MATH 521: GROUP FINAL PROJECT REPORT: IS IT A CAT OR A DOG?

STEVE BROWN, NEN HUYNH, JODY PRITCHETT

Abstract. This report documents the methods we used to process the cat/dog images and build classifiers for the Spring 2012 group final project for Math 521.

1. Introduction

The assignment for this project was to design and build classifiers that would effectively classify images of cats and dogs. One hundred sixty images, 80 cats and 80 dogs, were provided to the group as a training set. Each image was a 64×64 resolution gray scale image. Each team member chose a different strategy for a classifier, and experimented with the training set to determine its effectiveness. This was done by choosing a set of images (around 20% of the total) as a test set, training the classifier on the remaining images, and then computing the accuracy of the classifier using a "confusion matrix." The basic strategy followed by each team member was:

- Transform the raw image data to features of lower dimension with minimal loss of information
- Employ a technique for classifying the images based on these features

Once the classifiers were judged to be reasonably accurate using the training data, each was retrained on the complete set of 160 images, and then used to classify 38 new (19 cats and 19 dogs). In the next sections, we give the details of each classifier and its accuracy results.

2. Steve Brown: PCA followed by Kohonen Novelty Based Classification

2.1. Introduction.

The purpose of this section is to explain the classification of images of cats and dogs by using the PCA method.

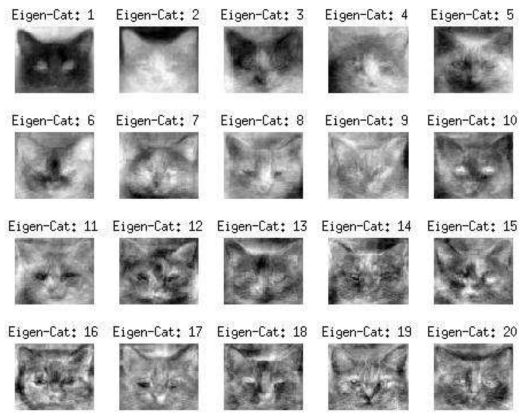
Principal component analysis (PCA) is a mathematical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components. (http://en.wikipedia.org/wiki/Principal_component_analysis)

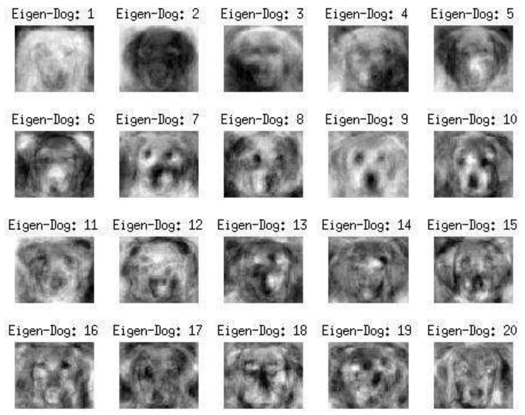
2.2. The Procedure.

The procedure was written in MATLAB. Here are the steps that were taken to classify cats and dogs using PCA:

- Loaded ensembles of training images of cats and of dogs.
- Mean-averaged both training ensembles.
- Called `svd` to get a training basis for cats and of dogs.
- Retained 95% of the cumulative energy in each ensemble.
- Loaded ensembles of testing images of cats and of dogs.
- Mean average both testing ensembles.
- For each test image:
  - Projected the test image onto the training basis for cats and the training basis for dogs.
  - Took the two norm of both projections.
  - If the cat-basis norm was larger than the dog-basis norm then classified the test image as a cat, else as a dog.
- Computed success rates.

2.3. **The Training Set Eigen-Pets.** To keep the percentage of cumulative energy in each training set, a little more than forty dimensions out of the original eighty were required for each basis. Here are some of the eigen-cats and eigen-dogs.

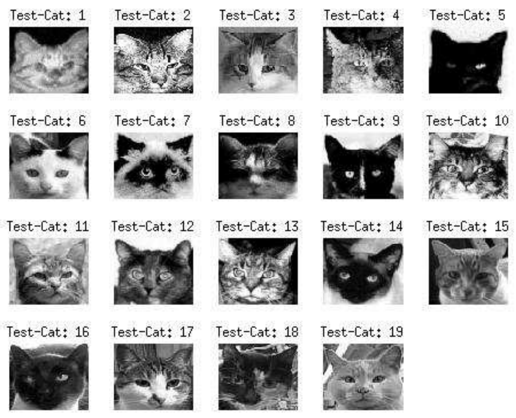


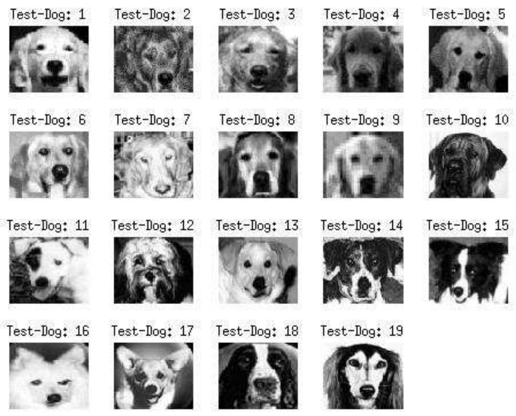


As human beings we are experts at classifying cats and dogs. Cats have perky ears while dogs generally have floppy ears. Cats have small snouts while dogs have elongated snouts. Cats have round faces while dogs are rather oval. Although the image don't show it, cats are generally smaller and fluffier that are dogs.

One might think that there is sufficient difference between cats and dogs that it should be easy to tell them apart. For us it is; for a computer, not quite so easy.

2.4. **The Test-Pets.** Here are the cats and dogs that were to be classified.





As human beings we are experts at classifying cats and dogs. Cats have perky ears while dogs generally have floppy ears. Cats have small snouts while dogs have elongated snouts. Cats have round faces while dogs are rather oval. Although the image don't show it, cats are generally smaller and fluffier that are dogs.

One might think that there is sufficient difference between cats and dogs that it should be easy to tell them apart. For us it is; for a computer, not quite so easy.

2.5. **The Results.** Here is a table showing the results of the classifications.

Pet	#Tested	IsCat?	isDog?	Success Ratio
Cat	19	16	3	84.42%
Dog	19	2	17	89.47%

Overall, the results were good but there is room for improvement. It appears that computers are not as good as we are at classifying cats and dogs, or at least when using the PCA method.

2.6. **The Misfits.** Here are the cats and dogs that were classified incorrectly.

Perhaps the two dogs were mistaken for cats because they are two of only four dogs that have perky ears while all the cats have perky ears.

As for why the three cats were taken to be dogs, it's a mystery. Perhaps the computer knows something that we don't?

3. Nen Huynh: Wavelet PCA Novelty Filter (WPCAN)

When classifying cats and dogs, the overall outline of an animal's face is one of the most distinguishing features of the two creatures. The fur color, though the most distinguishing feature the two creatures, does not classify the animals well. For example, a correlation between dogs may misclassify the brighter images as cats just because it is brighter. Still, the texture of the fur color is useful in distinguishing the two creatures.

Wavelets allow us to pick up the outline of the faces and, if we remove one of the wavelet layer of coefficients (which we will go in to detail), still disregard the fur color. Also, the coefficients specify the types of texture of the fur color. This texture feature facilitates the classification because the fur textures of the cats are fairly distinct from dogs. Thus, wavelets are appropriate to extract these features.

