Learning Globally-Consistent Local Distance Functions for Shape-Based Image Retrieval and Classification

Andrea Frome
EECS, UC Berkeley
andrea.frome@gmail.com

Yoram Singer
Google, Inc.
singer@google.com

Fei Sha
EECS, UC Berkeley
feisha@cs.berkeley.edu

Jitendra Malik
EECS, UC Berkeley
malik@cs.berkeley.edu

Abstract

We address the problem of visual category recognition by learning an image-to-image distance function that attempts to satisfy the following property: the distance between images from the same category should be less than the distance between images from different categories. We use patch-based feature vectors common in object recognition work as a basis for our image-to-image distance functions. Our large-margin formulation for learning the distance functions is similar to formulations used in the machine learning literature on distance metric learning, however we differ in that we learn local *distance functions—a different parameterized function for* every image *of our training set—whereas typically a single global distance function is learned. This was a novel approach first introduced in Frome, Singer, & Malik, NIPS 2006. In that work we learned the local distance functions independently, and the outputs of these functions could not be compared at test time without the use of additional heuristics or training. Here we introduce a different approach that has the advantage that it learns distance functions that are* globally consistent *in that they can be directly compared for purposes of retrieval and classification. The output of the learning algorithm are weights assigned to the image features, which is intuitively appealing in the computer vision setting: some features are more salient than others, and which are more salient depends on the category, or image, being considered. We train and test using the Caltech 101 object recognition benchmark. Using fifteen training images per category, we achieved a mean recognition rate of 63.2% and using twenty images per category, a rate of 66.6%.*

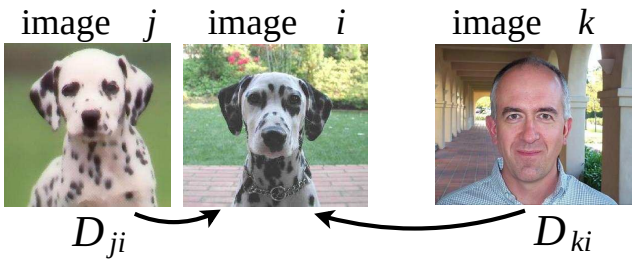

Figure 1. Three images from the Caltech101 data set, two from the `dog` category, one from the `Faces` category. We want to learn distance functions between pairs of images such that the distance from j to i (D_{ji}) is smaller than from k to i (D_{ki}). Triplets like this one form the basis of our learning algorithm.

1. Introduction

Consider the triplet of images, drawn from the Caltech101 dataset [4], shown in Figure 1. We want to classify a query image i, and we have stored exemplar images j and k. Let D_{ji} be the distance from image j to i, and D_{ki} be the distance from image k to i where $D_{ji} < D_{ki}$. Then a nearest neighbor classifier which assigns the category of the query image i based on which of D_{ji}, D_{ki} is smaller, would trivially do the right thing. Note that for this to work, the distance function need not be symmetric, in general $D_{ji} \neq D_{ij}$. To approach this problem, we parameterize the image-to-image distance functions using a weighted linear combination of distances between patch-based shape feature descriptors, such as SIFT [14] or geometric blur [2]. These features characterize image patches by fixed length vectors, which can be compared using L_1 or L_2 metrics. One possible approach to computing an image-to-image distance is to attempt to solve the correspondence problem by taking into account both distances between fea-

Figure 2. Visualizations of the weights learned by our algorithm for one type of shape feature (geometric blur with 42-pixel radius) for three images from our training set. Each circle is centered at the center point of the feature's patch and the color of the circle indicates the relative value of the weight. The colors are on Matlab's jet scale, where dark red is the highest weight (most salient feature) and dark blue is the lowest non-zero weight. Weights that were assigned a zero weight are not shown. Note that the circles are much smaller than the extent of the features and that the colors are scaled separately for each image. For (a), the algorithm learned zero weights for all but 83 of the 400 small geometric blur features computed for the image, and learned that the most important feature is the patch around the eye of the panda. For (b) it learned that the most useful features are on the breast and tail of the rooster, and assigned zero weights to all but 79 of the roughly 400 small geometric blur features. For (c) it learned that the best features aren't on the leopard at all, but are along the right edge and in the upper-right corner of the image. The `Leopards` images were drawn from the Corel image set and they have a thin black border around the image that algorithms can exploit, making it a surprisingly easy category.

