Automating Art Print Authentication Using Metric Learning

Charles Parker
Eastman Kodak Company
1999 Lake Avenue
Rochester, NY 14650
charles.parker1@kodak.com

Paul Messier
Paul Messier, LLC
103 Brooks Street
Boston, MA 02135
pm@paulmessier.com

Abstract

An important problem in the world of art historians is determining the type of paper on which a photograph is printed. One way to determine the paper type is to capture a highly magnified image of the paper, then to compare this image to a database of known paper images. Traditionally, this process is carried out by a human and is generally time-intensive. Here we propose an automated solution to this problem, using wavelet decomposition techniques from image processing, as well as metric learning from the machine learning area. We show, on a collection of real-world images of photographic paper, that the use of machine learning techniques produces a much better solution than image processing alone.

Introduction

One of the most basic determinations an art historian must make about an unknown photographic print is the date it was printed. Often, the date a print was made can determine whether it is a valuable work of art or a less valuable reproduction.

A reasonably accurate determination of the age of a photographic print can often be made if one knows the type of paper on which it is printed. This information, however, can often be difficult to obtain. One way to make this determination is to capture an image of the paper at very high magnification. Such an image reveals microscopic details of the paper that can be very different depending on the manufacturing process and final attributes of the paper.

In the past, this search was conducted by a human, who would manually compare the high magnification image of the unknown, or "query" paper to that of all known, or "target" papers. The human would then rank the known papers in order of visual similarity, and use other factors to determine the correct paper from this ranked list. Because there are many types of photographic paper (at least in the thousands), this would often take considerable time, especially if the database was particularly comprehensive.

To alleviate this problem, we propose a solution inspired by machine learning and image processing. In the image processing step, we reduce each image to a vector of features sufficient to discriminate between images. More specifically, we use a four-level *wavelet decomposition* (Akansu and Haddad 1992) of each image, taking the L_1 and L_2 norms of each wavelet sub-band in the decomposition. This turns each image into a vector $v \in \Re^n$ where n is the number of features. We show that this formulation of the feature vector outperforms several other common feature vectors.

The machine learning step is using *metric learning* (Xing et al. 2002) to learn a *Mahalanobis distance* that best incorporates our idea of "visual similarity" into the automated retrieval process. The general idea is to mark certain pairs of images as "similar" or "dissimilar", then to use this information to learn a projection matrix. After projecting the images into the space implied by this matrix, the similar images will be closer together, and the dissimilar images will be further apart. If we are then presented with a new query paper, we can perform the wavelet decomposition, multiply the resulting vector by the projection matrix, and compute the distance from the query paper to each target in the projected space. The distances to each target imply a similarity ranking of targets with respect to the given query.

The rest of this paper is organized as follows: We will first describe our feature extraction process, briefly reviewing wavelet transformations and generalized Gaussian densities. The following section reviews metric learning and ranking algorithms. We then show experimental results on a real-world database of photographic papers, and provide some concluding analysis.

Feature Extraction for Texture Recognition

The problem we have described is known in the image processing literature as *texture recognition* (Do and Vetterli 2002). While our version of the problem shares some qualities with the versions described in the literature, our version also has two important differences:

1. In the traditional instance of the problem, the number of possible textures is relatively small (say, < 30). In our version, this number is equal to the number of known photographic papers in our database, which could number in the thousands or more.
2. An "answer" to the traditional version of the problem consists of a single texture. That is, given a query image, we return the predicted texture. In our version, we would

prefer to have a *ranked list* of possible textures. In other words, our answer to the given query will be an ordering of the known targets rather than a single target.

These two differences combine to rule out many of the standard multiclass classification algorithms as viable approaches to the problem, as standard classifiers do not scale to thousands of classes, and many (such as decision trees) do not have an obvious way of producing an appropriate ordering for the classes.

Fortunately, one classifier that meets these conditions, the 1-Nearest-Neighbor classifier (Aha, Kibler, and Albert 1991), will prove flexible enough to incorporate several different learning methods inside its formalism. Essentially, the idea is to define a function $\mathbf{\Phi} : I \mapsto \Re^n$ that takes an image I and maps it to an n-dimensional vector, or a point in an n-dimensional space. We do this for all target images (known photographic papers) $\mathbf{t}$ in our target database, $\mathcal{T}$. When a new query image, $\mathbf{t}_i \in \mathcal{T}$, is given to the system, we simply apply the same function, mapping the image to an n-element vector. We can then order the targets according to their n-dimensional Euclidean distance from the query.

