Max-Margin Additive Classifiers for Detection

Subhransu Maji
U.C. Berkeley, EECS
smaji@eecs.berkeley.edu

Alexander C. Berg
Columbia University, CS
aberg@cs.columbia.edu

Abstract

We present methods for training high quality object detectors very quickly. The core contribution is a pair of fast training algorithms for piece-wise linear classifiers, which can approximate arbitrary additive models. The classifiers are trained in a max-margin framework and significantly outperform linear classifiers on a variety of vision datasets. We report experimental results quantifying training time and accuracy on image classification tasks and pedestrian detection, including detection results better than the best previous on the INRIA dataset with faster training.

1. Introduction

Too much training data can make learning a bottleneck. Quite suddenly this is becoming a real danger for computer vision research. Efficient marketplaces for small increments of human labeling effort such as Mechanical Turk [1, 37] are making possible huge collections of images labeled and verified by real people at a rate of multiple images per penny as exemplified by `image-net.org` [21] a repository of millions of image examples of the wordnet [10] hierarchy. This complements a range of dataset collection efforts from semi-automatic [2, 25, 34, 5, 31] with 10,000-600,000+ images to fully manual but unpaid [32] with 50,000+ labeled objects to more traditional datasets [9, 13]. All of which means that thousands to millions of training examples may become the norm for object recognition.

In a sense this is already the case for training object detectors. It is inexpensive to collect many positive images of, say, pedestrians and images of non-pedestrians. Training for a high quality detector typically proceeds in rounds of training a detector and then evaluating the detector on datasets to identify additional false positives to use for future training rounds. When detectors are run using a sliding window at multiple scales there can be 100,000 or more potential negative training examples per image.

This large amount of data dictates the algorithms that are used. For image classification problems, recent results have made approximate nearest neighbor techniques in high dimensional feature spaces, that require no training but may learn parameters for hashing, efficient enough to be used [38, 19, 35, 22] . Even these are too slow for detection where boosted decision trees and linear classifiers are the default [7, 40]. Contrast this with the most accurate systems for object recognition in settings where efficiency is less critical, usually obtained using kernelized support vector machines (SVMs) that must compare a test image (or region) to each support vector using one or more kernels [3, 39, 12, 24, 6].

Recently we pointed out that many of these SVMs use additive kernels and have classification functions with an additive form that can be efficiently approximated and evaluated nearly as fast as a linear classifier [26]. That work did not address the problem of efficiently training classifiers, instead relying on standard training for kernelized SVM classifiers and then fitting fast additive classifier to match the SVM classifier.

Our main contribution here is to show that classifiers based on additive models can be trained directly in a max margin framework extremely efficiently, and achieve approximately the same accuracy as first training an SVM and then fitting an additive classifier to the resulting decision function as was done in [26] while taking as little as **1%** of the training time. We achieve this remarkable speedup by a special encoding of the learning problem that allows us to take advantage of (our own modification of) recent techniques for training linear classifiers [36, 8].

The result is something of a “free lunch” (or at least very inexpensive) for computer vision researchers because the combination of our fast training techniques with our fast evaluation work make training and testing an additive classifier only a small (constant factor) slower than training a linear classifier. At the same time the additive classifiers produce error rates that are almost always significantly lower than those for a linear classifier on computer vision data.

In addition the optimization method used is derived from PEGASOS [36] and employs stochastic sub-gradient descent which allows us to present an “on-line” version of our approach – streaming data from out of core — as well as a very efficient interactive training scheme for classifiers used in detection.

We report experimental results quantifying training time and accuracy on image classification tasks and pedestrian detection, including detection results better than the best previous on the INRIA [7] dataset.

2. Background

In this paper we train parametric additive classifiers directly, but some of our choices for representation and embedding are motivated by considering SVMs both with linear and non-linear but additive kernels.

The history of the features and kernels used for pedestrian detection and image classification is quite complex – we round up only the most closely related work. Embeddings that allow sub-linear search for similar distributions of features with respect to the Earth Mover's distance were introduced by Thaper & Indyk [20, 19] and later combined with the intersection kernel (aka min-kernel) by Grauman *et al.* [12] to train accurate image classifiers. Lazebnik *et al.* [24] refined the embedding by using multiple levels for spatial coordinates, but not for other dimensions of features. One level of Lazebnik's simple features are very similar to the Histogram of Oriented Gradient (HOG) feature from Dalal and Triggs [7] which was carefully developed to work well with a linear kernel for pedestrian detection and has also been used as the basis of a structured prediction approach to pedestrian detection [11]. The mutli-scale features used by Maji *et al.* [26] fall between those of [24] and [7] – in this work we refer to them as spHOG for spatial pyramid histogram of gradient features.

Earlier Viola & Jones [40] developed their very successful boosting algorithm for training a cascade for face detection. Boosting may seem unrelated to the kernel discussion above, but recent work demonstrates random approximations to boosting using linear SVM training as an intermediate procedure [30, 29]. Furthermore if the weak learners are additive so is a boosted function. Using a random selection of weak learners and our approach may be an effective alternative to [30].

Additive models are well known in the machine learning community [15, 16, 14]. And efficient evaluation for non-parametric kernelized SVMs with additive kernels is addressed exactly in [17, 26] and approximately in [26] which motivates much of our analysis of the encodings we use to transform our problem to a form suitable for efficient optimization. These fast evaluation strategies can accelerate standard SVM training as in [41] which uses our previous work [26] to speed up stochastic gradient descent based SVM training [23].

