16-720 Computer Vision: Homework 1

Yanjia Duan

September 2020

Q1.1.1

(1) **Gaussian filter** blurs an image by doing a weighted sum over neighboring pixels. The bigger the sigma is, the more weights are put into the neighbors and the image becomes blurrier. Gaussian filter removes noise and sharp edges from the image and preserves the general shape of the image.

(2) **Laplacian of Gaussian filter** picks up sharp changes in both x and y directions, which are vertical and horizontal edges in the image.

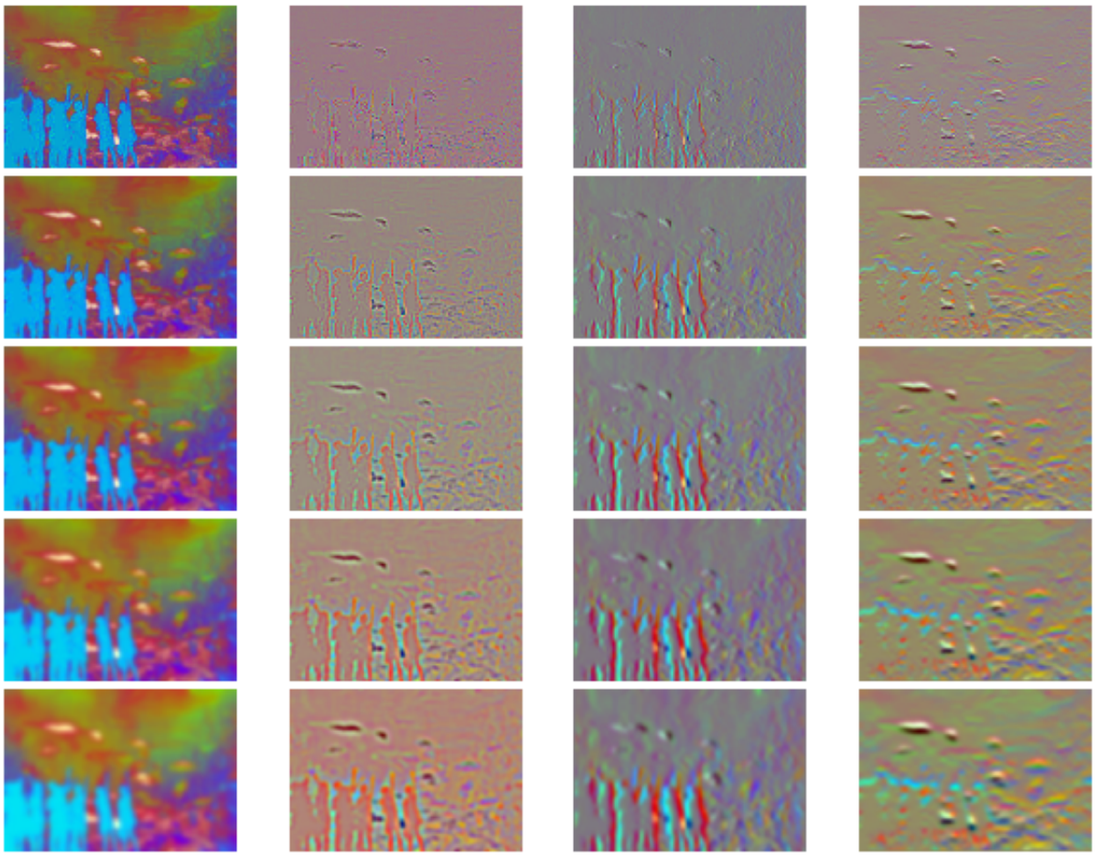
(3) **Derivative of Gaussian filter in the x direction** picks up sharp changes in the x direction, which are vertical edges in the image.

(4) **Derivative of Gaussian filter in the y direction** picks up sharp changes in the y direction, which are horizontal edges in the image.

We need different scales to find edges at different scales. Smaller scales find more detailed edges, while bigger scales find more general edges. Smaller scales preserve more details which is good in localizing edges. Bigger scales remove more noises which is better in detecting edges.

Q1.1.2

Filter responses of `aquarium/sun_aztvjgubyrgvirup.jpg` at scales [1, 2, 3, 4, 5]:



Q1.3 Computing Visual Words

The word boundaries are reasonable that objects of the same kind are labeled with the same color, with some exceptions that same objects with different colors are labeled differently. Different objects are labeled differently.

Since the texture and pixel values (colors) of same objects are similar, their filter responses are similar and therefore they belong to the same cluster centers. So on the wordmap, they have the same colors.

