Shastri Ram
shastir
16-720 Intro to Computer Vision
Spatial Pyramid Matching for Scene Classification

The Gaussians serve to blur the images. They remove noise and smoothen the image. The Laplacians are blob detectors, so they detect circular and elliptical-like features of the image. It can also help to detect edges. The x-derivative filter detects the vertical edges of the image. The y-derivative filter detects the horizontal edges of the image.

Extracting filter responses.

Below shows the original image and then the montage of the 20 filters applied to the image.

Figure 1: Original Image

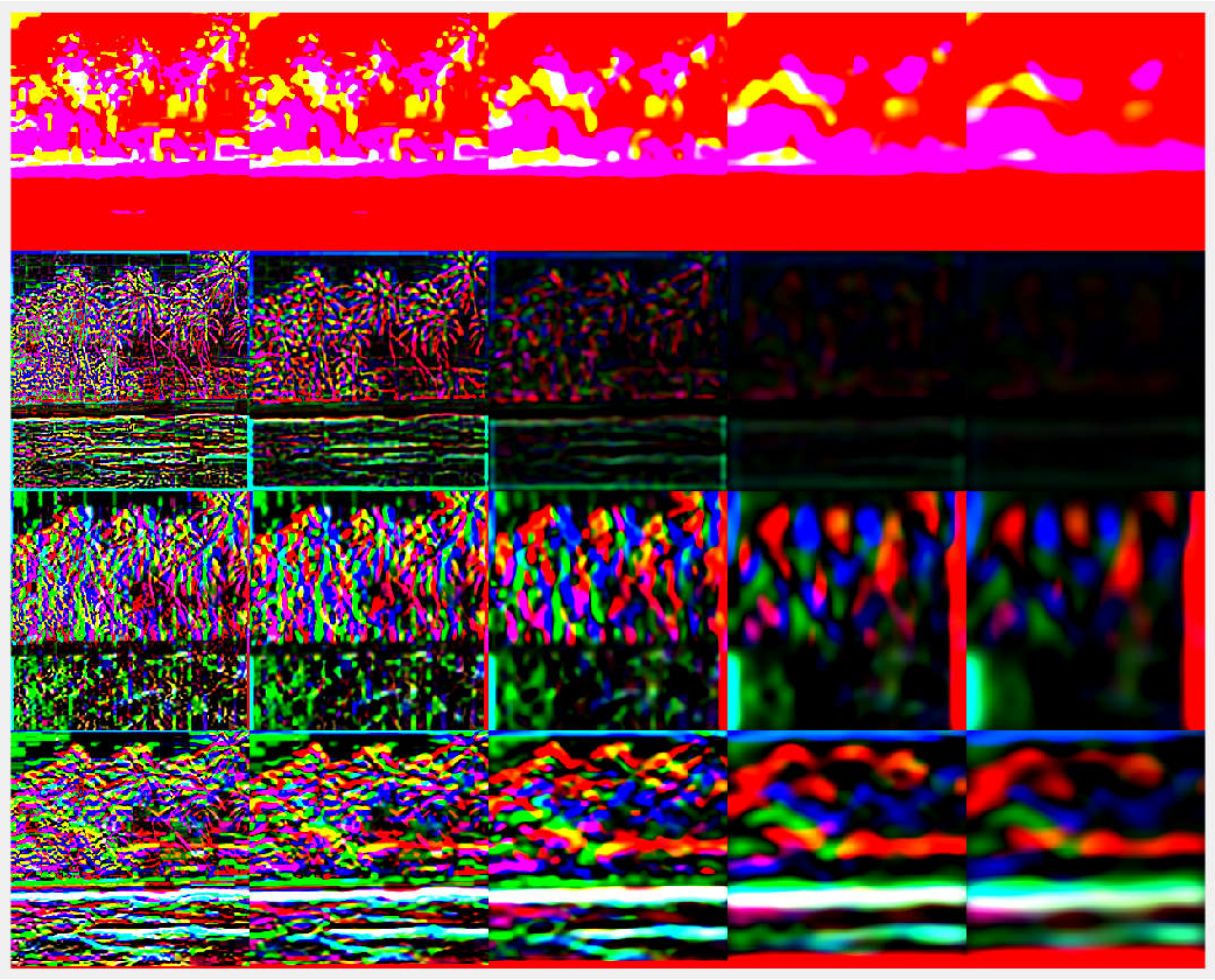


Figure 2: Montage of the 20 filters applied to the

Computing visual words.

