Scene Categorization

April 16, 2019

1 Scene Categorization

1.0.1 Importing the required libraries:

```
In [0]: import numpy as np
        import matplotlib.image as mpimg
        from matplotlib import pyplot as plt
        from sklearn.cluster import KMeans
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.metrics import accuracy_score, confusion_matrix
        from seaborn import heatmap
        import pandas as pd
        from collections import Counter
```

1.0.2 Mounting drive to Google Colab

Having the dataset in the google drive, we can mount it to be able to access from here.

```
In [0]: from google.colab import drive
        drive.mount("/content/drive")
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6

Enter your authorization code:
ůůůůůůůůůů
Mounted at /content/drive
```

1.1 Reading the SIFT-Features:

- Reading all the Sift features from Training data as well as Testing data.
- Storing them as dataFrames.

```
In [0]: dataFrames = [pd.read_csv(
            "/content/drive/My Drive/HW3_data/train_sift_features/1_train_sift.csv",
            header = None).drop(columns = [0,1,2,3])]
        for i in range(2, 1889):
          if(i % 100 == 0):
```

```
        print(i)
      dataFrames.append(pd.read_csv(
     "/content/drive/My Drive/HW3_data/train_sift_features/"+str(i)+"_train_sift.csv",
          header = None).drop(columns = [0,1,2,3]))
    data = pd.concat(dataFrames, axis = 0, ignore_index = True)
```

```
In [0]: testDataFrames = [pd.read_csv(
            "/content/drive/My Drive/HW3_data/test_sift_features/1_test_sift.csv",
            header = None).drop(columns = [0,1,2,3])]
        for i in range(2, 801):
          if(i % 100 == 0):
            print(i)
          testDataFrames.append(pd.read_csv(
           "/content/drive/My Drive/HW3_data/test_sift_features/"+str(i)+"_test_sift.csv",
              header = None).drop(columns = [0,1,2,3]))
```

```
In [0]: data
```

Out[0]:

	4	5	6	7	8	9	10	11	12	13	...	122	123	124	\
0	200	13	0	0	0	0	0	2	49	1	...	0	3	7	
1	5	0	0	0	0	0	0	0	18	0	...	0	6	11	
2	1	2	0	15	91	0	0	0	48	11	...	0	18	72	
3	0	0	0	10	130	79	3	1	67	13	...	15	115	44	
4	5	3	3	17	18	47	31	7	12	3	...	6	18	4	
5	0	0	0	22	136	34	0	0	64	1	...	19	71	69	
6	34	104	49	8	0	1	4	7	13	28	...	8	1	7	
7	0	0	3	32	94	81	23	0	58	21	...	0	8	46	
8	130	23	0	0	0	1	1	45	66	12	...	1	8	2	
9	1	0	0	3	32	27	14	2	12	14	...	13	37	147	
10	0	0	3	2	3	61	117	27	11	7	...	12	21	117	
11	56	0	0	0	102	40	2	16	153	1	...	2	6	14	
12	24	7	0	0	134	38	0	0	144	50	...	1	2	45	
13	0	1	0	1	140	37	0	0	109	29	...	6	65	57	
14	53	25	3	0	0	0	3	61	10	21	...	96	33	39	
15	28	5	0	1	6	6	9	39	126	13	...	102	126	16	
16	14	10	13	0	13	6	0	5	132	13	...	0	4	4	
17	11	19	8	0	1	3	26	52	42	48	...	2	4	10	
18	23	4	3	2	4	14	22	58	192	3	...	0	6	6	
19	42	128	18	0	0	0	0	0	10	79	...	6	8	40	
20	2	15	15	17	49	22	5	2	102	30	...	7	83	39	
21	115	20	0	0	1	4	1	27	29	8	...	27	27	102	
22	53	1	0	0	0	0	0	4	159	28	...	6	31	16	
23	19	7	3	0	0	5	58	52	62	6	...	50	12	1	
24	8	8	5	21	66	59	13	16	91	2	...	0	42	62	
25	5	0	27	124	17	7	40	107	0	0	...	8	50	4	
26	7	0	3	17	20	37	49	24	105	1	...	18	19	0	
27	13	0	0	0	35	89	76	14	100	54	...	0	1	10	
28	21	102	25	0	6	50	1	0	5	52	...	3	13	37	

