EE 569 Homework 4

Chenxin Xia 2838929158
chenxinx@usc.edu

1. Texture Analysis and segmentation

1.1. Texture classification --- feature Extraction

1.1.1. Method

1.1.1.1.Feature Extraction

To eliminate potential errors caused by every texture images' illuminance, before extract features from the 36 labeled texture images, we will make every train image subtract their own global mean.
After elimination of illuminance influence, each image will be applied 25 Law's Filters. The filters are given below:

L5	[1 4 6 4 1]
E5	[-1 -2 0 2 1]
S5	[-1 0 2 0 -1]
W5	[-1 2 0 -2 1]
R5	[1 -4 6 -4 1]

- L5: Low pass filter, cut-off frequency $\pi/2$
- E5: Band pass filter, cut-off frequency $\pi/10$, $7\pi/10$
- S5: Band pass filter, cut-off frequency $\pi/5$, $4\pi/5$
- W5: Band pass filter, cut-off frequency $3\pi/10$, $9\pi/10$
- R5: High pass filter, cut-off frequency $\pi/2$

There are 25 filters: L5L5 L5E5 L5S5 L5W5 L5R5 E5L5 E5E5 E5S5 E5W5 E5R5 S5L5 S5E5 S5S5 S5W5 S5R5 W5L5 W5E5 W5S5 W5W5 W5R5 R5L5 R5E5 R5S5 R5W5 R5R5.
Each filter is generated by the rule below:

25 2D filters

- $L5L5^T$
- $L5E5^T$
- $L5S5^T$
- …

$$\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix} \times \begin{bmatrix}-1 & -2 & 0 & 2 & 1\end{bmatrix} = \begin{bmatrix}-1 & -2 & 0 & 2 & 1\\-4 & -8 & 0 & 8 & 4\\-6 & 12 & 0 & 12 & 6\\-4 & -8 & 0 & 8 & 4\\-1 & -2 & 0 & 2 & 1\end{bmatrix}$$

An example of $L5E5^T$

1.1.1.2.Feature averaging

Then we will average the absolute value of each filter to generate a 25D feature vector for each image. Since some pair of filter results will generate close mean result will be combined to build a new 15-D feature vector. They are:
L5L5 L5E5/E5L5 L5S5/S5L5 L5W5/W5L5 L5R5/R5L5 E5E5 E5S5/S5E5 E5W5/ W5E5 E5R5/R5E5 S5S5 S5W5/W5S5 S5R5/R5S5 W5W5 W5R5/R5W5 R5R5

1.1.1.3.Feature Reduction

Principal component analysis (PCA)
PCA is a useful tool for feature dimensions reduction. It could bring out the maximum ratio of inter-class variance and intra-class variance.

I have programmed this algorithm by myself

- First centralize the feature matrix which make every feature mean to 0;
- Second calculate covariance matrix of the matrix above;
- Third find the eigen-vectors and its corresponding eigen-values to the covariance matrix.
- The result of PCA is constructed by the product of the largest n eigen-values and their eigen-vectors.

```
function result = pca_hand(ori,n)
    [hight,~] = size(ori);
    mean_ori = mean(ori);
    k = ones(hight,1);
    mean_ori = k*mean_ori;
    ori=ori-mean_ori;
    %centerlize original feature matrix
    mid=cov(ori);

    [vector,value]=eig(mid);
    [~,order]=sort(diag(value),'descend');
    vector=vector(:,order);
    T=vector(:,1:n);
    result=ori*T;

    for i = 1:n
        min1 = min(min(result(:,i)));
        max1 = max(max(result(:,i)));
        if abs(min1) > max1
            result(:,i) = -1*result(:,i);
        end
    end
end
```

1.1.2. Results

1.1.2.1. Result with L5L5 normalization

Since all kernel have a zero-mean except L5L5, then we could use the energy of L5L5 to do normalization in each 15-D feature vector to make every elements in the vector to divide L5L5.

15-D results: the feature dimension are followed by the order of :
L5L5 L5E5/E5L5 L5S5/S5L5 L5W5/W5L5 L5R5/R5L5 E5E5 E5S5/S5E5 E5W5/W5E5 E5R5/R5E5 S5S5 S5W5/W5S5 S5R5/R5S5 W5W5 W5R5/R5W5 R5R5
Training image result:

1	0.25068	0.21604	0.27832	0.57488	0.088824	0.096526	0.13995	0.25732	0.12216	0.18309	0.32683	0.27311	0.47665	0.81417
1	0.26971	0.24116	0.32019	0.66135	0.099509	0.11233	0.16488	0.30169	0.14405	0.21576	0.38389	0.32245	0.55657	0.96182
1	0.33207	0.25632	0.25648	0.31782	0.085802	0.047075	0.040419	0.067729	0.0276	0.025865	0.044367	0.025971	0.045501	0.085142
1	0.35109	0.27741	0.26708	0.36257	0.083959	0.05161	0.052165	0.097972	0.034384	0.035923	0.066145	0.038056	0.071316	0.13859
1	0.32015	0.24388	0.24349	0.34152	0.08062	0.048371	0.04832	0.091252	0.031914	0.033154	0.060598	0.034697	0.062977	0.11867
1	0.33178	0.25498	0.25496	0.33276	0.085111	0.048952	0.046147	0.083337	0.030823	0.030862	0.05511	0.031615	0.056627	0.10684
1	0.039631	0.018991	0.015074	0.02194	0.0085124	0.0047458	0.0044229	0.00716	0.0028372	0.0028756	0.0048136	0.0030883	0.0052247	0.0089762
1	0.068702	0.037301	0.031978	0.051499	0.015217	0.009262	0.0095753	0.017196	0.0060494	0.0067832	0.01241	0.0079526	0.014398	0.02643
1	0.054497	0.027886	0.022429	0.033403	0.012562	0.0070943	0.0065273	0.011128	0.0042631	0.004451	0.0080621	0.0049952	0.0090845	0.016243
1	0.1935	0.12212	0.11997	0.193	0.018085	0.011655	0.012199	0.023006	0.0073035	0.0076967	0.014846	0.0082532	0.01655	0.035103
1	0.18709	0.1209	0.12115	0.22005	0.022658	0.013965	0.014793	0.032074	0.0083028	0.0084888	0.017717	0.0083387	0.016624	0.032625
1	0.32353	0.22379	0.20008	0.38962	0.030846	0.018141	0.017452	0.032147	0.010445	0.0099012	0.018747	0.0095148	0.018886	0.038553
1	0.13826	0.10333	0.1212	0.2406	0.022705	0.015477	0.017382	0.03777	0.010854	0.012316	0.026869	0.014157	0.031403	0.070638
1	0.19136	0.12634	0.12214	0.19635	0.012267	0.0086444	0.01015	0.022519	0.0059041	0.0069343	0.015731	0.0081303	0.018546	0.042148
1	0.15964	0.1105	0.11517	0.19768	0.021088	0.014494	0.016357	0.03541	0.010105	0.011537	0.025399	0.013294	0.02943	0.06613
1	0.22034	0.17227	0.21847	0.48306	0.033718	0.024642	0.02897	0.065404	0.017889	0.021031	0.047884	0.024991	0.057261	0.13349
1	0.25217	0.16037	0.16431	0.2771	0.042232	0.026051	0.030471	0.06193	0.012817	0.013366	0.029515	0.011987	0.025182	0.048784
1	0.1937	0.11789	0.11226	0.18948	0.038011	0.022526	0.020463	0.033618	0.012486	0.010815	0.017603	0.0093665	0.015902	0.028647
1	0.22627	0.12323	0.10512	0.14721	0.059781	0.034674	0.030964	0.045413	0.020513	0.019024	0.0293	0.018267	0.029555	0.051901
1	0.2339	0.13233	0.11536	0.16945	0.066054	0.03997	0.036514	0.055689	0.02487	0.023346	0.037146	0.022809	0.038011	0.067466
1	0.20909	0.11656	0.10443	0.16167	0.057789	0.03482	0.033094	0.053735	0.022066	0.021769	0.036628	0.022074	0.038187	0.069055
1	0.23776	0.13969	0.13034	0.2102	0.069007	0.043676	0.04293	0.072397	0.028495	0.028739	0.049731	0.029406	0.052256	0.095578
1	0.21041	0.11245	0.098783	0.15615	0.056903	0.033863	0.031932	0.052483	0.021694	0.021473	0.036393	0.022154	0.039214	0.073629
1	0.24548	0.14348	0.13202	0.21031	0.072449	0.045764	0.044071	0.072081	0.02993	0.029577	0.049704	0.030232	0.052796	0.098306
1	0.23692	0.13782	0.12668	0.2033	0.06772	0.042667	0.042054	0.070669	0.027742	0.028108	0.049025	0.029101	0.052185	0.097397
1	0.18566	0.095812	0.079433	0.11656	0.049236	0.028519	0.026102	0.040833	0.01763	0.017012	0.027927	0.017016	0.028954	0.052014
1	0.18987	0.099904	0.083348	0.11907	0.051871	0.030476	0.027761	0.041415	0.018963	0.017883	0.027797	0.017461	0.028501	0.050796
1	0.13619	0.060024	0.046492	0.067112	0.032055	0.017223	0.015007	0.023297	0.01036	0.0098551	0.016514	0.010104	0.018036	0.034643
1	0.14528	0.068334	0.054949	0.082226	0.035377	0.019309	0.017273	0.027815	0.011574	0.011106	0.018921	0.011209	0.020085	0.038505
1	0.1609	0.074707	0.057904	0.084919	0.043914	0.024354	0.021062	0.032694	0.015256	0.014624	0.024295	0.015442	0.027226	0.053047
1	0.16274	0.077197	0.05881	0.080674	0.042676	0.022821	0.018799	0.027377	0.013279	0.01196	0.018719	0.011705	0.019585	0.035627
1	0.17134	0.082545	0.066063	0.096294	0.045341	0.025273	0.022286	0.035011	0.015382	0.014704	0.02466	0.014948	0.026188	0.049317
1	0.16263	0.05991	0.042471	0.059204	0.030405	0.015064	0.012816	0.019329	0.0086458	0.0080061	0.012922	0.0078291	0.01341	0.024541
1	0.15585	0.057235	0.040558	0.055939	0.028593	0.014067	0.011782	0.017695	0.0079844	0.0072707	0.011626	0.0070473	0.011894	0.02133
1	0.16601	0.060988	0.042686	0.059683	0.032197	0.015673	0.013198	0.020057	0.0088748	0.0082221	0.01337	0.0080425	0.01372	0.024649
1	0.14665	0.052427	0.036585	0.050683	0.026847	0.012954	0.010939	0.016738	0.0073571	0.0068536	0.0113	0.0067993	0.01186	0.021958

