Yusufu Feierdawusi
4929-7851-40
Feierdaw@usc.edu

EE569 Homework #4
March 27, 2022

Problem 1: Texture Analysis

1. Motivation

One of the main uses of machine learning in the image processing industry is the analysis of textures. More specifically, texture classification. It is the notion of recognizing different textures of surfaces such as rocks, grass and other common objects we see in the environment. The tool used to distinguish these textures is called pattern recognition, where we use a small patch of an image and analyze pixel-wise features and use a machine learning algorithm like K-means to label such textures. The main difficulties in this process are selecting a good filter size and extracting concrete and useful features in order to define a good cluster for the learning algorithms [1].

a. Texture Classification

2. Approaches and Procedures

In this problem, we first read the 36 training images with labels: blanket, brick, grass and stones. Next, we have five different 1D law vectors (assuming they are column vectors), then in order to project them to the image with a 5x5 window, we compute matrix multiplication from one kernel to the other five transposed kernels. This way we have 25 law filters to obtain 25 features from every 5x5 patch image (**Figure 1**). We can now start the training process. We first normalize and perform odd reflective boundary extensions in all 36 images. Then we obtain the filtered images with convolution using all 25 law filters. These resultant 25D filtered images represent the 25 features observed for all 36 images. Then, we calculate the average energy features for each dimension in every image (**Figure 2**). We first square every pixel value to

remove any negative features we have, we then take the total sum of all squared pixel values in every image and normalize the sum by the total pixel size which is 128*128. In order to avoid high luminance values in each image, we also normalize the other 24D energy vectors by the first L5L5_T filter value. Now, the basic training process is complete. In order to classify the test images with the unknown labels, we perform the same filtering steps above to compute the energy vectors of the test images (**Figure 3**). However, the 25 dimensional features are difficult for human comprehension, thus we use a built-in dimension reduction algorithm called principal component analysis (PCA). We need to use the 25D energy feature vectors of the training images as the input for this function. This function returns a matrix of principal components (25 in this case), we use the top 3 components to map both the 25D train and test image to 3D (**Figure 4, 5**). Now, we can visualize the 36 by 3 matrix using a 3D scatter plot (**Figure 6**). Lastly, in order to classify the test images we use a nearest neighbor algorithm based on the Mahalanobis distance between every 3D test energy features vector and the trained clusters (**Table 1**). We use the formula below to calculate the Mahalanobis distance, with the covariance matrix S being X_T * X (X being the trained energy feature vectors).

$$d(\vec{x},\vec{y}) = \sqrt{(\vec{x}-\vec{y})^{\mathsf{T}}\mathbf{S}^{-1}(\vec{x}-\vec{y})}.$$

3. Experimental Results

Figure 1: Sample of Law Filters

```
val(:,:,1) =

     1     4     6     4     1
     4    16    24    16     4
     6    24    36    24     6
     4    16    24    16     4
     1     4     6     4     1


val(:,:,2) =

    -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


val(:,:,3) =

    -1     0     2     0    -1
    -4     0     8     0    -4
    -6     0    12     0    -6
    -4     0     8     0    -4
    -1     0     2     0    -1


val(:,:,4) =

    -1     2     0    -2     1
    -4     8     0    -8     4
    -6    12     0   -12     6
    -4     8     0    -8     4
```

Figure 2: Sample of Train Energy Feature Vectors

36x25 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	3.1087e...	2.1515	1.4105	2.3833	10.1088	0.2600	0.2790	0.6687	2.8435	0.4186	1.0938	4.5647	2.0762	8.5777	18.7063
2	2.0636e...	15.5592	10.8324	18.9576	83.2006	1.9503	2.1892	5.2429	21.7819	3.4514	8.7952	35.4915	16.9880	67.9384	156.9036
3	1.6870e...	31.9414	15.6333	16.7184	31.3912	2.7342	0.8588	0.6432	1.7624	0.2758	0.2524	0.7603	0.2510	0.8080	2.7169
4	1.7008e...	26.0789	14.7712	15.7620	37.4222	1.9220	0.7314	0.7761	2.8491	0.3195	0.3600	1.2307	0.4005	1.4019	5.2750
5	1.7009e...	25.9125	13.1872	14.9589	36.1055	2.0952	0.7561	0.7445	2.7281	0.3254	0.3579	1.1968	0.3942	1.3056	4.6384
6	1.6898e...	28.6102	14.3887	15.4249	30.6354	2.4537	0.8085	0.6555	1.9688	0.3197	0.3088	0.9133	0.3468	1.0504	3.8621
7	1.7782e...	4.5220	1.2181	0.6004	0.9073	0.2232	0.0674	0.0416	0.0781	0.0254	0.0184	0.0382	0.0255	0.0585	0.2076
8	1.7571e...	14.9450	4.9298	2.8981	6.5411	0.6994	0.2442	0.1972	0.5418	0.1031	0.1090	0.3219	0.1685	0.5314	1.7114
9	1.7682e...	8.7844	3.0313	1.8310	3.7846	0.4292	0.1620	0.1217	0.3372	0.0506	0.0506	0.1664	0.0616	0.2175	0.6114
10	1.1327e...	6.6972	3.5556	5.7300	39.4315	1.0565	0.5757	0.8992	5.6949	0.3967	0.5976	3.6493	1.0661	6.3044	46.6635
11	2.4609e...	25.3123	8.1724	10.3768	60.9259	1.6163	0.7395	1.0517	6.7952	0.3930	0.5286	3.1464	0.7149	3.8816	20.2354
12	1.5998e...	18.2173	4.2099	4.6142	22.9005	0.2584	0.1262	0.1815	1.2025	0.0484	0.0635	0.4032	0.0672	0.3706	1.4348
13	1.5669e...	11.8889	7.6160	12.2672	62.7403	0.5651	0.2664	0.3348	1.3190	0.1224	0.1398	0.4892	0.1423	0.4958	1.9383
14	3.6724e...	13.0943	5.8618	5.2167	17.6264	0.4024	0.1711	0.1781	0.6052	0.0678	0.0677	0.2286	0.0734	0.2449	0.9093

Figure 3: Sample of Test Energy Feature Vectors

12x25 double

	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	36.0759	10.7406	9.0180	22.8171	3.1821	1.0888	0.9621	2.5090	0.4703	0.4384	1.1919	0.4765	1.3807	5.0673
2	35.2809	18.7487	19.3137	35.1345	2.6110	0.8771	0.7621	2.2005	0.2863	0.2953	0.9424	0.2981	0.9627	3.1984
3	9.8807	2.7571	1.6040	3.0040	0.5233	0.1621	0.1184	0.2572	0.0666	0.0584	0.1335	0.0884	0.2185	0.6967
4	40.2388	10.8526	10.8508	58.3620	4.4037	1.4180	1.5393	8.4833	0.8202	0.9806	5.5107	1.6761	9.7710	72.9722
5	84.1333	15.5958	13.2659	62.0270	3.8058	1.2234	1.3614	7.4061	0.5493	0.7164	4.2678	1.0755	6.8518	46.2009
6	38.7793	12.7332	11.1329	26.7933	3.1647	1.1934	1.1074	2.8771	0.4964	0.4849	1.3286	0.5008	1.4887	5.2466
7	65.1334	20.8155	18.1774	58.8108	3.5484	1.7534	2.1077	7.4364	0.9425	1.1952	4.3438	1.5961	5.9890	24.5831
8	26.1364	5.3296	5.2281	30.3843	2.0966	0.6624	0.8243	5.2198	0.4045	0.5626	3.6286	1.0328	6.6553	54.2078
9	25.3065	9.0160	6.7473	24.0290	0.3072	0.1317	0.1540	0.7550	0.0466	0.0557	0.2672	0.0569	0.2651	1.0828
10	15.7974	4.8634	4.6870	20.8850	1.2563	0.4877	0.5297	2.5089	0.2511	0.2805	1.3137	0.3806	1.8142	10.2711
11	19.5102	13.9306	17.3084	147.4030	6.6195	5.3352	5.5169	10.6852	3.7178	3.8209	7.1109	2.9772	6.4076	28.4968
12	38.9423	12.7145	10.4747	25.0392	2.9784	1.1010	0.9973	2.5516	0.4479	0.4265	1.1290	0.4461	1.2585	4.4461

Figure 4: Sample of Reduced 3D Training Energy Vectors

36x3 double

	1	2	3	4
1	3.1087e...	130.6361	-8.8053	
2	2.0635e...	256.2258	93.1574	
3	1.6870e...	124.0367	-43.4986	
4	1.7008e...	119.6381	-32.1395	
5	1.7008e...	122.1204	-35.4233	
6	1.6898e...	125.5670	-43.0683	
7	1.7782e...	64.4910	-13.7348	
8	1.7571e...	76.3268	-17.5761	
9	1.7682e...	67.8694	-14.0473	
10	1.1327e...	156.6056	-24.5680	
11	2.4608e...	207.9993	-52.1457	
12	1.5998e...	112.8929	-51.3198	
13	1.5668e...	234.2614	-126.4023	
14	3.6723e...	170.6486	-55.9928	
15	1.7402e...	122.7237	-60.9000	
16	1.3196e...	54.3898	-12.4527	
17	2.5813e...	133.6780	-51.4441	
18	2.0036e...	129.1440	-54.9995	
19	1.7018e...	109.5042	-41.0532	
20	1.6981e...	114.3518	-43.0094	

Figure 5: Reduced 3D Testing Energy Vectors

12x3 double

	1	2	3	4
1	1.7000e...	113.8544	-35.0355	
2	1.6902e...	119.7151	-35.4376	
3	1.7658e...	70.2686	-16.4675	
4	1.9912e...	288.5944	-64.9675	
5	1.9283e...	200.5250	-26.6246	
6	1.6988e...	114.1891	-31.0514	
7	1.4942e...	235.5149	-110.3343	
8	1.3233e...	161.5518	-22.2139	
9	1.4892e...	126.6040	-60.2795	
10	2.3181e...	152.0288	-51.1700	
11	1.3578e...	243.7369	-31.6263	
12	1.6975e...	110.3816	-29.9998	

