer to the average superpixel size as $k^s=N/k$. The distance D' between pixels in *lab*-color and *xy*-spatial space is computed as

$$d_c=\sqrt{(l_i-l_j)^2+(a_i-a_j)^2+(b_j-b_i)^2} \tag{1}$$

$$d_s=\sqrt{(x_j-x_i)^2+(y_i-y_j)^2} \tag{2}$$

$$D'=\sqrt{\left(\frac{d_c}{N_c}\right)^2+\left(\frac{d_s}{N_s}\right)^2} \tag{3}$$

where N_c and N_s are the maximum color distance and spatial distance, respectively, within the superpixel. In practice, N_c is difficult to estimate, and therefore fixed to a constant m. D' is thus scaled to the metric used in practice,

$$D=\sqrt{d_c^2+\left(\frac{d_s}{S}\right)^2 m^2} \tag{4}$$

Large values of m place more weight on spatial proximity, generating spatially compact superpixels. Small values of m generate superpixels with better color boundary adherence, but lower spatial compactness. In *lab*-color, *xy*-spatial space, $m\in[1, 40]$ yields good performance. d_c is easily adapted to other color spaces, such as *rgb*, though the range of suitable m may need to be adjusted.

Fig. 1 shows segmentations for a 4K (3840×2160) image with various parameters k^s. The superpixels have good boundary adherence for superpixels as large as $k^s=1024$, as evident in the adherence to the shape of the taxi.

Adaptive-SLIC [6], or ASLIC, is a modification of SLIC which removes m as an input parameter. At each iteration, the maximum observed spatial and color distances (m_s, m_c) observed in the previous iteration within each superpixel are used to estimate N_s and N_c in (3). In practice, the authors find ASLIC has reduced boundary adherence, but generates superpixels with consistent compactness [6].

2.2. ELM

An Extreme Learning Machine (ELM) [11] is a single hidden-layer feed-forward neural network. ELMs are extremely fast to train and perform favorably to established machine learning models. ELMs learn a model of the form

$$\mathbf{H}\beta=\mathbf{T} \tag{5}$$

$$H_{ij}^{n\times\tilde{N}}=\sigma(\mathbf{w_j}\cdot\mathbf{x_i}+b_j) \tag{6}$$

$$\beta_i^{\tilde{N}\times m}=\beta_i \tag{7}$$

$$\mathbf{T}^{N\times m}=\mathbf{t_i}^T \tag{8}$$

for σ an activation function, weights $\mathbf{w}$, $\boldsymbol{\beta}$ and biases b_i, $\tilde{N}$ hidden neurons, and m classes. After learning $\boldsymbol{\beta}$, predictions for a set of testing points X are made via $\mathbf{H}\beta$ where $\mathbf{H}$ is computed for the testing points. For single-class prediction, $\mathbf{T}$ contains a single column, while for categorical multi-class prediction, $\mathbf{T}$ contains a column for each class, with each row containing all zeros, and a one for the correct class [16]. Algorithm 1 details the basic ELM training algorithm.

Algorithm 1 ELM Algorithm.

Require: set S of training images, $\tilde{N}$ number of hidden neurons.
1: Randomly initialize $\mathbf{w_i}$ and b_i, $i=1,\ldots,\tilde{N}$
2: Compute $\mathbf{H}$
3: Compute $\beta=\mathbf{H}^\dagger\mathbf{T}$ where $\mathbf{H}^\dagger$ is the Moore–Penrose pseudoinverse of $\mathbf{H}$
4: **return** trained ELM Model

Several regularization methods such as l_1 and l_2 and elastic net prevent overfitting [17,18]. ELM also has extensions to imbalanced classes [19]. PRESS optimization [20] efficiently tunes the number

Fig. 1. Example segmentation including the original 4K image (*top-left*), a cut-out (*top-right*), the best-parameter segmentation, $k^s = 256$, (*middle-right*), as well as segmentations for $k^s = 256$ (*bottom-left*), $k^s = 576$ (*bottom-middle*), $k^s = 1024$ (*bottom-right*) for the cut-out image.

of hidden neurons $\tilde{N}$. In practice, ELM is several orders of magnitude faster than methods such as SVMs, and still achieves comparable performance [19], making ELM a promising candidate for our setting.

ELM also has an adaptation to the streaming setting, known as OS-ELM (Online-Sequential ELM) [21]. In this setting, consecutive images from a video stream are sequentially fed in batches to the ELM. In our setting, each batch contains a single image. The ELM must output a prediction for the received batch before receiving the next batch. Algorithm 2 details the OS-ELM algorithm. The initial training phase is used to generate initial estimates $\beta^{(0)}$. In the second, streaming phase, given the $(k+1)$th batch of N_{k+1} new observations, OS-ELM efficiently updates the computed weights $\boldsymbol{\beta}^{(k)}$ to $\beta^{(k+1)}$. It is worth noting OS-ELM can accept batches of varying sizes, that is N_k can differ from N_{k+1}.

Algorithm 2 OS-ELM Algorithm.

Step 1 (*Initial training phase*)

Require: Initial batch training set of N_0 training points $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N_0}$, $\tilde{N}$ number of hidden neurons.

1: Assign input weight and biases randomly in $[-1, 1]$

2: Compute hidden layer output matrix $\mathbf{H}_0$ via

$$\mathbf{H}_0 = \begin{bmatrix} \sigma(\mathbf{w_1} \cdot \mathbf{x_1} + b_1) & \dots & \sigma(\mathbf{w_{\tilde{N}}} \cdot \mathbf{x_1} + b_{\tilde{N}}) \\ \vdots & \dots & \vdots \\ \sigma(\mathbf{w_1} \cdot \mathbf{x_{N_0}} + b_1) & \dots & \sigma(\mathbf{w_{\tilde{N}}} \cdot \mathbf{x_{N_0}} + b_{\tilde{N}}) \end{bmatrix}$$

3: Compute $\beta^{(0)} = \mathbf{P}_0 \mathbf{H}_0^T \mathbf{T}_0$, where $\mathbf{P}_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1}$ and $T_0 = \left[\mathbf{t}_1, \dots, \mathbf{t}_{N_0}\right]^T$

Step 2 (*Streaming phase*)

Require: $(k+1)$th batch of N_{k+1} training points $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1+\sum_{j=0}^{k} N_j}^{\sum_{j=0}^{k+1} N_j}$, $\tilde{N}$ number of hidden neurons.