Test image result:

1	0.23323	0.13056	0.11681	0.18123	0.065856	0.039732	0.037648	0.060555	0.025129	0.024541	0.040888	0.024717	0.042627	0.078419
1	0.33861	0.25884	0.25021	0.30871	0.084479	0.047557	0.043682	0.074855	0.028328	0.027995	0.049557	0.028547	0.050906	0.094075
1	0.056369	0.0282	0.024604	0.037735	0.012952	0.0075454	0.0075225	0.012331	0.0047375	0.005096	0.0084517	0.0059523	0.0098587	0.016761
1	0.26731	0.20626	0.24312	0.46653	0.039977	0.026313	0.030207	0.063981	0.01494	0.016008	0.033074	0.016788	0.034637	0.07111
1	0.14953	0.070108	0.055316	0.081531	0.037056	0.0198	0.017558	0.028068	0.011611	0.011015	0.01875	0.010986	0.019632	0.036818
1	0.22633	0.12768	0.11589	0.17875	0.064196	0.0393	0.037233	0.058997	0.025059	0.024512	0.04012	0.024531	0.041911	0.076699
1	0.1662	0.12525	0.14975	0.30232	0.023371	0.015764	0.017416	0.037119	0.01086	0.012106	0.025968	0.013753	0.029993	0.065902
1	0.14128	0.064656	0.04948	0.069642	0.035298	0.019051	0.016087	0.024059	0.011248	0.01043	0.016712	0.010489	0.017858	0.032975
1	0.16336	0.061752	0.045197	0.065452	0.032909	0.016636	0.014313	0.022276	0.009879	0.009376	0.01561	0.0096059	0.016984	0.032311
1	0.20999	0.12882	0.13609	0.32352	0.041822	0.027266	0.030465	0.068578	0.017198	0.018366	0.0404	0.018502	0.039086	0.080709
1	0.33995	0.28378	0.31449	1.038	0.21169	0.18418	0.17092	0.21952	0.16014	0.14746	0.18269	0.13643	0.18435	0.4169
1	0.22311	0.12757	0.11429	0.17421	0.062664	0.038512	0.036682	0.058306	0.024374	0.023851	0.038788	0.023832	0.040263	0.073317

PCA result of train image

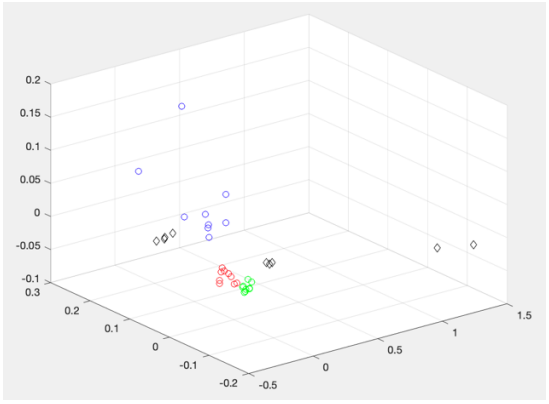
1.0267	-0.1781	0.0034706
1.2461	-0.19886	0.0018327
0.108	0.23031	-0.049927
0.19607	0.23865	-0.051256
0.15209	0.20389	-0.033271
0.14027	0.22169	-0.0466
-0.25578	-0.18023	0.049212
-0.21121	-0.15044	0.038903
-0.23692	-0.16628	0.042039
-0.090973	0.036483	0.033039
-0.077763	0.047083	0.050227
0.054942	0.25818	0.050514
-0.04258	0.0069159	0.08988
-0.084361	0.036079	0.037947
-0.064254	-0.0010425	0.05055
0.17398	0.1886	0.1611
-0.0011456	0.12476	0.021046
-0.092536	0.032778	0.019813
-0.076486	0.010931	-0.044025
-0.041945	0.022158	-0.044611
-0.054635	-0.0037652	-0.026741
0.013851	0.034017	-0.032006
-0.056169	-0.011538	-0.029291
0.018534	0.03845	-0.039243
0.0097179	0.027704	-0.034288
-0.10888	-0.042377	-0.026951
-0.10579	-0.03565	-0.030446
-0.17703	-0.10258	-0.0055216
-0.16028	-0.087326	-0.0063689
-0.13857	-0.084224	-0.022371
-0.15748	-0.073746	-0.022734
-0.13115	-0.066138	-0.024035
-0.18915	-0.089829	-0.022366
-0.1961	-0.09429	-0.018271
-0.18758	-0.087715	-0.025224
-0.20154	-0.10454	-0.014026

Test image

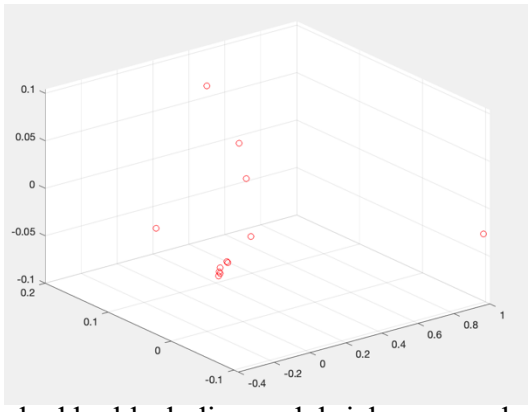
-0.074726	0.017359	-0.053167
0.1194	0.1723	-0.060441
-0.32065	-0.1045	0.038021
0.19289	0.11261	0.10426
-0.23343	-0.041177	-0.012361
-0.080109	0.013178	-0.049189
-0.0097279	0.0036518	0.087237
-0.2499	-0.047153	-0.010722
-0.25266	-0.036174	-0.01919
0.032483	0.0035785	0.04763
0.96327	-0.10685	-0.024745
-0.086824	0.013169	-0.047329

Plot of PCA result:

Training image



testing image



At here in the training image, blankets are marked by black diamond, bricks are marked by blue circle, grass are marked by red circle and rice are marked by green circle.
When calculating the discriminant power, I utilize the ratio of (inter-class variance)/(intra-class variance), the result is shown below:

```
0.15644    0.48704    0.64533    0.55317    0.66371    0.47162    0.34966    0.34635    0.31961    0.27829    0.27634    0.25674    0.25556    0.2546
```

Order of features are L5E5/E5L5 L5S5/S5L5 L5W5/W5L5 L5R5/R5L5 E5E5 E5S5/S5E5 E5W5/ W5E5 E5R5/R5E5 S5S5 S5W5/W5S5 S5R5/R5S5 W5W5 W5R5/R5W5 R5R5

So for features after normalization E5E5 has the strongest discriminant power and L5E5/E5L5 has the weakest power.

