BRISKOLA: BRISK OPTIMIZED FOR LOW-POWER ARM ARCHITECTURES

Luca Baroffio, Antonio Canclini, Matteo Cesana, Alessandro Redondi, Marco Tagliasacchi

Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Piazza Leonardo da Vinci, 32 - 20133 Milano - Italy

ABSTRACT

Local visual features are commonly adopted to accomplish analysis tasks such as object recognition/tracking and image retrieval. Recently, several visual features extraction algorithms tailored to low-power architectures have been proposed, in order to enable image analysis on energy-constrained devices such as smart-phones or Visual Sensor Networks (VSN). In this work, we dissect and analyze BRISK, a state-of-the-art low-power visual feature extractor, in order to evaluate the impact of its individual building blocks on the overall energy consumption. For each building block, we propose a solution to limit the energy consumption without affecting the overall analysis performance. The resulting BRISKOLA (BRISK Optimized for Low-power ARM architectures) feature extractor exhibits energy savings up to 30% with respect to the original implementation.

Index Terms— Local Visual Features, BRISK, Visual Sensor Networks, ARM

1. INTRODUCTION

In the last few years, we have assisted to an enormous proliferation of battery-operated intelligent devices capable of sensing and analyzing visual data from the environment. Besides smart-phones, many other examples can be mentioned, such as unmanned aerial vehicles and visual sensor networks. The latters are expected to play a major role in the Internet-of-Things paradigm, by supporting applications such as ambient assisted living, environmental monitoring and many other scenarios were image/video analysis are commonly employed. The majority of these visual analysis tasks is carried out by extracting a set of visual features from the pixel-domain content by means of specialized computer vision algorithms. Such visual features are then further processed in order to obtain a semantic representation of the original visual content, e.g., by recognizing the objects present in the scene or through the detection of a particular event occurring in the scene.

Energy efficiency is a major issue in this kind of applications. Indeed, especially when the application scenario requires to sense, process and transmit visual data, a careful selection of the underlying hardware and software tools is of paramount importance for meeting specified lifetime requirements. In the field of visual features extraction, many works in the past tried to alleviate the energy burden of the SIFT algorithm [1], widely accepted as a gold standard. Alternative algorithms such as SURF [2] heavily reduce the complexity of SIFT, with little impact on the discriminative power of the extracted features. Recently, very low complexity algorithms were presented for detecting features in an image (FAST [4], AGAST [5]) and for obtaining a fast, binary representation of such features (BRIEF [6], ORB [3], BRISK [7], FREAK [8]). The interested reader may refer to [9] and [10] for detailed evaluations and comparisons of the existing visual features algorithms, both in terms of their complexity as well as their accuracy and efficiency. In this paper, we focus on BRISK, which shows a performance approaching that of SIFT at a fraction of its computing cost. Our aim is to understand the complexity of each BRISK building block and propose modifications to further limit the energy expenditure, in the specific case in which the the target hardware platform is based on ARM architecture, as it is the case of many today's smart-phones and visual sensor network platforms. The result of our optimizations is named BRISKOLA (BRISK Optimized for Low-power ARM architectures), an improved version of BRISK which is made publicly available for download[1]. This paper is organized as follows: in Section 2 we review the main building blocks of the BRISK algorithm, analyzing experimentally their impact on the total energy expenditure. In Section 3 we propose, for each building block, an energy-aware optimization and show the obtained energy savings. In Section 4 the original implementation and the proposed optimization are compared in a rate-energy framework and finally, in Section 5, we conclude the paper and spot future research directions.

