CS364/AM792 : Computer Vision 1 October 2019

Assignment 5: Image Classification with Feature Vectors

Please prepare a single, condensed and neatly edited document for submission. For each problem include a short description of what you did, results, and a brief discussion/interpretation. It is not necessary to include any code in your document, unless a snippet helps to explain your method. Upload a zip-file containing your document and code to SUNLearn (under "Assignment 5") **before 17:00** on 14 October. Also hand in a printed copy of the document, at the latest during class on 15 October.

You are free to use any programming language. I recommend Matlab or Python. If you are asked to implement a specific technique, the idea is that you do so from scratch; do not simply use a function from an image processing or computer vision library. Collaboration is restricted to the exchange of a few ideas, and no form of plagiarism will be tolerated. All code, results, and write-up that you submit must be your own work.

1. Your task here is to implement the basic SVD approach to face recognition, from Lecture 23. You are given a dataset[1] split into training data (`faces_training.zip`) and test data (`faces_test.zip`).

(a) Use all the images in the training set to determine $\underline{a}$ and U_α. Plot the singular values of X in order to pick a value for α (but feel free to experiment with different choices later on). Display the average vector $\underline{a}$ and the first few eigenfaces (columns of U_α), as images. First reshape the vector you want to display back to a $p \times q$ matrix, and remember to scale the values of an eigenface to $[0, 255]$.

(b) Now use $\underline{a}$ and U_α to encode all the **test** images to eigenface representations (called $\underline{y}$ in the lecture slides). Reconstruct the test images from their encodings, and display a few next to the originals to get a sense of how successful your dimensionality reduction is at capturing image information.

(c) For each image in the test set, calculate its eigenface representation, find the nearest neighbour over the training set, and classify accordingly. Show examples of test image and image chosen by the system as the closest match, and report on the overall accuracy of your system (what percentage of test images are classified correctly). Note that there are two test images of every individual; file names indicate correct labels.

2. You are given a dataset[2] of a few thousand images, each belonging to one of 9 scene classes, split into training data and test data.

(a) Build a vocabulary of visual words from the training data, using SIFT features[3] over the entire training set and k-means clustering[4] all of the 128D descriptor vectors. Every cluster corresponds to a visual word, and we represent an image by a normalized histogram of these words. If our vocabulary has 50 words, an image is thus converted to a 50D feature vector.

(b) Now train linear, one-vs-one support vector machines for classification. That is, train 36 SVMs that each separates a pair of classes[5]. For a test image, we let the SVMs vote on a class label and use prediction scores to break ties. Report on the overall test accuracy[6], show a handful of test images correctly classified and incorrectly classified, give a confusion matrix, and discuss your findings.

Hand in: 14 October 2019

[1] from: `http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`

[2] from: Lazebnik et al., *Beyond bags of features: spatial pyramid matching for recognizing natural scene categories*, CVPR 2006.

[3] You are welcome to use the SIFT detections and descriptor vectors in this assignment's accompanying material.

[4] You may use an existing function for k-means clustering. Note that the set of points to cluster is quite large; I would suggest casting the data to single precision and looking for a computationally and memory efficient k-means implementation.

[5] You may train the SVMs using existing software. Make sure you have control over the regularization parameter, as you'll have to experiment a bit to find a suitable value for it.

[6] You should aim for a test accuracy of around 60% which, considering the challenging problem, is not bad at all!