ture vectors and the geometric arrangement of their patches (*e.g.* [1]). However, this is expensive, and recent approaches that use sets of features and absolute positions of patches provide good approximations that work well in practice. We work in a setting where we approximate correspondence by only the distance between feature vectors. More precisely, given the mth of M patch features from image j, find the best-matching patch in image i, and let the distance between them be $d_{ji,m}$. The image-to-image distance D_{ji} is defined to be a weighted sum of these distances

$$D_{ji} = \sum_{m=1}^{M} w_{j,m} d_{ji,m} \tag{1}$$

where the different patch features, indexed by m, in image j are assigned possibly different weights $w_{j,m}$. The intuition is that the weights will be high for "relevant" features and low or zero for "irrelevant" features for characterizing the visual category of j. Figure 2 is a visualization of the weights that our algorithm learned for three training images.

The core learning problem is to find the weights $w_{j,m}$ such that the distance relationships among triplets of images holds. We formalize the problem within a large-margin learning framework. At the general algorithmic level, we follow the formulation in [18], and in the context of image recognition, that in our earlier work, [6]. Both this work and [6] differ from [18] in that we both learn a parameterization for *every exemplar* (image) and neither our input distances nor our final distances are metrics. This is a departure from standard approaches in machine learning and computer vision, and the work of [6] was novel in this respect. However, in that work, local distance functions were learned independently for each training image, which means that the outputs from different distance functions are not comparable to one another at test time. To use the distance functions for retrieval and classification, [6] performed a second round of training using logistic regression to put their distance functions on the same scale. In this paper, we introduce a more principled approach to learning local distance functions, where the evaluation criteria for retrieval and classification are consistent with the learning formulation. We jointly learn the weights across our training set, yielding local distance functions that are *globally consistent* in that that are directly comparable at test time. When we are given a test image we can simply rank our training images using the outputs of the distance functions. Using such an ordering, we can make a category choice using a nearest-neighbor classifier.

We apply our technique to image classification and retrieval on the Caltech101 dataset. We achieve a 63.2% mean recognition rate using 15 training images per category and 66.6% using 20 images per category. These represent a significant improvement over the best previously published results.

The rest of the paper is organized as follows: Section 2 discusses related work in computer vision and machine learning; Section 3 formalizes the problem setting while Section 4 introduces the form of the optimization problem and the dual used to solve it; Section 5 gives experimental details, including our choice of patch features, how we choose our triplets for training, and how we choose the parameter to our learning algorithm; and Section 6 discusses our results on the Caltech101 data set.

2. Related Work

There has been much work in recent years using semi-local patch-based features such as SIFT [14] and geometric blur [2] for object (image) classification. When the Caltech101 data set [4] was introduced in 2004, the initial result was approximately only 16% mean recognition across categories. Since then, there has been great improvements in recognition performance on the 2004 benchmark, with most algorithms making use of some variant of geometric blur or SIFT [1, 25, 11, 13, 9, 8, 16, 19]. Of this work, [9], [13], and [8] focused specifically on defining good image-to-image kernel functions over sets of patch-based features for use with support vector machines (SVMs). In the first two of the three, the distance function is designed in advance and does not make use of the training data. In the third, the training data is used to structure a hierarchy over

the feature space, but the class labels are not used. In [25], the geometric blur descriptor was used with DAG SVMs and a nearest-neighbor pruning of the training set at test time to yield strong results. That work also linearly combined different types of feature information, though their combination was parameterized by a single variable tuned by cross-validation.

The objective function and constraints of our large-margin formulation are the same as those in [18], which is part of a larger recent body of work in metric learning, also including [24], [23], and [7]. In this line of work, the inputs $\mathbf{x}$ are points in some metric feature space, and the goal is to learn the matrix A which parameterizes a Mahalanobis distance of the form $(\mathbf{x}-\mathbf{x}')A(\mathbf{x}-\mathbf{x}')$. To our knowledge, this type of approach has not previously been applied to distances between sets of feature vectors as are used in the vision setting. Our approach deviates from these approaches in that they learn a single parameterized distance function for all exemplars, whereas this work and [6] go to the other extreme and learn a distance function for every exemplar.

