Fast, Accurate Detection of 100,000 Object Classes on a Single Machine: Technical Supplement

Thomas Dean   Mark A. Ruzon   Mark Segal
Jonathon Shlens   Sudheendra Vijayanarasimhan   Jay Yagnik[†]
Google, Mountain View, CA
{tld,ruzon,segal,shlens,svnaras,jyagnik}@google.com

Abstract

In the paper [1] published in CVPR, we presented a method that can directly use deformable part models (DPMs) trained as in [3]. After training, HOG based part filters are hashed, and, during inference, counts of hashing collisions summed over all hash bands serve as a proxy for part-filter / sliding-window dot products, i.e., filter responses. These counts are an approximation and so we take the original HOG-based filters for the top hash counts and calculate the exact dot products for scoring.

It is possible to train DPM models not on HOG data but on a hashed WTA [4] version of this data. The resulting part filters are sparse, real-valued vectors the size of WTA vectors computed from sliding windows. Given the WTA hash of a window, we exactly recover dot products of the top responses using an extension of locality-sensitive hashing. In this supplement, we sketch a method for training such WTA-based models.

1. Introduction

By applying our hashing method to a familiar object detector we demonstrated its utility and provided some indication of his general applicability. The WTA hash functions serve as the basis for locality-sensitive hashing and provide additional benefit due to their implementing a rank-correlation similarity measure. The number of hash-band matches is only a rough proxy for a dot product however and so to compensate we take the part filters with the most matches and compute the dot products exactly to come up with the final scores for the associated object models.

A more efficient approach is to operate entirely within the WTA space to reconstruct the dot product of a hashed HOG pyramid subwindow and a vector of filter weights by table lookup. In this supplement, we show how such an approach can be applied to improve both throughput and detection performance. Once again we use the basic framework in [3] to simplify the presentation but the method has more general application.

1.1. Detecting Objects

Figure 1 illustrates the basic detection and training architecture for the new method. Instead of a filter corresponding to matrix of weights the size of a particular subwindow of the HOG pyramid, our filters are in the same $(N * K)$-dimensional space as the descriptors produced by our WTA hash function. However, they are real-valued, not binary, and, while still sparse, they are not as sparse as the WTA descriptors, which have exactly N non-zero entries. During training we will be taking dot products of these $(N * K)$-weight vectors with $(N * K)$-hash vectors to compute gradients and adjust the filter weights.

Consider the $(N * K)$-dimensional binary vectors produced by the WTA hash function. Assuming W divides N evenly, divide the vector into equal-sized bands of size $W * K$. For $N = 2400$ and $K = 16$, $W = 4$ produces $M = 600$ bands, each taking 16 bits. Create one hash table for each band to accommodate the hashed subvectors of length $W * K$. Let w be the vector of weight coefficients for a part filter and v be the binary vector corresponding to the WTA hash of a subwindow of the HOG pyramid.

As in the paper, let B_m denote the m^{th} hash table, w_m the m^{th} $(W * K)$ span of weight coefficients in w, and v_m the m^{th} $(W * K)$ binary subvector of v. Given a particular v, we wish to (implicitly) compute $\sum_k^{N*K} w(k) * v(k)$ for all part filters and select the filters above a threshold. We do so using the decomposition of the dot product into M independent partial sums: $\sum_m^M \sum_k^{W*K} w_m(k) * v_m(k)$. Instead of storing just the filter indices in the hash tables and keeping track of the number hash-band matches, we store the partial sums along with the filter indices.

Let Q be a queue used to keep track of the filters and cumulative partial sums. Initialize Q to zero and set m to

[†] Corresponding author.