Building on this idea, we define $d(\mathbf{t}_i, \mathbf{q}) = |\mathbf{\Phi}(\mathbf{t}_i) - \mathbf{\Phi}(\mathbf{q})|$ to be the absolute value of the difference in feature vectors between the target $\mathbf{t}_i$ and the query $\mathbf{q}$. The distance between then becomes the norm of the difference vector, $||d(\mathbf{t}_i, \mathbf{q})||$. The important question, then, is how to construct the function $\mathbf{\Phi}$ so that distance in the implied space closely matches our notion of visual similarity. We will now explore several possibilities for construction of this function.

Some Feature Functions

The first feature functions we will explore are relatively simple in nature, and require little special purpose processing.

L_1 and L_2 Norms of the Entire Image One simple approach is to simply take the L_1 and L_2 norms (or the mean and variance) over all pixels in the image. This gives us a 2-d space (that is, the number of features, $n = 2$) in which our images are embedded.

Gray-level Histogram A slightly more complex vector is given by putting each pixel of the image into a "bin" according to its gray-level. The numbers of pixels in each bin then become the elements of our vector. For our experiment, we will use 16 equally spaced bins, giving us $n = 16$ dimensions in our final space.

Visual Vocabulary The previous two feature functions involve *global* features only. That is, features that are aggregated across the entire image. A more "local" way to construct features is to break the image into non-overlapping 16×16 pixel windows. Each of these windows will have its own gray level histogram. If we do this for several images, we will have a fairly large set of small histograms. We can then run k-means clustering (Jain and Dubes 1988) on this set to obtain a *visual vocabulary* (Li and Wang 2003) of size k. These k means, or "visual words" become the basis for our feature vector. That is, we begin processing each image by initializing a feature vector of k zeros. The images are then processed by obtaining a gray-level histogram for each window in the image, determining to which of the k means this histogram is closest, and then incrementing the appropriate count in our feature vector. This gives us a feature vector of length $n = k = 20$ in our experiments.

If there are distinguishing local features in certain parts of certain images, then the resulting feature vector should have high counts at the visual word that most closely matches this feature, and low counts for that word otherwise.

Wavelet-based Feature Functions

Following certain suggestions in the literature (Do and Vetterli 2002), we will now introduce several new feature functions based on wavelet decomposition.

Wavelet decomposition (Akansu and Haddad 1992) is a simple idea in principle: Given some signal, we subject it to a high-pass and a low-pass filter. This gives us one version of the signal that is *smoothed* and another that is *edge-detected*. If we choose the filters carefully, we end up with two signals at half the resolution of the original signal that can be combined in a clever way to retrieve the original signal. Essentially, we can use the edge-detected signal to replace the missing edge information in the smoothed image.

A key element of this is that it can be performed recursively, so that the smoothed image at a lower resolution becomes a candidate for decomposition itself. This can continue until the smoothed signal is too low in resolution to perform meaningful analysis. This gives us a series of signals, or wavelet bands, where the magnitude of the signal at each resolution tells us the amount of "energy" (how much the signal is changing) at that resolution.

When we perform the two-dimensional version of this analysis on an image, the result is four new images: Three edge-detected *subbands*, plus the smoothed image. In these experiments, we do four levels of decomposition, and so 13 images are generated during the composition. A two-level wavelet decomposition of an image is show in Figure 1.

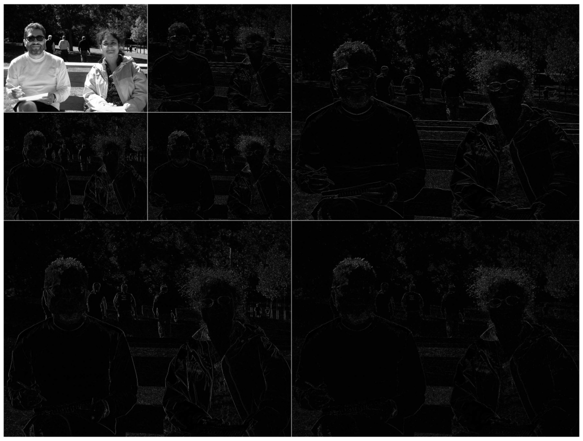

Figure 1: A two-level wavelet decomposition

Now that we have 12 images generated from our original, we can simply apply the techniques described in the previous subsection to each of those 13 images. This gives us 26 features for the L_1 and L_2 norms case, 208 features for the histogram case, and 260 features for the visual vocabulary case. In the latter two cases, we can use principle components analysis (Fukunaga 1990) to reduce the features to a manageable number, 25 and 50 respectively. In our experiments, however, this reduction was not a factor in any sense except reducing running time.

The Generalized Gaussian Distribution

Following other suggestions in the literature (Do and Vetterli 2002), we give one more possible feature vector formulation. In the last section, we used three techniques to "summarize" each wavelet sub-band: L_1 and L_2 norms, gray-level histograms, and visual vocabularies. We can also attempt to fit a generalized Gaussian distribution to each sub-band. The zero-mean generalized Gaussian distribution takes the following form:

$$\mathrm{ggd}(x; \alpha, \beta) = \frac{\beta}{2\alpha\Gamma(1/\beta)} e^{-(|x|/\alpha)^{\beta}}$$

where β is the "shape parameter" and α is related to the standard deviation. As β gets smaller, the distribution becomes shallower and more "pointed". The distribution has, as a special case, the standard Gaussian distribution at $\beta = 2$. Figure 2 shows several generalized Gaussian curves as β is varied.

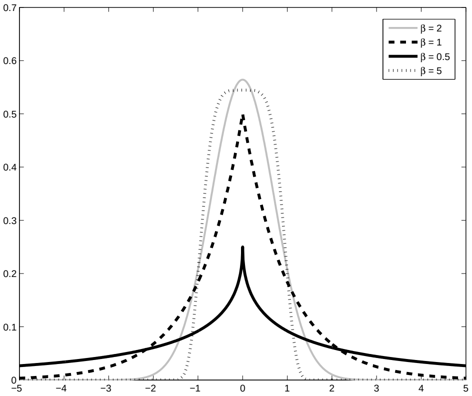

Figure 2: The generalized Gaussian distribution, under several values of β

While it is difficult to fit the generalized Gaussian distributions to a set of values (e.g., pixels from a wavelet subband), it is possible, and we have implemented a procedure to find these parameters (Do and Vetterli 2002) in MATLAB. Our final feature vector is α and β for each of the wavelet sub-bands, except the final smoothed image. This results in a 24-element feature vector.

Metric Learning and Ranking

Once we have defined our feature extraction function, we can easily sort our list of targets according to Euclidean distance from a given query image in the space defined by the feature function.

But is this the best we can do? Suppose we had some "side information" about which pairs of targets were similar and which were not. In this case, is it possible to "warp" the space in which our images are embedded so that our side information was better satisfied? That is, can we change the space so that those targets that we knew to be similar were closer together in the changed space and dissimilar targets were further apart? More specifically, suppose we imagine some of the targets are example queries, and other targets that look similar to these example queries are correct answers to the query, while other, dissimilar targets are incorrect answers. Given these examples, can we improve our retrieval performance?

Fortunately, there are several formalisms given in the literature that do exactly this. Many of them learn a weighting of the feature vector, while others learn a Mahalanobis distance. We experiment with seven such methods, described below:

Ranking Algorithms

Our first few learning algorithms are those that learn a weight for each element of the difference in feature vectors. If our learned weight vector is $\mathbf{w}$, then the "distance" from target $\mathbf{t}_i$ to query $\mathbf{q}_i$ is $\langle \mathbf{w}, d(\mathbf{t}_i, \mathbf{q}) \rangle$. This means that every weight in the weight vector corresponds to a dimension of the feature vectors. If the weight of a given dimension is higher, this means that the learning algorithm has decided that this dimension is "more discriminative" or "more meaningful" than the others.

Note that the distance from query to target in this formalism is no longer a proper distance at all, as it can be negative if the weight vector has sufficiently negative weight values. This matters little to us in this application; it simply means that the targets with negative distances will come first in our ordering.

We now briefly review some of the methods for learning this weight vector.

AdaRank AdaRank (Xu and Li 2007) is a ranking version of the famous AdaBoost (Freund and Schapire 1995) algorithm. The idea is essentially the same: We maintain a distribution over all of our example queries. We then construct some "weak" weight vector based on the current distribution. This vector will perform well on some queries (that is, rank most similar targets ahead of most dissimilar targets) and poorly on others. The distribution is updated to give more weight to queries on which the weight vector performs poorly. We add the weak vector to our collection of vectors, weighting it according to its overall performance. We then iterate the process, constructing a new weight vector based on the new distribution over queries.

The question is then how to construct a weak weight vector given a distribution over queries. We will follow (Xu and Li 2007), considering at each iteration all weight vectors with a "1" in one element of the vector and a "0" for all other elements. That is, each weight vector ranks the targets based on a single dimension. We choose, at each iteration, the one that performs best on the collection of sample queries according to weighted mean average precision, meaning the weighted average of the average precision of the sample queries, where the weights are defined by the distribution over queries.

SVM-Rank and SVM-MAP Another way to find an appropriate weight vector is to use the formalism of support vector machines. Suppose that, for our set of sample query images, $\mathcal{Q}$, we have a set of "similar" target images $\mathcal{S}$ and a set of "dissimilar" target images $\mathcal{D}$, so that each set is populated with pairs of images, and each pair contains a query and a target, which is either similar or dissimilar if the pair is in $\mathcal{S}$ or $\mathcal{D}$, respectively. We can then use standard SVM optimization techniques to find a solution to the following optimization:

$$\begin{aligned}&\min \frac{1}{2} + C \sum \xi_{i,j,k} \\ &\mathrm{s.\,t.:} \forall \mathbf{t}_i, \mathbf{t}_j, \mathbf{q}_k \in \mathcal{Q}; (\mathbf{q}_k, \mathbf{t}_i) \in \mathcal{S}; (\mathbf{q}_k, \mathbf{t}_j) \in \mathcal{D}: \\ &\quad \langle \mathbf{w}, |d(\mathbf{t}_j, \mathbf{q}_k)| \rangle > \langle \mathbf{w}, |d(\mathbf{t}_j, \mathbf{q}_k)| \rangle + 1 - \xi_{i,j,k}, \\ &\quad \xi_{i,j,k} \geq 0,\end{aligned}$$

That is, we solve for the $\mathbf{w}$ that makes all pairs in $\mathcal{S}$ for each query have smaller distance apart than those pairs in $\mathcal{D}$. We attempt to do this with some margin, but add in slack variables in case this cannot be accomplished, as is done in (Joachims 2002).

With some significant tweaking (Yue et al. 2007), we can phrase a version of this optimization that optimizes mean average precision directly, rather than just the overall accuracy of the ranking. We will try this version of the optimization in our experiments as well.

Metric Learning Algorithms

There is another family of algorithms that learn a *Mahalanobis distance,* rather than a simple weighting of the feature vector. These distances essentially learn a linear transformation of the points in the feature space, and so learn an $n \times n$ matrix, where n is the dimensionality of the feature space. The distance between two points in the new feature space is then $\sqrt{d(\mathbf{t}_i, \mathbf{q})^\top \mathbf{A} d(\mathbf{t}_i, \mathbf{q})}$ where $\mathbf{A}$ is the learned matrix. We again seek to learn a matrix that brings together the similar pairs in the set $\mathcal{S}$ and separates the dissimilar pairs in the set $\mathcal{D}$. We now describe a couple of different schools of thought on how this might be done.

Relevance Component Analysis (RCA) and Discriminative Component Analysis (DCA) Relevance Component Analysis (Shental et al. 2002) and Discriminative Component Analysis (Hoi et al. 2006) work on essentially the same principle as linear discriminant analysis: We use the training data to estimate the covariance matrices of the similar and dissimilar sets, then use these directly to construct our Mahalanobis matrix. In the case of RCA, we estimate only the covariance matrix of the set of similar pairs, and the Mahalanobis matrix is the inversion of this covariance matrix:

$$\begin{aligned}\mathbf{C}_{\mathcal{S}} &= \frac{1}{|\mathcal{S}|} \sum_{(\mathbf{t},\mathbf{q}) \in \mathcal{S}} d(\mathbf{t}, \mathbf{q}) d(\mathbf{t}, \mathbf{q})^\top \\ \mathbf{A}_{\mathrm{RCA}} &= \mathbf{C}_{\mathcal{S}}^{-1}\end{aligned}$$

So, intuitively, $\mathbf{A}$ shrinks the space in directions where the variance among similar pairs in the highest. In DCA, we do the same thing, except we also estimate the covariance of the dissimilar pairs:

$$\begin{aligned}\mathbf{C}_{\mathcal{D}} &= \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{t},\mathbf{q}) \in \mathcal{D}} d(\mathbf{t}, \mathbf{q}) d(\mathbf{t}, \mathbf{q})^\top \\ \mathbf{A}_{\mathrm{DCA}} &= \mathbf{C}_{\mathcal{S}}^{-1} \mathbf{C}_{\mathcal{D}}\end{aligned}$$

so that the space is also expanded in the directions where variance among dissimilar pairs is the smallest. There are strong theoretical reasons for these steps, and important issues to consider when computing matrix inversions. The reader is encouraged to consult (Hoi et al. 2006) for more information.

Xing's Method and Large Margin Nearest Neighbor (LMNN) Xing's method (Xing et al. 2002) and the Large Margin Nearest Neighbor classifier (Weinberger, Blitzer, and Saul 2006) both solve the Mahalanobis distance learning problem via an optimization. To simplify our notation, we follow (Xing et al. 2002) and define

$$||d(\mathbf{x}, \mathbf{y})||_{\mathbf{A}} = \sqrt{d(\mathbf{x}, \mathbf{y})^\top \mathbf{A} d(\mathbf{x}, \mathbf{y})}$$

to be the Mahalanobis norm of the vector $d(\mathbf{x}, \mathbf{y})$, and similarly, for the squared Mahalanobis norm $||d(\mathbf{x}, \mathbf{y})||_{\mathbf{A}}^2$. Using this notation, we can state Xing's optimization as follows:

$$\begin{aligned}&\min_{\mathbf{A}} \sum_{(\mathbf{t},\mathbf{q}) \in \mathcal{S}} ||d(\mathbf{t}, \mathbf{q})||_{\mathbf{A}}^2 \\ &\mathrm{s.\,t.:} \sum_{(\mathbf{t},\mathbf{q}) \in \mathcal{D}} ||d(\mathbf{t}, \mathbf{q})||_{\mathbf{A}} > 1, \\ &\quad \mathbf{A} \succeq 0\end{aligned}$$

The intuition is fairly simple: The optimization attempts to find $\mathbf{A}$ so that the distance between similar pairs is as small as possible, whereas the sum of the differences between dissimilar pairs is held above some arbitrary constant (in this case, 1).

The optimization for LMNN is slightly more involved. It blends the large margin ideas in the ranking SVM with Xing's method, requiring not simply that the sum of the differences be large, but that all similar targets for a given query be closer than all dissimilar targets by some margin:

$$\begin{aligned}&\min_{\mathbf{A}} \sum_{(\mathbf{t},\mathbf{q}) \in \mathcal{S}} ||d(\mathbf{t}, \mathbf{q})||_{\mathbf{A}}^2 + C \sum \xi_{i,j,k} \\ &\mathrm{s.\,t.:} \forall \mathbf{t}_i, \mathbf{t}_j, \mathbf{q}_k \in \mathcal{Q}; (\mathbf{q}_k, \mathbf{t}_i) \in \mathcal{S}; (\mathbf{q}_k, \mathbf{t}_j) \in \mathcal{D}: \\ &\quad ||d(\mathbf{t}_j, \mathbf{q}_k)||_{\mathbf{A}}^2 - ||d(\mathbf{t}_i, \mathbf{q}_k)||_{\mathbf{A}}^2 \geq 1 - \xi_{i,j,k}, \\ &\quad \xi_{i,j,k} \geq 0, \\ &\quad \mathbf{A} \succeq 0\end{aligned}$$

Both optimizations have advantages and disadvantages, as discussed in the original papers. Details on how to solve these optimizations are also discussed in the original papers, and we only mention here that both are solved via the method of *alternating projections* (Bauschke and Borwein 1996). We will try both in the experiments below.

	All L_1, L_2	**Histogram**	**Visual Vocab.**	**Band α, β**	**Band L_1, L_2**	**Band Hist.**	**Band VV**
Euclid	0.0389	0.3065	*0.2413*	0.1747	0.5300	*0.0283*	0.0235
Rank SVM		0.3150	0.0551	0.3172	0.4921	0.0023	0.0041
SVM-MAP		0.3040	*0.2404*	0.3830	0.5605	0.025	*0.0245*
AdaRank		0.2129	0.0874	0.1639	0.3989	0.0191	0.014
Xing		0.0663	0.0561	0.1703	0.2337	0.0107	0.0105
LMNN		0.3007	0.2188	*0.6069*	*0.7372*	0.0252	0.0191
RCA		*0.3287*	**0.2565**	**0.6366**	**0.7479**	0.0260	0.0221
DCA		**0.3439**	0.2231	0.3790	0.5294	**0.0307**	**0.0265**

Table 1: Results of the paper retrieval experiment, in terms of leave-one-out mean average precision over our sample dataset

Experiment and Results

We now perform experiments to find out which combination of feature set and learning algorithm are the most effective in this application. We use a set of high magnification photographic images collected by the second author for these experiments. These images are collected in laboratory-like conditions. Specifically, images are acquired using an Infinity 2-3 imager manufactured by the Lumenera Corporation. The camera incorporates an Interline Sony ICX262 3.3 megapixel color progressive scan CCD sensor, producing images incorporating 2080×1536, 3.45µm square pixels. The imager is attached to an Edmund Optics VZM 200i lens. Samples are illuminated using a 3"LED line light. The light is placed at a 15° raking angle to the surface of the photographic paper.

Using the image processing program *ImageJ* (Rasband 2006), image files are converted to 16-bit grayscale and cropped to 1024×1024 pixels. Slight edge-to-edge variations in exposure are reduced through flat field correction using a bandpass filter. The images are also sharpened slightly using an unsharp mask. Contrast is enhanced by equalizing the histogram. The image files are then saved as TIFFs. The resulting images cover 0.667 cm^2 of the surface of the photographic paper.

The resulting collection contains about 2050 images. We split each image into quarters for our experiments, giving us a database of 8200 targets. To simulate a query, we feed the algorithm a single quarter-image. We consider the "similar images" to be the other three quarters of the original image, and all other images to be dissimilar. For training, we use 100 images from the original set (or 400 quarter-images) and the rest for testing. We use visual inspection to verify that we are matching for visual similarity and not simply learning which quarter-images came from the same original. This is standard practice in the literature (Do and Vetterli 2002).

We measure the effectiveness of our system using mean average precision (Yue et al. 2007), where we try each of the 7800 test images as a query, compute the average precision of the query, and average the results over all 7800 images.

Table 1 shows the results of learning over all experiments and feature vectors discussed. The best algorithm for each feature set is shown in **boldface**, and the second best in *italics*. Performance differences of > 0.005 are statistically significant. In the final two columns, differences of > 0.001 are also statistically significant. We provide results from the first feature set, L_1 and L_2 norms of the entire image, as a reference point only; no metric learning is done on these features.

Among learning algorithms, RCA appears to be successful over a wide variety of features. LMNN is also very successful, and DCA is successful on a number of feature sets but not all. Xing's method has trouble with these datasets. Weighted linear models do not do as well as the full Mahalanobis distances. AdaRank in particular does poorly. SVM-MAP is the best of the group, always managing to find a weighting that is marginally useful or at least not worse than Euclidean distance. This is expected as SVM-MAP optimizes mean average precision directly, unlike all of the other methods.

Among feature sets, the wavelet-based feature sets, with two-value summaries for each band, are clear winners, as predicted by previous experiments (Do and Vetterli 2002). More than two values per band proves to be too fine of a granularity, whereas not using the wavelet decomposition at all clearly misses information. Importantly, we note the improvement that metric learning brings to the α, β feature set: Euclidean distance in this feature space performs worse even than the histogram or visual vocabulary features. After metric learning is applied, this feature set is second only to the L_1 and L_2 norms of the wavelet sub-bands.

In Figure 3 we see two sample queries, with the top two returned images for each one. Note that in this figure, the top two images are not quarter-images taken from the same original; these are the top two results from *different* originals.

Conclusion

We have here presented a challenging problem in information retrieval, that of matching high magnification images of photographic papers for art print authentication. We have shown that this problem can be solved effectively with a combination of feature extraction techniques from image processing and metric learning techniques from machine learning research. We showed that, on a real world database of photographic papers, we are able to achieve a mean average precision of 0.7479. More simply, an average of three out of the top four targets returned per query are relevant, an impressive feat on a test database of 7800 target images with only three relevant targets per query. Visual inspection shows that the system's notion of visual similarity closely matches the human notion.

