Programming Task 1

**Task general description**

In this task you will

* Implement an object recognition pipe
* Train it on a subset of the Caletch101 dataset
  + Including tuning of hyper parameters
* Test your performance on another subset
* Report your results

We will use publically available implementations for most of the algorithms, and the main programming effort is composing the code into a functional pipe and experimenting with it to get reasonable results.

The exercise goals: experience building a full pipe including representation. Get acquainted with classic representation techniques.

**Data set**

We will work with Caltech 101 dataset

* The data contains 101 classes, with 31-800 images each.
* Data homepage: <http://www.vision.caltech.edu/Image_Datasets/Caltech101/>
* We will train on 20 images per class , test on 20 others (unless there are less than 40 images for the class – in which case we will have less images for test)
* When loading the data:
  + The data is in 102 folders in the top folder 101\_ObjectCategories\, one folder per class
    - Note that the folder BACKGROUND\_Google\ is not a class. Erase it.
  + To load it write a function which uses the function dir() on the top folder 101\_ObjectCategories\, then loops through the subfolders and through the images in each subfolder, and gets all the images into a single 3D Data array of size (with a side length parameter, and the number of images) and a single Labels vector of size
    - Load images using imread()
    - Get them to gray scale (if required) using rgb2gray()
    - Resize them to using imresize() and stack them into the data array
* Data split
  + Debug and tune your pipe on the first 10 classes (fold 1)
  + When the final configuration is debugged and stable, run the algorithm with the best hyper parameter configuration found on classes 11-20 (fold 2).

**Optional pipes to implement**

You may choose to implement one of the following pipes:

* SVM + HOG pipe
* SVM + KMeans + SIFT pipe
* NBNN pipe (using SIFTs and a KDTree for nearest neighbor computation)

In pipes using SVM, you should use a linear kernel as the default.

Groups of 2 or 3 people are required to try at least one more kernel (polynomial, RBF) and report the best results.

Groups of 3 people can only choose the second or third pipe (not SVM+HOG).

**Required results**

You will submit the code and a short report of the results.

Code:

* The code is required to run directly, without modifications, on my machine.
* This machine which will have the libraries we use on the matlab path and the dataset in the folder '.\101\_ObjectCategories\' (i.e. 101\_ObjectCategories\ will be a sub-folder in the folder in which the code runs).
* The code will be generic and will be able to run on any subset of 10 classes (from the 101 classes) by changing a single variable called ClassIndices which will be defined in the main file, in one of the first 5 lines. ClassIndices will be a vector of 1\*10 containing the indices of classes on which the experiment will run. For example:

ClassIndices=[ 5 6 10 60 65 67 81 83 86 90 ]

The code you submit should set classIndices to fold 2 classes, i.e. ClassIndices =11:20. It should use fixed hyper parameters (the best configuration found). However, I will test your code on an arbitrary set of classes. You should verify that it is able to run on such an arbitrary set.

* The code should print to the matlab prompt as output the test error result (in a clear sentence) and the confusion matrix.

Report: The report should include

* Hyper parameter tuning graphs: The pipe includes hyper parameters (like – the image size, – the number of codewords for Kmeans, – the SVM tradeoff parameter, etc..). Such parameters should be tuned to get good accuracy. I list below the most important hyper parameters for each module, but you may consider tuning other parameters.

For at least two parameters, systematic tuning should be done, in which:

* + The training set is split to two subsets, termed 'training' and 'validation'
  + Several values of the parameter are tested, by training on the training set and estimate the error on the validation set.
  + The parameter value giving the lowest validation error is chosen.

The report should include graphs showing the validation error as a function of hyper parameter value for the hyper parameters that were systematically tuned. The chosen value should be stated.