3. Problem Setting

In the shape-based correspondence, retrieval, and recognition work of the past few years, techniques based on semi-local, or patch-based, features such as SIFT and geometric blur have shown some of the best performance. A typical algorithm chooses a set of patches in an image, and for each patch computes a fixed-length feature vector. This gives a set of vectors per image, where the size of the set can vary from image to image, which is different from a typical machine learning setting where there is a single fixed-length vector for each exemplar, often embedded in a metric space. Thus, any method which makes use of these types of features in a discriminative setting is dependent upon a good set-to-set (image-to-image) distance (or similarity) function, and this has been an active area of research in the last year [9, 8, 13]. More formally, if we let $\mathcal{F}_i$ be the set of feature vectors for image i, then these are asymmetric distance functions that take two sets $\mathcal{F}_i$ and $\mathcal{F}_j$ and give a value in $\mathbb{R}_+$. In an SVM setting, these functions give a positive semi-definite mapping, and the kernel matrix is composed of the image-to-image distances. In almost all machine vision work that makes use of a discriminative classifier and sets of features, learning is done only after the distances are computed between pairs of images.

In this work, we want to learn the distance function in a data-driven matter. Let $\mathbf{f}_{j,m}$ be the mth feature vector from image j. We assume a basic asymmetric distance from a single feature vector $\mathbf{f}_{j,m}$ from one image to the set of features $\mathcal{F}_i$ from another. (In our case, we use the smallest L_2 distance between $\mathbf{f}_{j,m}$ and any member of the set $\mathcal{F}_i$.) Denote the output of this function as $d_{ji,m}$. Assume in addition that each of the features in j is associated with an importance weight $w_{j,m}$. Denote the image-to-image distance between j and i as D_{ji}, as defined in Eq. 1.

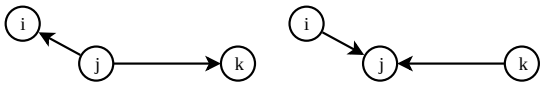

Figure 3. A triplet representing the constraint $D_{jk} > D_{ji}$ from [6] (left) and a triplet representing our constraint $D_{kj} > D_{ij}$ (right).

We borrow the term *local distance functions* from [6], where this setting was introduced. The functions are local in that they are defined with respect to a *single image* (image j in this case), which is a departure from the body of metric learning work and all work we know of in the machine vision community which makes use of a single "global" distance function that does not change depending upon the inputs. Our goal is to learn the weights $w_{j,m}$ for all images in the training set.

Typically, we can say that image i is more similar to image j than to image k, for example by assuming that images in the same class are likely to be more similar than images in different classes. We exploit this structure by casting a learning problem over *triplets* of images. Specifically, let i, j, k denote the indices of a three images such that image i is a reference image which is more similar to image j than it is to image k. Ideally, we would like $D_{ki} > D_{ji}$ where D is defined in Eq. 1. Denote by $\mathbf{d}_{ji}$ the concatenation of the features $d_{ji,m}$ into a vector. Analogously, we denote by $\mathbf{w}_j$ the concatenation to a vector of the weights $w_{j,m}$. Equipped with this notation we can cast the requirement $D_{kj} > D_{jk}$ as the inequality $\mathbf{w}_k \cdot \mathbf{d}_{ki} > \mathbf{w}_j \cdot \mathbf{d}_{ji}$. This constraint is fundamentally different than the constraints used in [6]. Consider the graphs in Figure 3, representing the way the triplet constraints are constructed. Note that the constraints from [6] each only involve *one* distance function; this decomposes their problem into a separate learning problem for *every image*, whereas each of our triplets involves *two* distance functions, and since in our setting all images serve as positive, negative, and reference images in other triplets, all the constraints are tied together. While decoupling the learning problem may reduce training time by making it easily parallelizable, it results in distance functions which are independent and not directly comparable to one another. Indeed, the approach in [6] requires additional heuristics for retrieval and classification. Because our constraints are tied together, we find a set of local weights that are globally consistent, and we are able to directly compare our distances at test time.

4. Large-Margin for Distance Learning

In an idealistic noise-free setting, we might be able to find weight vectors such the constraint $\mathbf{w}_k \cdot \mathbf{d}_{kj} > \mathbf{w}_i \cdot \mathbf{d}_{ij}$ would hold for all triplets i, j, k. Furthermore, by scaling

each weight vector we can ensure that in an idealistic setting the following inequality holds,

$$\mathbf{w}_k \cdot \mathbf{d}_{kj} \geq \mathbf{w}_i \cdot \mathbf{d}_{ij} + 1 \ . \tag{2}$$

When dealing with real image data we clearly cannot simultaneously satisfy the constraints for *all* triplets i, j, k. We therefore need to relax Eq. 2. We propose a relaxation which adapts and generalizes the notion of large-margin classification [21] to our setting. To make the connection to classification work clearer and to simplify our derivation we need to further expand our notation.