29	50	0	0	0	0	0	0	9	163	13	...	1	8	19
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
413941	0	0	0	3	20	87	25	0	40	4	...	7	48	5
413942	1	7	5	3	0	4	122	19	2	2	...	21	5	0
413943	27	12	2	3	7	37	47	38	21	5	...	2	11	1
413944	3	2	8	136	55	7	1	0	71	34	...	0	17	17
413945	20	0	0	0	0	0	0	25	35	4	...	30	129	21
413946	0	0	0	0	0	9	113	18	0	0	...	0	126	12
413947	7	6	8	46	107	38	7	1	12	5	...	0	19	92
413948	101	2	0	0	0	0	0	7	132	14	...	0	0	107
413949	4	29	38	32	28	20	17	3	4	10	...	0	10	0
413950	132	54	2	3	25	17	14	47	6	9	...	0	0	0
413951	0	0	0	0	0	0	0	0	0	0	...	21	13	43
413952	0	0	0	0	0	0	0	0	0	0	...	0	0	151
413953	0	0	0	0	0	0	0	0	0	0	...	0	2	9
413954	198	22	0	0	0	0	0	0	75	13	...	94	49	1
413955	17	0	0	0	4	15	10	7	116	24	...	17	38	116
413956	23	0	0	0	0	0	1	32	35	2	...	31	133	16
413957	0	0	0	1	4	24	112	15	2	1	...	0	79	7
413958	0	0	0	0	0	0	114	124	5	6	...	112	6	0
413959	50	3	0	0	0	0	0	4	66	18	...	0	0	9
413960	0	0	0	0	12	29	76	88	0	0	...	77	40	0
413961	81	15	8	21	7	0	0	36	20	7	...	13	104	23
413962	18	22	39	3	2	14	120	105	30	56	...	6	0	0
413963	69	0	0	0	0	0	0	41	101	0	...	2	1	2
413964	0	0	0	0	0	4	5	0	0	0	...	21	24	64
413965	0	0	0	0	0	0	0	0	0	0	...	3	62	140
413966	25	9	3	10	8	3	36	140	98	82	...	0	2	0
413967	118	0	0	0	0	18	26	32	9	0	...	49	54	1
413968	2	0	0	1	3	26	13	11	18	6	...	74	50	13
413969	0	0	0	0	0	0	0	0	51	4	...	114	85	0
413970	14	46	38	7	0	1	4	9	10	35	...	0	0	0

	125	126	127	128	129	130	131
0	0	1	3	6	6	1	7
1	12	21	13	5	2	5	24
2	2	0	0	0	0	0	15
3	16	11	2	2	9	6	19
4	41	43	5	6	18	10	1
5	0	0	0	0	0	0	33
6	2	3	7	64	104	11	4
7	65	13	1	4	3	0	2
8	11	10	25	43	10	5	1
9	4	2	1	0	0	16	104
10	84	13	20	6	33	109	40
11	68	8	0	0	0	0	0
12	52	5	0	0	2	1	2
13	5	2	5	5	7	5	12

14	3	0	0	2	3	56	69
15	0	0	2	23	22	39	126
16	29	48	13	34	34	1	1
17	2	0	3	91	125	5	2
18	1	0	4	13	9	1	1
19	25	0	0	18	72	31	11
20	4	3	44	50	27	15	27
21	2	0	0	0	0	8	77
22	4	8	10	9	40	28	33
23	0	0	0	0	1	48	56
24	1	1	6	1	0	0	10
25	15	37	17	3	7	6	1
26	0	0	59	122	9	2	0
27	36	73	33	14	10	0	0
28	121	2	0	0	0	1	1
29	125	42	5	0	0	0	0
...	...	...	...	...	...	...	...
413941	2	0	3	13	6	2	4
413942	9	25	65	63	69	10	0
413943	2	2	6	6	4	3	0
413944	6	4	3	2	1	0	1
413945	2	0	2	18	47	87	65
413946	1	10	13	4	0	0	16
413947	17	2	0	1	0	0	24
413948	19	0	0	0	0	0	0
413949	0	0	0	0	0	0	0
413950	0	0	0	0	0	0	0
413951	5	2	5	21	28	44	132
413952	26	0	0	0	0	0	0
413953	33	11	1	16	46	8	2
413954	0	0	0	116	39	31	53
413955	55	1	1	0	0	0	6
413956	1	0	0	9	30	112	87
413957	0	12	27	3	0	0	6
413958	0	0	0	0	8	58	1
413959	4	1	0	0	0	0	1
413960	0	0	0	0	34	103	13
413961	21	16	19	1	0	0	6
413962	0	0	0	0	54	7	0
413963	0	0	8	36	47	17	13
413964	25	5	1	0	1	17	58
413965	140	121	12	0	1	3	57
413966	0	0	0	0	0	0	0
413967	0	0	0	19	0	0	2
413968	3	8	26	11	18	32	48
413969	0	0	0	121	49	3	1
413970	0	0	2	0	0	0	0

```
[413971 rows x 128 columns]
```