Figure 6: The reduced 3-D Train Feature Vectors in 3D Scatter Plot

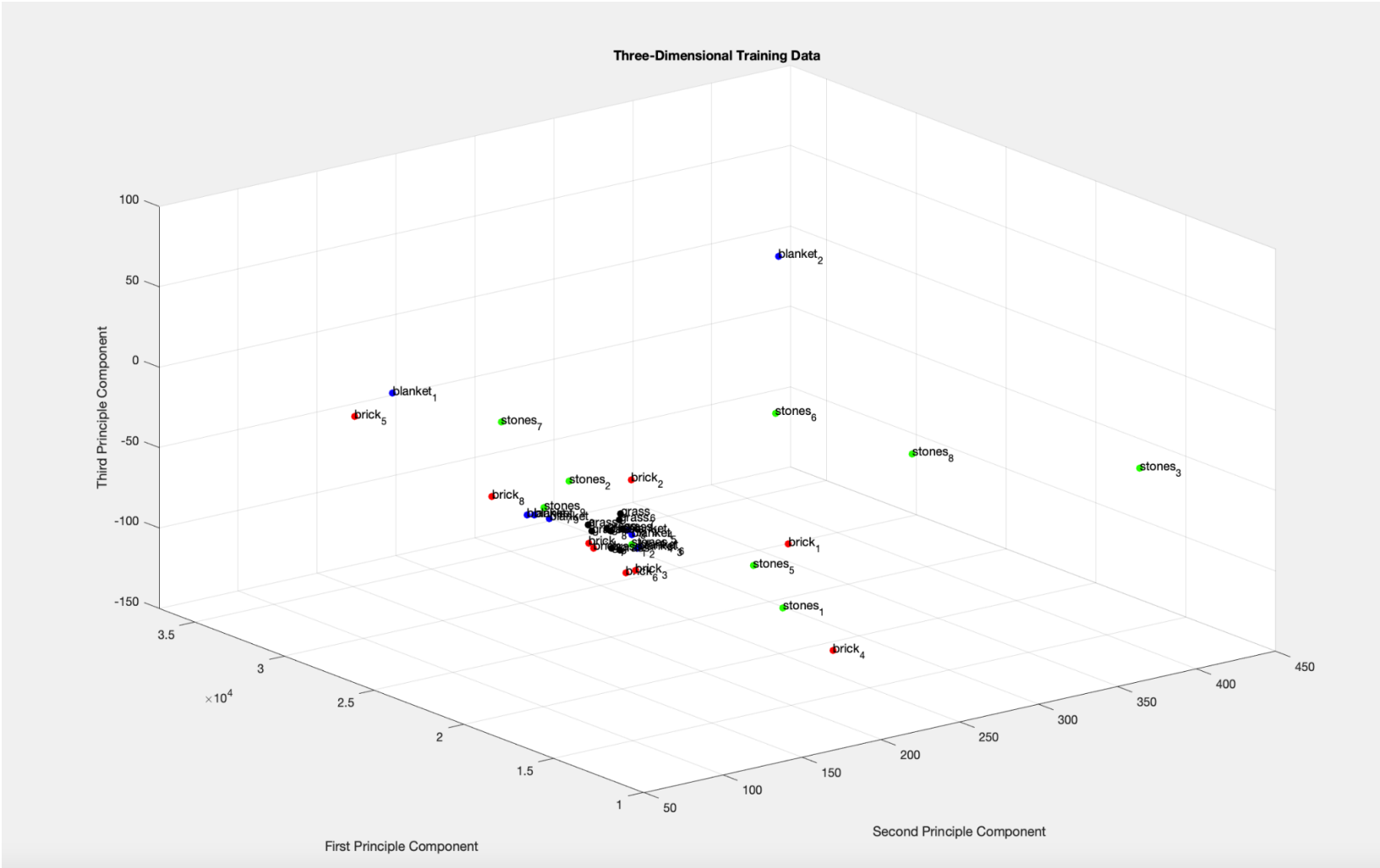


Figure 7: Zoomed-In View of the Clusters

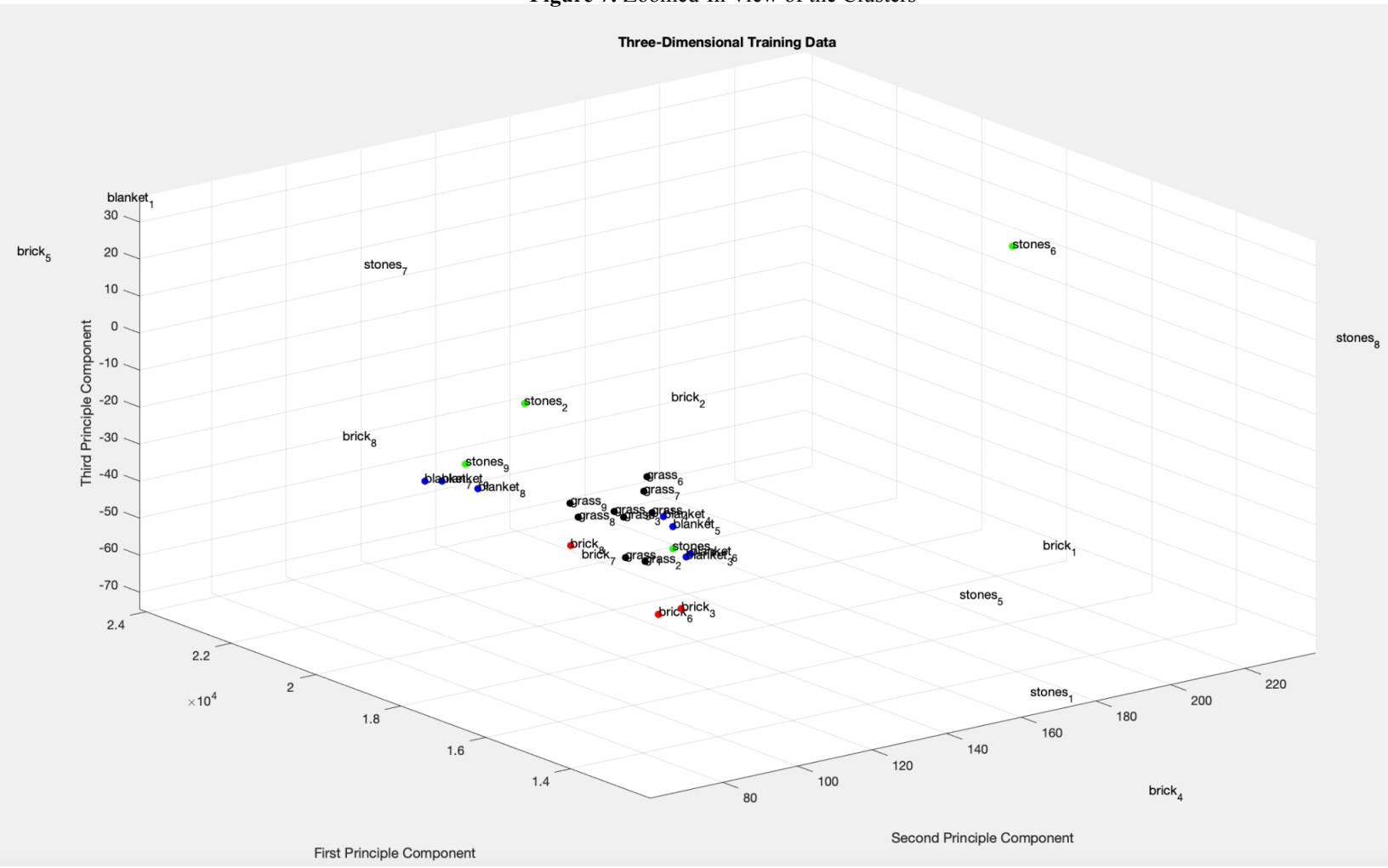
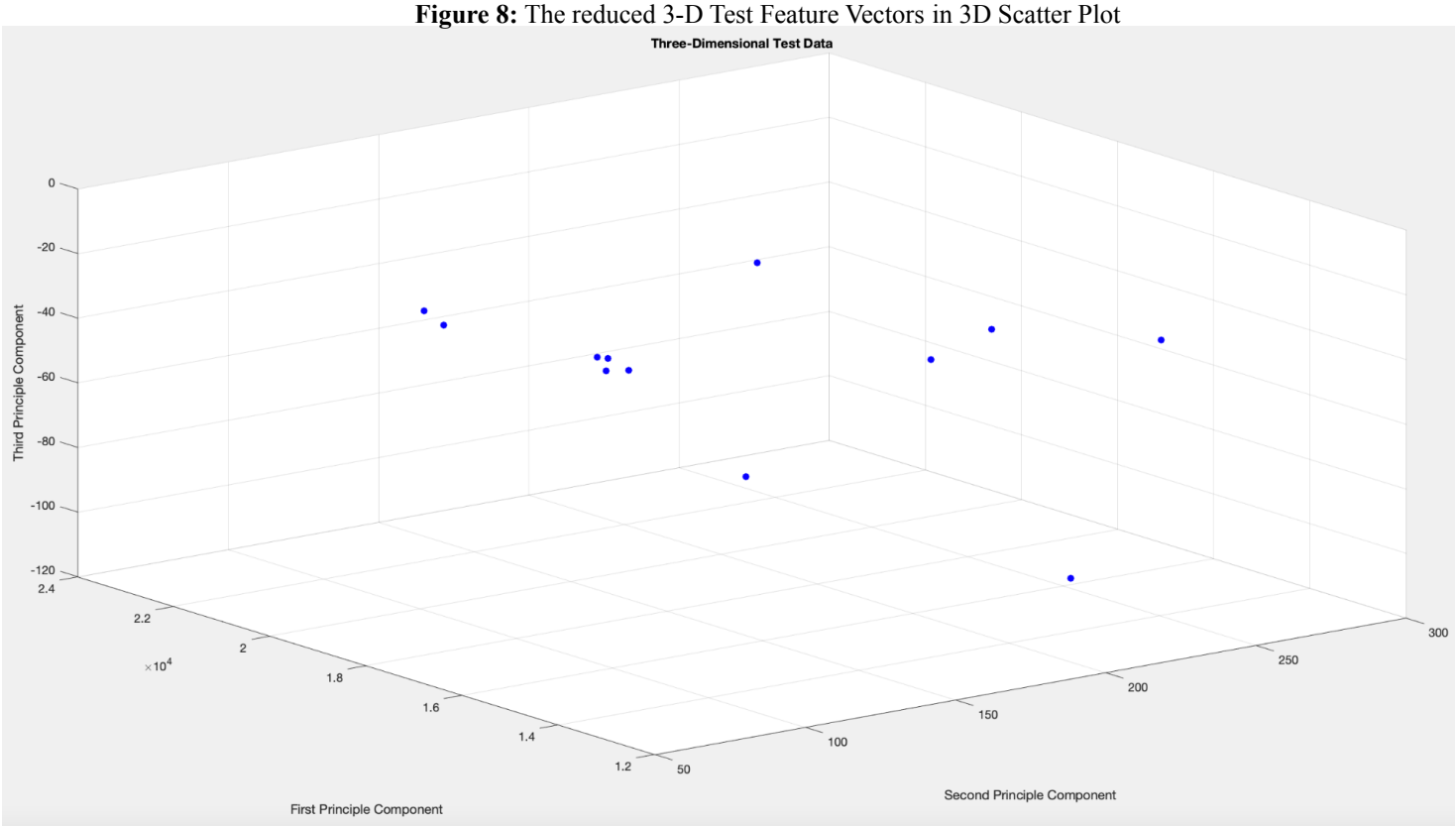


