Exact Acceleration of Linear Object Detectors

Charles Dubout and François Fleuret

Idiap Research Institute, Centre du Parc,
Rue Marconi 19, CH - 1920 Martigny, Switzerland
{charles.dubout,francois.fleuret}@idiap.ch

Abstract. We describe a general and exact method to considerably speed up linear object detection systems operating in a sliding, multi-scale window fashion, such as the individual part detectors of part-based models. The main bottleneck of many of those systems is the computational cost of the convolutions between the multiple rescalings of the image to process, and the linear filters. We make use of properties of the Fourier transform and of clever implementation strategies to obtain a speedup factor proportional to the filters' sizes. The gain in performance is demonstrated on the well known Pascal VOC benchmark, where we accelerate the speed of said convolutions by an order of magnitude.

Keywords: linear object detection, part-based models.

1 Introduction

A common technique for object detection is to apply a binary classifier at every possible position and scale of an image in a sliding-window fashion. However, searching the entire search space, even with a simple detector can be slow, especially if a large number of image features are used.

To that end, linear classifiers have gained a huge popularity in the last few years. Their simplicity allows for very large scale training and relatively fast testing, as they can be implemented in terms of convolutions. They can also reach state-of-the-art performance provided one use discriminant enough features. Indeed, such systems have constantly ranked atop of the popular Pascal VOC detection challenge [1,2]. Part-based deformable models [3,4] are the latest incarnations of such systems, and current winners of the challenge.

The algorithm we propose leverages the classical use of the Fourier transform to accelerate the multiple evaluations of a linear predictor in a multi-scale sliding-window detection scheme. Despite relying on a classic result of signal processing, the practical implementation of this strategy is not straightforward and requires a careful organization of the computation. It can be summarized in three main ideas: (a) we exploit the linearity of the Fourier transform to avoid having one such transform per image feature (see Sect. 2.2), (b) we control the memory usage required to store the transforms of the filters by building patchworks combining the multiple scales of an image (see Sect. 3.1), and finally (c) we optimize the use of the processor cache by computing the Fourier domain point-wise multiplications in small fragments (see Sect. 3.2).

A. Fitzgibbon et al. (Eds.): ECCV 2012, Part III, LNCS 7574, pp. 301–311, 2012.
© Springer-Verlag Berlin Heidelberg 2012

Our implementation is a drop-in replacement for the publicly available system from [5], and provides close to one order of magnitude acceleration of the convolutions (see Table 3). It is available under the GPL open source license at `http://www.idiap.ch/scientific-research/resources`.

1.1 Related Work

Popular methods to search a large space of candidate object locations include cascades of simple classifiers [6], salient regions [7], Hough transform based detection [8], branch-and-bound [9]. Regarding part-based model, only the first method, building a cascade of classifiers, was investigated [10]. Cascades in general and [10] in particular are approximate methods, with no guaranteed speedup in the worst case. Their thresholds are also notoriously hard to tune in order to obtain good performance without sacrificing too much accuracy, often requiring a dedicated validation set [11,6]. The approach we pursue here is akin to [12], taking advantage of properties of the Fourier transform to speed up linear object detectors using multiple features.

Besides accelerating the evaluation of the detector at each possible location, other works have already dealt with the problem of the efficient computation of the feature pyramid and, in the case of part-based models, of the optimal assignment of the parts' locations. The fast construction of the complete image pyramid and associated features computation at each scale has been addressed by [13]. Their idea is to compute such features only once per octave and interpolate the scales in-between, making the whole process typically an order of magnitude faster with only a minor loss in detection accuracy. [14] provides linear time algorithms for solving maximization problems involving an arbitrary sampled function and a spatial quadratic cost. By using deformation costs of this form, the optimal assignment of the parts' locations can be efficiently computed.