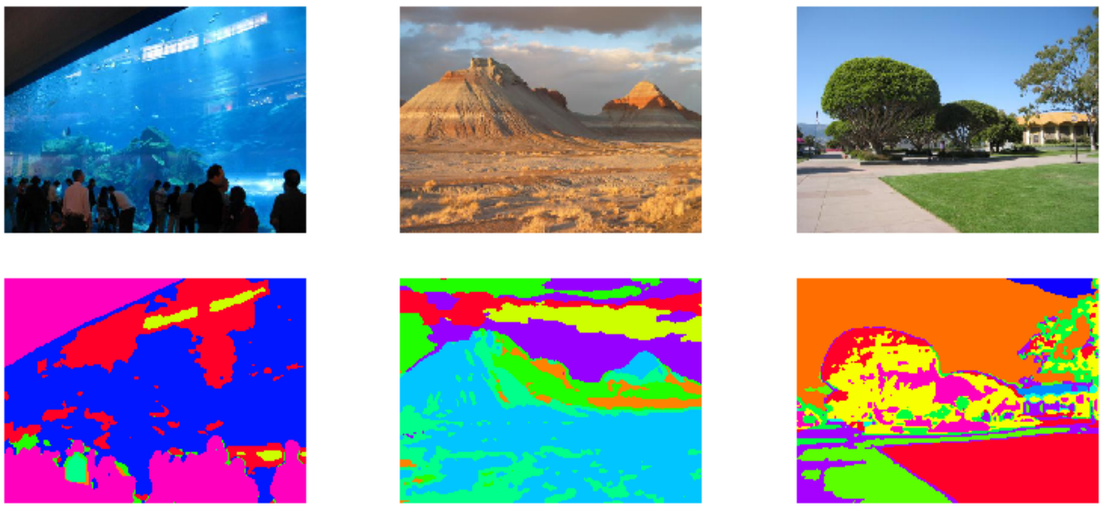


Figure 1: Original images and their wordmap visualizations.

Q2.5

Under default parameters `filter_scale = [1, 2], K = 10, alpha = 25, L = 3`:
Confusion Matrix:

```
[[25.  3.  3.  1.  2.  3.  5.  8.]
 [ 0. 22.  6. 13.  2.  0.  4.  3.]
 [ 1.  5. 22.  1.  2.  2.  8.  9.]
 [ 1.  4.  1. 34.  8.  1.  1.  0.]
 [ 1.  2.  1. 13. 18.  6.  5.  4.]
 [ 1.  0.  5.  1.  4. 29.  7.  3.]
 [10.  1.  2.  0.  7.  9. 17.  4.]
 [ 2.  2.  4.  2.  2.  5. 12. 21.]]
```

