Lab 05 - Textons

Jhony A. Meja
Universidad de los Andes
Biomedical Engineering Department
ja.mejia12@uniandes.edu.co

Abstract

A multi-category problem provided by Ponce group was approached with textons representation. The texton representation was modified by several parameters as size of the images (window), filtering method, number of train images per category and k-textons used for creating textons dictionary. Images classification was done by K-Nearest Neighbor (KNN) and Random Forest. Parameters modified were KNN and number of trees. The best ACA obtained was 0.5560 (1 Train, 0.1120 Test). This ACA was obtained using KNN and Random Forest with specific parameters. Better classification approach is suggested for obtaining optimal test classification.

1. Introduction

Textons are a local representation of an image's texture in a given space. Textons can be used for classifying images. The first step is to create a bank filter for representing local patterns as edges and blobs at different scales and orientations. After that, one filters the image with the bank filter. As a result, one has n-filters representation for every pixel in the image. Finally one uses k-means for clustering the textons. The centroids obtained from the clustering are now going to be the 'texton dictionary' [1].

The texton dictionary will be then used for giving labels to test images. The labels are assigned pixel to pixel. The label of a given pixel is assigned to the closest distance between it's texton representation and the previously defined centroid. The obtained image label representation is called texton map. After that, one represents the image as an histogram and trains a model for defining its category [1].

The classifier models that will be used for this paper are K-Nearest Neighbor (fitcknn in Matlab) and Random Forest (TreeBagger in Matlab). fitcknn most useful hyper-parameters are:

- 'NumNeighbors' that tells the number of neighbors that will be taken into account when deciding a pixel label.
- 'Distance' that tells which distance will be used for calculating distances between centroids and points.
- 'Cost' which is a square matrix that tells the cost of misclassification.

  TreeBagger most useful hyper-parameters are:

  - 'numTrees' which is actually a parameter that tells the number of trees that will be built in the forest.
  - 'Cost' which is a square matrix that tells the cost of misclassification.
  - 'OBBPred' that predicts the 'out of bag' informations.

  Additionally, for creating the texton dictionary one has to use k-means, which most useful hyper-parameters are:

- 'Max Iter' that tells at which iteration the centroid relocation should stop (if it keeps moving).
- 'Start' that modifies the centroid's initial location.
- 'Distance' that tells which distance will be used for calculating distances between centroids and points.
- 'Replicates' that runs the model n-times and shows its results.

2. Materials and methods

The process for classifying the images was divided in several phases. Each phase was represented by one or more Matlab functions. The first phase was loading the whole database. Later, a pre-processing phase was performed, followed by the separation of the database into train and test categories. After that, the filters were applied to the whole database. Next, the textons distribution was computed in the train images for creating a texton's dictionary. Later, textons of the whole database were assigned into different

categories depending on the dictionary previously defined. After that, a model was trained using k-nearest neighbor or random forests. Finally, the model was tested in both train and test categories.

2.1. Database Loading

The database was downloaded from Ponce's group web-page. The database was composed by 25 categories, each containing 40 images. Each category had its own folder, and all the folder's categories were contained in a folder called 'Data'. The full database was loaded in a matrix of mxnxp (where m and n corresponds to the size of each image, and p is the number of images) called 'oImgs.mat'. This process was performed using the function 'loadImgs.m'.

2.2. Pre-Processing

Two pre-processing algorithms were considered. The first one was building a Gaussian Pyramid of 7 levels for each image. This was done using the Matlab function 'GaussPyramid.m'. This function was running the function 'downsampling2.mat' for each image, whose functioning was described in previous works [4]. The pyramid was built using a Gaussian filtering with sigma of 0.5 and a window of 3x3 for each level. The output was a cell containing 7 Gaussian Pyramid levels for each image.

The second alternative of pre-processing was cropping the original images. This was done using the function 'redData.m'. This script received the original images and a window size (w). The algorithm located the center of the image and built a window of wxw around it. Three levels of w were considered: 50x50, 100x100 and 200x200. The output of the function was a matrix of (wxwxp) saved on 'redData.mat'.

2.3. Data separation

The data separation was performed depending on a parameter 'numTrain'. numTrain is the number of images that will be taken into account per category for the training set. The data separation algorithms divided the database from the first to the 'numTrain' image per category for Training, and the remaining for Test. The numTrain used for comparing different methods was 20.