Since we need only the unique sift features for bag of words representation, we will drop duplicates.

```
In [0]: data = data.drop_duplicates()
        data
```

Out[0]:

	4	5	6	7	8	9	10	11	12	13	...	122	123	124	\
0	200	13	0	0	0	0	0	2	49	1	...	0	3	7	
1	5	0	0	0	0	0	0	0	18	0	...	0	6	11	
2	1	2	0	15	91	0	0	0	48	11	...	0	18	72	
3	0	0	0	10	130	79	3	1	67	13	...	15	115	44	
4	5	3	3	17	18	47	31	7	12	3	...	6	18	4	
5	0	0	0	22	136	34	0	0	64	1	...	19	71	69	
6	34	104	49	8	0	1	4	7	13	28	...	8	1	7	
7	0	0	3	32	94	81	23	0	58	21	...	0	8	46	
8	130	23	0	0	0	1	1	45	66	12	...	1	8	2	
9	1	0	0	3	32	27	14	2	12	14	...	13	37	147	
10	0	0	3	2	3	61	117	27	11	7	...	12	21	117	
11	56	0	0	0	102	40	2	16	153	1	...	2	6	14	
12	24	7	0	0	134	38	0	0	144	50	...	1	2	45	
13	0	1	0	1	140	37	0	0	109	29	...	6	65	57	
14	53	25	3	0	0	0	3	61	10	21	...	96	33	39	
15	28	5	0	1	6	6	9	39	126	13	...	102	126	16	
16	14	10	13	0	13	6	0	5	132	13	...	0	4	4	
17	11	19	8	0	1	3	26	52	42	48	...	2	4	10	
18	23	4	3	2	4	14	22	58	192	3	...	0	6	6	
19	42	128	18	0	0	0	0	0	10	79	...	6	8	40	
20	2	15	15	17	49	22	5	2	102	30	...	7	83	39	
21	115	20	0	0	1	4	1	27	29	8	...	27	27	102	
22	53	1	0	0	0	0	0	4	159	28	...	6	31	16	
23	19	7	3	0	0	5	58	52	62	6	...	50	12	1	
24	8	8	5	21	66	59	13	16	91	2	...	0	42	62	
25	5	0	27	124	17	7	40	107	0	0	...	8	50	4	
26	7	0	3	17	20	37	49	24	105	1	...	18	19	0	
27	13	0	0	0	35	89	76	14	100	54	...	0	1	10	
28	21	102	25	0	6	50	1	0	5	52	...	3	13	37	
29	50	0	0	0	0	0	0	9	163	13	...	1	8	19	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
413941	0	0	0	3	20	87	25	0	40	4	...	7	48	5	
413942	1	7	5	3	0	4	122	19	2	2	...	21	5	0	
413943	27	12	2	3	7	37	47	38	21	5	...	2	11	1	
413944	3	2	8	136	55	7	1	0	71	34	...	0	17	17	
413945	20	0	0	0	0	0	0	25	35	4	...	30	129	21	
413946	0	0	0	0	0	9	113	18	0	0	...	0	126	12	
413947	7	6	8	46	107	38	7	1	12	5	...	0	19	92	
413948	101	2	0	0	0	0	0	7	132	14	...	0	0	107	