2. DISSECTING BRISK

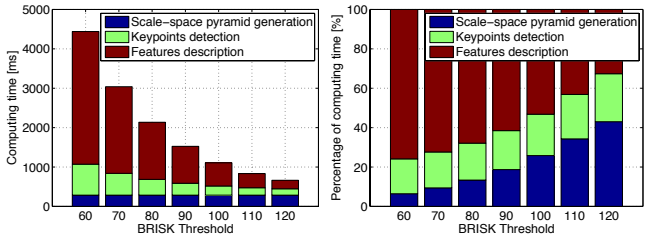
The Binary Robust Invariant Scalable Keypoints (BRISK) algorithm was proposed by Leutenegger et al. in 2011 as an alternative for high-efficient local visual features extraction. Like its well-known predecessor (e.g., SIFT, SURF), the algorithm follows a two-steps process: in the *detection* step, a scale-invariant detector identifies salient points (keypoints) across both the spatial and scale dimensions. Then, for each detected keypoint, the BRISK *descriptor* produces a fixed-length binary representation by concatenating the results of pairwise brightness comparisons obtained using a specific sampling pattern. In the following we analyze in details these two building blocks, focusing on the energy-critical aspects which are the subjects of our optimizations.

2.1. BRISK detector

The BRISK detector is a scale-invariant version of the FAST [4] keypoint detection. FAST is based on the Accelerated Segment Test (AST), which classifies a candidate point p (with intensity I_p) as a keypoint if n contiguous pixels in the Bresenham circle of radius 3

The project GreenEyes acknowledges the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under FET-Open grant number:296676.

[1] `http:\\www.greeneyesproject.eu`



(a) (b)

Fig. 1. (a) CPU processing time and (b) percentage of CPU usage of the individual BRISK building blocks on a SVGA image (1024x768 pixels)

around p are all brighter than $I_p + t$, or all darker than $I_p - t$, with t a predefined threshold. Thus, the highest the threshold, the lowest the number of keypoints which are detected and vice-versa. Each keypoint is finally given a score s, defined as the largest threshold for which p is classified as a keypoint.

2.1.1. Scale-space pyramid generation

Similarly to SIFT and SURF, scale-invariance is achieved in BRISK by building a scale-space pyramid consisting of a pre-determined number of octaves and intra-octaves. Each octave is formed by progressively half-sampling the original image, while the intra-octaves are obtained by downsampling the original image by a factor of 1.5 and by successive half-samplings. The original BRISK implementation from the authors[2] uses 4 octaves and shows high computation performance due to the use of Intel SSE2 and SSE3 expressions concerning the halfsampling as well as the 2/3 downsampling.

2.1.2. Keypoints detection

Once the scale-space pyramid is built, the FAST detector is applied on each octave and intra-octave separately to identify potential regions of interest. Each candidate keypoints is subjected to a non-maxima suppression in the scale-space, by comparing its score with the score of the 8 neighboring pixel in the same layer, and in the layers above and below. The candidate point is marked as keypoint if its score s is higher than the scores of all its neighbors. The keypoint score is considered as a continuous quantity in both the spatial and scale dimensions: in order to perform non maxima suppression, first saliency maxima are obtained with a 2D parabolic least-squares fitting in each layer separately. Then, the obtained values are interpolated with a 1D parabola in the scale dimension, to obtain the final scale estimate σ. As a final step, the location of the keypoint is obtained by re-interpolating the image coordinates between the patches in the layers next to the original scale.

2.2. BRISK descriptor

Similarly to other binary descriptors [6] [8], a BRISK descriptor is computed for each detected keypoint by concatenating the results of brightness comparison tests. The tests derive from a circular sampling pattern centered in each keypoint location.The pattern is composed by N sampling points $\mathbf{p}_i \in \mathbb{R}^2$, $i = 1, \ldots, N$, defined by the set:

$$\mathcal{A} = \{(\mathbf{p}^i, \mathbf{p}^j) \in \mathbb{R}^2 \times \mathbb{R}^2 | i < N \wedge j < N \wedge i, j \in \mathbb{N}\}. \quad (1)$$

[2]http://www.asl.ethz.ch/people/lestefan/personal/BRISK

To achieve scale and rotation invariance, before computing the final descriptor the sampling pattern is scaled according to the estimated scale σ and rotated according to the estimated orientation of the particular keypoint, θ.

