Textons and classifiers laboratory report

Jos Antonio Valero Medina
Universidad Distrital "Francisco Jos de Caldas"
Carrera 7 No. 40B - 53, Bogot D.C. - Repblica de Colombia
jvalero@udistrital.edu.co

Abstract

Here it is reported and discussed a code creation to represent images using textons in order to train and test nearest neighbor and random forest classifiers. Whole process was run on database with a training dataset of 750 images for 25 texture categories and a test dataset of 250 images for the same 25 texture categories. First, the training dataset was filtered using a filter bank made of 32 oriented filters and two center surround ones. From each filter bank processed image was taken only 1 pixel for each 256 of them in order to be able to manage a possible memory lack in k-means clustering. Next, 200 textons were produced using the k-means grouping. Then, each image, in both training and test datasets, had its filter bank processed pixels linked to the nearest texton, i.e. *cluster centroid location. Finally, the texton normalized frequency distribution (histogram) on each image was calculated and used as a descriptor for it in the database. So the training and test datasets was produced and used in the classification stage to train and predict the test image class,* i.e. *texture category, using a nearest neighbor and a random forest classifiers. The classification evaluations showed that the best classifier was the random forest one.*

1. Introduction

Here it is reported and discussed a code creation to represent images using textons in order to train and test nearest neighbor and random forest classifiers. Whole process was run on database with a training dataset of 750 images for 25 texture categories and a test dataset of 250 images for the same 25 texture categories.

First, each training dataset image was filtered using a filter bank made of 32 oriented filters and two center surround ones. The procedure followed for it in the included example script, *i.e*. to mosaic the images to form only one, was not followed here as the filter bank process aborted with a lack of memory error. Instead, each training image was individually filter bank processed. From each so produced image was taken only 1 of each 256 pixels in order to be able to manage a possible memory lack in k-means clustering. So sampled images, with 30 X 40 pixels, were stacked all together to form an image of 30 x 25 rows and 40 x 30 columns as there were 25 texture categories and each texture category had 30 variations.

Next, 200 textons, *i.e*. k-means clusters each one represented by the respective cluster centroid location, were produced. Then, each image, in both training and test datasets, had its filter bank processed pixels linked to the nearest texton, *i.e*. nearest cluster centroid location, using *assignTextons* provided function. Finally, a texton normalized frequency distribution (histogram) on each image was calculated and used as its descriptor in the database.

After having assigned a descriptor to each database image and produced training and test datasets, the classification stage was conducted. Two classifiers, a nearest neighbor and a random forest, were trained with the training dataset and used to predict the texture category of each test image in the testing dataset. *chi2* function was implemented as nearness measure for the nearest neighbor classifier and a bag with 50 decision trees was used for the random forest one. The classification evaluation stage was done using a confusion matrix from which the overall accuracy and texture category accuracy vector measurements were produced. The classification evaluation outcome showed that the best classifier was the random forest one as this nearly doubled overall accuracy obtained by nearest neighbor.

2. Data and methods

2.1. Functions in *lib* folder

fbCreate creates a filter bank at different scales containing even and odd symmetric filters in several orientations (except center surround filter as the respective creation code line is included but commented out). Each orientation is calculated as a π fraction depending on the wanted number of orientation (8 by default) and the scale (a given maxnification scaling rate). An elongation ratio, which should be greater than 1, determines how elongated, along the orien-

tation, the border is. For each orientation (and scale) two filters are built optionally doing for one of them a Hilbert transform in *y* direction but using for both a second *y* derivative. oeFilter function is used to create each filter in the bank. oeFilter produces a Gaussian square odd sized filter with a given orientation. The filter is built seizing the Gaussian separability feature applying *L1-norm* at the end.

fbRun applies a filter bank to an image reflecting its boundary border in agreement with the greater filter size found in the filter bank. If a filter size is less than 50 this makes a convolution otherwise this calculates a 2-D inverse fast Fourier transform of the product between 2-D fast Fourier transform on the image and filter. This function returns a cell with a filtered image for each filter in the filter bank organized as the filter bank by orientation and scale.