3.1. Wavelets.

The theory of wavelets provides us with a way to decompose an image using compact support wave-like functions. This set of wave-like functions form a basis for a vector space of continuous functions and are called scaling functions $\{\phi_i\}$ and wavelet functions $\{\psi_i\}$. Given a function $f(x)$, a set of scaling functions $\{\phi_i\}$, and a set of wavelet functions $\{\psi_i\}$, we can decompose it to:

$$f(x) \approx \sum_{i=1}^{N} c_i \phi_i(x) + \sum_{i=1}^{N} d_i \psi_i(x).$$

For an image $I(x, y)$ with one set of scaling functions $\{\phi_i\}$ and two sets of wavelets $\{\psi_i^H\}$, $\{\psi_i^V\}$, $\{\psi_i^D\}$ such that their union is also a basis for continuous functions. Thus an image is decomposed into

$$I(x,y) \approx \sum_{i=1}^{M}\sum_{j=1}^{N} c_{i,j}\phi_{i,j}(x,y) + \sum_{i=1}^{M}\sum_{j=1}^{N} d_{i,j}^{H}\phi_{i,j}^{H}(x,y) + \sum_{i=1}^{M}\sum_{j=1}^{N} d_{i,j}^{V}\phi_{i,j}^{V}(x,y) + \sum_{i=1}^{M}\sum_{j=1}^{N} d_{i,j}^{D}\phi_{i,j}^{D}(x,y).$$

We can decompose the $c_{i,j}$ coefficients by wavelets and do it again. This gives us a 2 level decomposition:

$$\begin{aligned} I(x,y) \approx & \sum_{i=1}^{M}\sum_{j=1}^{N} c^3_{i,j}\phi_{i,j}(x,y) + \sum_{i=1}^{M}\sum_{j=1}^{N} d^{H,1}_{i,j}\psi^{H,1}_{i,j}(x,y) + \sum_{i=1}^{M}\sum_{j=1}^{N} d^{V,1}_{i,j}\psi^{V,1}_{i,j}(x,y) \\ & + \sum_{i=1}^{M}\sum_{j=1}^{N} d^{D,1}_{i,j}\psi^{D,1}_{i,j}(x,y) + \sum_{i=1}^{M}\sum_{j=1}^{N} d^{V,2}_{i,j}\psi^{V,2}_{i,j}(x,y) + \sum_{i=1}^{M}\sum_{j=1}^{N} d^{D,2}_{i,j}\psi^{D,2}_{i,j}. \end{aligned}$$

There are many types of wavelets, Haar wavelets have the horizontal, vertical, and diagonal coefficients $\{d_{i,j}\}$,from the three level decomposition, contain the outline and texture information more than other types of wavelets. But because the faces have small and large scale outlines and texture information, the haar wavelet is evaluated in three levels. Figure 3.1 shows an example of the haar wavelet coefficients for a cat image. The image size of cats and dogs are relatively small so four or more levels of decomposition do not provide useful information so we only consider 2 levels.

The $c^2_{m,n}$ level contains mostly fur (and background) color, the $c^2_{m,n}$ coefficients will essentially be useless in classification. Thus we do not use these coefficients. Now that we have the outline and texture, we convert the $d_{i,j}$ coefficients into a column vector so that the features may be trained for our classification. Thus, for each image in the training set, we associate it to a vector from it's vertical, horizontal, and diagonal coefficients.

3.2. **Principle Component Analysis.** The wavelet coefficients vary dependently among each other. For example, a pixel on the outline of the animal face has neighbors that are likely to also be outlines. To get the salient features of a set of images from it's interdependency, we apply the principal component analysis (PCA). As an optional step, the PCA also reduces the dimension to decrease computation.

Principal component analysis takes a set of vectors $\{v_i\}_{i=1}^n$ of the same dimension and finds an optimal basis $\{\phi_k\}_{k=1}^n$. Orthonormal basis $\{\phi_k\}_{k=1}^n$ is optimal if given truncation number $D \le n$, $\|v_i - \sum_{k=1}^D a^i_k\phi_k\|$ is smallest over all vectors v_i of the above set and where a^i_k are coefficients of v_i for basis $\{\phi_k\}_{k=1}^n$. That is, optimal basis $\{\phi_k\}_{k=1}^n$ is a basis that gives iteratively over the dimensions the maximum information to the least information. Thus, an optimal situation is when truncation $D < n$ still gives a good approximation

$$v_i \approx \sum_{k=1}^{D} a^i_k\phi_k.$$

We can also think of this as mapping v_i to $(a^i_1, a^i_2, ..., a^i_D)$. So it's a projection P of v_i to a smaller dimensional subspace and we hope that a^i_k for $k \ge D$ is small in magnitude. If the vectors $\{v_i\}_{i=1}^n$ are similar meaning $\|v_i - v_j\|$ is small, then the optimal basis will give a small error $\|v_i - \sum_{k=1}^D a^i_k\phi_k\|$. This is equivalent to $\|v_i - P^TPv\|$ because $P^TPv = P^T\begin{bmatrix} a^i_1 & a^i_2 & ... & a^i_D\end{bmatrix}^T = \sum_{k=1}^D a^i_k\phi_k$.

To find the optimal basis, we use an important property. Given a finite set of m column vectors of the same dimension, we can concatenate these vectors column-wise to form a matrix A. The matrix A can be row demeaned to $\tilde{A}$; that is, let $b = [b_i]_i$ where $b_i = \frac{1}{m}\sum_{j=1}^m a_{ij}$ where a_{ij} are the elements of matrix A, and $\tilde{A} = [a_{ij} - b_i]_{ij}$. From the singular value decomposition $A = U\Sigma V$, the optimal basis for these column vectors is then the column vectors of U. And when truncation is performed, only the first D columns of U are kept. Thus, matrix P mentioned above is a matrix formed from the first D columns of matrix U and concatenated in order column-wise.

3.3. **Novelty Filter.** We are now ready to classify the unknown images. The process of applying PCA to the vectorized wavelet coefficients give us a projection matrix P that maps the vectorized wavelet coefficients X to the optimal subspace also provided by the PCA process. To map it back to

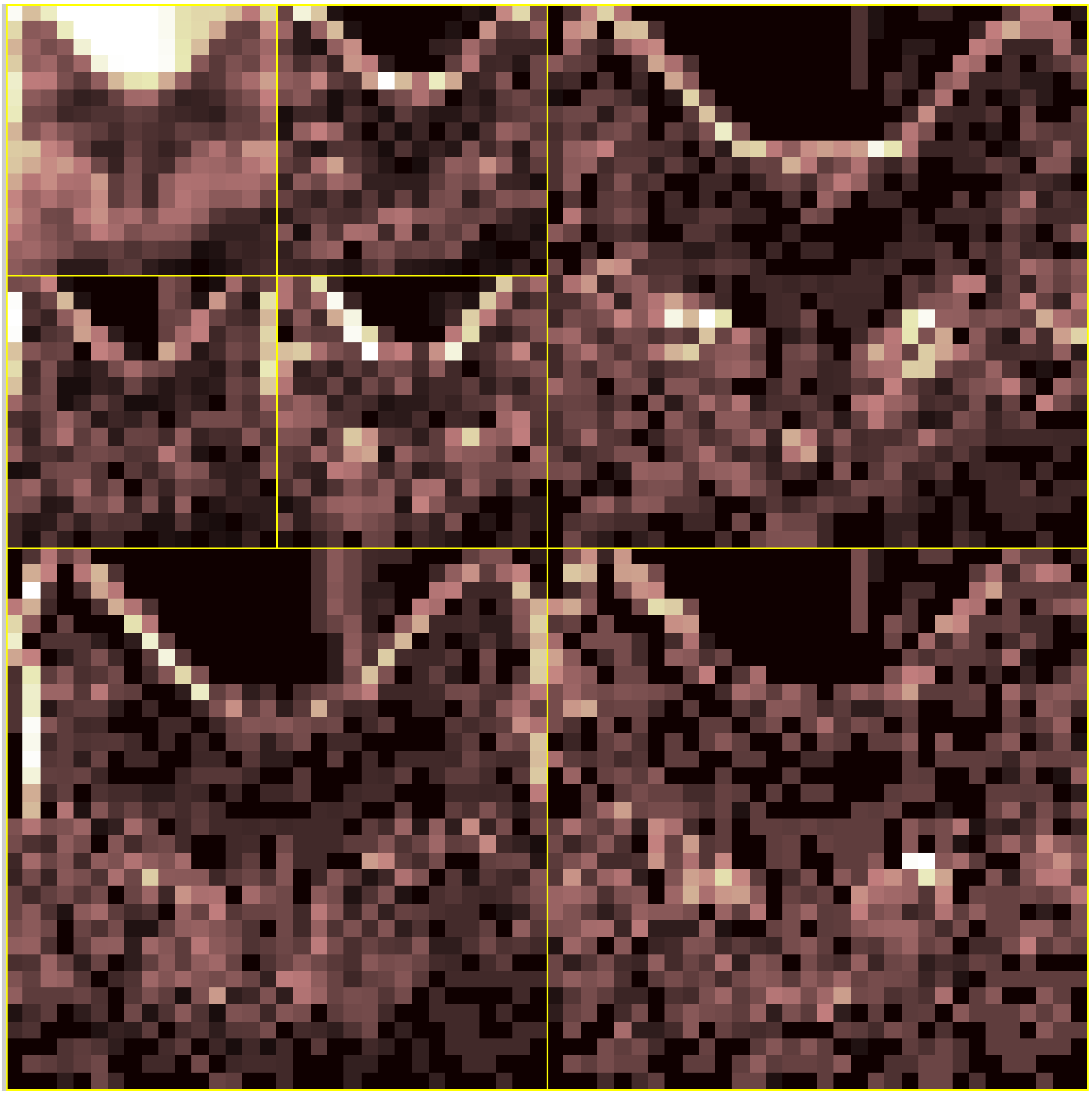


FIGURE 3.1. The top left box is the $c_{i,j}$ coefficients representation as color scale and the rest are the $d_{i,j}$ coefficients also as color scale.

