18551 - Fall 2009 Group 8

I’ve ‘Scene’ This Before!

Miriam Cha, Vini Oommen, Vani Rajan

12/7/2009

Contents

Background Information

The goal of Computer Vision in general is to automate the processes that a human eye undergoes and it is becoming very popular in research today. Our project is inspired by object recognition in Computer Vision. Object Recognition is still a major challenge for computer vision systems. The idea of object detection and recognition is based on the intuition that the information of real-world scenes is relevant for these tasks. This intuition is supported by psychophysical experiments in human scene perception and visual search, which provides evidence that the human visual system uses the relationship between the environment and the objects to facilitate object recognition. So, an object is better recognized if the surrounding environment can be correctly identified.

Project Definition under a Microscope

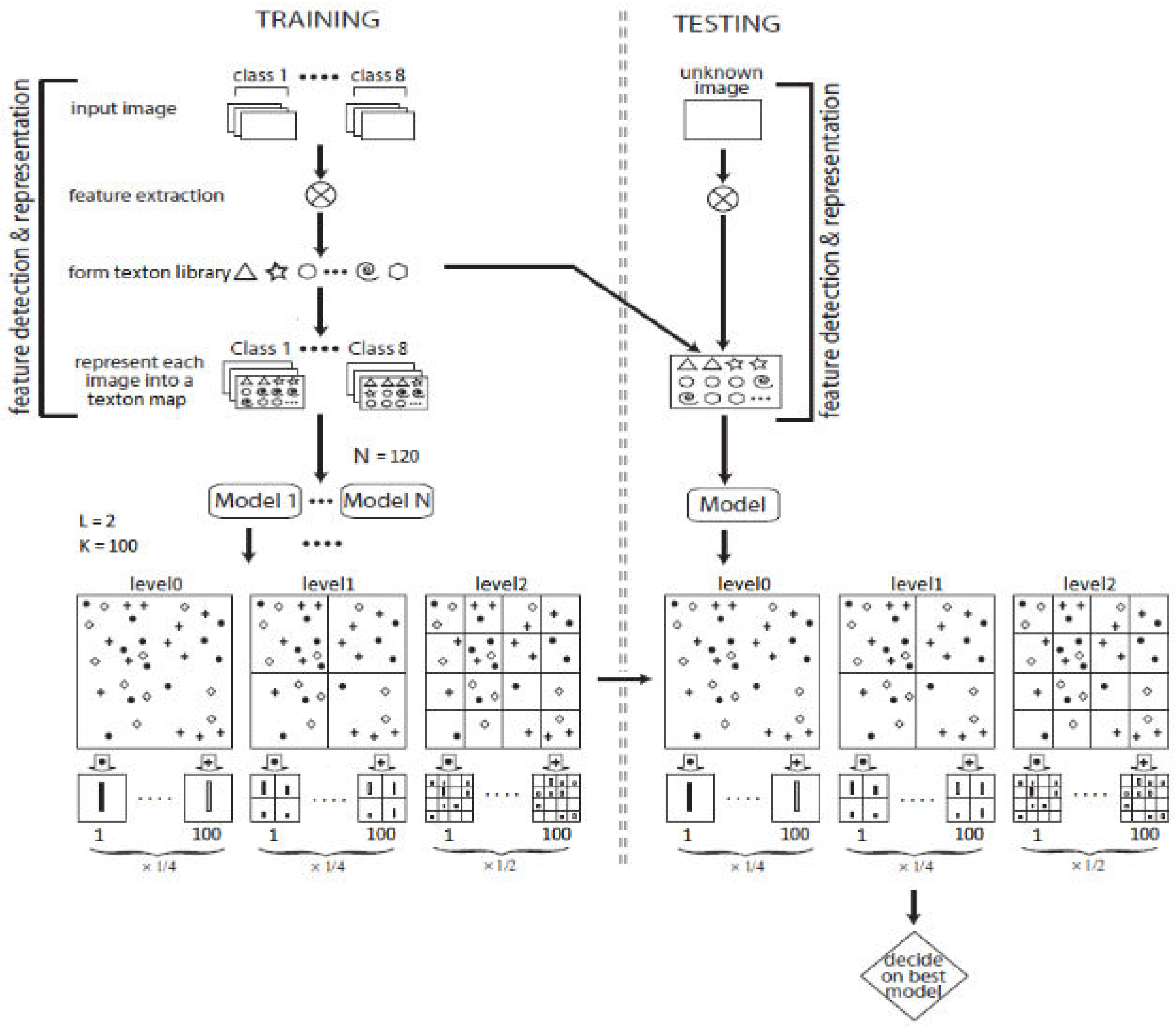
We propose to investigate scene categorization of a whole scenic image based on the Bag of Visual Words Model. In addition to it, we augment a basic bag-of-features representation with Spatial Pyramid Matching. We use Bag of Visual Words and Spatial Pyramid Matching to classify a scene according to the following 8 categories:

1. City
2. Forest
3. Beach
4. Mountain
5. Office
6. Street
7. Suburbs
8. Inside City

Novelty

We are the first ones to do a scene classification project in 18-551. The algorithms that we have used have been previously established, but the implementation of it on the C67 is new, so is the application of it for scene classification.