413949	4	29	38	32	28	20	17	3	4	10	...	0	10	0
413950	132	54	2	3	25	17	14	47	6	9	...	0	0	0
413951	0	0	0	0	0	0	0	0	0	0	...	21	13	43
413952	0	0	0	0	0	0	0	0	0	0	...	0	0	151
413953	0	0	0	0	0	0	0	0	0	0	...	0	2	9
413954	198	22	0	0	0	0	0	0	75	13	...	94	49	1
413955	17	0	0	0	4	15	10	7	116	24	...	17	38	116
413956	23	0	0	0	0	0	1	32	35	2	...	31	133	16
413957	0	0	0	1	4	24	112	15	2	1	...	0	79	7
413958	0	0	0	0	0	0	114	124	5	6	...	112	6	0
413959	50	3	0	0	0	0	0	4	66	18	...	0	0	9
413960	0	0	0	0	12	29	76	88	0	0	...	77	40	0
413961	81	15	8	21	7	0	0	36	20	7	...	13	104	23
413962	18	22	39	3	2	14	120	105	30	56	...	6	0	0
413963	69	0	0	0	0	0	0	41	101	0	...	2	1	2
413964	0	0	0	0	0	4	5	0	0	0	...	21	24	64
413965	0	0	0	0	0	0	0	0	0	0	...	3	62	140
413966	25	9	3	10	8	3	36	140	98	82	...	0	2	0
413967	118	0	0	0	0	18	26	32	9	0	...	49	54	1
413968	2	0	0	1	3	26	13	11	18	6	...	74	50	13
413969	0	0	0	0	0	0	0	0	51	4	...	114	85	0
413970	14	46	38	7	0	1	4	9	10	35	...	0	0	0

	125	126	127	128	129	130	131
0	0	1	3	6	6	1	7
1	12	21	13	5	2	5	24
2	2	0	0	0	0	0	15
3	16	11	2	2	9	6	19
4	41	43	5	6	18	10	1
5	0	0	0	0	0	0	33
6	2	3	7	64	104	11	4
7	65	13	1	4	3	0	2
8	11	10	25	43	10	5	1
9	4	2	1	0	0	16	104
10	84	13	20	6	33	109	40
11	68	8	0	0	0	0	0
12	52	5	0	0	2	1	2
13	5	2	5	5	7	5	12
14	3	0	0	2	3	56	69
15	0	0	2	23	22	39	126
16	29	48	13	34	34	1	1
17	2	0	3	91	125	5	2
18	1	0	4	13	9	1	1
19	25	0	0	18	72	31	11
20	4	3	44	50	27	15	27
21	2	0	0	0	0	8	77
22	4	8	10	9	40	28	33
23	0	0	0	0	1	48	56

```
24        1    1    6    1    0    0   10
25       15   37   17    3    7    6    1
26        0    0   59  122    9    2    0
27       36   73   33   14   10    0    0
28      121    2    0    0    0    1    1
29      125   42    5    0    0    0    0
...     ...  ...  ...  ...  ...  ...  ...
413941    2    0    3   13    6    2    4
413942    9   25   65   63   69   10    0
413943    2    2    6    6    4    3    0
413944    6    4    3    2    1    0    1
413945    2    0    2   18   47   87   65
413946    1   10   13    4    0    0   16
413947   17    2    0    1    0    0   24
413948   19    0    0    0    0    0    0
413949    0    0    0    0    0    0    0
413950    0    0    0    0    0    0    0
413951    5    2    5   21   28   44  132
413952   26    0    0    0    0    0    0
413953   33   11    1   16   46    8    2
413954    0    0    0  116   39   31   53
413955   55    1    1    0    0    0    6
413956    1    0    0    9   30  112   87
413957    0   12   27    3    0    0    6
413958    0    0    0    0    8   58    1
413959    4    1    0    0    0    0    1
413960    0    0    0    0   34  103   13
413961   21   16   19    1    0    0    6
413962    0    0    0    0   54    7    0
413963    0    0    8   36   47   17   13
413964   25    5    1    0    1   17   58
413965  140  121   12    0    1    3   57
413966    0    0    0    0    0    0    0
413967    0    0    0   19    0    0    2
413968    3    8   26   11   18   32   48
413969    0    0    0  121   49    3    1
413970    0    0    2    0    0    0    0

[413672 rows x 128 columns]
```

It was observed that there were 299 duplicates.

```
In [0]: data = np.array(data)
        data.shape
```

```
Out[0]: (413672, 128)
```

- Now that we have 4 lakh unique features, it would be difficult to represent each image as a Bag of Words representation of these many features.