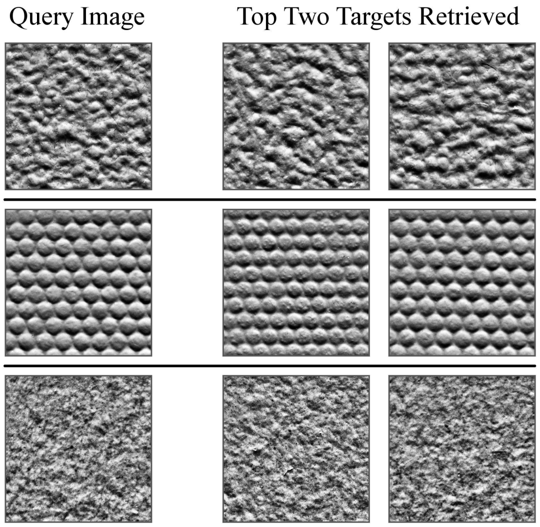

Figure 3: Examples of queries with similar targets returned by the constructed system.

We highlight two important messages in our results: First, as shown in previous work (Do and Vetterli 2002) wavelet-based feature functions are the best choice for this application, performing better than gray-level histograms and simple visual vocabularies, but the method of summarization of the wavelet sub-bands is important. Using too many values to summarize, such as gray-level histograms for each sub-band, overwhelms the system with irrelevant features. Fitting a Gaussian to the coefficients of each sub-band, either generalized or not, appears to be a good choice.

Second, learning a full-rank Mahalanobis matrix, using metric learning, nearly always outperforms learning a simple weight vector, but again the choice of learning algorithm is important. RCA, DCA, and LMNN all seem to do well under certain feature sets. If a linear weight vector must be learned (e.g., in applications where the feature space is large or speed is a priority), SVM-MAP is able to optimize its target criteria (mean average precision) more effectively than other linear learning methods.

Currently, we are attempting to combine several of our best performing models into a single, better model. We have experimented with different types of model fusion, and results so far are promising, giving mean average precision of 0.8 in early tests.

Finally, those in the image processing area may object to using the mean (or L_1 norm) of all pixel values in the final smoothed image as a feature for our model, as this feature could be drastically influenced by light level in the surrounding area. While this is true, our images are taken in conditions where the external light level is highly controlled. In this case, the L_1 norm becomes a useful, non-misleading feature. In cases where light levels are more variable, the α, β feature set may be a better choice.

References

Aha, D. W.; Kibler, D.; and Albert, M. K. 1991. Instance-based learning algorithms. *Machine Learning* 6(1):37–66.

Akansu, A. N., and Haddad, P. R. 1992. *Multiresolution Signal Decomposition: Transforms, Subbands, Wavelets*. Academic Press.

Bauschke, H. H., and Borwein, J. M. 1996. On projection algorithms for solving convex feasibility problems. *SIAM Review* 38:367–426.

Do, M. N., and Vetterli, M. 2002. Wavelet-based texture retrieval using generalized gaussian density and kullback-leibler distance. *IEEE Transactions on Image Processing* 11:146–158.

Freund, Y., and Schapire, R. E. 1995. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, 23–37.

Fukunaga, K. 1990. *Statistical Pattern Recognition*. Elsevier.

Hoi, S. C.; Liu, W.; Lyu, M. R.; and Ma, W.-Y. 2006. Learning distance metrics with contextual constraints for image retrieval. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 2072–2078.

Jain, A. K., and Dubes, R. C. 1988. *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Joachims, T. 2002. Optimizing search engines using clickthrough data. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)*, 133–142.

Li, J., and Wang, J. Z. 2003. Automatic linguistic indexing of pictures by a statistical modeling approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25:1075–1088.

Rasband, W. 2006. ImageJ: Image processing and analysis in java. `http://rsbweb.nih.gov/ij/`. Developed at the National Institute of Health in Bethesda, MD, USA.

Shental, N.; Hertz, T.; Weinshall, D.; and Pavel, M. 2002. Adjustment learning and relevant component analysis. In *European Conference on Computer Vision (ECCV)*, 776–792.

Weinberger, K. Q.; Blitzer, J.; and Saul, L. K. 2006. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems (NIPS)*, 1473–1480.

Xing, E.; Ng, A.; Jordan, M.; and Russell, S. 2002. Distance metric learning with application to clustering with side-information. In *Advances in Neural Information Processing Systems (NIPS)*.

Xu, J., and Li, H. 2007. Adarank: a boosting algorithm for information retrieval. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 391–398.

Yue, Y.; Finley, T.; Radlinski, F.; and Joachims, T. 2007. A support vector method for optimizing average precision. In *ACM Conference on Research and Development in Information Retrieval (SIGIR)*, 271–278.