

Student Name: Christopher Seagraves Student ID: _____

Data set: the NWPU aerial data set contains approximately 45 categories of 700 images for each category in 256x256 RGB format. The data set can be downloaded from:

<https://umkc.box.com/s/fxvzh5qq2tiob6eklfxwn89kg3e1io1>



For HW-2, let us just take the first 15 classes (shown above) {airplane, airport, bridge, ..., freeway} from the full data set, and 60 images per class and form a data set of $N=900$ images with 15 labels. Objectives are using the gradient image features (SIFT, Dense SIFT) with aggregation (VLAD, FisherVector) to test which one can give us the best solution.

[1] Gradient Features, for each image, implement the following function in either Matlab and Python, show your implementation [25pts]

```
% im - input images, let us make them all grayscale only, so it is a h x w matrix
% opt.type = { 'sift' , 'hogf' , 'dsft' } for sift and densesift
% f - n x d matrix containing n features of d dimension, d=128 for SIFT, for e.g,
function [f]=getImageFeatures(im, opt)
.....
```

Name: _____

Id: _____

Email: _____

Student Name: Christopher Seagraves Student ID: _____

```

32  if COMPUTE_FEATURES
33
34      for i = 1:numel(labels)
35          label = labels{i};
36          label_images = images(label);
37          fprintf('Processing %s images...\n', label);
38
39          for j = 1:numel(images(label))
40              image = label_images(j);
41
42              [x, sift_features] = get_image_features(single(image.grayscale_image), ImageFeatureOperations.SIFT);
43              f_sift = [f_sift, sift_features];
44              image.sift_features = sift_features;
45
46              [x, dsift_features] = get_image_features(single(image.grayscale_image), ImageFeatureOperations.DSIFT);
47              f_dsift = [f_dsift, dsift_features];
48              image.dsift_features = dsift_features;
49
50              label_images(j) = image;
51          end
52
53          images(label) = label_images;
54      end
55
56      fprintf('Saving features...\n')
57      save('images.mat', 'images', '-v7.3')
58
59
243 function [keypoints, descriptors] = get_image_features(image, operation)
244
245     if strcmp(operation, ImageFeatureOperations.SIFT)
246         [keypoints, descriptors] = vl_sift(image);
247     elseif strcmp(operation, ImageFeatureOperations.DSIFT)
248         [keypoints, descriptors] = vl_dsift(image);
249     else
250         error('Invalid operation');
251     end
252
253 end

```

Save features as a cell structure, `f_sift{}`, `f_dsift{}`, for N images, submit as a `gradient_features.mat` file. Use cell structure.

Early in the homework I did save a `gradient_features.mat` file, but after converting some of my HW1 stuff from Python to MATLAB, I ended up making an `Image` class and just put the features in there and saved to `images.mat` throughout the process.

Name: _____

Id: _____

Email: _____

Student Name: Christopher Seagraves Student ID: _____

```
1  classdef Image
2
3      properties
4          path
5          label
6          rgb_image
7          name
8          hsv_image
9          flattened_hsv_image
10         grayscale_image
11         sift_features
12         hog_features
13         dsift_features
14         sift_fisher_vectors
15         dsift_fisher_vectors
16     end
17
18     methods
19
20         function obj = Image(path, label, rgb_image)
21             obj.path = path;
22             obj.label = label;
23             obj.rgb_image = rgb_image;
24
25             obj.name = strsplit(path, '/');
26             obj.name = obj.name{end};
27             obj.hsv_image = rgb2hsv(rgb_image);
28             obj.flattened_hsv_image = reshape(obj.hsv_image, [], 3);
29             obj.grayscale_image = rgb2gray(rgb_image);
30
31             obj.sift_features = [];
32             obj.hog_features = [];
33             obj.dsift_features = [];
34
35             obj.sift_fisher_vectors = {};
36             obj.dsift_fisher_vectors = {};
37         end
38     end
39 end
40
41 end
42
```

Name: _____

Id: _____

Email: _____

Student Name: Christopher Seagraves Student ID: _____

[2] Compute the PCA and GMM model for each feature space:[25 pts]