We denote by $\mathbf{W}$ the vector which is the concatenation of the image-specific vectors $\mathbf{w}_i$ for every image of our training set. Thus, each image-specific vector corresponds to a subrange of $\mathbf{W}$. We also need to introduce a similar expansion for the distances. Let $\mathbf{X}_{ijk}$ denote a vector of the same length as $\mathbf{W}$ such that all of its entries are 0 except the subranges corresponding to images k and j, which are set to $\mathbf{d}_{ki}$ and $-\mathbf{d}_{ji}$, respectively. It is straightforward to verify that the term $\mathbf{w}_k \cdot \mathbf{d}_{ki} - \mathbf{w}_j \cdot \mathbf{d}_{ji}$ can now be simply written as $\mathbf{W} \cdot \mathbf{X}_{ijk}$ and Eq. 2 distills to $\mathbf{W} \cdot \mathbf{X}_{ijk} \geq 1$.

The relaxation penalizes linearly for deviating from the constraint $\mathbf{W} \cdot \mathbf{X}_{ijk} \geq 1$. We are interested in obtaining a vector $\mathbf{W}$ whose *cumulative* deviation over *all* triplets is small. Formally, let $[z]_+$ denote the function $\max\{0, z\}$. Then, the empirical loss of $\mathbf{W}$ over a collection of triplets is defined as the sum of hinge losses

$$\sum_{i,j,k} [1 - \mathbf{W} \cdot \mathbf{X}_{ijk}]_+ \ .$$

Focusing solely on the empirical loss in order to find $\mathbf{W}$ may result in over-fitting. Here again we follow the work on vector machines and impose an L_2 regularization penalty on $\mathbf{W}$. Thus, the above objective function becomes,

$$\frac{1}{2} \|\mathbf{W}\|^2 + C \sum_{i,j,k} [1 - \mathbf{W} \cdot \mathbf{X}_{ijk}]_+ \ . \tag{3}$$

The scalar C is a trade-off parameter between the squared-norm regularization term and the empirical loss. The larger C is, the greater the emphasis on obtaining a small empirical error. We discuss the method we used for setting C in the next section.

We now briefly describe the numerical algorithm for solving the convex problem given in Eq. 3. We do so by casting the hinge-loss in the objective in Eq. 3 as a linear objective function with multiple linear constraints. We associate a slack variable ξ_{ijk} as in the standard soft-margin SVM form, and get the following constrained optimization problem,

$$\begin{array}{ll} \min\limits_{\mathbf{W},\boldsymbol{\xi}} & \frac{1}{2} \|\mathbf{W}\|^2 + C \sum_{ijk} \xi_{ijk} \\ \text{s.t.} & \forall i, j, k: \ \xi_{ijk} \geq 0 \\ & \forall i, j, k: \ \mathbf{W} \cdot \mathbf{X}_{ijk} \geq 1 - \xi_{ijk} \\ & \forall m: \ W_m \geq 0 \end{array} \tag{4}$$

where C controls the trade-off between the loss and regularization terms and is an input to the optimization (Section 5.4 discusses the choice of C parameter). The setting is the same as that used in [6] and [18], even though the assumptions and the data model underlying it are different from both. The positivity constraint on the elements of $\mathbf{W}$ is due to the fact that our goal is to define a distance function which, by definition, is a positive definite operator. If we assume that the values $d_{ji,m}$ form a PSD matrix, then a sufficient and necessary condition is that the elements of W are non-negative.

We solve this optimization using a dual method. The dual problem of the primal in Eq. 4 is

$$\begin{array}{ll} \max\limits_{\boldsymbol{\alpha},\boldsymbol{\mu}} & -\frac{1}{2} \left\| \sum_{ijk} \alpha_{ijk} \mathbf{X}_{ijk} + \boldsymbol{\mu} \right\|^2 + \sum_{ijk} \alpha_{ijk} \\ \text{s.t.} & \forall i, j, k: \ 0 \leq \alpha_{ijk} \leq C \\ & \forall m: \ \mu_m \geq 0 \end{array} \tag{5}$$