1.1.2.2. Result without L5L5normalization

15-D results: the feature dimension are followed by the order of :
L5L5 L5E5/E5L5 L5S5/S5L5 L5W5/W5L5 L5R5/R5L5 E5E5 E5S5/S5E5 E5W5/ W5E5 E5R5/R5E5 S5S5 S5W5/W5S5 S5R5/R5S5 W5W5 W5R5/R5W5 R5R5
Training image result:

```
1135.8   284.72   245.38   316.12   652.94   100.89   109.63   158.96   292.26   138.75   207.95   371.21    310.2   541.38   924.73
2757.5   743.71      665   882.91   1823.6   274.39   309.74   454.65   831.91   397.21   594.94   1058.6   889.13   1534.7   2652.2
3841.1   1275.5   984.56   985.17   1220.8   329.58   180.82   155.25   260.16   106.02   99.352   170.42   99.757   174.78   327.04
3300.2   1158.7   915.49   881.41   1196.6   277.08   170.32   172.16   323.33   113.47   118.55   218.29   125.59   235.36   457.38
3604.7     1154   879.11   877.68     1231   290.61   174.36   174.18   328.93   115.04   119.51   218.43   125.07   227.01   427.75
3696.3   1226.4   942.47   942.41     1230   314.59   180.94   170.57   308.04   113.93   114.07    203.7   116.86   209.31   394.93
8910.2   353.12   169.22   134.31   195.49   75.847   42.286   39.409   63.797    25.28   25.622   42.891   27.517   46.553    79.98
7961.1   546.95   296.96   254.58   409.99   121.15   73.736    76.23    136.9    48.16    54.002   98.798   63.312   114.63   210.41
8684.9    473.3   242.19    194.8    290.1    109.1   61.614   56.689    96.65   37.025   38.656   70.019   43.383   78.898   141.07
1651.7   319.61    201.7   198.15   318.77   29.871   19.251   20.149   37.998   12.063   12.712   24.521   13.632   27.335   57.979
7803.2   1459.9   943.37   945.39   1717.1    176.8   108.97   115.43   250.28   64.788    66.24   138.25   65.069   129.72   254.58
  1845   596.93    412.9   369.16   718.87   56.912   33.471   32.199   59.312   19.271   18.268    34.59   17.555   34.845   71.133
9062.9     1253   936.51   1098.4   2180.6   205.77   140.27   157.53   342.31   98.372   111.62   243.51   128.31    284.6   640.19
4716.6   902.58   595.88   576.09   926.08   57.858   40.772   47.872   106.21   27.847   32.706   74.198   38.347   87.472    198.8
3254.5   519.53   359.63   374.82   643.33   68.629   47.171   53.235   115.24   32.885   37.546   82.661   43.266    95.78   215.22
608.61    134.1   104.85   132.96      294   20.521   14.997   17.631   39.806   10.887     12.8   29.143    15.21    34.85   81.242
2741.8    691.4    439.7    450.5   759.77   115.79   71.428   83.546    169.8   35.141   36.648   80.926   32.867   69.045   133.76
4945.3    957.9   583.02   555.15   937.04   187.98    111.4   101.19   166.25   61.746   53.481   87.054    46.32   78.642   141.67
5738.8   1298.5   707.17   603.29   844.81   343.07   198.98   177.69   260.61   117.72   109.17   168.14   104.83   169.61   297.84
5462.5   1277.7   722.84   630.18    925.6   360.82   218.34   199.46    304.2   135.85   127.53   202.91   124.59   207.64   368.53
5634.6   1178.1    656.8   588.44   910.93   325.62    196.2   186.47   302.78   124.33   122.66   206.39   124.38   215.17    389.1
4940.9   1174.7   690.18   644.02   1038.6   340.96    215.8   212.11   357.71   140.79      142   245.72   145.29   258.19   472.24
5663.6   1191.7   636.87   559.46   884.36   322.27   191.78   180.85   297.24   122.87   121.61   206.11   125.47   222.09      417
4962.7   1218.3   712.02   655.16   1043.7   359.54   227.11   218.71   357.71   148.53   146.78   246.66   150.03   262.01   487.86
5003.7   1185.5   689.59   633.85   1017.3   338.85   213.49   210.42   353.61   138.81   140.64   245.31   145.61   261.12   487.35
6261.1   1162.4   599.89   497.34   729.81   308.27   178.56   163.43   255.66   110.38   106.51   174.85   106.54   181.28   325.67
6210.8   1179.2   620.49   517.66   739.54   322.16   189.28   172.42   257.22   117.77   111.07   172.64   108.45   177.01   315.49
6738.7   917.75   404.49    313.3   452.25   216.01   116.06   101.13   156.99   69.812   66.411   111.28    68.09   121.54   233.45
6383.5   927.37   436.21   350.77   524.89   225.83   123.26   110.26   177.56    73.88   70.893   120.78   71.552   128.22    245.8
6169.5   992.67   460.91   357.24   523.91   270.93   150.26   129.94   201.71   94.123   90.224   149.89   95.272   167.97   327.28
6403.2   1042.1   494.31   376.57   516.57   273.26   146.13   120.37    175.3   85.028   76.585   119.86   74.948   125.41   228.13
6034.5     1034   498.12   398.66   581.09   273.61   152.51   134.48   211.27   92.826   88.732   148.81   90.205   158.03   297.61
6874.6     1118   411.86   291.97      407   209.02   103.56   88.101   132.88   59.436   55.038   88.835   53.822    92.19   168.71
6958.9   1084.6   398.29   282.24   389.28   198.97   97.892   81.993   123.13   55.562   50.596   80.906   49.041   82.768   148.43
6877.3   1141.7   419.43   293.56   410.46   221.43   107.78   90.765   137.94   61.034   56.546    91.95   55.311   94.358   169.52
7035.7   1031.8   368.86    257.4   356.59   188.89   91.143   76.966   117.76   51.762   48.219   79.505   47.838   83.445   154.49
```

Testing image result:

5358.5	1249.8	699.6	625.91	971.14	352.89	212.9	201.74	324.48	134.65	131.5	219.1	132.45	228.42	420.21
3786.8	1282.2	980.17	947.51	1169	319.91	180.09	165.41	283.46	107.27	106.01	187.66	108.1	192.77	356.24
8719.5	491.51	245.89	214.54	329.03	112.94	65.792	65.593	107.52	41.309	44.435	73.695	51.901	85.963	146.15
1243.9	332.51	256.57	302.42	580.33	49.728	32.731	37.575	79.587	18.584	19.912	41.141	20.882	43.086	88.454
6465.4	966.78	453.27	357.64	527.13	239.58	128.02	113.52	181.47	75.07	71.215	121.23	71.026	126.93	238.04
5396.8	1221.4	689.06	625.43	964.67	346.45	212.09	200.94	318.39	135.24	132.29	216.52	132.39	226.19	413.93
3252.4	540.55	407.36	487.06	983.29	76.014	51.272	56.645	120.73	35.32	39.375	84.459	44.73	97.551	214.34
6746	953.08	436.18	333.79	469.81	238.12	128.52	108.52	162.3	75.877	70.36	112.74	70.757	120.47	222.45
6702.8	1094.9	413.91	302.95	438.71	220.58	111.51	95.936	149.31	66.217	62.846	104.63	64.386	113.84	216.58
3757.9	789.1	484.1	511.41	1215.7	157.16	102.46	114.49	257.71	64.63	69.018	151.82	69.528	146.88	303.29
2605.1	885.59	739.26	819.25	2704.2	551.47	479.81	445.25	571.87	417.17	384.13	475.92	355.42	480.24	1086
5388.6	1202.3	687.43	615.88	938.77	337.67	207.52	197.66	314.19	131.34	128.52	209.01	128.42	216.96	395.08

PCA result of train image

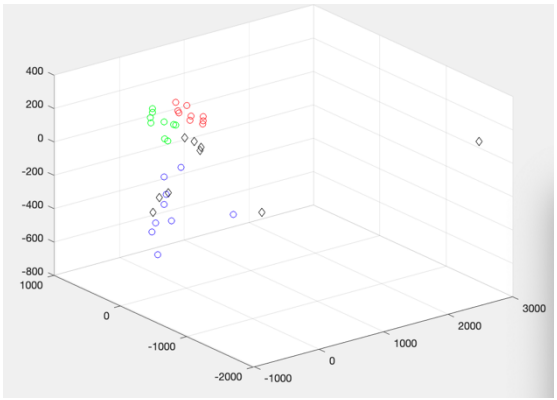
86.201	-1057.5	-162.81
2720.2	-1763	115.39
568.65	575.53	-65.179
583.72	344.76	-79.686
578.01	358.01	-102.23
602.44	464.47	-70.371
-989.06	-457.29	-157.47
-623.86	-337.91	-100.4
-808.69	-381.01	-104.52
-934.84	-391.53	-294.75
781.73	842.57	-311.85
-514.75	-36.923	-350.37
1342.2	599.24	-610.45
-113.16	192.42	-261.6
-468.27	-228.98	-305.68
-1023	-561.28	-389.98
-348.72	-7.3926	-261.29
-93.355	247.31	-168.25
161.16	315.98	238.15
286.91	274.42	213.71
231.25	170.1	151.11
412.15	153.14	110.42
218.42	139.69	182.15
447.89	177.67	146.72
405.73	141.8	132.19
14.86	129.32	222.76
33.72	159.57	230.69
-432.08	-92.814	157.69
-351.37	-60.793	128.37
-252.14	-83.168	219.35
-308.47	36.036	221.32
-203.59	-3.6122	202.93
-472.57	51.856	291.3
-516.26	39.041	269.19
-458.04	66.774	311.01
-562.99	-16.571	252.42

Test image

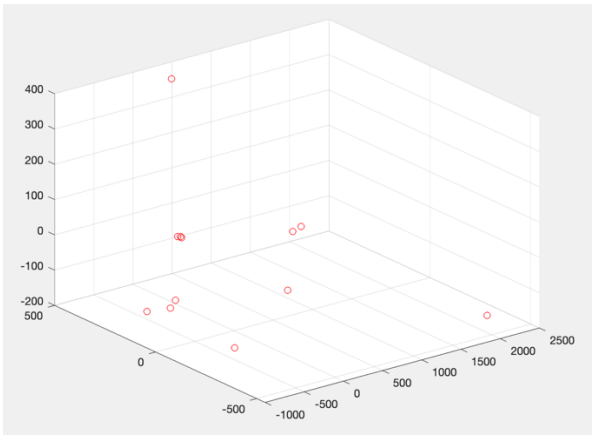
223.74	366.53	-44.077
455.02	485.57	358.36
-846.19	-331.44	-110.6
-709.79	-541.53	100.69
-449.14	116.22	-117.59
207.6	340.59	-39.312
-213.44	-416.43	217.32
-516.2	112.79	-135.07
-547.17	218.37	-169.61
130.51	-240.81	137.39
2095.9	-439.83	-166.41
169.13	329.97	-31.097

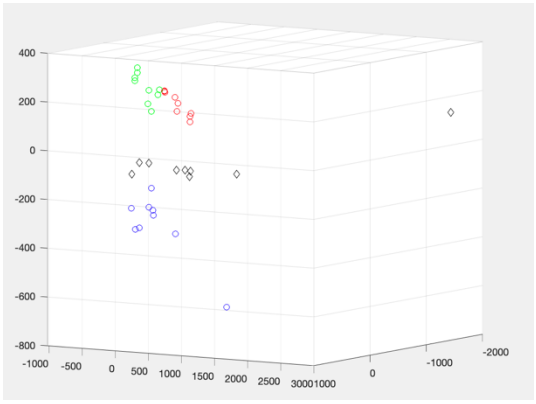
Plot of PCA result:

Training image



testing image





another view point of training image

At here in the training image, blankets are marked by black diamond, bricks are marked by blue circle, grass are marked by red circle and rice are marked by green circle.