The data encoding strategy used in Sparse Network of Winnows(SNoW) [42], can be seen as a special case of our own and we include experiments that verify advantages of the SNoW encoding strategy over linear in some cases, while showing analytically and empirically that our approach provides significantly better performance.

Our own optimization procedure generalizes that of the very impressive PEGASOS [36] and in comparison experiments we use LIBLINEAR [8] based on [18]. Both represent amazing progress in training efficiency. While the stochastic nature of PEGASOS may be reminiscent of neural network approaches, differences are the max margin formulation, and one key to its efficiency, the renormalization at each step based on the regularization parameter. This is what changes the iterations required for error bound ϵ from $O(\frac{1}{\epsilon^2})$ for other stochastic gradient descent methods to $O(\frac{1}{\epsilon})$.

3. Overview

We are interested in learning classifiers based on additive models. The decision functions are $sign(f(x))$ where

$$f(x) = \sum_i f_i(x_i) \tag{1}$$

We call f_i the i^{th} "coordinate function", it operates on the i^{th} coordinate of x. Although additive models are often non-parametric, here we are specifically interested in parametric coordinate functions that can be learned efficiently. For labeled training data $\{(x^k, y^k)\}_{k=1\ldots n}$ with the labels $y^k \in \{-1, +1\}$ and the data $x^k \in \mathbb{R}^d$ learning involves finding the f that minimizes a cost function measuring both the training error or loss ℓ and a regularization penalty R

$$f^* = argmin_f R(f) + \frac{1}{n}\sum_k \ell(y^k, f(x^k)) \tag{2}$$

In the rest of the paper we will use the hinge loss $\ell(y^k, f(x^k)) = max(0, 1 - y^k f(x^k))$ motivated by the generalization advantages of large margins, and by the (optional) interpretation of f as the decision function of an SVM with additive kernel.

We explore representations that transform Equation 2 into an efficiently solvable optimization. If w is a vector of parameters specifying the parametric function f^w then we want to encode w as $\hat{w}$ and a data point x as $\hat{x}$ so that $f^w(x) \approx \hat{w}'\hat{x}$ where we emphasize that this may be approximate. After encoding we can write the optimization in Equation 2 as

$$f^{w^*} = argmin_{f^w} R(\hat{w}) + \frac{1}{n}\sum_k max(0, 1 - y^k(\hat{w}'\hat{x}^k)) \tag{3}$$

If each coordinate function $f_i(x_i) = w_i'x_i$ and $R(w) = \lambda w'w$ then $\hat{w} = w$ and $\hat{x} = x$ and this is simply a linear support vector machine (without bias). More generally we can use such a formulation whenever the f_i are a linear combination of a finite number of basis functions. In our experiments f_i are piecewise linear with uniformly spaced breaks. This choice is motivated by simplicity and our analysis [26] showing that decision functions with that form can effectively approximate the decision function of SVMs using the min kernel while being very efficient to evaluate, but we emphasize that other spline functions can be used easily[1].

Depending on the form of the regularization function R we can use different approaches for optimization. For instance when $R(\hat{w}) = \lambda \hat{w}'\hat{w}$ we can use an "off the shelf" SVM package on the encoded data $\{(\hat{x}^k, y^k)\}$. On the other hand, for the "full" version of our approach – motivated by regularization for kernelized SVMs – we present a modified version of the the PEGASOS [36] stochastic sub-gradient descent (with careful normalization) linear SVM solver that can handle $R(\hat{w}) = \lambda w'Hw$ for positive definite H. Only the case of $H = I$, the identity, is addressed by [36].

Section 4 goes into options for encoding in detail, including analysis of representation error and the implications for choices of regularization R. Then Section 5 presents our modified version of PEGASOS. Section 6 presents experimental results.

4. Encoding

We now consider options for the encoding described in Section 3 as well as the related choice of rgularization. Most of the discussion will be motivated by approximating the embedding implied by a specific additive kernel, the histogram intersection kernel, K_{int}, also known as the intersection kernel or min kernel:

$$K_{int}(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^{n} \min\left(x(i), z(i)\right) \tag{4}$$

We note that this is not as restrictive as it might first appear. In fact, for *any* additive function $f(x)$, given $\{f(x^i)\}$ we can find $\{\alpha^i\}$ and b so that $f(x) = \sum_i \alpha^i K_{int}(x, x^i) + b$ for all $x \in \{x^i\}$. This coupled with the fact the $\min$ kernel is conditionally positive definite (CPD) for real valued $\mathbf{x}$ (Section 8) allows one to use $\min$ kernels to learn general additive models on real valued features. CPD kernels can be easily modified to yield Positive Definite (PD) kernels, which correspond to shifting the origin of the RKHS and share the same generalization properties of the corresponding PD kernel[33, 4]. Other additive kernels like the $-\chi^2$ have similar properties. In practice differences between the kernels tend not to matter on large training sets.

