FAST ALGORITHMS FOR THE COMPUTATION OF RANKLETS

Fabrizio Smeraldi

Queen Mary University of London
School of Electronic Engineering and Computer Science
Mile End Road, London E1 4NS, UK

ABSTRACT

Ranklets are orientation selective rank features with applications to tracking, face detection, texture and medical imaging. We introduce efficient algorithms that reduce their computational complexity from $O(N \log N)$ to $O(\sqrt{N} + k)$, where N is the area of the filter. Timing tests show a speedup of one order of magnitude for typical usage, which should make Ranklets attractive for real-time applications.

***Index Terms*—** Ranklets, Rank features, Orientation decomposition, Wilcoxon statistics, Distribution counting sort

1. INTRODUCTION

Rank features have long been appreciated by the Computer Vision community because of their invariance to monotonic brightness transformations, that results in an outstanding robustness. After the classical applications to wide-baseline stereo matching [1, 2] and texture segmentation [3], rank features have recently known a renewed interest in connection with object recognition [4, 5].

Ranklets introduced Haar wavelet–style orientation selectivity to rank features [6]. They are based on the Wilcoxon statistics and have an interpretation in terms of pairwise pixel comparisons; applications include face detection [4], point tracking [7], texture classification [8] and the detection of masses in mammographic images [9, 10]. One of the disadvantages of rank features is their relatively high computational cost. Ranklets were introduced as having $O(N \log N)$ complexity, were N is the number of pixels in the support window of the filter [6]. While this is relatively low, it has arguably hindered the application to real-time or high-throughput image analysis, where linear or constant-time descriptors are preferred (see for instance [11]). The complexity quoted was based on the computation of the Wilcoxon statistics using the Quicksort algorithm. To the best of our knowledge, most applications have so far been based on Quicksort, many of them using the C library originally developed by the author of [6] or closely related implementations.

We introduce alternative, faster algorithms for the computation of Ranklets. Specifically, we note that due to the quantised nature of image brightness, the Distribution Counting algorithm [12] can be used to compute Ranklets with linear complexity. This can be further improved in the case that the same Ranklet needs to be computed over the entire image, leading to $O(\sqrt{N} + k)$ complexity and a dramatic speedup.

In the case that Ranklets with different support windows must be computed at sparse image locations a caching mechanism using sparse matrices (in some way an extension of the Integral Image of Viola and Jones [11]) also achieves $O(\sqrt{N})$ complexity, albeit at the cost of a preprocessing step that is linear in the size of the image.

We present a detailed discussion of these algorithms and of an Insertion Sort alternative and report timing experiments on standard database images.

2. BACKGROUND: RANKLETS

Ranklets are a family of multiscale rank filters that display an orientation selectivity pattern similar to Haar wavelets. Like all other rank features, they are defined on the relative order of pixel intensity values rather than on intensity itself; thus their computation entails a sorting operation.

With reference to Figure 1, consider the three Haar wavelets $h_i(\vec{x}), i = 1, 2, 3$ supported on a local window W. Let $\mathsf{T}_i = h_i^{-1}(\{+1\})$ and $\mathsf{C}_i = h_j^{-1}(\{-1\})$ be the counter-images of $+1$ and -1 in the three cases; the idea is to perform a nonparametric comparison of the relative brightness of these two regions, for each orientation i.

Let $\pi(\vec{x})$ be a ranking of the pixels in W according to their intensity $I(\vec{x})$ (we are for the moment assuming that no two pixels share the same intensity; this restriction will be lifted in Section 4.1 below). We compute the Wilcoxon statistics $\mathcal{W}_s^i = \sum_{\vec{x} \in \mathsf{T}_i} \pi(\vec{x})$ for the comparison of the intensity of the pixels in T_i and C_i; $\mathcal{W}_s^i$ is an increasing function of the relative brightness of T_i [13]. We centre $\mathcal{W}_s^i$ around its average value by computing $\mathcal{W}_{YX}^i = \mathcal{W}_s^i - (N/2+1)N/4$, where N is the number of pixels in W. The Mann-Whitney statistics $\mathcal{W}_{YX}^i$ is equal to the number of pairs of pixels $(\vec{x}, \vec{y})$, with $\vec{x} \in \mathsf{T}_i$ and $\vec{y} \in \mathsf{C}_i$, such that $I(\vec{x}) > I(\vec{y})$ (for a proof, see [13]); its value thus ranges from 0 to $N^2/4$. We define Ranklets as $\mathcal{R}_{\mathsf{W}}^i = 8\,\mathcal{W}_{YX}^i / N^2 - 1$, so that their value ranges from -1 to $+1$ as the contrast between T_i and C_i is reversed.