Table 1. Notations

C_{std}	Computational cost in flops of a *standard* convolution
F	size (number of coefficients) of a fragment (see Sect. 3.2)
K	number of features
L	number of linear filters
R	number of patchworks (see Sect. 3.1)
ρ	rescaling factor of the image pyramid
$u(F)^\dagger$	time it takes to point-wise multiply together two planes of F coefficients
$v(F)^\dagger$	time it takes to read (resp. write) F coefficients to (resp. from) the CPU cache
$M \times N$	size on an image
$\mathbf{x}^k \in \mathbb{R}^{M \times N}$	the k^{th} feature plane of a particular image
$P \times Q$	size of a filter
$\mathbf{y}^k \in \mathbb{R}^{P \times Q}$	the k^{th} feature plane of a particular filter
$\mathbf{z} \in \mathbb{R}^{(M+P-1) \times (N+Q-1)}$	the scores of a particular filter evaluated on a particular image

$^\dagger u(F)$ and $v(F)$ are linear for large enough values of F

2 Linear Object Detectors and Fourier Transform

Typical linear object detectors – including the individual part detectors of part-based models – extract low-level features densely from every scale of an image pyramid. Those features are arranged in a coarse grid with several features extracted from each grid cell. For example, the Histogram of Oriented Gradients (HOG) [15] correspond to the bins of an histogram of the gradient orientations of the pixels within the cell. Typically cells of size 8×8 pixels are used [15,4], while the number of features per cell vary from around ten to a hundred (32 in [5], that we use as a baseline). An alternative description of the arrangement of the features is to view them as organized in planes as depicted in Fig. 1. Those planes are analogous to the *RGB* channels of standard color images, but instead of colors they contain distinct features from each cell of the grid. The filters trained by the detector are similar in composition.

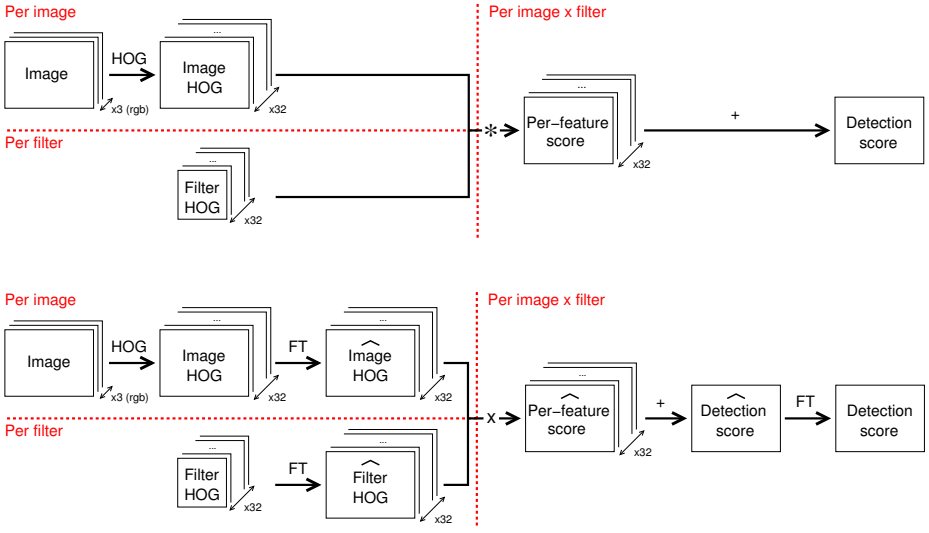

Fig. 1. The top figure shows the standard process, convolving and summing all image and filter planes. The bottom figure depicts how such a process can be sped up by taking advantage of the fact that the inverse Fourier transform that produces the final detection score needs to be done only once per image / filter pair, and not once per feature, since the sum across planes can be done in the Fourier domain.

2.1 Evaluation of a Linear Detector as a Convolution

Let K stands for the number of features, $\mathbf{x}^k \in \mathbb{R}^{M \times N}$ for the k^{th} feature plane of a particular image, and $\mathbf{y}^k \in \mathbb{R}^{P \times Q}$ for the k^{th} feature plane of a particular filter. The scores $\mathbf{z} \in \mathbb{R}^{(M+P-1)\times(N+Q-1)}$ of a filter evaluated on an image are given by the following formula:

$$z_{ij} = \sum_{k=0}^{K-1} \sum_{p=0}^{P-1} \sum_{q=0}^{Q-1} x^k_{i+p,j+q} y^k_{pq} \tag{1}$$

that is the sum across planes of the Frobenius inner products of the image's sub-window of size $P \times Q$ starting at position (i, j) and the filter. Computing the responses of a filter at all possible locations can thus be done by summing the results of the (full) convolutions of the image and the (reversed) filter, i.e.

$$\mathbf{z} = \sum_{k=0}^{K-1} \mathbf{x}^k * \bar{\mathbf{y}}^k \tag{2}$$

where $\bar{\mathbf{y}}$ is the reversed filter ($\bar{y}_{ij} = y_{P-1-i,Q-1-j}$).