First we show explicitly that any SVM $h(x)$ with support vectors $\{v^k\}$ and additive kernel $k(x, y) = \sum_i k_i(x_i, y_i)$ is additive:

$$\begin{aligned} h(x) &= \sum_j \alpha_j k(x, v^j) + b && (5)\\ &= \sum_j \alpha_j \sum_i k_i(x_i, v_i^j) + b && (6)\\ &= \sum_i \sum_j \alpha_j k_i(x_i, v_i^j) + b && (7)\\ &= \sum_i f_i(x_i) + b && (8) \end{aligned}$$

where $f_i(x_i) = \sum_j \alpha_j k_i(x_i, v_i^j)$. For the histogram intersection kernel k_i is simply $\min$. It is sufficient to consider encoding for each dimension separately (as the h is additive) so consider encoding two coordinate values x and z both in $[0, 1]$ for simplicity. In this case the goal of an encoding for the two is that $\min(x, z) = \hat{x}'\hat{z}$

One straight forward encoding is to choose a fixed discretization scale and represent the features in the "unary". Let N denote the number discrete levels and $U(n)$ for $n \in Z$ denote the unary representation of the number n, i.e. $U(3) = 1, 1, 1, 0, 0, 0$, $U(6) = 1, 1, 1, 1, 1, 1$, etc, and $R(.)$ denote the rounding function, then we define our first feature encoding:

$$\phi_1(x) = \sqrt{\frac{1}{N}} U\left(R\left(Nx\right)\right) \tag{9}$$

Intuitively this encoding discretizes the feature into a fixed set of levels and represents each feature using the unary representation. The kernel can then be defined by

$$\begin{aligned} \min(x, y) &\approx\ <\phi_1(x), \phi_1(y)> \\ &=\ <\sqrt{\frac{1}{N}} U\left(R\left(Nx\right)\right), \sqrt{\frac{1}{N}} U\left(R\left(Ny\right)\right)> \\ &=\ \frac{1}{N} <U\left(R\left(Nx\right)\right), U\left(R\left(Ny\right)\right)> \end{aligned}$$

An alternate representation is to use an encoding which instead of rounding to the nearest bin, keeps more detailed information about the values. We define the alternate representation $U'(r)$ for any real number $r >= 0$ as the unary representation, but replacing the first zero in the unary representation of $U(\lfloor r \rfloor)$ by $\alpha(r) = r - \lfloor r \rfloor$. As an example $U'(3.5) = 1, 1, 1, 0.5, 0, 0$

$$\phi_2(x) = \sqrt{\frac{1}{N}} U'\left(Nx\right) \tag{10}$$

The dot product then becomes:

$$\begin{aligned} \min(x, y) &\approx\ <\phi_2(x), \phi_2(y)> \\ &=\ \frac{1}{N} <U'(Nx), U'(Ny)> \end{aligned}$$

We consider the approximation quality for both these linear encodings in the next section.

[1] Different representations will change the effect of regularization.

4.1. Approximation Quality

We will present the worst case and average approximation errors for both these encodings. In both cases the maximum error ($E_\phi^{\max}$) is:

$$E_\phi^{\max}(x,y) = |\min(x,y) - <\phi(x),\phi(y)>| < \frac{1}{N} \quad (11)$$

However we can be more precise about these errors for each of the encodings. The $\min$ operation is symmetric so we need only consider the case when $x \leq y$ and $\min(x,y) = x$.

ϕ_1 : Since $x \leq y$ we have $R(Nx) \leq R(Ny)$. So $< U(R(Nx)), U(R(Ny)) >= R(N(x))$. Therefore the max approximation error is:

$$E_{\phi_1}^{\max}(x,y) = \max|x - R(Nx)/N| = \frac{1}{2N} \quad (12)$$

ϕ_2 : Since $x \leq y$, there are two cases:

1. $\lfloor Nx \rfloor < \lfloor Ny \rfloor$, in which case the embedding is exact.

$$< \phi_2(x), \phi_2(y) > \quad = \quad \min(x,y)$$

2. $\lfloor Nx \rfloor = \lfloor Ny \rfloor = Nm$: Denote $\alpha(Nx)$ by a and $\alpha(Ny)$ by b. Then we have $\min(x,y) = x = m + \frac{a}{N}$, and

$$\begin{aligned} < \phi_2(x), \phi_2(y) > &= < U'(Nx), U'(Ny) > \\ &= \min(x,y) + \frac{1}{N}(ab - a) \end{aligned}$$

Notice that the dot product is always an underestimate of $\min$ value as $ab - a \leq 0$ with $a, b \in [0,1]$. Also the max approximation error is :

$$E_{\phi_2}^{\max}(x,y) = \max_{a \leq b, a,b \in [0,1]} \frac{1}{N}|ab - a| = \frac{1}{4N}$$

If we assume that x, y are distributed uniformly in $[0,1] \times [0,1]$, then we can also compute the expected error, $E_{\phi_1}^{avg} = \frac{1}{4N}$, while $E_{\phi_2}^{avg} = \frac{1}{12N^2}$. This shows that the encoding error decreases with the number of bins and the ϕ_2 encoding is twice as accurate as the ϕ_1 encoding in terms of max error, and significantly better if we care about the average error under a uniform distribution. Figure 4.1 shows the kernel function for min, ϕ_1 and ϕ_2 for $N = 10$. Notice how the ϕ_2 encoding approximates the $\min$ much better.