Figure 8: The reduced 3-D Test Feature Vectors in 3D Scatter Plot



Img_1	Img_2	Img_3	Img_4	Img_5	Img_6	Img_7	Img_8	Img_9	Img_10	Img_11	Img_12
0.3102	0.1944	0.3198	0.6075	0.3786	0.2973	0.7579	0.4136	0.4508	0.3104	0.5731	0.2896
0.7393	0.7083	0.7354	0.7805	0.6230	0.7205	1.0464	0.6493	0.8510	0.7740	0.6696	0.7196
0.0466	0.1790	0.0401	0.3151	0.1765	0.0639	0.4141	0.1703	0.1022	0.0669	0.3218	0.0709
0.0202	0.1359	0.0170	0.3384	0.1554	0.0122	0.4645	0.1586	0.1583	0.0767	0.3259	0.0212
0.0179	0.1487	0.0039	0.3290	0.1577	0.0257	0.4490	0.1580	0.1420	0.0702	0.3217	0.0343
0.0459	0.1793	0.0383	0.3126	0.1725	0.0625	0.4142	0.1665	0.1038	0.0681	0.3184	0.0698
0.1538	0.0193	0.1658	0.4893	0.2738	0.1431	0.6118	0.2886	0.2934	0.1733	0.4623	0.1341
0.1206	0.0152	0.1323	0.4561	0.2443	0.1094	0.5796	0.2576	0.2624	0.1442	0.4313	0.1005
0.1462	0.0127	0.1579	0.4806	0.2649	0.1350	0.6048	0.2795	0.2870	0.1673	0.4533	0.1260
0.1908	0.2973	0.1771	0.2040	0.0895	0.1878	0.3986	0.0350	0.2057	0.2294	0.1559	0.1954
0.1093	0.2229	0.0989	0.2617	0.1230	0.1128	0.4067	0.1420	0.1491	0.1022	0.2719	0.1203
0.0881	0.2081	0.0860	0.3235	0.2184	0.1074	0.3876	0.2065	0.0673	0.0869	0.3448	0.1126
0.5117	0.6410	0.5031	0.3494	0.5223	0.5277	0.0769	0.4953	0.3674	0.4972	0.4616	0.5355
0.2535	0.1913	0.2648	0.5516	0.3736	0.2527	0.6423	0.4037	0.3504	0.2167	0.5587	0.2468
0.1236	0.2380	0.1220	0.3255	0.2463	0.1431	0.3649	0.2346	0.0536	0.1068	0.3603	0.1481
0.1061	0.0535	0.1164	0.4378	0.2295	0.0944	0.5606	0.2337	0.2446	0.1473	0.4087	0.0861
0.1239	0.1285	0.1354	0.4370	0.2740	0.1297	0.5194	0.2902	0.2130	0.0848	0.4476	0.1254
0.0820	0.1852	0.0850	0.3495	0.2284	0.1007	0.4174	0.2276	0.1044	0.0517	0.3714	0.1038
0.0323	0.1491	0.0384	0.3498	0.1974	0.0516	0.4437	0.1952	0.1261	0.0536	0.3533	0.0546
0.0394	0.1629	0.0405	0.3376	0.1935	0.0590	0.4298	0.1894	0.1129	0.0562	0.3440	0.0637
0.0246	0.1131	0.0339	0.3619	0.1740	0.0115	0.4838	0.1791	0.1725	0.0764	0.3483	0.0035
0.0240	0.1266	0.0257	0.3469	0.1587	0.0061	0.4744	0.1634	0.1673	0.0795	0.3322	0.0130
0.0341	0.1046	0.0432	0.3691	0.1764	0.0195	0.4933	0.1829	0.1819	0.0819	0.3533	0.0106
0.0730	0.1083	0.0749	0.3683	0.1523	0.0533	0.5165	0.1649	0.2155	0.1194	0.3381	0.0501
0.0530	0.1108	0.0556	0.3614	0.1555	0.0333	0.5010	0.1651	0.1969	0.1017	0.3375	0.0304
0.0460	0.0886	0.0583	0.3881	0.1963	0.0378	0.5053	0.2031	0.1901	0.0835	0.3737	0.0291
0.0629	0.0742	0.0735	0.3982	0.1964	0.0501	0.5229	0.2060	0.2092	0.0995	0.3783	0.0412
0.2433	0.3765	0.2325	0.1947	0.2520	0.2563	0.2271	0.2174	0.1212	0.2506	0.2542	0.2646
0.1099	0.0806	0.1159	0.4096	0.1867	0.0928	0.5598	0.2111	0.2561	0.1340	0.3800	0.0868
0.6844	0.7896	0.6702	0.3737	0.5331	0.6826	0.5506	0.5194	0.6462	0.7053	0.3458	0.6905
0.0272	0.1607	0.0215	0.3267	0.1725	0.0445	0.4330	0.1689	0.1213	0.0636	0.3270	0.0513
0.3139	0.4467	0.3038	0.1910	0.3139	0.3280	0.1543	0.2935	0.1851	0.3012	0.2945	0.3363
0.3608	0.3841	0.3530	0.3841	0.2170	0.3450	0.6396	0.2408	0.4487	0.3976	0.2790	0.3471
0.1805	0.1176	0.1868	0.4582	0.2342	0.1662	0.6169	0.2715	0.3202	0.1832	0.4294	0.1605
0.4256	0.4889	0.4140	0.3112	0.2593	0.4149	0.5751	0.2595	0.4653	0.4623	0.1903	0.4201
0.1079	0.0331	0.1196	0.4423	0.2339	0.0974	0.5656	0.2498	0.2499	0.1253	0.4218	0.0886

Table 1: Mahalanobis distance between each test energy vector and 36 train energy vectors

(rows: 3D train, columns :3D test)

4. Discussion

1). Filter computation: When filtering the image with the first law filter (which is L5*L5T), the resultant energy feature vector for this dimension becomes very large (*10^4) compared to the other energy vectors. This directly affects the range of the first principle component presented in the 3D scatter plot.

2). Discriminant power: The table to the right shows the discriminant power of the 25 features. From this we can see that the strongest feature dimension is 24. The weakest is dimension 2. This is because discriminant power is calculated by intra-variety/inter-variety. Thus, dimension 24 has a very high intra-variety and low inter-variety, while dimension 2 has a very low intra-variety and high inter-variety.

25x1 double

	1
1	17.9216
2	0.9553
3	3.4525
4	7.7430
5	4.3059
6	1.3692
7	2.6974
8	8.2555
9	5.3335
10	11.3500
11	12.2753
12	11.1189
13	11.3790
14	10.9082
15	4.4705
16	8.1963
17	5.5838
18	2.8816
19	2.9490
20	13.3279
21	6.2992
22	21.7700
23	2.8306
24	27.3354
25	2.2690

3). Texture classification results: This result analysis refers to **Table 1**. We label each test image based on the class of the nearest neighbor (out of 36 trained labels). Rows 1-9 is Blanket, Rows 10-18 is Brick, Rows 19-27 is Grass, Rows 28-36 is Stones. For test Img_1, the closest mahalanobis distance is 0.0179, which is row 5 (Blanket). For test Img_2, the closest mahalanobis distance is 0.0127, which is row 9 (Blanket). For test Img_3, the closest mahalanobis distance is 0.0039, which is row 5 (Blanket). For test Img_4, the closest mahalanobis distance is 0.1910, which is row 32 (Stones). For test Img_5, the closest mahalanobis distance is 0.0895, which is row 10 (Brick). For test Img_6, the closest mahalanobis distance is 0.0061, which is row 22 (Grass). For test Img_7, the closest mahalanobis distance is 0.0769, which is row 13 (Brick). For test Img_8, the closest mahalanobis distance is 0.0350, which is row 10 (Brick). For test Img_9, the closest mahalanobis distance is 0.0536, which is row 15 (Brick). For test Img_10, the closest mahalanobis distance is 0.0517, which is row 18 (Brick). For test Img_11, the closest mahalanobis distance is 0.1559, which is row 10 (Brick). For test Img_12, the closest mahalanobis distance is 0.0035, which is row 21 (Grass). Thus, their classified labels are **[Blanket, Blanket, Blanket, Stones, Brick, Grass, Brick, Brick, Brick, Brick, Brick, Grass]**. Compared to their actual labels which are:

[Grass, Blanket, Blanket, Stones, Stones, Grass, Brick, Stones, Brick, Brick, Blanket, Grass]. **We have 8 correct predictions and 4 false predictions, which means that the error rate is 33%.** One of the possible explanations for having these false predictions is the outliers that exist in our training features vector. This can be seen in the 3D plot above where a few blanket/stones/brick data points are further away from the clusters that are formed in the center.

b. Advanced Texture Classification

2. Approaches and Procedures

1. Unsupervised: In this problem, we first create a string array to input all the given ground truth test labels. We then separately use the resultant 25D and 3D test energy

features vector from Q1.a as our input in the in-built K-means learning algorithm with a parameter K = 4 (**Figure 1,2**). In order to calculate the purity of each resultant cluster, we find the majority class in each cluster and label all the members of that cluster as that class (**Figure 3,4**). Then we count the number of incorrect classes compared to the new class label in each cluster (**Figure 5,6**). To calculate the error rate for these two methods, we compare the new class label on each image to the class label in the ground truth array (**Figure 7**).