Below are three images and their wordMaps from the beach folder.

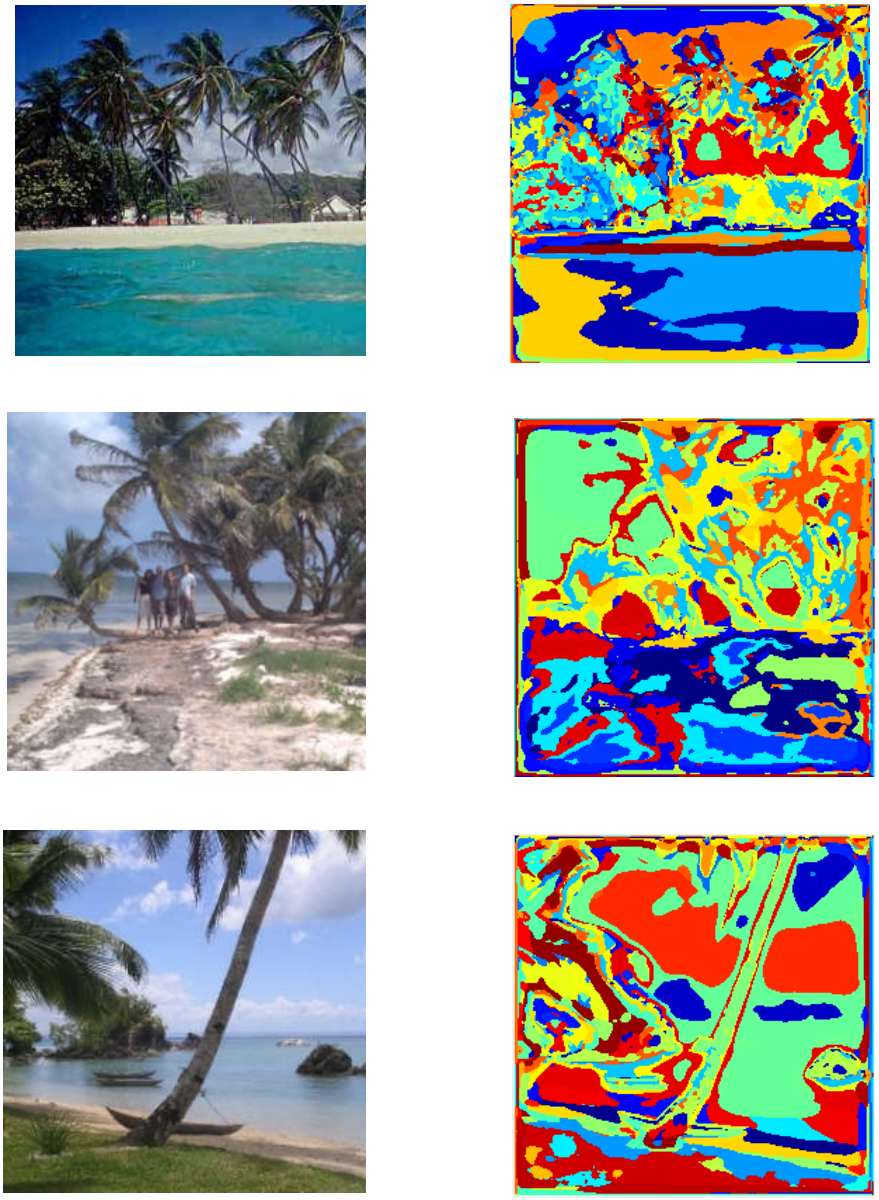


Figure 3: Beach images and their corresponding wordMap

The figure above shows the original image juxtaposed with its wordMap. It can clearly be seen that the wordMap is capturing semantic meaning of the scene. For example in the first and third wordMaps, the sea itself is shown with one or two clearly defining words. Similarly, the coconut trees in all the images also have words which consistently describe them wordMap. Furthermore, it can be seen that there are labels (colours) which are consistent between all three images. For example the dark blue, light blue, red, green and yellow labels are very common among the three images. It gives credibility and proof that the bag of words approach can identify scenes because one can image the histograms of these three images being very similar.