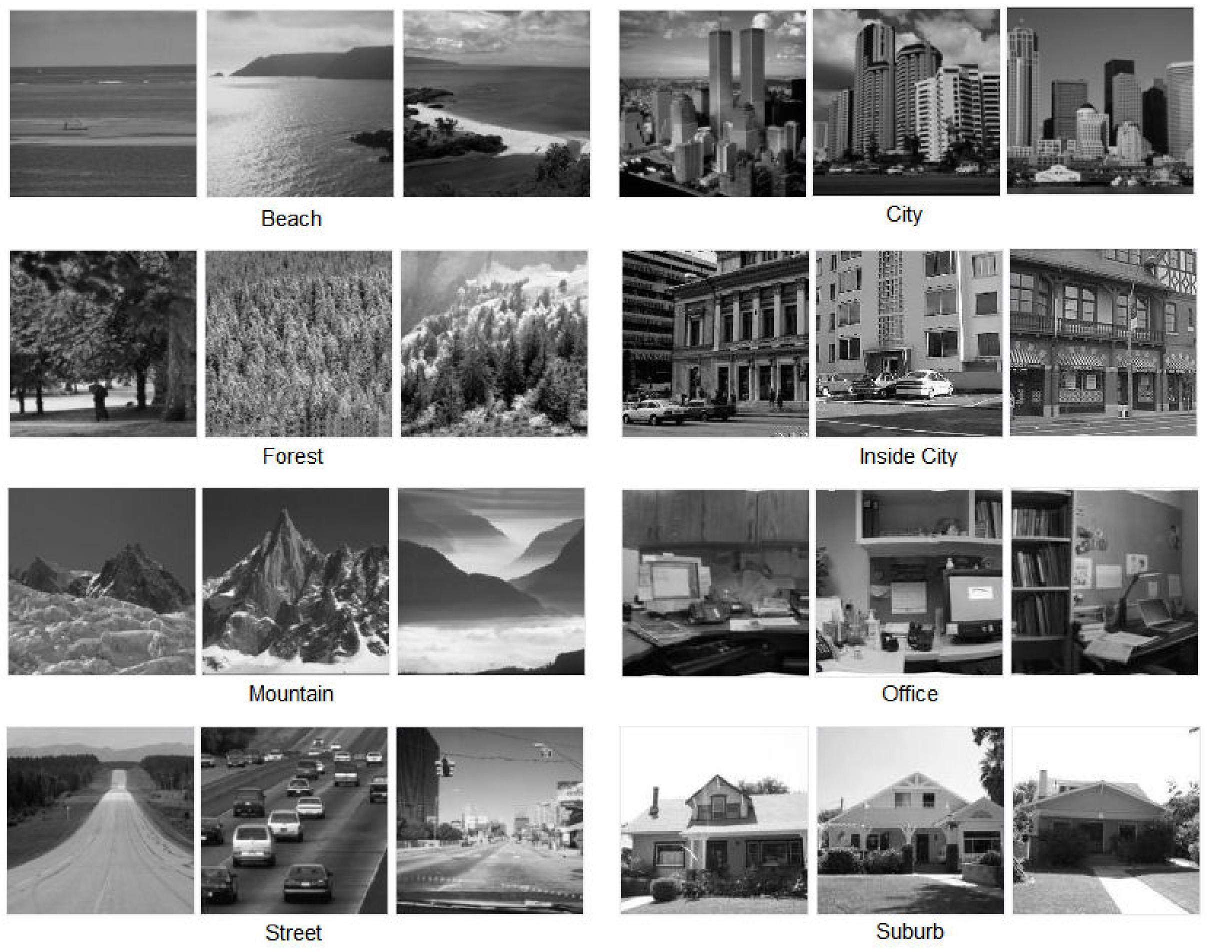
Data Flow Diagram



A description of the above tasks is to be followed in the sections to come. We will also be detailing the algorithms used.

Training

This section will highlight the major steps implemented in our training phase to classify scenes. Throughout the project for both training and testing, we used 128x128 black and white square scenic images from the Caltech 101 dataset. Below is a snapshot of our training library. The training library contained 15 images from each category: 15x8 = 120 training images in total.



We take the training images and convolve it with filters to collect various features as is shown in Figure 1 below

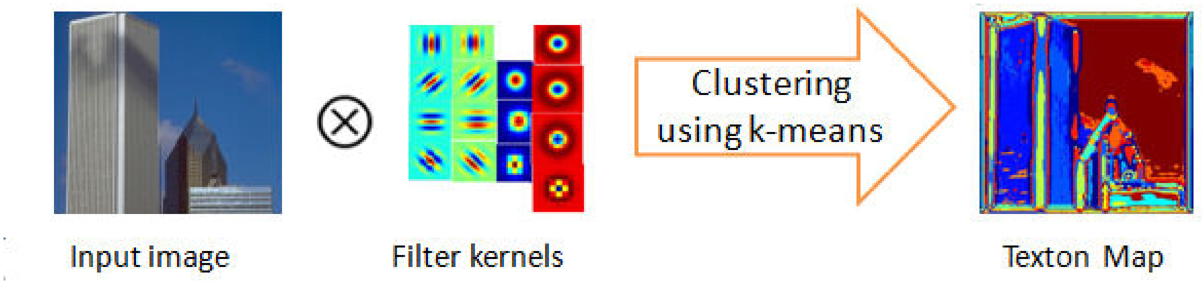


Figure 1: Convolution of input image with filter kernels

Filters

So, we take an RGB image and convert it to CIELab color space. CIELab color space is a color-opponent space with dimension L for lightness and a and b for the color-opponent dimensions. The main reason why we chose Lab color space is that its L component closely matches human perception of lightness. The following filters are applied at every pixel in the L,a,b color space.

The total number of filter responses from all the implemented filters is = 4 Gabor +9 Gaussian +4 LaPlacian = 17 filter responses in total. Below is a short summary on why we used the filters.

Gabor Filter

It is needed to capture lines oriented at specific angles in an image. In our particular case, we use angles of 0, 45, 90 and 135 degrees as θ[1]. Even and odd filters are used in Gabor filter and the reason for that is further explained in **Page 21**. The Gabor filter is only applied to the L component of the image.
In our implementation, we are using 8 Gabor filters as follows:

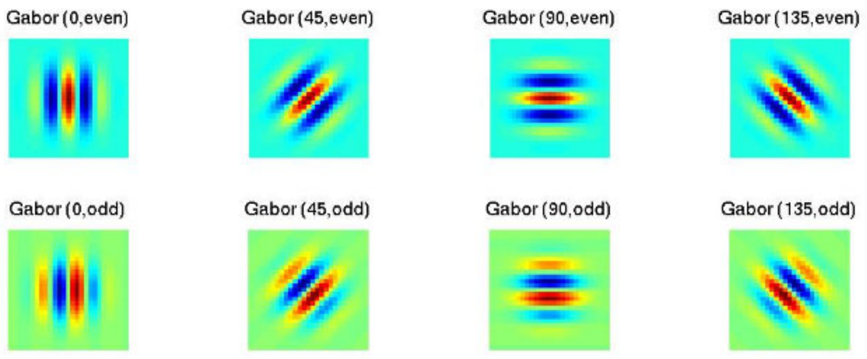


Figure 2: Gabor filters used in algorithm

Gaussian Filter

To eliminate high frequency noise and find the major features in an image. The Gaussian filters in Figure 3 are applied individually to all three channels (L,a and b) of the input image. This brings a total of 3x3 = 9 filter responses generated for Gaussian. We are using 3 Gaussian filters with $\sigma = 1,2$ and 4 as follows[1]:

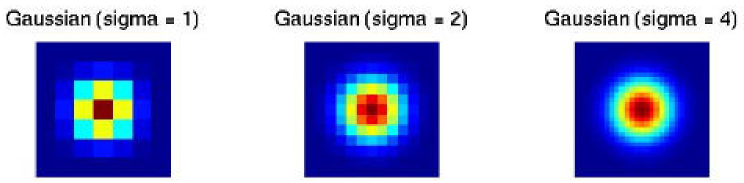


Figure 3: Gaussian filters with various sigma

LaPlacian of Gaussian Filter (LoG)

The LaPlacian of Gaussian filter is applied to the L channel of the input image. So we have a total of 4 filter responses for the LaPlacian of Gaussian filter. The primary motivation behind using LoG filters is to find the edges in an image. Figure 4 illustrates the 4 LoG filters used with σ =1,2,4 and 8[1].

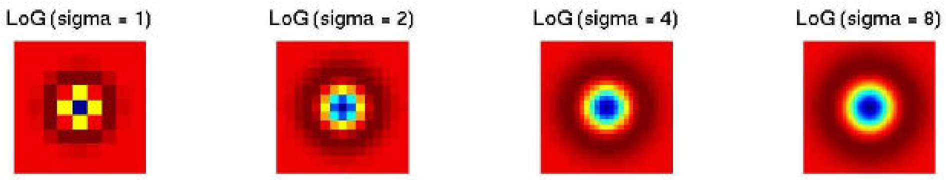


Figure 4: LoG filters with various sigma

Clustering using k-means

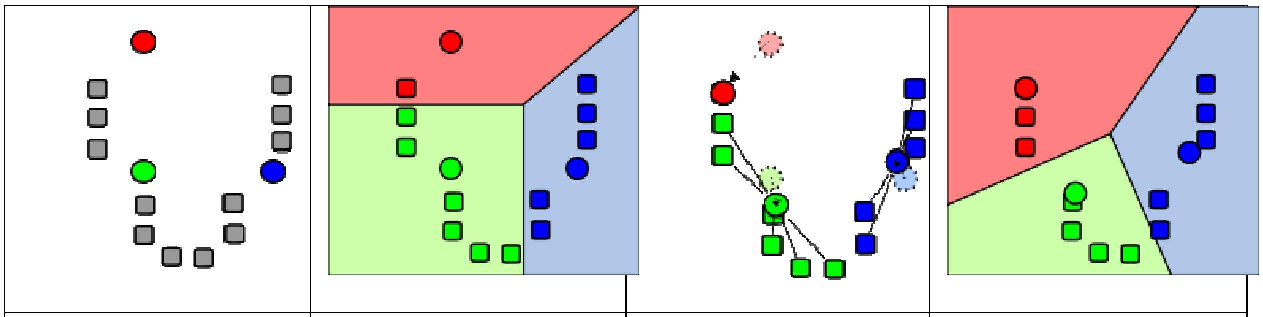
K-means is an important clustering algorithm as it greatly reduces the number of pixels to compare. K- means uses the 17 filters responses from each image in the training set to find 100 features that best represent the whole training set of 120 images that are of size 128*128. Each image in the training set is convolved with the 17 filters which results in a matrix of features that is 16384 x 17 (128*128 = 16384). It is too computationally expensive to use all the features. Thus we randomly chose 100 features from each set of filter responses. This is done with a random permutation generator function on MATLAB called randperm. This leaves us with a matrix that is 100 x 17 of features for each image. Since the idea of kmeans is to get 100 features that could describe all of the images, each matrix of 100 x 17 is concatenated into a huge feature matrix. This giant feature matrix of 12000 x 17 is the input matrix to the kmeans function.

The algorithm kmeans uses to cluster this large amount of points is a two phase iterative algorithm to minimize the sum of point-to-center distances, summed over all k clusters. Before the beginning of the iterative steps a starting set of cluster means is chosen. This gives a starting point, that can be updated as necessary. There are ways to cleverly choose these clusters but in our algorithm, we randomly chose 100 features from the giant feature matrix as starting means.

“The first phase uses what the literature often describes as "batch" updates, where each iteration consists of reassigning points to their nearest cluster mean, all at once, followed by recalculation of cluster means. You can think of this phase as providing a fast but potentially only approximate solution as a starting point for the second phase.

The second phase uses what the literature often describes as "online" updates, where points are individually reassigned if doing so will reduce the sum of distances, and cluster means are recomputed after each reassignment. Each iteration during this second phase consists of one pass though all the points.

Kmeans can converge to a local optimum, in this case, a partition of points in which moving any single point to a different cluster increases the total sum of distances. This problem can only be solved by a clever (or lucky, or exhaustive) choice of starting points.” (kmeans Math works documentation) A standard demonstration of this algorithm is shown in the images below.



1) Initially k means (k=3) are randomly selected from the data	2) k clusters are created by associating all data with the nearest mean.	3) The mean of each cluster becomes the new center.	4) Steps 2 and 3 and repeated until convergence is received.

Figure 5: Standard implementation of Kmeans [3]

Besides the feature matrix and the number of clusters wanted, there are a few other parameters of the kmeans function that we use in our algorithm. There is a parameter called ‘Empty Action’, which deals with what to do when a cluster loses all of its members because they are reassigned to another cluster. We have chosen to have empty clusters dropped since there are better points to approximate a common feature.

The last parameter of kmeans that is used in our algorithm is the method of finding the distance between points. Since we do not explicitly state the algorithm to be used, the default of Squared Euclidian distance is used to calculate the mean of each cluster every time an iteration goes through.

How to Create a Texton Map

Once the kmeans algorithm is finished we are left with a texton library that is 100 x 17. We use these universal features to find the texton map of each image. This can be done through a comparison of the features of an image with the texton library. Features in the image that are close a feature in the texton library are then associated by the features from the library.

The closeness of the vector of features from the image and the vector of features from the texton library is defined by L2-norm, which is also referred to as the Euclidean norm. The Euclidean norm is a comparison of the distance between two points. Thus given a texton library of 100 x 17 and a feature matrix of 16384 x 17, the L2-norm returns a matrix of 100 x 16384.

The way this matrix is generated is by comparing one row of the texton library (1x17) with each row of the feature matrix, 16384 rows of (1 x17). In a smaller view, for one iteration we compare one row of (1 x 17) with 16384 rows of (1x17) to generate one row (1x16384) of the output matrix. In the end, this matrix is the squared distance of all the pairs of points from the texton library and the feature matrix.

The next step is to find the smallest distance in each column of the 100 x 16384 matrix and what index in the image this is at. The smallest distance in each column should aim to be close to zero because we want close matches of the features in the image to textons from the library. The matrix of 100x16384 is a comparison of each pixel in the image to each of the 100 textons. Thus indices are necessary because the index of the minimum tells which texton from 1 to 100 maps the closest to the pixel in the image.

Once we have the minimum from each 16384 columns, we reshape the column to be 128x128 and this is the final texton map.

Spatial Pyramid Matching

Spatial Pyramid Matching is a kernel-based recognition method that works by partitioning the image into increasingly fine sub-regions and computing histograms of local features found inside each sub-region.

In this paper, we would like to elaborate on how spatial pyramid matching technique is being used in our 18-551 project, I've 'Scene' This Before. In our project, spatial pyramid matching method is used in both training and testing sets to generate histograms. The histograms generated from the training set are collected in the form of a histogram libarary and later the histrograms generated from the testing images are compared against the library to be categorized.

Since a pyramid match kernel works with a texton representation, we first textonify the testing image. Figure 6 shows an example of testing image (128x128) and the textonified result of the image. In the project, we use 15 testing images from each of 8 classes but for the ease of explanation, in this paper, we woule like to explain the algorithm using just one testing image. The generated texton map approximates all the variety of texture in the beach image with 100 set of textons (K=100).

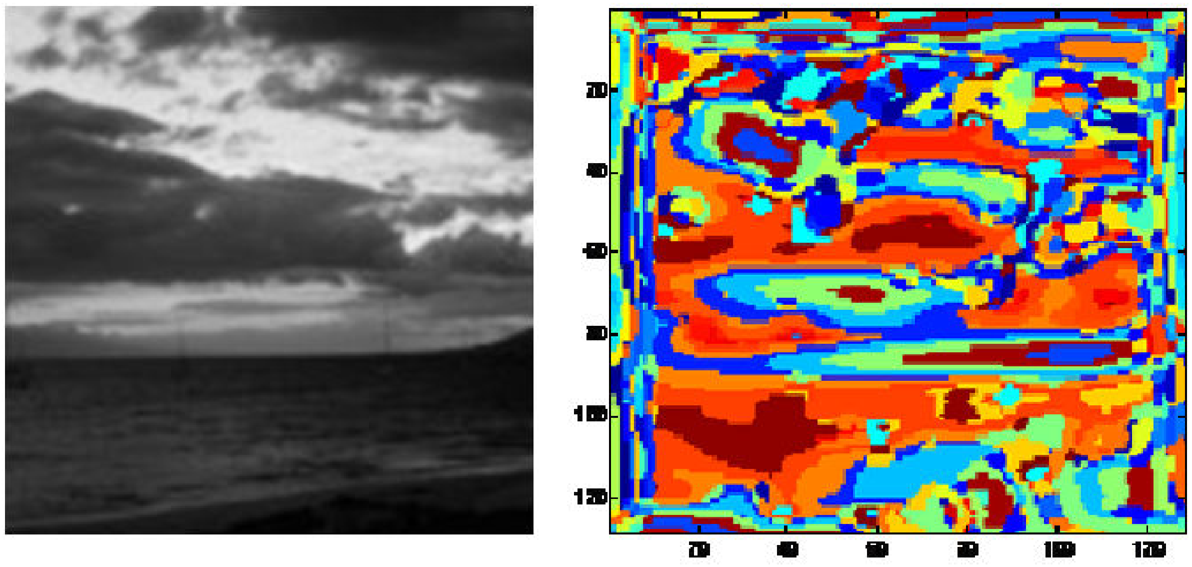


Figure 6: The left image is a sample image of the beach class used for this project and the right image is the texton map of the left image.

Using the texton map generated, the pyramid matching method works by placing a sequence of increasingly coarser grids over the texton map and taking a weighted sum of the number of matches that occur at each level of resolution. At any fixed resolution, two points are said to match if they fall into the same cell of the grid; matches found at finer resolutions are weighted more highly than matches found at coarser resolution [1].

In the project, we decided to construct a histogram with a sequence of grids at resolutions from 0 to 2(L=2), such that the grid at level l has 2^lcells along each dimensions, for a total of 2^{Dl} cells. Therefore for the first level, there is only one cell (2^{2*0}), for the second level, there are four cells (2^{2*1}), and for the third level, there are 16 cells (2^{2*2}).

From the definition of a pyramid match kernel, we get the following equation:

$$\begin{aligned} \kappa^L(X,Y) &= \mathcal{I}^L + \sum_{\ell=0}^{L-1} \frac{1}{2^{L-\ell}} \left(\mathcal{I}^\ell - \mathcal{I}^{\ell+1}\right) \\ &= \frac{1}{2^L}\mathcal{I}^0 + \sum_{\ell=1}^{L} \frac{1}{2^{L-\ell+1}}\mathcal{I}^\ell . \end{aligned} \tag{1}$$

In this equation, $\mathcal{I}^\ell$ denote the number of points that fall into the grid at level *l*. Note that the number of matches found at level *l* also includes all the matches found at the finer level *l* +1. Therefore, the number of *new* matches found at level *l* is given by $\mathcal{I}^\ell - \mathcal{I}^{\ell+1}$ for $l = 0, \ldots, L-1$. The weight associated with level *l* is set to $\frac{1}{2^{L-\ell}}$[1]

In our case, we have the texton map of the testing image that has 100 texton types (K=100). We subdivide the image at three different levels of resolution (L=3). Next, for each level of resolution and each channel, we count the textons that fall in each spatial bin. Finally, we weight each spatial histogram according to equation (1). Figure 7 shows the histogram generated from level 0 that consists of one cell.

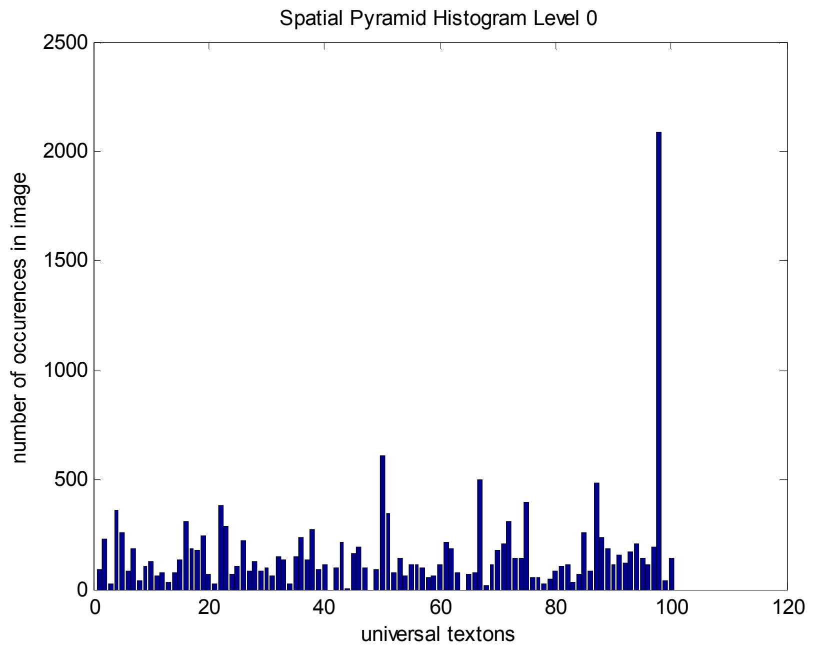


Figure 7: Histogram of the testing image (Figure 6) in Level 0.

Figure 8 shows the histogram generated from level 1 that consists of four cells.

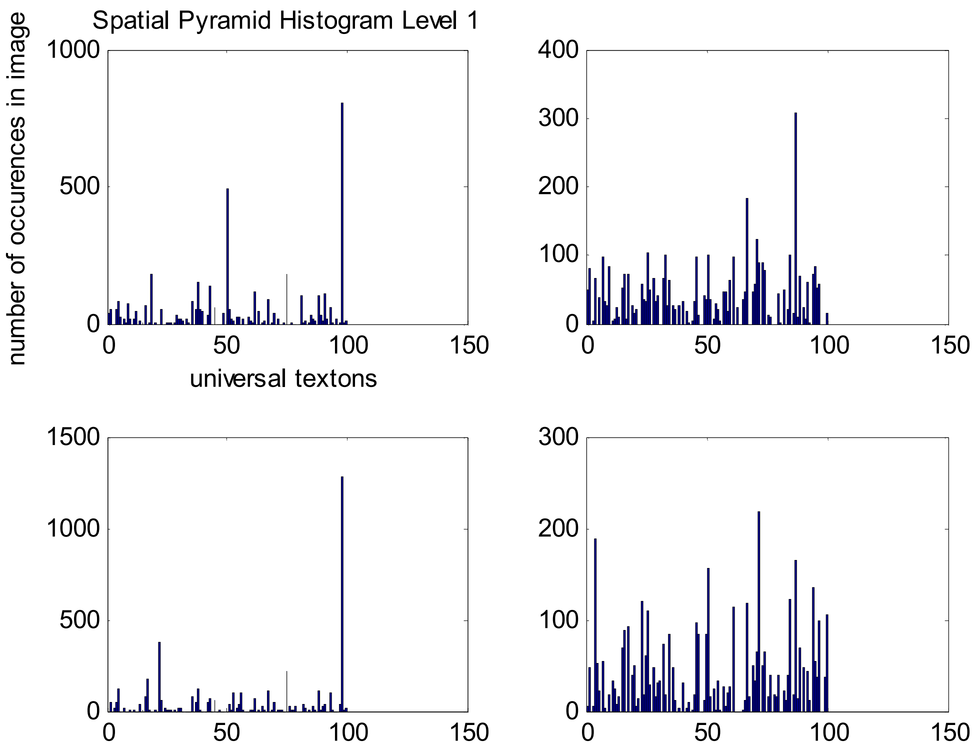


Figure 8: Histogram of the testing image (Figure 6) in Level 1.

Figure 9 shows the histogram generated from level 2 that consists of 16 cells. The top cell on the first column represents the histogram of local features for the top left corner of the texton map when the image is partitioned into 16 sub-regions. Other histograms also represent the number of textons that fall into each bin in similar manner.

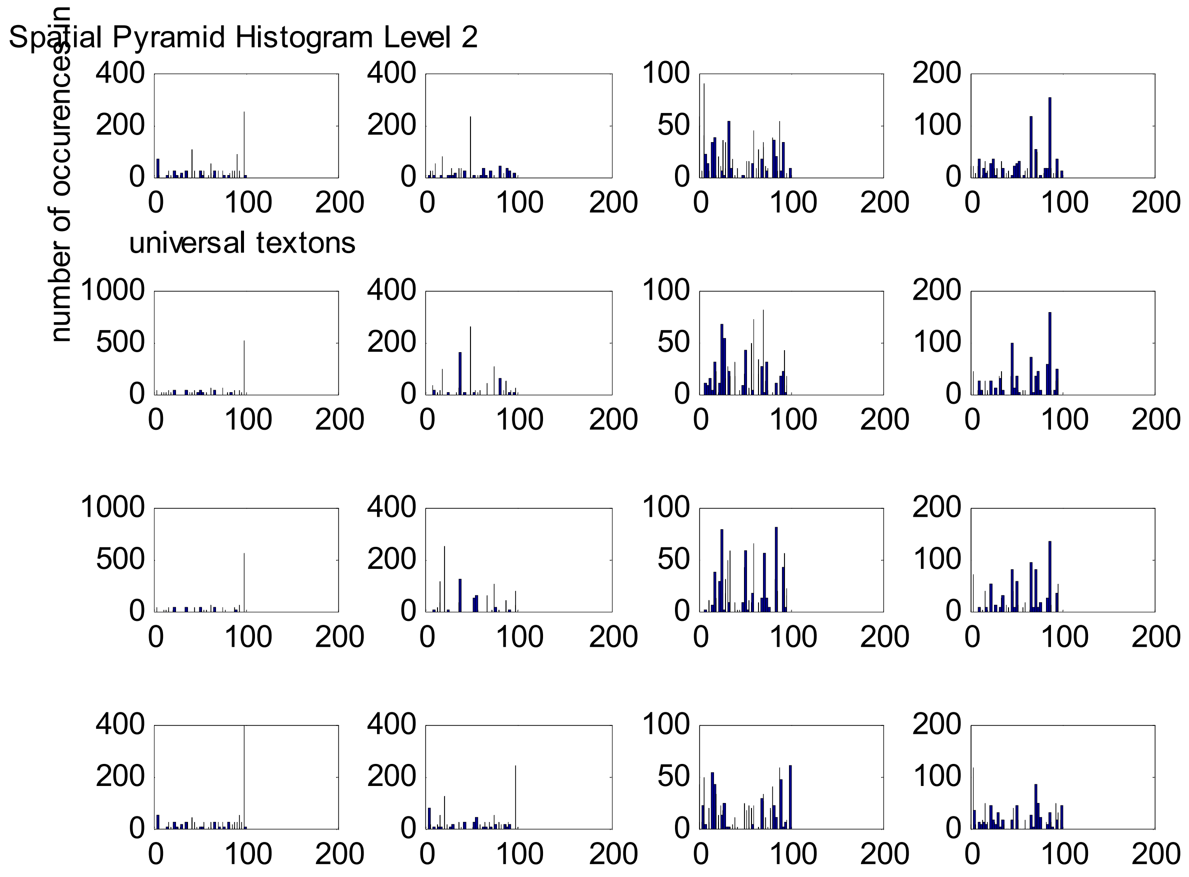


Figure 9: Histogram of the testing image (Figure 6) in Level 2.

The final kernel is computed by summing all the separate channel kernels:

$$K^L(X,Y) = \sum_{m=1}^{M} \kappa^L(X_m, Y_m) \quad (2)$$

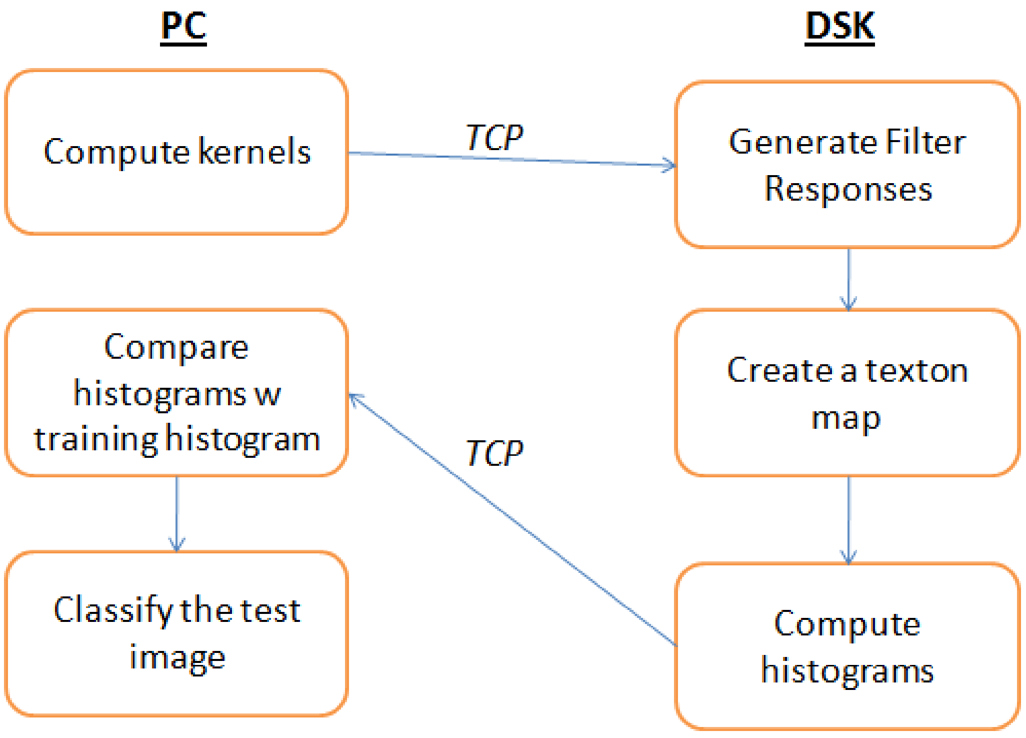
Based on equation (1), we can see that the pyramid match kernel is simply a weighted sum of histogram intersections. Therefore we can implement K^L as a single histogram intersection of long vectors formed by concatenating the weighted histogram of all channels at all resolutions.

For L=2 and K=100, the resulting vector has dimensionality $K \sum_{\ell=0}^{L} 4^\ell = K\frac{1}{3}(4^{L+1} - 1)$ which is 2100-dimensional histogram intersections.

Testing

Testing library contained 15 images from each category: 15x8 = 120 testing images in total. We made sure not to include any images from the training set as the test image. We perform similar operations to the test image as we did for the training images. Namely, creating a texton map, perform spatial pyramid matching on the texton map generated and then we compare histograms by nearest neighbor. This was computationally expensive to do on the DSK so we did it offline on the PC. Here is our signal flow graph on how we divided the processes between DSK and PC.

Signal Flow Graph



Data Analysis:

Initially, when we performed scene classification on our test set, we observed some fluctuation in classification rates. On average, the classification rate was about 70%. This is the results matrix that gave us the best classification rate with the original dataset.

	Beach	**City**	**Forest**	**Inside City**	**Mountain**	**Office**	**Street**	**Suburb**
Beach =73.3%	11	0	0	0	2	0	2	0
City =87%	0	13	0	2	0	0	0	0
Forest =87%	0	0	13	0	1	1	0	0
Inside City=73%	0	1	0	11	0	2	0	1
Mountain=87%	1	0	0	0	13	0	0	1
Office =53%	0	1	0	1	2	8	1	2
Street =53%	4	0	1	1	0	0	8	1
Suburb=80%	1	0	0	0	1	1	0	12

Table 1: Results Matrix with an accuracy rate of 74.16%

Rejection

Our project classifies 120 scenic images into 8 different classes. The one thing that this algorithm does not account for is rejection. Given any image the algorithm will chose to which class that image is closest to even if the image is not a scene at all. Thus we chose to first find average histograms that would be representative of each class. Then use these average histograms to compare to the ones being chosen by the algorithm by using a percent error calculation. When we examine the percentages being calculated from the histogram comparisons we realized that it is not going to be possible to threshold between classes.

For instance, we looked at the images in beach that got classified as beach or street. The percent error between the histograms of the images a and b is about 5%. The percentage of error between images c and d is about 2.5% but as we can see image d is not a beach.

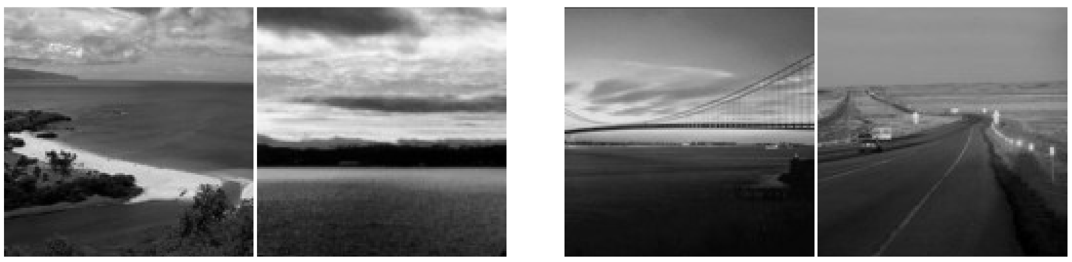


Figure 10. Left: Beach images that classify as Beach. Right: Beach images that classify as Street

If this was just one case, it would be possible to dismiss it as a false negative, which is the algorithm fails to detect the difference in the beach and street image. When we take a closer look at the values or the percent errors between the chosen classifying histogram and the average histogram for the appropriate class, the values fluctuate for both correct and incorrect classifications. Thus this makes it impossible to select one thresholding value for each class.

Optimization

The accuracy of classification from the data set originally was around 70% but there were major problems in the beach and street classification. As seen in the results below, many of the street images are classified as beaches.

Original - 69.1667% accuracy								
	Beach	City	Forest	Inside City	Mountain	Office	Street	Suburb
Beach	11	0	0	0	1	0	3	0
City	0	13	0	2	0	0	0	0
Forest	0	0	12	0	3	0	0	0
Inside City	0	2	0	12	0	1	0	0
Mountain	3	1	0	0	10	1	0	0
Office	0	1	0	2	0	8	2	2
Street	5	0	1	0	1	0	6	2
Suburb	2	0	0	0	0	2	0	11

Table 2: Classification results from original data set

Thus a closer look into this shows there are certain images which consistently threw off the algorithm from the correct classification. One such instance is the following 3 test street images which are all classified as the training beach image on the right. The cars are seen as scattered clouds which make the street images below classify as a beach.

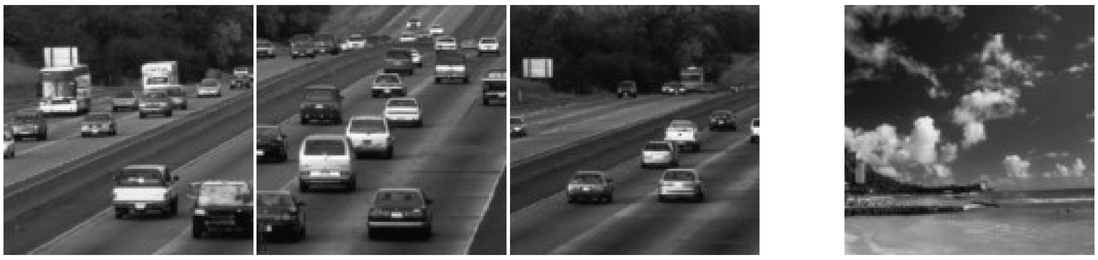


Figure 11: Classification error of street image to beach on the right.

Another instance of misclassification came from the following beach images being classified as street on the right.

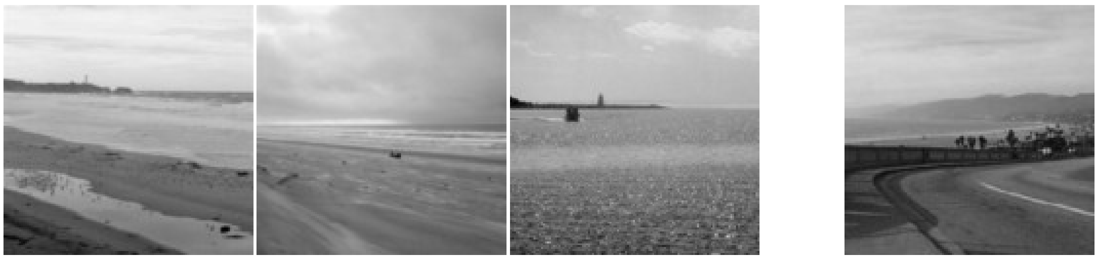


Figure 12: Classification of beach image to street on the right.

Because of these misclassifications we replaced images in the beach training set and saw that the classifcation accuracy rose from 70% to about 73% as shown in the result table below. Changing an image in the beach training set helps to clarrify classification of street images as beaches but the algorithm still confuses street with office and suburb.

