Similarity Metrics for Categorization: from Monolithic to Category Specific

Boris Babenko, Steve Branson, Serge Belongie
Department of Computer Science and Engineering
University of California, San Diego
{bbabenko,sbranson,sjb}@cs.ucsd.edu

Abstract

Similarity metrics that are learned from labeled training data can be advantageous in terms of performance and/or efficiency. These learned metrics can then be used in conjunction with a nearest neighbor classifier, or can be plugged in as kernels to an SVM. For the task of categorization two scenarios have thus far been explored. The first is to train a single "monolithic" similarity metric that is then used for all examples. The other is to train a metric for each category in a 1-vs-all manner. While the former approach seems to be at a disadvantage in terms of performance, the latter is not practical for large numbers of categories. In this paper we explore the space in between these two extremes. We present an algorithm that learns a few similarity metrics, while simultaneously grouping categories together and assigning one of these metrics to each group. We present promising results and show how the learned metrics generalize to novel categories.

1. Introduction

Classification methods that are typically used in object recognition require some notion of similarity over the inputs (*e.g.* a distance metric for nearest neighbor, a kernel for SVM). Therefore, a major focus in object recognition has been placed on developing powerful image representations that capture the necessary similarity information. For example, it is well known that combining features of many different cues significantly improves recognition performance [14, 24]; the problem can therefore be reduced to finding the proper combination or weighting of these cues. This can be done by designing good features by hand, or by starting with many simple features (*e.g.* pixel values, or haar wavelets) and using training data to learn a good similarity metric (or equivalently an embedding). The latter goes by many different names and appears in various subareas of machine learning and computer vision: metric learning, cue combination/weighting, kernel combination/learning, feature selection, *etc*. Though the specifics of each subarea may differ, the basic idea boils down to finding a powerful and discriminative image representation.

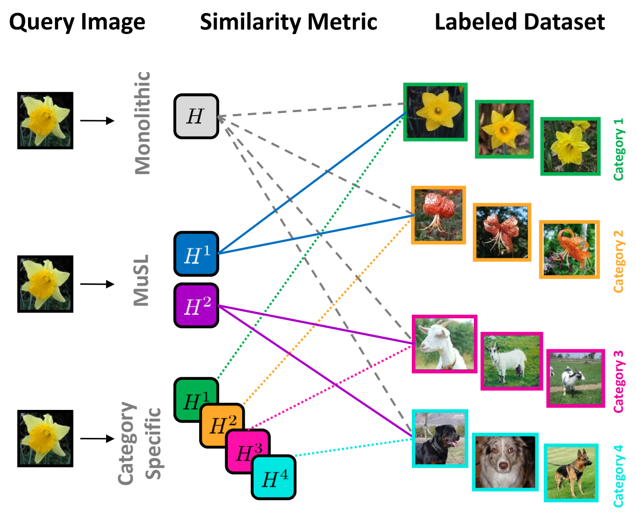

Figure 1. **Computing similarities:** At left, the top and bottom scenarios correspond to the two common ways of computing similarities between a query image and a labeled training set: in the top row we use a global, or "monolithic", similarity metric; in the bottom row we associate a different similarity metric with each category. The latter is more powerful, but does not scale well to large numbers of categories and cannot generalize to novel categories. The middle row shows a compromise between these two extremes, which we study in this paper.

For the purpose of categorization, two approaches have thus far been explored: learning a global or "monolithic" similarity metric, and learning a similarity metric per category. The former problem is known as metric learning in the machine learning community. These methods learn either a linear embedding, which is often a much lower dimensionality than the input [26, 8, 25], or a non-linear one [4, 11]. More recently, these types of methods have shown up in computer vision. In particular, the line of work on similarity sensitive hashing (or learning an embedding into a binary space) [20, 22] has produced very promising results, enabling extremely efficient image retrieval.

There are several advantages to training a monolithic similarity metric. Such a metric can be used in a nearest neighbor classifier, which can lend itself to efficient classification [21]. Furthermore, the representation is the same for all data. This is convenient because the metric can easily generalize to novel categories. In other words, we could train a good similarity metric on C categories, and if we later receive training data for a few more categories, no retraining is necessary (assuming the training data from before was reasonably representative). This form of generalization will be an important focus of our work.

The other end of the spectrum is to train a similarity metric per category. This is typically done in a 1-vs-all manner. A common example of this is training a 1-vs-all classifier that performs some form of feature/cue weighting or selection (*e.g*. SVM with kernel combination). The obvious advantage of these methods is improved performance. In fact, these types of methods have recently been shown to perform extremely well for object categorization [24]. This is not surprising since certain cues and features are important for some categories and not others. Therefore, it is difficult to capture all the relevant information in a single global metric. Other work has gone so far as to train a similarity metric per training example [7, 15].