2. Supervised: For the Random Forest Classifier, we use the in-built TreeBagger function with the train energy features vector from Q1.a and the train labels array as our inputs (number of trees = 50). Then to make predictions on the test images, we use the inbuilt Predict function with the trained random forest and the test energy features vector from Q1.a as our inputs. We perform identical steps for SVM classifier, except we use multisvm in-built function.

3. Experimental Results

Figure 1: K-Means with 25D

12x1 double

	1
1	3
2	3
3	3
4	1
5	1
6	3
7	2
8	2
9	2
10	4
11	2
12	3

Figure 2: K-Means with 3D

12x1 double

	1
1	2
2	2
3	2
4	3
5	3
6	2
7	4
8	4
9	4
10	1
11	4
12	2

Figure 3: Majority with 25D

12x1 double

	1
1	3
2	3
3	3
4	4
5	4
6	3
7	2
8	2
9	2
10	2
11	2
12	3

Figure 4: Majority with 3D

12x1 double

	1
1	3
2	3
3	3
4	4
5	4
6	3
7	2
8	2
9	2
10	2
11	2
12	3

Figure 5: Purity of clusters with 25D K-means

```
"The purity of K-m…"    "1"    "Cluster 2: "    "0.5"    "Cluster 3: "    "0.6"    "Cluster 4: "    "1"
```

Figure 6: Purity of clusters with 3D K-means

```
"The purity of K-m…"    "1"    "Cluster 2: "    "0.6"    "Cluster 3: "    "1"    "Cluster 4: "    "0.5"
```

Figure 7: Error rates of both methods

```
"The error rate for K-means with 25D: "    "0.33333"

"The error rate for K-means with 3D: "    "0.33333"
```

Figure 8: Error rates of both classifiers

```
"The error rate for RF classifier with 3D: "    "0.083333"
"The error rate for SVM classifier wit…"    "0.5"
```

4. Discussion

1). Unsupervised: The purity rate of K-means with 25D for clusters 1, 4 is 100%, for cluster 2 is 50% and cluster 3 is 60%. The error rate for this method is 33.3%. The purity rate of K-means with 3D for clusters 1, 3 is 100%, for cluster 2 is 60% and cluster 3 is 50%. The error rate for this method is also 33.3%. As we can see from these results, the dimension reduction does not necessarily increase the effectiveness of K-means clustering. However, after running the algorithm multiple times, some results with 3D K-means actually increased the error rate to 41.67%.

2). Supervised: The error rate for the RF classifier is 8.3%, which is 1 incorrect prediction out of 12 samples. The error rate for the SVM classifier is 66.67%, which is 8 incorrect predictions out of 12 samples. The random forest classifier performs a lot better than the SVM classifier here. One possible explanation here is that the open source SVM function is not as efficient as the built-in RF classifier.

Problem 2: Texture Segmentation

1. Motivation

One of the main purposes of doing texture analysis is to recognize the type of texture and separate one texture from another, which is texture segmentation. The usage of texture segmentation is to identify and differentiate regions of textures within an image. In this problem, we perform mask based feature extraction and machine learning to identify different regions in a mosaic image [2].

2. Approaches and Procedures

In this problem, the filtering process is identical to the filtering process in Q1.a. Instead of computing the 25D energy feature vectors based on the entire image, we use a window approach (15*15 and 23*23) to calculate the 25D energy feature vectors for every pixel (**Figure**

1). We first take the sum of all pixel values within the window size and normalize it by the total number of pixels within that window. Since all the vectors have a zero-mean other than the L5L5_T filter, we normalize those vectors by the L5L5_T filter (**Figure 2**). Then we discard this energy vector. Using the rest of the 24D vectors as the input to the built-in K-means algorithm (with K = 6), we classify each pixel into one of 6 classes (**Figure 3**). This is the result of the segmentation, in order to visualize this result, we reassign six different RGB values to the pixels belonging to each of the six classes (**Figure 4, 6**).

b. Advanced Texture Segmentation

We use the resultant 25D energy feature vectors for each pixel from the previous section as the input to the in-built PCA function. This function reduces the 25 dimensional vectors to 3 dimensional vectors. We then perform post-processing by filling the holes that exist in the texture regions (**Figure 5, 7**).

3. Experimental Results

Figure 1: Sample of Pixel-wise Energy Features Matrix

512x512 double

	1	2	3	4	5	6	7	8	9	10	11	12
1	1.0559e...	1.0565e...	1.0584e...	1.0620e...	1.0669e...	1.0727e...	1.0800e...	1.0904e...	1.1045e...	1.1218e...	1.1432e...	1.1710e...
2	1.0540e...	1.0545e...	1.0563e...	1.0597e...	1.0645e...	1.0703e...	1.0778e...	1.0887e...	1.1034e...	1.1213e...	1.1430e...	1.1708e...
3	1.0496e...	1.0500e...	1.0515e...	1.0544e...	1.0589e...	1.0647e...	1.0730e...	1.0850e...	1.1011e...	1.1201e...	1.1426e...	1.1706e...
4	1.0450e...	1.0453e...	1.0465e...	1.0491e...	1.0534e...	1.0596e...	1.0688e...	1.0822e...	1.0996e...	1.1198e...	1.1428e...	1.1706e...
5	1.0417e...	1.0421e...	1.0434e...	1.0461e...	1.0507e...	1.0574e...	1.0676e...	1.0821e...	1.1003e...	1.1210e...	1.1442e...	1.1716e...
6	1.0402e...	1.0407e...	1.0424e...	1.0457e...	1.0508e...	1.0582e...	1.0690e...	1.0841e...	1.1027e...	1.1236e...	1.1467e...	1.1737e...
7	1.0396e...	1.0402e...	1.0422e...	1.0460e...	1.0518e...	1.0598e...	1.0711e...	1.0865e...	1.1052e...	1.1262e...	1.1493e...	1.1763e...
8	1.0386e...	1.0393e...	1.0415e...	1.0456e...	1.0519e...	1.0605e...	1.0724e...	1.0880e...	1.1068e...	1.1278e...	1.1513e...	1.1785e...
9	1.0373e...	1.0379e...	1.0401e...	1.0446e...	1.0515e...	1.0608e...	1.0733e...	1.0892e...	1.1080e...	1.1291e...	1.1528e...	1.1804e...
10	1.0363e...	1.0370e...	1.0395e...	1.0444e...	1.0520e...	1.0621e...	1.0751e...	1.0912e...	1.1100e...	1.1311e...	1.1550e...	1.1826e...
11	1.0374e...	1.0383e...	1.0412e...	1.0466e...	1.0545e...	1.0652e...	1.0786e...	1.0948e...	1.1135e...	1.1345e...	1.1582e...	1.1853e...
12	1.0419e...	1.0429e...	1.0461e...	1.0517e...	1.0598e...	1.0705e...	1.0840e...	1.1001e...	1.1185e...	1.1391e...	1.1622e...	1.1883e...
13	1.0510e...	1.0521e...	1.0552e...	1.0605e...	1.0680e...	1.0783e...	1.0915e...	1.1074e...	1.1255e...	1.1455e...	1.1677e...	1.1922e...
14	1.0652e...	1.0661e...	1.0688e...	1.0732e...	1.0797e...	1.0889e...	1.1015e...	1.1171e...	1.1349e...	1.1542e...	1.1750e...	1.1976e...
15	1.0841e...	1.0848e...	1.0867e...	1.0900e...	1.0951e...	1.1030e...	1.1148e...	1.1298e...	1.1468e...	1.1649e...	1.1837e...	1.2035e...
16	1.1069e...	1.1073e...	1.1086e...	1.1109e...	1.1147e...	1.1213e...	1.1319e...	1.1457e...	1.1611e...	1.1769e...	1.1927e...	1.2086e...
17	1.1328e...	1.1331e...	1.1340e...	1.1356e...	1.1384e...	1.1438e...	1.1528e...	1.1645e...	1.1772e...	1.1896e...	1.2014e...	1.2128e...
18	1.1609e...	1.1611e...	1.1619e...	1.1632e...	1.1652e...	1.1693e...	1.1762e...	1.1851e...	1.1941e...	1		
19	1.1901e...	1.1904e...	1.1912e...	1.1921e...	1.1933e...	1.1957e...	1.2001e...	1.2055e...	1.2104e...	1		
20	1.2194e...	1.2196e...	1.2203e...	1.2207e...	1.2208e...	1.2214e...	1.2230e...	1.2248e...	1.2253e...	1		

