Accelerated Training of Linear Object Detectors

Charles Dubout and François Fleuret
Idiap Research Institute, Switzerland
École Polytechnique Fédérale de Lausanne, Switzerland
firstname.lastname@idiap.ch

Abstract

We describe a general and exact method to speed up the training of linear object detection systems operating in a sliding, multi-scale window fashion, such as deformable part-based models. Our approach consists of reformulating the computation of the gradient as a convolution, and making use of properties of the Fourier transform to obtain a speedup factor proportional to the linear filters' sizes. This technique does not rely on the sparsity induced by a specific loss, nor on a stochastic sub-sampling of the training examples. Experiments on the PASCAL VOC benchmark show a speedup factor of more than one order of magnitude compared to a standard exact generic method.

1. Introduction

Linear classifiers have become very popular in recent years for the detection of objects in complex scenes, for two reasons. First, they can achieve good prediction performances, provided that they take as input discriminant enough features. Second they have a very low computational cost, both for training and testing, enabling training with very large sets of examples, and testing at near real-time speeds. Indeed, linear systems are the winners of many of the last PASCAL VOC detection challenges [4, 5], the latest being deformable part-based models [7, 9].

Such classifiers are typically used in a sliding-window fashion, predicting a score related to the presence or absence of an object for each possible position and scale in the scene to process. Those scores are computed by taking the inner product between the corresponding image sub-windows and the classifier weights. It is straightforward to see that the entire score matrix can be computed by taking the convolution of the image with the (reversed) linear filter corresponding to the learned classifier weights. Sophisticated methods combine multiple such detectors, either in mixtures, and/or in multi-part models.

Given a loss which has the form of a sum over all locations and scales of a per-sample loss, we can similarly reformulate the value of its gradient as a convolution. As we show in § 3, it is the convolution of the map of point-wise derivatives of the loss – that is, at each point, how the loss changes when the response of the predictor changes there – with the map of feature responses.

By leveraging this form, the computation of the gradient can be sped up by using Fourier transforms, exploiting the redundancy between overlapping samples, as revealed by the analysis of § 4. In practice, as demonstrated in § 5, such organization of the computation removes the increase of the cost with the size of the filters, which are always smaller than the scene to process.

2. Related works

A large amount of literature deals with the problem of efficiently training linear classifiers, such as linear SVMs, by exploiting the particular nature of the associated loss function, novel convex optimization algorithms, or clever implementation strategies [14, 13, 15, 6, 12]. We follow an orthogonal approach extending the one of [3] and look for algorithmic gains in the specific case where the training examples are overlapping sub-windows extracted from training images. As most computer vision learning problems involve very large training sets, a consensus in the vision community is to use online or stochastic gradient descent algorithms [1, 9, 16].

The efficient computation of the classifier scores in such a scenario was already addressed in [3]. By formulating this computation as a sum of convolutions across different feature planes, and using the linearity of the Fourier transform, the authors were able to obtain substantial gain by summing those convolutions in Fourier space, left in the end with only one inverse transform per filter.

The initial cost of transforming the feature planes is quickly amortized as long as the classifier contains more than one filter – for instance in the case of mixture or multi-parts model – while the cost of transforming the filters is amortized as long as there are more than one scene to process. They also describe two important implementation strategies to reduce the memory usage and to limit cache

Table 1. Notations

K number of features
L number of linear filters
R number of images
$M \times N$ size of an image
$\mathbf{x}_r^k \in \mathbb{R}^{M \times N}$ the k^{th} feature plane of image r
$P \times Q$ size of a filter
$\mathbf{w}^k \in \mathbb{R}^{P \times Q}$ the k^{th} feature plane of a particular filter
$\mathbf{y}_r \in \{-1, 1\}^{(M+P-1)\times(N+Q-1)}$ the labels of the sub-images of image r
$\mathbf{f}_r \in \mathbb{R}^{(M+P-1)\times(N+Q-1)}$ the scores of a particular filter evaluated on image r
l, l' the loss per sample and its point-wise derivative
λ regularization constant
J relative importance of the positive examples relative to the negatives
T number of stochastic gradient descent iterations
η_t learning rate at iteration t
β bias of the classifier
$c_r(i, j)$ mixture component associated to the positive at position (i, j) in image r
$\lvert \cdot \rvert^+ = \max(0, \cdot)$ threshold negative values to zero