The cost of a standard convolution between an image of size $M \times N$ and a filter of size $P \times Q$ is $\mathcal{O}(MNPQ)$. More precisely the number of floating point operations of a standard (full) convolution is

$$C_{\text{std}} = 2MNPQ \tag{3}$$

corresponding to one multiplication and one addition for each image and each filter coefficient. Ultimately one needs to convolve L filters and sum them across K feature planes (see Fig. 1), bringing the total number of operations per image to

$$C_{\text{std/image}} = KLC_{\text{std}}. \tag{4}$$

2.2 Leveraging the Fourier Transform

It is well known that convolving in the original signal space is equivalent to point-wise multiplying in the Fourier domain. Convolutions done by first computing the Fourier transforms of the input signals, multiplying them in Fourier domain, before taking the inverse transform of the result can also be more efficient if the filter size is big enough. Indeed, the cost of a convolution done with the help of the Fourier Transform is $\mathcal{O}(MN \log MN)$.

If we define

$$C_{\text{FFT}} \approx 2.5MN \log_2 MN \tag{5}$$

$$C_{\text{mul}} = 4MN \tag{6}$$

the costs of one FT (the approximation of C_{FFT} comes from [16]) and of the point-wise multiplications respectively, the total cost is approximately

$$C_{\text{Fourier}} = 3\, C_{\text{FFT}} + C_{\text{mul}} \tag{7}$$

per product for the three (two forward and one inverse) transforms using a Fast Fourier Transform algorithm [16]. Note that the filters' forward FTs can be done off-line, and thus should not be counted in the overall detection time, and that an image's forward FT has to be done only once, independently of the number

of filters. Moreover, in the case of learning methods based on bootstrapping samples, the images' forward FTs can also be done off-line for training.

Taking all this into account, and using the linearity property of the FT, one can drastically reduce the cost per image from KLC_{Fourier}. Since the FT is linear, it does not matter if the sum across planes is done before or after the inverse transforms. If done before, only one inverse transform per filter will be needed even if there are multiple planes. Together with the fact that the forward transforms need to be done only once per filter or per image, the total cost per image is

$$C_{\text{Fourier/image}} = \underbrace{KC_{\text{FFT}}}_{\text{forward FFTs}} + \underbrace{LC_{\text{FFT}}}_{\text{inverse FFTs}} + \underbrace{KLC_{\text{mul}}}_{\text{multiplications}} \tag{8}$$

enabling large computational gains if $K + L \ll KL$.

Plugging in typical numbers ($M, N = 64$, $P, Q = 6$, $K = 32$, $L = 54$ as in [5]), doing the convolutions with Fourier results in a theoretical speedup factor of 13. The cost is independent of the filters' size $P \times Q$, resulting in even larger gains for bigger filters. The FT is also very numerically accurate, as demonstrated by our experiments. There is no precision loss for small filter sizes, and even an increase in precision for larger ones. Finally, one can also reduce by half the cost of computing the FTs of the filters if they are symmetric [4], or come by symmetric pairs [5].

3 Implementation Strategies

Implementing the convolutions with the help of the Fourier transform is straightforward, but involves two difficulties: memory over-consumption and lack of memory bandwidth. Those two problems can be remedied using methods presented in the following subsections.

3.1 Patchworks of Pyramid Scales

The computational cost analysis of Sect. 2.2 was done under the assumption that the Fourier Transforms of the filters were already precomputed. But the computation of the point-wise multiplications between the FT of an image and that of a filter requires to first pad them to the same size. Images can be of various sizes and aspect-ratio, especially since images are parsed at multiple scales, and precomputing filters at all possible sizes as in Fig. 2(a) is unrealistic in term of memory consumption. Another approach could be to precompute the FTs of the images and the filters padded only to the biggest image size, as shown in Fig. 2(b). This would require as little memory as possible for the filters, but would result in an additional computational burden to compute the FTs of the images, and more importantly to perform the point-wise multiplications.