One downside to training category specific metrics is, of course, the difficulty in scaling to large datasets. Computer vision datasets for recognition are growing at exponential rates. Consider, then, a problem with 10,000 categories. Training this many separate similarity metrics is impractical. Furthermore, we speculate that even if we did train this many metrics, many of them would be redundant. Another down side is that, unlike a monolithic similarity metric, it is unclear how this approach could generalize to new categories without an additional training phase.

Our intuition is that a *few good similarity metrics* could capture most of the necessary information and attain good performance without compromising efficiency. We see this as a happy medium in between the two extremes of generic and category specific (*cf*. Fig. 1). The contributions of this work are twofold: (1) we study how performance changes across this spectrum, and (2) we propose an algorithm called MuSL that simultaneously groups categories together and trains a few similarity metrics, one for each group. We show how to assign one of the learned metrics to novel categories, and study how well these metrics generalize.

1.1. Related Work

There are several lines of work that are similar in spirit and motivation to ours. [23] proposes a procedure for training several boosted detectors in a multi-task fashion, forcing them to share features; [18] extends this work to train classifiers incrementally rather than in a single batch. This work, like ours, aims to exploit the redundancies in representation of similar categories. Similarly, in [5] prior information is shared between categories enabling the training of new classifiers with much less training data. An approach called "dynamic learning" [27] offers an efficient way of training additional classifiers when new categories are introduced or existing categories are split. All of these approaches, however, still require the training of C classifiers for C categories. Instead we take a metric learning view; this enables us to reap some of the benefits of learning a monolithic similarity metric such as the ability to generalize to new categories without re-training.

Work on object taxonomies has similar themes. For example, [10] uses confusion matrices of trained classifiers to group categories together hierarchically. In our work, categories get grouped by virtue of sharing a similarity metric.

Perhaps the most similar approach to ours comes from the machine learning community: in [25] the input space is partitioned, and a metric is learned for each partition. This work requires the user to specify the partitions a priori (in practice, k-means on the input space is used). Our algorithm integrates grouping into the training procedure.

The rest of this paper is organized as follows. In Section 2 we review boosted similarity metric learning and introduce our algorithm. In Section 3 we go over implementation details and present object categorization results. We conclude in Section 4.

2. Boosting Similarity Classifiers

Recently, [20, 22] explored a boosting approach to embedding images into a binary space. Although this representation sacrifices some recognition power, it is efficient both in terms of computing similarities (which can be done with bit operations) and storage. Furthermore, training a boosted classifier is typically done in phases, which makes it well-suited for the type of problem we address. For these reasons, we choose this paradigm for our study, though the algorithms we discuss could be extended to other representations.

In this section we review the gradient boosting approach to learning similarity metrics, and introduce our proposed algorithm. We receive a dataset of the feature vectors $\{x_1, \ldots, x_n\}$, $x \in \mathbb{R}^d$ and category labels $\{\ell_1, \ldots, \ell_n\}$, where $\ell_j \in \{1, \ldots, C\}$. Using this dataset we can construct pairs (x_{i1}, x_{i2}); for convenience we also define pair labels $y_i = \mathbf{1}[\ell_{i1} = \ell_{i2}]$. If n_c is the number of examples for each category, then there are $\mathcal{O}(n_c^2)$ possible positive pairs, and $\mathcal{O}(Cn_c^2)$ possible negative pairs. Since the latter can be quite large, in practice we subsample negative pairs.

The goal is to train a boosted similarity metric, which we can think of as a binary classifier that takes a pair as input:

$$H(x_{i1}, x_{i2}) = \sum_{t=1}^{T} h_t(x_{i1}, x_{i2}) \tag{1}$$

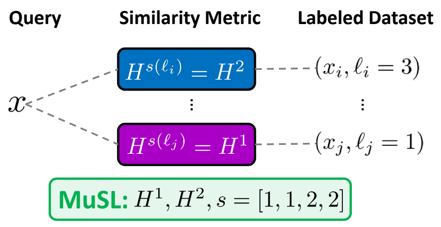

Figure 2. **The MuSL system** consists of a few similarity metrics (in this example just two: H^1, H^2) and an assignment vector s that maps each category to one of the metrics. To compute the distance from a query image to the i^{th} example in our labeled training set, we use the similarity metric $H^{s(\ell_i)}$ where ℓ_i is the label of the training example.

where each h_t is a weak similarity classifier. Following [20, 22], in our model these weak similarity metrics take the following form:

$$h_t(x_{i1}, x_{i2}) = |f_t(x_{i1}) - f_t(x_{i2})| \tag{2}$$

where $f_t : \mathbb{R}^d \to \mathbb{R}^1$. In this case, to compute H we embed each example/image x into a vector $F(x) = [f_1(x), f_2(x), \ldots f_T(x)]$, and then compute the L_1 distance in this new space: $H(x_{i1}, x_{i2}) = ||F(x_{i1}) - F(x_{i2})||_1$. Therefore, training such a strong classifier induces an embedding. We choose the function $f_t : \mathbb{R}^d \to \{0, 1\}$ so that our embedding is binary; this enables us to compute these distances very efficiently. In the simplest case, these functions are computed by thresholding a particular feature of x, $f_t(x) = \mathbf{1}[x[t] < \theta]$. [20] proposes an efficient algorithm to learn such a threshold given some training data. Note that for this choice of f_t, the overall metric $H(x_{i1}, x_{i2})$ ranges from 0 to T.