2.2.1. Orientation estimation

Let $\mathcal{I}(\mathbf{p}_i, \rho_i)$ denote a smoothed intensity value obtained by averaging the pixel values at locations around $\mathbf{p}_i$. Although different averaging filters can be used, the publicly available implementation of BRISK adopts a simple box mean filter with floating point boundaries and side length equal to ρ_i, which depends on the distance from the center of the sampling pattern. To estimate the orientation of a patch, BRISK computes the local gradient for each one of the so-called long-pairs, defined as

$$\mathcal{L} = \{(\mathbf{p}^i, \mathbf{p}^j) \in \mathcal{A} | \|\mathbf{p}^j - \mathbf{p}^i\| > \delta_{min}\}, \quad (2)$$

and estimate the overall pattern direction by computing the arctangent of the average of the local gradients.

2.2.2. Descriptor computation

After scaling and rotating the sampling pattern, it is possible to compute up to $N(N-1)/2$ binary comparisons, i.e., one for each pair in $\mathcal{A}$. That is,

$$b = \begin{cases} 1, & \mathcal{I}(\mathbf{p}^j, \rho_j) > \mathcal{I}(\mathbf{p}^i, \rho_i) \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

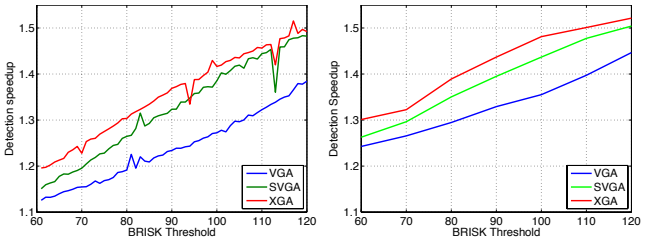
BRISK uses the set of short pairs in the sampling pattern to compute the descriptor, which are defined as:

$$\mathcal{S} = \{(\mathbf{p}^i, \mathbf{p}^j) \in \mathcal{A} | \|\mathbf{p}^j - \mathbf{p}^i\| < \delta_{max}\}. \quad (4)$$

The threshold δ_{max} is set so that the final descriptor size is 512 bits. Note that the descriptor dimension may be tuned by properly selecting how many and which pairs to compare [11].

2.3. Energy analysis

To understand the computational weight of each step in the BRISK pipeline, we have run a series of tests on a BeagleBone embedded computer. The BeagleBone platform features an ARM Cortex-A8 processor, with 256 MB of DDR2 RAM and is capable of running a Linux based operating system, such as Ubuntu. Due to the possibility of stacking on top of the main board several additional interfaces such as a camera and a wireless radio communication interface (e.g., WiFi, Zigbee), the Beaglebone is particularly suited as a reference platform for Visual Sensor Networks. Moreover, it shows very low power consumption, limited to 1.75 W at 700 MHz [12], and has the possibility of being battery operated. The tests were executed on images with different resolutions and varying the BRISK threshold t, which controls the number of keypoints which are detected and consequently the number of descriptors produced. Figure 1 shows the computational complexity of each step of the BRISK algorithm for a SVGA input image (1024x768 pixels) at different detection thresholds. As one can see, at low thresholds (when the number of detected keypoints is high), description is the most intensive phase (constituting up to 78% of the total processing time). Conversely, at high thresholds (when the number of detected keypoints is high), detection is the is the most intensive step. Overall, the keypoint detection step takes about one fifth of the total time to be executed. We repeated the experiment also for different image resolutions, obtaining similar results.



(a) (b)

Fig. 2. (a) Speedup obtained with the NEON implementation of the scale-space pyramid generation routines (b) Total detection speedup including the NEON optimization and the proposed non-maxima suppression method

3. OPTIMIZING BRISK

In this section, we introduce an optimization for each building block of the BRISK algorithm and we show the corresponding speed-ups and energy savings obtained on a BeagleBone platform. The main assumption is that the consumed energy is directly proportional to the time spent during the computation. Thus, reducing the processing time will reduce the energy consumption by the same factor, as long as the CPU frequency remains fixed during the processing time[3]. The resulting modified algorithm is named BRISKOLA (BRISK Optimized for Low-power ARM Architectures).