computeTextons produces, based on a k-means grouping, a wanted number of *textons* classes from a filtered image set. Each filtered image corresponds to a dimension of the new feature space with as many dimension as the number of images in the filtered image set. The k-means grouping method is applied on the new feature space for creating as many *textons* as the number of clusters so generated, *i.e*. each cluster is a *texton*. Finally, this function returns an image whose pixel values correspond with the *texton* class, *i.e*. cluster code as generated by k-means, as a *Map* and the cluster centroid locations for each cluster as the descriptor for each *texton* class in the *textons* matrix.

assignTextons assigns a texton to a filtered image set based on a *textons* matrix, *i.e*. a matrix whose rows corresponds with a texton and the respective column value set with the cluster centroid location. The filtered image set must have as many filtered images as the second dimension size of *texton* matrix. The filtered image set is put in a feature space whose elements are assigned to the nearest texton, *i.e*. nearest to the respective cluster centroid location taking the squared distance as the nearness measurement. Finally, this function returns an image whose pixel values correspond with the *texton* class (*i.e*. cluster code as generated by k-means) as a *Map*.

2.2. Database

The database downloaded from `http://157.253.63.7/textures.tar.gz` is made of training and test texture gray scale images. This database provides 25 different categories of texture each one with 30 variations in the training dataset and 10 in the test one. The variations can be in orientation, and scale. The database also includes a dictionary which provides a semantic value for each category.

2.3. Methods

The procedure followed in the included example script, *i.e*. to mosaic the images to form only one, was not followed here as the filter bank and k-means grouping processes aborted with a lack of memory error. In the following implemented custom matlab functions are discussed.

First activity was to adjust the *fbCreate* algorithm since it had included center surround filter creation but as comment. The respective line code was *Uncomment* in matlab and the csFilter invoked was created as that available in `https://github.com/jflalonde/colorRealism/blob/master/3rd_party/segmentationBerkeley/lib/matlab/csFilter.m`. It was necessary to do that because some textures were center surround ones.

DatabaseDescTiTj function was implemented to produce the texton descriptors of the database. First, each training dataset image was individually filtered using a filter bank made of 32 oriented filters and two center surround ones. Next, from each so produced image was taken only 1 of each 256 pixels in order to be able to manage a possible memory lack in k-means clustering. So sampled images, with 30 X 40 pixels, were stacked all together to form an image of 30 x 25 rows and 40 x 30 columns as there were 25 texture categories and each texture category had 30 variations. Finally, 200 textons, *i.e*. k-means clusters each one represented by the respective cluster centroid location, were produced using *computeTextons* provided function. *DatabaseDescTiTj* is invoked for being executed between two texture categories from the starting one (tIni) and the ending one (tEnd). Both the filter bank processed pixel sampled set and the produced texton set are saved on disk.

assignDesc function performs the following process for each image in the dataset. First it runs the filter bank, using *fbRun* provided function, and assign the texton to each pixel, using *assignTextons* provided function. On the so marked image *assignDesc* calculates a texton normalized frequency distribution (histogram) and used it as the image descriptor and adds the respective vector to the descriptor dataset. The final image descriptors can optionally have a predictor column.

nearestNeighborClassifier function first creates a nearest neighbor classifier based on the training dataset provided and then uses it to predict the class of each descriptor in the test dataset provided using chi square distance as a similarity measure. The last column in the provided training dataset will be used as the predictor one. As many columns as there are before the last column in the training dataset are used from the provided testing dataset.

randomForestClassifier function first creates and trains a random forest classifier model based on the training dataset provided and then uses it to predict the class of each descriptor in the test dataset provided. The model bags 50 decision trees. As many columns as there are before the last column in the training dataset are used from the provided testing dataset.

chi2 function implements chi square similarity metric but as required by fitcknn funcion used in nearest neigh-

bor classifier as is shown in the following *matlab* function snip:

```
                              ...

d2 = zeros(size(h2,1),1);

for j = 1:size(h2,1)
 d2(j) = ...
   sum(((h1 - h2(j,:)).^2)./(h1 + h2(j,:)))
end

                          ...
```