The data separation was performed by 'DataSeparation.mat' if the pre-processing part was based on Gaussian Pyramid. This function received as parameters the Gaussian Pyramid, the 'k' level that wants to be studied, and 'numTrain. The output was two matrices, each containing Train and Test images.

If the pre-processing was based on chopping from the center, the function 'redDataSep.m' was used. This function's parameters were 'redData.mat' and 'numTrain'. The output was two matrices, each containing Train and Test images.

2.4. Filtering

Two filtering methods were used. The first one was based on a Filter Bank containing Gaussian second derivative and its Hilbert transform. This Filter Bank contained 32 filters (16 of 13x13 and 16 of 19x19) [Fig. 1]. This means that for this method, each image had 32 texton's representations. This Filter Bank was developed by David R. Martin in 2003. This filter bank was applied using the function 'ApplyFilters.m'. The input of the function was the Train and Test reduced images, and the output was the filtered input.

The second filtering method was based on Maximum Responses filters (MR). This filter bank contained 38 filters, all of them of 21x21. This means that for this method, each image had 38 texton's representations. This bank filter contained 2 configurations, each with 6 orientations and 3 different sizes for each orientation [Fig. 2]. This filter bank was taken from University of Oxford's web-page [3]. This filtering method was applied using the function 'ApplyFilters2.m'. The input of the function was the Train and Test reduced images, and the output was the filtered input.

2.5. Textons Computation and Assignation

The textons computation was created using the function 'TextonsComputation.mat'. This function received as parameters the filtered train matrix and the k-clusters that one wants to use for creating the texton dictionary. It is important to notice that the input was already filtered, so what one did was concatenating the filter responses previously defined on the filtering phase. We considered three options of k-clusters: 40, 20 and 10. The clusters were defined using the default k-means algorithm provided by Matlab. The output of this function was 'textMat.mat' (filtered images with their corresponding labels) and 'cMat.mat' (centroids of textons clusters).

After that, one used 'textonAssignation.mat' for obtaining the image represented with the previously defined texton dictionary. Here what one does is finding the nearest centroid (using euclidean distance) and assigning its corresponding label per pixel. The input of this function are the filtered Train and Test images, and the textons centroids ('cMat.mat'). The output was the texton Map representation for Train and Test images.

2.6. Model creation and evaluation

Two methods were used for creating models for approaching the classification problem. The first one was K-Nearest Neighbor and the second one was Random Forests. The first one was created using 'kModel.m'. This function received as inputs the Train texton map representation, the kN-neighbors that want to be taken into account for creating the model, and the previously defined 'numTrain'. Four variations were taken into account in kN-neighbors: 1 (nearest), 3, 5 and 10. The output of the function was the

KNN-model. This function was based on Matlab's function 'fitcknn'.

The second model (Random Forest) was created using 'treeModel.m'. This function's parameters were the Train texton map representation, the number of trees that want to be taken into account for creating the model, and the previously defined 'numTrain'. Four variations of the number of trees were taken into account: 5, 10, 20 and 50. The output of the function was the Random Forest model. This function was based on Matlab's function 'TreeBagger'.

Finally, the models performance was evaluated using 'evalData.m'. The inputs were the model, the textons map representation of train and test images, and 'numTrain' (previously defined). The ground-truth was created into the same function. The output of the function was train, test and global ACA. Also, the confusion matrix for each group was generated and saved as a .mat. The predictions based on the specified model were created using Matlab's 'predict' function. The confusion matrix was created using Matlab's 'confusionmat' function.

Additionally one can see the best performance in the script 'Demo.m' uploaded in the 'code' folder.

3. Results

Time was one of the method's performance metrics. An approximate of time per execution was registered for each function with its different parameters. The other method's performance metric was the ACA obtained from each model, and its corresponding confusion matrix. The balance between these two metrics should define the best method (not so slow and with an acceptable ACA).

The Database Loading ('loadImgs.m') was executed only once and its result was used for the following phases. The time was not registered but it was approximately less than 10 minutes.

The pre-processing goal was to reduce the image's resolution, in order to diminish the computational cost. The preprocessing method using Gaussian Pyramid ('GaussPyramid.m') couldn't be developed as it took to long (more than 7 hours). Other problem for using the Gaussian Pyramid approach was constant disconnections of the terminal (even if the connection was told to be kept alive). Gaussian Pyramid was thought as a possibility because it worked well for a binary classification problem (distinguishing between goats and persons). The goal was to reduce the resolution but without losing significant texture information. For accomplishing that, the filtering window and the sigma were really low. Anyway, it failed because of time.