4.2. Sparse Version of Encoding and Regularization

We saw that the $\min$ kernel can be approximated to within ϵ using $O(1/\epsilon)$ bins for ϕ_1 and $O(1/\sqrt{\epsilon})$ bins for ϕ_2. Hypothetically we could train a linear SVM on those encodings which would be an approximation the the SVM on the original data using an intersection kernel. However these representations are dense, and training a linear SVM on such dense representations becomes infeasible as the number of dimensions becomes large. Instead we propose a sparse representation for each of the embeddings given by:

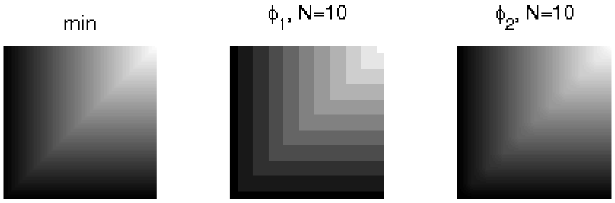

Figure 1. From left to right $\min(x,y)$, $\phi_1(x)\phi_1(y)$ and $\phi_2(x)\phi_2(y)$ with $N = 10$. Note that the ϕ_2 encoding is very close to $\min$.

$$\phi_2^s(x) = \frac{1}{\sqrt{N}}(i : 1-a, i+1 : a) \quad (13)$$

(a vector of all zeros except $\frac{1}{\sqrt{N}}(1-a)$ at position i and $\frac{1}{\sqrt{N}}a$ at position $i+1$) where $a = \alpha(Nx)$ as defined earlier, and $i = \lceil Nx \rceil$ and features are represented by $index : value$ pairs. The transform for ϕ_1^s is the same except both $1-a$ and a are rounded to 0 or 1, resulting in an encoding similar to that of SNoW [42] where they train a linear SVM on these sparse features. The SNoW encoding however does not preserve the underlying $\min$ based similarity measure. We now propose an encoding for w (as in Equation 3) that is compatible with using $\phi_{\{1,2\}}^s$ to encode x.

If $w \in \mathbb{R}^N$ is a weight vector (for instance found by fitting an SVM) on encoded data $\phi_2(x) \in \mathbb{R}^N$ and $w^s \in \mathbb{R}^{N+1}$ a weight vector on the same data encoded as $\phi_2^s(x) \in \mathbb{R}^{N+1}$. We want w such that $w \cdot \phi_2(x) = w^s \cdot \phi_2^s(x)$. The required relationship is

$$w^s(i+1) = w^s(i) + w(i), \text{and } w^s(1) = 0 \quad (14)$$

An important point is how to compute the regularization penalty on w^s. Again if we were hypothetically training a linear SVM on the dense ϕ_2 encodings of the data the regularization penalty would be $w'w$.

The corresponding regularization penalty for w^s is then:

$$||w||^2 = \sum_{i=1}^{N} w(i)^2 = \sum_{i=1}^{N} (w^s(i+1) - w^s(i))^2 \quad (15)$$

This can expressed as $w^{s\prime} H_{tri} s^s$ where H_{tri} is tridiagonal, with the form:

$$\mathbf{H_{tri}} = \left(\begin{array}{ccccc} 1 & -1 & 0 & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & -1 & \\ & & \dots & & \\ & & -1 & 2 & -1 \\ & & & -1 & 1 \end{array} \right)$$

So far our discussion has dealt with only a single coordinate. All encodings are done on a coordinate by coordinate basis and appended. For instance if we use $N = 20$ divisions per coordinate and have 100 dimensional features $x \in \mathbb{R}^{100}$ then $\phi_2^s(x) \in \mathbb{R}^{2000}$ but has at most 2×100 non-zero entries. Similarly H_{tri} is 2000×2000 and all zeros except for 100 blocks as described above along the diagonal. In what follows we only use the sparse encodings, so ϕ_1 will mean ϕ_1^s and ϕ_2 will mean ϕ_2^s.

5. Optimization

Once encoding the data is done and we have chosen a regularization penalty of the form $R(\hat{w}) = \hat{w}' H \hat{w}$ as described above we need to find parameters $\hat{w}^*$ that minimize the cost function c,

$$c(\hat{w}) = \frac{\lambda}{2}\hat{w}' H \hat{w} + \frac{1}{n}\sum_{k=1\ldots n} max(0, 1 - y^k(\hat{w}'\hat{x}^k)) \quad (16)$$

where λ is the regularization vs loss tradeoff. When H is the identity this is simply optimization to fit a linear SVM. In that case a standard linear SVM solver can be used, although ideally one that can efficiently utilize a sparse representation for $\hat{x}$ such as [8, 36].

For our regularization motivated by the min kernel, H is the tri-diagonal matrix described in Section 4.2, H_{tri}. To minimize equation 16 we modify the PEGASOS algorithm originally designed to fit linear SVMs [36]. The original analysis of PEGASOS depends on two aspects of the objective function c – first that c be strongly convex which is true in our case as long as H is positive definite[2], and second that the optimum $\hat{w}^*$ has norm $\hat{w}^{*\prime}\hat{w}^* \leq \frac{1}{\lambda}$, in our case $\hat{w}^{*\prime} H \hat{w}^* \leq \frac{1}{\lambda}$.

Next we show our modification of PEGASOS in its entirety in Algorithm 1. Note that if H is replaced with the identity matrix then this is exactly the PEGASOS algorithm (on possibly encoded data). When we use the tri-diagonal H_{tri} and either encoding, ϕ_1 or ϕ_2 for $\hat{x}$ as described in Sec. 4.2 we call the algorithm "piecewise linear sub-gradient descent"(PWLSGD).