1: Compute $\mathbf{H}_{k+1}$ corresponding to new batch

2: Set $\mathbf{T}_{k+1} = [\mathbf{T_k}, t_{1+\sum_{j=0}^{k} N_j}, \dots, t_{\sum_{j=0}^{k+1} N_j}]$

3: Calculate $\beta^{(k+1)}$ via

$$\mathbf{P}_{k+1} = \mathbf{P_k} - \mathbf{P}_k \mathbf{H}_{k+1}^T \left(\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T\right)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k$$
$$\beta^{(k+1)} = \beta^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T \left(\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \beta^{(k)}\right)$$

OS-ELM has a further adaptation, OS-WELM [22], to the weighted setting, in which previously-trained datapoints are weighted by a decay factor $0 < \omega < 1$. Algorithm 3 details the OS-WELM algorithm for the special case in which batches N_k are single datapoints. OS-WELM has been applied to time travel forecasting, in which adaptation to recently-streamed information is especially important [22].

Algorithm 3 OS-WELM Algorithm.

Step 1 (*Initial training phase*)

Require: $(k+1)$th training point $(\mathbf{x}_i, \mathbf{t}_i)$, $\tilde{N}$ number of hidden neurons, decay factor $0 < \omega < 1$

1: Assign input weight and biases randomly in $[-1, 1]$

2: Compute hidden layer output matrix $\mathbf{H}_0$ via

$$\mathbf{H}_0 = \begin{bmatrix} \sigma(\mathbf{w_1} \cdot \mathbf{x_1} + b_1) & \dots & \sigma(\mathbf{w_{\tilde{N}}} \cdot \mathbf{x_1} + b_{\tilde{N}}) \\ \vdots & \dots & \vdots \\ \sigma(\mathbf{w_1} \cdot \mathbf{x_{N_0}} + b_1) & \dots & \sigma(\mathbf{w_{\tilde{N}}} \cdot \mathbf{x_{N_0}} + b_{\tilde{N}}) \end{bmatrix}$$

3: Compute $\beta^{(0)} = \mathbf{P}_0 \mathbf{H}_0^T \mathbf{T}_0$, where $\mathbf{P}_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1}$ and $T_0 = \left[\mathbf{t}_1, \dots, \mathbf{t}_{N_0}\right]^T$

Step 2 (*Streaming phase*)

Require: $(k+1)$th training point $(\mathbf{x}_i, \mathbf{t}_i)$, $\tilde{N}$ number of hidden neurons, decay factor $0 < \omega < 1$

1: Compute $\mathbf{h}_{k+1}^{1 \times \tilde{N}} = \left[\sigma(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1), \dots, \sigma(\mathbf{w_{\tilde{N}}} \cdot \mathbf{x}_{\tilde{N}} + b_{\tilde{N}})\right]$ corresponding to new batch

2: Calculate $\beta^{(k+1)}$ via

$$\mathbf{P}_{k+1} = \frac{\mathbf{P_k}}{\omega} - \frac{\mathbf{Q_k} \mathbf{Q_k}^T}{\omega(\omega + \mathbf{h}_{k+1} \mathbf{Q_k})}$$
$$\beta_{k+1} = \beta_k + \mathbf{P}_{k+1} \mathbf{h}_{k+1}^T (t_{k+1} - h_{k+1} \beta_k)$$

3. Method

We use SLIC and various modifications of the basic ELM to develop an automated image segmentation framework which scales to 4K images and streams, with segmentation performance comparable to standard machine learning models (Decision Tree, Random Forest, Support Vector Machine).

SLIC is very fast in practice and maintains excellent boundary adherence. The compactness parameter m can vary between 1 and

40, in *lab*-color space, though we find this range is also suitable for *rgb*-color space. The number of clusters k can be set to any value $1 \leq k \leq N$, though the extreme values of this range produce meaningless superpixels. Its one-parameter variant ASLIC [6] requires tuning only the single parameter k. SLIC, therefore, is a promising candidate for a baseline image segmentation algorithm whose parameters must be tuned.

We take a similar approach to [23] and fix $m = 20$. This represents an empirically-determined tradeoff between spatial and color proximity. Future work would improve upon our method to select m adaptively. For better intuition, we instead tune the average superpixel size $k^s = N/k$. SLIC tends to produce identical or very similar segmentations for similar superpixel sizes. We therefore select the optimal k^s_{opt} from a set $K_{a,b} = \{(2n)^2 : a \leq n \leq b, n \in \mathbb{N}\}$. Setting $a = 1, b = N$ is sufficient to cover the entire range of possible k^s. However, in practice, only white-noise images have optimal $k^s_{opt} \approx 1$, and only monochrome images have $k^s_{opt} \approx N$.

To find the optimal parameter $k^s_{opt} \in K_{a,b}$, we formulate a machine learning setting. This specific setting calls for a supervised multiclass prediction problem, with discrete ordered labels $l_1 \prec l_2 \prec \cdots \prec l_{|K_{a,b}|}$. This differs from a generic multiclass prediction problem because mispredicting class l_m for a datapoint with true class l_n is more acceptable for small $|l_m - l_n|$ than large $|l_m - l_n|$. We choose the loss function for a single prediction $f(l_m, l_n) = (\sqrt{l_m} - \sqrt{l_n})^2$, and sum $f(\cdot, \cdot)$ over all images to obtain the global loss function. This loss function penalizes predictions based on the distance to true class. In practice, it is important to define a and b such that $K_{a,b}$ is large enough to contain the range of training images but small enough such that each class $k \in K_{a,b}$ contains several images.

We choose ELM as the machine learning model due to its very fast training speed and good generalization in practice. This ensures the produced superpixel segmentation remains fast, as desired.