However, a simpler and more efficient approach exists, combining the advantages of both alternatives. By grouping images together in patchworks of the

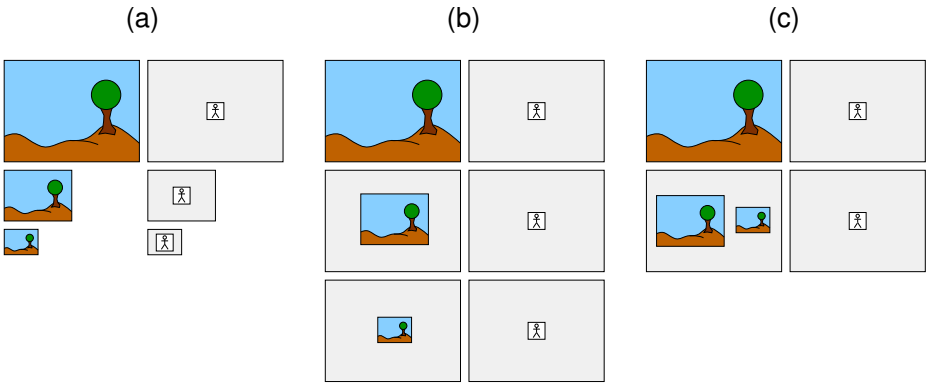

Fig. 2. The computation of the point-wise multiplications between the Fourier Transform of an image and that of a filter requires to pad them to the same size before computing the FTs. Given that images have to be parsed at multiple scales, the naive strategy is either to store for each filter the FTs corresponding to a variety of sizes (a), or to store only one version of the filter's FT and to pad the multiple scales of each image (b). Both of these strategies are unsatisfactory either in term of memory consumption (a) or computational cost (b). We propose instead a patchwork approach (c), which consists of creating patchwork images composed of the multiple scales of the image, and has the advantages of both alternatives. See Sect. 3.1 and Table 2 for details.

size of the largest image, one needs to compute the FTs of the filters only at that size, while the amount of padding needed is much less than required by the second approach. We observed it experimentally to be less than 20%, vs. 87% for the second approach. The performance thus stays competitive with the first approach while retaining the memory footprint of the second (see Table 2 for an asymptotical analysis). The grouping of the images does not need to be optimal, and very fast heuristics yielding good results exist, such as the bottom-left bin-packing heuristic [17].

3.2 Taking Advantage of the Cache

A naive implementation of the main computation, that is the point-wise multiplications between the patchworks' Fourier Transforms and the filters' FTs would simply loop over all patchworks and all filters. This would require to reload both from memory for each pairwise product as they are likely too large to all fit in cache. We observed in practice that such an implementation is indeed memory limited.

However, reorganizing the computation allows to remove this bottleneck. Let R be the total number of patchworks to process, L the number of filters, K the number of features, $M \times N$ the size of the patchworks' FTs, $u(F)$ the time it takes to point-wise multiply together two planes of F coefficients, and $v(F)$ the

Table 2. Asymptotic memory footprint and computational cost for the three approaches described in Sect. 3.1, to process one image of size $M \times N$ with L filters, at scales $1, \rho, \rho^2, \ldots$ The factor $\frac{1}{1-\rho^2} = \sum_{k=0}^{+\infty} \rho^{2k}$ accounts for the multiple scales of the image pyramid, while $\frac{\log MN}{1-\rho^2} \approx -\frac{\log MN}{\log \rho^2}$ for $\rho \approx 1$ is the number of scales to visit. Taking the same typical values as in Sect. 2.2 for $M, N = 64$, and $\rho = 0.9$ gives $\frac{1}{1-\rho^2} \approx 5.3$ and $\frac{\log MN}{1-\rho^2} \approx 44$. Our patchwork method (c) combines the advantages of both methods (a) and (b).