the space of wavelet coefficients, we map using P^T. We note that P is not invertible so $P^T PX$, the mapping of the wavelet coefficient to the optimal subspace and back, does not give back the original wavelet coefficients. However, $P^T P$ gives us the best approximation of the wavelet coefficients X that is projected from the optimal subspace; that is, $\|X - P^T PX\|_2$ is small. Here, $X - P^T PX$ is the novelty of the image compared to a set of vectors that P is defined from.

The smaller $\|X-P^T PX\|_2$ is, the more PX belongs in this optimal subspace. Thus, if we have an optimal subspace for cats V_C and optimal subspace V_D from PCA and we have their corresponding projections P_C and P_D, we can calculate $R_C := \|X - P_C^T P_C X\|_2$ and $R_D := \|X - P_D^T P_D X\|_2$ to find a cat/dog label for wavelet coefficient vector X. If $R_C < R_D$, then we label X as a cat and if

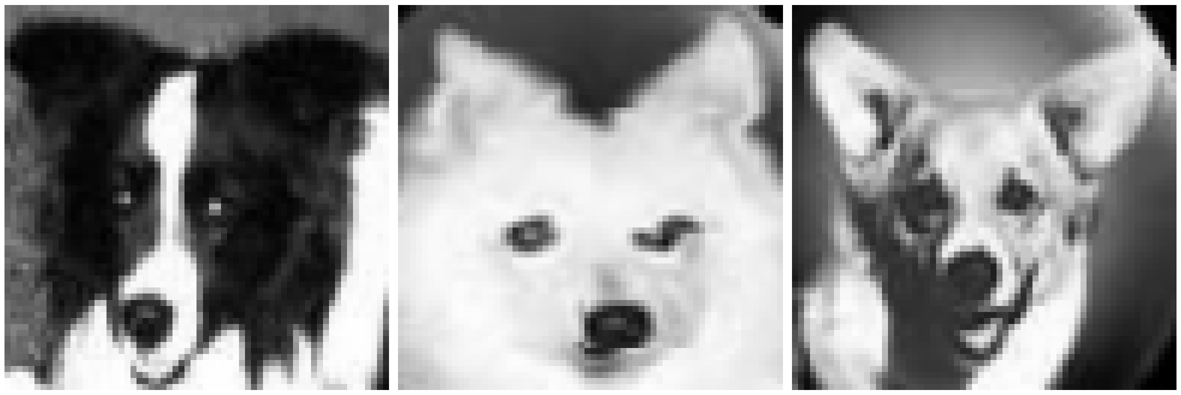


FIGURE 3.2. The dog images that were misclassified as cats.

$R_D < R_C$, then we label X as a dog. $R_C < R_D$ means that the projection $P_C X$ "fits" to the optimal subspace of dogs. Therefore, to classify an image I, we take 2 levels of Haar wavelet transformation of the image, and vectorize the vertical, horizontal, and diagonal coefficients X. From these wavelet coefficients we find R_C and R_D as defined as above.

3.4. **Testing.** Two tests are devised for testing this classification method. For the first, 80 images of cats and 80 images of dogs of dimension 64×64 are provided. 90% of cat and 90% of dog images randomly chosen are used to train the classifier. The rest are then used to attempt a classification of them. The number of misclassification is kept track as this process is repeated 10,000 times. The confusion matrix is then formed from the average misclassifications of cats and dogs. Below is the confusion matrix:

$$\left[\begin{array}{ccc} \underline{Confusionmatrix} & \text{Cats} & \text{Dogs} \\ \text{Classified as Cat} & 97.3\% & 0.8\% \\ \text{Classified as Dog} & 2.7\% & 99.2\% \end{array}\right].$$

We can see that cats are easier to distinguish than dogs. This is potentially due to higher diversity of faces of dogs compared to cats causing a larger number of overlaps with cat images.

The second test gives us an example of when the images fail. All of the 160 images of cats and dogs from above are used as training and 38 images not of this set are tested. The confusion matrix here is:

$$\left[\begin{array}{ccc} \underline{Confusionmatrix} & \text{Cats} & \text{Dogs} \\ \text{Classified as Cat} & 19/19 & 3/19 \\ \text{Classified as Dog} & 0/19 & 16/19 \end{array}\right].$$

So we have perfect recognition for cats and 84% accuracy for dogs, which is relatively low. The misclassification probably comes from the ears as the dogs with ears pointed up are the ones that were misclassified. Figure 3.2 shows the three dogs misclassified as cats.

3.5. **Analysis.** The novelty filter is best when the images have all the images have the same foreground and background (i.e. binary images). Here, the $d_{i,j}$ coefficients have similar in scale values so it is appropriate to use the novelty filter. PCA orders the salient features that are not apparent in a set of vectors. This gives us the important features that we can then use to classify. The Haar wavelet scale functions do not approximate edges on an image as well as others. This allows the $d_{i,j}$ coefficients to contain the edge and texture information used in this classification method.

There is one particularly strange characteristic; we take the wavelet transform of a set of images and get one novelty filter for each of the of the two classes. To classify, we find the novelty of the testing images compared to the two classes. But, as an analogy, this is like having people in one

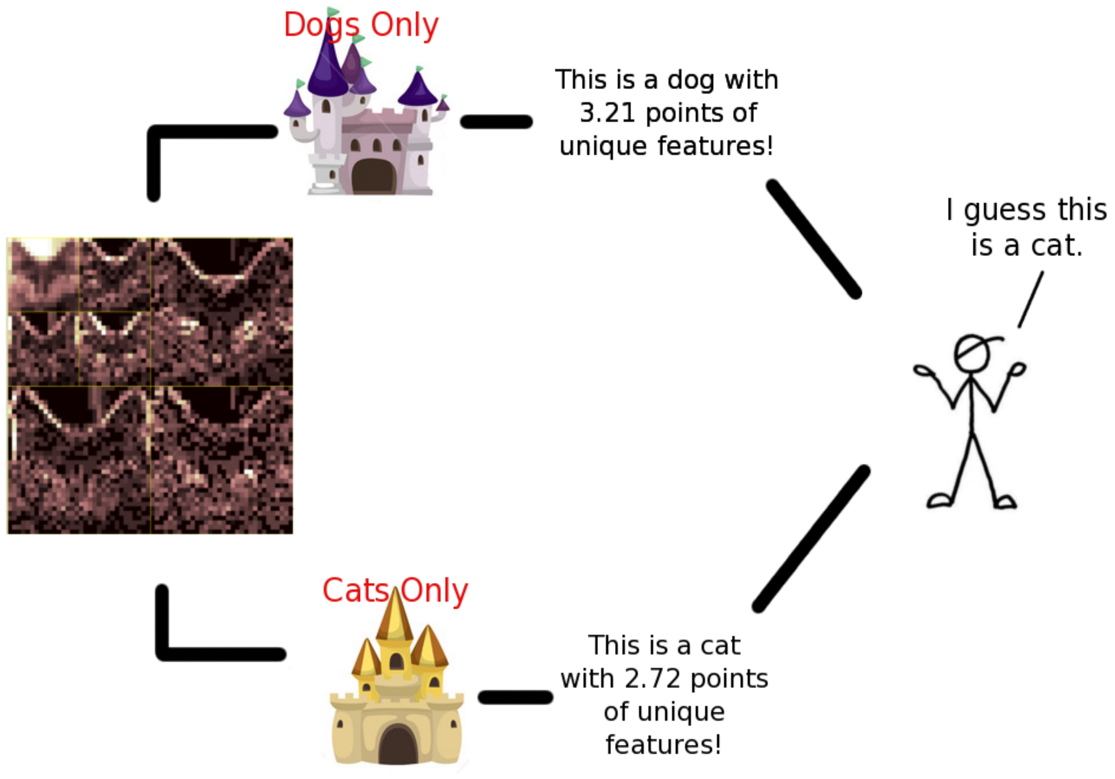

Dogs Only
This is a dog with 3.21 points of unique features!
I guess this is a cat.
Cats Only
This is a cat with 2.72 points of unique features!