Here $S = \{(x^k, x^y)\}_{k=1\ldots n}$ is all of the training data, A_t is a random subset of l data items chosen for the t^{th} iteration, and A_t^+ is the subset of these which violate the margin constraint using estimate of weight vector $\hat{w}_t$ in step t. From [36] the error is $c(\hat{w}_t) - c(\hat{w}^*) \leq \epsilon$ after $\tilde{O}(\frac{1}{\delta\epsilon\lambda})$ steps with probability $1 - \delta$ when $l = 1$ and after $\tilde{O}(\frac{1}{\epsilon\lambda})$ steps when $l = n$. Intermediate values of l fall between these bounds. In practice the convergence depends on the number of margin violations – basically the difficulty of the classification problem.

Algorithm 1 Our modification of PEGASOS (PWLSGD)

```
Require: $S, T, \lambda > 0$ and $k > 0$
  initialize $\hat{\mathbf{w}}_1$ randomly, such that $\hat{\mathbf{w}}_1' \mathbf{H} \hat{\mathbf{w}}_1 \leq \frac{1}{\sqrt{\lambda}}$
  for $t = 1$ to $T$ do
    Choose $A_t \subset S$, where $|A_t| = l$
    Set $A_t^+ = \{(\hat{\mathbf{x}}, y) \in A_t : y \langle \hat{\mathbf{w}}, \hat{\mathbf{x}} \rangle < 1\}$
    Set $\eta_t = \frac{1}{\lambda t}$
    Set $\hat{\mathbf{w}}_{t+\frac{1}{2}} = \hat{\mathbf{w}}_t - \eta_t \left(\lambda \hat{\mathbf{w}}_t \mathbf{H} + \frac{1}{l}\sum_{(\mathbf{x},y)\in A_t^+} y\hat{\mathbf{x}}\right)$
    Set $\hat{\mathbf{w}}_{t+1} = \min\left(1, \frac{1/\sqrt{\lambda}}{\hat{\mathbf{w}}_{t+\frac{1}{2}}' \mathbf{H} \hat{\mathbf{w}}_{t+\frac{1}{2}}}\right) \hat{\mathbf{w}}_{t+\frac{1}{2}}$
  end for
```

We mention briefly some differences in computational complexity from the original PEGASOS. Our variation requires computing $H'\hat{w}_t$ and $\hat{w}_t' H \hat{w}_t$ for each update. For tridiagonal H this costs roughly 3 times the computation for $H = I$, hence the small multiple in computation time. It is possible that more efficient implementations than our current one, using loop unrolling and other techniques, might be able to hide some of this added complexity. In addition for the particular encodings ϕ_1 and ϕ_2 the encoding pattern in our data $\hat{x}$ is known and fixed (exactly one or exactly two coefficients can be non-zero in each coordinate block) so we can avoid using linked lists for representing the data.

On-line and Interactive Learning

One significant benefit of basing optimization on a stochastic sub-gradient descent method such as PEGASOS is that we can perform learning in stages. For instance in the process of training a detector it is evaluated on many images, false positives (or missed detections) are added to a new training set. After evaluating several hundred or thousand of these a new classifier is trained [7]. We can actually update the classifier by running additional steps of stochastic gradient descent using the new data from each image. As long as the distribution of images is randomized the convergence estimates are very similar. This approach also avoids the memory bottle-neck reported by [7].

The above assumes apriori labeled data, but this does not need to be the case. A human could mark false positives and missed detections in each successive image in an interactive setting. Running a few iterations of training per image can be done faster than humans can label.

6. Experimental Results

We present training time and testing accuracy numbers for each of the proposed methods. We have a choice of

[2] Our tri-diagonal H is not positive definite. Adding a small constant (*e.g.* 0.01) to the first diagonal entry in each coordinate block the diagonal makes it positive definite without effecting the accuracy on experiments. Except for small $l \leq 3$ using the original semidefinite H has no effect on the convergence rate.

encoding for the features: no encoding, ϕ_1, or ϕ_2, and a choice of learning algorithm : linear ($w'w$ regularization) using an off the shelf linear SVM solver (LIBLINEAR), intersection kernel SVM (int. + LIBSVM) or a piecewise linear classifier ($w'H_{tri}w$ regularization) using our PWLSGD algorithm on the encoded features. We present results on Caltech 101, the Daimler Crysler Dataset and the INRIA Pedestrian Dataset and show that both ϕ_1 and ϕ_2 encodings outperform linear classifiers on the non-encoded features by significant amount, and that the encoding and training can be done in a small time compared to training an intersection kernel SVM.