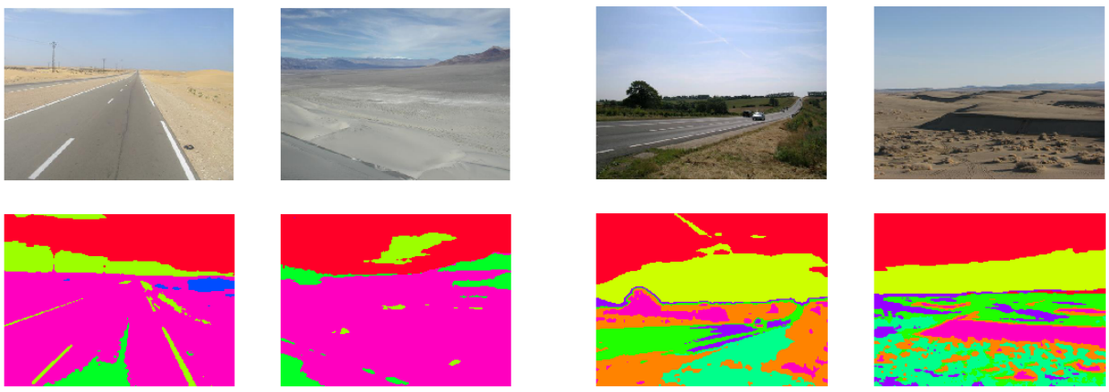
Accuracy:

```
0.47
```

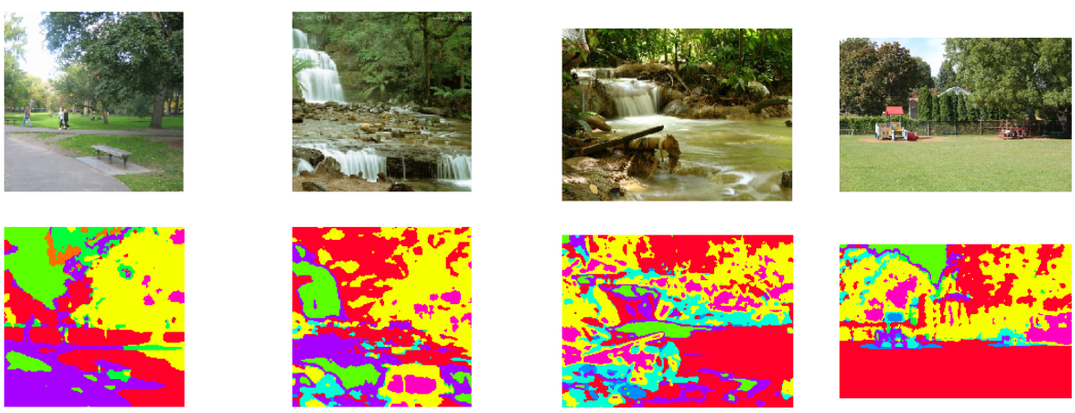
Q2.6 Find the failures

Some classes that are easily get confused are: desert and highway, park and waterfall, laundromat and kitchen.

Desert and Highway:



Park and Waterfall:



Laundromat and Kitchen:

The wordmaps of two images look similar: the color components are similar, which means that their histograms are similar; similar components are in similar positions, which means their Spatial Pyramid Matching representations are similar. Furthermore, the similarity score is computed as the nearest distance between two histograms, which is a 1-nearest-neighbor approach. So those test images are easily got misclassified.

Q3.1 Hyperparameter tuning

I experimented around `filter_scales`, `L` and `alpha`. The processes are shown below:

First, fix `alpha` and `K` to be small (`alpha=25, K=10`) for fast training and evaluation. Search over different combinations of `filter_scales` and `L`. Test accuracies are shown in Table 1.

`filter_scales` \ `L`	1	2	3	4
$[1, 2, 4, 8, 8\sqrt{2}]$	0.44	0.5375	0.545	0.54
$[1, 2, 4, 8, 12]$	0.48	0.52	0.5725	0.5925
$[1, 2, 4, 8, 16]$	0.46	0.5425	0.5525	0.535

Table 1: The test accuracy under different combinations of `filter_scale` and `K`, fixing `alpha=25` and `K=10`.

After finding out the best choice of `filter_scales` and `L`, which are `[1,2,4,8,12]` and `4`, I searched over different values of `alpha` with `K=200`. The results are shown in Table 2.

`alpha`	100	200	300
Accuracy	0.665	0.66	0.6675

Table 2: The test accuracy under different `alpha` values, fixing `filter_scale=[1,2,4,8,12], K=200` and `L=4`.

The best accuracy lies in `alpha=300`, and the **final results** are:

Confusion Matrix:

```
[[40.  0.  2.  1.  3.  2.  1.  1.]
 [ 0. 38.  4.  2.  2.  2.  1.  1.]
 [ 2.  2. 33.  0.  0.  2.  0. 11.]
 [ 3.  2.  1. 34.  9.  0.  0.  1.]
 [ 0.  2.  2. 12. 26.  4.  3.  1.]
 [ 1.  0.  3.  2.  1. 38.  2.  3.]
 [ 4.  1.  2.  2.  2.  7. 31.  1.]
 [ 2.  4. 11.  2.  0.  3.  1. 27.]]
```

Accuracy:

```
0.6675
```

Relationships of Parameters and Accuracy

Increasing the sigma of `filter_scales` will first increase the accuracy and then decrease the accuracy. Because as sigma becomes larger, the filter will remove more noises in the image and will detect more general edges. As sigma goes larger and larger, the filter will eventually blur the edges and make the detection poor.

Increasing the number of SPM layers `L` will significantly increase the accuracy, because more SPM layers take into account more sizes of blocks and include in the histogram spatial information of different ranges.

Increasing `alpha` will first increase the accuracy and then the accuracy flattens. Because when `alpha` is very small, only a small amount of pixels are sampled from each image, which are not very representative and omit a lot of details in the image. Increase the number of sampled pixels can represent the image features more. But when there are too many sampled pixels, noises will be included, which will decrease the classification accuracy.