To generate a training set, a collection of sample images is segmented for each $k^s \in K_{a,b}$. The optimal parameter k^s_{opt} produces a superpixel segmentation which maintains good boundary adherence and detail, with as few superpixels as possible. To estimate k^s_{opt}, we compute

- Levine and Nazif's interclass contrast [24,25]: C_{Inter} is the sum of contrasts of each superpixel S_i, balanced by the number of pixels in the superpixel $|S_i|$. C_{Inter} is computed as

$$C_{Inter} = \frac{\sum_i |S_i| c_i}{\sum_i |S_i|} \tag{9}$$

  where $c_i = \sum_i \frac{l_{ij}|m_i - m_j|}{m_i + m_j}$, m_i is the mean gray-level of superpixel S_i, and l_{ij} is the length of the boundary between S_i and S_j. Large interclass contrast suggests superpixels encode color boundaries in the image well, and is therefore indicative of a good segmentation.
- Levine and Nazif's Intraclass uniformity [24,25]: C_{Intra}: the sum of standard deviations of each superpixel. C_{Intra} is computed as

$$C_{Intra} = \sum_{i=1}^{k^s} \frac{1}{|S_i|} \sum_{p \in S_i} |p - m_i| \tag{10}$$

  Large intraclass uniformity suggests superpixels encode their constituent pixels well, and is therefore indicative of a good segmentation.
- The ratio of interclass contrast to intraclass uniformity: $r = C_{Inter}/C_{Intra}$.

Interclass contrast and intraclass uniformity can be adapted to any color space as well, though we use the formulations in (9) and (10). Interclass contrast tends to increase with superpixel size, as adjacent superpixels have increasing contrast. Intraclass uniformity tends to decrease with superpixel size, as larger superpixels tend to include pixels of differing colors. These trends suggest k^s_{opt} occurs at some ratio $r_{opt} = C_{Inter}/C_{Intra}$. We use visual inspection on a number of test images to set $r_{opt} = 438$. Specifically, $r_{opt} = \min_{r \in \mathbb{R}} \sum_{i=1}^{N} (r - k_{opt,i})^2$ for $k_{opt,\,i} \in K_{5,\,18}$ visually determined, and N a training set large enough to include a variety of scenes and amount of detail. Then, a larger training set is generated using r_{opt} to choose the optimal k^s_{opt} for each training image. Visual inspection suggests this approach generates a reasonable parameter k^s_{opt}. We note a more accurate indication of the quality of a segmentation is the result of any subsequent image processing, in a particular application. In absence of a specific application, and in an effort to provide a general image segmentation framework, we use r_{opt} as a proxy for image segmentation quality.

For a given image, we construct training features from the image itself, and from a rapid segmentation on a downscaled version of the image. For each full-resolution image, we construct a co-occurrence matrix for $d = 1, \theta \in \{0, 45, 90, 135\}$, and compute the contrast, correlation, energy, entropy, maximum-probability, and inverse difference moment [26]. These features capture basic texture information about the image.

To generate additional features, the full-resolution image was downscaled by a factor of 8 in each dimension to 480×270. The downscaled image was then segmented very quickly based on the adaptive graph-partitioning method in [27]. In particular, the average segment size, interclass contrast and intraclass uniformity for this segmentation are used to predict k^s_{opt} for the full-resolution image.

Finally, to encode the error in the downscaled segmentation, we first color each segment in the downscaled segmentation with the average image color within that segment. The we construct the difference image between the downscaled original image, and this average-colored image. We also compute the contrast, correlation, energy, entropy, maximum-probability, and inverse difference moment from this difference image.

In total, we compute 54 features for each training image. Computing these features for each image requires roughly 3 s, so these features are computationally feasible.[1] We then use an ELM to learn k^s_{opt}. The inherent randomness of the ELM inner weights necessitates caution to avoid an unstable algorithm. In practice, instability can be avoided by training several ELM models, and taking either the best-performing model, or using each trained model to vote for a classification, as in Voting-ELM [28]. Algorithm 4 summarizes the framework, which we call Parameter-Free SLIC (PF-SLIC).

Algorithm 4 PF-SLIC Framework.

Require: set S of training images, set K of superpixel sizes.
1: **for** each image $I \in S$ **do**
2: compute features, such as texture features
3: **for** each $k^s \in K$ **do**
4: use SLIC to segment image I with superpixel sizes k^s
5: compute C_{Inter}, C_{Intra}, and the ratio $r_{I,k} = \frac{C_{Inter}}{C_{Intra}}$
6: **end for**
7: **end for**
8: use sample image set to find optimal ratio r_{opt}
9: use r_{opt} to classify images
10: train ELM model on training images, using several trials or Voting-ELM to mitigate instability due to randomness
11: **return** trained ELM Model

[1] All reported runtimes in this paper are from single-threaded computation on a a Intel Core i5-4258U processor at 2.40 GHz.

PF-SLIC has a natural extension to the streaming setting, which we call Streaming-PF-SLIC (SPF-SLIC). In this setting, we are given a stream $\mathcal{S}$ of images, corresponding to batches with a single datapoint. The goal is to determine the best parameter k^s for each image $I \in \mathcal{S}$. For each image, we allow SPF-SLIC to query a single SLIC image segmentation. SPF-SLIC uses OS-ELM to efficiently process streamed images. Algorithm 5 outlines SPF-SLIC. In SPF-SLIC,

Algorithm 5 SPF-SLIC Framework.

Require: stream $\mathcal{S} = I_1, I_2, \ldots$ of images, database $\mathcal{D}$ of training points and their segmentations, number of hidden neurons $\tilde{N}$
1: train ELM on database $\mathcal{D}$
2: train prediction-refinement model M on $\mathcal{D}$
3: **for** $i = 1, 2, \ldots$ **do**
4: compute features X_i for image I_i
5: use ELM to predict k^s for image I_i.
6: compute segmentation of I_i with superpixel size k^s.
7: use segmentation and M to improve prediction to k'^s
8: stream (X_i, k'^s) to the ELM
9: **end for**

the stream $\mathcal{S}$ is assumed to contain images only of the same scene, that is each image is highly correlated with the previous image. If not, Algorithm 5 can be performed on each scene present in the stream.

SPF-SLIC first trains an ELM model on the database $\mathcal{D}$. Then, for each image, SPF-SLIC is allowed to compute a single segmentation, using the predicted class k^s. To improve the prediction, we then train a linear model M (line 2). For a given segmentation with superpixel size k^s, M maps the ratio $r^s = C_{Inter}/C_{Intra}$ to the number of classes $\sqrt{k^s} - \sqrt{k^s_{opt}}$ by which k^s differs from the correct class k^s_{opt}. M is trained on every segmentation $k^s \in K$ for each image in $\mathcal{D}$. The improved prediction is then streamed to the ELM.