Quantitative evaluation

evaluateRecognitionSystem submitted in matlab folder.

Here alpha = 50, K =100

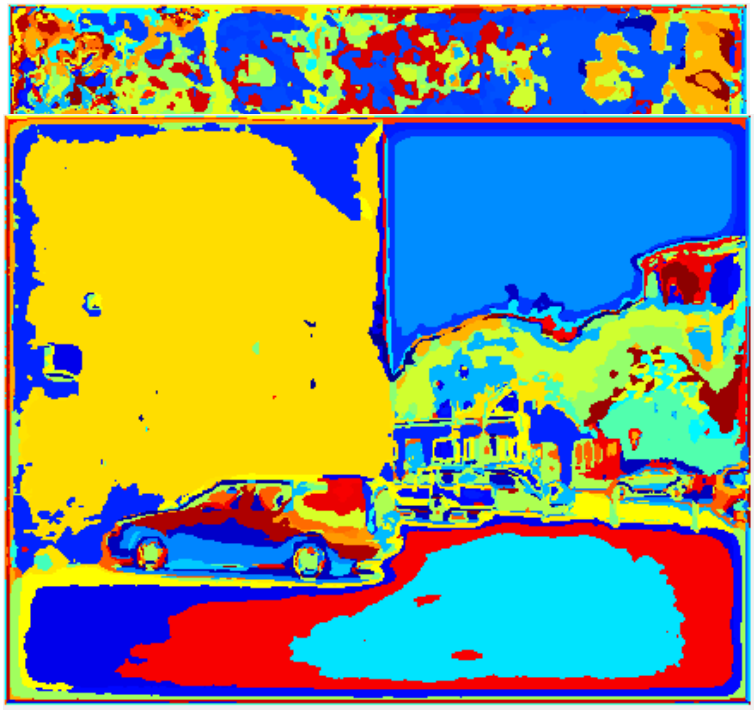
Confusion matrix:

15	1	1	0	1	0	1	1
0	12	1	1	4	0	2	0
1	2	9	3	0	0	3	2
2	1	1	12	0	0	3	1
1	2	1	3	11	1	1	0
0	1	3	1	1	11	2	1
0	1	1	2	3	1	11	1
1	1	1	0	1	6	5	5

Accuracy: 53.75%

Find out the failed cases.

Inspection of the confusion matrix above reveals that the most misclassified scene was the waterfall scene. Only 5 out of the 20 were correct. Most of the waterfall images were classified as park and parking lot. The following images show the wordMap for a park, parking lot and waterfall respectively.



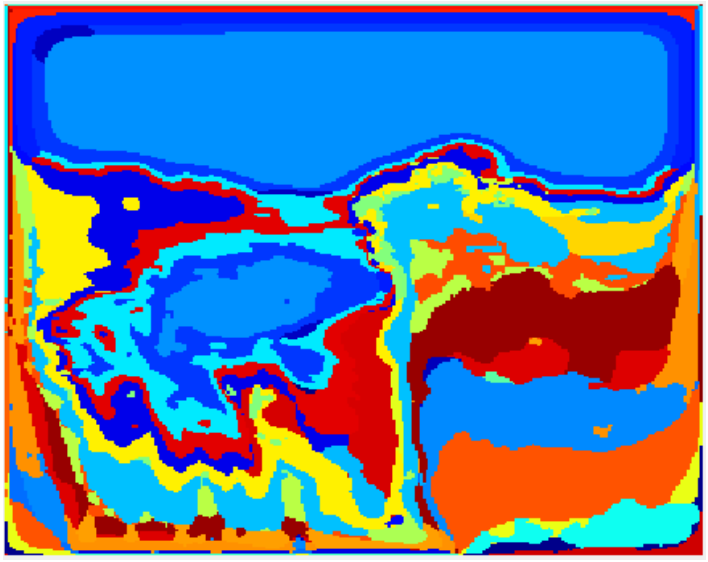


Figure 6: Waterfall wordMap

Analysis of these images could give some insight into the reason for the misclassification of the waterfall images. The colours in the wordMaps indicate the labels or dictionary words for the image. The image feature is obtained by creating a histogram of the words of the wordMap. Now it can be seen that there are labels/colours which occur in the waterfall wordMap that also occur very frequently in the wordMaps of the park and the parking lot. In particular, the dark blue, light blue, red, orange and yellow seem to occur in similar amounts in the wordMaps.