2.1. Monolithic Similarity Classifier

We begin with a review of how to train a single similarity metric for all data. For a given classifier H we can define the probability that a particular pair is positive as

$$p_i = \sigma\Big(\alpha\big(T/2 - H(x_{i1}, x_{i2})\big)\Big) \tag{3}$$

where $\sigma(a) = \frac{1}{1-\exp(-a)}$ is the sigmoid function, and α is a global scalar parameter. Using the above definition we can define the log likelihood over a set of training pairs:

$$\mathcal{L}(H) = \sum_{i|\ell_{i1}=\ell_{i2}} \log(p_i) + \sum_{i|\ell_{i1}\neq\ell_{i2}} \log(1 - p_i) \tag{4}$$

To derive a boosting algorithm that optimizes the above objective function we apply Friedman's gradient boosting framework [6]. We interpret boosting as gradient ascent in function space. Each step of the ascent equates to adding a new weak classifier to H. The gradient $\frac{\partial \mathcal{L}(H)}{\partial H}$ gives us the direction in function space in which we should move; however, we are limited by our choice of weak classifier, and cannot move in arbitrary directions. We therefore seek a weak classifier that is as close as possible to this gradient. When the weak classifiers have binary output (as is the case for us), this is equivalent to minimizing weighted error on the training data [1], where the weights are defined as:

$$w_i = \left|\frac{\partial \mathcal{L}(H)}{\partial H}\right| \tag{5}$$

In each phase of training we compute these weights and train a weak classifier.

Note that the algorithm presented in [20, 22] is slightly different because an exponential objective function is used instead of log likelihood; we have found that empirical differences are insignificant between these two choices.

Algorithm 1 Multiple Similarity Learning (MuSL)

```
INPUT: Training data $(x_{i1}, x_{i2})$ and labels $(\ell_{i1}, \ell_{i2})$
1: for $t = 1$ to $T$ do
2:   for $k = 1$ to $K$ do
3:     Compute weights
         $w_i^k = \frac{\exp(r\mathcal{L}_c^k)}{\sum_j \exp(r\mathcal{L}_c^j)} \left|p_i^k - y_i\right|$
4:     Train weak classifier $h_t^k$ using weights $w_i^k$
         $h_t^k = \mathrm{argmin}_h \sum_i w_i^k \mathbf{1}[h(x_{i1}, x_{i2}) \neq y_i]$
5:     Update strong classifier $H^k \leftarrow H^k + h_t^k$.
6:   end for
7: end for
8: Assign $s(c) = \mathrm{argmax}_k \mathcal{L}_c(H^k)$ for $c = 1 \ldots C$
OUTPUT: Classifiers $H^1 \ldots H^K$, assignment vector $s$
```

2.2. Per-Category Similarity Classifiers

In this scenario we train a similarity classifier for each category: $H^1, \ldots, H^C$. We can then use these in a kNN framework as follows: upon receiving a novel example x we will compute similarities to all the examples in the training set. To compute the similarity of x and a training example x_i we will use the classifier H^{ℓ_i}.

For each class c, we construct a set of training pairs (x_{i1}, x_{i2}). A pair is positive if $\ell_{i1} = \ell_{i2} = c$ and negative if $\ell_{i1} \neq \ell_{i2}$ and $\ell_{i2} = c$. All other training pairs are not relevant to training H^c. Hence, we optimize the following objective:

$$\mathcal{L}_c(H) = \sum_{i|\ell_{i1}=c,\ell_{i2}=c} \log(p_i) + \sum_{i|\ell_{i1}\neq c,\ell_{i2}=c} \log(1 - p_i) \tag{6}$$

We can train each of the C classifiers separately, using the same training procedure as before – the only change is that we train only on relevant pairs rather than all pairs.

Recall that due to our choice of binary weak classifiers, all of the similarity metrics have the same range of output (0

