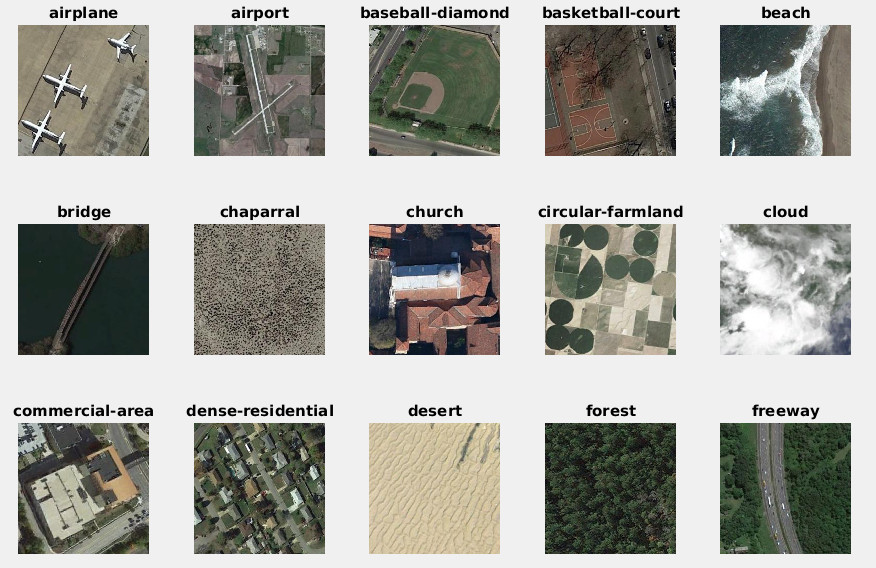
Data set: the NWPU aerial data set contains approximately 45 categories of 700 images for each category in 256x256 RGB format. The data set can be downloaded from:

<https://umkc.box.com/s/fxvzh5qq2tiob6eklfxfwn89kg3e1io1>



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)  ...... |

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.

Text

Description automatically generated

1. 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; |

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

Chart

Description automatically generated Chart

Description automatically generated

(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 👍Text

Description automatically generated

1. 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  ...... |

(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:

*SIFT* FV aggregation recognition accuracy: sum(diag(cm))/sum(cm(:))

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

*Dense SIFT* FV aggregation recognition accuracy: sum(diag(cm))/sum(cm(:))

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

*HoG* FV aggregation recognition accuracy: sum(diag(cm))/sum(cm(:))

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

(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.  
  
Chart

Description automatically generated

For comparison, here’s the dense sift

Chart

Description automatically generated

1. 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: sum(diag(cm))/sum(cm(:))

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

Dense SIFT VLAD aggregation recognition accuracy: sum(diag(cm))/sum(cm(:))

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

HoG VLAD aggregation recognition accuracy: sum(diag(cm))/sum(cm(:))

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