Figure 2: Sample of normalized Pixel-Wise Energy Features Vectors

24x262144 double

	1	2	3	4	5	6	7	8	9
1	4.7896e-...	4.6294e-...	4.3738e-...	4.1902e-...	3.9329e-...	3.6512e-...	3.3682e-...	3.1246e-...	2.9034e-...
2	1.0169e-...	1.0090e-...	1.0003e-...	9.8759e-...	9.3987e-...	8.6621e-...	7.9007e-...	7.2263e-...	6.4624e-...
3	5.0740e-...	5.1195e-...	5.3068e-...	5.6311e-...	5.7442e-...	5.4517e-...	5.1299e-...	4.9912e-...	4.7926e-...
4	1.5003e-...	1.4968e-...	1.5227e-...	1.6607e-...	1.8863e-...	1.9953e-...	1.9951e-...	2.0063e-...	1.9804e-...
5	1.4876e-...	1.6433e-...	1.7365e-...	1.5967e-...	1.8507e-...	1.9724e-...	2.1356e-...	2.1707e-...	2.2155e-...
6	4.5369e-...	4.6361e-...	4.5678e-...	4.3298e-...	4.6965e-...	4.9974e-...	5.3036e-...	5.2370e-...	5.4547e-...
7	2.6488e-...	2.6740e-...	2.7942e-...	2.8050e-...	3.0121e-...	3.5381e-...	3.8589e-...	3.7474e-...	4.0260e-...
8	6.7388e-...	6.9405e-...	7.6749e-...	8.6406e-...	9.1246e-...	1.1204e-...	1.2970e-...	1.2448e-...	1.3289e-...
9	9.4560e-...	8.7181e-...	8.5178e-...	1.0960e-...	1.1503e-...	1.3939e-...	1.3949e-...	1.5455e-...	1.5263e-...
10	6.2660e-...	6.1968e-...	6.1500e-...	7.6320e-...	8.7579e-...	9.6814e-...	9.8965e-...	1.1310e-...	1.1003e-...
11	2.7249e-...	2.8068e-...	2.8961e-...	3.3115e-...	4.0876e-...	4.4449e-...	4.7297e-...	5.2928e-...	4.9391e-...
12	7.9076e-...	7.8258e-...	8.0203e-...	9.0762e-...	1.1571e-...	1.1243e-...	1.1185e-...	1.1383e-...	1.2057e-...
13	5.0050e-...	4.7979e-...	5.2305e-...	5.8829e-...	6.6922e-...	7.3692e-...	7.3707e-...	7.3397e-...	7.5907e-...
14	7.3292e-...	7.7072e-...	8.4513e-...	9.1336e-...	9.2114e-...	8.6682e-...	7.8860e-...	7.0489e-...	6.3058e-...
15	1.1835e-...	1.2497e-...	1.4100e-...	1.5305e-...	1.5423e-...	1.5302e-...	1.3981e-...	1.1687e-...	9.4706e-...
16	1.4101e-...	1.4589e-...	1.6002e-...	1.7293e-...	1.7379e-...	1.7237e-...	1.5520e-...	1.2723e-...	1.0063e-...
17	2.6846e-...	2.7550e-...	3.0200e-...	3.4929e-...	3.5424e-...	3.5565e-...	3.3423e-...	2.8263e-...	2.1023e-...
18	2.4412e-...	2.6031e-...	2.8171e-...	3.2593e-...	3.3179e-...	3.4241e-...	3.2487e-...	2.7829e-...	2.1245e-...
19	8.2287e-...	8.4587e-...	8.8981e-...	9.6937e-...	1.3655e-...	1.2883e-...	1.3113e-...	1.3328e-...	1.4303e-...
20	3.0838e-...	3.0164e-...	3.4611e-...	3.2948e-...	5.0120e-...	4.9385e-...	5.2255e-...	4.9383e-...	5.2280e-...
21	5.0075e-...	4.0293e-...	4.5124e-...	4.1296e-...	6.0778e-...	5.9982e-...	6.3509e-...	6.0134e-...	6.2746e-...
22	3.6488e-...	3.2603e-...	2.9198e-...	4.4962e-...	4.5165e-...	5.9241e-...	6.0245e-...	6.5292e-...	6.4067e-...
23	6.3703e-...	6.7386e-...	5.6602e-...	6.9657e-...	6.9830e-...	8.3211e-...	8.2997e-...	8.7303e-...	8.3049e-...
24	2.6347e-...	2.5381e-...	2.5874e-...	2.5211e-...	2.7309e-...	2.8793e-...	3.0921e-...	3.1176e-...	3.1429e-...

k_means_seg

512x512 double

	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486
143	1	1	1	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
144	1	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
145	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
146	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
147	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
148	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
149	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	4
150	4	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5
151	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5
152	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5
153	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5
154	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5
155	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5
156	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5	5
157	4	4	4	4	4	4	4	4	4	4	4	4	5	5	5	5	5	5
158	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
159	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
160	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
161	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
162	4	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4	4
163	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
164	1	1	1	1	1	1	1	1	1	1	1	4	4	4	4	4	4	4
165	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
166	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
167	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
168	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 3: Sample of K-means Segmentation Result

Figure 4: 15x15 Window Size Segmentation Result

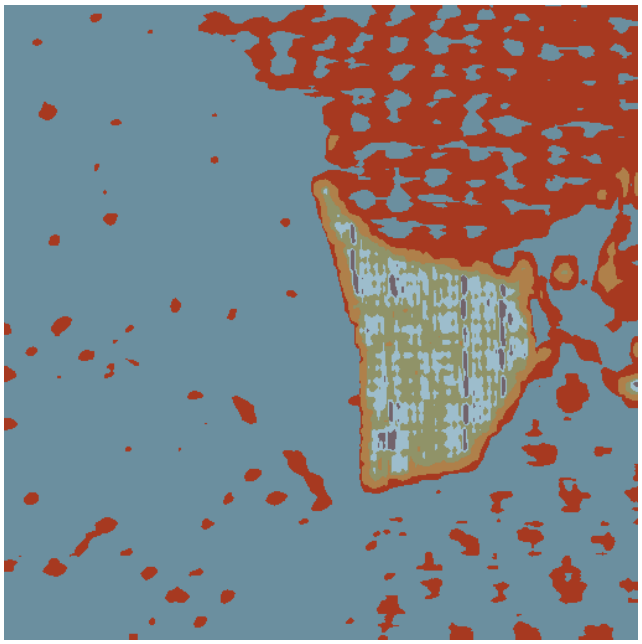


Figure 5: 15x15 Window Size Segmentation Filled Result with PCA

Figure 6: 35x35 Window Size Segmentation Result

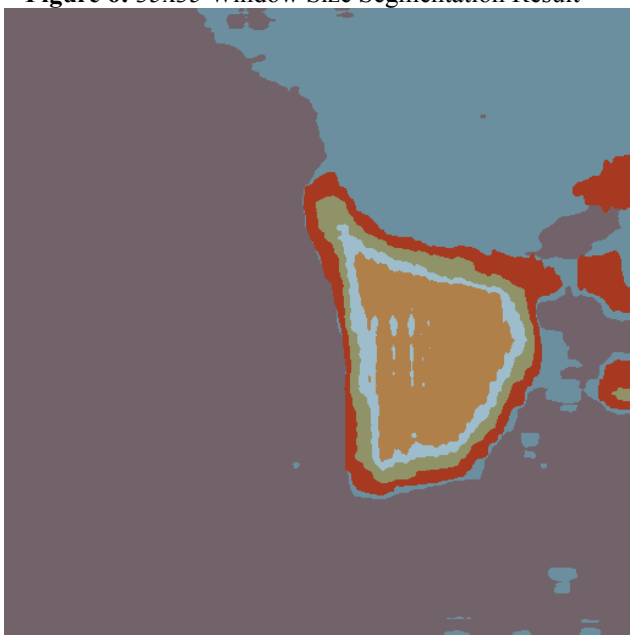
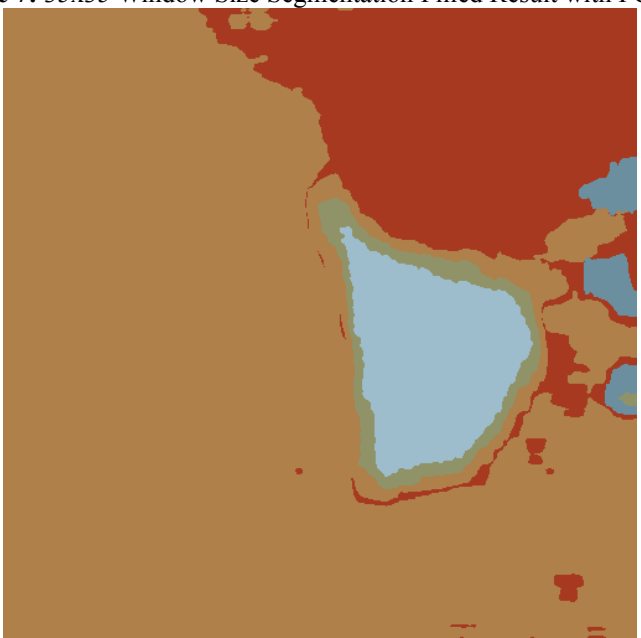
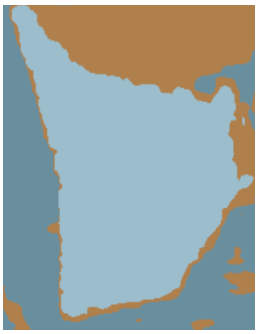


Figure 7: 35x35 Window Size Segmentation Filled Result with PCA



4. Discussion

The segmentation results with window size 15 by 15 is not really efficient at identifying entire regions of textures, but instead it finds the segmentations in smaller areas of a region. This makes sense because the window size is small enough to identify those areas as one texture. The segmentation results with window size 35 by 35 is much better at identifying the desired texture regions, in the resulting images we can separate about four different regions out of the six regions. Using the PCA function creates very small improvement for the segmentation. However, by filling the holes within each region (like the figure on the right), we get a much smoother segmentation. And we can avoid misidentifying inner textures of a region as a separate texture.



Problem 3: SIFT and Image Matching

1. Motivation

A crucial aspect of image processing is to find similarities between two or more images. We achieve this by obtaining something called SIFT features. These features are key points in images that are invariants to image rotation and scaling. This specific feature transform extraction algorithm was invented by David Lowe in 1999. It detects and matches local features of images that are similar regardless of scaling and rotation. There are many uses of SIFT today, some include object detection, orientation and image stitching [3].