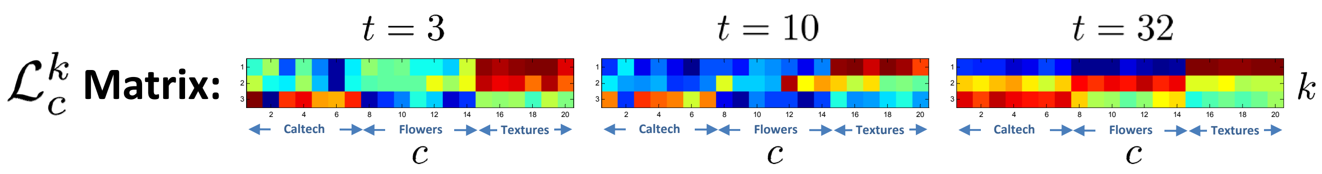

Figure 3. **Evolution of likelihoods:** as training proceeds the values of the matrix $\mathcal{L}_c^k$ converge, and clear category groupings become apparent. Above is a snapshot of the matrix (red indicates high likelihood, blue indicates low likelihood) for 3 different stages of training for the MERGED 20 dataset, where we combine 7 Caltech 256 categories, 7 Oxford Flowers categories, and 6 UIUC Texture categories (*cf*. Section 3.3.1 for details). Above we see that by the end of the training procedure, the matrix $\mathcal{L}_c^k$ reflects the discovered grouping.

to T). An advantage of this choice is that there are no issues about calibration that sometimes arise in 1-vs-all classification [19]. For other choices of weak classifiers, this issue would need to be addressed.

2.3. Multiple Similarity Learning (MuSL)

Finally, we are interested in training a small number of similarity metrics $H^1, \ldots, H^K$ where $K < C$. To do this, our algorithm groups categories together into K "super-categories". In other words, we will need to recover a vector s of length C, each entry of which is $s(c) \in \{1, \ldots, K\}$. To compute the similarity between a novel example x and an example from our training set x_i we use the similarity classifier $H^{s(\ell_i)}$ (*cf*. Fig. 2 for an example). If this assignment vector were known a priori, our problem would reduce to standard training. We can therefore think of this vector as a latent variable. It is plausible to think of heuristic methods of finding this assignment vector as a pre-processing step. However, by separating this step from the training, the two cannot be jointly optimized.

We would like to solve the following optimization problem:

$$\max_{s, H^1 \ldots H^K} \sum_c \mathcal{L}_c^{s(c)}$$

where $\mathcal{L}_c^k = \mathcal{L}_c(H^k)$ is the log likelihood of category c when evaluated with classifier H^k. We can split the max over metrics H and entries of the assignment vector s, and the move the latter into the sum:

$$\max_{H^1, \ldots, H^K} \sum_c \max_{s(c) \in \{1 \ldots K\}} \mathcal{L}_c^{s(c)}$$

Therefore, to solve for the best similarity metrics we use the following objective function[1]:

$$\mathcal{L}(H^1, \ldots, H^K) = \sum_c \max_{k \in \{1 \ldots K\}} \mathcal{L}_c^k \tag{7}$$

When $K = 1$ this objection function reduces to the one in Eqn. 6. As before we will derive a boosting algorithm to optimize this objective by performing gradient ascent in function space. However, when we try to take the derivative of the above objective function we run into trouble since the max operator is not differentiable. We therefore replace the max with a differentiable approximation [3]:

$$\begin{aligned} \mathcal{G}(a_1, \ldots, a_K) &= \frac{1}{r} \log\Big(\sum_k \exp(r a_k)\Big) \qquad (8) \\ &\approx \max_k \{a_1, \ldots, a_K\} \end{aligned}$$

where r is a parameter controls the accuracy of the approximation (we discuss how to deal with possible numerical instability issues of the above equation in Section 3.1.2).

We will optimize the objective $\mathcal{L}(H^1, \ldots, H^K)$ by coordinate ascent, updating each of the K classifiers one at a time. In each step we will add a new weak classifier h_t^k to the strong classifier H^k. We train the weak classifier h_t^k with weights derived as follows:

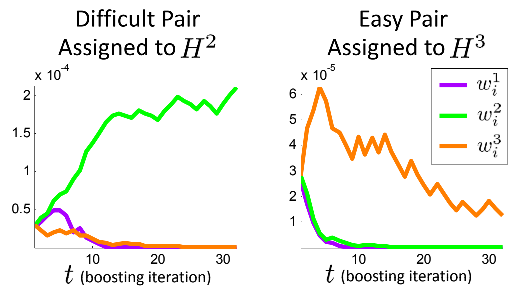

Figure 4. **Evolution of pair weights:** the plots above show the weight of two training pairs from the MERGED 20 dataset (*cf*. Section 3.3.1 for details) with $K = 3$. As training proceeds, the leftmost term in Eqn. 9 softly assigns each category, and hence each training pair, to one of the K metrics; the weights corresponding to the other metrics quickly drop down to 0. In later stages of training the rightmost term in Eqn. 9 begins to take effect, and the weights of the "difficult" pairs (left plot) stay high, while the weights of the "easy" pairs (right plot) begin to decline; this is similar to traditional boosting.