When calculating the discriminant power,
I utilize the ratio of (inter-class variance)/(intra-class variance), the larger the better.
The result is shown below:

```
0.41669     0.17044     0.23466     0.25069     1.7589      1.1342      0.56529     0.38593     0.43883     0.27287     0.24211     0.21612     0.20552     0.19263
```

Order of features are L5E5/E5L5 L5S5/S5L5 L5W5/W5L5 L5R5/R5L5 E5E5 E5S5/S5E5 E5W5/W5E5 E5R5/R5E5 S5S5 S5W5/W5S5 S5R5/R5S5 W5W5 W5R5/R5W5 R5R5

We can find if we do not apply L5L5 normalization, E5E5 has the strongest discriminant power and L5S5/S5L5 has the weakest power.

1.1.3. Discussion

Normalization of vectors by L5L5 may change the results a lot, it could influence the discriminant ability of each dimension.

1.2. Advanced Texture Classification --- Classifier Explore

1.2.1. Method

1.2.1.1. Kmeans

Kmeans is an unsupervised learning

(1) Random find k center position

(2) Cluster elements to the nearest center

(3) Update center based on the clustered elements. The new center is a mean of elements clustered to the old center

(4) Repeat (1)-(3) until center positions does not change

I have programmed the kmean program by myself, here is the program:

```
function result = kmean(set,k)
    [num,dimension] = size(set);
    label = zeros(num,1);
    center = zeros(k,dimension);
    d = floor(num/k);
    for i = 1:k
        center(i,:) = set(i*d,:);
    end
    flag = 1;
    while flag == 1
        center_update = zeros(k,dimension);
        center_account = zeros(k,1);
        label_update = zeros(num,1);
        for i = 1 : num
            distance = 3276700000000000;
            mark = 0;
            d = set(i,:);
            for j = 1:k
                p = d - center(j,:);
                p = p .* p;
                p = sum(sum(p));
                if p < distance
                    distance = p;
                    mark = j;
                end
            end
            center_update(mark,:) = center_update(mark,:)+ d;
            center_account(mark) = center_account(mark) +1;
            label_update(i) = mark;
        end
        for j = 1:k
            center_update(j,:) = center_update(j,:)/center_account(j);
        end
        center = center_update;
        mark = 0;
        for j = 1:num
            if label(j) ~= label_update(j)
                mark = 1;
            end
        end
        label = label_update;
        if mark == 0
            flag = 0;
        end
    end
    result = label';
end
```

1.2.1.2. Random forest

I have copied the method of random forest from my submitted homework 2.

Random forest is a classifier which is constructed by a mount of sub classifiers which vote to get the classify result. It has some good characters; it could deal with high dimensional sample and can handle missing date and indicate the most important feature. The random forest gathers decisions made by decision trees and integrate them to make a final decision.

Construction:

If the size of training data is N.

- Training decision tree with n randomly selected n bootstrap samples from the training data set.
- If the sample has M different characters, when training the decision node to construct branches in the decision tree, select m characters randomly from the M characters where m<<M. Then use several methods to choose one character of m to construct this node.

- The every inter node are all established on step ii until cannot generate more branches. Notice there are no pruning to the decision tree.
- Establish amount of decision trees

In this homework I will establish 100 trees.

1.2.1.3. SVM

SVM is a method to find a hyperplane between two categories.

In this problem we will use SVM four times to make the classification.

There are four set for svm training

- Is blanket or not blanket
- Is brick or not brick
- Is grass or not grass
- Is rice or not rice

After training of the four above, for each test image I will make it to do four matches with the classifier. The match with best matching score is the test image's category.

1.2.2. Result

The test images are:

1 6 12 grass
4 7 10 brick
2 3 11 blankets
5 8 9 rice

1.2.2.1. Results with L5L5 normalization

Test	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Error rate
#1	11	2 4 7 10	3 5 8 9	1 6 12	16.7%
#2	1 6 7 10 12	3 5 8 9	2 4	11	33.4%
#3	1 2 4 6 7 10 12	11	3	5 8 9	41.6%
#4	2 4	1 6 7 10 12	11	3 5 8 9	33.4%
14-D Kmeans with L5L5 normalization					

Test	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Error rate
#1	1 6 7 10 12	3 5 8 9	11	2 4	33.4%
#2	3 5 8 9	11	2 4	1 6 7 10 12	33.4%
#3	11	1 6 7 10 12	2 4	3 5 8 9	33.4%
3-D Kmeans with L5L5 normalization					

Test	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Error rate
14-D RF	2 3 11	1 6 12	5 8 9	4 7 10	0%
3-D RF	2 3	1 6 12	5 8 9 11	4 7 10	8.3%
14-D SVM	2 3 11	1 6 12	5 8 9	4 7 10	0%
3-D SVM	2 3 11	1 6 12	5 8 9	4 7 10	0%

1.2.2.2. Results without L5L5 normalization

1.2.2.2.1. Doing PCA on train and test image features separately

Test	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Error rate
#1	2 7 10 11	4	3 5 8 9	1 6 12	25%
#2	1 5 6 8 9 12	2 4 7 10	3	11	33.4%
#3	2 7 10 11	3	1 5 6 8 9 12	4	50%
#4	3 5 8 9	11	1 2 6 10 12	4 7	25%
14-D Kmeans without L5L5 normalization					

Test	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Error rate
#1	3 4 7	1 2 6 10 12	11	5 8 9	25%
#2	7 10	11	1 2 6 10 12	3 4 5 8 9	33.4%
#3	3 4 7	11	1 2 6 10 12	5 8 9	25%
3-D Kmeans without L5L5 normalization					

Test	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Error rate
14-D RF	2 3 11	1 6 12	5 8 9	4 7 10	0%
3-D RF	1 3 4 5 6 8 11 12	9	2 10	7	75%
14-D SVM	2 3	1 6 11 12	5 8 9	4 7 10	8.3%
3-D SVM	1 3 4 6 12	5 8 9 11	2	8 10	83.3%

1.2.2.2.2. Doing PCA on train and test image features together

Test	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Error rate
#1	2 7 10 11	4	3 5 8 9	1 6 12	25%
#2	1 5 6 8 9 12	2 4 7 10	3	11	33.4%
#3	2 7 10 11	3	1 5 6 8 9 12	4	50%
#4	3 5 8 9	11	1 2 6 10 12	4 7	25%
14-D Kmeans without L5L5 normalization					

Test	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Error rate
#1	11	4 7 10	3 5 8 9	1 2 6 12	13.7%
#2	3 4 7	1 2 6 10 12	5 8 9	11	25%
#3	2	1 6 10 12	11	3 4 5 7 8 9	33.3%
3-D Kmeans without L5L5 normalization					

Test	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Error rate
14-D RF	2 3 11	1 6 12	5 8 9	4 7 10	0%
3-D RF	2 3	1 6 12 11	5 8 9	4 7 10	8.3%
14-D SVM	2 3	1 6 11 12	5 8 9	4 7 10	8.3%
3-D SVM	2 3	1 6 11 12	5 8 9	4 7 10	8.3%

1.2.3. Discussion

When comparing k-means to RF and SVM, k-means is the worst algorithm. Since it is original center sensitive algorithm. So this is not robust. Under the same condition, RF and SVM may generate better results than k-means.

The error rate from 3-D PCA by K-mean is more stable than that of 14-D K-mean, since PCA could find the best hyper-plane to maximum inter class variance and minimize intra class variance.

But k-means has an advantage over RF and SVM, this is an unsupervised learning which does not need training samples.

For L5L5 normalization, since different image has different characters, so normalization is of vital importance. Comparing results obtained from 1.2.2.1 and 1.2.2.2 normalized features can get better results.

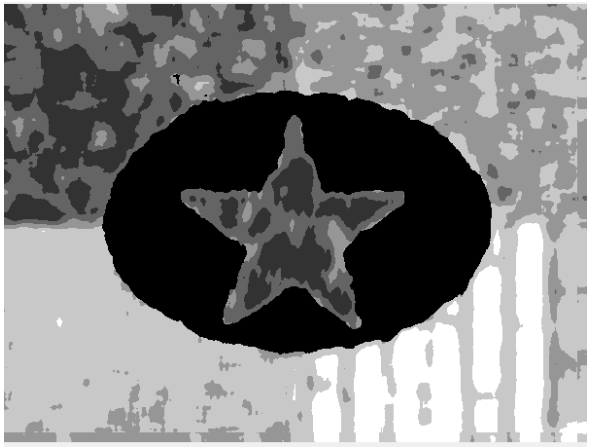
For PCA over train and test images, it should operate over train and test image together. In 1.2.2.1 since all features are normalized, the projection hyperplane of train images and test images' feature are not different so they could get good results. But for features without normalization, the two hyperplanes have huge different. We can find in 1.2.2.2 the k-means result of 3D error rate as usual, but error rate for RF and SVM is really high. And for the results generated by doing PCA on train and test image together, the error rate is fairly low. We can conclude doing PCA over train and test image is needed.

1.3. Texture Segmentation

1.3.1. Method

- Apply Law's filters comp.raw to obtain 15-D features,
- Computer energy measure with a N*N window,
- Energy normalization with corresponding L5L5
- K-mean to cluster the image into 6 colors(0,51,102,153,204,255)

1.3.2. Result



N = 17

N = 25

N = 33

N = 35

N = 39

N = 43

1.3.3. Discussion

The size of window will influence the output of segmentation image. Texture are sensitive for window size. For example, when window size is small, like 17 then part cross the star man be marked black monotonously. But for the brick part when window size is large to 43, some part of it cannot even be recognized.

Also when window is large, some sharp angle may not able to be shown. When window size is larger than 25, the angles of stars will be like oval. When window size is small, some texture cannot be classified in the same category, and is the window size is too large, there will be great distortion of edge. In all a proper size of window is needed. For this problem, N = 39 or N = 41 can be a good choice.

This method cannot deal with texture which has a sparse texture, for example bricks.

