Computer Vision IBIO4490 - Textons Lab Report

Paula Florez Herrera (pp.florez2215@uniandes.edu.co)

***Abstract*—The necessity for identifying that in nature, some boundaries are actually not defining of the limits of an object but the texture within said object, makes it indispensable to develop a method to understand the behavior of boundaries in a repetitive texture. This laboratory used a texture database to evaluate a series of training and testing images and the results obtained are presented in the following report.**

I. Introduction and Database

THE database used for this laboratory comes from the Ponce Group website [1], where a wide amount of other databases can be found and downloaded for training and testing of algorithms.

The database contains a folder with 750 images intended for training and 250 intended for testing. These images contain a wide variety of textures including straight lines, curves, circles and many others in different orientations and scales in order to better prove the efficiency of the classifiers to be developed and tested.

II. Methods and Filters Used

A. Functions Used

First of all, and in order to fully understand the methodology of the process of texture recognition and classification, the given Script (Example.m) was ran and comprehended. For this, the following subfunctions are explained:

fbCreate: This function generates a filter bank with the amount of variations of filters given by the input. In case of not giving any input number, the default value for the function generates 32 texture patterns (a 16 by 2 cell).

fbRun: This function generates a set of filtered images by the convolution of the input image and a filter bank such as the one created with the previous function. In the case of the example, the image is convoluted with each one of the 32 texture patterns and a resulting set of 16 by 2 images is produced.

computeTextons: This function generates a texton dictionary using the filtered images and the number of clusters to be obtained. This texton dictionary contains a matrix for the indexes of the clusters and a matrix for the locations of the centroids of said clusters. Both these matrices are built with the MATLAB function 'kmeans'.

assignTextons: This function evaluates the centroids of the clusters obtained with the texton dictionary against the filtered images product of running the filter bank with the original image. It obtains the Euclidean distances between the textons and the actual data and chooses the minimum one to assign said point of data to its closest texton. Finally, it returns a matrix with the assigned texton for each point.

B. Descriptors

The following part of the laboratory involved using the provided function 'computeTextons.m' to generate a set of descriptors of texture. By evaluating the first of every one of the 25 categories of texture in the train folder with the 32 textons generated by the fbCreate function, a set of responses to textures was created. This was done for 40 clusters. This number was chosen randomly after trying with different numbers of clusters and finding this number an appropriate one.

III. Classifiers and Distance Metrics

A. Nearest Neighbor

The Nearest Neighbor classifier is based on the distance or difference of the histograms that represent the distribution of points in all the clusters for images done in training and testing. For this classifier, the chi-squared distance was used, which means there was no need for a training of the classifier itself, since it only obtains the difference between the given histograms. For the test, the results are obtained by comparing the test histograms with the training histograms in order to draw conclusions.

B. Random Forest

The random forest classifier is based on a series of decisions made over an image in order to assign it a final label. The graphic representation of this process resembles a tree. The parameters used are: Up to 80 trees in the forest and at least 5 decisions before reaching the end of a branch of said tree (a leaf). This parameters where chosen because the error for less trees or decisions per branch was still too high to be conclusive. More, however, implied a much greater computational time.

To train the classifiers, the function assignTextons was used.

IV. Results and Discussion

A. Descriptors

After creating the texton dictionary and running the function to compute the textons, the results obtained are shown on Figure 1.

To know which of the filters is more discriminating, the data was viewed as a set of points, each of which had a probability to be assigned to a centroid, this made every cluster seem

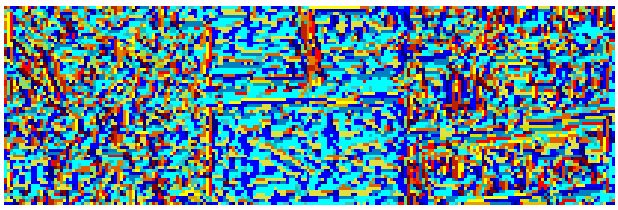

Fig. 1. Examples of Results of Texture Computing

like a Gaussian 3D bell. If the filter is highly discriminating, the response will be ample and disperse, so by comparing the dispersion of the clusters between them and on every filter it's more clear to determine the most discriminating filter. This one, after the respective testing, corresponds to filter No. 26 for all clusters. This filter is shown in figure 2 by plotting the resulting matrix of the filter bank array in the position 26:

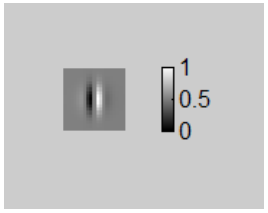

Fig. 2. Most discriminating filter: Number 26

After running the whole code with the original filter bank (that contained filters such as the one shown in figure 2) two circular gaussian filters were added in order to compare the performance of the model with these.

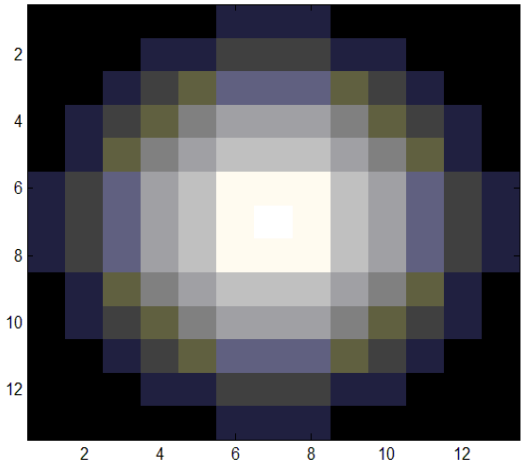

Fig. 3. One of the Circular Gaussian Filters added to the Filter Bank

The filters added looked like the one shown in figure 3 and their addition resulted on the creation of a new spatial layer in the Filter Bank. The same code used for the regular filter bank was used with this modified filter bank in order to obtain the new confusion matrices for this type of filter bank.

B. Classifiers

The classifiers were used within a function and the comparisons between the histograms of training and testing for each method are shown in this section.

1) **Nearest Neighbor**:

For the nearest neighbor classification, the result was generally faster, as it is shown on Table I. Given that the training of the classifier implied to calculate the difference from the training images with themselves, the result is of 100 percent accuracy, because the closest to each image will always be the image itself. This result is shown in Figure 4.

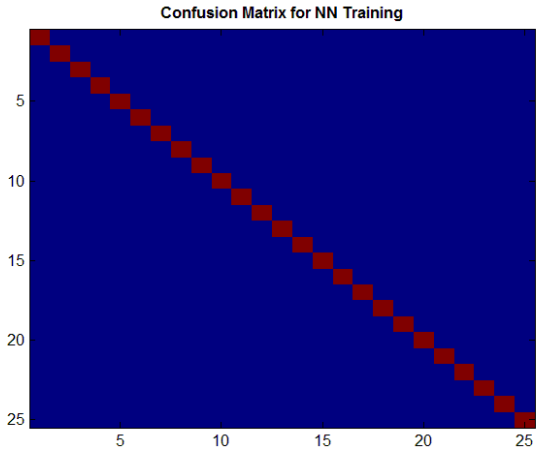

Fig. 4. Confusion Matrix for Nearest Neighbor Training

The next step was to use the testing images to compare to the training images just as it was done for the previous confusion matrix. The result is shown in Figure 5 and it is clear that in this case, the result is not perfect. The value of the the trace for this matrix is 47.6, which means the error is strong and present on this method. With the diagonal of the matrix and a maximum analysis, the category with less confusion is number 7 and the ones with more are number 6 and 23.

Between the limitations of this method is that if the database contains images with more variety of textures, a simple chi squared difference between the histograms won't establish a real classification between separate categories. However, for the Ponce Group dataset, this method seems to be doing a good job.