[1]Note that this objective is only concave for $K = 1$; in all other cases it is possible to get stuck in local maxima, though in our experiments we have not found this to be too problematic.

$$
\begin{aligned}
w_i^k &= \left|\frac{\partial \mathcal{L}(H^1,\dots,H^K)}{\partial H^k}\right| \\
&= \left|\frac{\partial \mathcal{G}(\mathcal{L}_c^1,\dots,\mathcal{L}_c^K)}{\partial \mathcal{L}_c^k}\frac{\partial \mathcal{L}_c^k}{\partial p_i^k}\frac{\partial p_i^k}{\partial H^k}\right| \\
&= \frac{\exp(r\mathcal{L}_c^k)}{\sum_j \exp(r\mathcal{L}_c^j)}\left|p_i^k - y_i\right| \qquad (9)
\end{aligned}
$$

where p_i^k is the probability of a pair (computed using Eqn. 3) according to classifier H^k. The overall algorithm is summarized in Algorithm 1. The above formula has an intuitive interpretation. It is composed of two terms; the rightmost term gives higher weights to pairs that are currently misclassified (*e.g.* "difficult" pairs), similar to traditional boosting. The leftmost term is the familiar softmax equation [2], applied to the category specific likelihood $\mathcal{L}_c^k$. We can think of this as a soft approximation of $\mathbf{1}[k = \operatorname{argmax}_j \mathcal{L}_c^j]$. This term will give higher weight to pairs where $\ell_{i2} = c$ if the similarity metric H^k is the "best" for category c. To gain further intuition for how these two terms interact we plot the evolution of weights for two training pairs in Fig. 4.

We initialize all weights to be uniform. As a result, the first few weak classifiers chosen by the procedure are not "tuned" to particular categories. However, as the training proceeds, the leftmost term in the weight equation starts to converge and effectively assign categories to each classifier (*cf*. Fig. 3 to see how the values of $\mathcal{L}_c^k$ evolve during training).

2.3.1 Assignments and Out of Sample Extensions

At the end of the training procedure we recover the assignment vector s as follows:

$$
s(c) = \operatorname*{argmax}_k \mathcal{L}_c^k \qquad (10)
$$

This may be obvious for the categories that were included in the training data. The more interesting aspect of the above equation is that it can also be used for out of sample extension to novel categories. That is, suppose we trained K metrics on a few categories, and now we receive training data for one more category; we would like to assign one of the existing metrics to this new category. The above equation is appropriate for this scenario as well: we generate positive and negative training pairs for this category as before, compute log likelihood of each similarity classifier, and pick the best one.

Note that we do not enforce any sort of balance on the assignments; it is possible that all categories get assigned to the same similarity metric. In practice, however, this does not seem to be an issue because assigning metrics to categories more evenly is advantageous in terms of optimizing the objective function.

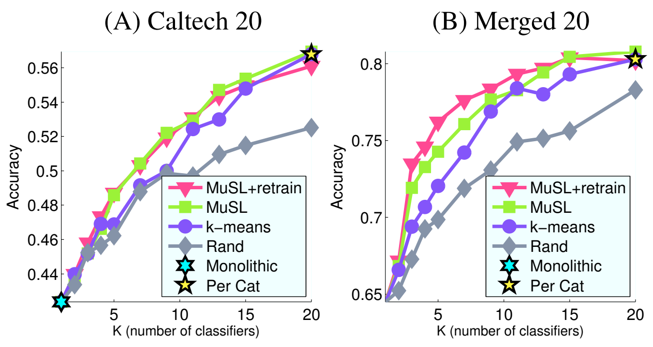

Figure 5. **Categorization performance** for two datasets, showing the two extremes, monolithic and per-category similarity metrics, as well as the algorithms discussed for a range of K values. For low values of K we are able to get significant improvement in performance. See Section 3.2 for details.

2.3.2 Re-training

Recall that because the weights are initialized uniformly, the first weak classifiers are not tuned to specific categories. If the total number of weak classifiers is large, this would not have a significant effect. However, in our implementation we use a small number of weak classifiers (for efficiency reasons). Once the training phase is complete and we have recovered the category assignments, we *re-train* the similarity metrics in a standard way using the known assignments. This step tends to improve performance.

3. Experiments

In this section we present our results on object categorization. We begin with an overview of implementation details, which are not required to follow the rest of this Section.

3.1. Implementation Details

3.1.1 Image Representation

We use a bag of words framework for constructing our weak classifiers for boosting, where each decision stump is a threshold on the count of some visual word. To construct our visual codebook, we use an algorithm similar to [16], where the vocabulary is constructed from a forest of random trees, and each node in a tree corresponds to a visual word. Tree nodes are split by randomly sampling a small number of decision stumps (we used thresholded haar-like filters computed on some image channel), and choosing the split that minimizes the total class entropy on a labeled training set. We constructed two random forests, one using CIE-LUV color images, and another using histogram of gradients images with 8 orientation bins. The main advantage of this style of features is speed: assigning an interest point to a codeword involves evaluating a small number of haar-like features, and thus an image can be processed in a fraction of a second. It also allows for easy integration of multiple