4. Experimental results

4.1. PF-SLIC

We generate 645, 4K resolution (3840×2160) images by sampling 4K videos every 100 frames, and discarding images which are essentially monochrome or highly correlated with other images in the dataset. We choose $K_{5,18} = \{10^2, 12^2, 14^2, \ldots, 36^2\}$ which was sufficient to capture the range of optimal k^s_{opt}, while also reducing the number of label classes to a feasible number.

Fig. 2 shows some sample images and segmentations in our dataset. The image in Fig. 2(a) is quite detailed, so the relatively small $k^s_{opt} = 256$ is expected. Larger values of k^s fail to capture the smooth boundaries of, for examples, the peoples' body outlines.

The image in Fig. 2(b), has $k^s_{opt} = 400$, the most-common parameter in our dataset. The details of the window shapes are lost for larger k^s, while smaller k^s create unnecessary detail.

Fig. 2(c) shows an image consisting mostly of open water. Naturally, therefore, the image has a large $k^s_{opt} = 900$. We see for $k^s = 900$, the boat's boundaries are well-preserved, while for $k^s = 576$ and $k^s = 256$, the superpixels encode unnecessary detail.

We use the ELM implementation from [29] with Tikhonov-regularized PRESS based on Algorithm 2 in [18]. In training the ELM, we found PRESS optimization generally selected $\tilde{N} \approx 100$ hidden neurons. Regularizing the ELM with a l_1, l_2, or elastic net using MLPack [30] did not significantly improve performance. Among several activation functions such as a triangular basis, hard limit, Gaussian, $\tanh(\cdot)$, and $\sin(\cdot)^2$, tanh provided the best results. Therefore, we trained a basic ELM as in Algorithm 1, with $\tilde{N} = 100$ hidden neurons, and activation function $\sigma(x) = \tanh(x)$. Repeated trials, however, improved performance, with the best time-performance tradeoff at roughly 100 trials. All reported results therefore consider runtime of 100 consecutive trials training a new ELM. The reported results are based on the ELM with the lowest validation error on a validation set comprising 50% of the total training set. This choice mitigates the instability due to the inherent randomness of the ELM. Potential improvements, to be explored in future work, may arise by using Voting-ELM [31].

We compare against standard machine learning models (Decision Tree, Random Forest, Support Vector Machine) implemented in Python's sklearn package [32]. Each model was tuned using a grid search over the parameter space. Fig. 3 shows the performance. The ELM has second-highest classification accuracy ($|\sqrt{k^s} - \sqrt{k^s_{opt}}| = 0$). For incorrect predictions ($|\sqrt{k^s} - \sqrt{k^s_{opt}}| > 0$), the ELM matches the Decision Tree performance quite closely, with slightly worse performance than the Support Vector Machine. Overall, the Random Forest performs best.

One of the greatest advantages to ELM is the rapid runtime. Table 1 details the runtime of ELM as compared to alternatives. Although our implementation of SLIC achieved an average classification time of 7.50 s, an efficient implementation suggests this can be reduced to 40 ms, extrapolating from results in [33]. Under this assumption, a large percentage of the runtime is spent classifying, motivating the need for a computationally-efficient classifier. Note, with sufficient parallel computing power, the time spent clustering can be further reduced by computing each clustering $k^s \in K$ in parallel.

Thus, although Random Forest has best classification performance, it has roughly three times the runtime of ELM. Optimal ELM performance was achieved by taking the best of 100 trials of the basic ELM implementation. Otherwise the runtime of the ELM would be another two orders of magnitude faster than that of the Random Forest. In practice, the ELM performance is easily parallelized, allowing for reduction in runtime. Nevertheless, the superior performance of the Random Forest suggests ensemble ELM methods such as Voting-ELM [31] or an Adaboost-inspired ELM [34] may improve ELM accuracy, while, if removing the need for repeated trials, drastically reducing the runtime.

Together, the results present a speed-accuracy tradeoff. The ELM provides the best tradeoff for applications in which speed is critical.

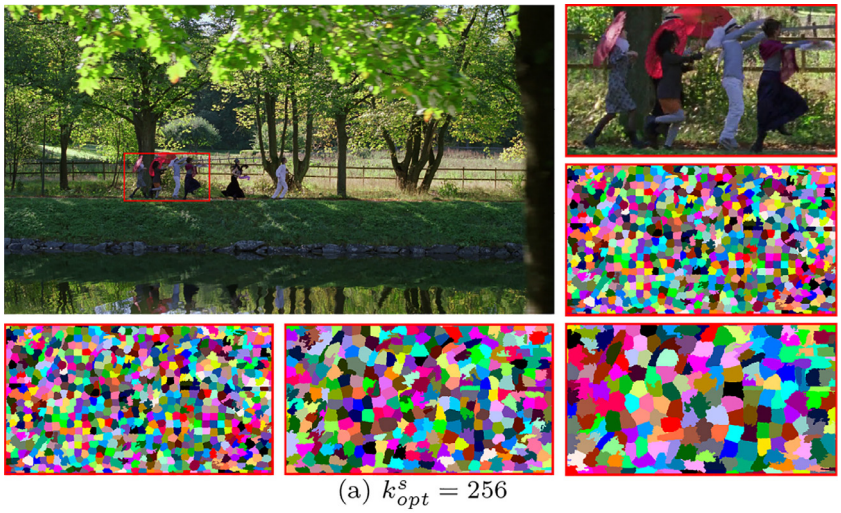
4.1.1. Comparison to ASLIC

We compare the performance of PF-SLIC to the baseline naturally suggested by ASLIC, detailed in Algorithm 6. We use the median of entries in K, $k^s = 529$, as the initial superpixel size used to compute the spatial mean m_s.

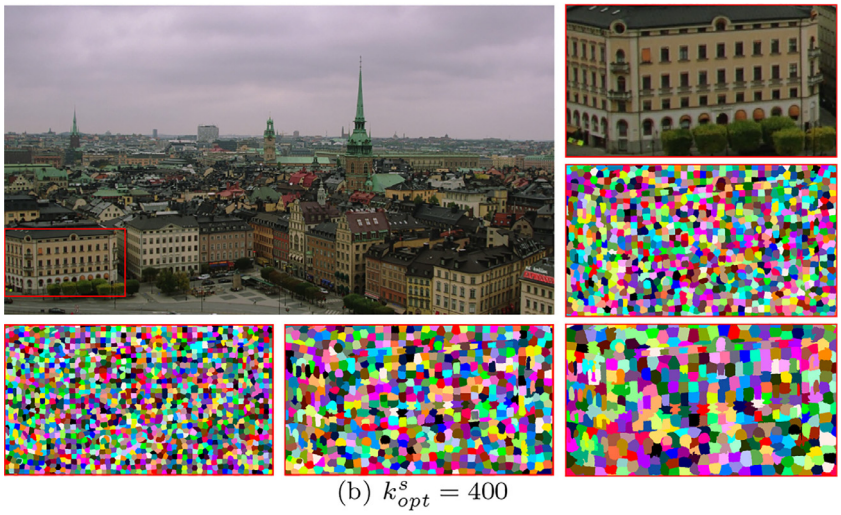
Algorithm 6 ASLIC Parameter tuning.