FIGURE 3.3. A visual analog for the WPCAN.

castle having only seen cats and another castle having only seen dogs. So, given an image to each of the two castles, we determine the class just by their rank of how unique to them to image is to other images they've seen of their respective class. But no one in this model has ever seen both of the creatures, which is very strange.

3.6. **Attempted Improvement.** To improve this situation, we can weigh the two novelty filter values $R_C := \|X - P_C^T P_C X\|_2$ and $R_D := \|X - P_D^T P_D X\|_2$ instead of using $\min\{\|R_C\|, \|R_D\|\}$. We find a value $t \in [0, 1]$ such that using $\min\{t\|R_C\|, (1-t)\|R_D\|\}$ to reclassify the training set will give us the minimum number of misclassification of the training set.

Solving for t,

$$\text{We want } t < \frac{\|C_k - P_D P_D^T C_k\|}{\|C_k - P_C P_C^T C_k\| + \|C_k - P_D P_D^T C_k\|} =: t_k^C$$

meaning t small enough that as much cats from the training sets $\{C_k\}$ are identified as cats.

$$\text{We want } t > \frac{\|D_k - P_D P_D^T D_k\|}{\|D_k - P_C P_C^T D_k\| + \|D_k - P_D P_D^T D_k\|} =: t_k^D$$

meaning t large enough that as much dogs from the training sets $\{D_k\}$ are identified as dogs.

So we want to minimize the number of misclassifications,

$$M(t) = \sum_{k=1}^{m} \chi_{(-\infty, t_k^C]}(t) + \sum_{k=1}^{n} \chi_{[t_k^D, \infty)}(t).$$

This reduces to plugging values from the finite set $\{t_k^C\} \cup \{t_k^D\} \cup \{\frac{t_k^C + t_k^D}{2} | 1 \le i \le m, 1 \le j \le n\}$ to find the minimization $M(t)$. We note that without the improvement to find the best t, the original version before is equivalent to when $t = 1/2$.

This result is surprising in that we have $t = 1/2$ so the classification is the same without this improvement. But this is interesting and we have no idea why this is so.

4. Jody Pritchett: Prefilter the Images (Median, Laplacian) / PCA Followed by Self-Organizing Map / Learning Vector Quantization Based Classification

4.1. **Approach and Results.** My approach to extracting feature vectors was to convert the images to 4096×1 vectors, and perform a PCA to reduce the dimension. That is, I computed the KL expansion for each image vector in terms of the principle-component basis, using only the first D KL coefficients of the expansion as my feature vector. The value of D was chosen to include 95% of normalized energy (i.e. the so-called numerical rank of the data matrix). I tried this approach on three versions of the training images:

- raw images
- three-by-three median filtered images
- three-by-three laplacian filtered images

In my experiments, I picked 32 test images at random from the 160 original images, performed the feature extraction as described above, trained my classifier on the remaining 128 images, and examined the accuracy results. I used Kohonen's Learning Vector Quantization (LVQ) as my classification technique, which was implemented in a neural network architecture within Matlab's Neural Network Toolbox (I will describe LVQ in detail below). Because the training of the LVQ is slow and computationally intense, I experimented with only one test set at a time to see if prefiltering the images had any benefits. The numerical ranks for the data matrix for the raw, median filtered, and Laplacian filtered images were D=78, 65, and 131. In my experiments, the median filtering showed slightly better results than the raw data, and both of these seemed to be superior to the Laplacian filtering. Accuracies ranged from 84% to 100% for classifying cats as cats, and 63% to 84% for classifying dogs as dogs, with the median filtered-images tending to give results higher in these ranges than the other two methods, though not substantially different from the raw images. Therefore, I used both the raw images and the median filtered images to do the final classification of the 38 new images using the original 160 as the training set. Next, I discuss the details of the LVQ classification method due to Kohonen. These results will be shown at the end of this section.

An outstanding reference for the neural network implementation of the LVQ is the book [3]. This book provides much of the foundation of the Matlab neural network toolbox. Two additional references that were used in writing this section are [1] and [2]. The LVQ is a classifier based on unsupervised learning and supervised learning. It creates, iteratively, a small number (compared to the number of training vectors) of "protoype" vectors for each class (cat or dog) as a Self Organized Map (SOM). Throughout training, the prototypes evolve to represent the distribution of training vectors in each class. Once fully trained, the resulting prototypes are used to classify new inputs. The Matlab Neural Network implementation of the LVQ initializes the prototype vectors randomly, but these could also be initialized using other methods (e.g. using a separate SOM).

Figure 4.1 below shows the final SOM (i.e. the final collection of prototypes) after a sample run of the LVQ based on 300 iterations using 40 prototypes mapped to a 10 by 4 lattice. Remember that the training vectors here are the KL coefficients corresponding to the test images. Thus, the prototypes are actually prototype KL coefficients, and converted to these images by treating them as the coefficients in the KL expansion with respect to the basis for the training data computed using PCA. This map used the 160 training images of cats and dogs. I set the LVQ to organize 60% of the prototypes as dogs (i.e. 24 of the 40) and 40% as cats (i.e. 16 of the 40), because my first experiments showed that cats seemed much easier to distinguish than dogs, so my intuition was that dogs would need slightly more prototypes to represent them in the classifier.