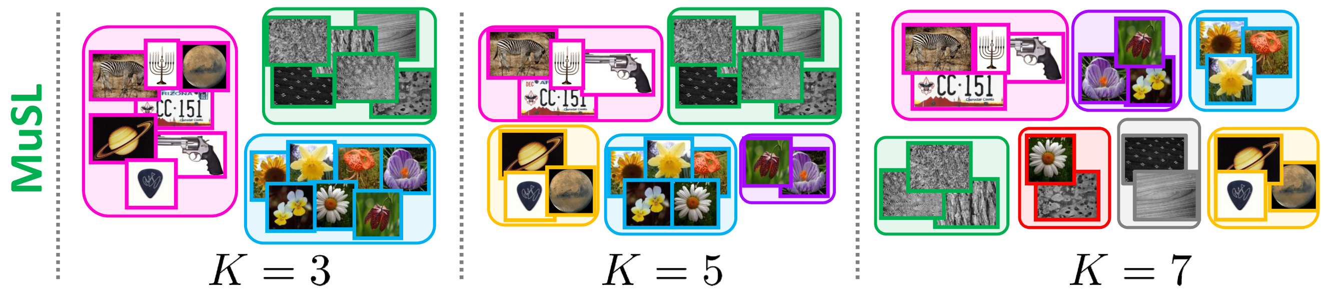

Figure 6. Discovered category groupings for the Merged 20 dataset for various values of K using MuSL. See Section 3.3.1 for details.

cues by using different types of image channels. We emphasize that our metric learning algorithm is not in any way tied to this style of features, and we expect that absolute performance could be improved by using other more advanced features.

3.1.2 Algorithm Parameters

Here we briefly discuss the parameters and caveats of our algorithm. When constructing training pairs given labeled data we randomly sample 1,000 negative pairs for each example and take every possible positive pair.

One potential problem with the expression in Eqn. 9 is that the exponent can blow up for large values of $\mathcal{L}_c^k$. Recall the parameter r in Eqn. 8; we can set this parameter to a small value to avoid numerical instability. The value of the likelihood $\mathcal{L}_c^k$ depends on the number of examples n. We found that setting $r = 500/n$ is a good compromise between avoid numerical problems and having reasonable accuracy in the max approximation. We set the number of weak classifiers $T = 32$. Finally, as suggested in [22], we set $\alpha = 0.1$.

3.2. Categorization

Our first task is to measure categorization performance as a function of K. We plug our learned similarity metrics into a nearest neighbor classifier, as described in Section 2. Recall that we are using embeddings into binary spaces, as was done in [22, 20] (*e.g.* $f_t : \mathbb{R}^d \rightarrow \{0, 1\}$). There are a few advantages to this choice. First, computing distances in binary space is extremely efficient. Furthermore, since each weak classifier relies on only one feature, boosting essentially performs feature selection. This can further speed up run time as we need to compute fewer features for an incoming image. Nevertheless, as mentioned before, this choice also limits the power of our learned metrics to some degree. Therefore, our results are not state of the art in terms of absolute performance – we are mainly interested in studying the relative performance of these algorithms as we change the value of K.

We compare the MuSL algorithm to two baselines, both of which perform grouping of categories as a pre-processing step, and train similarity metrics in a standard way. The first baseline is to group the categories randomly. The second is to use the k-means clustering algorithm [2] to group the categories. To do this, we first need a way of representing each category in some feature space. While it is not clear what the "correct" way of doing this is, we use the following heuristic: we take a mean of all the data points corresponding to that category $\overline{x}_c = 1/n_c \sum_{\ell_i = c} x_i$; this results in C vectors, one for each category. We experimented with other k-means based heuristics, but found this to be the only one that gives reasonable performance.

We perform experiments with two different datasets described below. For all experiments we use 30 training images per category for training, and 10 to 20 images per category for testing. The results shown are averages of 20 trials with different train/test splits. In all experiments we report the average recognition accuracy (mean of the diagonal of the confusion matrix).

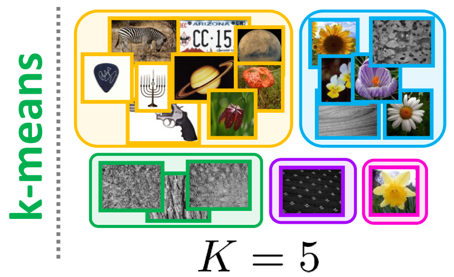

Figure 7. Discovered category groupings for the Merged 20 dataset for various values of K using the k-means heuristic. See Section 3.3.1 for details.

3.3. Caltech

Our first experiment uses the Caltech 256 (CT) dataset [9]. We took the 30 easiest categories[2] and randomly chose a subset of 20 out of these (the other 10 will be used in later experiments); we call this dataset "Caltech 20". Catego-

[2]Difficulty is measured by performance of a standard classification method; see [9] for details.

rization accuracy for various values of K are shown in Fig. 5(A). As we slide the value of K from 1 to 20 the recognition performance gradually increases; the difference between the two extremes is 14%. Randomly grouping categories together performs worse than the "smarter" ways of discovering super-categories, suggesting that certain groupings are better than others. Finally, MuSL tends to perform the best out of all the methods, though in this application the k-means heuristic performs fairly well. We will see some disadvantages to the k-means heuristic in the next two sections.