(2a, 10pts), implement the following function that computes various gradient features:

```
% f - n x d matrix containing n features from say 100 images.  
% nc - number of GMM components  
% kd - desired lower dimension of the feature  
% fv_gmm - FisherVector GMM model:  
%     fv_gmm.m - mean, fv_gmm.cov - variance, fv_gmm.p - prior  
% A - dxd PCA for dimension reduction  
function [gmm, A]=getFisherVectorModel(f, kd, nc)  
[A,s,lat]=princomp(f);  
f0 = f*A(:,1:kd); % this is the feature with desired d-dimensions  
% call vl_gmm here: gmm.mean, gmm.var, gmm.prior are kd x nc dimensions  
.....  
return;
```

Name: _____

Id: _____

Email: _____

Student Name: Christopher Seagraves Student ID: _____

```

255 function [A, sift_gmm] = train_sift_gmm(sift_train_gmm, kds, ncs, operation)
256     dbg = 0;
257
258     if dbg
259         data = 'data';
260         addpath(genpath([pwd, filesep, data]));
261         load([data, filesep, 'sift_train_gmm.mat']);
262         kds = [8 12 16 24 32];
263         ncs = [16, 24, 32, 64, 128];
264     end
265
266     tic;
267     [A, s, lat] = pca(double(sift_train_gmm));
268     t1 = toc;
269
270     for j = 1:length(kds)
271         x = double(sift_train_gmm) * A(:, 1:kds(j)); % data projected
272
273         for k = 1:length(ncs)
274             fprintf('\n t=%1.2f training gmm(%d %d)', toc, kds(j), ncs(k));
275             [sift_gmm(j, k).mean, sift_gmm(j, k).cov, sift_gmm(j, k).prior, sift_gmm(j, k).ll] = vl_gmm(x', ncs(k), 'MaxNumIterations', 200);
276             fprintf(' ll = %1.2f ', sift_gmm(j, k).ll);
277         end % for k
278     end % for j
279
280     if strcmp(operation, ImageFeatureOperations.SIFT)
281         fprintf('Saving SIFT GMM...\n');
282         sift_A = A;
283         save('sift_gmm.mat', 'sift_A', 'sift_gmm', '-v7.3');
284     elseif strcmp(operation, ImageFeatureOperations.DSIFT)
285         fprintf('Saving dense SIFT GMM...\n');
286         dsift_gmm = sift_gmm;
287         dsift_A = A;
288         save('dsift_gmm.mat', 'dsift_A', 'dsift_gmm', '-v7.3');
289     else
290         error('Invalid operation');
291     end
292
293     % if dbg
294     % figure(41); title('SIFT GMM (8, 16)'); grid on; hold on;
295     % plot3d(sift_gmm(1, 1).mean, '*r');
296     % end
297
298     return;
299 end
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

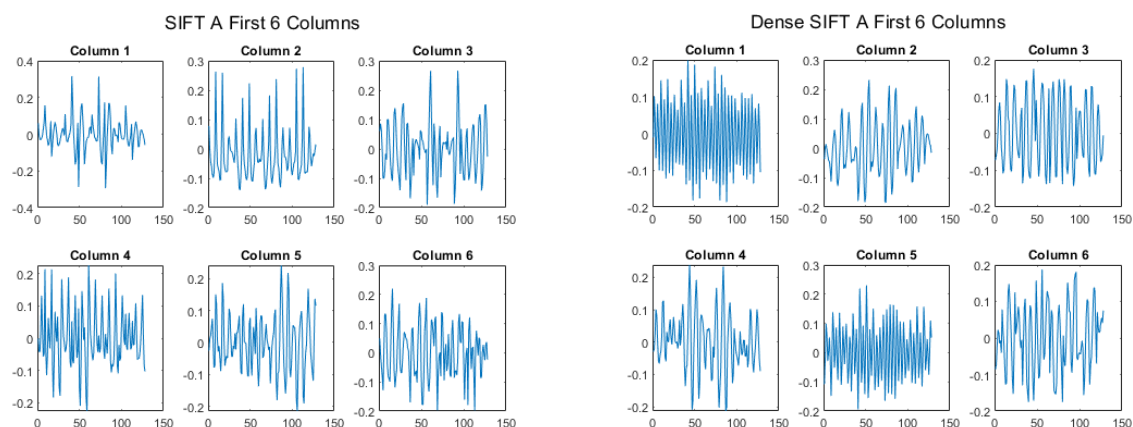
```

(2b, 5pts): Visualize the eigen values for SIFT, and Dense SIFT features here, example of SIFT is shown in lecture notes.