2. Approaches and Procedures

b. Image Matching

In this problem, we use an open source Matlab toolbox called VLfeat to extract SIFT features from images. First, we read all four RGB images and convert them to grayscale. Then, we use the vl_sift function to get scale-invariant feature transform keypoints and feature descriptors for Cat_1 and Cat_dog images (**Figures 1,2**). Based on the keypoint with the highest scale we obtained for the Cat_1 image, we find the corresponding closest descriptor in the

Cat_dog image. This is our resulting SIFT pair for these two images. Lastly, we plot these two similar local features using vl_ploframe and vl_plotsiftdescriptor (**Figures 5,6**). We apply the same procedures onto the rest of the image pairs (**Figures 7-12**).

c. Bag of Words

After identifying all the feature descriptors using the SIFT function, we concatenate them into one matrix. Next, we use this total features matrix to train the K-means clustering algorithm (with K = 8). The trained K-means model is our codebook with 8 codewords representing 8 types of key patterns. In order to match Dog_2's Bag of Words representation with Cat_1 and Dog_1, we first calculate the Euclidean distance between all SIFT descriptors of these three images and the 8 cluster centers. Then, we determine which codeword each descriptor belongs to based on the nearest neighbor distance. Furthermore, we use the built-in histogram function to plot the frequency of each codeword in each image. In order to calculate the similarity between these histograms, we first find the CDF of each plot, then we compare each bin value of Cat_1 and Dog_1 with Dog_2 (**Figures 13-20**). Finally, we obtain the similarity score by summing all eight ratios between the minimum bin values and maximum bin values (**Figure 21**).

3. Experimental Results

a. Salient Point Descriptor

See Discussion section below.

b. Image Matching

Figure 1: SIFT Features of Cat_1

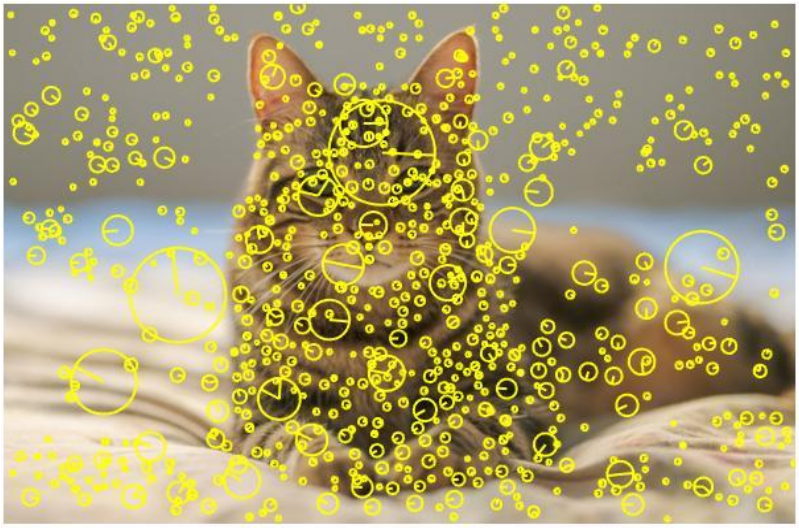
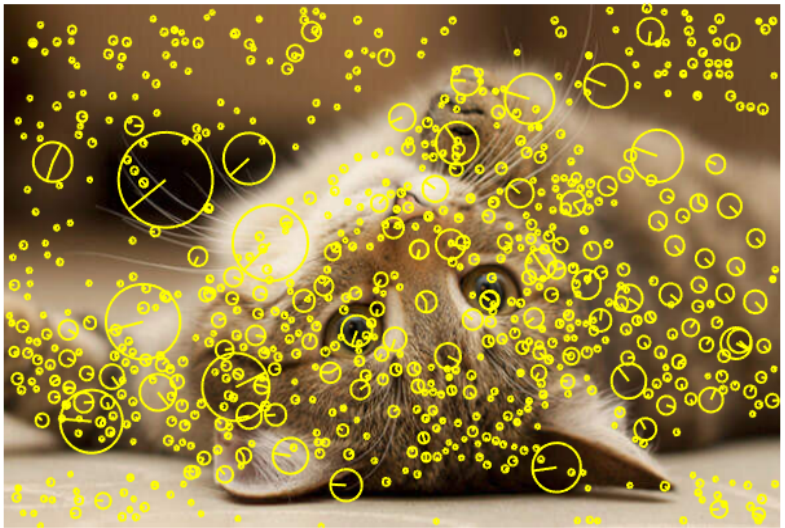