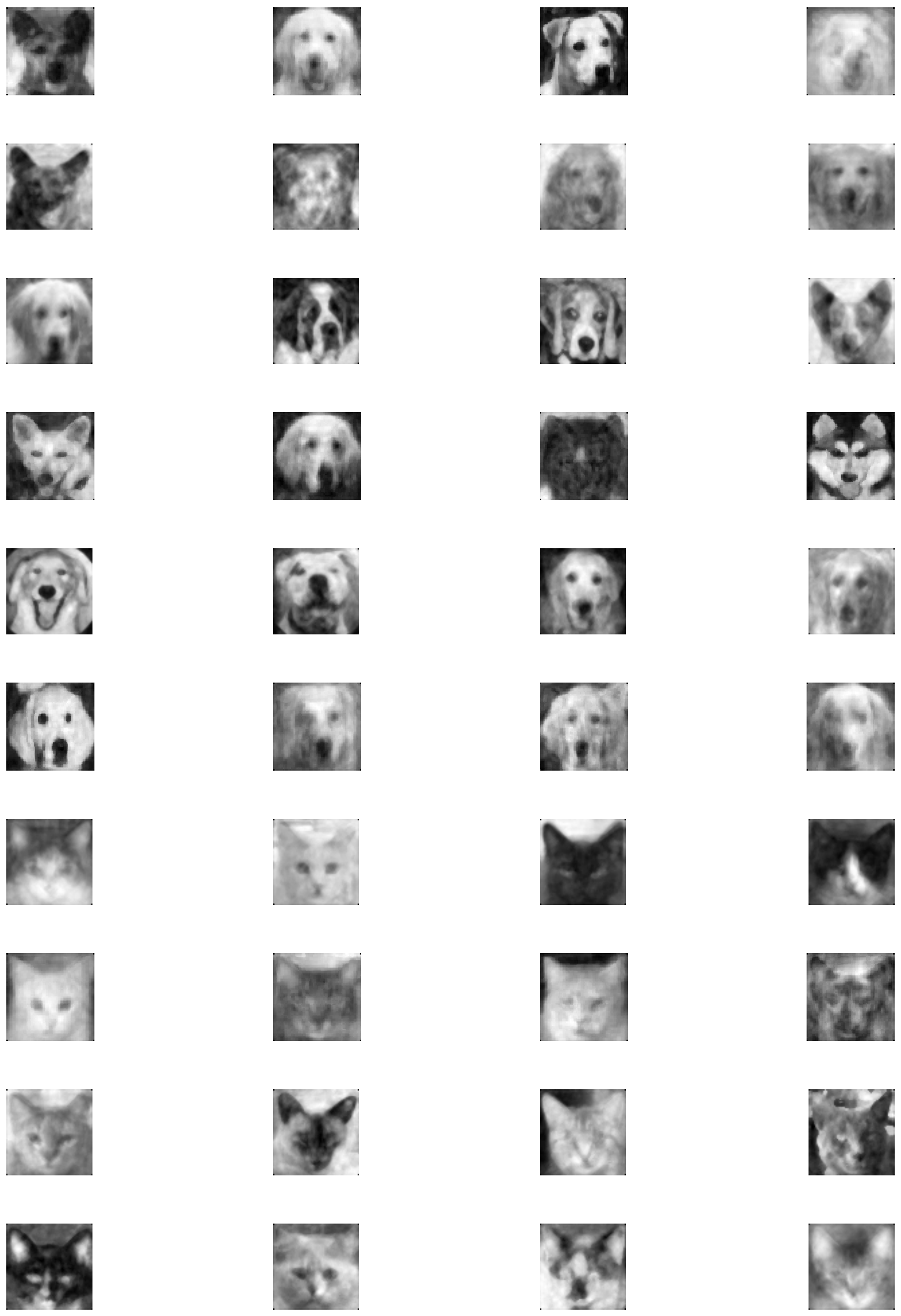


Figure 4.1. Self organized map on a 10 by 4 lattice for the 160 cat/dog training images. There are 24 prototype dogs, and 16 prototype cats.

The specifics of the Kohonen LVQ algorithm are described as follows for a two class problem.

(1) Set $q = 0$, and choose the initial prototypes for each of the two classes $c = 1, c = 2$: ${}_iw(q, c), i = 1, 2, ..., S_c; c = 1, 2$
(2) Choose a training point x_i at random with replacement, and let (j, c) be such that prototype ${}_jw(q, c)$ is closest in Euclidean norm to x_i.
    (a) If the classification of x_i is equal to c, then the update is:
$$ {}_jw(q + 1, c) = {}_jw(q, c) + \alpha\,(x_i - {}_jw(q, c)) $$
    That is, move the prototype closer to x_i .
    (b) If the classification of x_i is not equal to c, then the update is:
$$ {}_jw(q + 1, c) = {}_jw(q, c) - \alpha\,(x_i - {}_jw(q, c)) $$
    That is, move the prototype farther from x_i.
(3) Replace q with $q + 1$, decrease the learning rate $\alpha > 0$ towards 0, and repeat step 2
(4) Stop when there is insignificant change in the prototypes.

Once the LVQ is trained, a new vector x is classified into the same class as its nearest prototype.

I used the Matlab Neural Network Toolbox functions to train an LVQ on the 160 images, and test on the 38 new images. I did this twice: once using the raw images, followed by the PCA analysis (D=78 for the dimension of the KL feature vectors), and once using the median filtered images, followed by the PCA analysis (D=64 for the KL feature vectors). Figures 4.2 and 4.3 show the confusion matrix and misclassified images using the raw data images. Figures 4.4 and 4.5 show the confusion matrix and misclassified images using the median filtered images.

Prefiltering the images using the 3 by 3 median filter seems to improve performance of the LVQ classifier over using raw images. With the exception of one image (test dog number 16, the dog in the lower right corner of figure 4.5), there appears to be no obvious reason for the classifier to be confused over any of the images that were misclassified.

As final conclusions, I would say that given the available information (i.e. 2d low resolution gray scale images), the SOM/LVQ seems to work very well. I would speculate that with additional information available in the images, such as depth for example, the LVQ (and most of the methods we learned in this course) could be nearly perfect for distinguishing cats and dogs. However, the LVQ has several downsides, the most difficult to deal with being the selection of the number of prototypes, and the initialization of the prototypes. I searched the literature, and could find specific guidance as to how to select the number of prototypes. The number 40 (24 dogs and 16 cats), was arrived at by trial and error during the testing of the classifier (i.e. selecting test samples of size 32 from the 160 training cases, and training on the remaining 128), seeming to give the most consistent results classification. A full cross validation analysis, varying the size of the prototype classes, could potentially identify a more optimal choice. But this brings me to the next downside of the LVQ: the computation time. The neural network implementation of the LVQ runs so slowly as to make such a cross validation analysis impractical without a more powerful computing platform. Several literature sources suggested using a separate SOM to initialize the prototypes. This could potentially speed up training of the network, and would be an interesting follow-on study.

'Confusion matrix'	'Cats'	'Dogs'
'Classified as Cat'	[16]	[5]
'Classified as Dog'	[3]	[14]
'Total'	[19]	[19]

'Probibility'	'Cats'	'Dogs'
'Classified as Cat'	[0.8421]	[0.2632]
'Classified as Dog'	[0.1579]	[0.7368]
'Total'	[1]	[1]

Figure 4.2. Confusion matrix for LVQ applied to raw images reduced using PCA

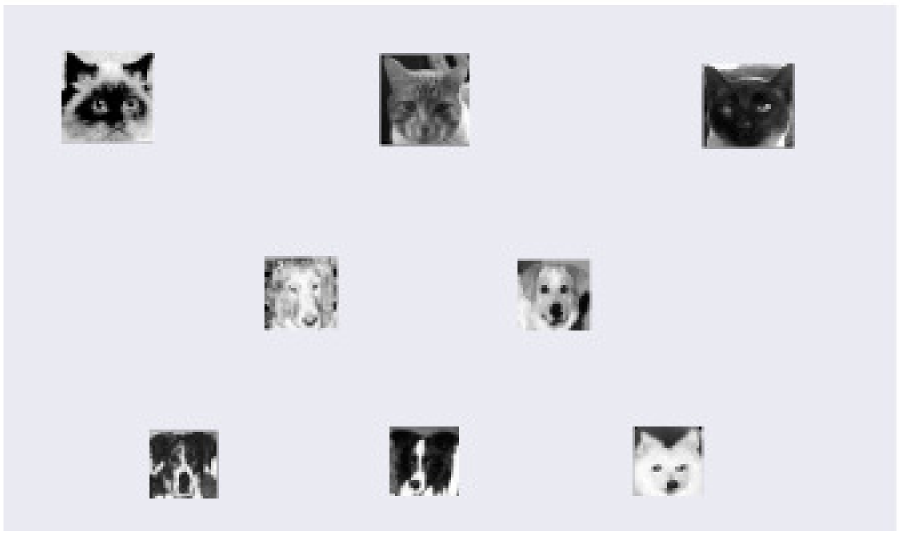
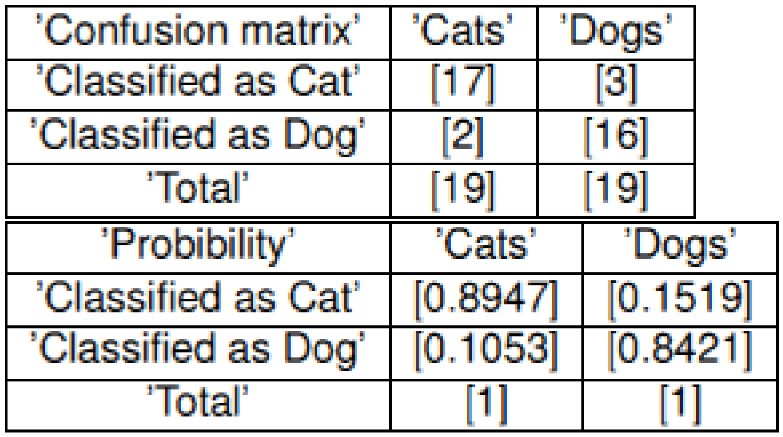


Figure 4.3. Misclassified images using raw images



'Confusion matrix'	'Cats'	'Dogs'
'Classified as Cat'	[17]	[3]
'Classified as Dog'	[2]	[16]
'Total'	[19]	[19]

'Probibility'	'Cats'	'Dogs'
'Classified as Cat'	[0.8947]	[0.1519]
'Classified as Dog'	[0.1053]	[0.8421]
'Total'	[1]	[1]