Though spatial pyramid matching is supposed to help with this by taking in consideration the spatial layout of the image, it is not inconceivable that there would be some images of a parking lot and park which could have a similar spatial layout as a waterfall. Hence the waterfall could be misclassified as these.

A solution for this problem would be to introduce more filters into the filterBank to help each class gain some form of uniqueness. This would be explored in the extra credit section.

Improving Performance

Note: All of the code for this section is in the custom folder. I have included the best dictionary and vision system, named dictionary_a_100_K_150_filters.mat and vision_a_100_K_150_filters.mat respectively. I have also included the code for libsvm as well as its vision system named vision_SVM_filters.mat

Step 1: Finding optimum alpha and K.

It was very clear that the accuracy of the classification depended the value of alpha, the number of sample points used, and K, the dictionary size. In general I thought that the more sample points and more words used to describe the image, the more accurate the classification would become. On my initial run, I did forgot to use weighting with the spatial pyramid matching and I got an accuracy of 54.37%. This is indicated by the orange row. I then implemented the weighting and experimented with different values of alpha and K to get the best accuracy. All of

the other rows of the table were the results obtained when the weighting was used. The following table shows my results.

Alpha (# of points)	K (Dictionary Size)	Accuracy
100	100	54.37
100	100	53.75
100	150	61.25
150	150	54.37
150	200	60.00

As it can be seen from the table, the highest accuracy was obtained with an alpha of 100 and a K of 150. As the alpha and K values were increased, the accuracy fluctuated but did not pass the highest. I would have liked to explore more values with a smaller resolution but time constraints prevented this from happening.

It was interesting to note that results went against my intuition that I previously mentioned. I thought that having more points and more word would lead to higher accuracy, but the testing results prove otherwise. It could be that as the alpha and K increased, the bag of words lost the generalization power. Having too much words to describe the image can make it difficult to match similar images.

The confusion matrix for alpha = 100 and K = 150 is shown below

```
    16     1     0     0     1     1     0     1
     0    12     2     1     4     0     1     0
     0     3    10     2     1     1     1     2
     1     0     1    13     0     1     1     3
     1     2     0     2    13     0     2     0
     0     1     0     0     2    17     0     0
     0     1     1     1     2     1    12     2
     0     1     1     0     2     7     4     5
```

Step 2: Interest points

Having found the best alpha and K given my resources, I began to think about the nature of the random points. Having learnt about interest detectors in class for the last few lectures, I thought that if the alpha points selected were points of interest instead of random points, it might have better accuracy. As such I implemented a Harris corner detector. The code is harrisCornerDetect.m. I used the same alpha and K, but the accuracy was 53.13%, the lowest yet. As such I did not bother to pursue this any further. It speaks to the power of randomness. Getting random points better describe the image as opposed to Harris corner points. The Harris corner

points could be tightly clustered in a small area whereas the random points would be distributed throughout the image and hence would capture more semantic and spatial information. As such I decided to not pursue this method any further.

Step 3: Machine Learning

Machine learning is all the buzz nowadays so I decided to try to use some machine learning techniques to see if the accuracy could be improved. I used libsvm library created by Chih-Chung Chang and Chih-Jen Lin from the National University of Taiwan. The package can be included on top of the existing code easily, which just changes in the buildRecognitionSystem and evaluateRecognitionSystem code. The code for this part is named buildRecognitionSystem_SVM and evaluateRecognitionSystem_SVM. To run it, you will need to run computeDictionary then batchToVisualWords, then buildRecognitionSystem_SVM and finally evaluateRecognitionSystem_SVM.