Require: input image, initial superpixel size k^s.
1: Run ASLIC on input image with N/k^s clusters
2: Compute spatial mean m_s for each superpixel
3: $m_s^{avg} \leftarrow$ average m_s
4: **return** $k^s_{opt} = N/m_s^{avg}$

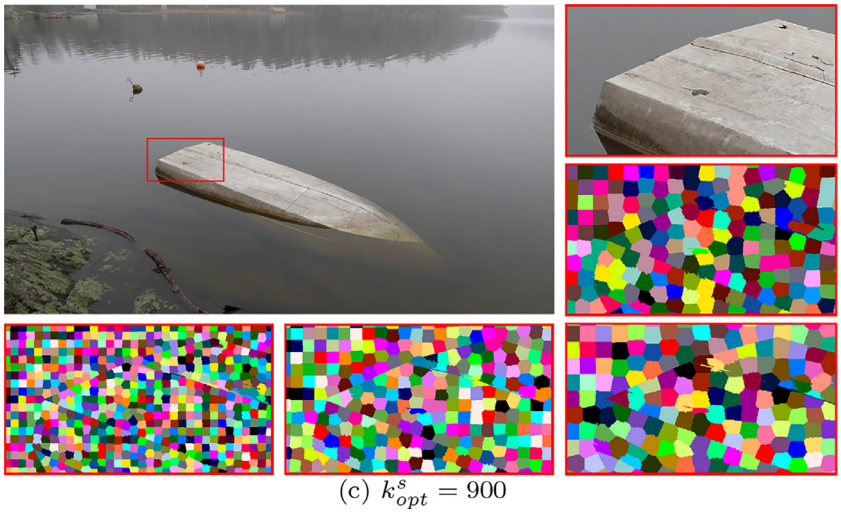
Table 2 summarizes the results. ASLIC runs faster than PF-SLIC, due to the time (3 s) spent constructing features. Feature selection as well as parallel computation of features would greatly reduce the runtime of PF-SLIC to a time comparable to that of ASLIC. However, PF-SLIC produces superpixels close to the optimal, while ASLIC produces superpixel sizes far too small. Fig. 4 shows a training image for which this is particularly evident. See Fig. 2(c) for the PF-SLIC equivalent. We also observe ASLIC produces superpixels which are not consistently compact for small k^s, such as $k^s = 256$. However, for k^s large enough, such as $k^s \in \{576, 1024\}$,



(a) $k^s_{opt} = 256$



(b) $k^s_{opt} = 400$



(c) $k^s_{opt} = 900$

Fig. 2. Example segmentations for various $k^s_{opt} = 256$ (*a*), $k^s_{opt} = 576$ (*b*), $k^s_{opt} = 1024$ (*c*). Each subfigure includes the original 4K image (*top-left*), a cut-out (*top-right*), the best-parameter segmentation (*middle-right*), as well as segmentations for $k^s = 256$ (*bottom-left*), $k^s = 576$ (*bottom-middle*), $k^s = 1024$ (*bottom-right*).

the superpixels are quite compact, forming a hexagonal grid for regions of uniform color.

4.2. SPF-SLIC

To evaluate the performance of SPF-SLIC, we consider OS-ELM, and OS-WELM. In OS-WELM, ω is chosen adaptively as $\frac{n}{n+30}$, where $n = |\mathcal{D}| + i$ is the number of images on which the model has been trained on so far. This places more weight on the latest streamed datapoint. We choose a 30 s video stream of 937 images at 4K resolution, split into 7 scenes of roughly equal length. We use a self-implemented OS-ELM and OS-WELM using python and NumPy [35], and python sklearn implementations of Random Forest (RF) and Support Vector-Machine with Stochastic Gradient

Table 1
Mean and standard deviation of runtimes for Random Forest (*RF*), Decision Tree (*DT*), Support Vector Machine (*SVM*), and ELM, measured on dataset of 645 4K images. Classification time for ELM includes 100 trials. Clustering time is based on values reported in [33]. Actual clustering time using CPU implementation of SLIC was 7.50 ± 0.6 s.

Model	Classification time (s)	Clustering time* (s)	Percentage of time spent classifying
RF	21.26 ± 3.16	0.04	97.43
SVM	7.95 ± 0.34	0.04	93.42
DT	13.03 ± 1.06	0.04	95.88
ELM	6.69 ± 0.87	0.04	92.28

Table 2
PF-SLIC compared to the optimal k_{opt} and ASLIC. Parameter results are taken as the median over the entire dataset. Runtime for PF-SLIC includes feature construction, clustering and classification time. Reported runtimes based on results in [33]. Actual runtimes are $14.19 \pm .75$ for PF-SLIC and 8.33 ± 0.64 for PF-SLIC.

	k^s	$1000 \cdot C_{Intra}$	C_{Inter}	C_{Inter}/C_{Intra}	Runtime* (s)	% time spent classifying
k^s_{opt}	484.0	16.03	6.97	427.62		
PF-SLIC	576.0	13.44	8.10	604.80	3.08	1.29
ASLIC	260.0	29.09	4.86	175.37	0.12	66.66