3.1. BRISKOLA Detector

3.1.1. Scale-space pyramid generation

The original implementation from the authors of BRISK uses Intel SSE instructions to optimize the computation of the scale-space pyramid. On ARM architectures such optimizations cannot be used and the routines responsible for the pyramid constructions must be converted to standard C language[4], with a clear loss in performance. Hence, as a first optimization we completely rewrote the pyramid generation routines in order to use ARM specific SIMD instruction set, named NEON. NEON is included in all Cortex-A8 devices (including the BeagleBone) and features a comprehensive instruction set, separate register files and independent execution hardware. Figure 2(a) shows the speedup obtained by our NEON implementation of the scale-space pyramid generation routines with respect to the standard C one, for what concerns the keypoints detection step. As one can see, we obtained speedups from 1.1 to 1.5 times, depending on the test configuration. This optimization is in fact more effective at high thresholds and for high image resolution, i.e., when the impact of generating the scale-space pyramid is high with respect to the actual keypoints detection process.

3.1.2. Keypoints detection

As explained in Section 2.1.2, for each candidate keypoint a non-maxima suppression step is performed in a scale-space volume of pixels. The score of the candidate keypoint is compared to the score of its 8 neighbors in the same layer, and to the scores of the maxima in the layers above and below. Such maxima are obtained after a 2D parabolic interpolation in space. We observed that the interpolation

[3]we used the *cpufreq* tool on the BeagleBone to fix the CPU speed to 700 MHz

[4]as it is done for the BRISK version included in the latest OpenCV release

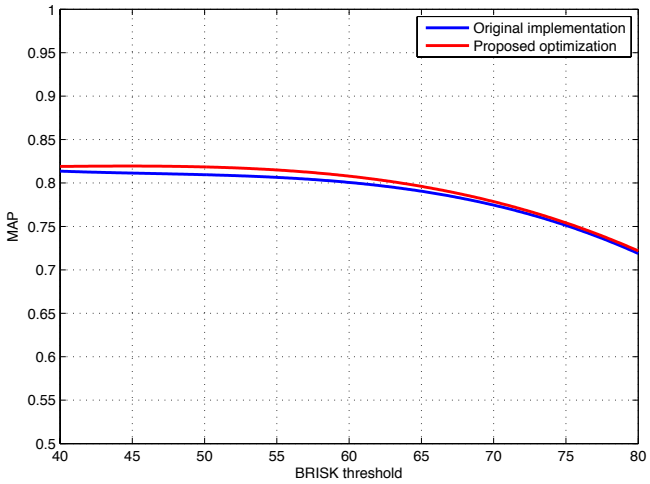


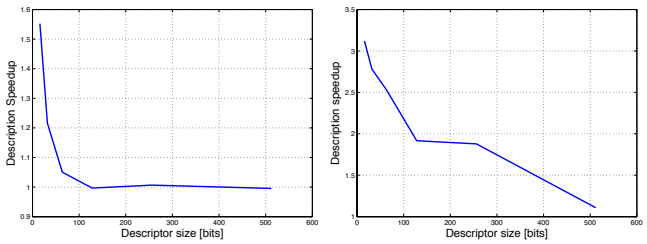
Fig. 3. MAP comparison of the original implementation and the proposed modification for the non-maxima suppressio step on the ZuBuD dataset

operations are particularly intensive, and constitute up to 15% of the total detection time. In a scenario where energy is scarce, such as the one envisioned by VSNs, the cost of these interpolations may be too high, especially when the position of each keypoint does not need to be known with fine sub-pixel accuracy. In BRISKOLA, we propose to perform non-maxima suppression in a simpler way. We avoid the 2D interpolation in space on the layers above and below the selected one and we compare the score of the candidate keypoint with the value obtained by simply picking the scores at the same location in the layers above and below. This modification clearly impacts on the final detection results as a keypoint may now be rejected or accepted due to the different result of the non-maxima suppression step. Moreover, it's location may change due to the final re-interpolation step, which now operates on different maxima values. To quantify the impact of this change on the analysis performance, we set up an image retrieval pipeline based on the ZuBuD building images dataset [13] and we computed the Mean of Average Precision over a set of image queries when using the original BRISK algorithm or our proposed modification at different detection threshold. As one can see from Figure 3, the proposed optimization obtains even better performance than the original BRISK implementation. The final speedup obtained for the detection process, including the scale-space pyramid generation with NEON instruction and the proposed modification of the non-maxima suppression step is illustrated in Figure 2(b). As one can see, this latter optimization allows to increase the speedups at low thresholds and low resolutions, now in the range of 1.25 - 1.3 times with respect to the original implementation.