3.3.1 Merged

One strength of our algorithm is its ability to handle datasets that consist of a wide variety of heterogenous categories. We constructed a dataset we called "Merged 20"; it consists of 7 categories from CT [9], 6 categories from Oxford Flowers-17 (FL) [17], and 6 categories from UIUC Textures (UT) [13]. The CT categories were chosen randomly from the 20 categories in the previous section; the FL and UT categories were chosen randomly from their respective datasets. Fig. 5(B) shows a plot of recognition accuracy versus K. We see that for low values of K there is a dramatic increase in performance – for $K = 5$ recognition accuracy is 10% higher than for a single metric. MuSL outperforms both the random and the k-means heuristic methods of grouping; for $K = 5$ MuSL achieves an accuracy almost 5% higher than the others. Finally, we see that the re-trained MuSL metrics increase performance by a couple percent.

It is interesting to qualitatively inspect the super-categories that MuSL discovers; one of the advantages of this dataset is that we have some expectation for what these super-categories could be. Fig. 6 shows the discovered groupings for $K = 3, 5, 7$ for a particular train/test fold. For $K = 3$ the groupings correspond to the 3 source datasets that we merged; MuSL recovers these super-categories automatically. The k-means method of grouping, on the other hand, does not recover these super-categories. For $K = 5$ MuSL breaks the 3 source datasets down further, in a seemingly intuitive manner. For example, the 'mars', 'saturn' and 'guitar-pick' categories get grouped together; these objects are roughly circular and have strong gradients on their boundaries due to constant backgrounds. Again, the super-categories discovered by k-means are semantically arbitrary (*cf*. Fig. 7 for an example).

Finally, we note that the groupings produced by MuSL are much more consistent, or "stable", over the different train/test folds: using the stability measure defined in [12], for $K = 3$ MuSL groupings are 96% stable, while k-means groupings are only 50% stable. This means that super-categories discovered by MuSL are almost always the same, whereas those discovered by k-means are fairly random.

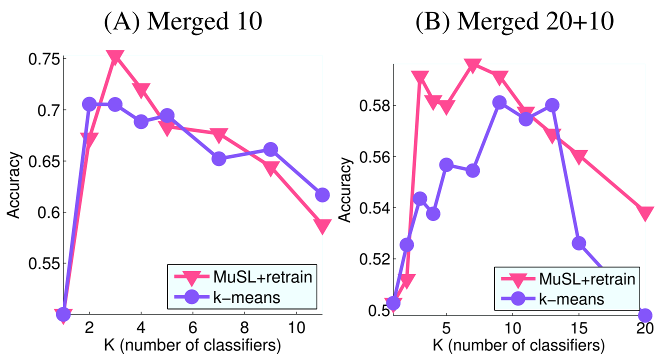

Figure 8. **Generalizing to novel categories:** using a few similarity metrics achieves much better performance than using just one; however, using too many can overfit to the original categories. See Section 3.4 for details.

3.4. Generalizing to New Categories

We would like to study the ability of the learned similarity metrics to generalize to new categories. It is important to highlight the difference of this generalization as opposed to the traditional definition. Here we are interested in generalizing to novel categories (both train and test data), rather than generalizing to test data that consists of the same categories as the training data. To this end, we create a new dataset we call "Merged 10"; it consists of 10 new categories (4 from CT, 3 from FL, and 3 from UT) that were not included in Merged 20. We train similarity metrics only on the original categories from the previous section (Merged 20), and test on these new categories, as well as a combination of all 30 categories, which we call "Merged 20+10". To do this for $K > 1$ we use the procedure described in Section 2.3.1 to assign one of the learned similarity metrics to each of the new categories. For the Merged 10 data, when $K = 3$ we observe that the assignments are as expected – each of the source datasets get grouped together as before. This suggests that the assignment procedure is reasonable.

We plot the performance of MuSL (the re-trained version) and the k-means heuristic method as we sweep through values of K in Fig. 8. The left plot shows performance only on the new categories, and the right plot shows performance on all 30 categories. The resulting plots are surprising. Unlike before, performance does *not* strictly improve as K gets larger. Instead, performance actually starts to degrade for higher values of K. This suggests that training with a lower value of K results in more general similarity metrics. Therefore, having too many similarity metrics is not only computationally wasteful (in terms of training time and storage), the performance is actually worse when adding new categories to the data. At the same time, training just one similarity metric performs much worse (almost 25% worse than when $K = 3$ on the Merged 10 dataset). Furthermore, we see that MuSL performs better than the k-means heuristic method for both datasets – for the Merged 20+10 dataset the difference is about 5%.