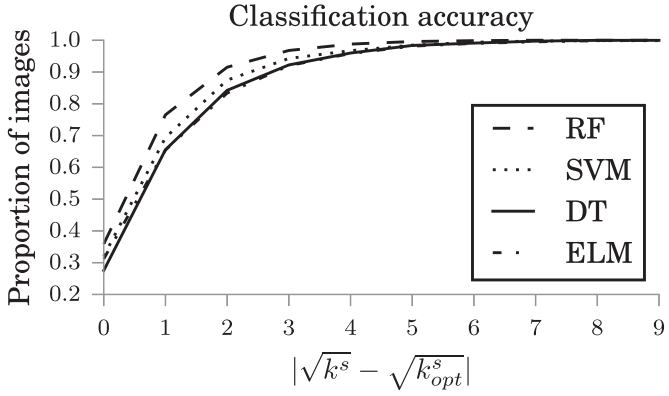


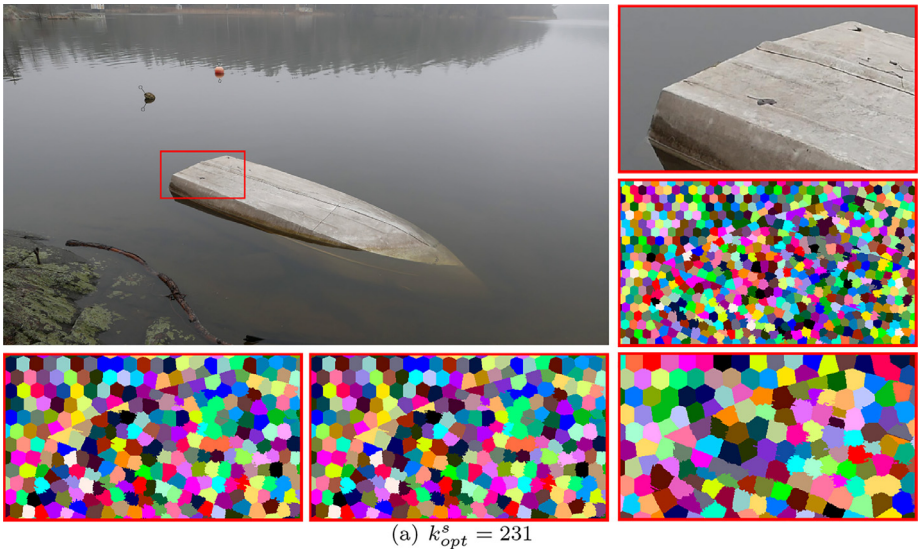
Fig. 3. Performance of the ELM as compared to Random Forest (*RF*), Support Vector Machine (*SVM*), and Decision Tree (*DT*).

Descent (SVM-SGD) [32]. The Random Forest implementation does not provide an online learning framework, so the model must be entirely retrained using previous datapoint for each newly-streamed image, and is therefore not well-adapted to the streaming setting.

We run Algorithm 5 for each scene. Table 3 summarizes the performance of SPF-SLIC using ELM as compared to SPF-SLIC using other models, and Table 4 summarizes the runtime.

Table 3 shows the Random Forest performs best on four of the seven scenes, two more than the next-closest, OS-WELM. However, the Random Forest tends to produce ratios $r = C_{Inter}/C_{Intra}$ larger than optimal. OS-WELM performs best on two of the scenes, but produces ratios which average close to the optimal ratio. That is, OS-WELM seems to be unbiased, while the Random Forest seems to produce biased ratios larger than optimal. SVM-SGD provides the best segmentation for two of the scenes, but has unstable performance, as indicated by the large mean and larger standard deviation for scene 6. The OS-WELM performs better than OS-ELM on all scenes, due to the larger weighting of recently-streamed images. In this video setting, images within the same scene are highly correlated, so the most-recently streamed image provides an important datapoint for predicting on the next image in the stream.

Table 4 shows OS-ELM and OS-WELM are by far the fastest algorithms, two orders of magnitude faster than the Random Forest,



(a) $k^s_{opt} = 231$

Fig. 4. Example segmentations. Each subfigure includes the original 4K image (*top-left*), a cut-out (*top-right*), the best-parameter segmentation (*middle-right*), as well as segmentations for $k^s = 256$ (*bottom-left*), $k^s = 576$ (*bottom-middle*), $k^s = 1024$ (*bottom-right*).

Table 3
Streaming segmentation metrics for each model tested. The ratio reported is $r = C_{Inter}/C_{Intra}$, and the ratios closest to the best ratio for each scene are in bold.

Model	Metrics	Scene							
		0	1	2	3	4	5	6	Avg
OS-ELM	C_{Inter}	6.15 ± 0.43	4.05 ± 0.32	5.58 ± 0.71	18.6 ± 1.12	13.2 ± 2.47	16.5 ± 1.98	10.2 ± 1.79	9.5 ± 5.42
	$1000 \cdot C_{Intra}$	14.5 ± 0.88	10.7 ± 1.23	12.1 ± 2.12	36.7 ± 1.28	26.5 ± 2.96	32.6 ± 2.98	19.9 ± 2.47	19.9 ± 9.73
	ratio	428 ± 45.1	384 ± 41.0	475 ± 95.2	510 ± 47.2	523 ± 262	520 ± 151	531 ± 176	466 ± 57.9
OS-WELM	C_{Inter}	6.24 ± 0.37	4.11 ± 0.32	5.56 ± 0.74	18.6 ± 1.12	12.9 ± 1.45	16.4 ± 1.26	10.1 ± 1.48	9.48 ± 5.37
	$1000 \cdot C_{Intra}$	14.2 ± 1.07	10.5 ± 1.2	12.2 ± 2.04	36.7 ± 1.28	27.0 ± 2.19	32.6 ± 1.22	20.2 ± 2.23	19.9 ± 9.82
	ratio	**444** ± 44.4	397 ± 42.2	471 ± 100	510 ± 47.2	486 ± 137	**506** ± 57.6	512 ± 142	**465** ± 45.5
SVM-SGD	C_{Inter}	6.34 ± 0.53	4.28 ± 0.42	5.43 ± 1.01	18.6 ± 1.12	12.8 ± 1.52	16.4 ± 1.26	11.3 ± 6.63	9.7 ± 5.37
	$1000 \cdot C_{Intra}$	13.9 ± 1.22	9.91 ± 0.55	13.2 ± 4.9	36.7 ± 1.28	27.2 ± 2.79	32.6 ± 1.22	22.1 ± 11.3	20.2 ± 9.95
	ratio	464 ± 132	433 ± 46.3	**457** ± 132	510 ± 47.2	483 ± 144	**506** ± 57.6	1049 ± 2071	559 ± 213
RF	C_{Inter}	6.26 ± 0.37	4.22 ± 0.36	5.71 ± 0.46	18.6 ± 1.12	12.8 ± 1.01	16.5 ± 1.39	10.0 ± 1.51	9.52 ± 5.34
	$1000 \cdot C_{Intra}$	14.1 ± 1.05	10.1 ± 0.69	11.6 ± 0.8	36.8 ± 1.07	27.1 ± 2.73	32.5 ± 1.57	20.3 ± 2.31	19.8 ± 10.0
	ratio	446 ± 43.8	**419** ± 37.7	495 ± 55.5	**508** ± 43.8	**479** ± 88.6	510 ± 73.0	**504** ± 117	472 ± 36.9
Best	C_{Inter}	6.25 ± 0.35	4.24 ± 0.34	5.43 ± 0.19	18.1 ± 1.32	12.7 ± 0.58	15.2 ± 0.88	9.47 ± 1.02	9.21 ± 5.08
	$1000 \cdot C_{Intra}$	14.1 ± 1.02	10.0 ± 0.64	12.3 ± 0.62	38.4 ± 4.2	27.2 ± 1.64	36.3 ± 4.11	21.8 ± 2.76	20.6 ± 10.8
	ratio	444 ± 41.9	424 ± 28.7	442 ± 26.9	478 ± 69.4	468 ± 24.2	428 ± 65.0	440 ± 60.5	444 ± 18.7