6.1. Caltech 101

Our first set of experiments are on the Caltech-101 dataset [9]. We use this dataset to show that the accuracy using spatial pyramid match kernel introduced in [24] can be matched using our embeddings. For each category we select either 15 or 30 random examples for training and test on a random set of at most 50 training examples as some categories have fewer than 50 remaining for test. We report numbers by averaging the class accuracy for 101 categories using 5-fold cross validation. All the parameters for the models are obtained using by optimizing the performance on a fixed set of 15 training and 15 test examples per category and we use the same parameters for both 15 and 30 training images. We use our own implementation of the 'weak features' introduced in [24] and achieve an accuracy of 50.15% and 56.49% , with 15 and 30 training examples per class and one-vs-all SVM classifiers based on the spatial pyramid match kernel. This kernel reduces to a $\min$ (or intersection) kernel on histograms from each level concatenated together after suitable weighting. Table 1 shows the cumulative training time and accuracies of various methods on this dataset. Linear SVMs are the fastest but also perform the worst. The ϕ_2 encoding with our piecewise linear training algorithm achieves accuracy similar to the intersection kernel SVM at lower training times. Even a linear SVM trained on the ϕ_2 encoded features offers a good accuracy improvement over a linear SVM trained on the raw features at the cost of a small increase in training time. The accuracy using snow encoding (ϕ_1) is quite a bit worse possibly because of quantization.

6.2. Daimler Chrysler Pedestrian Dataset

Our second set of experiments are on the Daimler Chrysler Pedestrian Benchmark dataset, created by Munder and Gavrila [27]. The dataset is split into five disjoint sets, three for training and two for testing. Each training set has 5000 positive and negative examples each, while each test set has 4900 positive and negative examples each. We report the training times and accuracies by training on two out of three training sets at a time and testing on each of the test sets. We use the same spatial pyramid of histograms of oriented gradients features as before. Once again we optimize the parameters on one split and keep the parameters fixed for all the remaining runs. Table 2, shows the performance of various algorithms on this dataset. Once again the ϕ_2 encoded features with the piecewise linear training obtains accuracy similar to the intersection kernel SVM requiring only about 1% of its training time.

6.3. INRIA Pedestrians

We present results on the INRIA pedestrian dataset – the largest dataset we use in this paper – training on up to about $50,000$ features of about 4000 dimensions. We describe how we collect our training/test sets below.

Hard Training Data(HOG) : We use the Dalal and Triggs implementation and collect all the "hard" training examples after the first round of training of a linear SVM. The dataset consists of $47,327$ features of 3780 dimension each, which is the largest dataset we test our algorithms on. We report accuracies by randomly splitting the dataset into 80% training and 20% testing.

Hard Training Data(spHOG) : We use an implementation of the spatial pyramid HOG (spHOG) from [26] where an intersection kernel on these features is used to train a SVM classifier. The primary goal was to see if we could approximate the classifier learned by the expensive SVM learning framework using our fast approximation. There are $38,862$ features of 2268 dimensions. We report accuracies by randomly splitting the dataset into 90% training and 10% testing.

Figure 2, shows the classification accuracy of various methods and features on this dataset. Linear SVM on the HOG features performs quite well, and the intersection kernel SVM offers a slight improvement in accuracy. On the spHOG features the performance of the linear SVM is quite poor and there is a significant improvement in accuracy obtained by using the intersection kernel. In both these datasets the intersection kernel SVM performance is closely matched by the ϕ_2 encoding with PWLSGD. Table 3 shows the training times taken by the various training algorithms. Training an intersection kernel SVM classifier on the entire dataset using LIBSVM can take several hours, while our technique takes less than two minutes.

Our performance for detection in the INRIA data is shown in Figure 2. In order to produce these, the classifier was run on a sliding window and non max suppression to the results was applied according to the same procedures described by Dalal and Triggs.

Acknowledgements: We had helpful discussion with Peter Bartlett and quite useful comments from the reviewers. Special thanks to LIBLINEAR, LIBSVM, and PEGASOS! Funding from ARO MURI W911NF-06-1-0076, ONR MURI N00014-06-1-0734, and ONR MURI N00014-08-1-0638.

		15 examples		30 examples	
Encoding	Training Algorithm	Training Time(s)	Accuracy(%)	Training Time(s)	Accuracy(%)
identity	LIBLINEAR	**18.57 (0.87)**	41.19 (0.94)	**40.49 (0.80)**	46.15 (1.33)
identity	LIBSVM (int kernel)	844.13 (2.10)	**50.15 (0.61)**	2686.87 (4.30)	**56.49 (0.78)**
snow=ϕ_1	LIBLINEAR	45.22 (1.17)	46.02 (0.58)	89.68 (0.93)	51.64 (1.02)
ϕ_2	LIBLINEAR	42.31 (1.43)	48.70 (0.61)	101.97 (1.09)	54.79 (1.24)
ϕ_2	PWLSGD	238.98 (2.49)	**49.89 (0.45)**	291.30 (1.98)	**55.35 (0.72)**

Table 1. Cumulative training time in seconds *(stdev)* and mean class accuracy *(stdev)* for various encodings and algorithms on Caltech 101 dataset using 5 fold cross validation.

Encoding	Training Algorithm	Training Time(s)	Accuracy(%)
identity	LIBLINEAR	**1.89 (00.10)**	72.98 (4.44)
identity	LIBSVM (int. kernel)	363.10 (27.85)	**89.05 (1.42)**
snow=ϕ_1	LIBLINEAR	2.98 (00.33)	85.71 (1.43)
ϕ_2	LIBLINEAR	1.86 (00.04)	88.80 (1.62)
ϕ_2	PWLSGD	3.18 (00.01)	**89.25 (1.58)**

Table 2. Training time in seconds (*stdev*) and accuracy (*stdev*) of various algorithms on the Daimler Chrysler Pedestrian dataset. Each training set has $20,000$ features of 656 dimensions and it takes about $1.84(0.006)$ seconds to encode them.