* Test results:
  + The error rate obtained over the test set
  + A confusion matrix (use confusionmat())
* Error visualization: For each class, show images of the two largest errors on images of the class (i.e. images from the class which were miss-classified). The error images should be displayed only if they exist (i.e. if there are at least two errors from the class). If there is only one error from the class –show it, and if there are none – just state that there were no errors for this class. By *largest error* I mean the images which got the lowest margin, following this definition:
  + - Class\_score(: For SVM-based system Class\_score( is defined as the SVM score of the classifier. For NBNN, Class\_score( is the image-to-class-i similarity.
    - The margin for an example of class i is . This is the difference between the score of the correct class score and the maximal score of incorrect classes. Larger values indicate good classification. A value below 0 is an error.

**Public code we use:**

The VLFeat library – download from <http://www.vlfeat.org/index.html>. Extract and Read the README.md regarding installation and documentation. If you do not have a zip utility for .zip extraction, download 7zip from <http://www.7-zip.org/>

The SVM from East Anglia university- download from <http://theoval.cmp.uea.ac.uk/svm/toolbox/>, or use the code version I put in the site (see below) in 'SVM' section

**Software modules we build and use**

* Sift extraction
  + Use the function vl\_dsift() from the VLFeat library (documented in <http://www.vlfeat.org/overview/dsift.html>) for dense SIFT extraction
  + Important parameters in dense SIFT extraction: we extract SIFTs around every point in a dense 2D grid of points over the image. The grid stride (i.e. the distance between points) is likely to be important. Also, at each point SIFTS may be extracted at multiple scales. I suggest using the scales used at <http://vision.cse.psu.edu/seminars/talks/2009/random_tff/bosch07a.pdf>, section 5.’appearance’ as a good starting point.
* KMeans
  + Use vl\_kmeans() from the VLFeat library (documented in <http://www.vlfeat.org/matlab/vl_kmeans.html>)
  + Important parameters: when training the dictionary, it is enough to extract some SIFTS from some images (that is: you do not have to use all the SIFTS from all the images). For example, using a subset of 100 SIFTS from 1000 images (100,000 SIFTs overall) should be enough for good clustering). The most Important Parameter is K of course. Several hundred is a good area to look in.
* HOG
  + Use vl\_hog()from the VLFeat library (documented in : <http://www.vlfeat.org/overview/hog.html> )
  + Important Parameters: spatial cell size, number of orientation bins
* KDTree
  + KDTree is a data structure for fast nearest neighbor finding, supported in the VLFeat library. Use createns() for tree creation from a data set and KDTreeSearcher() for finding the nearest neighbor of a query example. Type ' help KDTreeSearcher' to see a simple usage example
* SVM
  + We will use the SVM from the university of East Anglia
  + Option 1:
    - Download the SVM from <http://theoval.cmp.uea.ac.uk/svm/toolbox/>
    - Run compilemex.m at the matlab command promt to compile.
    - Run the demo demo.m and look at it to understand how the SVM is operated
  + Option 2:
    - Download from the module a slightly modified version in which the compilation was already made, and the demo has a KernelType variable allowing you to switch the kernel type used. Try running the demo with KernelType=1,2,3,4,5 to see the borders created by SVM with different kernels
  + Important parameters: C – the SVM tradeoff parameters, when using an RBF kernel, – the polynomial degree when using a polynomial kernel
  + Using SVM for multiple classes:
    - Write a function MClassSVM\_Train which loops through the M classes and trains M binary classifiers in one-versus-all method
      * That is: for classifier , the examples of class get the label +1, and the rest of the classes get label.
      * The function returns an MClassSVM structure containing all the SVM models as fields
    - Write an MClassSVM\_Predict() function which accepts a set of examples to test as a matrix, and an MClassSVM structure returned by MClassSVM\_Train. The function predicts by
      * Apply the SVMs to the data and put the predictions in an class score matrix.
      * Compute the predicted class (an vector) by taking the argmax over the class score matrix columns (the highest score in the row determines the winning class)
      * Return the predicted class and the class score matrix