Taking the previous statements into account, the only pre-processing method used was the cropping method ('redData.m'). This process took less than 1 minute for all the possible window sizes. It is important to say that the 50x50 window was faster than the 200x200 window, but the difference of time was not significant.

The Data Separation was always done with 20 images per category. The script's results were obtained in less than 1 minute. As in the cropping, the time differences between windows was not significant.

The first filtering method ('ApplyFilters.m') was dependent of the window size. The 50x50 window filtering took less than 2 minutes, and the time duration was approximately duplicated as the window size increased to 100x100 and 200x200. The second filtering method ('ApplyFilters2.m') was also dependent of window size and was significantly slower than the first method. The 50x50 window filtering took less than 10 minutes, and the time rose up to less than 30 minutes for the 200x200 window.

Textons Computation was the slower function. It was strongly dependent of the window size. One could compute textons, assign textons, create and evaluate a model with all of its variants in a slower resolution (50x50), while the k=40 textons computation of a higher resolution (100x100) was running. The same situation applies for 100x100 and 200x200 but with higher times. Textons computation was also dependent of the filtering method. The second one took longer as it had 6 more dimensions than the first one. Finally, the time's difference between k-textons levels (40, 20 and 10) was approximately doubled between each level. The k-textons levels were low because higher k-textons took too long.

Textons computation (or textons dictionary) is a really slow and expensive process. This is a result of representing every pixel of an image as the response of n-filters. As n goes up, and the number of pixels goes up, the process will be slower and more expensive. That is why a good preprocessing method is crucial for obtaining good results.

The slower texton computation was obtained in the 200x200 window with k=40 (more than 3 hours), and it showed an 'Out of Memory' error after that time was elapsed. The same error was shown in the 200x200 window with k=10. One doesn't know why the error didn't appear when k=20. It is important to say that 'TextonsComputation.mat' used the default k-means' Matlab function (max 100 iterations). This caused that none of the kmeans clustering converged (regarding its variations). One suggests that the maximum of iterations should be higher, but the time it takes to execute will be longer. Another suggestion is finding a way for guaranteeing a good initialization of the k-centroids.

Texton Assignation was faster than the filtering phase and the Texton Computation phase. The slowest process took less than 10 minutes. Again, the time rose up as the resolution, k-textons and number and size of filters rose up.

Finally, the creation and evaluation of K-Nearest Neighbor was faster than the one of Random Forest. The longest

K-Nearest Neighbor took less than 2 minutes disregarding its parameters. The longest Random Forest process took less than 5 minutes.

The ACA results of each method can be seen in Tables 1-4. As one can see in the tables, the best result was obtained using Random Forest with the first filtering method, k-textons=10 and 50 trees (ACA = 0.5560). This method classified all of the training images right (ACA = 1), and 0.1120 of the test images correctly. As one can see, the ACA for the Test category is low, which suggests that the method's performance isn't the best. Anyway, taking into account the balance between time and results, the ACA is acceptable. It is fair to say that this method classifies test images better than luck (0.04).

Surprisingly, a similar result was obtained using k-Nearest Neighbor with the second method of filtering, k-textons=10 and k-Nearest Neighbor = 1 (Nearest). The ACA obtained using this method was the same as the previously discussed (gblACA = 0.5560, trACA = 1, tsACA = 0.1120). This method was slower than the previous one, so that is why it isn't considered the best.

As con note in Tables 1 and 3, the first filtering method has a better clustering (cluster's components are more similar between them than compared with other clusters) than the second method. This was deduced as the ACA of train declined faster in the second method compared with the first one. This suggests that the first method has a more robust clustering than the second one. In other words, it is easier to say if one element is part of one or another group. For more information about confusion between categories one can see the file 't2.mat' (test confusion matrix for the best result - Random Forest), and 't3.mat' (test confusion matrix for the 2nd best result - KNN).

As previously stated, the best method was obtained using Random Forest. This should be because Random Forest allows the comparison of multiple trees at a time. As one can note in Tables 2 and 4, the model performance was improved as the number of trees was higher. The effect of considering a higher number of trees, is taking into account more possibilities of distribution. One can get a similar effect in k-means using the option 'Replicates'. This allows the user to see different results depending on the initial positioning of the centroids. However, this option does not combine the results for obtaining a more robust method, as Random Forest does. Anyway, one can see some cases of over-fitting in Random Forest (Tables 2 and 4).