- Hence, we do K-Means clustering to have a limited number of features (Bag of Visual words).
- We now have 30 lengthed vector (bag of quantised local visual features).

```
In [0]: ans = KMeans(30, max_iter = 4).fit(data)
```

```
In [0]: ans
```

```
Out[0]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=4,
            n_clusters=30, n_init=10, n_jobs=None, precompute_distances='auto',
            random_state=None, tol=0.0001, verbose=0)
```

1.1.1 Now, based on our KMeans model, all our sift-features are fit into this.

```
In [0]: ans.labels_.shape
```

```
Out[0]: (413672,)
```

1.1.2 The first few sift-features are displayed below.

1.1.3 Each one is mapped to one of the 30 classes.

```
In [0]: ans.labels_[:200]
```

```
Out[0]: array([25, 11, 21, 21, 22, 21, 24, 18,  7, 11, 27, 23, 23, 12, 28, 16, 15,
               18, 15, 20, 14, 11,  4, 18, 21, 13, 29, 18,  9,  7,  1, 26, 12, 24,
               10, 11, 19, 16,  8, 18,  7, 14, 22,  2,  1, 20,  7, 25,  4,  1, 15,
               21, 26, 21, 14, 29, 15, 21,  4,  2, 15, 21,  4, 12,  9,  3, 23,  1,
               15, 24,  4,  3, 22, 25, 27, 20, 16,  8, 26, 15,  0, 18, 15, 21, 10,
                1, 20, 21,  9,  9, 10,  1, 29,  3, 29, 15,  5, 16,  3, 15, 15, 20,
               29,  4, 24, 22, 29, 21, 21, 13, 16,  0,  8, 24,  1,  4, 26,  4, 27,
                7, 15, 13,  5,  8, 21,  7, 10, 26,  3, 13, 29,  7,  6,  4, 21, 21,
                7, 16, 24, 27,  0, 18, 11, 22,  7,  8,  3, 20,  1, 29,  3, 23, 29,
                7, 29, 21,  2, 26, 27, 20, 24,  4, 23,  9,  7,  0,  8, 18, 29,  3,
                1, 11, 24, 21, 20, 19, 29,  7, 10, 29, 22, 17, 28,  6, 25, 12, 19,
               15,  0, 28, 16,  0, 20, 20, 18, 21, 21, 17, 14,  0], dtype=int32)
```

```
In [0]: N = len(dataFrames)
        train_counters = list(map(lambda x: Counter(ans.predict(np.array(x))), dataFrames))
        train_histograms = np.zeros((N, 30))
        for i in range(N):
          for key in train_counters[i]:
            train_histograms[i][key]=train_counters[i][key]
```

1.1.4 Now for each of the training examples, we get a 30-lengthed vector (a histogram), which stores the count of sift-features getting mapped to one of the 30 clusters.

```
In [0]: train_histograms.shape
```

```
Out[0]: (1888, 30)
```

```
In [0]: N_ = len(testDataFrames)
        test_counters = list(map(lambda x: Counter(ans.predict(np.array(x))), testDataFrames))
        test_histograms = np.zeros((N_, 30))
        for i in range(N_):
          for key in test_counters[i]:
            test_histograms[i][key] = test_counters[i][key]

In [0]: test_histograms.shape, test_histograms

Out[0]: array([[14.,  7.,  6., ..., 12.,  6.,  5.],
               [ 1.,  3.,  3., ...,  3.,  1.,  6.],
               [ 4., 16.,  3., ...,  3.,  4.,  7.],
               ...,
               [23.,  5.,  2., ..., 15., 37.,  2.],
               [18.,  6.,  2., ...,  7., 19., 14.],
               [ 0.,  2.,  8., ...,  0.,  2.,  3.]])
```

1.1.5 We now obtain the labels for all the training images which we will use to classify the test images based on K-nearest Neighbour classifier.

- Here we took k as 10 (10-Nearest Neighbour classifier).

```
In [0]: train_labels = np.array(
            pd.read_csv("/content/drive/My Drive/HW3_data/train_labels.csv",
                        header = None))[0]
        test_labels = np.array(
            pd.read_csv("/content/drive/My Drive/HW3_data/test_labels.csv",
                        header = None))[0]

In [0]: knn = KNeighborsClassifier(n_neighbors = 10).fit(train_histograms, train_labels)
```

1.2 Now, for each test-image, we predict the class label based on K-Nearest Neighbour classifier and then generate the Confusion Matrix and find the Accuracy.

```
In [0]: #test_classes = list(map())
        test_classes=[]
        for i in test_histograms:
          test_classes.append(knn.predict([i])[0])
        test_classes = np.array(test_classes)
        print("Accuracy: ", accuracy_score(test_labels, test_classes)*100, "%")
        print("Confusion matrix:")
        heatmap(confusion_matrix(test_labels, test_classes), cmap="gist_rainbow")
        plt.xlabel('True Values')
        plt.ylabel('Predicted Values')
        plt.show()
        #count = 0
        #for i in range(N_):
        #  if(test_classes[i] == test_labels[i]):
        #    count+=1
```