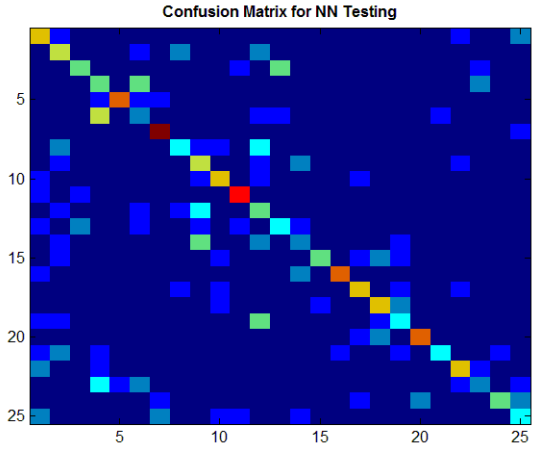

Fig. 5. Confusion Matrix for Nearest Neighbor Testing

When the above classification was finished, the code was re-ran with the filter bank designed to contain the new circular filters, the result of training and testing is shown in Figure 6 and Figure 7 respectively.

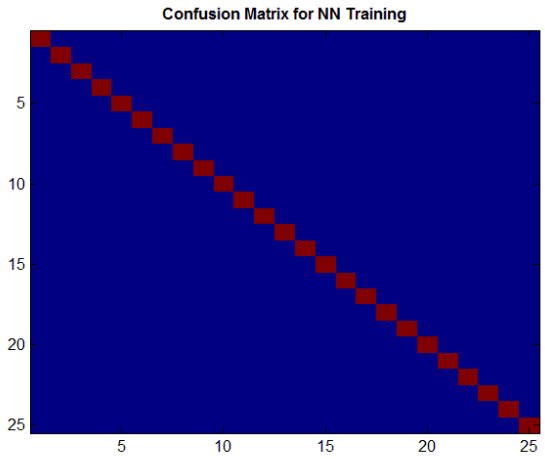

Fig. 6. Confusion Matrix for Nearest Neighbor Training with the additional Filter Bank

As expected, the confusion matrix obtained for training has no error, for the same reason that this happened with the original Filter Bank. However, an unexpected error in the testing stage caused every image to be sorted into the same category. After checking the code and understanding the dynamic of the classificator, one of the reasons for this could be the weight of the gaussian filters created to be added to the original filter bank, this could have had an effect on the classification of the testing images by generating histograms so generic that most of the images were closer to them than to their own category.

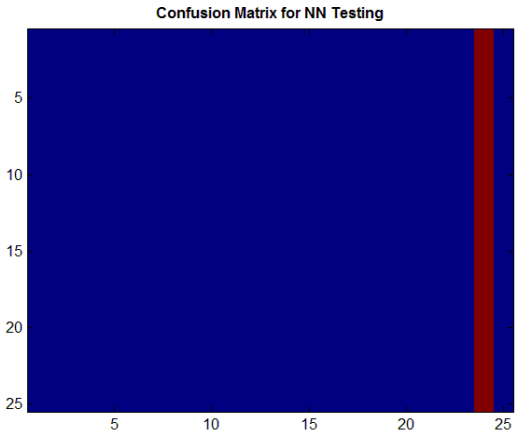

Fig. 7. Confusion Matrix for Nearest Neighbor Testing with the additional Filter Bank

2) **Random Forest**:

The implementation of the Random Forest method using the TreeBagger function of Matlab proved to be more complicated than the implementation of the first method. On Figure 8 it is shown the confusion matrix for the training of the 80 tree classificator, however, the error on this matrix is huge (around 90 percent) and proves there was an error on the implementation that will limit the accuracy of testing. The computational time taken for this was about 3 minutes as seen on Table I.

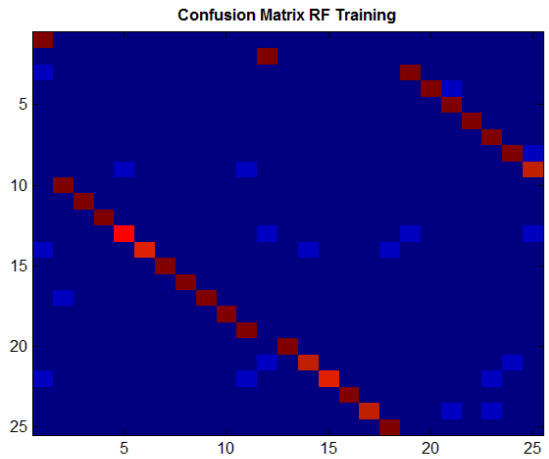

Fig. 8. Confusion Matrix for Random Forest Training

As expected from the previous result, the confusion matrix for testing the Random Forest method shows a pattern of error identifiable on Figure 9.
Lastly, the TreeBagger function allows to obtain an