Figure 4.4. Confusion matrix for LVQ applied to median filtered images using PCA

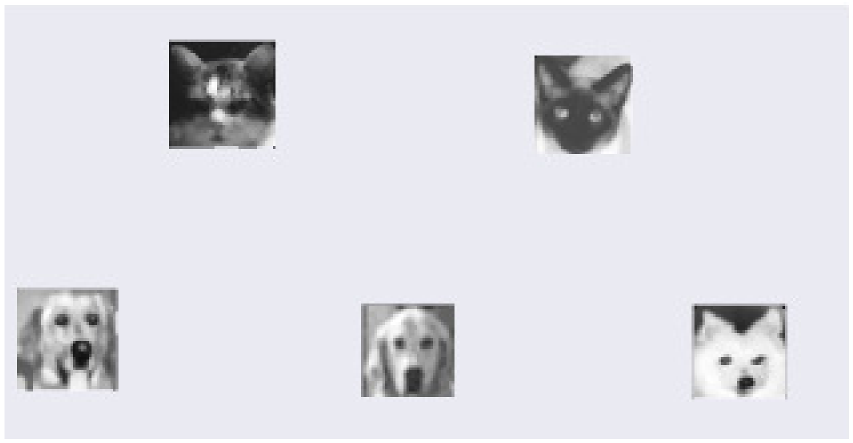


Figure 4.5. Misclassified images using median filtered images

5. Conclusions and observations

Our group tried three apparently different approaches to classifying the 38 test cases after training on the 160 separate images:

(1) Reduction of dimension of the training set by PCA for each class, with classification done by projecting the test case onto both bases, and selecting the class that gives the smallest L_2 norm of residuals (i.e. the Kohonen Novelty filter)
(2) 2D Wavelet Transform of the images, followed by PCA reduction of the resulting wavelet coefficients for each training class, with classification of test cases done via the Kohonen Novelty filter
(3) Prefiltering all the images using the 3 by 3 median filter, followed by reduction in dimension of the training set using PCA, training an LVQ to recognize the KL coefficients for

the two classes, with classification of a test case done by running the resulting LVQ in simulation mode (i.e. passing the test case forward through the network with no error back-propagation).

Over all, method 2 gave the fewest misclassifications (3 dogs were misclassified), and was the best classifier of cats (all 19 classified correctly). Methods 1 and 3 both had 5 total misclassifications. However, method 1 was the best over all at classifying dogs (only 2 dogs misclassified), and method 3 was better than method 1 at recognizing cats.

Examining the animals that were misclassified, we see that none of the methods could correctly classify test dog number 16, and methods 1 and 2 could not classify test dog 15 (see figure below). Two of the dogs that method 3 could not classify were not among the animals misclassified by methods 1 or 2. Finally, the third dog misclassified by method 2 was correctly classified by the other two methods. Therefore, if we pooled the three classifiers by majority rule, we would have misclassified 2 dogs.

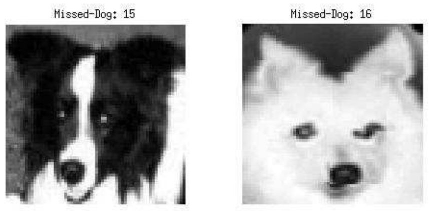


FIGURE 5.1. Misclassified dogs: No method could correctly classify test dog 16; both methods 1 and 2 could not correctly classify test dog 15. Pooling the three methods using a majority vote would have led to the dogs pictured here being misclassified.

Turning to cats, we see that none of the misclassified cats from methods 1 and 3 are common to both methods; method 1 misclassified test cats 16, 18, and 19, while method 3 misclassified test cats 8 and 14. Method 2 was perfect on cats, so combining the three methods in a majority vote would have still returned no misclassified cats.

In summary, by majority-voting the three methods, the result from the group effort would be:

Confusion Matrix	Cats	Dogs
Classified as Cat	19	2
Classified as Dog	0	17
Total	19	19

Finally, based on this limited study, it appears that relying on any one method for classification exposes the possibility of classification bias. What we see in our results is that each of our three methods has its own unique weaknesses. Majority voting seems to take advantage of this fact, by giving a final classifier that is better than any one of the individual methods. This may be a good principle to follow in general when faced with classification problems of this nature.

6. Matlab Scripts Used in Conducting the Analysis

6.1. **Steve Brown.**

```
function [] = Brown_S_Math521()
% Final project: Categorize images of cats and dogs using PCA method
%

%{
Output with 'nCount = 1':
   Success rates:
      Cats: 91.25
      Dogs: 82.50
========================================================================
=======
Student: Steve Brown, ID: 007713418
Class  : CSULB Spring 2012, Math-521 Matrix Meth in Data/Pattern, Jen-
Mei Chang
Title  : Final project
Due    : Th 2012-05-10
========================================================================
=======
%}
   % Define variables
   setPath = 'TIFFtraining/'; % Path of training sets
   nCount =  1;    % Number of pets to test at at time
   energy = 0.97;  % Cumulative energy level to retain

   % Find best basis for cats
   X = catEnsemble(setPath);
   numCats = size(X, 2);
   Uc = PCA(X, energy);

   % Find best basis for dogs
   Y = dogEnsemble(setPath);
   Ud = PCA(Y, energy);
   numDogs = size(Y, 2);

   % Test group of cats against rest of cats and all of dogs
   % Test group of dogs against rest of dogs and all of cats
   Rc = testProbes(X, Ud, nCount, energy);
   Rd = testProbes(Y, Uc, nCount, energy);

   %Compute success rates
   catRate = sum(Rc == 1) / numCats;
   dogRate = sum(Rd == 1) / numDogs;
   display('Success rates:')
   display(sprintf('   Cats: %2.2f', catRate * 100));
   display(sprintf('   Dogs: %2.2f', dogRate * 100));
end

%-----------------------------------------------------------------------
-------
%-----------------------------------------------------------------------
-------

% Project test probes onto ensemble and basis
% Return vector where each element is 1 if probe is closer to ensemble,
% else element is 2 meaning probe is closer to basis
function [R] = testEProbes(P, X, U, CE)
   % Find basis for ensemble
   B = PCA(X, CE);

   % Test each probe against both bases
   R = zeros(size(P, 2), 1);
   for p = 1:size(P, 2)
      R(p) = 1 + (norm(B'*P(:, p)) < norm(U'*P(:, p)));
   end
end

% Project specified number of probes at a time from ensemble onto
remaining
% probes in ensemble and on specified basic
% Return vector where each element is 1 if probe is closer to ensemble,
% else element is 2 meaning probe is closer to basis
function [R] = testProbes(X, U, n, CE)
   numProbes = size(X, 2);
   R = zeros(numProbes, 1);
   b = 0;
   while b < numProbes
      a = b + 1;
      b = b + n;
      if b > numProbes
         b = numProbes;
      end
      R(a:b) = testEProbes(X(:,a:b), X(:,[1:a-1 b+1:numProbes]), U, CE);
   end
end

%-----------------------------------------------------------------------
-------

function [U A D] = PCA(X, CE)
% Principal Component Analysis
%   Reduce ensemble to best D-subspace

% Input:  X  : Ensemble to find best D-subspace of
%         CE : cumulative energy as % to retain
% Output: U  : D-subspace basis
%         A  : Coefficient matrix
%         D  : Lowest D-value with cumulative energy

   % Remove mean from ensemble
   XX = X - repmat(mean(X,2), 1, size(X, 2));

   % Find D-value
   [U S V] = svd(XX, 0);
   L = diag(S).^2; %Lambda values
   CE_max = CE * sum(L);
   energy = 0;
   D = 0;
   while energy < CE_max
      D = D + 1;
      energy = energy + L(D);
   end

   % Find D-subspace basis and coefficient matrix
   U = U(:, 1:D);
   A = S(1:D, 1:D) * V(:, 1:D)';
end

%-----------------------------------------------------------------------
-------

% Create matrix where each column is a dog/cat image
function [X] = buildPetEnsemble(petType, path)
```

```
   % Search directory for pet images
   mask = sprintf('%s%s%%02d.tif', path, petType);
   pets = zeros(1, 100);
   for col = 0:99
      pets(col+1) = exist(sprintf(mask, col),'file');
   end
   pets = find(pets == 2) - 1;

   % Load pet images
   X = zeros(prod([64 64]), length(pets));
   for col = 1:length(pets)
       Img = imread(sprintf(mask, pets(col)));
       Img = Img(:,:,1); %To correct a bug in one of the files
       X(:, col) = Img(:);
   end
end

% Create matrix where each column is a cat image
function [X] = catEnsemble(path)
   X = buildPetEnsemble('Cat', path);
end

% Create matrix where each column is a dog image
function [X] = dogEnsemble(path)
   X = buildPetEnsemble('Dog', path);
end

%----------------------------------------------------------------------
-------

% % Display an image
% function [] = showImage(Img, vDots, hDots)
%    if 2 < nargin
%       Img = reshape(Img(:), vDots, hDots);
%    elseif 1 < nargin && 1 < numel(vDots)
%       Img = reshape(Img(:), vDots(1), vDots(2));
%    end
%    imagesc(Img);
%    axis off;
%    colormap(gray);
% end
%
% % Display a pet image
% function [] = showPet(ImgPet, n)
%    if 1 < nargin
%       if n < 0;  n = 1; end
%       if n <= size(ImgPet, 2);  n = size(ImgPet, 2);  end;
%       ImgPet = ImgPet(:, n);
%    end
%    showImage(ImgPet, [64 64]);
% end

%----------------------------------------------------------------------
-------
```