4. Conclusions & Future Work

In this paper we presented a method for learning a few similarity metrics from labeled data. We studied how performance changes in between the two extremes of a single metric and a metric per object category, and showed that the performance of the latter can be matched fairly closely with a small number of metrics. We also studied how these learned metrics generalize to novel categories; such generalization is strongly desirable if we wish to scale to large datasets. Here we saw that training *too many* metrics is actually detrimental to this type of generalization. The algorithm we proposed, MuSL, simultaneously trains a few similarity metrics and groups categories together. Though a number of heuristic methods for grouping categories as a pre-processing step are possible, our method is principled and tends to perform better because the optimization of metrics and grouping is done jointly. In particular, our method is well suited to scenarios where the categories exhibit a super-categorical structure.

There are several paths we would like to explore in the future. First, we are currently working on experiments with larger datasets to see how these methods scale, and investigating more powerful features. Though our focus in this work is categorization, the ideas we presented could prove to also be useful in the domain of image retrieval. For example, given an image query and a large unlabeled dataset of images, the task of retrieving images similar to the query is ambiguous when we acknowledge the fact that there is no single "correct" similarity metric. Perhaps it is best to return several sets of results to the user, one for each of the similarity metrics. We intend to explore these ideas in the future.

Acknowledgements

This research was supported in part by the UCSD FW-Grid Project (NSF Infrastructure Grant no. EIA-0303622), NSF CAREER Grant #0448615, NSF IGERT Grant DGE-0333451, and ONR MURI Grant #N00014-08-1-0638. The authors would also like to thank Vincent Rabaud and Piotr Dollár for valuable help and feedback.

References

[1] B. Babenko, P. Dollár, Z. Tu, and S. Belongie. Simultaneous Learning and Alignment: Multi-Instance and Multi-Pose Learning. In *Faces in Real-Life Images*, 2008.

[2] C. Bishop. *Pattern Recognition and Machine Learning*. NJ, USA. Springer, Heidelberg, 2006.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge Univ. Press, 2004.

[4] S. Chopra, R. Hadsell, and Y. LeCun. Learning a similarity metric discriminatively, with application to face verification. In *CVPR*, 2005.

[5] L. Fe-Fei, R. Fergus, and P. Perona. A Bayesian approach to unsupervised one-shot learning of object categories. In *ICCV*, 2003.

[6] J. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Stat.*, 29(5):1189–1232, 2001.

[7] A. Frome, Y. Singer, F. Sha, and J. Malik. Learning Globally-Consistent Local Distance Functions for Shape-Based Image Retrieval and Classification. In *ICCV*, 2007.

[8] J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *NIPS*, 2005.

[9] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, CalTech, 2007.

[10] G. Griffin and P. Perona. Learning and Using Taxonomies For Fast Visual Categorization. In *CVPR*, 2008.

[11] M. Jones and P. Viola. Face Recognition Using Boosted Local Features. Technical Report TR2003-25, MERL, 2003.

[12] T. Lange, V. Roth, M. Braun, and J. Buhmann. Stability-based validation of clustering solutions. *Neural Computation*, 16(6):1299–1323, 2004.

[13] S. Lazebnik, C. Schmid, and J. Ponce. Sparse texture representation using affine-invariant neighborhoods. In *CVPR*, 2003.

[14] B. Leibe and B. Schiele. Analyzing Appearance and Contour Based Methods for Object Categorization. In *CVPR*, 2003.

[15] Y. Lin, T. Liu, C. Fuh, and T. Sinica. Local ensemble kernel learning for object category recognition. In *CVPR*, 2007.

[16] F. Moosmann, B. Triggs, and F. Jurie. Fast discriminative visual codebooks using randomized clustering forests. In *NIPS*, 2007.

[17] M. Nilsback and A. Zisserman. A visual vocabulary for flower classification. In *CVPR*, 2006.

[18] A. Opelt, A. Pinz, and A. Zisserman. Incremental learning of object detectors using a visual shape alphabet. In *CVPR*, 2006.

[19] R. Rifkin and A. Klautau. In Defense of One-Vs-All Classification. *JMLR*, 5:101–141, 2004.

[20] G. Shakhnarovich. *Learning Task-Specific Similarity*. PhD thesis, Massachusetts Institute of Technology, 2006.

[21] G. Shakhnarovich, T. Darrell, and P. Indyk. *Nearest-neighbor methods in learning and vision: Theory and practice*. MIT Press, 2005.

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

[23] A. Torralba, K. Murphy, and W. Freeman. Sharing features: efficient boosting procedures for multiclass object detection. In *CVPR*, 2004.

[24] M. Varma and D. Ray. Learning The Discriminative Power-Invariance Trade-Off. In *ICCV*, 2007.

[25] K. Q. Weinberger, J. Blitzer, and L. K. Saul. Fast Solvers and Efficient Implementations for Distance Metric Learning. In *ICML*, 2008.

[26] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In *NIPS*, 2003.

[27] T. Yeh and T. Darrell. Dynamic visual category learning. In *CVPR*, 2008.