Table 4
Mean and standard deviation runtimes of streaming models. Runtimes include only fitting times.

Model	Runtime (s)
OS-ELM	$4.27 \times 10^{-4} \pm 7.0 \times 10^{-5}$
OS-WELM	$4.59 \times 10^{-4} \pm 6.2 \times 10^{-5}$
RF	$4.48 \times 10^{-2} \pm 3.1 \times 10^{-3}$
SVM-SGD	$1.20 \times 10^{-3} \pm 1.6 \times 10^{-4}$

and one order magnitude faster than the SVM-SGD. The runtime of the Random Forest is nearly 100 times that of OS-WELM and ELM. Unlike the Random Forest, OS-WELM and ELM are well-adapted to the streaming setting.

OS-WELM thus provides the best unbiased performance, though the Random Forest typically provides better performance on average. Nevertheless, the two orders of magnitude by which OS-WELM is faster than the Random Forest make OS-WELM much more viable in the streaming setting, where runtime is extremely important.

5. Future work

Future work includes using variations of ELM to improve PF-SLIC and SPF-SLIC. In particular, the superior Random Forest performance in the classic (non-streaming) setting suggests ensemble ELM methods such as Voting-ELM [31] or an Adaboost-inspired ELM [34] may improve PF-SLIC and SPF-SLIC performance. Further work can also compare PF-SLIC and SPF-SLIC performance against image segmentation derived from parameter-free clustering algorithms such as PFClust [36] and, very recently, DSets-DBSCAN [37]. Use of PF-SLIC or SPF-SLIC in subsequent image processing, such as video compression, can also lead to a more refined method of judging image segmentation quality, via the result of the application, rather than r_{opt}.

6. Conclusion

We developed a parameter-free image segmentation algorithm using SLIC superpixel segmentation and an ELM to predict the SLIC parameter k, the number of superpixel clusters. The developed framework, PF-SLIC, predicts much better superpixel segmentations than a baseline model, Adaptive-SLIC (ASLIC). PF-SLIC with ELM performs comparably to PF-SLIC with Random Forest, Decision Tree, and Support Vector Machine, but at a fraction of the runtime. The true benefit to ELM is made clear in the streaming setting. Streaming-PF-SLIC (SPF-SLIC) uses Online-sequential ELM (OS-ELM), an adaptation of ELM to the streaming setting, and a weighted variant, OS-WELM. SPF-SLIC using OS-WELM performs comparably to classical models like Random Forest and Support Vector Machine with Stochastic Gradient Descent, but with orders of magnitude lower training time. The parameter-free image segmentation frameworks PF-SLIC and SPF-SLIC are especially well-suited to processing ultra-high-definition images and videos, whose large size make slower algorithms infeasible.

References

[1] P. Arbelaez, M. Maire, C. Fowlkes, J. Malik, Contour detection and hierarchical image segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 33 (5) (2011) 898–916.
[2] M. Gao, J. Huang, X. Huang, S. Zhang, D.N. Metaxas, Simplified labeling process for medical image segmentation, in: Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2012, pp. 387–394.
[3] D. Chinchkhede, N. Uke, Image segmentation in video sequences using modified background subtraction, Int. J. Comput. Sci. Inf. Technol. 4 (1) (2012) 93–104.
[4] W. Khan, Image segmentation techniques: a survey, J. Image Graph. 1 (4) (2013) 166–170.
[5] D. Jeyakumari, Analysis on current trends in color image segmentation, Int. J. Modern Sci. Eng. Technol. 1 (5) (2014) 1–12.
[6] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, Slic superpixels compared to state-of-the-art superpixel methods, IEEE Trans. Pattern Anal. Mach. Intell. 34 (11) (2012) 2274–2282.
[7] H. Zhu, F. Meng, J. Cai, S. Lu, Beyond pixels: a comprehensive survey from bottom-up to semantic image segmentation and cosegmentation, J. Vis. Commun. Image Represent. 34 (2016) 12–27.
[8] A. Levinshtein, A. Stere, K.N. Kutulakos, D.J. Fleet, S.J. Dickinson, K. Siddiqi, Turbopixels: fast superpixels using geometric flows, IEEE Trans. Pattern Anal. Mach. Intell. 31 (12) (2009) 2290–2297.
[9] O. Veksler, Y. Boykov, P. Mehrani, Superpixels and supervoxels in an energy optimization framework, in: Proceedings of the European Conference on Computer Vision, Springer, 2010, pp. 211–224.
[10] P.F. Felzenszwalb, D.P. Huttenlocher, Efficient graph-based image segmentation, Int. J. Comput. Vis. 59 (2) (2004) 167–181.
[11] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: a new learning scheme of feedforward neural networks, in: Proceedings of the 2004 IEEE International Joint Conference on Neural Networks, Vol. 2, IEEE, 2004, pp. 985–990.
[12] J.B. Roerdink, A. Meijster, The watershed transform: definitions, algorithms and parallelization strategies, Fundam. Inf. 41 (1,2) (2000) 187–228.
[13] J. Shi, J. Malik, Normalized cuts and image segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 22 (8) (2000) 888–905.
[14] A. Vedaldi, S. Soatto, Quick shift and kernel methods for mode seeking, in: Proceedings of the European Conference on Computer Vision, Springer, 2008, pp. 705–718.
[15] A.P. Moore, S.J. Prince, J. Warrell, U. Mohammed, G. Jones, Superpixel lattices, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2008, IEEE, 2008, pp. 1–8.