Figure 2: SIFT Features of Cat_2



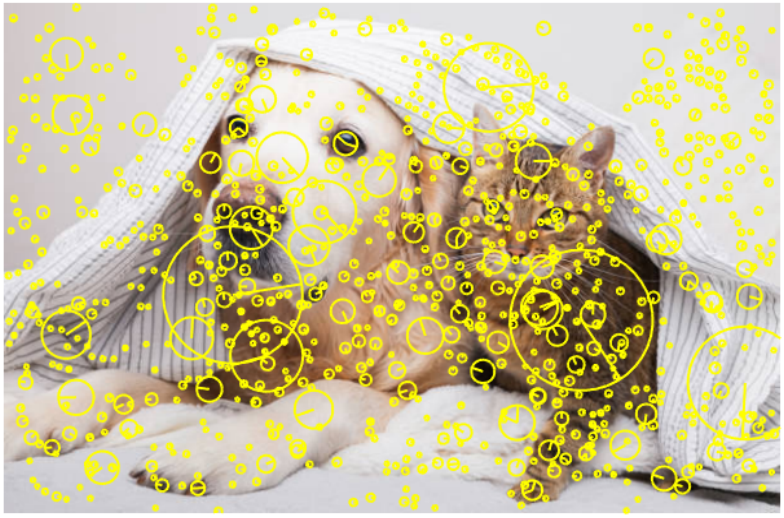


Figure 3: SIFT Features of Cat_dog

Figure 4: SIFT Features of Dog_1

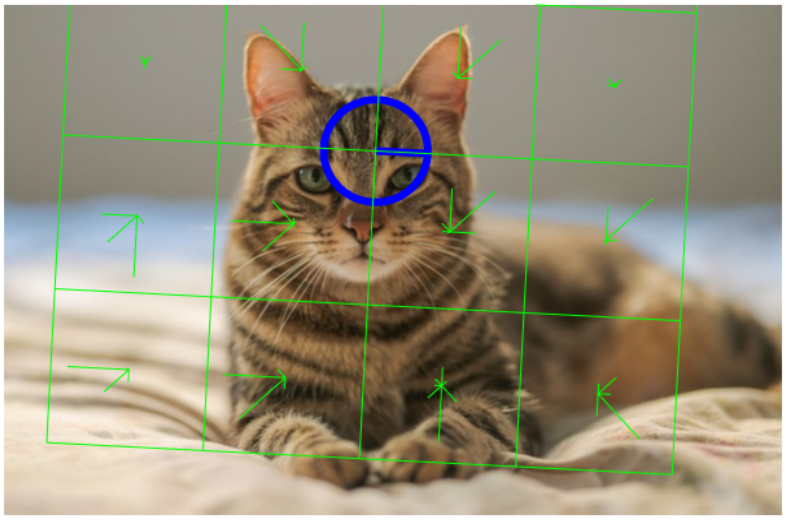


Figure 5: Matching Feature of Cat_1

Figure 6: Matching Feature of Cat_dog

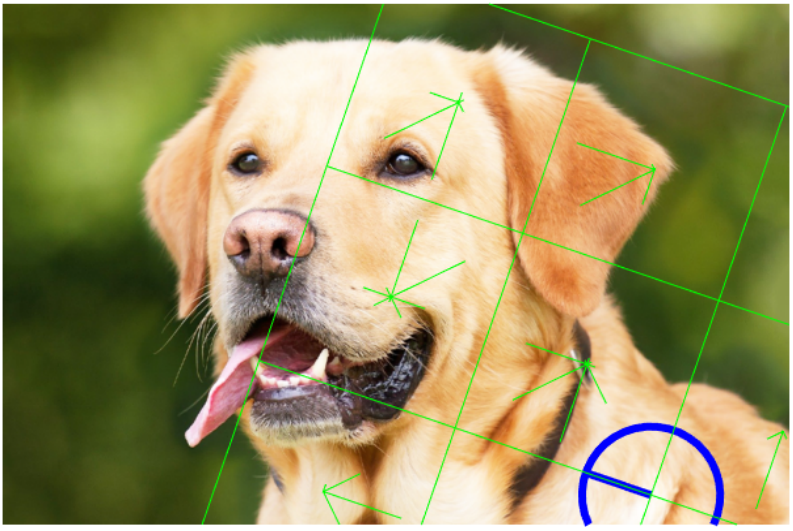


Figure 7: Matching Feature of Dog_1

Figure 8: Matching Feature of Cat_dog

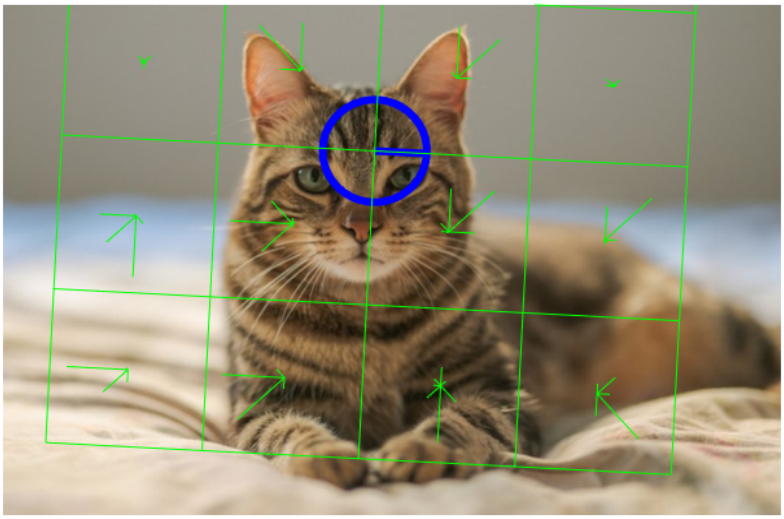


Figure 9: Matching Feature of Cat_1

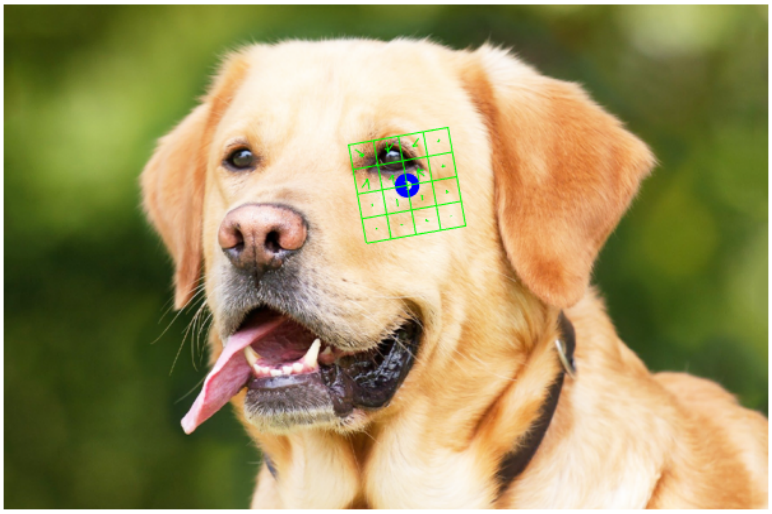


Figure 10: Matching Feature of Dog_1

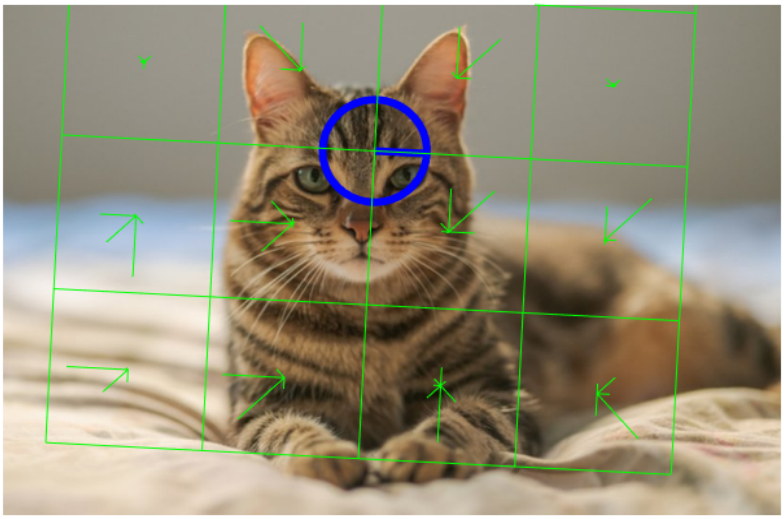


Figure 11: Matching Feature of Cat_1

Figure 12: Matching Feature of Cat_2

c. Bag of Words

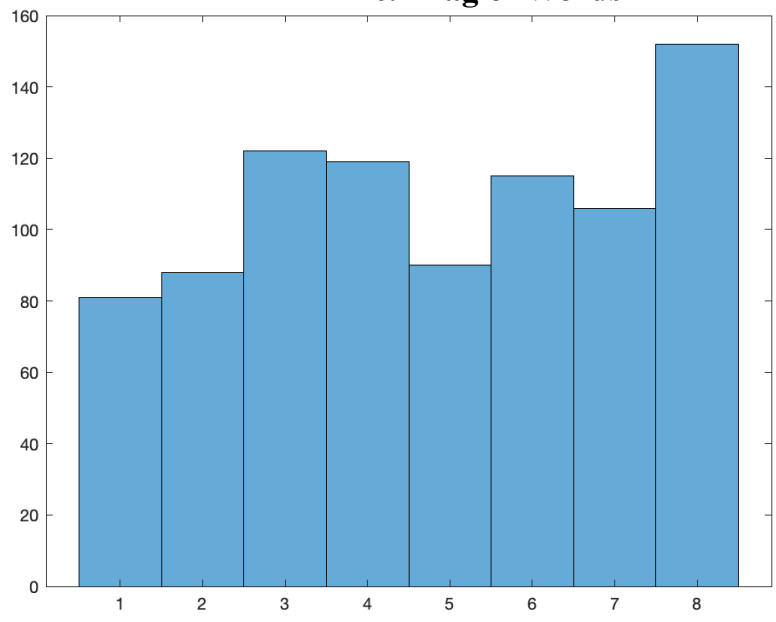


Figure 13: Cat_1 BoW Histogram

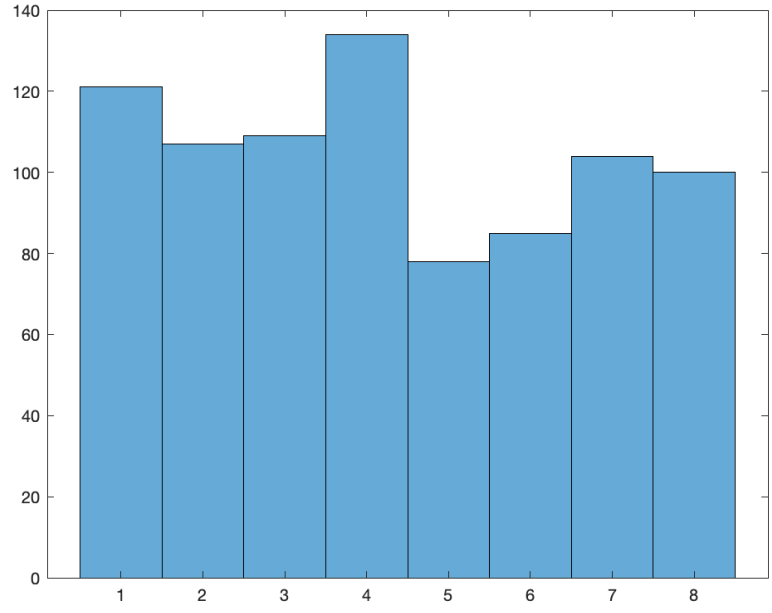


Figure 14: Cat_2 BoW Histogram

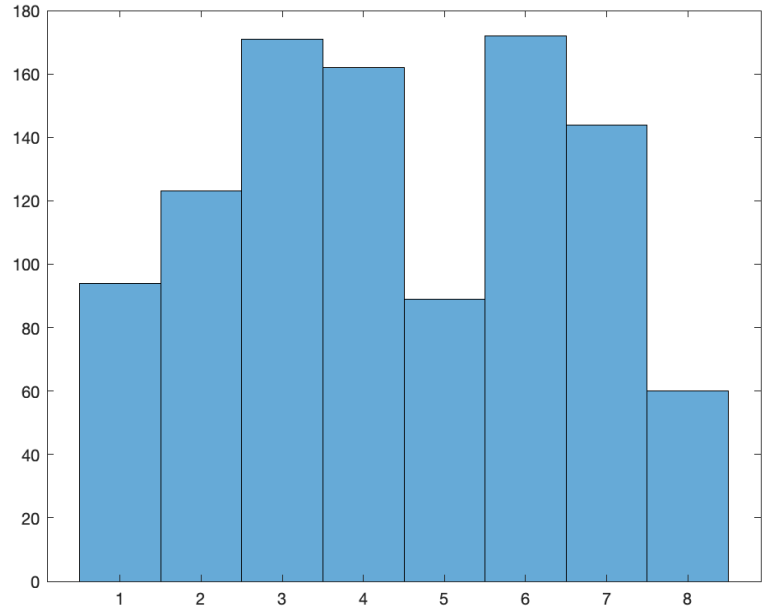


Figure 15: Cat_dog BoW Histogram

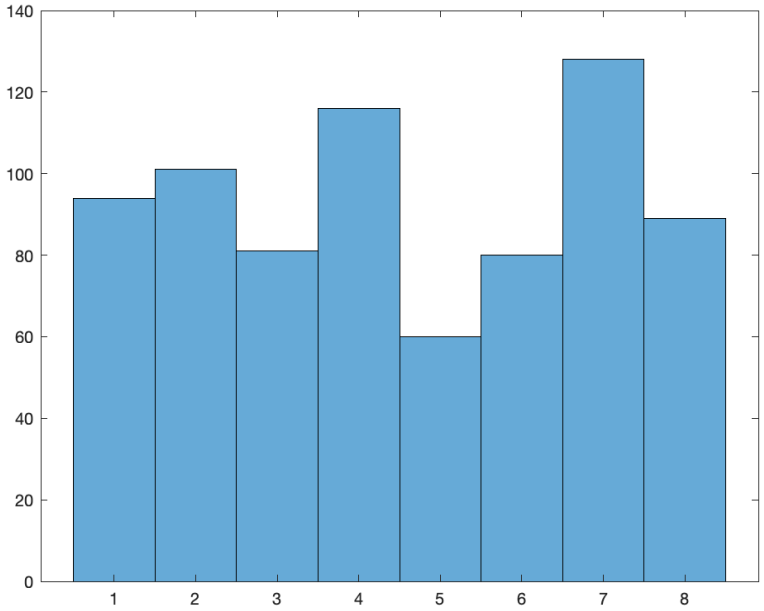


Figure 16: Dog_1 BoW Histogram

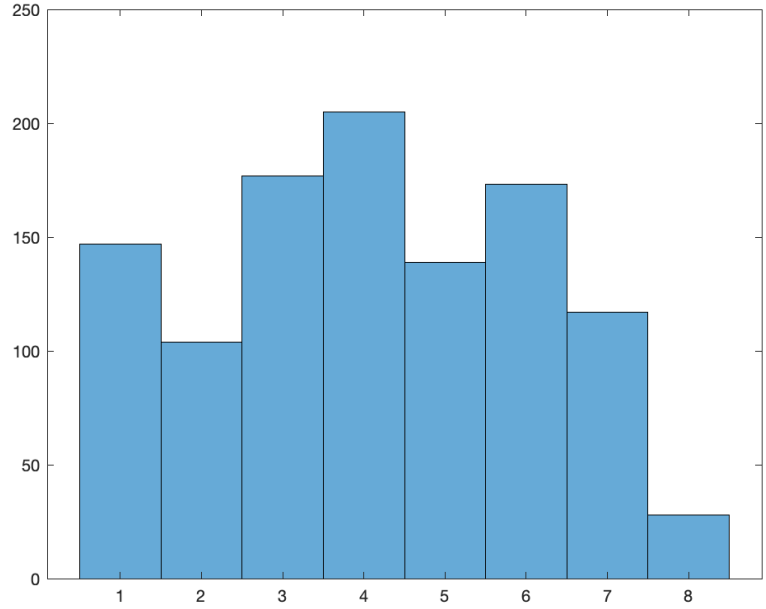


Figure 17: Dog_2 BoW Histogram

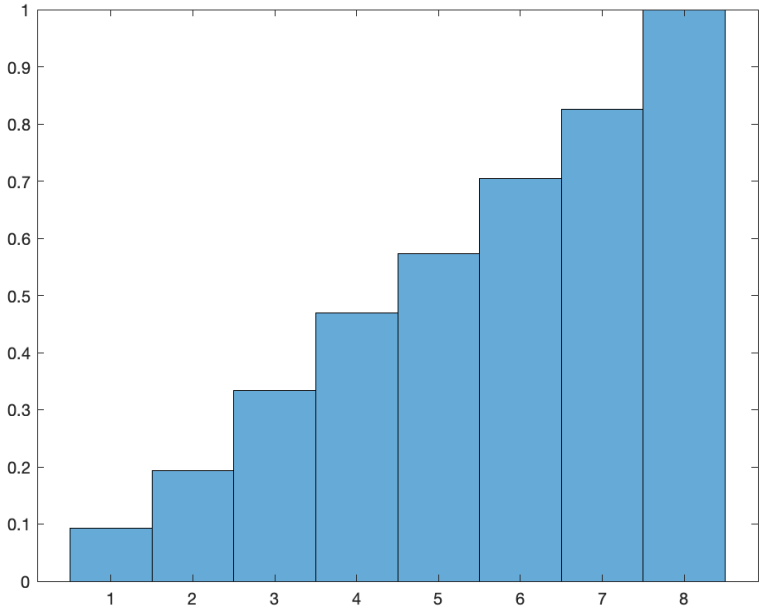
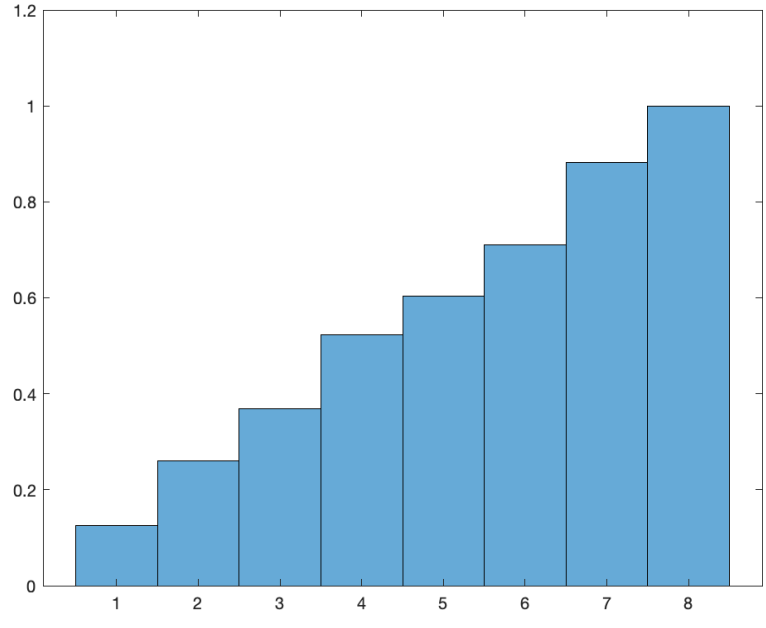
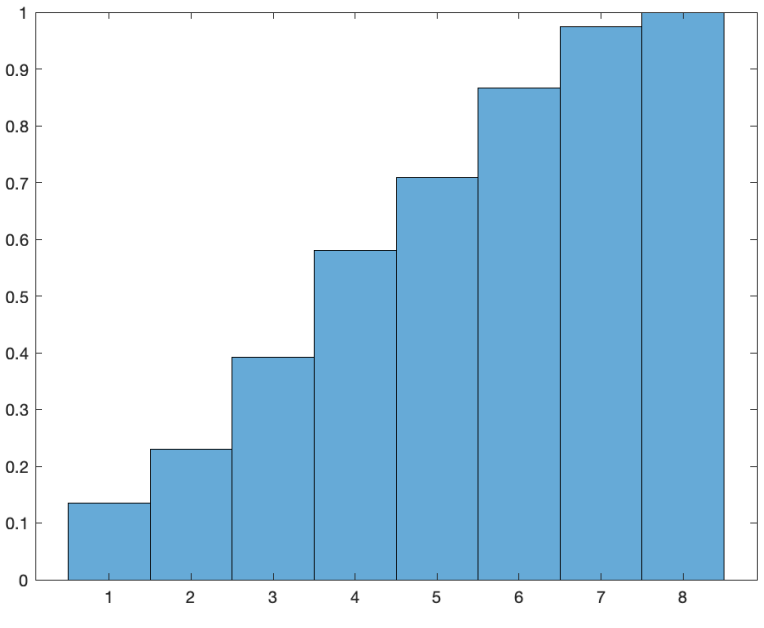


Figure 18: Cat_1 CDF Histogram

Figure 19: Dog_1 CDF Histogram

Figure 20: Dog_2 CDF Histogram





```
"The similarity between Cat_1 Image an…"    "0.85769"
"The similarity between Dog_1 Image an…"    "0.9033"
```

Figure 21: Histogram Similarity Comparison

4. Discussion

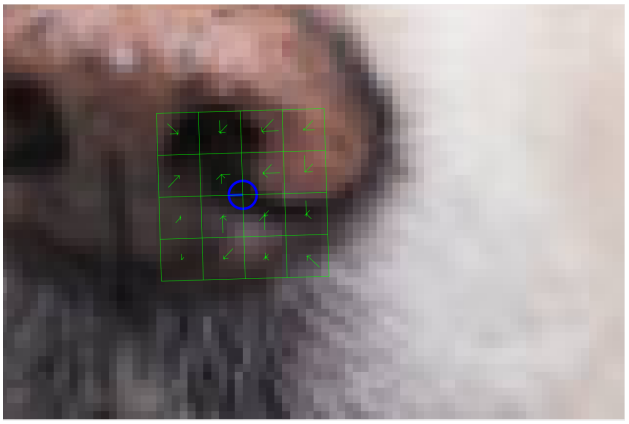
a. 1). In the paper abstract, the author Lowe mentions that the SIFT features are robust to Image Scaling and Rotation [4].

2). SIFT achieves its robustness to Image Scaling and Image Rotations by selecting well localized and highly distinctive features as keypoints. In order to select these unique features, the algorithm uses a cascade filtering approach. The first step is to identify potential key points by searching all image locations with all possible scales and comparing their different levels of gaussian. Then, at each desired location, a more sophisticated model is used to determine the stability and fit for a keypoint. Furthermore, orientation is assigned at each keypoint based on the gradient directions, which ensures the robustness regarding rotation resampling using random angels [4].

3). SIFT generates a vector containing the descriptors of each keypoint containing values of orientation details. To obtain robustness to illumination changes, this feature vector is normalized. This normalization would further normalize the gradient values at each image location. Meaning that a constant value applied to change image contrast will not affect these gradient values as they are calculated between pixel values. Thus, SIFT feature vectors are invariant to illumination changes [4].