Name: _____

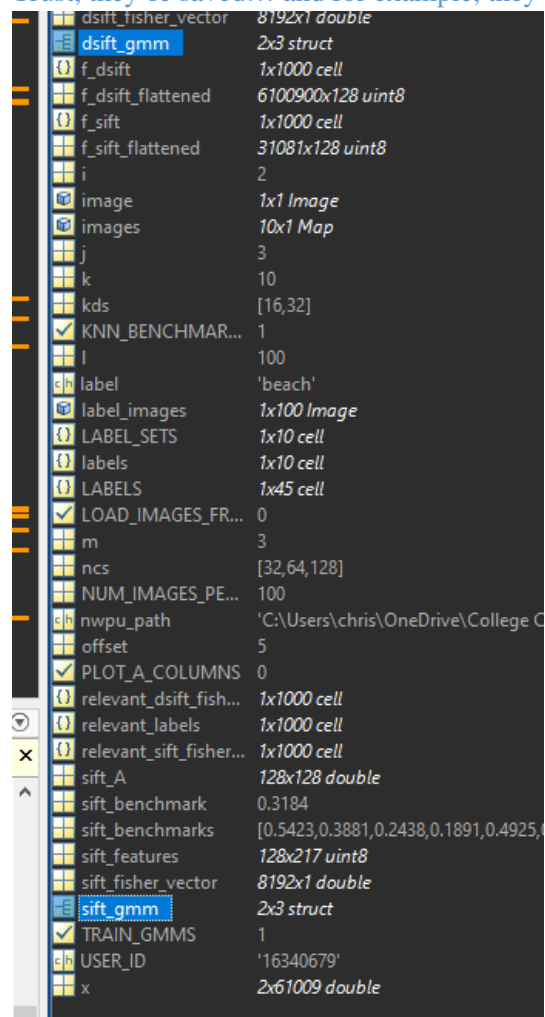
Id: _____

Email: _____

Student Name: Christopher Seagraves Student ID: _____

(2c, 10pts): Save the GMM models for SIFT, DenseSIFT features for a combination of PCA dimensions $kd=[16, 32]$, and GMM components $nc=[32, 64, 128]$. {**Hint: example code and training data provided**}

Trust, they're saved... and for example, they're a 2x3 struct to correlate with the kds and ncs 👍



Name: _____

Id: _____

Email: _____

Student Name: Christopher Seagraves Student ID: _____

[3] Compute the gradient feature aggregations and benchmark its recognition accuracy. [50pts]

(3a: 20pts), implement the FisherVector aggregation function here based on vl_feat library:

```
function [fv]=getFisherVector(f, A, gmm, kd, nc)
```

```
% f: n x d features
```

```
% A: d x d PCA matrix
```

```
% gmm: gmm.mean, gmm.var, gmm.prior, the GMM of dimension kd by nc
```

```
.....
```

```

114     if COMPUTE_FISHER_VECTORS
115
116         for i = 1:numel(labels)
117             label = labels{i};
118             label_images = images(label);
119             fprintf('Getting fisher vectors %s images...\n', label);
120
121             for j = 1:numel(images(label))
122                 image = label_images(j);
123
124                 image.sift_fisher_vectors = {};
125                 image.dsift_fisher_vectors = {};
126
127                 for k = 1:size(sift_gmm, 1)
128
129                     for m = 1:size(sift_gmm, 2)
130                         sift_fisher_vector = get_fisher_vector(image.sift_features', sift_A, sift_gmm(k, m), kds(k), ncs(m));
131                         dsift_fisher_vector = get_fisher_vector(image.dsift_features', dsift_A, dsift_gmm(k, m), kds(k), ncs(m));
132                         image.sift_fisher_vectors = [image.sift_fisher_vectors, sift_fisher_vector];
133                         image.dsift_fisher_vectors = [image.dsift_fisher_vectors, dsift_fisher_vector];
134                     end
135                 end
136             end
137
138             label_images(j) = image;
139
140         end
141
142         images(label) = label_images;
143
144     end
145
146     fprintf('Saving fisher vectors...\n')
147     save('images.mat', 'images', '-v7.3')
148 end