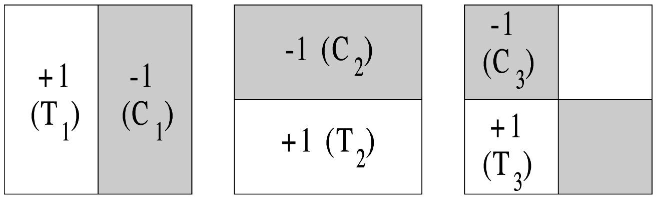

Fig. 1. The three two-dimensional Haar wavelets $h_1(\vec{x})$, $h_2(\vec{x})$ and $h_3(\vec{x})$ (from left to right). Letters in parentheses refer to the "treatment" and "control" pixel sets.

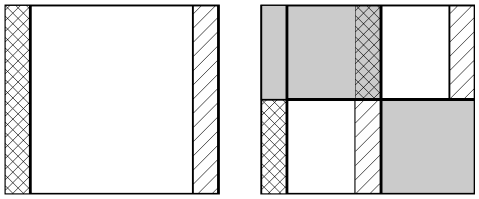

Fig. 2. Histogram update for H_{W} (left) and H_{T} for a diagonal Ranklet (right). The dashed areas are added to the histograms, while the hatched areas are removed.

3. DISTRIBUTION COUNTING

The computational cost of Ranklets is dominated by the ranking π. In [6] this is put at $O(N \log N)$, i.e. the complexity of the Quicksort algorithm. However, in the case of quantities with a limited set of values such as pixel intensities, sorting by accumulation is a more efficient strategy.

Let $\mathcal{H}_{\mathsf{W}}(i) = \#\{\vec{x} \in \mathsf{W} | (\vec{x}) \leq i\}$ be the cumulative histogram of W. Then the rank of pixel $\vec{x}$ is bound by

$$\mathcal{H}_{\mathsf{W}}(I(\vec{x}) - 1) + 1 \leq \pi(\vec{x}) \leq \mathcal{H}_{\mathsf{W}}(I(\vec{x})), \qquad (1)$$

with the added convention that $\mathcal{H}_{\mathsf{W}}(-1) = 0$. This is known as the Distribution Counting algorithm [12]; the equality signs apply to the case that exactly one pixel in W has intensity i. In the case of tied measurements, it is common practice in rank statistics to introduce average ranks π^* (known as midranks [13]). These are easily obtained by setting

$$\pi^*(\vec{x}) = 1/2 \left(\mathcal{H}_{\mathsf{W}}\left(I(\vec{x}) - 1\right) + \mathcal{H}_{\mathsf{W}}\left(I(\vec{x})\right)\right) + 1/2. \qquad (2)$$

The Distribution Counting algorithm has complexity $O(N + \ell)$ where the number of grey levels ℓ accounts for the integration of the histogram to obtain $\mathcal{H}_{\mathsf{W}}$ (or π^*).

Distribution counting is efficient for $N \ll \ell$, which is not true for windows W of small size. In this case, it is convenient to keep track of the minimum and maximum intensity in W to reduce the computation of π^* to this interval. Because of the statistical properties of natural images, we can expect the effective range of intensities $\ell^* = I_{max} - I_{min} + 1$ to be much smaller than ℓ.

4. WHOLE IMAGE FILTERING

When the same filter is to be applied to the entire image some optimisations are possible, leading to the Incremental Distribution Counting algorithm introduced below. We compare this algorithm with a similarly optimised Insertion Sort and the reference Quicksort.

4.1. Incremental Distribution Counting

In the case of Distribution Counting, considerable computational savings are possible by exploiting the almost complete overlap between two consecutive positions of the support window W as it scans the image [14, 15]. To this end, one only needs to maintain the histogram H_{W} of W together with the histogram H_{T_i} of the T_i for each of the three ranklet orientations $i = 1, 2, 3$. When W is slid one pixel to the right, the histograms can be efficiently updated as illustrated in Figure 2. Let W′ denote the translated window: the pixels in $\mathsf{W} \setminus \mathsf{W}'$ are subtracted from H_{W}, while those in $\mathsf{W}' \setminus \mathsf{W}$ are added (I_{min} and I_{max} also need to be updated). Similarly, for each of the three orientations, the pixels in $\mathsf{T}_i \setminus \mathsf{T}'_i$ are discounted from H_{T_i}, while those in $\mathsf{T}'_i \setminus \mathsf{T}_i$ are added. Figure 2 (right) shows the most complicated case, i.e. a diagonal Ranklet.