The library has the ability to use different types of SVMs and different kernel types. As such I just experimented with different SVM types and different kernels to see which would give the best accuracy. The results are as follows:

- nu-SVR SVM, RBF kernel- 59.38%
- one class SVM, RBF kernel- 59.38%
- C-SVC SMV,RBF Kernel- 59.38%
- nu-SVR SMV, linear kernel- 59.38%
- nu-SVR SVM, polynomial kernel, degree 2- 59.38%

Interestingly, they all gave the same accuracy but were all still lower than the best I got so far. Furthermore, there was little variation with their confusion matrices. This was really strange to me. Without wanting to waste too much time, I decided to table this for a little while.

Step 4: More Filters

I thought back to Q2.6 and noticed that even with better alpha and K values, the last class still have the poorest classification accuracy. I decided to include some Gabor filters into the filter bank. I implemented Gabor filters with wavelength 2,3, and 4 with orientations of 0,30,60,60,120 and 150 degrees. This brought the filter bank count to 38. I then ran the entire pipeline with this new change and alpha = 100 and K = 150. The results are as follows:

Confusion Matrix:

```
    15     0     0     1     3     0     1     0
     0    13     1     0     3     0     3     0
     0     2    14     0     0     1     2     1
     1     0     1    17     1     0     0     0
     0     1     2     0    13     2     2     0
     0     1     2     1     0    14     2     0
     0     1     2     0     5     2    10     0
     1     0     0     0     0     7     4     8
```

Accuracy: 65 %

This was the highest accuracy so far. It could be seen that the waterfall class got better classification accuracy in particular. I still thought I could eke out a few more percentages of accuracy.

Step 5: Machine Learning Again

I decided to turn to machine learning again and try running libsvm one more time. This time my results were more promising. They are as follows:

- nu-SVR SVM, RBF kernel- 67.5%
- one class SVM, RBF kernel- 67.5%
- C-SVC SMV, RBF Kernel- 68.13%
- nu-SVR SMV, linear kernel- 68.13%
- nu-SVR SVM, polynomial kernel, degree 2- 61.88%

Confusion Matrix for C-SVC SMV, RBF Kernel:

17	0	2	0	0	0	1	0
0	10	1	1	1	0	7	0
1	5	11	1	1	1	0	0
0	0	0	17	0	0	2	1
2	1	1	2	14	0	0	0
0	1	2	0	0	15	1	1
0	1	1	1	5	1	10	1
0	1	3	0	0	0	1	15

Confusion Matrix for nu-SVR SMV, linear kernel:

16	0	2	1	0	0	1	0
0	10	1	1	1	0	7	0
1	6	10	1	1	1	0	0
0	0	0	17	0	0	2	1
2	1	1	2	14	0	0	0
0	1	1	0	0	16	1	1
0	1	0	1	5	1	11	1
0	1	3	0	0	0	1	15

Both confusion matrices are very similar. A point to note is that we now have very good classification for the waterfall class.

Part of why I think SVM works better is due to its implementation. SVM is a binary classifier but to implement multiclass classification, I had to check one class, in turn, against all others. In other words, it checks to see if the wordMap of the test image falls within a specified boundary for one specific class at a time. This is a better method of checking than looking for the nearest neighbours with a distance metric. As such, SVM is a better classifier and would yield better results. Another advantage of SVM is that there is less overfitting and it is more robust to noise (Source: http://condor.depaul.edu/ntomuro/courses/578/notes/SVM-overview.pdf).

In general, it seems like the RBF (Gaussian) kernel works best. RBF uses the squared Euclidean distance measure which as we have seen before produces good results. This may be another reason why it is the best performing kernel. The linear kernel may have worked so well because the data itself may be linearly separable.

There may be other parameters to adjust to get even better accuracy but having a very limited knowledge of machine learning, I did not want to waste my resources.

Sources:

- https://www.csie.ntu.edu.tw/~cjlin/libsvm/
- http://stackoverflow.com/questions/9041753/multi-class-classification-in-libsvm
- http://www.robots.ox.ac.uk/~az/lectures/ml/lect3.pdf
- http://condor.depaul.edu/ntomuro/courses/578/notes/SVM-overview.pdf
- https://en.wikipedia.org/wiki/Radial_basis_function_kernel