As one can see in Tables 1-4, in most of the cases the best development is obtained in a 50x50 window with k-textons = 10. This could be caused by two reasons. The first one is that one doesn't have to much information for dividing the data in more clusters than 10 (without causing bad clustering). This reason can be understood as seeking differences where they aren't. The second reason could be that as the k-textons increases, the number of iterations that it needs to converge also increases (maxIter increase suggested). Also, an increase of k-textons can be related to a major possibility of bad initial centroids positioning.

Finally, Random Forest usually gets better results than K-means for multi-classification problems. Random forests asks a question every time that it decides to split between to categories, while kmeans question is by default minimizing the distances inside a cluster and maximizing between clusters. In this way, Random forest usually produces more robust models.

It is important to notice that for building the models, the hole image was passed to the classifier as a vector. One could use histograms, but it would lose the spatial information. That is why I decided to build the models directly with the texton map representations. Anyway, histograms can be useful for detecting which filters are the most discriminating. The ones that show the least distance are not very useful when deciding if its one class or another. Histograms can be used for optimizing the Textons Computation phase by the elimination of non-discriminative filters.

This method can be improved by modifying some of the hyper-parameters, specially when creating the textons dictionary (as distance and maximum number of iterations). Also, it would be interesting to see the model's response when modifying the training set. It would be useful to see how it worked in the train subsample proposed by Felipe Romero [2].

4. Conclusions

Classification using textons is a slow and expensive process. It can be useful when the textures are clearly dissimilar between them. In this database there were textures that look alike, so the classification performance might be poor.

The biggest limitations as stated before are the memory and time it takes. Also, this process is strongly dependent of the creation of the dictionary. If the dictionary is not the best (as in this case), the results won't be the best. For improving this one can use initialization methods for the cluster's centroids.

Additionally, the dictionary is usually created using kmeans (one of the most basic classifiers). One could try to create a dictionary with a most sophisticated classifier. Although kmeans is a basic classification method, it can provide decent results. At least it produced better performance than luck in the training set.

Another limitation is that the textons dictionary quality will be highly determined by the filtering method used. One must choose the filters that optimize the classification process, taking into account its orientation and its scale compared to the image's texture scale.

Finally, it is important to say that if one is approaching a multi-category classification problem, it is recommended

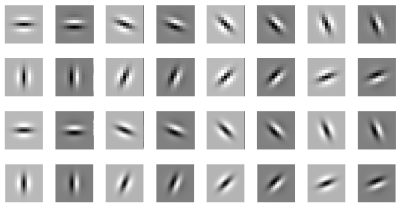

Figure 1. Filter Bank based on Gaussian second derivatives and its Hilbert transform.

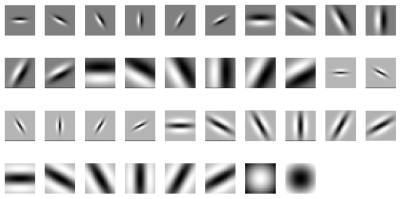

Figure 2. Filter Bank based on Maximum Response (MR).

to use Random Forest instead of K-Nearest Neighbor. Anyway, there are some special cases in which KNN can show similar results, but this is strictly related with luck in the centroids initialization.

References

[1] Arbelaez, P. *Lecture 6: Representation*. Universidad de Los Andes. 2018.

[2] Romero, F. *05-Textons*. Universidad de los Andes. 2018.

[3] Oxford University. *Texture Classification*. Visual Geometry Group. Department of Engineering Science. 2007.

[4] Mejia, J. *Lab4: Hybrids*. Universidad de los Andes. 2018.

5. Images

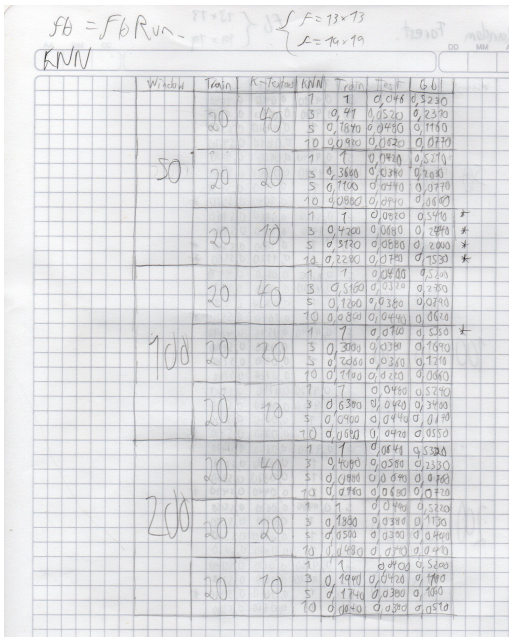

Figure 3. Table 1. ACA for different parameters using KNN and filtering with the first method. Best results highlighted with a '*'.