For each position of W, one can obtain π^* from H_{W} as described in Section 3 above. The Wilcoxon statistics is efficiently computed as $\mathcal{W}_s = \sum_{j=I_{min}}^{I_{max}} H_{\mathsf{T}_i}(j) \pi^*(j)$, where with a slight abuse of notation we have treated π^* as a function of the intensity j rather than of the pixel $\vec{x}$; I_{min} and I_{max} are the minimum and maximum intensities in W.

Histogram updates only involve two full columns of W (similarly, a shift in the y position of W only involves two rows). Therefore, for a square window, the entire decomposition can be computed at a cost of $O(2\sqrt{N} + \ell^*)$ per Ranklet, where ℓ^* is the average range of grey levels appearing in W. This represents a considerable improvement over $O(N \log N)$, which is the complexity of Quicksort.

4.2. Incremental Insertion Sort

The time complexity of Insertion Sort (IS) is $O(N + d)$, where d is the number of inversions [12]. Although the average case is quadratic in N, its very simple structure makes the algorithm effective for small sets of points. IS is also more efficient on partially ordered arrays (in the optimal case of an already ordered array, $d = 0$ and the algorithm is linear). This is clearly the case of a scanning support window, as most of the pixels in W are unchanged between neighbouring positions. We apply an incremental update similar to that used in Section 4.1 above, and refer to this as Incremental Insertion Sort (IIS).

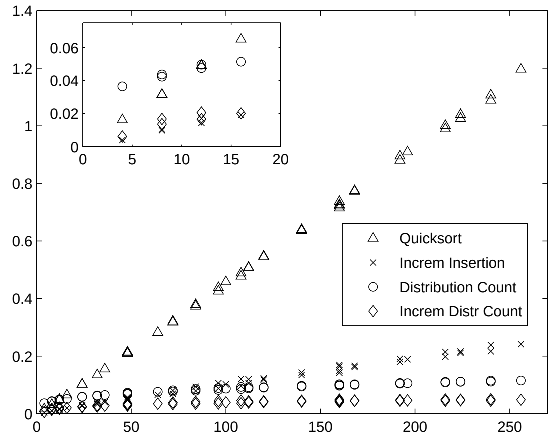

Fig. 3. CPU time (s) as a function of the *area* of W for the raster-scan computation of the 3 ranklet orientations.

4.3. Experimental results

We evaluated the comparative performance of Quicksort, the Incremental Distribution Counting (IDC) algorithm and IIS over the 241 images of the Caltech Pasadena Houses data-set (246×163 pixel series) [16]. Tests were run using C implementations on an Intel Core2 Duo E8200 CPU running at 2.66GHz. Our system has 3GB of RAM installed and runs Linux Fedora 10. Timing with `clock()` provides an accuracy of 0.01s; The standard `qsort()` function was used for Quicksort.

Figure 3 shows the time for the computation of all 3 ranklet orientations over the entire image as a function of $N = \#\mathsf{W}$. Times are averaged across the 241 images in the database; multiple values for the same N and the same algorithm correspond to different aspect ratios of W. As can be seen, IDC represents by far the best choice, outperforming the other algorithms for most window sizes. The speedup with respect to Quicksort is dramatic (a factor of 5 for $N = 32$ pixels only, or 10 for $N = 84$). IIS is slightly advantageous for $N \leq 16$ (Figure 3, inset).

5. SPARSE COMPUTATION

We may not be interested in computing each filter over the entire image — notably if an attentional mechanism is used, or in the later stages of a cascading approach. When locations are sparse, it is no longer possible to exploit window overlap to increase efficiency.

As our experiments below show, a straightforward use of the Distribution Counting algorithm generally provides the best solution in this case, with complexity O(N). However, a competitive alternative is presented by a Sparse Distribution Counting (SDC) algorithm reminiscent of the Integral Image used by Viola and Jones for the computation of Haar wavelets [11]. SDC requires a preprocessing step that is linear in the number of pixels *of the entire image*, but then allows computing Ranklets of any size with linear complexity in the y size of W (that is, $O(\sqrt{N})$ for Ranklets of fixed aspect ratio).