[16] A. Akusok, K.M. Björk, Y. Miche, A. Lendasse, High-performance extreme learning machines: a complete toolbox for big data applications, IEEE Access 3 (2015) 1011–1025, doi:10.1109/ACCESS.2015.2450498.
[17] Q. Yu, Y. Miche, E. Eirola, M. Van Heeswijk, E. Séverin, A. Lendasse, Regularized extreme learning machine for regression with missing data, Neurocomputing 102 (2013) 45–51.
[18] Y. Miche, M. Van Heeswijk, P. Bas, O. Simula, A. Lendasse, Trop-elm: a double-regularized elm using lars and tikhonov regularization, Neurocomputing 74 (16) (2011) 2413–2421.
[19] G. Huang, G.-B. Huang, S. Song, K. You, Trends in extreme learning machines: a review, Neural Netw. 61 (2015) 32–48.
[20] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, Op-elm: optimally pruned extreme learning machine, IEEE Trans. Neural Netw. 21 (1) (2010) 158–162.
[21] Y. Liang, M. Zhang, W.N. Browne, Image segmentation: a survey of methods based on evolutionary computation, in: Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning, Springer, 2014, pp. 847–859.
[22] Z. Gui, H. Yu, Trip travel time forecasting based on selective forgetting extreme learning machine, Math. Probl. Eng. 2014 (2014) 1–7, doi:10.1155/2014/829256.
[23] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, SLIC Superpixels, EPFL Technical Report 149300, June 2010.
[24] S. Chabrier, B. Emile, H. Laurent, C. Rosenberger, P. Marche, Unsupervised evaluation of image segmentation application to multi-spectral images, in: Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004, Vol. 1, IEEE, 2004, pp. 576–579.
[25] A.M. Nazif, M.D. Levine, Low level image segmentation: an expert system, IEEE Trans. Pattern Anal. Mach. Intell. PAMI-6 (5) (1984) 555–577.
[26] M. Sonka, V. Hlavac, R. Boyle, Image processing, analysis, and machine vision, Cengage Learn. 3 (2014) 723–724.
[27] I. Frosio, E.R. Ratner, Adaptive segmentation based on a learned quality metric, in: Proceedings of the 10th International Conference on Computer Vision Theory and Applications (VISIGRAPP 2015), 2015, pp. 283–292, doi:10.5220/0005257202830292.
[28] J. López-Fandiño, P. Quesada-Barriuso, D.B. Heras, F. Argüello, Efficient elm-based techniques for the classification of hyperspectral remote sensing images on commodity gpus, IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. 8 (6) (2015) 2884–2893.
[29] A. Gritsenko, E. Eirola, D. Schupp, E. Ratner, A. Lendasse, Probabilistic methods for multiclass classification problems, in: Proceedings of ELM-2015, Vol. 2, Springer, 2016, pp. 385–397.
[30] R.R. Curtin, J.R. Cline, N.P. Slagle, W.B. March, P. Ram, N.A. Mehta, A.G. Gray, Mlpack: a scalable c++ machine learning library, J. Mach. Learn. Res. 14 (2013) 801–805.
[31] J. Cao, Z. Lin, G.-B. Huang, N. Liu, Voting based extreme learning machine, Inf. Sci. 185 (1) (2012) 66–77.
[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[33] R. Birkus, Accelerated gslic for superpixel generation used in object segmentation, Proceedings of CESCG 2015, 2015, pp. 27–30.
[34] H.-X. Tian, Z.-Z. Mao, An ensemble elm based on modified adaboost. rt algorithm for predicting the temperature of molten steel in ladle furnace, IEEE Trans. Autom. Sci. Eng. 7 (1) (2010) 73–80.
[35] S. Van Der Walt, S.C. Colbert, G. Varoquaux, The numpy array: a structure for efficient numerical computation, Comput. Sci. Eng. 13 (2) (2011) 22–30.
[36] L. Mavridis, N. Nath, J.B. Mitchell, Pfclust: a novel parameter free clustering algorithm, BMC Bioinf. 14 (1) (2013) 1–21.
[37] J. Hou, H. Gao, X. Li, Dsets-dbscan: a parameter-free clustering algorithm, IEEE Trans. Image Process. 25 (7) (2016) 3182–3193.

Fabian Boemer is an undergraduate student at the California Institute of Technology, where he will graduate with a BS in Computer Science and Applied and Computational Mathematics in June 2017. Fabian served as a research and development intern at Lyrical Labs in 2016.

Ed Ratner is the CTO and founder of Lyrical Labs. Ed has over 15 years of industry experience in creating novel algorithms from conception through development to product with several products deployed to a wide customer base. His work over the course of his career has resulted in over 30 U.S. patents. Ed was elevated to Senior Member of Institute of Electrical and Electronics Engineers in 2004. During his undergraduate studies at Caltech, he received the Froehlich Prize. He was also a Hertz Foundation Fellow at Stanford University where he received his Ph.D. In 2009, he was listed as one of Caltech's notable alumni by Forbes magazine.

Amaury Lendasse was born in 1972, in Belgium. He received a M.S. degree in Mechanical Engineering from the Universite Catholique de Louvain (Belgium) in 1996, a M.S. in Control in 1997 and a Ph.D. in Applied Mathematics in 2003 from the same university. In 2003, he was a postdoctoral researcher in the Computational Neurodynamics Lab at the University of Memphis. From 2004 to 2014, he was a senior researcher and an Adjunct Professor in the Adaptive Informatics Research Centre in the Aalto University School of Science (better known as the Helsinki University of Technology) in Finland. He has created and lead the Environmental and Industrial Machine Learning at Aalto. He is now an Associate Professor at The University of Iowa (USA) and a visiting Professor at Arcada University of Applied Sciences in Finland. He was the Chairman of the annual ESTSP conference (European Symposium on Time Series Prediction) and member of the editorial board and program committee of several journals and conferences on machine learning. He is the author or coauthor of more than 200 scientific papers in international journals, books or communications to conferences with reviewing committee. His research includes Big Data, time series prediction, chemometrics, variable selection, noise variance estimation, determination of missing values in temporal databases, nonlinear approximation in financial problems, functional neural networks and classification.