memory violations, making the whole approach practical. The first strategy is to group the multiple scales of the images together into *patchworks* of constant sizes. It is necessary as the Fourier transforms of the images and the filters need to be of the same dimensions in order to be point-wise multiplied together, while the images might be of different sizes and aspect-ratios. The second strategy is to decompose the point-wise multiplications into *fragments*, so that the fragments at the same particular position among all filters can fit in the CPU cache and do not need to be reloaded multiple times to compute the point-wise products with the corresponding patchwork fragments.

3. Linear object detectors and Fourier transforms

As in [3], we handle the parsing at multiple scales by considering that we process *patchworks*, each composed of multiple scales of one of the original images of the training set. In the rest of the article, an *image* can refer to either one of the original images of the training set, or one of these constructed patchworks.

In the case of a single linear predictor for object detection, the loss usually has a data-driven term of the form

$$L(\mathbf{w}) = \sum_r \sum_{i,j} l(y_r(i,j) f_r(i,j)) \tag{1}$$

where $\mathbf{w}$ are the model weights, $y_r(i,j) \in \{-1, 1\}$ is the label of the sub-image located at (i, j) in image r, l is the loss per sample, and in the case of a single feature per pixel

$$f_r(i,j) = \sum_{a,b} w(a,b)\, x_r(i+a, j+b) \tag{2}$$

is the response of the predictor in image r at location (i, j), where $x_r(i, j)$ is the feature at location (i, j). This expression extends naturally to multiple filters and multiple features by adding sums over the filters and features.

From this, we get

$$\nabla L(a,b) = \frac{\partial L(\mathbf{w})}{\partial w(a,b)} \tag{3}$$

$$= \sum_r \sum_{i,j} \frac{\partial l(y_r(i,j) f_r(i,j))}{\partial w(a,b)} \tag{4}$$

$$= \sum_r \sum_{i,j} y_r(i,j)\, l'(y_r(i,j) f_r(i,j)) \frac{\partial f_r(i,j)}{\partial w(a,b)} \tag{5}$$

$$= \sum_r \sum_{i,j} \overline{(y_r\, l'(y_r \cdot f_r))}(i,j)\; x_r(a-i, b-j) \tag{6}$$

$$= \left(\sum_r \overline{\mathbf{y}_r\, l'(\mathbf{y}_r \cdot \mathbf{f}_r)} * x_r \right)(a,b) \tag{7}$$

hence our main result

$$\nabla L = \sum_r \overline{\mathbf{y}_r\, l'(\mathbf{y}_r \cdot \mathbf{f}_r)} * \mathbf{x}_r \tag{8}$$

where $\overline{\alpha}(i,j) = \alpha(-i,-j)$, the operator $\cdot$ stands for the point-wise multiplication, and we use l' for both the loss derivative and its point-wise evaluation.

This point-wise derivative can be interpreted as the signed sample weights, since it quantifies the importance of the sample at that location of the image in the change of the filter weights.

4. Computational cost

We analyze the asymptotic costs of both the computation of the classifier's scores and the calculation of the gradient of the loss, which together usually constitute most of the computational effort. These two computational steps are depicted on Figure 1.

4.1. Computational cost of the classifier evaluation

In order to compute the gradient of the loss of equation (1), one needs first to compute the response of the predictor at every position, using equation (2). Directly generalizing to the multiple features case, the scores of all the positions present in image t are directly given by the convolution

$$\mathbf{f}_r = \sum_k \overline{\mathbf{w}^k} * \mathbf{x}_r^k. \tag{9}$$

where k go through all the feature indices.