6.2. **Nen Huynh.**

```
%% Cross-validation test

cd('ENTER DIRECTORY CONTAINING TRAINING IMAGES');
Dir = dir('ENTER DIRECTORY CONTAINING TRAINING IMAGES/*.tif');

imcell = cell(1,numel(Dir));
for i = 1 : numel(Dir)
  imcell{i} = imread(Dir(i).name);
end

temp = imcell{end-1};
imcell{end-1} = temp(:,:,1);

N = 10000;

% 10% missing
withhold = 0.10;
Confuse = zeros(2,2);

for i = 1 : N
disp(i);

Perm1 = sort(randsample(80,80*(1-withhold)));
temp = true(80,1);
temp(Perm1) = false;
nPerm1 = find(temp);

Perm2 = sort(randsample(80,80*(1-withhold)));
temp = true(80,1);
temp(Perm2) = false;
nPerm2 = find(temp);

predict = myCatDogClassify([Cats(:,nPerm1) Dogs(:,nPerm2)],Cats
(:,Perm1),Dogs(:,Perm1),m,n)-1;

cpred = predict(1:size(predict,2)/2);
dpred = predict(size(predict,2)/2+1:end);

Confuse = Confuse + [sum(cpred == 0) sum(dpred == 0);...
                     sum(cpred == 1) sum(dpred == 1)];

if rem(i,10) == 0
    disp(Confuse/(8*i));
end
end

Confuse = { 'Confusion matrix',      'Cats'    ,     'Dogs'    ;...
           'Classified as Cat', Confuse(1,1)/N , Confuse(1,2)/N;...
           'Classified as Dog', Confuse(2,1)/N , Confuse(2,2)/N;...
                 'Total'      ,  size(cpred,2) , size(dpred,2)};

disp(Confuse);

disp(cell2mat(Confuse(2:3,2:3))/(80*(1-withhold)));


%% Test on separate Images
m = 64;
n = 64;

Dir = dir('ENTER DIRECTORY CONTAINING TRAINING IMAGES/*.tif');
```

```
imcell = cell(1,numel(Dir));
for i = 1 : numel(Dir)
  imcell{i} = imread(Dir(i).name);
end

temp = imcell{end-1};
imcell{end-1} = temp(:,:,1);

A = cell2mat(cellfun(@(a) reshape(a,
[],1),imcell,'UniformOutput',false));
Cats = A(:,1:80);
Dogs = A(:,81:end);

cd('ENTER DIRECTORY CONTAINING TESTING IMAGES');
sDir = dir('ENTER DIRECTORY CONTAINING TESTING IMAGES/*.tif');

sampleCell = cell(1,numel(sDir));
for i = 1 : numel(sDir)
  sampleCell{i} = imread(sDir(i).name);
end

samples = cell2mat(cellfun(@(a) reshape(a,
[],1),sampleCell,'UniformOutput',false));

% Input the sample into sample
predict = myCatDogClassify(samples,Cats,Dogs,m,n)-1;
```

```
function [ predict ] = myCatDogClassify(S,X,Y,m,n)
%% Get classification information
WaveX = GetWave2D(X,m,n,'haar');
WaveY = GetWave2D(Y,m,n,'haar');
% Reduce the dimension using KL basis with 95% energy.
[ PX,mnX ] = PCAreduce( WaveX,0.95 );
[ PY,mnY ] = PCAreduce( WaveY,0.95 );

%% Classify
WaveS = GetWave2D(S,m,n,'haar');

% Get the novelty relative to the classifiers.
noveltiesCell = cellfun(@(P,mn) myPCAnovelty(P,mn,WaveS),...
                        {PX,PY},{mnX,mnY},'UniformOutput',false);

% Calculate the norm of each of these novelties.
Pjnorm = cellfun(@(novelties) arrayfun(@(i) norm(novelties(:,i)),...
                                                    1:size
(novelties,2)),...

noveltiesCell,'UniformOutput',false);
% Find the classification that has the minimal novelty norm.
[~,predict] = min(cell2mat(reshape(Pjnorm,[],1)));
end

function [W2] = GetWave2D(Gallery,m,n,wavetype)
W2 = zeros(3840,size(Gallery,2));
for i = 1 : size(Gallery,2)
    [coefs,sizes] = wavedec2(reshape(Gallery(:,i),m,n),3,wavetype);
    W2(:,i) = cat(1,reshape(detcoef2('v',coefs,sizes,1),[],1),...
                          reshape(detcoef2('h',coefs,sizes,1),[],1),...
                          reshape(detcoef2('d',coefs,sizes,1),[],1),...
                          reshape(detcoef2('v',coefs,sizes,2),[],1),...
                          reshape(detcoef2('h',coefs,sizes,2),[],1),...
                          reshape(detcoef2('d',coefs,sizes,2),[],1));
end
end
function [ novelty ] = myPCAnovelty( P,mn,probe )
% This function finds the novelty vector compared to the classifier that
% uses the Karhunen-Loeve basis.
% INPUT:
%   clss  - The classification information coming from function
%           myPCAclassifier.m.
%   probe - A matrix where the column vectors are the vectors we want to
%           compare to the classification information, clss.
% OUTPUT:
%   novelty - A matrix where the column vectors are the novelty compared
to
%             the classification clss. NOTE: the novelty may be
considered
%             to have been demeaned; try:
%                             novelty + repmat(clss.mean,1,size
(novelty,2))
%             for a possibly more appropriate result.

% demean
dprobe = probe - repmat(mn,1,size(probe,2));
% Calculate the novelty
novelty = dprobe - P*(P'*dprobe);
end
```

```
function [ P,mn ] = PCAreduce( Gallery,gamma )
% This function gives the Karhunenâ€“LoÃ¨ve basis and the PCA mean.
% INPUT:
%   Gallery  - A matrix where the column vectors are the indivdual
vectors
%              in the gallery.
% OUTPUT:
%   P        - The columns are the reduced Karhunenâ€“LoÃ¨ve basis.
%   mn       - The column-wise mean.

% Get the mean
mn = mean(Gallery,2);

% Get the Karhunen-Loeve basis (optimal basis) using svd
[U,S,~] = svd(Gallery-repmat(mn,1,size(Gallery,2)),0);

% Constant (along with gamma) used to choose the approximation D.
delta = 0.01;

% Find a cut-off of U for efficiency
lambda = diag(S);
ED = cumsum(lambda);

% D is the maximum of the KL energy and stretching dimension
D = find((ED/ED(end) > gamma) & ([lambda(2:end); 0]/lambda(1) <
delta),1,'last');

% Get the reduced KL basis.
P = U(:,1:D);
end
```