where α_{ijk} is the dual variable corresponding to the constraint on triplet i, j, k, and $\boldsymbol{\mu}$ is a vector of dual variables that maintain the positivity constraint on the elements of $\mathbf{W}$. The conversion to the dual yields the equivalence $\mathbf{W} = \sum_{ijk} \alpha_{ijk} \mathbf{X}_{ijk} + \boldsymbol{\mu}$. We solve the dual using a custom dual solver that iterates over the dual variables, and performs the following two updates:

$$\mathbf{W} \leftarrow \left[\sum_{ijk} \alpha_{ijk} \mathbf{X}_{ijk} \right]_+$$

$$\alpha_{ijk} \leftarrow \left[\frac{1 - \langle \mathbf{W} \cdot \mathbf{X}_{ijk} \rangle}{\|\mathbf{X}_{ijk}\|^2} + \alpha_{ijk}, \right]_{[0,C]}$$

where $\mathbf{W}$ in the first update is clipped to zero if negative, and α_{ijk} in the second update is clipped to the closest value in $[0, C]$. We stop iterating when all KKT conditions are met, within some precision. This technique is a generalized row-action method, closely related to online learning of SVM [3], and is described in more detail in [5].

There are some clear alternatives to the machine learning choices we have made in this work and in [6]. In particular, using an L_1 regularization that promotes sparsity is likely to increase the number of features with zero weight, which would reduce the number of feature comparisons required at test time. However, using L_1 regularization may make the algorithm more sensitive to noise. There are also alternatives to the dual solver we described. In particular, we could use a primal method that directly optimizes the regularized objective function or one that optimizes the hinge loss subject to L_2 or L_1 norm constraints on $\mathbf{W}$. The latter method works by augmenting gradient steps with projections onto an L_p-ball of a predefined radius. We performed some preliminary experiments with primal methods but have not yet achieved results comparable to those from the dual solver.

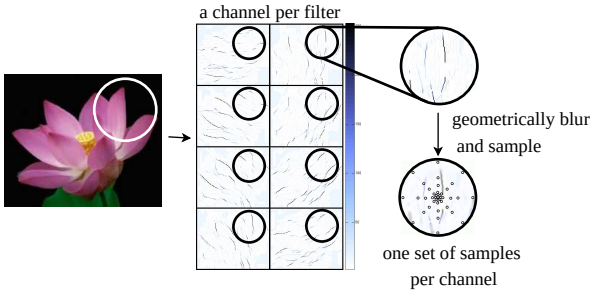

Figure 4. Computation of geometric blur features. A set of sparse signals is derived from the image, using filters, oriented energy, or a more sophisticated boundary detector such as Pb[15]. If we have a set of eight signals, then for a given patch in the original image, we have eight signal patches. Each signal patch is blurred geometrically, meaning that the standard deviation of the Gaussian filter is increased with increasing distance from the center of the patch. The blurred patch is sampled at a set of points, and the final feature vector is the concatenation of the samples for all eight signals. In our experiments, we use a geometric blur feature computed from four orientations of Pb.

5. Details

5.1. Choosing features

As with any approach built upon patch-based features, the choice of feature type is crucial. If the features do not capture the visually salient elements of images for the purposes of categorization, the algorithm as a whole will perform poorly. Some of the best results on Caltech101[25, 6] have been achieved using geometric blur features[2]. At a high level, these are similar to SIFT in that they capture the relationship of oriented edges in a patch of the image, but they differ in the details. A diagram of the extraction process is shown in Figure 4. We use the same set of geometric blur features used in [25] and [6], which are centered at edges in the images and computed from patches of fixed scale. One advantage of our method and [6] over methods such as [25] that use a fixed distance function is we are able to naturally combine different features by simple concatenating their individual distances together into a single $\mathbf{d}_{ji}$ vector. As in [6], we use geometric blur features computed at two different scales (radii of 70 and 42 pixels). In most of our experiments we also use the simple color features used in [6] —HSV histograms of the 42-pixel patches used for the smaller geometric blur features— to demonstrate that we are able to naturally combine features of different types. For both shape features and the color feature, we use the L_2 distance to compute our feature-to-set distances.

We are able to achieve very good performance without using information about the location of patches in the images, whereas many of the best results make use of the absolute position of the patches [13, 25, 16]. We could easily incorporate this information in the style of [16] by only comparing a feature in image j to the set of features in image i that are in roughly the same position (as opposed to comparing to all features in image i), and this is a possible avenue for future work.

5.2. Choosing triplets

We train on a subset of the exhaustive set of possible triplets since training with all triplets is neither practical nor advantageous. To build our set of triplets from a labeled dataset such as Caltech101, we could use every possible combination of reference, positive, and negative images from the training set. For an experiment using 15 images per category, we would have 15 reference images per category (1515) times the number of positive examples for each reference image ($\times 14$), times all negative examples for each of those positive pairs ($\times 1500$) for a total of about 31.8 million triplets. Training with the exhaustive set of triplets has a few drawbacks. First, if the training data is too large to fit into memory, then we need to perform disk seeks within our iterations. Second, the amount of time for each iteration over the data increases linearly with the number of triplets, and the time required to run to completion or reach a good answer may increase super-linearly. Many of the triplet constraints the exhaustive approach creates are very easy to satisfy, so while they contribute to the size of the training data, they do not improve the final weights. Third, many of these triplets involve "bad" positive pairs; even with the best possible weights on their features, the constraints involving these pairs are still badly violated. This is a form of label noise and we believe it *should* be present in a challenging object recognition data set. A large-margin classifier, while relatively less sensitive to outliers than some other algorithms (*e.g.* logistic regression), still attempts to fit the noise.

We address both these issues by pruning the possible set of triplets using the feature-to-set distances. For an image j, for each feature, we order the other images in the training set by their feature-to-set distance and make use of the top N closest for each feature of j. Consider a positive example in this short list: we know that this positive pair is similar according at least one feature, so it is likely we could find a weighting that makes the distance small. A negative example in this short list is also a good candidate because it will probably give a constraint close to the margin, which the algorithm should focus on. Thus, given that image i was in the short list for j, we want to use the distance vector $\mathbf{d}_{ji}$ in some of our triplets. We group these pairs by the reference images (i in this case) and then form triplets from all pairs involving that reference image. We chose a depth of $N = 5$ without testing other parameter choices, and found it to give a good reduction in the number of triplets. For 15 images per category, we reduce the set of triplets by roughly half,

to "only" 15.7 million.

5.3. Early Stopping

Our dual solver terminates when it can make a full pass over all constraints without any updates. A given constraint may not change because either (1) it has satisfied the KKT conditions within some precision [17], or (2) the update to the dual variable falls below a threshold for a "useful"update (we use the threshold from [17]). The solver often stops before full convergence, but for large data set sizes it still takes a long time to run. Using 5 images per category, the optimization ran to completion in about 11 minutes. Using 15 images per category, it took 10 hours, and with 20 images, approximately 16 hours.

An advantage of the dual solver is that, like online learning methods such as [20], it finds a near-optimal solution very quickly. We record the value of the dual objective after every pass over the data, and we use the rate of change of the dual as an indicator of progress; when the rate of change of the dual becomes small (*e.g.* 0.001% of the value of the dual), most of the progress has already been made, and we can use the weights learned up to that point. In practice this works well; with 15 images per category, the recognition performance was the same using weights taken after running to completion (10 hours) and weights sampled after one hour of training. The results reported in this paper for 20 images per category are from our only sample of the weights, taken after about $2\frac{1}{2}$ hours of training.

5.4. Setting the Trade-off Parameter

The trade-off parameter C used in the convex optimization problem defined in Eq. 3 plays a crucial role. Using a large value for C might put too much emphasis on the empirical loss which often results in over-fitting the training set. An excessively small value as the choice of C typically yields an over-regularized setting which leads again to poor performance in practice. A popular and practical approach for choosing C is to run the full learning procedure with multiple suggestions for C on a held-out portion of the training set, also called a validation set. This approach entertains some formal properties [12] and often yields very good results in practice. However, the approach is quite time consuming as it requires running the training algorithm several times for different partitions of the data.

Due to the size of our problem we chose an alternative approach which is based on recent advances in research on online learning algorithms and fits nicely with our dual formulation of the problem. For a choice of C, we make *one pass* over our set of triplets, and for each triplet i, j, k, we (1) evaluate the loss for that example using the formula $[1 - \mathbf{W} \cdot \mathbf{X}_{ijk}]_+$ then (2) make an update to its α_{ijk} dual variable (effectively updating the weight vector $\mathbf{W}$). In this way, every example in the set serves once as a "held-out" example before contributing to the model. By taking the average loss across the training examples after one sweep through the data, we get a number that we can compare to runs with different values of C. The value of C that gives the smallest average loss is the parameter that we use to run the full learning algorithm. The predictions of this online algorithm, known as Passive-Aggressive [3], are guaranteed to be competitive with the predictions of the optimal solution of Eq. 3. Though we did not make use of it, the vector W obtained after a single online pass through the training set can serve as a very good initialization for the batch optimization process. We sampled values of C very coarsely, at half-orders of magnitude. For the data set using 20 images per category, each C test took only a couple minutes.