7. Conclusion

We have shown how to train additive classifiers motivated by [26] very efficiently – within a small multiple of the time required by the very fastest linear SVM training algorithms, shown both theoretically and in experiments. Our resulting additive classifiers consistently perform better than linear classifiers on vision tasks. In particular we can train our piece-wise linear additive classifier for pedestrian detection (based on spHOG features) which produces better results than than Dalal & Triggs' linear detector (based on HOG) in only 76.13 seconds, more than 100 times faster than the training used by [26]. Our fast learning algorithm (PWLSGD) makes piece-wise linear models efficient enough to be part of every vision researchers' standard toolbox of classifiers – we will release code with the publication of this paper.

References

[1] amazon.com. Mechanical turk. `http://mturk.com`, 2008.

[2] T. L. Berg, A. C. Berg, J. Edwards, M. Maire, R. White, Y.-W. Teh, E. Learned-Miller, and D. Forsyth. Names and faces in the news. *CVPR*, 2004.

[3] A. Bosch, A. Zisserman, and X. Munoz. Representing shape with a spatial pyramid kernel. In *ICIVR*, 2007.

[4] S. Boughorbel, J.-P. Tarel, and N. Boujemaa. Conditionally positive definite kernels for svm based image recognition. In *ICME*, 2005.

[5] B. Collins, J. Deng, L. Kai, and L. Fei-Fei. Towards scalable dataset construction: An active learning approach. In *ECCV*, 2008.

[6] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3), 1995.

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. *CVPR*, 2005.

[8] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *JMLR*, 9:1871–1874, 2008.

[9] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *CVPR*, 2004.

[10] C. Fellbaum. *WordNet: An electronic lexical database*. MIT Press, 1998.

[11] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. *CVPR*, 2008.

[12] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. *ICCV*, 2005.

[13] G. Griffin, A. Holub, and P. Perona. The caltech-256. In *Caltech Technical Report*, 2006.

[14] Hastie, Tibshirani, and Friedman. *The Elements of Statistical Learning (2nd Ed)*. Springer-Verlag, 2008.

[15] T. Hastie and R. Tibshirani. Generalized additive models. *Statistical Science*, 1:297–318, 1986.

[16] T. Hastie and R. Tibshirani. *Generalized Additive Models*. Chapman & Hall/CRC, 1990.

[17] M. Herbster. Learning additive models online with fast evaluating kernels. In *COLT '01/EuroCOLT '01*, 2001.

[18] 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 *ICML*, 2008.

[19] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC*, 1998.

[20] P. Indyk and N. Thaper. "fast image retrieval via embeddings". In *"Workshop of Statistical and Computational Theories of Vision"*, "2003".

[21] W. J. Deng, R. Dong, L.-J. Socher, K. L. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[22] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. *CVPR*, 2008.

[23] J. Kivinen, A. AJ Smola, and R. Williamson. Online learning with kernels. *IEEE T. Signal Processing*, 52(8):2165–2167, 2004.

[24] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. *CVPR*, 2, 2006.

[25] L.-J. Li, G. Wang, and L. Fei-Fei. Optimol: automatic object picture collection via incremental model learning. In *CVPR*, 2007.

[26] S. Maji, A. Berg, and J. Malik. Classification using intersection kernel support vector machines is efficient. *CVPR*, 2008.

[27] S. Munder and D. M. Gavrila. An experimental study on pedestrian classification. *PAMI*, 28(11):1863–1868, 2006.

[28] A. Odone F. Barla, A. Verri. Building kernels from binary strings for image matching. *Tr. Image Proc.*, 14(2):169–180, 2005.

[29] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*. 2007.

[30] A. Rahimi and B. Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *NIPS*. 2008.

[31] D. Ramanan, S. Baker, and S. Kakade. Leveraging archival video for building face datasets. In *ICCV*, 2007.

[32] B. C. Russell, A. Torralba, K. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. In *IJCV*, 2007.

[33] B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. The MIT Press, 2001.

Encoding	Training Algorithm	Training Time (HOG)	Training Time (spHOG)
identity	LIBLINEAR	-	20.12s
identity	LIBSVM (lin. lernel)	>180 min	140 min
identity	LIBSVM (int. kernel)	>180 min	148 min
snow=ϕ_1	LIBLINEAR	35.52s	121.81s
ϕ_2	LIBLINEAR	22.45s	26.76s
ϕ_2	PWLSGD	99.85s	76.12s

Table 3. **(HOG)** $47,327$ features of 3780 dimension. Encoding Time $87.22s$. Dalal and Triggs use a modified SVMLIGHT which is faster than LIBSVM, but still takes several minutes to train, slower than our PWLSGD on ϕ_2 encoding which produces both better classification using either HOG or spHOG (below) and better detection (Fig. 2 using spHOG). (LIBLINEAR failed to train on the raw HOG data) **(spHOG)** : Training $38,862$ features of 2268 dimension using PWLSGD on the ϕ_2 encoding takes only about 1% of the time taken to train an intersection kernel SVM using LIBSVM, and performs as well for classification (see below).