3.2. BRISKOLA Descriptor

3.2.1. Descriptor computation

For each detected keypoint, the original implementation of BRISK produces a 512-bits descriptor by concatenating the results of the intensity comparisons of the short pairs, as explained in Section 2.2.2. However, one may want to change the length of each descriptor, e.g, to trade-off bandwidth with accuracy [11], or to optimize the rate-accuracy behavior of the application. In [14], we introduced the concept of *internal allocation* for visual analysis tasks such as object recognition, which states that the optimal descriptor length



(a) (b)

Fig. 4. (a) Speedup obtained by computing the smoothed intensity values only on the input pairs (b) Speedup obtained including the removal of the rotation estimation process

changes as the available bandwidth increases. In a nutshell, when the available bandwidth is low, the optimal strategy is to perform object recognition using few long descriptors. Conversely, as the available bandwidth increases, the optimal rate-accuracy point is reached using many short-sized descriptors. BRISKOLA is designed to efficiently produce variable length descriptors through the following two modifications with respect to the original BRISK implementation:

- Instead of tuning the descriptor length via a distance threshold as done in BRISK, in BRISKOLA it is possible to directly specify the intensity pairs to use in order to build the descriptor. Such pairs may be learnt according to different methods [11].
- As explained in Section 2.2.2, the comparisons are performed on smoothed intensity values, obtained through box filters. In the original BRISK implementation, such filtering operations were performed on all the points in the pattern. Conversely, in BRISKOLA we leverage the knowledge of the input pairs to compute the smoothed intensity values only on those points which are effectively used in the descriptor computation. Figure 4(a) shows the speedup obtained using the proposed optimization: as one can see, for short descriptors (i.e., less than 100 bits/descriptor), notable speedups may be achieved.

3.2.2. Orientation estimation

As a last thing, we analyzed the computational cost of the orientation estimation, which is needed for applications which require invariance to rotation. We observed that this is a highly consuming process: Figure 4(b) the total description speedup when the orientation estimation process is disabled. As one can see, it is possible to obtain a speedup in the order of 2 to 3 times for short descriptors. Note that for Visual Sensor Networks application, where cameras are usually fixed (e.g., attached on walls/ceilings/light posts) rotation invariance is not a strict requirements and may be easily disregarded without affecting the analysis performance.

4. RATE-ENERGY COMPARISON

In order to understand in a better way the improvement of BRISKOLA with respect to the original implementation, we have set up an experimental testbed composed by a BeagleBone-based Visual Sensor Network. The testbed is responsible for acquiring images, extracting visual features and transmitting them to a central controller using a IEEE 802.15.4 compliant radio interface. At the central controller, these features are matched against a database of labelled features to perform object recognition. The input images were taken from the widely known ZuBud and Oxford dataset. Relying on the concept of *internal allocation*, for each image query in the dataset we were able to obtain the optimal number and length of BRISK/BRISKOLA features to extract and transmit in order to obtain the maximum accuracy in the recognition task at a determined target bitrate (see [14] for further details). This allowed us to draw curves in the rate-energy plane, taking into account both the cost of detecting keypoints from the input image and the cost of building the descriptor for each detected keypoints. Energy costs were estimated from the knowledge of the time spent during each phase of the features extraction process (as shown in Figure 1), by multiplication with the BeagleBone peak power consumption of 1.75 W. Figure 5 shows the rate-energy behavior for the ZuBuD dataset, i.e., the expected consumed energy for each query at different target bitrates. As one can see our BRISKOLA implementation outperforms the original BRISK one, with energy savings in the order of 30%. Similarly, we also drawn the energy-rate curve corresponding to BRISKOLA when the rotation estimation was disabled: this time the energy savings with respect to the original implementation are as high as 60%. We repeated the same experiment also for the Oxford[5] dataset and obtained similar results (which are omitted here for reasons of space).

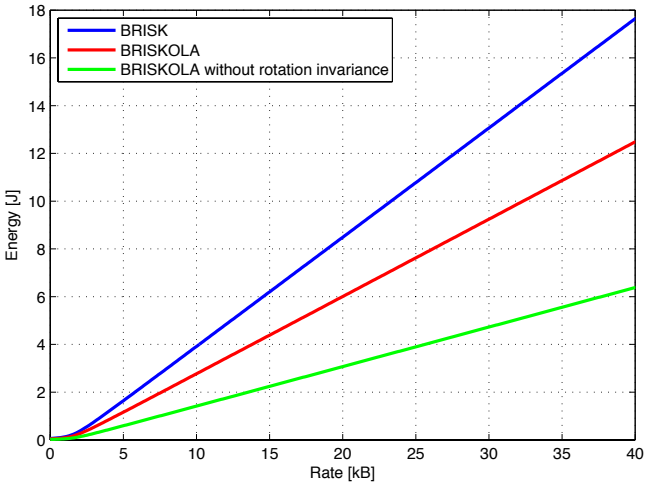


Fig. 5. Rate-energy curves for the ZuBuD dataset.

5. CONCLUSIONS

In this work we proposed the BRISKOLA visual features extractor, an optimized version of BRISK tailored to low-power ARM architectures. The proposed optimization allows to obtain speedups and hence energy savings in the order of 30% with respect to the original implementation. Moreover we highligthed that orientation estimation is one of the most energy-consuming task in rotation-invariant visual features extraction. Future research direction will study efficient methods for orientation estimation. The source code of BRISKOLA is publicly available for download at `http:\\www.greeneyesproject.eu`.

[5] http://www.robots.ox.ac.uk/ vgg/data/oxbuildings/

6. REFERENCES

[1] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[2] Herbert Bay, Tinne Tuytelaars, and Luc J. Van Gool, "Surf: Speeded up robust features," in *ECCV (1)*, 2006, pp. 404–417.

[3] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski, "Orb: An efficient alternative to sift or surf," in *ICCV*, 2011, pp. 2564–2571.

[4] Edward Rosten, Reid Porter, and Tom Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 1, pp. 105–119, 2010.

[5] Elmar Mair, Gregory D. Hager, Darius Burschka, Michael Suppa, and Gerd Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test," in *ECCV (2)*, 2010, pp. 183–196.

[6] Michael Calonder, Vincent Lepetit, Mustafa Özuysal, Tomasz Trzcinski, Christoph Strecha, and Pascal Fua, "Brief: Computing a local binary descriptor very fast," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 7, pp. 1281–1298, 2012.

[7] Stefan Leutenegger, Margarita Chli, and Roland Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *ICCV*, 2011, pp. 2548–2555.

[8] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst, "Freak: Fast retina keypoint," in *CVPR*, 2012, pp. 510–517.

[9] A. Canclini, M. Cesana, A. Redondi, M. Tagliasacchi, J. Ascenso, and R. Cilla, "Evaluation of low-complexity visual feature detectors and descriptors," in *IEEE International Conference on Digital Signal Processing 2013*, 2013, pp. 900–903.

[10] Jared Heinly, Enrique Dunn, and Jan-Michael Frahm, "Comparative Evaluation of Binary Features," in *Computer Vision ECCV 2012*, Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, Eds., Lecture Notes in Computer Science, pp. 759–773+. Springer Berlin Heidelberg, 2012.

[11] A. Redondi, L. Baroffio, J. Ascenso, M. Cesana, and M. Tagliasacchi, "Rate-accuracy optimization of binary descriptors," in *IEEE International Conference on Image Processing*, sept. 2013.

[12] Gerald Coley, "Beaglebone rev a6 system reference manual," 2012.

[13] H. Shao, T. Svoboda, and L. Van Gool, "ZuBuD — Zurich buildings database for image based recognition," Tech. Rep. 260, Computer Vision Laboratory, Swiss Federal Institute of Technology, Mar. 2003.

[14] Alessandro Redondi, Matteo Cesana, and Marco Tagliasacchi, "Rate-accuracy optimization in visual wireless sensor networks," in *ICIP*, 2012, pp. 1105–1108.