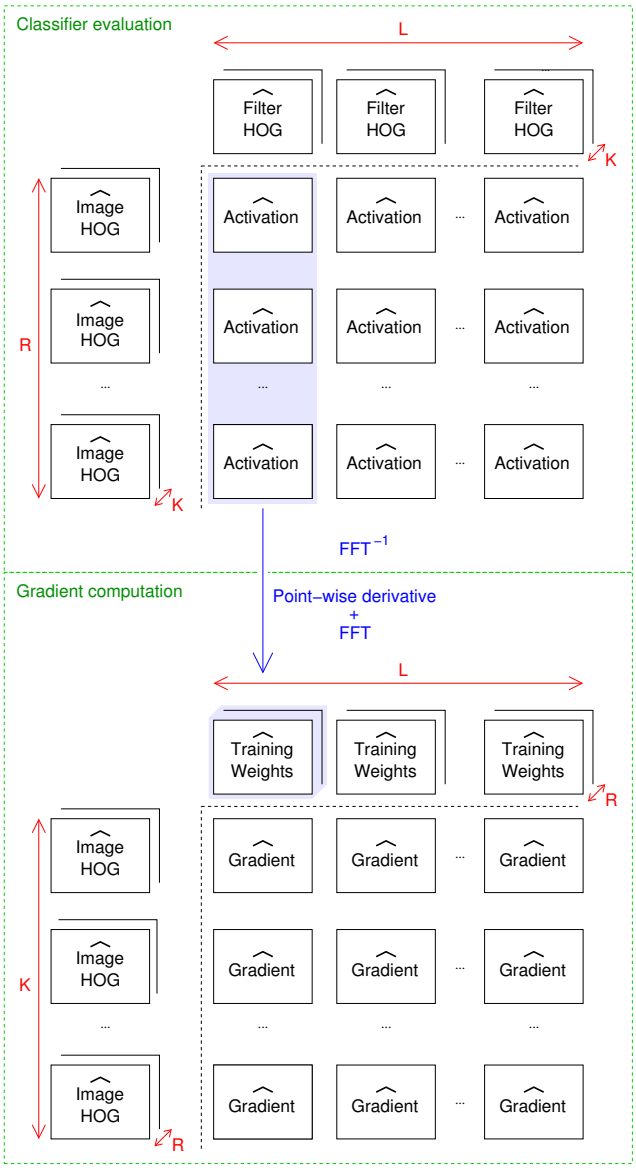

Figure 1. Computation of the gradient of the loss with respect to the model weights, in the case of R images, K features, and L linear filters. The top figure depicts the computation of [3], and consists of the series of point-wise products between the Fourier transforms of the filters, and the Fourier transforms of the images, followed by the inverse Fourier transforms. This produces the maps of point-wise evaluations of the detector in each image. The bottom figure depicts the computation of the gradient. It first computes the point-wise derivatives of the loss to obtain the point-wise training weights, and then the Fourier transforms of the obtained maps. Then, for each feature and each filter, the Fourier transform of the gradient of the loss is obtained by summing the point-wise products of the R training weight maps with the R image maps for that feature.

Let K stand for the number of features, $M \times N$ for the size of the feature planes of the images, and $P \times Q$ for the size of the feature planes of the filters. The computational cost of a standard convolution between an image and L filters across K features is $\mathcal{O}(KLMNPQ)$. Using the method of [3], that cost can be reduced to

$$\underbrace{\mathcal{O}(KMN\log(MN))}_{\text{Forward FFTs of the images}} + \underbrace{\mathcal{O}(KLMN)}_{\text{Multiplications}} + \underbrace{\mathcal{O}(LMN\log(MN))}_{\text{Inverse FFTs of the scores}}, \tag{10}$$

which for both K and L large enough, that is of order $\log(MN)$ or more, is $\mathcal{O}(KLMN)$, a gain by a factor $\mathcal{O}(PQ)$ compared to the standard procedure.