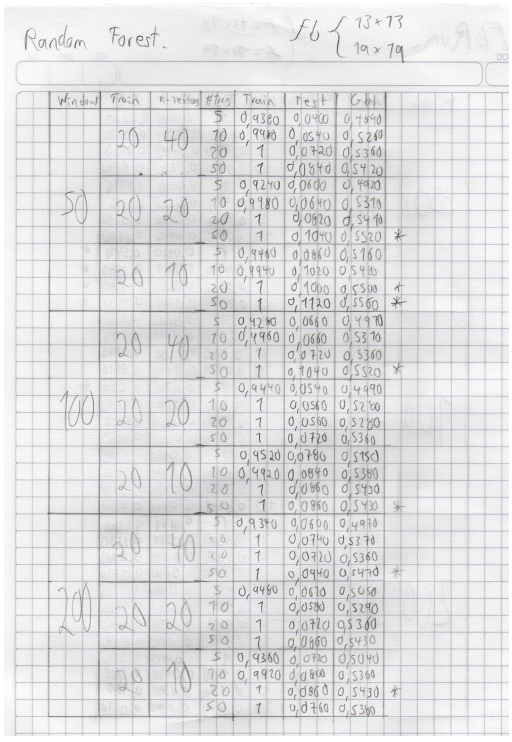

Figure 4. Table 2. ACA for different parameters using Random Forests and filtering with the first method. Best results highlighted with a '*'.

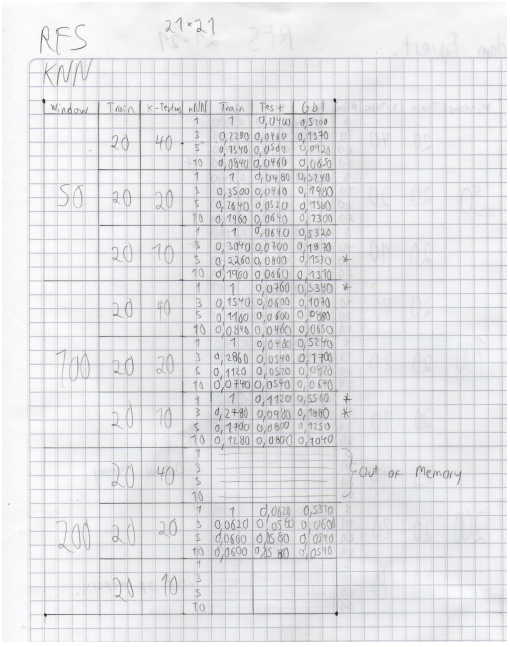

Figure 5. Table 3. ACA for different parameters using KNN and filtering with the second method. Best results highlighted with a '*'.

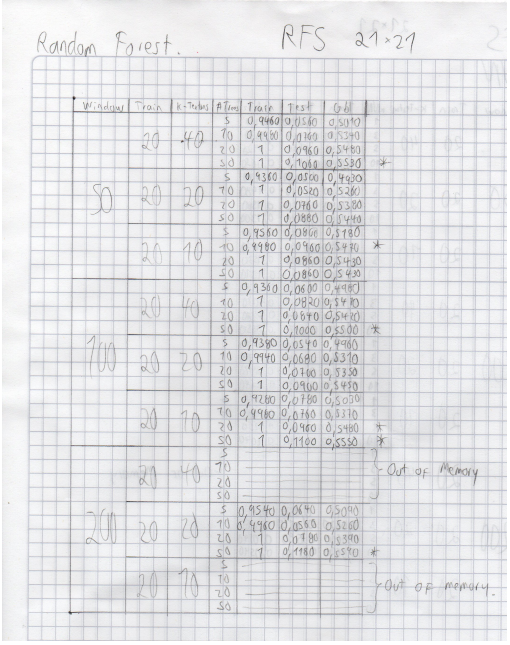

Figure 6. Table 4. ACA for different parameters using Random Forest and filtering with the second method. Best results highlighted with a '*'.