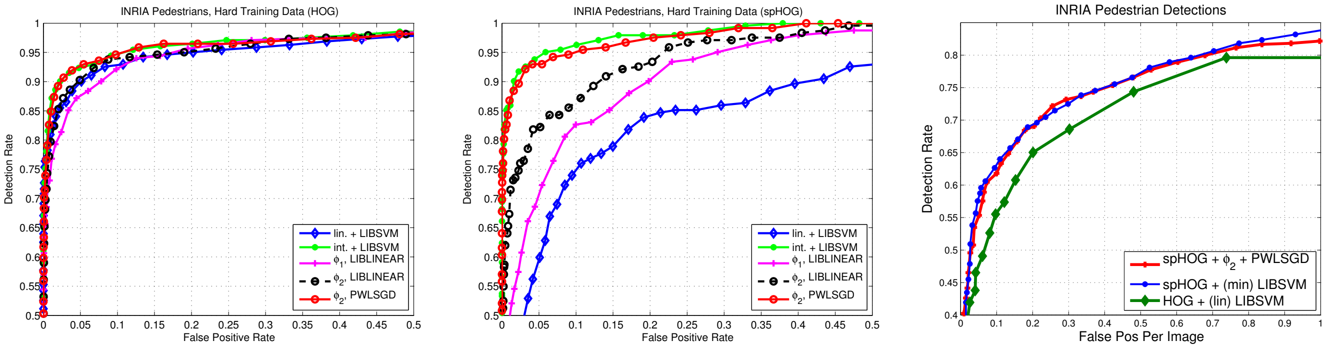

Figure 2. **Left & Middle** show *classification* results on the INRIA Pedestrian Hard Training Data (see Sec. 6.3). Left plot: HOG features and middle plot: spHOG features. The HOG features were designed for a linear classifier, but there is still a small advantage to using the IKSVM (int+ LIBSVM) or PWLSGD over linear. The spHOG features show a larger improvement from using IKSVM or PWLSGD, and have slightly higher overall performance. **Right** shows *Detection* plots on the INRIA benchmark. We compare our detector, spHOG features with IKSVM or PWLSGD, to Dalal and Triggs, HOG + (lin.) LIBSVM. All the detectors are run at a stride of 8×8 pixels, and scaleratio of $2^{1/8}$. The correct detection criteria is ratio of bounding box intersection to union above 50%.

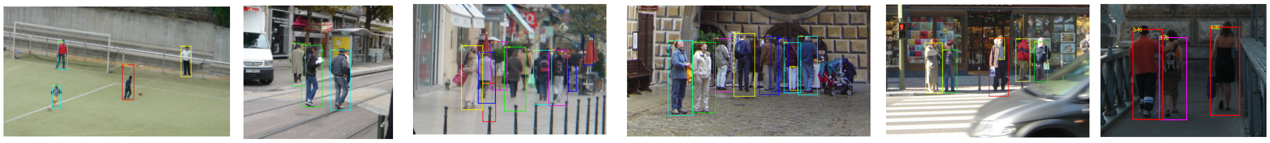

Figure 3. Sample detections on the INRIA Pedestrian dataset using ϕ_2 + PWLSGD algorithm.

[34] F. Schroff, A. Criminisi, and A. Zisserman. Harvesting image databases from the web. In *ICCV*, 2007.

[35] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. *ICCV*, 2003.

[36] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML*, 2007.

[37] A. Sorokin and D. A. Forsyth. Utility data annotation with amazon mechanical turk. In *1st Internet Vision Workshop*, 2008.

[38] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *CVPR*, 2008.

[39] M. Varma and D. Ray. Learning the discriminative power-invariance trade-off. In *ICCV*, 2007.

[40] P. Viola and M. J. Jones. Robust real-time face detection. *IJCV*, 57(2):137–154, 2004.

[41] G. Wang, D. Hoeim, and D. Forsyth. Learning image similarity from flickr groups using stochastic intersection kernel machines. In *ICCV*, 2009.

[42] M.-H. Yang, D. Roth, and N. Ahuja. A tale of two classifiers: Snow vs. svm in visual recognition. In *ECCV*, 2002.

8. K_{int} is Conditionally Positive Definite (CPD)

There are two requirements for $K_{int}(x, y) = \sum_k \min(x_k, y_k)$ where x_k is the k^{th} coordinate of x to be CPD [33]. First K_{int} is clearly symmetric in x and y. Second, let $t_k = \min_i x_k^i$ and $\tilde{x}_k^i = x_k^i - t_k$ so that whenever $\sum_{i=1}^n c^i = 0$ we have:

$$\begin{aligned}
& \sum_{i=1}^n \sum_{j=1}^n c^i c^j \left(\sum_k \min(x_k^i, x_k^j) \right) \\
= & \sum_{i=1}^n \sum_{j=1}^n c^i c^j \left(\sum_k \min(x_k^i - t_k, x_k^j - t_k) + t_k \right) \\
= & \sum_{i=1}^n \sum_{j=1}^n c^i c^j \sum_k \min(x_k^i - t_k, x_k^j - t_k) + \sum_k t_k \sum_{i=1}^n \sum_{j=1}^n c^i c^j \\
= & \sum_{i=1}^n \sum_{j=1}^n c^i c^j \sum_k \min(\tilde{x}_k^i, \tilde{x}_k^j) \geq 0
\end{aligned}$$

As $K_{int}(x, y)$ is positive definite [28] when $x, y \geq 0$, and $\forall i \; \tilde{x}^i \geq 0$.