```

```

302 function [fisher_vector] = get_fisher_vector(features, A, gmm, kd, nc)
303     f0 = double(features) * A(:, 1:kd);
304     fisher_vector = vl_fisher(f0', gmm.mean, gmm.cov, gmm.prior);
305 end

```

(3b: 20 pts), compute the knnclassify() based recognition (leave 1 out) results and confusion map, just like in HW-1 here for a combination of features, GMM sizes, fill in the table below, and also show confusion map:

Name: _____

Id: _____

Email: _____

Student Name: Christopher Seagraves Student ID: _____*SIFT* FV aggregation recognition accuracy: $\text{sum}(\text{diag}(\text{cm}))/\text{sum}(\text{cm}(:))$

GMM Size (Kd/nc)	32	64	128
16	54%	39%	24%
32	19%	49%	32%

Dense SIFT FV aggregation recognition accuracy: $\text{sum}(\text{diag}(\text{cm}))/\text{sum}(\text{cm}(:))$

GMM Size (Kd/nc)	32	64	128
16	66%	62%	64%
32	66%	69%	67%

HoG FV aggregation recognition accuracy: $\text{sum}(\text{diag}(\text{cm}))/\text{sum}(\text{cm}(:))$

GMM Size (Kd/nc)	32	64	128
16			
32			

Name: _____

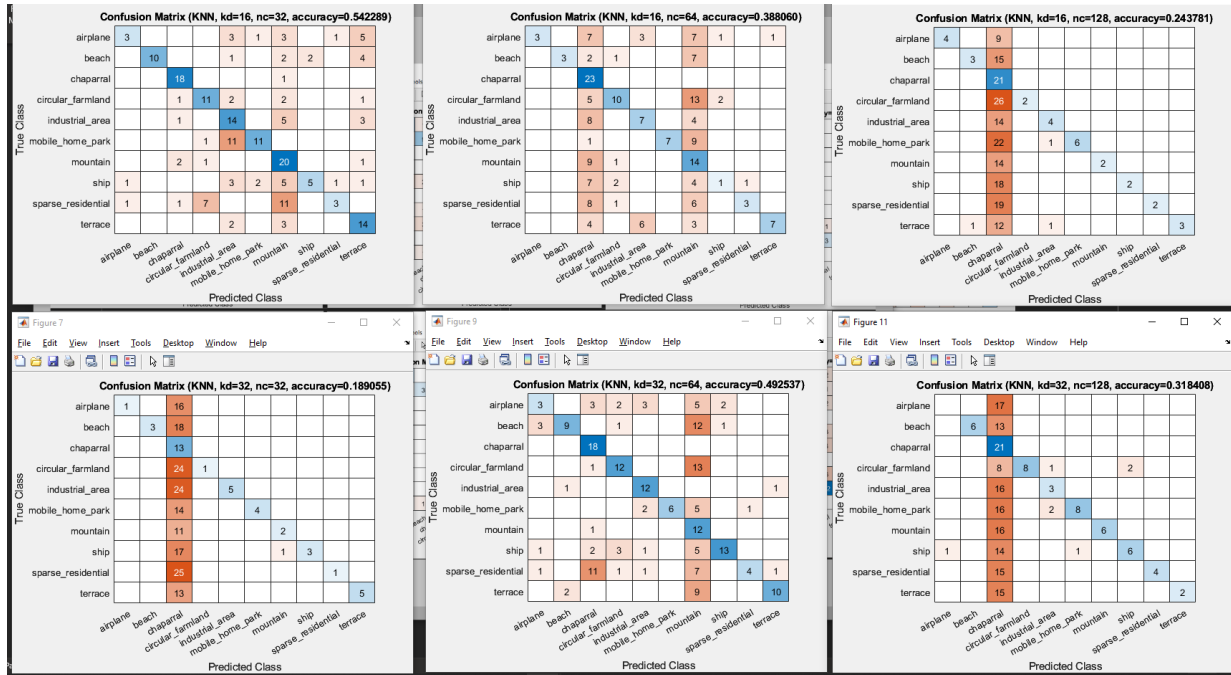
Id: _____

Email: _____

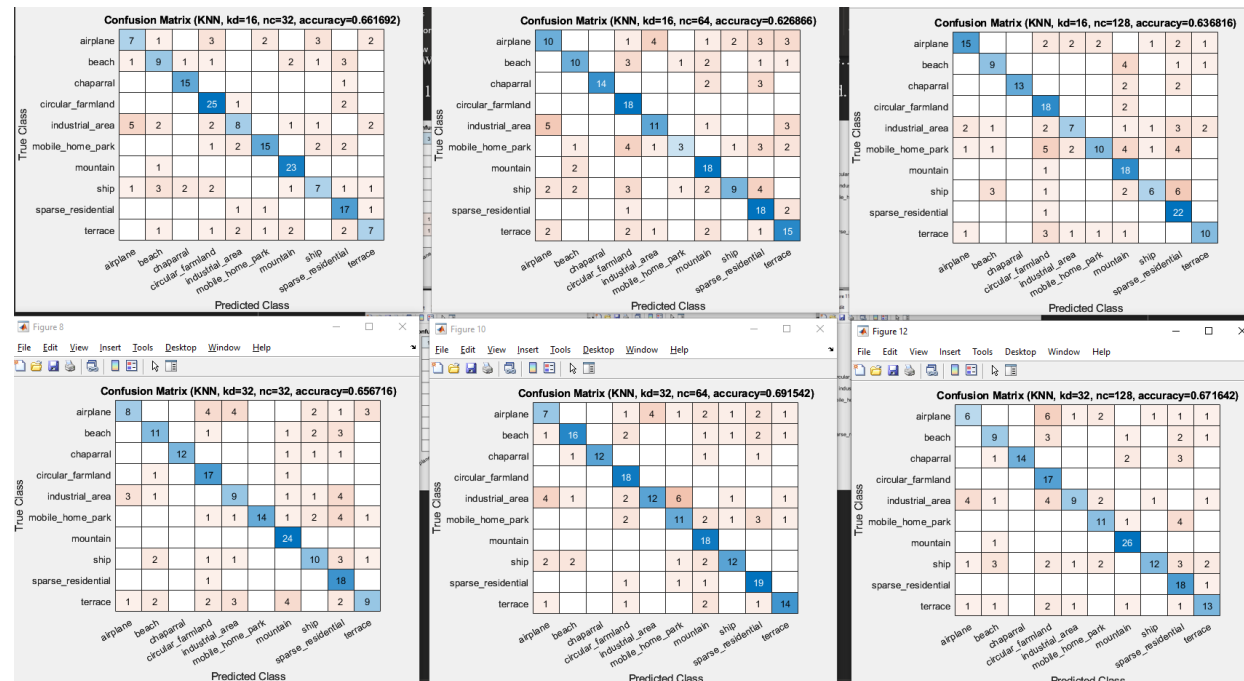
Student Name: Christopher Seagraves Student ID: _____

(3c: 10 pts), Plot the confusion map for 6 combinations for each feature, one example for SIFT, kd=16, nc=64 is shown below: {Hint: confusionmat() }