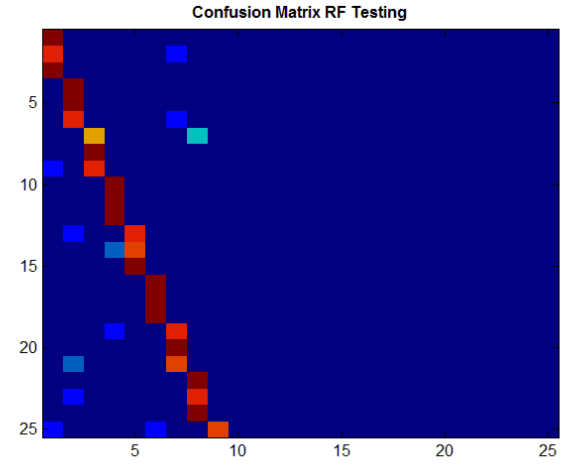

Fig. 9. Confusion Matrix for Random Forest Testing

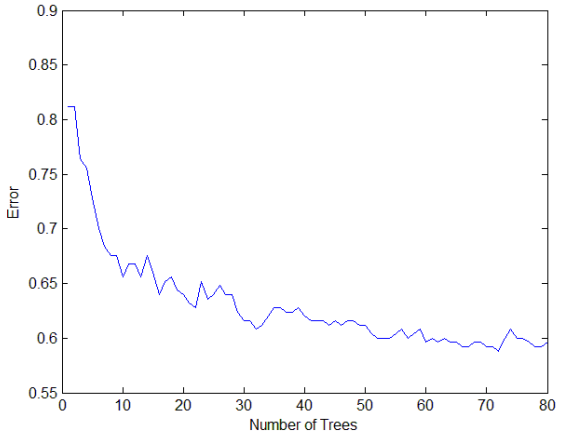

Fig. 10. Error Graph for Random Forest Classification

output that generates an error graph and relates it to the amount of trees used in the forest, this graph is shown on Figure 10 and it shows how the number of trees is mostly directly related to the final error of the method. However, the drop on the graph is prominent between 0 and 30 trees, and after that it gets less and less pronounced, which suggests that the error saturates at one point, making it a good choice to pick 80 trees since a much higher number wouldn't have made as much of a difference but would have consumed a lot more of computational time.

Between the limitations of the method is the consumption of computational time, as well as the difficulty to successfully apply it, because it can produce testing errors greater than expected when compared to a simple method such as Nearest Neighbor. Random Forest was expected (and should be) the superior method, but the implementation proved that Nearest Neighbor is easier and produces less unexpected errors.

TABLE I
TIMES OF DIFFERENT PROCESSES IN MATLAB

Process	Time
Nearest Neighbor: Training	9.59 sec
Nearest Neighbor: Testing	3.28 sec
Random Forest: Training	180.69 sec
Random Forest: Testing	53.88 sec

The final process was to use the modified Filter Bank with the Random Forest classification, in order to observe if the results of the gaussian filters were as different to the original with this type of classifier as they were with Nearest Neighbor. The training and testing confusion matrices are displayed on Figures 11 and 12 respectively, as well as the error provided by the TreeBagger function on Figure 13.

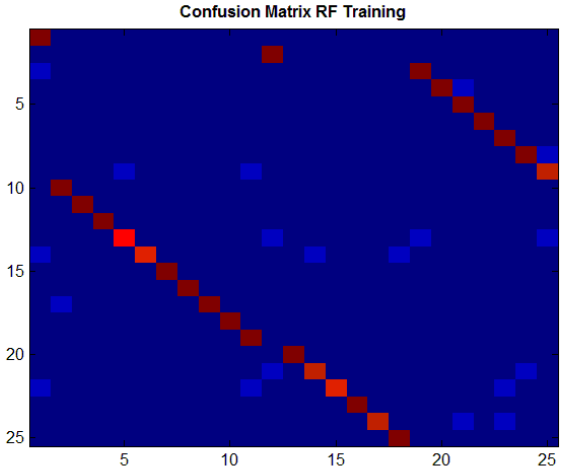

Fig. 11. Confusion Matrix for Random Forest Training with the additional Filter Bank