1.4. Advanced Texture Segmentation

1.4.1. Method

- Apply Law's filters comp.raw to obtain 15-D features,
- Apply PCA to the 15-D features,
- Computer energy measure with a N*N window,
- K-mean cluster,
- Using deep first searching to find out small holes,
- Change the color of the small holes

1.4.2. Result

At here I applied 3-D PCA and the window size N = 37.

Original image after k-mean:

Post-processing to merge small holes:

1.4.3. Discussion

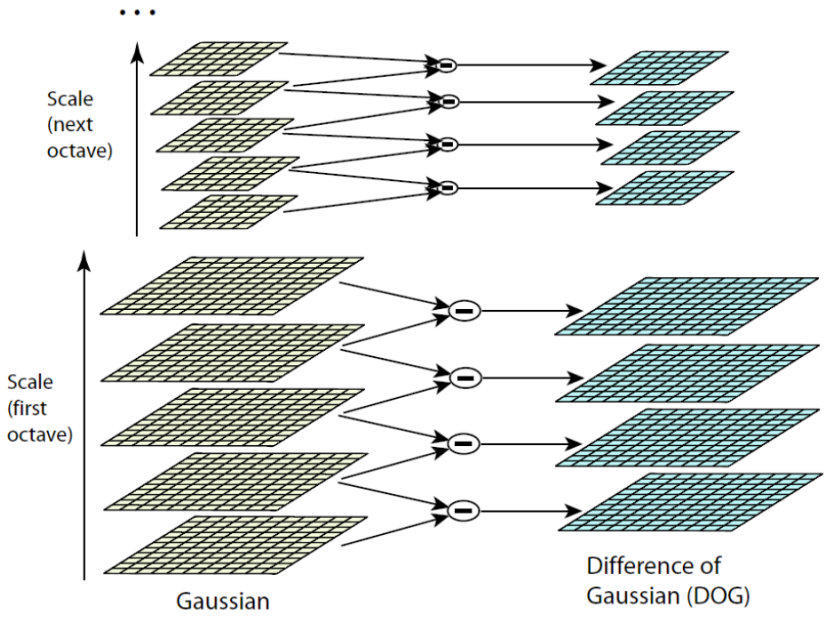
Because the right part brightness of brick patch is different greatly from left part, the K-means method over features extracted by Law's filter cannot perform well. Although 3-D PCA could generate the largest inter class variance. To enhance the result, we may able to apply other kind of filter to deal with bricks park or apply CNN.

2. Image Feature Extractors

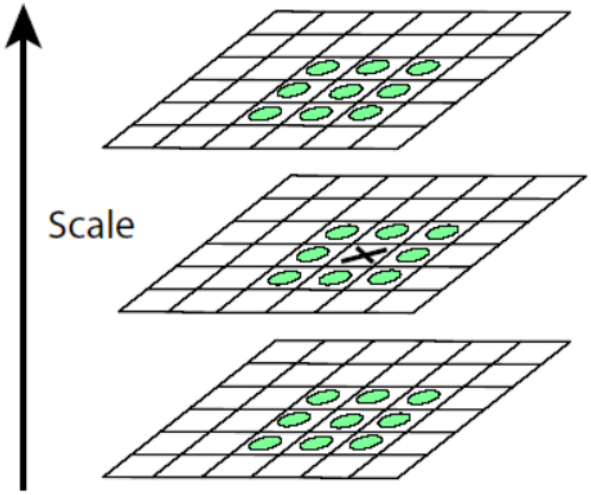
SIFT

For sift there are four steps

- Scale-space extrema detection
  The goal of this step is to find all potential interest point. To find out these interest points, we will use Laplacian-of-gaussian second derivative filters. Images are scaled in size smoothed by gaussian window to construct a set of octaves. Each octave is used to do generate DOG. The way is shown below.



The interest point is selected when the point is a minimum or a maximum of nearby points.



- Keypoint localization
  The goal of this step is to eliminate low contrast candidate and edge location.
  eliminate low contrast candidate

  Talyor expansion: $D(\mathbf{x}) = D + \frac{\partial D^T}{\partial \mathbf{x}}\mathbf{x} + \frac{1}{2}\mathbf{x}^T \frac{\partial^2 D}{\partial \mathbf{x}^2}\mathbf{x}$, where $\mathbf{x} = (x, y, \sigma)^T$

  Taking derivative, the location of extrema is $\hat{\mathbf{x}} = -\frac{\partial^2 D^{-1}}{\partial \mathbf{x}^2}\frac{\partial D}{\partial \mathbf{x}}$

  $D(\hat{\mathbf{x}}) = D + \frac{1}{2}\frac{\partial D^T}{\partial \mathbf{x}}\hat{\mathbf{x}}$

  Discard extrema with $|D(\hat{\mathbf{x}})| < 0.03$

  Eliminate edge location

  Use ratio of principle curvatures to throw out poorly defined peaks

  because poorly defined peaks has a large principal curvature across the edge but a small one in the perpendicular direction.

  The principle curvatures comes from Hessian matrix: $\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$

  Compute eigenvalues of H: $\alpha \; and \; \beta$, where $\alpha > \beta$

  Let $\alpha = r\beta$, then $\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r+1)^2}{r}$ depends only on r

  Throw keypoints that have a ratio greater than $r = 10$

- Orientation assignment
  The goal if this is to achieve rotation invariant.
  Compute central derivatives, gradient magnitude and direction or scale of keypoint.

  $$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$

  $$\theta(x, y) = tan^{-1}((L(x+1, y) - L(x, y-1))/(L(x+1, y) - L(x-1, y)))$$

  A histogram is formed by quantizing the orientations into 36 bins

  Peaks in the histogram correspond to the orientations of the patch

  Compute relative orientation and magnitude in a 16×16 neighborhood at key point

- Keypoint descriptor
  - The magnitude is weighted by a Gaussian window
  - Form weighted histogram (8 bin) for 4×4 regions by summing the gradient magnitudes near that direction within the region
  - Distribute each sample to adjacent bins by trilinear interpolation
  - Concatenate 16 histograms in one long vector of 128 dimensions
  - The feature vector is normalized to unit length to reduce effect of illumination change
  - Cap each element to be 0.2, normalize again to reduce non-linear illumination change

2.1. Salient Point Descriptor

2.1.1. Question answers

2.1.1.1.

This method is robust to scaling, translation and rotation. These are methods of geometric modification. It is also partially invariant to affine distortion, change in 3D viewpoint and illumination.

2.1.1.2.

Scaling: when using gaussian filter to smooth original image, the image pyramid is built which is scaled in image size, as result the key point is kept. This algorithm is robust to scaling.

Translation: Key points' output vector is based on their neighborhood region, which makes the location of key point can't affect the output vector. This algorithm is robust to translation.

Rotation: The key point is selected if and only if the point is the local maximum or local minimum, so rotation could not change the discovery of key point. The histogram is formed by quantizing the orientations in to 36 bins, the main orientation if computed by the maximum in the histogram. Then compute relative orientation and magnitude in a 16*16 neighborhood at key point. As the information above, this algorithm can compute local maximum or local minimum, and find their corresponding orientation-based feature vector. This is robust to rotation.

Affine distortion change in 3D : As we can find, this algorithm can produce a large dimension of vector which could tolerant some tiny errors.

Illumination is shown in 2.1.1.3

2.1.1.3.

The feature vector is normalized to unit length, so it cannot be affected by illumination change. This is robust to illumination change.

2.1.1.4.

DOG is used to approximate LOG, because the implementation of DOG needs less run-time complexity than LOG. It could speed the algorithm.

2.1.1.5.

In the paper published in 2004, the best results are achieved with a 4*4 array of histograms with 8 orientations. So the output vector size is 4*4*8=128.

2.2. Image matching

2.2.1. Method

2.2.1.1.

Using SIFT to find out key-points in Husky_3 and Husky_1. Find the maximum scale key-point in Husky_3 and match it to the key-points in Husky_1.
The match is based on the minimum Euclidean distance.

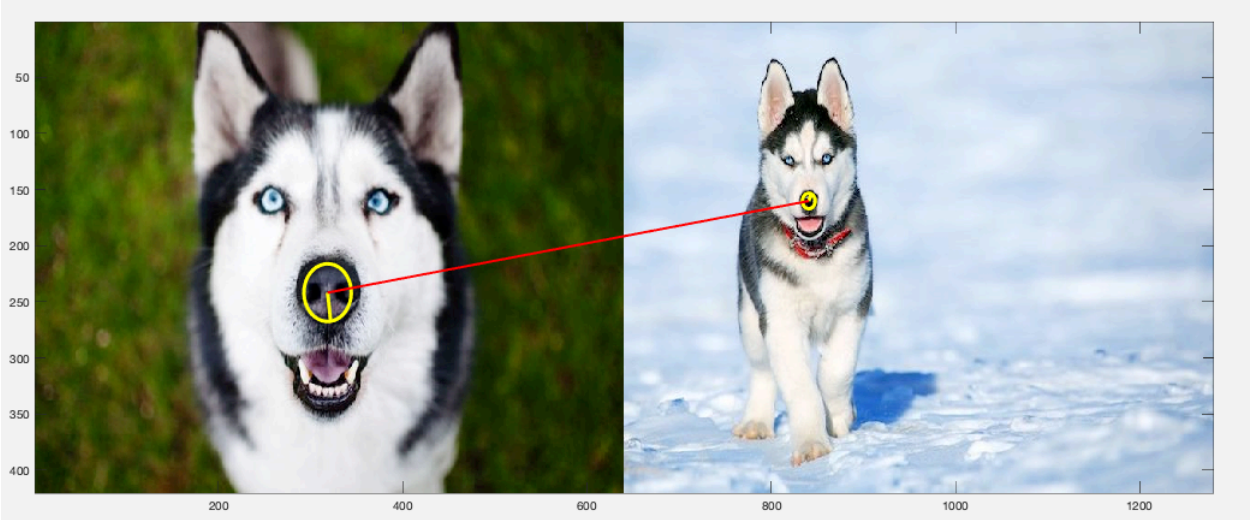
2.2.1.2.