Approach	Memory (image + filters)	Computational cost
(a)	$\frac{1}{1-\rho^2} MN + \frac{1}{1-\rho^2} LMN$	$\frac{1}{1-\rho^2} LMN$
(b)	$\frac{\log MN}{1-\rho^2} MN + LMN$	$\frac{\log MN}{1-\rho^2} LMN$
(c)	$\frac{1}{1-\rho^2} MN + LMN$	$\frac{1}{1-\rho^2} LMN$

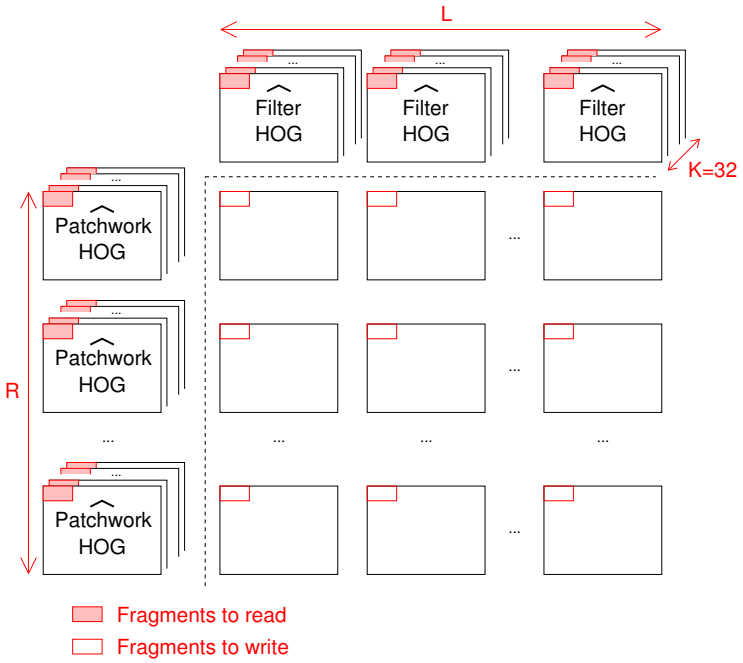

Fig. 3. To compute the point-wise products between each of the Fourier Transform of the R patchworks, and each of the FT of the L filters, the naive procedure loops through every pair. This strategy unfortunately requires multiple CPU cache violations, since the transforms are likely to be too large to all fit in cache, resulting in a slow computation of each one of the LR products. We propose to decompose the transforms into *fragments* (here shown as red rectangles), and to have an outer loop through them. With such a strategy, by loading a total of $L + R$ fragments in the CPU cache, we end up computing LR point-wise products between fragments. See Sect. 3.2 for details.

time it takes to read (resp. write) F coefficients from (resp. into) the memory to (resp. from) the CPU cache.

A naive strategy going through every patchwork / filter pair results in a total processing time of

$$T_{\text{naive}} = \underbrace{KLR2v(MN)}_{\text{reading}} + \underbrace{KLRu(MN)}_{\text{multiplications}} + \underbrace{LRv(MN)}_{\text{writing}}. \tag{9}$$

This is mainly due to the bad use of the cache, which is constantly reloaded with new data from the main memory.

We can improve this strategy by decomposing transforms into *fragments* of size F, and by adding an outer loop through these $\frac{MN}{F}$ fragments (see Fig. 3). The cache usage will be $K(R+1)F$, and the time to process all patchwork / filter pairs will become

$$T_{\text{fast}} = \underbrace{\frac{MN}{F}}_{\text{number of fragments}} \left(\underbrace{K(L+R)v(F)}_{\text{reading}} + \underbrace{KLRu(F)}_{\text{multiplications}} + \underbrace{LRv(F)}_{\text{writing}} \right) \tag{10}$$

$$= \underbrace{K(L+R)v(MN)}_{\text{reading}} + \underbrace{KLRu(MN)}_{\text{multiplications}} + \underbrace{LRv(MN)}_{\text{writing}}. \tag{11}$$

By making F small, we could reduce the cache usage arbitrarily. However, CPUs are able to load from the main memory in bursts, which makes values smaller than that burst size sub-optimal (see Fig. 4). The speed ratio between the naive and the fast methods is

$$\frac{T_{\text{naive}}}{T_{\text{fast}}} = \frac{(2 + \frac{1}{K}) + \frac{u(MN)}{v(MN)}}{(\frac{L+R}{LR} + \frac{1}{K}) + \frac{u(MN)}{v(MN)}} \tag{12}$$

$$\approx 2\frac{v(MN)}{u(MN)} + 1 \tag{13}$$

In practice, the cache can hold at least one patchwork of size MN and the actual speedup we observe is around 5.7. Decomposing the transforms into fragments also scales better across multiple CPU cores, as they can focus on distinct parts of the transforms, instead of all loading the same patchwork or filter.