6. Results

We tested our algorithm using the Caltech101 data set. Introduced in 2004[4], it had more categories than any other object recognition dataset at the time by an order of magnitude, and the state of the art in image classification algorithms has improved greatly since its introduction. The dataset has several drawbacks, however. For one, there is little variation in pose or scale within many classes. Worse, there are artifacts that make some classes very easy to identify, such as rotated pictures (*e.g.*`minaret`) or thin borders (*e.g.*`Leopards`, see Figure 2). While it is true our approach (rightfully) exploits the artifacts in the dataset, it is a general approach that can be applied using any patch features and is not tuned to the Caltech101 data set.

We performed experiments using 5, 10, 15, and 20 images per category, using the remaining images in the dataset for testing, as in [9]. For each test image, we order the training images according to their distance to the test image using their learned distance functions. To make a class choice, we use a modified 3-NN classifier, where if no two images agree on the class within the top 10 matches, then we take the class of the top-ranked image. The choice of three and ten were arbitrary and we did not tune them. The number of test images varies between classes, with some of the easiest classes having the greatest number, so we compute our average recognition as in [9] by first computing the percentage correct for each class, and then averaging those numbers to get mean recognition. This is equivalent to computing the average of the diagonal of the confusion matrix. We use all 101 categories in training and testing, but do not make use of the background class. Most results to date have been reported using all 101 categories except [10] which omits the `Faces_easy` category, a confuser for the `Faces` category.

The graph in Figure 5 shows our results with several of the results published in the last few years. At five and ten images per category, we perform below the best results from [25], and cross somewhere between ten and fifteen images per category. At fifteen images per category, we achieve

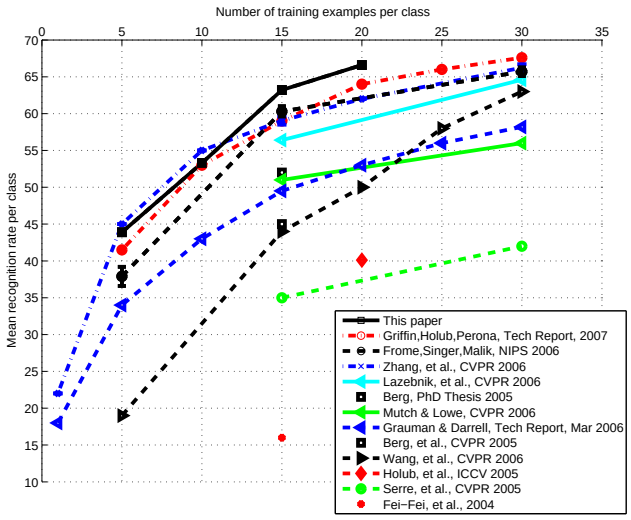

Figure 5. Number of images per category versus mean recognition rate. Our results are the solid black line that cross above the others between 10 and 15 images per category. Also shows results from [10], [25], [13], [16], [9], [1], [22], [11], [19], and [4]. Note that the results just below ours at 20 images per category are computed differently; they do not include the `Faces_easy` category in training or testing, thus eliminating a prominent confuser.

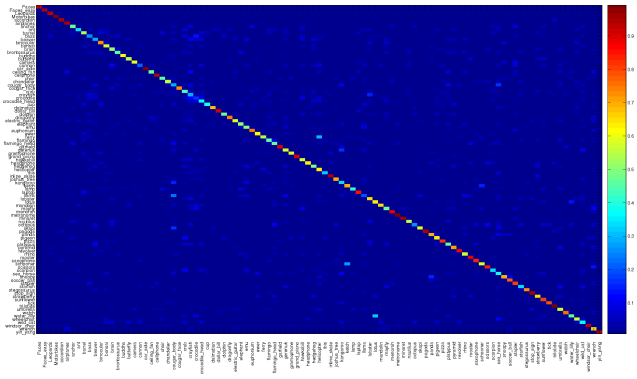

Figure 6. Confusion matrix for 15 images per category, shown using the jet color scale from Matlab. Dark red indicates 100% while dark blue indicates 0%, with a gradient from warm to cool colors in between (see scale, right). A perfect matrix would be dark blue matrix except for a dark red diagonal.

a recognition rate of 63.2%, about 3% better than the best previously published results, and at 20, we achieve 66.6%, about 5% better than the next best result that trains and tests on all 101 categories. It makes sense that our performance would increase dramatically with the number of categories: we use a nearest-neighbor classifier, plus at the core of our method is an assumption that there are pairs of similar images in the training set. This is more likely to be true as the number of training images grows.