Doing SIFT matching between Husky_1 and Husky_3, Husky_3 and Husky_2, Husky_3 and Puppy_1, Husky_1 and Puppy_1

2.2.2. Result

2.2.2.1.

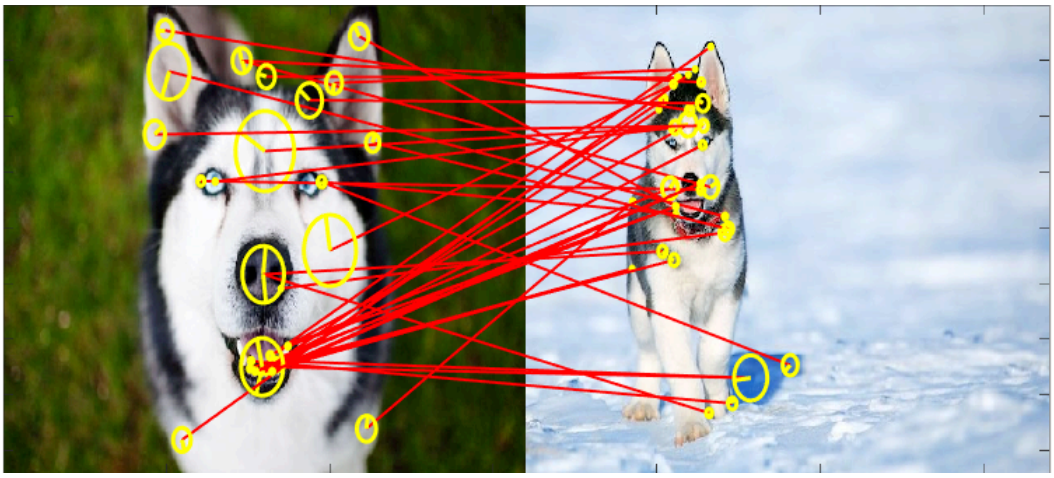
When applying the SIFT program the parameter I have chosen are: the number of octaves of the DOG scale space equal to 7 and peak selection threshold equal to 9. Then the result is shown below:



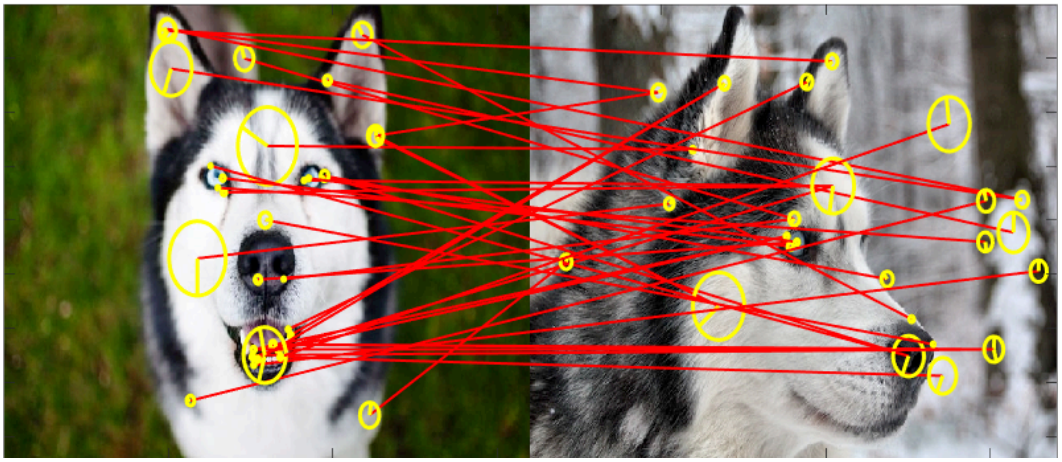
These two matching points have different scale and different orientation, though they are different in scale and different in orientation, they are the closet pair in the two images. Since they are different dog under different environment, this kind of difference is acceptable.

2.2.2.2.

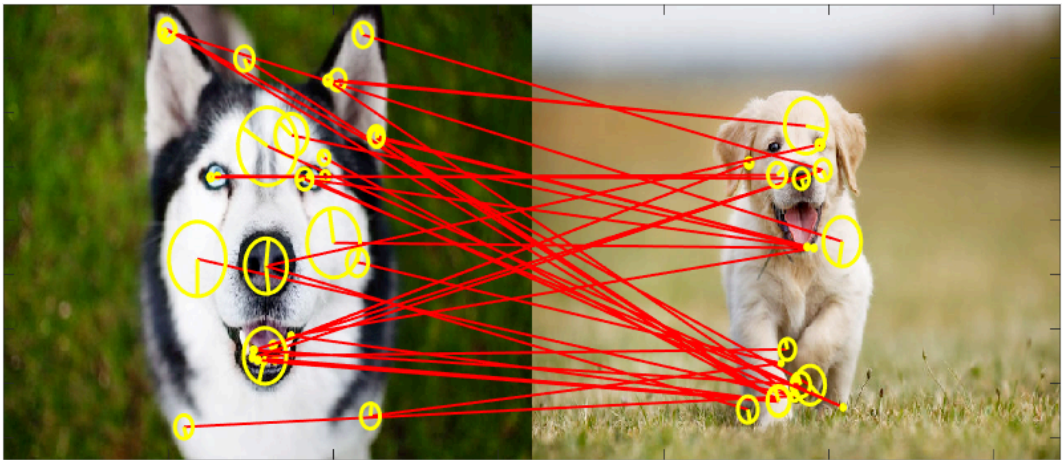
When applying the SIFT program the parameter I have chosen is: the number of octaves of the DOG scale space equal to 6 and peak selection threshold equal to 5. Then the result is shown below:



Husky_1 and Husky_3



Husky_3 and Husky_2



Husky_3 and Puppy_1

Husky_1 and Puppy_1

2.2.3. Discussion

Scale invariant feature transform is useful to detect the same object in different geometric modification. But according to the result of 2.2, it is not useful for similar object. Some matches in 2.2.2.2 are correct but most of them are wrong. If we want to implement SIFT we should implement it on the same object.

SIFT algorithm is sensitive to parameters selection, different choices of parameters will lead to different results.

SIFT can obtain key points from the object we are interest in, not from background and not from edge, this is a useful property.

2.3. Bag of words

2.3.1. Method

Extract sift features from each image, then using K-means to cluster these features to form 8-bin codebook. We could obtain four different code books for the four images. Then match the other three code books to the code book extract from Husky_3. The match is based on the minimum Euclid distance.

2.3.2. Result

Codebook of Husky_3

#1

```
Columns 1 through 15
    31.897    17.943    8.6031    8.1392    12.263    12.304      11.5    23.237    82.763    40.485    11.464    4.5309     2.866    3.3093    6.2629
Columns 16 through 30
    39.541    88.041    57.644    16.979     5.299    2.1856    2.5361    4.4021    29.742    35.876    37.072    26.412    15.309    12.918    6.3247
Columns 31 through 45
    5.7165    13.603     45.32    23.794    10.206    10.459    14.371    12.227    8.7371    21.969    119.07    49.974    9.0567    3.8969    3.2371
Columns 46 through 60
    2.8093    5.1082     38.18    117.27     56.33    16.325    6.1959    3.1701     2.366    4.7165    40.521    55.598    40.289    23.284    14.665
Columns 61 through 75
    14.954    8.7835    9.3763    23.242    44.613    24.562    13.299     13.82    14.046    9.1186    7.9227     20.17    121.46    43.124    8.3041
Columns 76 through 90
    3.8041    3.6701    2.5722    4.7165    45.799    124.03    48.938     8.268    3.5412    2.5876    2.3557       9.5    47.361     57.68    31.737
Columns 91 through 105
    15.082    10.567    15.046    12.603     17.18    31.567    29.603    25.747    20.758    15.454    9.1134    5.0876    4.5361    13.814     88.34
Columns 106 through 120
    46.552    13.665    5.2629    3.1495    2.5722    5.0928     34.99    97.211     39.68    6.6701    2.4381    2.9124    4.7423    13.521    50.954
Columns 121 through 128
    40.361    20.557    7.8144     7.134    11.412     14.82     22.01    33.912
```

#2

Columns 1 through 15

13.331 12.937 11.556 18.085 23.901 21.373 12.232 9.6197 32.817 26.606 14.93 13.93 19.599 13.711 14.451

Columns 16 through 30

21.725 83.134 55.669 16.31 7.0986 5.3592 4.4296 9.6056 38.859 77.014 60.113 21.599 7.2465 2.9718 2.3803

Columns 31 through 45

7.0141 33.937 17.944 16.739 19.282 28.408 24.585 17.986 9.2394 9.7606 56.718 27.169 16.458 24.134 33.042

Columns 46 through 60

18.725 11.718 20.493 115.8 55.232 10.331 6.8451 6.4155 5.1831 7.169 42.204 107.32 57 16.373 8.3873

Columns 61 through 75

5.838 4.2113 8.4577 44.408 17.761 16.451 16.014 24.141 23.894 17.732 13.915 11.789 48.028 21.056 21.57

Columns 76 through 90

29.958 35.887 20.57 12.979 21.859 112.05 38.479 10.768 7.9507 6.3732 7.0493 13.768 62.211 100.44 29.718

Columns 91 through 105

7.7042 5.5211 6.9366 7.0352 19.035 72.88 13.472 11.303 16.901 22.796 18.535 16.789 12.613 9.3028 21.127

Columns 106 through 120

16.275 19.324 21.415 21.007 18.211 15.535 16.944 67.042 28.352 12.972 6.7958 5.8873 9.1761 16.986 55.183