Not sure what happened with predicting chaparral overload on 3 of these... This was 100 images per label, and using 80% train-test split, but yeah, odd.



For comparison, here's the dense sift



Name: _____

Id: _____

Email: _____

Student Name: Christopher Seagraves Student ID: _____

[4] **Extra Credit**: Compute the gradient feature VLAD aggregations and benchmark its recognition accuracy. [30pts]

(4a: 15pts), implement the VLAD aggregation function here based on vl_feat library:

```
function [fv]=getVLAD(f, gmm, kd, nc)
% use the GMM means as kmeans centroids
.....
```

(4b: 15pts), compute the knnclassify() based recognition (leave 1 out) results and confusion map, just like in HW-1 here for a combination of features, GMM sizes, fill in the table below, and also show confusion map:

SIFT VLAD aggregation recognition accuracy: $\text{sum}(\text{diag}(\text{cm}))/\text{sum}(\text{cm}(:))$

GMM Size (Kd/nc)	32	64	128
16			
32			

Dense SIFT VLAD aggregation recognition accuracy: $\text{sum}(\text{diag}(\text{cm}))/\text{sum}(\text{cm}(:))$

GMM Size (Kd/nc)	32	64	128
16			
32			

HoG VLAD aggregation recognition accuracy: $\text{sum}(\text{diag}(\text{cm}))/\text{sum}(\text{cm}(:))$

GMM Size (Kd/nc)	32	64	128
16			
32			

Name: _____ Id: _____ Email: _____

Student Name: Christopher Seagraves Student ID: _____

Name: _____

Id: _____

Email: _____