All other approaches other than [6] use only shape features, whereas our work and [6] make use of rudimentary color features. Most of our performance is gained from the shape features; at 15 images per category, the color features add only 2% to the mean recognition. At the same time, the fact that these simple color features improve performance at all demonstrates that our method is able to naturally combine features of very different types. In Figure 6, we show our confusion matrix for 15 images per category.

Acknowledgments

Thank you to Alex Berg and Hao Zhang who provided feature data and valuable feedback; to Simon Lacoste-Julien for helpful machine learning discussions; to Mike Howard for network support; and to Michael Maire, Adam Kirk, and Bryan Feldman for help with editing. This research was supported by a gift from Google and ONR Grant N00014-06-1-0734.

References

[1] A. Berg, T. Berg, and J. Malik. Shape matching and object recognition using low distortion correspondence. In *CVPR*, 2005. 2, 7

[2] A. Berg and J. Malik. Geometric blur for template matching. In *CVPR*, pages 607–614, 2001. 1, 2, 5

[3] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive aggressive algorithms. *Journal of Machine Learning Research*, 7, Mar 2006. 4, 6

[4] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach testing on 101 object categories. In *Workshop on Generative-Model Based Vision, CVPR*, 2004. 1, 2, 6, 7

[5] A. Frome. *Learning Distance Functions for Exemplar-Based Object Recognition*. PhD thesis, UC Berkeley, 2007. 4

[6] A. Frome, Y. Singer, and J. Malik. Image retrieval and classification using local distance functions. In *NIPS*, 2006. 2, 3, 4, 5, 7

[7] A. Globerson and S. Roweis. Metric learning by collapsing classes. In *NIPS*, 2005. 3

[8] K. Grauman and T. Darrell. Approximate correspondences in high dimensions. In *NIPS*, 2006. 2, 3

[9] K. Grauman and T. Darrell. Pyramic match kernels: Discriminative classficiation with sets of image features (version 2). Technical Report MIT_CSAIL_TR_2006-020, MIT, March 2006. 2, 3, 6, 7

[10] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report UCB/CSD-04-1366, California Institute of Technology, 2007. 6, 7

[11] A. D. Holub, M. Welling, and P. Perona. Combining generative models and fisher kernels for object recognition. In *ICCV*, 2005. 2, 7

[12] M. Kearns and D. Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. *Neural Computation*, 11:1427–1453, 1999. 6

[13] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 2, 3, 5, 7

[14] D. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1000–1015, Sep 1999. 1, 2

[15] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color and texture cues. *TPAMI*, 26(5):530–549, May 2004. 5

[16] J. Mutch and D. G. Lowe. Multiclass object recognition with sparse, localized features. In *CVPR*, 2006. 2, 5, 7

[17] J. Platt. *Advances in Kernel Methods - Support Vector Learning*, chapter Fast Training of Support Vector Machines using Sequential Minimal Optimization, pages 185–208. MIT Press, 1998. 6

[18] M. Schutlz and T. Joachims. Learning a distance metric from relative comparisons. In *NIPS*, 2003. 2, 3, 4

[19] T. Serre, L. Wolf, and T. Poggio. Object recognition with features inspired by visual cortex. In *CVPR*, 2005. 2, 7

[20] S. Shalev-Shwartz, Y. Singer, and A. Ng. Online and batch learning of pseudo-metrics. 2004. 6

[21] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998. 4

[22] G. Wang, Y. Zhang, and L. Fei-Fei. Using dependent regions for object categorization in a generative framework. In *CVPR*, 2006. 7

[23] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Distance metric learning for large margin nearest neighbor classification. In *NIPS*, 2005. 3

[24] E. Xing, A. Ng, and M. Jordan. Distance metric learning with application to clustering with side-information. In *NIPS*, 2002. 3

[25] H. Zhang, A. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative Nearset Neighbor Classification for Visual Category Recognition. In *CVPR*, 2006. 2, 3, 5, 6, 7