5.1. Sparse Distribution Counting

As we have seen, the Distribution Counting algorithm depends on the possibility to compute H_{W} with a limited number of operations. To this end, let the "tally image" $B_i(x,y) = \sum_{p \leq x, q \leq y} \delta(i - I(p,q))$ be the number of pixels of intensity i in the rectangular region delimited by the origin and point (x,y). For any sub-window W with a diagonal specified by (x_0,y_0) and (x_1,y_1) we have

$$H_{\mathsf{W}}(i) = B_i(x_1,y_1) - B_i(x_0,y_1) - B_i(x_1,y_0) + B_i(x_0,y_0). \quad (3)$$

This is essentially equivalent to the Integral Image algorithm of Viola and Jones [11], except that $B_i(x,y)$ counts the pixels instead than cumulating their intensity. This method has the advantage that, once B_i is computed at a linear cost, any window W can be processed in constant time [17]. Unfortunately, this is impractical for large images, as it requires storing $\ell = 256$ tally images. However, since each unit increment of x or y adds only a relatively small number of pixels to the tally, one can expect the locations at which each B_i changes to be sparse, at least in a region around the origin. This can be exploited to reduce storage space. We overlay a square grid of step Δ to the image. For each grid cell (r,s) we compute the following cell-wise tally images: $B_i^{r,s}(x,y) = \sum_{p \leq x, s\Delta \leq q \leq y} \delta(i - I(p,q))$. Instead of caching all values of these function, we use a variant of the Yale sparse matrix representation to efficiently store only the points $(x_{i,k}, y_{i,k})$ such that $B_i^{r,s}(x,y) < B_i^{r,s}(x_{i,k}, y_{i,k}) \quad \forall x < x_{i,k}, \forall y < y_{i,k}$ [18]. Matrix rows are implemented as linked lists: each element is a structure containing the column number $x_{i,k}$ as well as the actual value of $B_i^{r,s}(x_{i,k}, y_{i,k})$. Pointers to the lists are stored in an array of length equal to the number of rows in the cell. Only the rows corresponding to some $y_{i,k}$ are allocated; the other rows y point to the list for the largest $y_{i,k}$ so that $y_{i,k} < y$. Retrieving a value of $B_i^{r,s}(x,y)$ thus amounts to scanning a list for the largest $x_{i,k}$ smaller than x; the cost of this operation is limited by the size of the cell Δ. A separate, finer grid with step $\Gamma < \Delta$ is used to keep track of the minimum and maximum grey values in the image neighbourhood, so that matrix look-up is not performed in un-needed parts of the dynamic range.

For each support window W, the histogram H_{W} can be obtained according to Equation 3, except that each horizontal "band" of W with $\Delta s \leq y < \Delta(s+1)$ needs to be treated separately, and the results summed (the asymmetry in the treatment of x and y is due to the sparse matrix representation used). This introduces a linear dependence on the vertical size of the Ranklet.

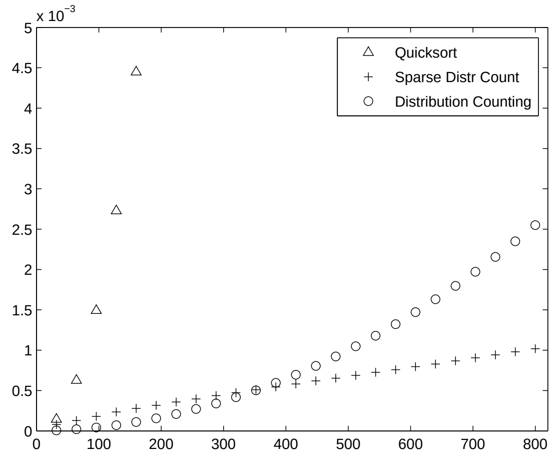

Fig. 4. Average CPU time (s) vs *side* of W for the computation of all 3 ranklet orientations at randomly chosen locations

5.2. Experimental results

We compared Quicksort, the SDC algorithm and a straightforward implementation of Distribution Counting (DC) over the 241 images of data-set [16], 1760×1168 pixel series (hardware and timing method as in Section 4.3).

Fig. 4 shows the time required for the computation of all 3 orientations of a square Ranklet vs the *side* of W. Times are averaged across $10,000$ random locations on each of the 124 images. Both DC and SDC outperform QS. As can be seen SDC is linear in the *side* of W, which makes it faster for larger window sizes. We set $\Delta = 64$ and $\Gamma = 8$ pix; for each image, the average preprocessing time is 2.05s and the algorithm cashes only 6.5% of the $1760 \times 1168 \times 256$ tally image values required by a naive approach. However, since DC is faster than our development implementation of SDC for window sizes up to about 350×350 pix and does not require any preprocessing, it is probably the best choice for most applications. The lower complexity of SDC may make it advantageous as optimised implementations become available.

6. CONCLUSIONS

We introduced four algorithms for the computation of Ranklets that improve on the complexity of the classical Quicksort approach. Distribution Counting is linear in the number of pixels of the support window (plus the number of grey levels), and is applicable to both raster–scan and sparse computations. The Incremental Distribution Counting algorithm applies to full image scans, and is linear in the *side* of the support window of the filter. Incremental Insertion Sort is also a convenient option for scanning with small sized filters. Finally, the Sparse Distribution Counting algorithm achieves a complexity proportional to the *side* of the window for the sparse computation of Ranklets of arbitrary size; however, a linear preprocessing of the image is required.

Speedups of an order of magnitude are easily achieved using these algorithms, which we believe should make Ranklets attractive for real–time applications. To facilitate these, the C implementations used in this paper will be made available through `www.ranklets.net`. Finally, we hope that some of the ideas discussed here may be applicable to the computation of other features based on local pixel rankings.

7. REFERENCES

[1] R. Zabih and J. Woodfill, “Non-parametric local transforms for computing visual correspondence,” in *Proceedings of the 3rd ECCV*, 1994, pp. 151–158.

[2] D. N. Bhat and S. K. Nayar, “Ordinal measures for visual correspondence,” in *CVPR*, 1996, pp. 351–357.

[3] T. Ojala, M. Pietikäinen, and D. Harwood, “A comparative study of texture measures with classification based on featured distribution,” *PR*, vol. 29, no. 1, pp. 51–59, 1996.

[4] E. Franceschi, F. Odone, F. Smeraldi, and A. Verri, “Feature selection with nonparametric statistics,” in *ICIP*, September 2005, vol. I, pp. 325–328.

[5] M. Heikkilä, M. Pietikäinen, and C. Schmid, “Description of interest regions with local binary patterns,” *Pattern recognition*, vol. 42, pp. 425–436, 2009.

[6] F. Smeraldi, “Ranklets: orientation selective non-parametric features applied to face detection,” in *Proc. of the 16th ICPR*, August 2002, vol. 3, pp. 379–382.

[7] A. Del Bue, F. Smeraldi, and L. Agapito, “Non-rigid structure from motion using ranklet-based tracking and non-linear optimization,” *Image and Vision Computing*, vol. 25, no. 3, pp. 297–310, March 2007.

[8] M. Masotti and R. Campanini, “Texture classification using invariant ranklet features,” *PRL*, vol. 29, pp. 1980–1986, 2008.

[9] M. Masotti, “A ranklet-based image representation for mass classification in digital mammograms,” *Medical Physics*, vol. 33, no. 10, pp. 3951–3961, 2006.

[10] A. Mohammed, R. A. El-Khoribi, and L. Fekry, “Discrete Hidden Markov Tree modelling of ranklet transform for mass classification in mammograms,” *GVOP Special Issue on Mammograms*, vol. 7, pp. 61–68, 2007.

[11] P. Viola and M. Jones, “Robust real-time object detection,” in *II Int. W. on Stat. & Comp. Th. of Vision*, 2001.

[12] D. E. Knuth, *The art of computer programming*, vol. III, Addison–Wesley, 1973.

[13] E. L. Lehmann, *Nonparametrics: Statistical methods based on ranks*, Holden-Day, 1975.

[14] G. Garibotto and L. Lambarelli, “Fast on-line implementation of two–dimensional median filtering,” *Electronics letters*, vol. 15, no. 1, pp. 24–25, January 1979.

[15] T. S. Huang, G. J. Yang, and G. Y. Tang, “A fast two–dimensional median filtering algorithm,” *IEEE Trans. on ASSP*, vol. 27, no. 1, pp. 13–18, February 1979.

[16] “Pasadena houses 2000,” accessed January 2009, http://www.vision.caltech.edu/html-files/archive.html/.

[17] F. Porikli, “Integral histogram: a fast way to extract histograms in cartesian spaces,” in *Proc. of CVPR*, 2005, pp. 829–836.

[18] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, “Yale sparse matrix package i: the symmetric codes,” *IJNME*, vol. 18, pp. 1145–1151, 1982.