4). Laplacian of Gaussian without scale-normalization is not truly invariant to image scaling. However, the Difference of Gaussian achieves this scale-normalized LoG by performing a sigma squared normalization on top of the Laplacian gaussians. Thus, producing more stable features to withstand image scaling [4].

5). The SIFT's output descriptor vector size provided in the paper is $r * n^2$, where r is the frequency of orientations and n is the width of the array containing the orientations histograms [4].



b. The resulting SIFT pair in **figure 5 and 6** are not the most expected results. It would make more intuitive sense if the head region of the cat in the Cat_dog image was matched with the feature in Cat_1. The reason why they did not match is because the fur pattern of the second cat is significantly different from the first. If we take a closer look at the SIFT key point of the second image, we can see that the orientation draws similarity to the cat image. For the rest of the SIFT pair matchings (7-10) was not able to successfully match good features, one reason could be the huge contrast differences and large viewing angle differences. Matching between 11 and 12 is decent as it identifies similar patterns but it is difficult to tell due to blur effects.

c. The Bag of Words representations for each image is shown in **figures 13-17**. However, this might not be the result every time due to K-means being an unsupervised algorithm. From the CDF histogram (**figures 18-20)** and the similarity index calculation (**figure 21**), we can conclude that the Dog_1 image gives better matching to the Dog_2 image than the Cat_1 image.

Image References

All images are either directly given in the homework description/discussion slides or are the results of my programs.

References

[1]: University of Waterloo, Vision and Image Processing Lab, Texture Classification, https://uwaterloo.ca/vision-image-processing-lab/research-demos/texture-classification#:~:text=Texture%20Classification%20is%20the%20problem,from%20a%20given%20textured%20image.

[2]: 3D Projection, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/3D_projection

[3]: Scale-Invariant Feature Transform, From Wikipedia, the free encyclopedia, https://en.wikipedia.org/wiki/Scale-invariant_feature_transform

[4]: David G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", 2004, https://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf