Homework 1

Rajat Vikram Singh
rajats@andrew.cmu.edu

2nd February, 2016

1.0. The visualization of the filter bank is given below:

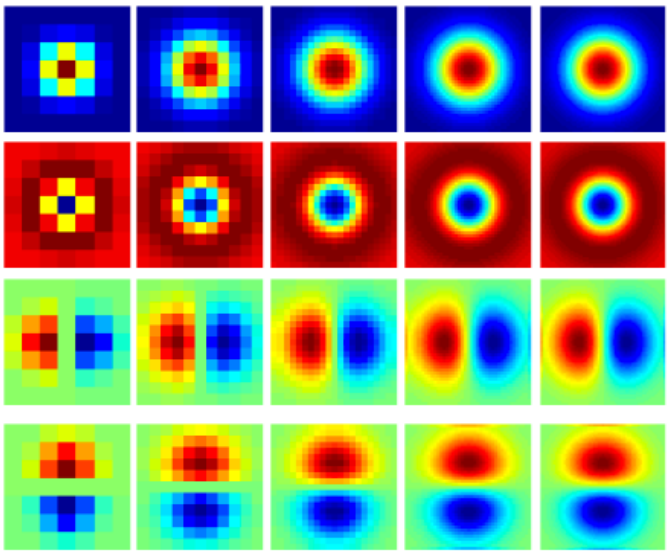

Figure 1. 2D visualization of filterBank

The first row consists of gaussian filters with different scales. The second row is made up of laplacian of gaussians. The last two rows are derivatives of Gaussian filters with different scales and orientations.

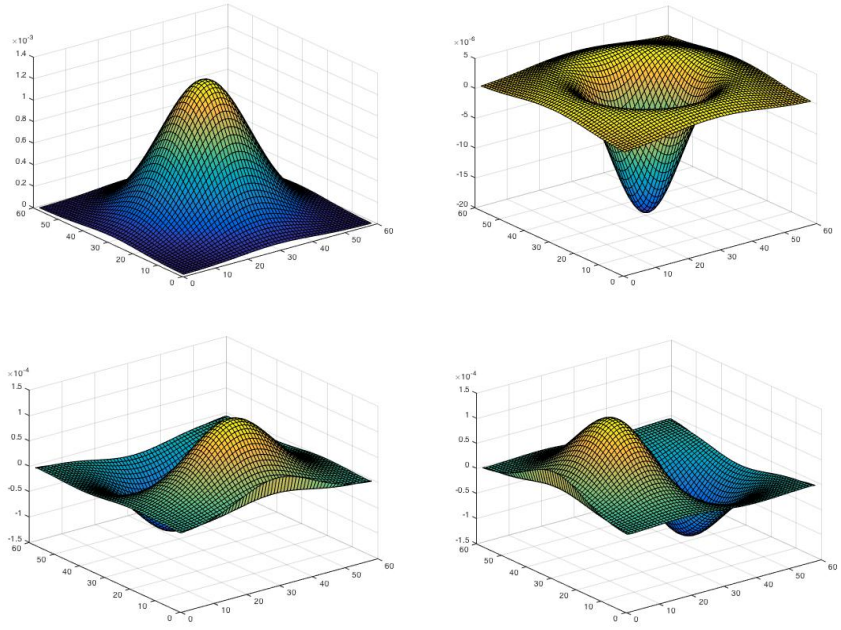

Figure 2. surf plot of filters 5, 10, 15 and 20 (clockwise)

These filters will extract the feature information from the images which will be used for the representation of the image as filterResponses. The filters and their use is described below:
Gaussian filters – The Gaussian filters work as a blurring filter and with different variances it can cause multiple levels of blurring in the image. Blurring can be considered as a kind of image feature in the way that it stores the prominent features of the image.
Laplacian of gaussians – These filters help to extract the edge features of the image.
Derivatives of gaussians – These filters also help in extracting edge information from the image. The different rows of the filterbank help in extracting edge information in x and y directions.

1.1. Implemented the function *extractFilterResponse,* for an image of size 240x320, *filter_response* is of size 76800x(3x20).

1.2. Alpha is the number of random points which we select from the image. This is done to make sure that all the feature vectors have the same dimension. Although we are using random points, if we choose a fixed number of interest points (For Ex. SIFT), it would help improve the recognition accuracy.
K is the size of the dictionary, the center-points of the clusters, which we use to describe the image in the nest step.
Changing values of alpha and K can improve the accuracy of the system, which depends on running multiple iterations

1.3. Created *wordMaps* using the function *getVisualWords.* Following are some examples to visualize the wordMaps:

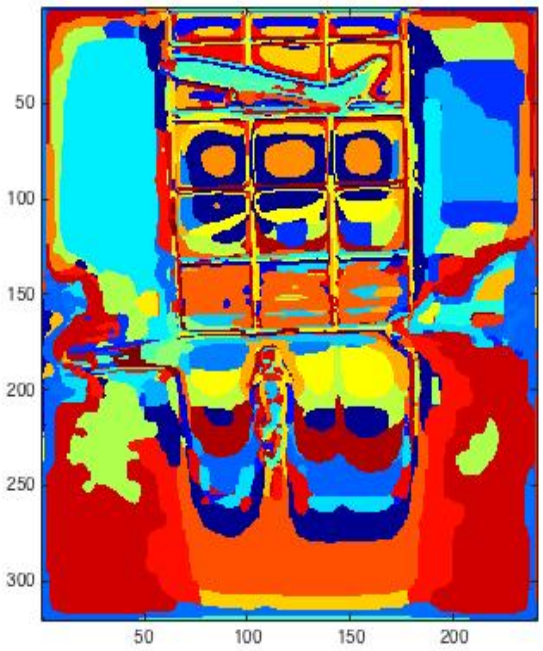

Figure 2. Original Image (Left) and it's corresponding wordMap (Right)

2.1. Implemented the function *getImageFeatures,* which returns a normalized histogram of the wordMap. This histogram is used as the feature vector for this image, which is compared with histograms of other images to find the closest match.
Verified that the histograms are normalized by taking a sum over the histogram.
2.2. Completed the function *getImageFeaturesSPM,* according to the description in the handout. Verified that the histogram sum up to 1.
Implemented helper function *checkPaddedRequired.m,* which pads the images so that it is easy to divide into different blocks for calculating histograms.
Some performance improvement can be achieved by using some dynamic programming tricks, i.e. to first compute the values at layers with smaller parts and using it to find values for bigger image parts. The performance with the current implementation was not a major time-consumer though, but this approach would work better with larger layerNum values.

2.3. Since we're using nearest neighbor approach. *distanceToSet* Calculates the distance of the histograms (image features) with all the other training images to find the nearest match. Used *repmat* to get performance benefit.

2.4. Implemented the script, *batchToVisualWords.m,* and tested it with the *guessImage* function.

2.5. The confusion matrix on the test data:

5	3	2	2	1	0	4	3
4	9	1	0	4	0	2	0
1	6	7	1	1	1	2	1
2	2	1	8	1	1	3	2
1	3	3	0	10	0	3	0
4	2	4	2	0	5	2	1
2	2	0	3	1	2	7	3
1	1	1	2	1	0	1	13

The overall accuracy of the system was 40%.
Since the expected accuracy of the system was 54%, all effort was used to look for errors in the code. The system gave the same accuracy using just a plain histogram (without SPM), also gave the same accuracy. The system gave an accuracy of 100%, when used with the training dataset as part of the debugging process. Even after multiple code walk-throughs, I still found out the reason for this accuracy and I'll seek help to understand the problem.

2.6. As you can see from the results, classes 1 and 6 were the most difficult classes to recognize. The classes 1 and 6 are airports and football_stadiums. If you browse through the images from these datasets, you can see that images from these categories have straight and long lines characteristic of their structures. These structures can be visualized using edges and

this filterbank although uses derivatives of Gaussian filters which can detect edges, they are from a horizontal and a vertical orientation only. It would help if there were more filters from different orientations. Filterbanks like LMFilterBank (library in custom folder), can be used to get a better accuracy with such cases.