Change training beach(29).jpg – 73.33%								
	Beach	City	Forest	Inside City	Mountain	Office	Street	Suburb
Beach	13	0	0	0	0	0	2	0
City	0	13	0	1	0	1	0	0
Forest	0	0	13	0	2	0	0	0
Inside City	0	2	0	10	1	2	0	0
Mountain	1	0	0	0	13	1	0	0
Office	0	2	0	1	1	8	2	1
Street	1	0	1	1	0	2	7	3
Suburb	2	0	0	0	1	1	0	11

Table 3: Classification Results After Changing a Beach Training Image

The next step we took in fixing the street classification was directly looking at the street training versus the street testing images. Many of the street training images look like the 2 images on the left in Figure 13 but the street testing images include images like those in the training plus ones pictured below in Figure 13 on the right. Thus the street algorithm was only trained to handle one types of street image, which lead to images like in Figure 13 to the right being classified as both office and suburb.

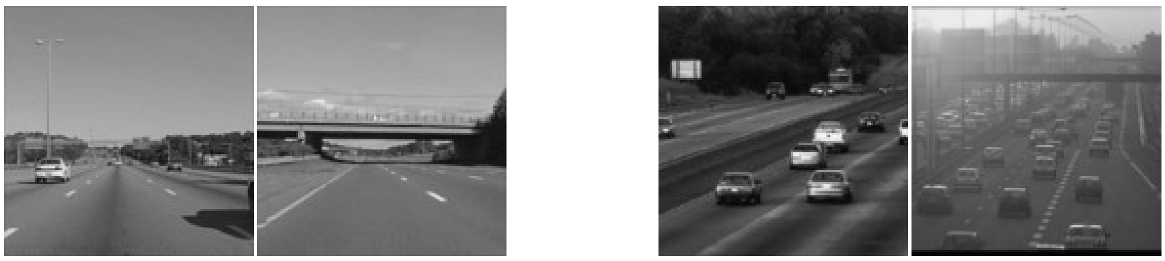


Figure 13: Street Training: Images on left

Street Testing: Images on the Right

To compensate for overlooking this, we switched the street training images with the street testing images. This allowed the algorithm to train on street images that were a bigger variety. Once this was done the overall accuracy of the algorithm rose once again from 73% to about 76%, as shown in the results table below.

switch street training and testing – 77.5%								
	Beach	City	Forest	Inside City	Mountain	Office	Street	Suburb
Beach	11	0	0	0	0	0	4	0
City	0	12	0	3	0	0	0	0
Forest	0	0	13	0	2	0	0	0
Inside City	0	2	0	12	0	1	0	0
Mountain	3	0	0	0	11	1	0	0
Office	1	0	0	2	0	9	1	2
Street	1	0	0	0	0	0	14	0
Suburb	1	0	0	0	0	2	1	11

Table 4: Classification Results After Switching the Street Training and Testing Images.

From this table, we can see that compared to Table 2, the correct classifications of street have raised but there are still a large number of beach images being classified as street. From further analysis of the beach versus street histograms, it was seen that given the images that we are using, it will not be possible to independently increase the classification accuracy of both street and beach. There are still many images in both classes that, even to the human eye, are difficult to distinguish as either beach or street, which gives rise to texton maps and histograms that are almost interchangeable between the two.

Our best category was Street. This was previously one of the worst categories, but once we took out the problematic training and testing images, we got a much better classification rate. Our worst classification category in terms of performance was Office. For some reason, indoor scenes in general did not work as well as we expected, since the features kept getting matched up with that of outdoor scenes. However, Beach, though it got a classification rate of 73.3%, it was a very problematic category because it seemed to be the most common wrong classifier as indicated in Table 4.

Confirmation of our chosen settings (Ks and Ls)

Table 5 shows the detailed results of classification experiments using the database we specified above. The highest results are shown in bold. Based on the classification experiment, we decided to take K = 100 and L = 2 as our final parameters because they yielded the highest classification result (74.1%) with the lowest operational cost.

First, let us examine the performance of single-level (standard bag of visual words) and pyramid of all three kinds of features. The improvement in classification rate is very clear from single-level to pyramid representation.

It is interesting to compare the performance of L = 2 and L =3 for strong features (K = 100 and K = 200). For strong features, single-level performance drops as we go from L = 2 to L = 3. This means the highest level of the L=3 pyramid has resolution that is too finely subdivided, with individual bins yielding too few matches. However, performance of spatial pyramid representation does not drop as much as single-level. This is the main advantage of the pyramid representation because it combines multiple resolutions; it is robust to failures at individual levels. For weak feature (K = 16), performance continues to increase from L = 2 to L =3 because weak features have a much higher density and much smaller spatial extent than strong features.

	K = 16		K = 100		K = 200	
L	Single-level	Pyramid	Single-level	Pyramid	Single-level	Pyramid
0 (1 x 1)	63.3%		69.1%		71.6%	
1 (2 x 2)	64.1%	66.6%	70.8%	72.0%	72.0%	72.5%
2 (4 x 4)	65.0%	67.5%	70.8%	**74.1%**	72.5%	**74.1%**
3 (8 x 8)	66.6%	**69.2%**	67.5%	72.5%	66.6%	73.3%

Table 5: Classification results (see text). The highest result is shown in bold.

Even and Odd Gabor Filter Analysis

For optimization, we tested a number of different filter banks containing various combinations of Gabor, Laplacian of Gaussians, and Gaussians kernels. Many filter-banks produced comparable results but the best one was the combination of the three kernels. In our case, we use both even and odd components of the Gabor filter. We found that this is the most compatible for our project. We tried extracting filter responses using LoG, Gaussian and odd components of the Gabor filter and similarly, LoG, Gaussian and even components of the Gabor filter. The classification rate obtained using the former was 70.00% which is 4.16% lower than that obtained when we use both. Using the even component, we obtained an accuracy rate of 68.33%, which is 5.83% lower than that obtained when we used both. So, we decided to use both components in our project.

Data Flow

PC->DSK

- L,A,B per image = 65,536 bytes = 64 kB per image
- Gab1 – Gab8 = 27x27x4x8 = 23,328 Bytes
- LaPlacian Filters
    - LoG1 = 9x9x4 = 324
    - LoG2 = 15x15x4 = 900
    - LoG3 = 27x27x4 = 2,916
    - LoG4 = 50x50x4 = 10,000
    - LoG1 – LoG4 = 14,140 Bytes
- Gau1 – Gau3 = 3,788 Bytes
- TextonLibrary = (100x17)x4 = 6,800 Bytes
    - Filters+ TextonLibrary = Total of 48,056 Bytes