Columns 121 through 128

65.373 23.782 7.1197 3.7394 4.8521 7.2535 23.746 63.979

#3

Columns 1 through 15

17.785 11.983 8.6281 6.4959 15.05 49.669 56.124 31.182 17.397 6.6942 7.124 11.512 24.033 57.025 56.058

Columns 16 through 30

34.033 24.983 14.802 17.769 19.612 21.727 34.826 32.256 30.719 18.289 17.38 23.669 24.05 18.124 17.967

Columns 31 through 45

17.05 19.099 19.306 7.6529 7.7769 7.9669 22.769 57.298 63.983 37.95 53.157 15.298 8.876 5.7521 14.95

Columns 46 through 60

47.174 60.612 51.587 82.438 34.777 28.347 17.579 10.397 13.074 18.926 43.512 30.76 34.157 39.298 34.074

Columns 61 through 75

19.678 8.9256 7.3802 13.694 24 13.446 12.091 17.612 30.711 40.033 36.727 26.322 87.843 35.479 11.008

Columns 76 through 90

6.7273 9.5289 24.165 28.529 39.711 103.75 58.157 15.851 7.1405 5.0413 4.9256 8.876 30.248 41 35.14

Columns 91 through 105

31.669 27.479 20.868 10.686 10.628 16.835 14.537 24.57 27.802 25.339 27.273 14.926 14.959 10.331 69

Columns 106 through 120

55.661 20.223 7.7603 6.1488 5.0579 12.165 25.207 76.645 42.959 11.273 5.4628 4.8347 6.438 14.628 36.256

Columns 121 through 128

32.215 20.298 12.289 10.95 16.165 20.843 25.19 26.165

#4

Columns 1 through 15

24.056 9.2535 2.4014 3.4507 12.711 29.824 49.19 48.789 67.19 15.352 2.2254 0.96479 2.4507 14.732 44.014

Columns 16 through 30

80.099 53.394 20.493 7.4085 5.6549 7.6338 20.049 44.352 59.085 22.155 12.183 11.204 14.627 15.768 24.782

Columns 31 through 45

39.204 34.937 39.627 20.732 15.099 24.296 23.556 23.296 25.092 28.993 113.15 42.634 7.7958 5.5845 4.6268

Columns 46 through 60

5.9225 18.387 55.148 68.563 19.493 9.8099 11.12 15.183 25.155 40.972 48.592 22.732 12.845 10.345 10.254

Columns 61 through 75

12.007 29.254 53.775 41.507 29.289 41.479 47.451 45.704 21.444 8.2324 5.1268 6.8803 92.613 50.725 29.915

Columns 76 through 90

17.655 6.5141 8.6056 18.268 38.62 39.676 18.669 13.859 10.887 18.81 41.817 48.535 34.225 29.81 24.38

Columns 91 through 105

14.12 9.9296 11.085 24.479 40.465 33.873 16.056 37.908 48.62 32.704 16.254 8.838 5.7958 9.6408 21.289

Columns 106 through 120

39.866 50.979 32.049 17.141 12.042 11.732 13.887 18.169 28.563 35.366 25.972 22.838 21.049 17.345 16.5

Columns 121 through 128

28.993 29.479 20.852 14.049 12.042 12.627 16.754 23.014

#5

Columns 1 through 15

22.016 35.448 29.5 18.542 16.141 14.333 12.698 12.302 42.089 52.807 34.255 21.068 16.333 11.406 8.4792

Columns 16 through 30

14.786 49.943 67.953 44.094 21.911 12.609 5.9635 4.7396 15.271 18.969 36.771 46.104 38.87 27.922 12.979

Columns 31 through 45

5.6354 9.1979 30.49 40.667 29.714 18.406 15.708 12.417 12.339 18.49 56.078 36.797 23.573 23.87 27.859

Columns 46 through 60

26.719 16.234 23.641 110.49 57.823 20.042 10.5 9.4062 9.2917 10.016 38.703 29.703 29.427 33.094 41.417

Columns 61 through 75

53.745 32.125 11.083 11.786 30.573 29.38 22.87 17.547 15.38 14.859 18.88 24.203 45.12 27.224 30.156

Columns 76 through 90

33.594 30.172 21.401 15.495 22.609 102.8 32.875 11.448 9.2865 10.172 18.594 32.021 59.266 25.391 8.1198

Columns 91 through 105

7.276 17.922 48.536 60.792 44.849 30.979 22.995 17.432 11.469 14.568 15.292 17.859 24.865 25.292 22.927

Columns 106 through 120

14.229 12.208 18.078 22.724 26.714 35.078 32.812 32.172 13.422 8.224 11.125 18.911 32.323 51.036 49.536

Columns 121 through 128

15.036 12.568 8.2083 10.151 27.885 46.443 49.62 30.745

#6

Columns 1 through 15

30.647 20.065 10.784 11.95 17.856 16.734 20.058 24.338 52.849 35.547 17.489 15.158 15.381 12.647 15.993

Columns 16 through 30

32.187 22.82 22.619 23.072 30.784 36.043 25.36 17.388 19.324 12.626 13.403 15.734 24.345 35.043 30.086

Columns 31 through 45

21.187 21.604 45.676 17.122 7.1511 13.446 22.396 25.029 31.547 33.453 114.1 52.655 19.763 12.993 10.856

Columns 46 through 60

7.9784 12.662 41.432 27.05 21.647 33.727 57.892 66.698 40.129 11.345 10.741 15.597 16.151 21.252 28.986

Columns 61 through 75

51.827 43.151 18.417 16.079 48.863 35.626 27.59 22.266 21.885 14.95 13.173 19.036 118.47 46.165 13.64

Columns 76 through 90

8.6978 10.734 10.842 14.763 52.583 34.64 15.619 22.633 46.072 68.144 47.986 23.072 19.281 15.072 19.763

Columns 91 through 105

36.36 43.662 52.022 30.022 9.1871 11.014 29.612 40.813 35.245 20.086 15.216 10.489 10.964 13.281 61.058

Columns 106 through 120

46.353 27.554 15.755 13.734 14.468 15.23 29.252 32.245 25.022 23.971 26.403 34.914 31.165 19.165 20.561

Columns 121 through 128

12.727 19.791 27.554 31.46 41.741 25.906 11.122 11.403

#7

Columns 1 through 15

8.5065 5.4675 6.4026 33.636 101.25 54.494 12.701 9.4026 50.675 24.247 7.4935 8.4026 31.558 30.247 20.87

Columns 16 through 30

24.948 40.312 22.156 10.922 9.4935 18.104 12.494 11.987 21.182 13.571 12.714 6.8442 11.364 17.286 12.818

Columns 31 through 45

8.5325 12.065 13.325 5.2078 8.961 57.26 127.94 61.156 6.3636 4.7013 92.065 35.857 11.662 16.403 34.623

Columns 46 through 60

21.013 7.7532 20.74 62.143 28.701 15.351 17.948 25.143 15.896 7.9351 18.597 17.727 16.831 11.584 16.766

Columns 61 through 75

17.169 13.896 9.9091 12.779 15.312 4.8571 7.7143 67.039 128.34 46.753 3.4805 4.8701 90.494 20.753 11.429

Columns 76 through 90

28.675 36.74 11.299 6.7013 31.065 59.442 19.364 11.065 11.519 25.312 22.286 16.779 23.506 24.143 16.416

Columns 91 through 105

10.403 11.948 13.792 12.221 12.325 15.234 12.455 9.1299 13.519 69.065 98.182 24.351 3.8312 5.9351 52.221

Columns 106 through 120

29.558 19.052 38.117 38.675 8.1818 4.8961 18.052 35.636 23.455 14.974 12.013 14.779 9.8701 11.26 23.351