Whole applied procedure is included in the *Classifier* matlab script. First, the filter bank is created using the default options to produce 34 filters. After, texton descriptors are created in a sampled way as described before using 200 textons as with them was possible to get a good overall accuracy for the random forest classifier as shown below. Then, the training and test image descriptors were created with a predictor column for both dataset. This predictor is used in the case of the test dataset only during classification evaluation stage for producing the confusions matrices. After having assigned a descriptor to each database image and produced training and test datasets, the classification stage was conducted. Two classifiers, a nearest neighbor and a random forest, were trained with the training dataset and used to predict the texture category of each test image in the testing dataset. *chi2* function was implemented as nearness measure for the nearest neighbor classifier and a bag with 50 decision trees was used for the random forest one. The classification evaluation stage was done using a confusion matrix (as shown in the following *matlab* function snip) from which the overall accuracy and texture category accuracy vector measurements were produced. The classification evaluation outcome showed that the best classifier was the random forest one as this nearly doubled overall accuracy obtained by nearest neighbor.

```
                                  ...

%% Evaluate nearest neighbor classification
    % Overall accuracy
confNearNeig = ...
  confusionmat(testData(:,201), ...
  classifNearNeig);
TPNN = sum(diag(confNearNeig));
overallAccuracyNN = ...
  TPNN / sum(sum(confNearNeig));

    % Texture category accuracy
textureCategAccNN = ...
  diag(confNearNeig)./sum(confNearNeig,2);

%% Evaluate random forest classification
    % Overall accuracy
confRanFor = ...
  confusionmat(testData(:,201), ...
  cellfun(@str2num,classifRanFor));
TPRF = sum (diag(confRanFor));
overallAccuracyRF = TPRF/sum(sum (confRanFor));

    % Texture category accuracy
textureCategAccRF =
  diag(confRanFor)./sum(confRanFor,2);

              ...
```

3. Results

The outcome of applying the filter bank on each training image can be seen on the shown in Figure 1 which shows an example. On the figure, beside the original texture image, the assigned texton image and its histogram descriptor are also shown. For every category the filer bank allowed a good discrimination in agreement with the specific texture but the more homogeneous is the texture pattern, the Textones best fit as less orientations there are, and better discrimination is obtained. In the case of plaid texture the final histogram descriptor allows to discriminate the most at its maximum values are more far between them. Now the important thing is that the more different the histograms distributions are, better differentiated will be the respective texture images and it will be easier for a classifier to learn that differences.

A classification outcome sample of 25 test texture images is shown in Table 1. The better random forest classifier overall accuracy (last column) is evident in respect of nearest neighbor. Table 2 shows the confusion matrix for the nearest neighbor classification in which any of wood1, wood2, wood3, water,glass2, wallpaper and knit (columns 4,5,6,7,17,21 and 23 respectively) texture images could be classified. This fact can be seen in the accuracy of texture