4.2. Computational cost of the gradient computation

The cost of computing the gradient of the loss over one image for the standard method using equation (5) is in $\mathcal{O}(KLMNPQ)$, same as the cost required to compute the scores. Leveraging the Fourier transform to compute it as a convolution, as highlighted in equation (8), it can be reduced by realizing that the transform of the point-wise derivatives of the left-hand side of the convolution operator, i.e. $\mathbf{y}_r\, l'(\mathbf{y}_r \cdot \mathbf{f}_r)$, does not depend on k and thus can be shared across features. The cost to compute the point-wise derivatives themselves is negligible, as it is in $\mathcal{O}(LMN)$. Assuming that the images were already transformed (already required in order to compute the scores by the method of [3]), the cost to compute the gradient leveraging the FFT is

$$\underbrace{\mathcal{O}(LMN\log(MN))}_{\text{Forward FFTs of the derivatives}} + \underbrace{\mathcal{O}(KLMN)}_{\text{Multiplications}}, \tag{11}$$

where the left term is the cost to transform the point-wise derivatives of each filter, and the right term is the cost of the convolutions. Since during training the filters are usually updated by adding them together with the gradients (scaled), one can typically keep them exclusively in Fourier space, removing the cost of transforming the filters back and forth. If it proves impossible, an additional $\mathcal{O}(KLMN\log(MN))$ term is required, reducing the gain compared to the standard process from $\mathcal{O}(PQ)$ to $\mathcal{O}\left(\frac{PQ}{\log(MN)}\right)$, but still keeping the total cost independent of the filters' sizes.

As in [3], one can use the linearity of the Fourier transform to reduce the number of inverse transforms by summing across images in the frequency space. In that case, even if one has to transform back and forth the filters, the cost to compute the gradient over R images is R times that of (11) plus the optional transforms of the filters, in $\mathcal{O}(KLMN\log(MN))$, which for both K and R large enough (of order $\log(MN)$ or more), is $\mathcal{O}(RKLMN)$, again a gain by a factor $\mathcal{O}(PQ)$ compared to the standard process.

5. Experiments

To evaluate our approach to speed up the training of linear object detection systems, we trained a mixture of 6 filters, similar to the roots of the part-based models of [9]. Even though we only trained root filters, nothing prevents us from training full-fledged part-based models, their loss consisting of a sum over part filters, which can be computed using our method, and a deformation penalty term, which can be handled separately at negligible cost.

We used the same modified Histogram of Oriented Gradients (HOG) features [2, 9], the same initialization of the filters' positions, sizes, and left/right pose assignments as in [8], and trained them on the PASCAL VOC 2007 challenge data-set [4]. We used the implementation of [3] to compute the scores, and extended it to compute the gradients. It uses the *FFTW* (version 3.3) library [10] to compute the FFTs, and the *Eigen* (version 3.0) library [11] for the remaining linear algebra.

Algorithm 1 Our Fourier-based stochastic gradient descent algorithm, inspired from the *Pegasos* algorithm [15], taking a whole scene as mini-batch. It minimizes a loss of the form $L(\mathbf{w}) = \frac{\lambda}{2}||\mathbf{w}||^2 + \frac{1}{RMN}\sum_r \sum_{i,j} l(y_r(i,j) f_r(i,j))$.

Input: λ, T

$\hat{\mathbf{w}}_1 \leftarrow \mathbf{0}$ # Initialize the filter to zero

for $t \leftarrow 1, \ldots, T$ **do**

$r \leftarrow rand(1, \ldots, R)$ # Pick a scene at random

$\hat{\mathbf{x}}_r \leftarrow \mathrm{FFT}(\mathbf{x}_r)$ # Transform the scene

$\hat{\mathbf{f}} \leftarrow \hat{\mathbf{w}}_t \cdot \hat{\mathbf{x}}_r$ # Convolve it with the current filter