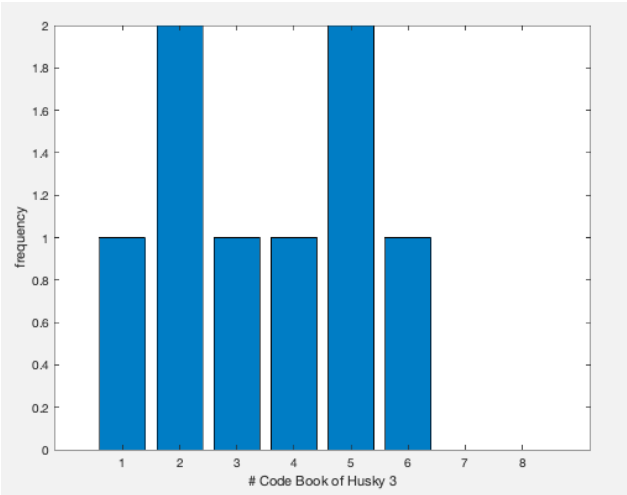
Columns 121 through 128

21.584 19.753 13.571 10.662 9.6234 7.2468 6.5195 11.766

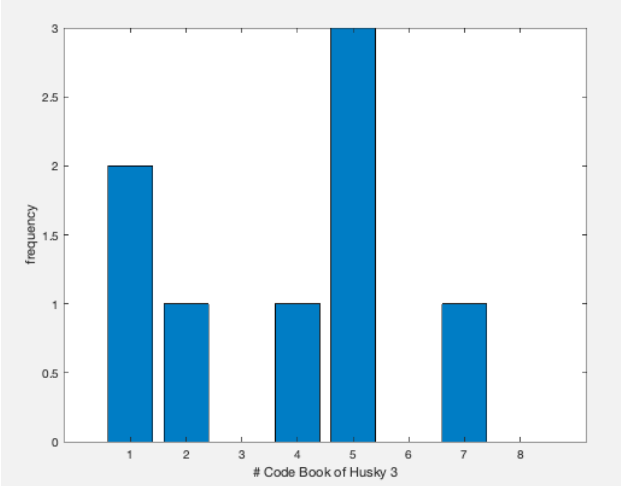
#8

```
Columns 1 through 15
    8.5065    5.4675    6.4026   33.636   101.25    54.494   12.701    9.4026   50.675   24.247    7.4935    8.4026   31.558   30.247    20.87
Columns 16 through 30
   24.948    40.312    22.156   10.922    9.4935   18.104   12.494   11.987    21.182   13.571   12.714    6.8442   11.364   17.286   12.818
Columns 31 through 45
    8.5325   12.065    13.325    5.2078    8.961    57.26   127.94   61.156    6.3636    4.7013   92.065   35.857   11.662   16.403   34.623
Columns 46 through 60
   21.013    7.7532    20.74    62.143   28.701   15.351   17.948   25.143   15.896    7.9351   18.597   17.727   16.831   11.584   16.766
Columns 61 through 75
   17.169   13.896    9.9091   12.779   15.312    4.8571    7.7143   67.039   128.34   46.753    3.4805    4.8701   90.494   20.753   11.429
Columns 76 through 90
   28.675    36.74    11.299    6.7013   31.065   59.442   19.364   11.065   11.519   25.312   22.286   16.779   23.506   24.143   16.416
Columns 91 through 105
   10.403   11.948   13.792   12.221   12.325   15.234   12.455    9.1299   13.519   69.065   98.182   24.351    3.8312    5.9351   52.221
Columns 106 through 120
   29.558   19.052   38.117   38.675    8.1818    4.8961   18.052   35.636   23.455   14.974   12.013   14.779    9.8701   11.26    23.351
Columns 121 through 128
   21.584   19.753   13.571   10.662    9.6234    7.2468    6.5195   11.766
```

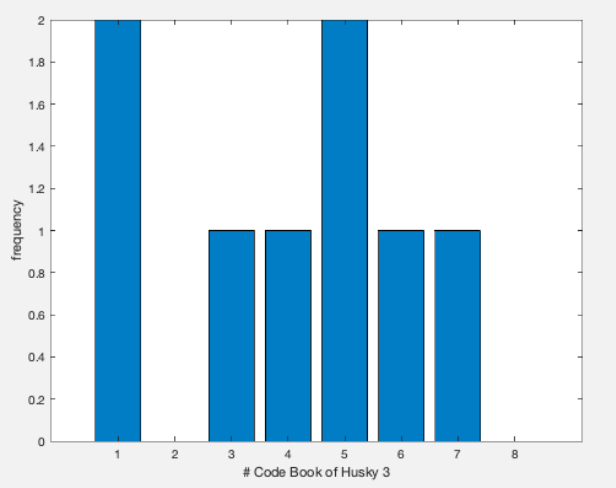
Codebook match between Husky_1 and Husky_3



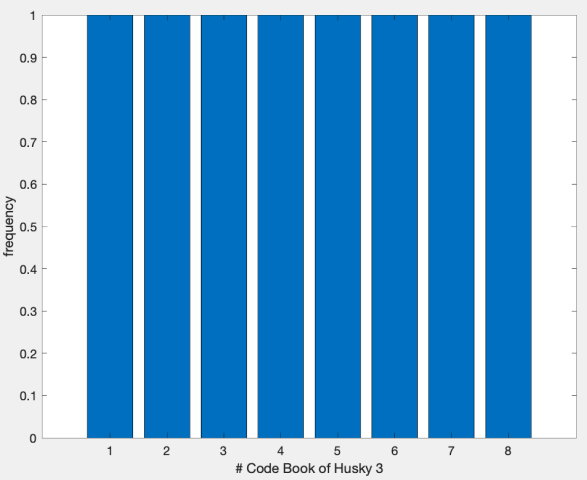
Codebook match between Husky_2 and Husky_3



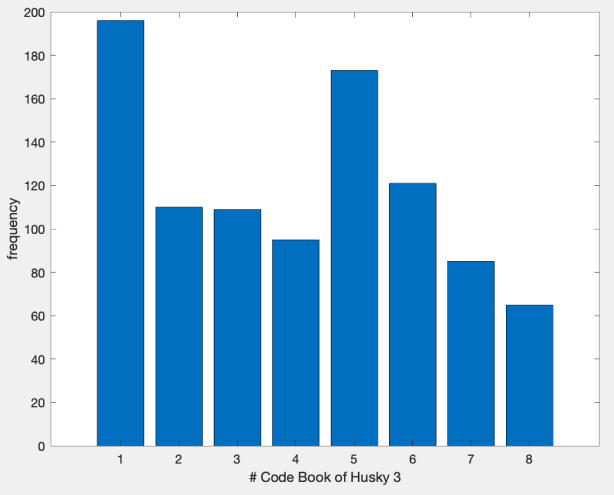
Codebook match between Puppy_1 and Husky_3



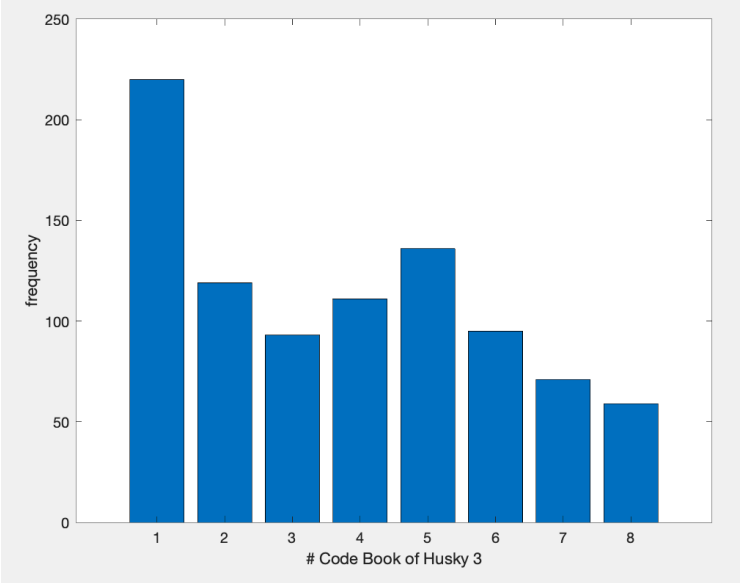
Codebook match between Husky_3 and Husky_3



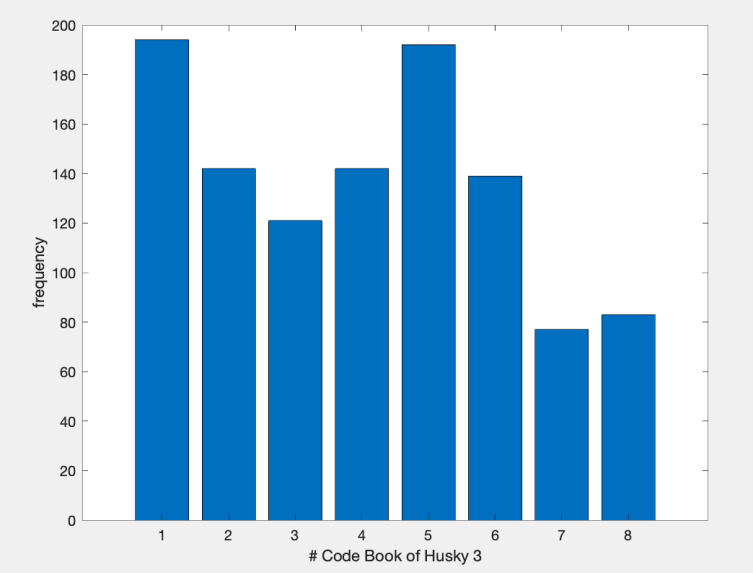
Matching features from Husky_1 to Husky_3's codebook



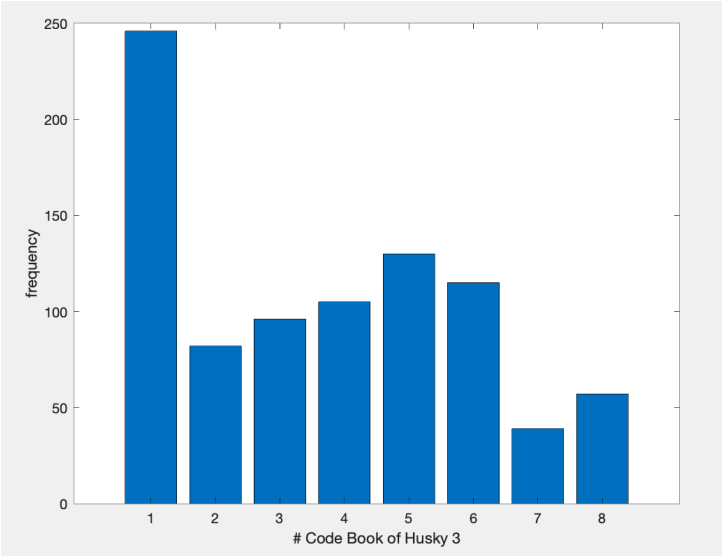
Matching features from Husky_2 to Husky_3's codebook



Matching features from Husky_3 to Husky_3's codebook



Matching features from Puppy_1 to Husky_3's codebook



2.3.3. Discussion

- From the matching between codebook and code book, we can find:

#5 code book: It has a fairly high frequency in the three matches, so it may be a feature of dogs.
#3 #6 code book: They are absent from the match between Husky_2 and Husky_3, maybe this represent the front face of dog.
#2 code book: It is absent from the match between Husky_3 and Puppy_1, this may be a feature special to husky.
#8 code book: It is a special cluster of features to Husky_3.
Husky_1 is the image which look likes Husky_3 the most in histogram.

- From the matching between features and code book, we can find:
  The histogram of Husky_1, Husky_2 and husky_3 are almost the same, but the histogram of puppy_1 is far different from the three.
  We can find the Husky_1 is the image which most like Husky_3 because their histograms are closer than that of Husky_2.