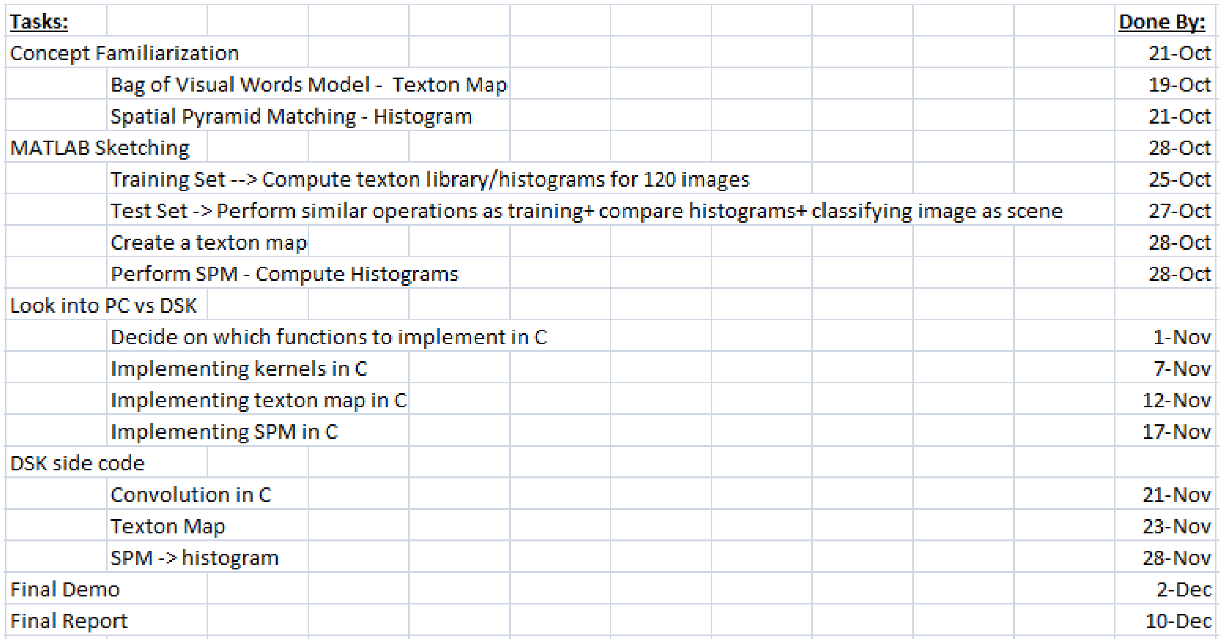
DSK->PC

- textonMap = 128x128x(sizeof(float) = 4) = 65,536 Bytes
- SPM(histogram) = 1x2100 = 2100 Bytes

Demonstration

Our C-DSK communication did not go as well as planned. **We implemented the following functions in C: the 3 kernels, creating texton map and performing Spatial Pyramid Matching on the generated texton map and it completely worked according to plan.** At first, we were a bit concerned that generating the texton map in C will look completely different from that generated in MATLAB**, but we could examine that we obtained an exact match by subtracting the texton map obtained in MATLAB and that obtained by the C code**. Our classification rate after performing SPM on the texton Map was even a bit better than that obtained using MATLAB. **Our C code completely worked**; however, the C-DSK network code was not stable. It worked somewhat to a point, but continued debugging made the performance worse and worse. This rendered the DSK useless, as we could not get data to it. We did demonstrate the entire project on the PC, using MATLAB.

Semester Schedule



Tasks:		Done By:
Concept Familiarization		21-Oct
	Bag of Visual Words Model - Texton Map	19-Oct
	Spatial Pyramid Matching - Histogram	21-Oct
MATLAB Sketching		28-Oct
	Training Set --> Compute texton library/histograms for 120 images	25-Oct
	Test Set -> Perform similar operations as training+ compare histograms+ classifying image as scene	27-Oct
	Create a texton map	28-Oct
	Perform SPM - Compute Histograms	28-Oct
Look into PC vs DSK		
	Decide on which functions to implement in C	1-Nov
	Implementing kernels in C	7-Nov
	Implementing texton map in C	12-Nov
	Implementing SPM in C	17-Nov
DSK side code		
	Convolution in C	21-Nov
	Texton Map	23-Nov
	SPM -> histogram	28-Nov
Final Demo		2-Dec
Final Report		10-Dec

Future Improvements and General Recommendations

A) A new technique for the indoor classification:

In the beginning stage of our project, we realized that our scene recognition model works well for outdoor scenes but performs poorly in the indoor domain. The main difficulty was while some indoor scenes (e.g. corridors) can be well characterized by global spatial properties, others (e.g., bookstores) are better characterized by the objects they contain. More generally, to address the indoor scenes recognition problem, we needed a model that can exploit local and global discriminative information. For this reason, we decided to modify our database by adding more outdoor scenes such as suburb images and inside city images and removing some of the indoor images. We believe in the next stage of the project we could use several global and local feature descriptors that lead better result for not only outdoor but also indoor classification.

B) Alternative method for Nearest Neighbor:

The choice of a good classifier is important. *K*-means has been criticized for being slow and difficult in determining the correct value of $\underline{k}$ from a validation set [4]. Unfortunately, we examined that k nearest neighbor classifier does not produce a computationally cheap and reliable technique of indoor/outdoor classification that can be used in real-time imaging because of the computational complexity of feature determination. A future improvement of this project can be finding an approach that reduces or avoids dimensionality problems.

C) Optimized Histogram Analysis:

From the classification result table(Table 1), we can examine that the coarse grained geometric cues provided by the pyramid matching technique have more discriminative power than an enlarged number of features (increasing K). The optimal way to exploit structure both in the image and in the feature space may be to combine them in a unified multi-resolution framework; this is also subject for future research

References

- [1] J.Winn, A. Criminsi and T.Minka. *Object Categorization by Learned Universal Visual Dictionary.* ICCV. 2005.
    - The paper presents object categorization method, the *Bag of Visual Words* that automatically recognizes the object classes from images. We were able to study combinations of effective filters and their parameters based on the filter banks described on this paper.
    - Active Link: http://johnwinn.org/Publications/papers/Winn_Criminisi_Minka_VisualDictionary_ICCV2005.pdf

- [2] S. Lazebnik, S. Schmid and J. Ponce. *Beyond Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories.* CVPR. 2006.
    - Just like the title of the reference paper, this paper presents a method called *Spatial Pyramid Matching* that extends a bag of visual words method to recognize natural scene categories. The paper describes the Spatial Pyramid Matching algorithm in very detail and became the major reference for our project.
    - Active Link: http://www-cvr.ai.uiuc.edu/ponce_grp/publication/paper/cvpr06b.pdf

- [3] "K-means clustering -." *Wikipedia, the free encyclopedia*. Web. 13 Nov. 2009. <http://en.wikipedia.org/wiki/Kmeans>.
    - A pictorial description of k-means that clearly illustrates the concept.

- [4] N. Serrano, A. Savakis, J. Luo, Improved scene classification using efficient low-level features and semantic cues, Pattern Recognition, 2004, in press.
    - This paper describes the nature of k-means in its truest form.
    - Active Link: http://www.sciencedirect.com/science?_ob=MImg&_imagekey=B6V14-4CBV1X2-1-3V&_cdi=5664&_user=525223&_orig=search&_coverDate=09%2F30%2F2004&_sk=999629990&view=c&wchp=dGLbVtb-zSkzV&md5=76d4f430e78c25b5f40fcaa6b523e743&ie=/sdarticle.pdf