SVM-KNN based Geometric Blur Descriptor for Object Class Recognition on Caltech 101*

Abhijit Bendale
University of Colorado -Colorado Springs
abendale@uccs.edu

*(some parts of README and this report are redundant)
Description of the code:

1. Creation of training and testing set
2. Extraction of geometric blur descriptor for each image in training set
3. Classifying each test image based on SVM-KNN
4. Error Analysis
5. My Comments on the code

Creation of training and testing set:

From each category in Caltech 101 dataset, 15 training images are extracted and remaining images are left as test images. This does make results skewed (since each for category has different number of images).

Extraction of geometric blur descriptor for each image in training set

All the processing takes place on grayscale images.

For the purpose of extraction of geometric blur descriptor, each image is read and keypoints are detected based on David Lowe's SIFT keypoint detector. Other keypoint detection methods could also have been used but SIFT was chosen for easy availability of the code. The *.key file is transformed to *.xy file which contains only (x,y) coordinates of the keypoint. Each image is separated into sparse channels. Here, the humble sobel operator is used. Better boundary detection techniques would have given better results. Later, each image channel is blurred with varying levels of blur. In our experiments we have used 4 channels and 4 blur levels.

Around each keypoint, circles at varying radii are sampled with 24 points per circle. Intensity values at varying radii and corresponding blur from each sparse channel (sobel filtered image) are extracted and stored in the descriptor. Since there are 4 radii, 4 channels and 24 sample points per radii, there are 384 (“filtered”)intensity values for descriptor for each keypoint. We also store the (x,y) location of the keypoint in the descriptor. Thus our geometric blur descriptor becomes 386 values list of numbers in python. All such descriptors from the image are stored in *.decp file. Thus, we considered creation of *.decp files for all training images as a pre-processing step.

The radii considered are [5,15,25,35]. We ignore the keypoints near the boundary (i.e. within 35 pixels of the image boundary). We could have done without ignoring, but this would have led to uneven descriptors.

Classifying each test image based on SVM-KNN

For each test image, we first compute the geometric blur descriptor by the same method as mentioned in earlier section. Once this is done, we compute the following distance (as mentioned in section 4.4 Algorithm B of SVM-KNN paper) from the test image to all the training image

Distance Measure:

$$D^B(I_L \to I_R) = \frac{1}{m}\sum_{i=1}^{m} \min_{j=1..n} \left[\|F_i^L - F_j^R\|^2 + \frac{\lambda}{r_0}\|r_i^L - r_j^R\| \right]$$

$$D^B(I_L, I_R) = D^B(I_L \to I_R) + D^B(I_R \to I_L)$$

Once this distance is computed from the test image to all the training images, 30 nearest neighbors are selected. The descriptors from these 30 training images are used to train a multi-Class SVM (used from PyML). The descriptor list of the test image is classified based on the classifier created by 30 nearest neighbors. The classification success rate and classified category by SVM is noted and is used for evaluation purpose.

Error Analysis:

Mean recognition rate is obtained from the ground truth and the classifier assigned category in the above step. The mean recognition rate per category for our experiments was :

My Comments on the code:

1. The total time taken for finishing this assignment was about 10 days of good work.
2. If this was a “real” project, I would have used NumPy instead of straight use of lists for data representation which would have given more speed up
3. There are some ugly things in the code, like hard-coding on sobel operator, blur etc. This could have easily been avoided.
4. The use of OO was very tempting but I decided to stick to plain “scripting” approach of Python
5. In my experiments I have carried out only one iteration. Some more iterations would have been better
6. If this code was going out for OpenSource or something similar, more extensive debugging would have been advisable. I have tried to get rid of conceptual bugs. But there are many places where “quick hacks” are clearly evident. The only person to blame for this is me since I picked up Python specifically for this project so I clearly lacked the expertise.
7. I have gone with use of the default kernel of PyML for multi class classifer.
8. There are still some gray areas in terms of SVM training. If this was a real project, I would have definitely taken some help from the authors/mentor to clarify certain things.