$\mathbf{f} \leftarrow \mathrm{FFT}^{-1}(\hat{\mathbf{f}})$ # Get back the scores

$\hat{\mathbf{y}} \leftarrow \mathrm{FFT}(\mathbf{y}_r \, l'(\mathbf{y}_r \cdot \mathbf{f}))$ # Transform the derivatives

$\eta_t \leftarrow \frac{1}{\lambda t}$ # Current learning rate

$\hat{\mathbf{w}}_{t+1} \leftarrow (1 - \eta_t \lambda)\hat{\mathbf{w}}_t + \frac{\eta_t}{MN}(\hat{\mathbf{x}}_r \cdot \hat{\mathbf{y}})$ # Update the filter with the gradient

end for

Output: $\mathbf{w} \leftarrow \mathrm{FFT}^{-1}(\hat{\mathbf{w}}_{T+1})$ # Return the filter

5.1. Implementation details

Typical computer vision data-sets contain thousands of images, and thus potentially millions of (mostly negative) training examples, i.e. one per image sub-window at multiple scales. As recommended in [9, 1] in such situations, we chose to train our classifier using a variant of the stochastic gradient descent algorithm. It is derived from the *Pegasos* algorithm [15], using the Fourier transform to compute the convolutions, and without the projection step as it made no difference in our experiments. Since the method of [3] and our own gradient computation technique are efficient only at processing entire scenes, we took all the examples of a scene as a mini-batch at each stochastic gradient descent iteration. Algorithm 1 details the sketch of the algorithm.

We made some modifications to this algorithm in our experiments to adapt it to train a mixture model, and to improve its convergence speed as well as the quality of its final solution.

First as in [8] we modified it to minimize the loss

$$L(\mathbf{w}) = \frac{\lambda}{2}\max_c ||\mathbf{w}_c||^2 + \frac{J}{N}\sum_{r, y_r(i,j)=+1} \left|1 - f^{c_r(i,j)}_{r,\beta}(i,j)\right|^+ + \frac{1}{N}\sum_{r, y_r(i,j)=-1} \left|1 + \max_c f^c_{r,\beta}(i,j)\right|^+, \quad (12)$$

where $\mathbf{w}_c$ is the filter of mixture component c, J is a scale factor re-weighting the importance of the positive examples, n^+, n^- are the total number of positives, reps. negatives, $n = Jn^+ + n^-$ is the total number of examples taking J into account, $\lambda = (Cn)^{-1}$ is the regularization constant, and $|\cdot|^+ = \max(0, \cdot)$. $f^c_{r,\beta}$ is similar to f_r in equation (2), except that it uses the corresponding mixture component's filter $\mathbf{w}_c$, and contains a bias term β, i.e.

$$f^c_{r,\beta}(i,j) = \sum_{a,b} w_c(a,b)\, x_r(i+a, j+b) + \beta. \quad (13)$$

We pick the optimal bias β at the beginning of each stochastic gradient descent iteration in order to minimize the loss, i.e. $\beta_t = \mathrm{argmin}_\beta \, L(\mathbf{w}_t)$, as recommended in [15] when dealing with large mini-batches. Since the bias β is identical among all mixture components, it does not influence their relative scores at test time, but we observed that it is of tremendous importance as removing it significantly reduces both the speed of the convergence of the algorithm and the quality of the final solution. Finally $c_r(i,j)$ are the latent variables (mixture components) of the positive examples, that have to be fixed in order for the loss to be convex [9]. At every iteration we take all the negatives of a whole scene as a mini-batch, and add all the positive examples of the data-set to it. Considering all the positives at every iteration has a very small impact on the training time, the number of positives being usually very small even compared to the number of negatives of an unique scene, but improves drastically the convergence speed of the algorithm. The parameters we used were tuned on the provided validation set and were kept fixed in all experiments. They are $\lambda = 0.01$, $J = 5000$, and $T = 5000$.