4 Experiments

To evaluate our approach for linear object detector acceleration we compared it to the publicly available system from [5]. We used the trained models already present in the system, trained on the Pascal VOC 2007 challenge [1] dataset, which achieve state-of-the-art detection results. Note that [5] provides several implementations of the convolutions, ranging from the most basic to the most heavily optimized.

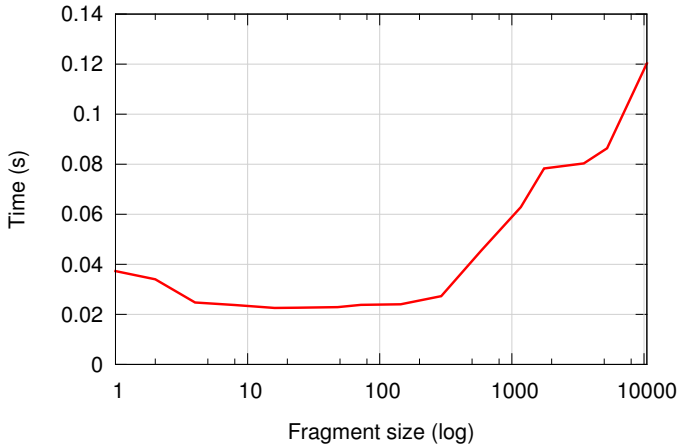

Fig. 4. Average time taken by the point-wise multiplications (in seconds) for different fragment sizes (number of coefficients) for one image of the Pascal VOC 2007 challenge

The evaluation was done over all 20 classes of the challenge by looking at the detection time speedup with respect to the fastest baseline convolution implementation on the same machine. The baseline is written in assembly and makes use of both CPU SIMD instructions and multi-threading. As our method is exact, the average precision should stay the same up to numerical precision issues. The results are given in Table 4 for verification purposes.

We used the *FFTW* (version 3.3) library [16] to compute the FFTs, and the *Eigen* (version 3.0) library [18] for the remaining linear algebra. Both libraries are very fast as they make use of the CPU SIMD instruction sets. Our experiments show that our approach achieves a significant speedup, being more than seven times faster (see Table 3). We compare only the time taken by the convolutions in order to be fair to the baseline, some of its other components being written in Matlab, while our implementation is written fully in C++. The average time taken by the baseline implementation to convolve a feature pyramid (10 scales per octave) with all the filters of a particular class (54 filters, most of them of size 6×6) was 413 ms. The average time taken by our implementation was 56 ms, including the forward FFTs of the images. For comparison, the time taken in our implementation to compute the HOG features (including loading and resizing the image) was on average 64 ms, while the time taken by the distance transforms was 42 ms, the time taken by the remaining components of the system being negligible.

We also tested the numerical precision of both approaches. The maximum absolute difference that we observed between the baseline and a more precise implementation (using double precision) was 9.5×10^{-7}, while for our approach it was 4.8×10^{-7}. The mean absolute difference were respectively 2.4×10^{-8} and 1.8×10^{-8}.

While the speed and numerical accuracy of the baseline degrade proportionally with the filters' sizes, they remain constant with our approach, enabling one to use bigger filters for free. For example if one were to use filters of size 8×8

instead of 6×6 as in most of the current models, the speedup of our method over the baseline would increase by a factor $\frac{8\times8}{6\times6} \approx 1.78$ and similarly for the numerical precision.