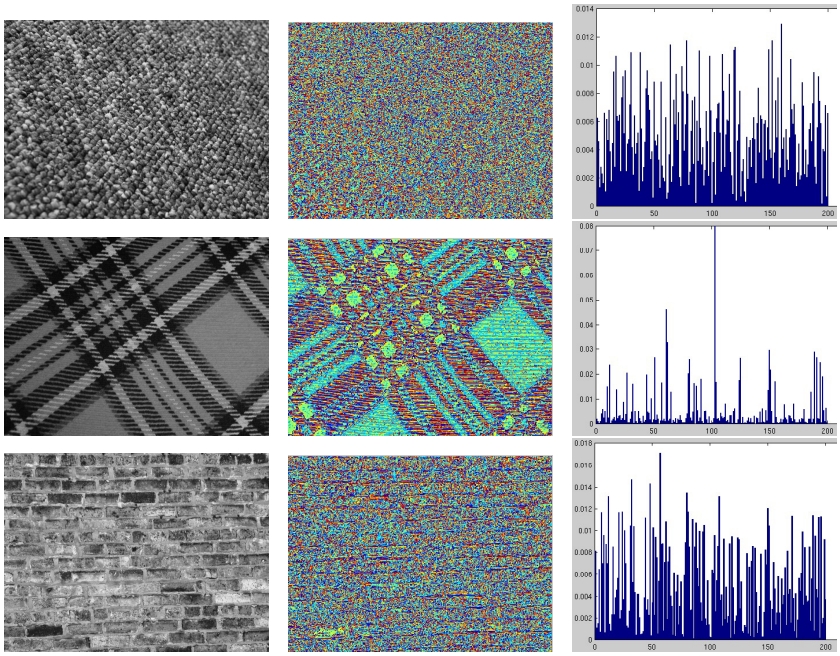

Figure 1: Texture categories for carpet (top), plaid (middle) and brick(bottom). Each original texture image (left) is shown with their texton assignment (center) and histogram descriptor (right).

9	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	4	0	0	0	0	0	3	0	2	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	9	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	0	0	0	0	0	5	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	5	0	0	0	0	0	3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	1
0	0	0	0	0	0	0	8	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	8	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	3	4	0	2	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1	0	8	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	8	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	2	0	0	0	0	0	5	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0	0	0	2	0	5	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	1	0	0	7	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	2	0	1	0	0	0	2	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	1	0	0	0	0	0	0
0	3	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	7	0	0	0	0	0
0	0	0	0	0	0	0	0	8	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	7	0	0	0
5	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1
2	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	1	0	0	0	0	0	0	3	0
2	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	6

Table 2: Nearest neighbor classification confusion matrix. Rows are the ground truth and columns the prediction.

Ground truth code	Near.Neig.	Rand.For.
1	1	1
2	2	2
3	3	3
4	12	4
5	3	5
6	3	6
7	22	22
8	8	8
9	9	9
10	9	15
11	11	11
12	8	12
13	13	13
14	8	14
15	15	15
16	16	16
17	1	19
18	18	18
19	19	19
20	20	20
21	20	10
22	12	22
23	25	23
24	14	24
25	25	25

Table 1: Sampled predictions outcomes by each Classifier compared to the ground truth.

Ground truth code	Near.Neig.	Rand.For.
1	0.9	1.0
2	0.4	0.5
3	0.9	0.9
4	0	1.0
5	0	1.0
6	0	0.9
7	0	0.7
8	0.8	0.8
9	0.8	0.9
10	0.4	0.8
11	0.8	1.0
12	0.8	0.8
13	0.1	1.0
14	0.5	0.7
15	0.7	1.0
16	1.0	0.9
17	0	0.8
18	0.9	1.0
19	0.4	0.5
20	0.7	0.8
21	0	0.8
22	0.7	0.7
23	0	0.8
24	0.3	1.0
25	0.6	0.7

Table 3: Accuracy by texture catecory

category for both classifiers shown in Table 3. The overall accuracy for the nearest neighbor classification was 0.468 while for the random forest classification was 0.84.

4. Discussion

After evaluating the classification outcomes it is clear that the random forest classifier works best nearly almost doubling the nearest neighbor classification overall accuracy. The training time for the nearest neighbor classifier was virtually zero since only had to create the classifier with the training dataset and the training time for the random forest classifier was of some few seconds. It took a really short time to apply both kinds of classifiers. The most delayed stage is the texture image descriptor production as it is k-means dependent.

Table 3 shows that for the nearest neighbor classifier wood1, wood2, wood3, water,glass2, wallpaper and knit (codes 4,5,6,7,17,21 and 23 respectively) texture categories cause the most confusion as were not possible to classify any of that test texture images. For the random forest classifier carpet2 texture category cause the most confusion.

In order to avoid a lack of memory condition during the k-means clustering was necessary to sample each image until taking in account 1 of each 16 rows and 1 of each 16 rows that means that for each sampled pixel 256 were no considered in the processing losing too much information for the image descriptor production stage.

The texture image database provides a very small number of texture categories, so this number should be incremented in order to be able to evaluate more general algorithms.

The filter bank stage was conducted using the default values, it is possible to make a better choice for a specific filters trying to change and test other parameters, *e.g*. the elongation factor on y axis or the scale magnification factor.

5. Improvements

It is necessary to research in an alternative grouping method in order to be less vulnerable to lack of memory. Matlab could not be as suitable for this specific topic as it is needed in describing a database based on the texton histogram.