5.2. Results

An example of a trained model is represented in figure 2. We also trained models twice as large as the sizes recommended in [9] for the root filters, and we show the same model, this time of twice the size in figure 3. The performances of those mixture models on all 20 classes of the

PASCAL VOC challenge are displayed in Table 2. We do not hope to compete with [8], which trains more complex models including deformable parts, but only want to prove that our results are relevant with respect to the current state-of-the-art.

We implemented two versions of the gradient computation procedure, one using the standard method as in equation (2), and one using the FFT, as detailed in § 3. Both versions make use of the CPU SIMD instruction sets as well as multi-threading. We timed their executions in the same conditions on the same 2.2 GHz Intel Core i7 Quad machine, and provide the results in Table 3. We also tried to use larger mini-batches, processing 10 scenes together, which improves the advantage of our method over the generic one even more. Even though exploiting the sparsity of the loss was by far the fastest method in our experiment, this is due to the use of the hinge loss, and is strongly parameter (λ) and data dependent. The advantage of our method, as concluded from our analysis in § 4, is that it is faster without leveraging sparsity, and always take the same time, independently of the data, the loss, or the filters' sizes. The time taken by the rest of the algorithm, mostly spent convolving the current scene with the filters is also independent of the size of the filters, and below 20ms per iteration. The algorithm typically converges to an acceptable solution in less than one epoch, which corresponds to 2 to 3 minutes.

6. Conclusion

We have presented a novel method to speed up the training of object detectors based on a linear classifier. Existing implementations of such methods relies on sparsity and sub-sampling of the training examples. Our approach by contrast, is based on a formulation of the gradient computation as a convolution, which allows to leverage the Fourier transform, and make the overall computation independent of the filter's size. Experimental validation demonstrates that the gain in speed compared to a generic approach can be more than one order of magnitude.

This new technique provides a generic framework for extension of object detection methods, as it relieves all the constraints inherent to sparse and approximate methods. It can in particular be used with any loss, without the need for it to be sparse inducing, and does not require the tuning of any meta-parameter related to sub-sampling or approximate speed-up strategies.

Acknowledgements

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] L. Bottou and Y. LeCun. Large scale online learning. In *Advances in Neural Information Processing Systems*, 2003. 1, 4

[2] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005. 4

[3] C. Dubout and F. Fleuret. Exact acceleration of linear object detectors. In *European Conference on Computer Vision*, 2012. 1, 2, 3, 4

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html. 1, 4

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2011 (VOC2011) Results. http://www.pascal-network.org/challenges/VOC/voc2011/workshop/index.html. 1

[6] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 6 2008. 1

[7] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. In *International Journal of Computer Vision*, 2005. 1

[8] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Discriminatively trained deformable part models, release 4. http://people.cs.uchicago.edu/~pff/latent-release4/, 2011. 4, 5, 6

[9] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010. 1, 4, 6

[10] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. In *Proceedings of the IEEE*, volume 93 (2), pages 216–231, 2005. 4

[11] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010. 4

[12] C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *International Conference on Machine Learning*, pages 408–415, 2008. 1

[13] T. Joachims. Training linear svms in linear time. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 217–226, 2006. 1

[14] J. C. Platt. Advances in kernel methods. In *Fast training of support vector machines using sequential minimal optimization*, pages 185–208, 1999. 1

[15] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *International Conference on Machine Learning*, pages 807–814, 2007. 1, 4

[16] R. G. J. Wijnhoven and P. H. N. de With. Fast training of object detection using stochastic gradient descent. In *International Conference on Pattern Recognition*, pages 424–427, 2010. 1

Table 2. Average precision scores of the base system of VOC release 4 [8], as well as our trained mixture on the PASCAL VOC 2007 challenge. As we trained only root filters, and not full part-based deformable models, we do not hope to compete with the V4 baseline. These results are provided only to demonstrate the relevance of our approach with respect to the state-of-the-art.

	aero	**bike**	**bird**	**boat**	**bottle**	**bus**	**car**	**cat**	**chair**	**cow**	**table**
V4 normal filters (%)	28.9	59.5	10.0	15.2	25.5	49.6	57.9	19.3	22.4	25.2	23.3
Ours normal filters (%)	18.2	40.8	4.2	11.1	15.0	24.7	34.0	4.7	11.5	27.9	10.7
Ours large filters (%)	18.4	47.3	2.5	13.1	16.9	29.1	41.2	10.3	12.5	26.7	11.2

	dog	**horse**	**mbike**	**person**	**plant**	**sheep**	**sofa**	**train**	**tv**	**mean**
V4 normal filters (%)	11.1	56.8	48.7	41.9	12.2	17.8	33.6	45.1	41.6	32.3
Ours normal filters (%)	5.2	27.7	34.2	18.5	10.8	18.5	12.9	27.1	20.6	18.9
Ours large filters (%)	5.7	37.4	32.6	22.5	11.4	19.3	18.4	24.8	22.9	21.2

Table 3. Average time to compute the gradient of the loss for one stochastic gradient descent iteration. The standard sparse method relies on the sparsity of the samples weights induced by the hinge loss, and computes the gradient by visiting only the samples with non-zero weight. The acceleration it provides is strongly data-dependent.

	1 scene per batch		**10 scenes per batch**	
	Normal filters	**Large filters**	**Normal filters**	**Large filters**
Standard (ms)	41.3	70.9	390	699
Ours (ms)	7.2	7.4	33.1	33.1
Standard sparse (ms)	1.1	1.3	6.6	8.1

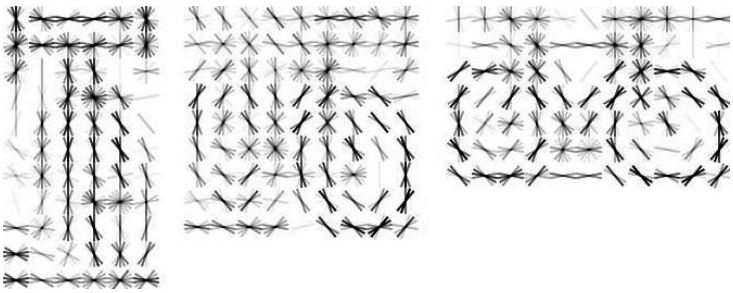

Figure 2. Root filters for a bicycle model of normal size ([9]) learned on the PASCAL VOC 2007 data-set.

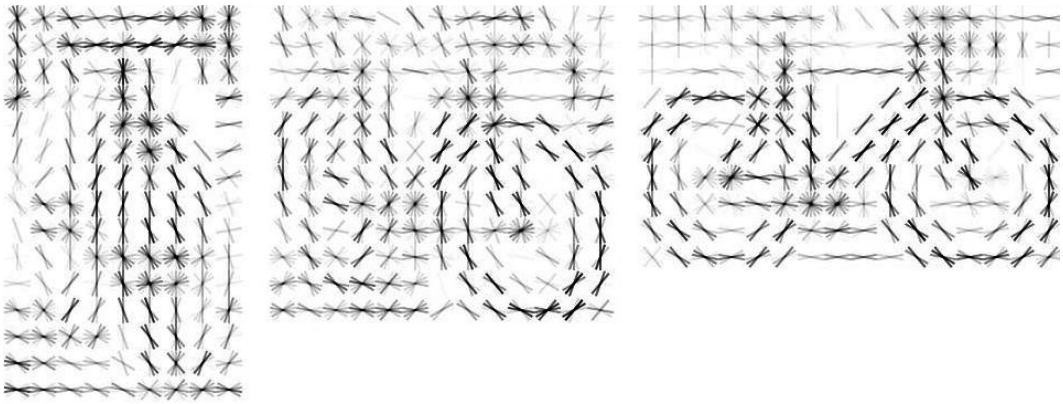

Figure 3. Root filters for a bicycle model of double the normal size ([9]) learned on the PASCAL VOC 2007 data-set.