Q3.2 Further Improvement

1. Histogram Distance Functions

I tried Cosine Distance, Cramer-von Mises Distance and Match Distance for measuring histogram similarity. Their functions and results are listed below:

(1) Cosine Distance: $D_{CO} = 1 - \sum_i h_1(i)h_2(i)$

Confusion Matrix:

```
[[31.  1.  5.  0.  1.  0.  5.  7.]
 [ 0. 33.  3.  3.  1.  1.  1.  8.]
 [ 1.  5. 17.  0.  0.  2.  4. 21.]
 [ 0. 27.  1.  9.  1.  1.  2.  9.]
 [ 0. 21.  3.  1. 11.  2.  3.  9.]
 [ 1.  9.  6.  0.  0. 19.  1. 14.]
 [ 6.  7.  4.  0.  1.  5. 18.  9.]
 [ 5.  6.  7.  0.  1.  3.  2. 26.]]
```

Accuracy: `0.41`

Evaluation Time: `220.45s`

(2) Cramer-von Mises Distance: $D_{CM} = \sum_i (h_1(i) - h_2(i))^2$

Confusion Matrix:

```
[[39.  0.  0.  2.  2.  0.  4.  3.]
 [ 1. 16.  2. 17.  8.  2.  1.  3.]
 [ 2.  1. 12.  4.  5.  7.  6. 13.]
 [ 8.  0.  0. 32.  7.  1.  2.  0.]
 [ 3.  0.  1. 13. 22.  6.  5.  0.]
 [ 3.  0.  1.  3.  3. 31.  8.  1.]
 [ 4.  1.  0.  4.  3.  8. 28.  2.]
 [ 1.  1.  7.  6.  4.  9.  6. 16.]]
```

Accuracy: `0.445`

Evaluation Time: `221.82s`

(3) Match Distance: $D_{MA} = \sum_i |h_1(i) - h_2(i)|$

Confusion Matrix:

```
[[42.  0.  1.  1.  2.  1.  2.  1.]
 [ 0. 33.  4.  6.  2.  3.  1.  1.]
 [ 1.  1. 32.  1.  0.  2.  0. 13.]
 [ 4.  2.  0. 35.  8.  1.  0.  0.]
 [ 0.  1.  2. 12. 26.  4.  5.  0.]
 [ 1.  0.  2.  2.  2. 38.  2.  3.]
 [ 4.  1.  1.  1.  3.  6. 33.  1.]
 [ 2.  2.  9.  2.  1.  4.  1. 29.]]
```

Accuracy: `0.67`

Evaluation Time: `232.53s`

The original histogram intersection approach has `0.6675` accuracy and costs 228.94s for evaluation. The Match Distance approach is slightly better than histogram intersection, which is within my expectation. A possible reason is that if a histogram value for two images are both very small, their L1 distance is small while their min is also small, which makes the Match distance to be small and intersection distance to be big. So Match distance approach is slightly better than intersection approach.

Cosine Distance and are worse than intersection distance, which is beyond my expectation. My guess is that when a histogram value is high in one image and low in another, their multiplication will be big, which means that the cosine distance is small, and that doesn't make sense. For Cramer-von Mises Distance, it is more accurate than Cosine Distance because it takes into account the relative size of each value.

2. Weights for SPM Layers

I tried the following two ways to assign weights. Their results are also listed.

(1) Assign exponentially decreasing weights. Layer l has weight 2^{-l-1}, except that layer $L-1$ has weight 2^{-L+1} (*e.g.*, in a four layer spatial pyramid, $L = 4$ and weights are set to $\frac{1}{2}$, $\frac{1}{4}$, $\frac{1}{8}$, $\frac{1}{8}$ for layer 0, 1, 2, 3).

Confusion Matrix:

```
[[39.  0.  1.  2.  2.  0.  4.  2.]
 [ 0. 24.  5. 12.  4.  3.  0.  2.]
 [ 1.  1. 29.  0.  2.  2.  2. 13.]
 [ 3.  2.  0. 36.  7.  1.  0.  1.]
```

```
 [ 0.  0.  1. 16. 26.  2.  5.  0.]
 [ 0.  0.  3.  2.  2. 37.  4.  2.]
 [ 3.  1.  1.  2.  3. 10. 30.  0.]
 [ 2.  2. 13.  2.  1.  5.  1. 24.]]
```

Accuracy:

```
0.6125
```

(2) Use uniform weights $\frac{1}{L}$ for all layers.

Confusion Matrix:

```
[[42.  0.  1.  1.  2.  1.  2.  1.]
 [ 0. 31.  4.  7.  3.  3.  1.  1.]
 [ 1.  1. 32.  1.  0.  2.  0. 13.]
 [ 4.  2.  0. 35.  8.  1.  0.  0.]
 [ 0.  1.  2. 14. 24.  4.  5.  0.]
 [ 1.  0.  2.  2.  2. 37.  3.  3.]
 [ 4.  1.  1.  1.  4.  6. 32.  1.]
 [ 2.  2. 10.  2.  1.  4.  1. 28.]]
```

Accuracy:

```
0.6525
```

Both results are not as good as using exponentially increasing weights, which are consistent with my expect. In fact, putting more weights on coarser layers will result in less spatial information, because coarser layers have big blocks and are difficult to locate objects.