Table 3. Pascal VOC 2007 challenge convolution time and speedup

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
V4 (ms)	409	437	403	414	366	439	352	432	417	429	450
Ours (ms)	55	56	53	56	57	56	54	56	56	57	57
Speedup (x)	7.4	7.8	7.6	7.4	6.4	7.9	6.5	7.7	7.5	7.5	8.0

	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
V4 (ms)	445	439	429	379	358	351	425	458	433	413
Ours (ms)	57	59	57	54	54	55	57	58	55	56
Speedup (x)	7.8	7.5	7.6	7.0	6.6	6.4	7.4	7.9	7.9	7.4

Table 4. Pascal VOC 2007 challenge results

	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table
V4 (%)	28.9	59.5	10.0	15.2	25.5	49.6	57.9	19.3	22.4	25.2	23.3
Ours (%)	29.4	58.9	10.0	13.4	25.3	50.6	57.6	18.9	22.6	24.9	24.4

	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mean
V4 (%)	11.1	56.8	48.7	41.9	12.2	17.8	33.6	45.1	41.6	32.3
Ours (%)	11.5	56.7	47.3	42.4	13.0	19.2	34.8	46.3	40.4	32.4

5 Conclusion

The idea motivating our work is that the Fourier transform is linear, enabling one to do the addition of the convolutions across feature planes in Fourier space, and be left in the end with only one inverse Fourier transform to do. To take advantage of this, we proposed two additional implementation strategies, ensuring maximum efficiency without requiring huge memory space and/or bandwidth, and thus making the whole approach practical.

The method increases the speed of many state-of-the-art object detectors severalfold with no loss in accuracy when using small filters, and becomes even faster and more accurate with larger ones. That such an approach is possible is not entirely trivial (the reference implementation of [5] contains five different ways to do the convolutions, all at least an order of magnitude slower); nevertheless, the analysis we developed is readily applicable to many other systems.

Acknowledgments. Charles Dubout was supported by the Swiss National Science Foundation under grant 200021-124822 – VELASH, and François Fleuret was supported in part by the European Community's 7th Framework Programme under grant agreement 247022 – MASH.

References

1. Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2007 (VOC 2007) (2007) Results, http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html
2. Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2011 (VOC 2011) (2011) Results, http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html
3. Felzenszwalb, P., Huttenlocher, D.: Pictorial Structures for Object Recognition. International Journal of Computer Vision 61, 55–79 (2005)
4. Felzenszwalb, P., Girshick, R., McAllester, D., Ramanan, D.: Object Detection with Discriminatively Trained Part-Based Models. IEEE Transactions on Pattern Analysis and Machine Intelligence 32(9) (2010)
5. Felzenszwalb, P., Girshick, R., McAllester, D.: Discriminatively Trained Deformable Part Models, Release 4, http://people.cs.uchicago.edu/~pff/latent-release4/
6. Viola, P., Jones, M.: Robust Real-time Object Detection. International Journal of Computer Vision (2001)
7. Perko, R., Leonardis, A.: Context Driven Focus of Attention for Object Detection. In: Paletta, L., Rome, E. (eds.) WAPCV 2007. LNCS (LNAI), vol. 4840, pp. 216–233. Springer, Heidelberg (2007)
8. Maji, S., Malik, J.: Object detection using a max-margin Hough transform. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1038–1045 (2009)
9. Lampert, C., Blaschko, M., Hofmann, T.: Beyond sliding windows: Object localization by efficient subwindow search. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2008)
10. Felzenszwalb, P., Girshick, R., McAllester, D.: Cascade object detection with deformable part models. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 2241–2248 (2010)
11. Zhang, C., Viola, P.: Multiple-Instance Pruning For Learning Efficient Cascade Detectors. In: Advances in Neural Information Processing Systems (2007)
12. Cecotti, H., Graeser, A.: Convolutional Neural Network with embedded Fourier Transform for EEG classification. In: International Conference on Pattern Recognition, pp. 1–4 (2008)
13. Dollar, P., Belongie, S., Perona, P.: The Fastest Pedestrian Detector in the West. In: British Machine Vision Conference (2010)
14. Felzenszwalb, P., Huttenlocher, D.: Distance transforms of sampled functions. Technical report, Cornell Computing and Information Science (2004)
15. Dalal, N., Triggs, B.: Histograms of Oriented Gradients for Human Detection. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 886–893 (2005)
16. Frigo, M., Johnson, S.: The design and implementation of FFTW3. Proceedings of the IEEE 93(2), 216–231 (2005)
17. Chazelle, B.: The Bottom-Left Bin-Packing Heuristic: An Efficient Implementation. IEEE Transactions on Computers, 697–707 (1983)
18. Guennebaud, G., Jacob, B.: Eigen v3, http://eigen.tuxfamily.org