Given that the code for Random Forest presented a series of errors while using the original Filter Bank, it was expected that the results for the modified filter bank presented at least the same amount of errors. The general pattern for both training and testing when compared to the original filter bank is the same, this could mean that the extension of the filter bank with the gaussian filters and the extra layer do not solve the error in the Random Forest classification, however, they do not amplify it either.

When analyzing the error graph provided by the Tree Bagger function, the main form of the graph is the same as it is for the original filter bank, however, the values change an it is clear that the addition of this layer and new filters reduces the main error as the number of trees increases.

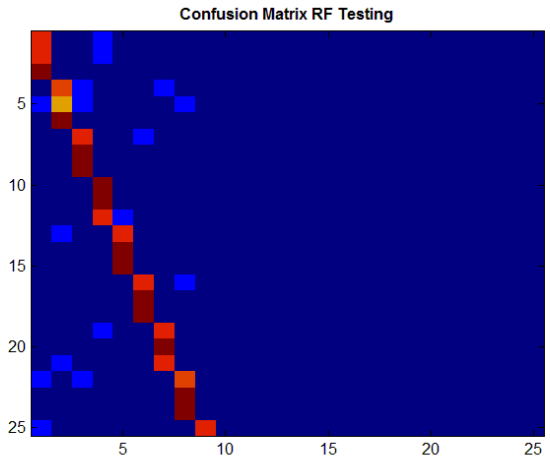

Fig. 12. Confusion Matrix for Random Forest Testing with the additional Filter Bank

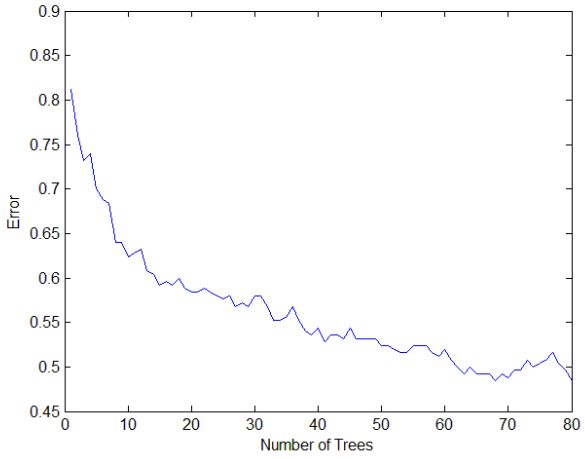

Fig. 13. Error Graph for Random Forest Classification with the additional Filter Bank

V. Conclusions

As a conclusion, there is a great amount of classificators that, which a properly designed and applied training, could work to solve complicated problems from a simple approach. The implementation of the methods is a crucial part, because the best method can get easily confused if it's not trained appropriately.

The addition of the gaussian filters and the extra layer to the original filter bank proved to cause a decrease in the effectiveness of the Nearest Neighbor method, however, it improved the result of performance on the Random Forest method. This was not what was expected to happen, given that the implementation of a more detailed and complete filter bank was supposed to improve the performance of both methods used on this laboratory. The failure of this could be attributed to the propagation of errors from the methods implemented, or as it was explained in the past section, the design of the gaussian filters itself.

Regarding the computational times shown in Table I, Nearest Neighbor is predominantly faster because it is a simpler classifier, and testing time is always less than training because of the amount of images for each set of images differs. This pattern also applies to the modified filter bank used with each classifi.

There must be a huge improvement on the code in order to obtain useful dataset analysis, given than the error for all the matrices is wildly high and would not be comparable to the classification results seen on the literature for these kind of algorithms in Artificial Vision.

References

[1] Www-cvr.ai.uiuc.edu,. 'Ponce Research Group: Datasets'. N.p., 2015. Web. 25 Mar. 2015.