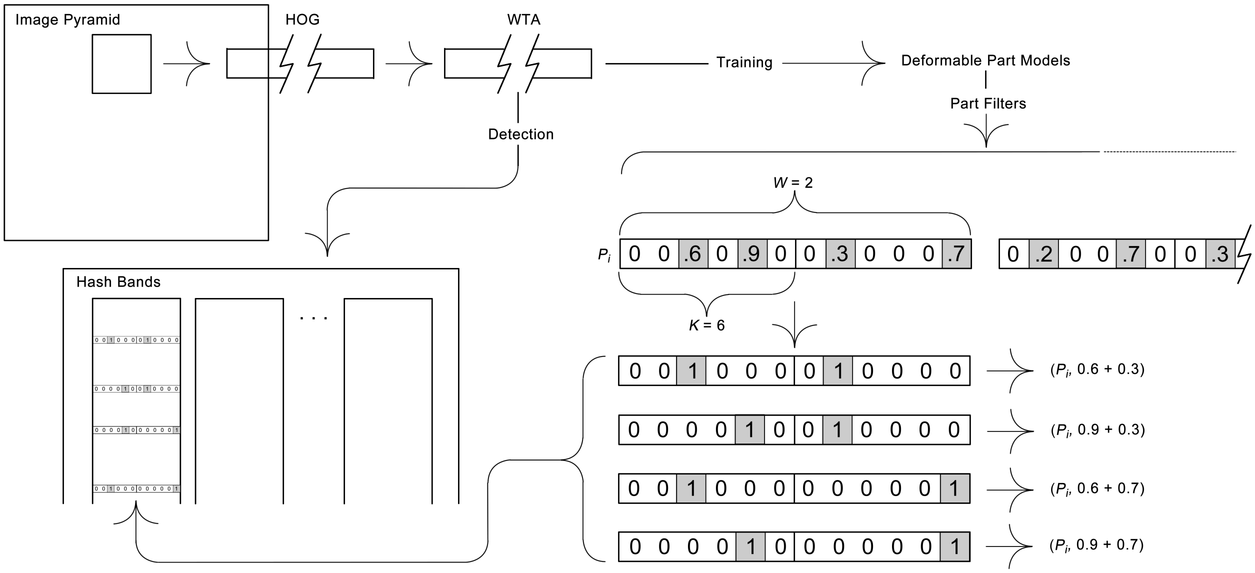

Figure 1. The basic system architecture illustrating the key steps in detecting objects and training models. Unlike the version in the submitted paper in which training examples are comprised of HOG features, training examples in this all-WTA architecture are comprised of the WTA hash codes of HOG features, thus allowing us to take better advantage of the properties of WTA hash functions. Filter weight vectors are real valued and made to be sparse by adding a L_1 term to the loss function. Each such weight vector gives rise to one or more WTA hash codes which are divided into sub codes associated with hash bands and inserted into the corresponding hash tables. The entry in the hash table records the filter index and a scalar value equal to the contribution of that sub code to the dot product of the weight vector and a hash code that matches on that band. During detection, instead of using a count of hash-band matches to generate candidate filters and then computing exact dot products for the candidates with the highest number of matches, we exactly reconstruct the filter responses by summing the previously computed component contributions stored in the hash tables.

1. The entry returned from hashing v_m in B_m is a list of pairs of the form (I_i, S_i) such that I_i is a filter index and $S_i = \sum_k^{W*K} w_m(k) * v_m(k)$ where w is the weight vector for the I_ith filter. We update all the respective cumulative partial sums in Q, increment m and iterate through M. The final cumulative partial sums exactly reconstruct the dot products assuming perfect hashing. We can be clever about keeping Q sorted and maintaining cache coherence, but the more important performance issues involve ensuring that the entries corresponding to most hash keys are empty or reference only a small fraction of all filters. To understand how we accomplish this, we now describe to our approach to training object models.

1.2. Training Models

Training as described in [2] occurs in several phases, each phase consisting of two stages. In the first stage in all but the first, initialization phase, we start by collecting sets of positive and negative examples. In the case of positive examples, we use the current model parameters to find an optimal placement of the parts within the bounding box supplied as annotation with the training example.

In the case of negative examples, we mine the training data to find *hard* negatives corresponding to windows for which the current model parameters produce a false-positive detection. Once the positive and negatives examples for a given phase are assembled, we package the examples in a compact representation suitable for the second stage in which we adjust the model parameters.

In this stage, each training example is encoded as a vector corresponding to a concatenation of the part-filter-sized subwindows of the HOG pyramid where the location of these subwindows were identified in the preceding data mining stage using the current model parameters. The part locations are also encoded in the example vector to be used in adjusting the model deformation parameters.

We're glossing over some details concerning the fact that the models in [2] are mixtures consisting of components that may use different filters of varying size. For simplicity, in the sequel we assume that each model has a single component and all training examples are the same size and that we can encode the model as a concatenation of filter weights in the same order as the examples and hence the same size.

The alignment between parameters and examples implies we can take dot products of one with the other. We now proceed in general agreement with [2] except that we eliminate the root filter and apply the WTA hash function to the HOG-pyramid subwindows and exploit the fact that the WTA hash vectors are sparse to expedite the dot products performed during parameter adjustment. The score for a training example x is defined as follows:

$$f_\beta(x) = \max_{z \in Z(x)} \beta \cdot \Phi(x, z)$$

where β is the vector of model parameters, $Z(x)$ is the set of possible latent values corresponding to part positions, $z \in Z$ is an assignment of positions — one for each part, and $\Phi(x, z)$ is the example vector with the substitution of the position assignments. The objective function as defined in [2] is the sum of an L_2 regularizer and a hinge-loss:

$$L_D(\beta) = \frac{1}{2}\|\beta\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i f_\beta(x_i))$$

where $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ is the training data such that x_i is a positive or negative training example and y_i is 1 if the example is positive and -1 otherwise.

We modify the objective by adding an L_1 term, $\lambda\|\beta\|_1$ to encourage sparsity in the weight vector. By adjusting λ, we control the number of non-zero entries in the filter weights. This has the usual benefits of sparsity plus it serves to reduce the size of the M hash tables and accelerate hashing. If the number of non-zeros within each K-length span is reduced to H, then for each filter f there are no more than $(H + 1)^W$ hash keys for a given hash band whose contribution to dot products involving f is other than zero. This property allows us to reduce the number of (I, S) pairs we need to store in the hash tables. As illustrated on the right in Figure 1, after training the model we determine the (I, S) pairs required to populate each hash band and update the M hash tables accordingly.

References

[1] T. Dean, M. A. Ruzon, M. Segal, J. Shlens, S. Vijayanarasimhan, and J. Yagnik. Fast, accurate detection of 100,000 object classes on a single machine. In *IEEE Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 2013. IEEE Computer Society. 1

[2] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:1627–1645, 2010. 2, 3

[3] P. F. Felzenszwalb, D. A. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008. 1

[4] J. Yagnik, D. Strelow, D. A. Ross, and R.-s. Lin. The power of comparative reasoning. In *IEEE International Conference on Computer Vision*. IEEE, 2011. 1