6.3. Jody Pritchett.

```
function readimages
directory=dir('TIFFtraining');
names=[];
nam=[];
for i=3:162
    names=[names;directory(i).name;];
end
%names
X=[];
Xmed=[];
Xtest=[];
Xtestmed=[];
Xlap=[];
Xtestlap=[];
h = fspecial('laplacian',1)
for i=1:160
    img=imread(['TIFFtraining/' directory(i+2).name]);
    X=[X reshape(img,4096,1)];
    Xmed=[Xmed reshape(medfilt2(double(img)),4096,1)];
    Xlap=[Xlap reshape(imfilter(double(img),h),4096,1)];
end
directory=dir('TIFFtesting');
for i=3:40
    nam=[nam;directory(i).name;];
end
for i=1:38
    img=imread(['TIFFtesting/' directory(i+2).name]);
    Xtest=[Xtest reshape(img,4096,1)];
    Xtestmed=[Xtestmed reshape(medfilt2(double(img)),4096,1)];
    Xtestlap=[Xtestlap reshape(imfilter(double(img),h),4096,1)];
end
size(X)
size(Xmed)
size(Xtest)
size(Xtestmed)
for i=1:160
    if names(i,1)=='C'
        sublabels(i,1)=1;
    end
    if names(i,1)=='D'
        sublabels(i,1)=0;
    end
end
for i=1:38
    if nam(i,1)=='C'
        testlabels(i,1)=1;
    end
    if nam(i,1)=='D'
        testlabels(i,1)=0;
    end
end

save 'tifftraining.mat' X Xmed sublabels Xtest Xtestmed testlabels Xlap
Xtestlap
```

```
function
[X,sublabels,Xtest,testset,classtest,Xtrain,trainset,classtrain]
=choose_images(numtest)
load('tifftraining.mat')
randcats=randsample(80,numtest/2);
randdogs=randsample([81:160]',numtest/2);
testset=[randcats;randdogs];
trainset=setdiff([1:160]',testset);
Xtest=X(:,testset);
Xtrain=X(:,trainset);
classtrain=sublabels(trainset);
classtest=sublabels(testset);
```

```
function
[X,sublabels,Xtest,testset,classtest,Xtrain,trainset,classtrain]
=choose_med_images(numtest)
load('tifftraining.mat')
randcats=randsample(80,numtest/2);
randdogs=randsample([81:160]',numtest/2);
testset=[randcats;randdogs];
trainset=setdiff([1:160]',testset);
Xtest=Xmed(:,testset);
Xtrain=Xmed(:,trainset);
classtrain=sublabels(trainset);
classtest=sublabels(testset);
```

```
function [phi,sig,v,A,m,x] = compute_snapshot(X)
%Snapshot method
%Input: a real matrix X, n by p, where n>p
%Computes:
%   r orthonormal basis vectors spanning the columns of X-mean(X,2)(the
columns of phi: n by r), where r is the rank of X-mean(X)
%   singular values (sig: an r by r diagonal matrix)
%   orthonormal basis vectors (columns of v: r by r) for (X-mean
(X,2))'(X-mean(X,2))
%   KL expansion coefficients (columns of A: r by p)
%   The mean of the columns of X (m: n by 1)
%   The columns of X, reconstructed by the full KL expansion (x: n by p)
%
X=double(X);
[n,p]=size(X);
m=mean(X,2);
X=X-repmat(m,1,p);
C=X'*X;
r=rank(C);
u=zeros(n,r);
%Compute orthonormal set of eigenvalues for C, and reorder them
%to correspond to eigenvalues in descending order.
[G,d]=eig(C);
dd=diag(d);
[lam,idx]=sort(dd,'descend');
lam=diag(lam);
G=G(:,idx);
%Compute the positive singular values of X
lambda=lam(1:r,1:r);
sig=sqrt(lambda);
v=G(:,1:r);
for j=1:r
    u(:,j)=zeros(n,1);
    for k=1:p
        u(:,j)=u(:,j)+v(k,j)*X(:,k);
    end
    u(:,j)=u(:,j)/sig(j,j);
end
phi=u;
A=sig*v';
%Orthogonal expansion
x=zeros(n,p);
for i=1:p
    for j=1:r
        x(:,i)=x(:,i)+A(j,i)*u(:,j);
    end
    x(:,i)=x(:,i)+m;
end
```

```
function [class,classtest,net,T]=som_lvq
%
%  numtest=32; %number of test cases
%
%Read in the data, creating a training set and a test set
% X=complete data, sublabels = classifications of complete data
% Xtest = test data, testset = indices of the test data in X,
% classtest=classifications of the test data
% Xtrain = training data, trainset=indices of training data, classtrain=
% classifications of the training data
  %[X,sublabels,Xtest,testset,classtest,Xtrain,trainset,classtrain]
=choose_images(numtest);
  %[X,sublabels,Xtest,testset,classtest,Xtrain,trainset,classtrain]
=choose_med_images(numtest);
  % The line above uses median filtered images.  Comment this line and
  % uncomment the one above it if you want to use the raw images.
%
%clear all
load('tifftraining.mat')
  X=double(Xmed);
  Y=double(Xtestmed);
  [r,numtest]=size(Y);
  classtrain=sublabels;
%
%reduce dimension using PCA
%
  [phi,sig,v,A,m,x]=compute_snapshot(X);
%
%Perform analysis to determine D
  normalized_energy=cumsum(diag(sig.^2));
  normalized_energy=normalized_energy/normalized_energy(rank(sig));
  D=min(find(normalized_energy>.95))
%Compute the KL coefficient vectors for each training case.
  a=A(1:D,:);
%Compute the KL weight vectors for the test images
  ap=zeros(D,numtest);
  for i=1:numtest;
      ap(:,i)=phi(:,1:D)'*(Y(:,i)-m);
  end
  %for i=1:160
  %    a(:,i)=a(:,i)/norm(a(:,i));
  %end
  %for i=1:numtest
  %    ap(:,i)=ap(:,i)/norm(ap(:,i));
  %end

%
%Set up inputs for the SOM/LQV

  C=classtrain'+1; %  change to 1=dog, 2=cat for training purposes

  T=ind2vec(C);   %  converts the training classes to points in R^2
(positive integer coordinates)

%load('run_3_1.mat')  %uncomment this line to display the results of
this
%particular run
  net=newlvq(minmax(a),40,[.6 .4],0.1);  %Defines an LVQ
  %
  %iw=[a(:,randsample(80,20)) a(:,randsample([81:160]',20))];
  %iw=iw';
  %net.IW{1,1}=iw';

  %net.trainParam.goal = 0.1;

  net.trainParam.epochs=300;  %Use maximum of 300 iterations of training

  net.trainParam.show=Inf; %Don't show the training parameters at each
iteration

  net=train(net,a,T);  %Call up the program to train the LVQ

for i=1:numtest
    class(i,1)=vec2ind(sim(net,ap(:,i)))-1;  %Run the trained LVQ in
"simulation mode", i.e. put in an input, and get an output
end
%report results
classtest=testlabels;
percent_overall_classfified_correctly = 100*sum
(classtest==class)/numtest
'True Predicted'

[classtest class]
% This stuff below computes the confusion matrix, and the classification
% probabilities, and prints them out.
mm(1,1)=sum((classtest==1)&(class==1));
mm(2,1)=sum((classtest==1)&(class==0));
mm(1,2)=sum((classtest==0)&(class==1));
mm(2,2)=sum((classtest==0)&(class==0));
mp(:,1)=mm(:,1)/sum(mm(:,1));
mp(:,2)=mm(:,2)/sum(mm(:,2));
confusion_matrix = { 'Confusion matrix',      'Cats'    ,     'Dogs'
;...
           'Classified as Cat', mm(1,1) , mm(1,2);...
           'Classified as Dog', mm(2,1) , mm(2,2);...
                 'Total'      ,  sum(mm(:,1)) , sum(mm(:,2))};
disp(confusion_matrix);

class_prob = { 'Probabilities',      'Cats'    ,     'Dogs'    ;...
           'Classified as Cat', mp(1,1) , mp(1,2);...
           'Classified as Dog', mp(2,1) , mp(2,2);...
                 'Total'      ,  sum(mp(:,1)) , sum(mp(:,2))};
disp(class_prob);

%plot final SOM
w=net.IW{1,1};
w=w';
figure
for l=1:40
    subplot(10,4,l)
    imshow(reshape(phi(:,1:D)*w(:,l)+m,64,64),[]);
end
figure
for i=1:38
    subplot(10,4,i)
    imshow(reshape(Y(:,i),64,64),[]);
    if i<=19
        title(['Cat test image ' num2str(i)])
    else
        title(['Dog test image ' num2str(i-19)])
    end
end
```

References

[1] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *Elements of Statistical Learning: Data Mining, Inference, and Prediction.* New York: Springer-Verlag.

[2] Haykin, S. (1999). *Neural Networks: A Comprehensive Introduction*, 2nd ed. New Jersey: Prentice Hall.

[3] Hagan, M., Demuth, H., and Beale, M. (2003). *Neural Network Design.* University of Colorado Bookstore: Martin Hagan.