Image Classification using texture Information

Alejandro Trujillo
Universidad de los Andes
Bogot, Colombia
af.trujilloa@uniandes.edu.co

Santiago Martnez
Universidad de los Andes
Bogot, Colombia
s.martinez1@uniandes.edu.co

Abstract

1. Introduction

One of the objectives with respect to the detection of images is to be able to label them, using pixels as a reference, that is to say having given them a previous category, in order to realize this play with the pixels one must have a classification of objects in which is included information of the image as the form, texture this is achieved through the use of textones. Once the model is trained it is when it can be used for visual recognition and to be able to segment images [1] the result is obtained using the probability between 0 and 1 of each pixel belonging to each of the images and to work better this probability you have to enter a large amount of material

2. Methodology

A database provided by cifar-10 was used for this task, this database consists of 60 000 labeled images within 10 semantic categories for training and another 10 000 labeled images for testing. All these pictures have a resolution of 32x32 and are in RGB format. The categories included in this database are: trucks, ships, horses, frogs, dogs, deers, cats, birds, automobiles and airplanes. The pictures, labels and filenames were ”pickled”, which made necessary the usage of the pickle python library in order to get the dictionary containing the necessary data. The first step is to unpickle the information, especially the images and their labels, as they play a crucial role in designing a classification model. As a training set, 1000 images were selected (100 for each cathegory) contained in the second batch of training images. Like said before, we want to use two different methods for supervised classification, which are K-Nearest-Neighbors (KNN) and Random Forests (RF). The python library ”Sklearn” has already implemented these algorithms, so we decided to use it. In order to use these algorithms, we must convert the train images into a 1D representation, that takes into account texture, knowing this, we approached the problem by using the textons representation of the images, and then computing the histogram of the texton map for each one. The basic theory behind this procedure consists of filtering the image multiple times with different filters, which include borders (oriented in multiple orientations) and dots of different sizes, and save the filter response of the image. This was done using the provided function fbCreate. which parameters are support and StartSigma, initialized in 2 and 0.6 respectively. Next, we use the filter bank created on the image, the function fbRun was used for this purpose. This function takes an input image and filters it using the input bank filter, and returns a structure with the response of the image for every filter. This process was done for all training images in order to get a full structure of all possible textures within the database. After this, we need to find the texton library for the problems. This was done by using the given function computeTextons, which uses k-means to cluster the textons given a declared k (number of clusters) and using the centroids as textons. Computing this library demands a high amount of computional resources depending on the number of clusters selected. After having the library, we proceeded to assign the textons to all train images, which gave us the texton maps for them. The histogram of these matrices is the descriptor used in the classification algorithms. this process was made for 19 different ks while computing the texton library, in order to optimize the results. Both methods, KNN abd RF require the histograms of the texton maps and the labels. After the model is finished, it can be used to predict a semantic category given a specific texton map histogram. The model was used again over the train images and over the 10 000 test images and the confusion matrix was computed. In order to get another metric for the efficiency of the model, the ACA was calculated (sum of the diagonal of the normalized confusion matrix). The Random forest algorithm is slightly slower to compute in comparison to the KNN algorithm, especially as the k augments.

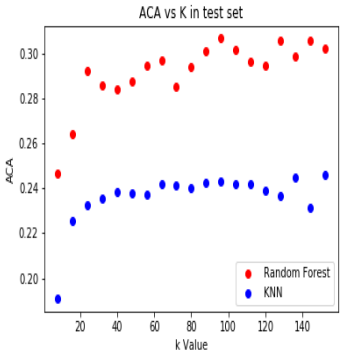


Figure 1. ACA vs k in the train set

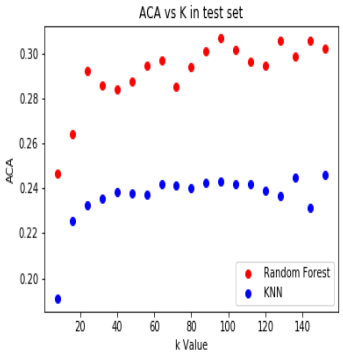


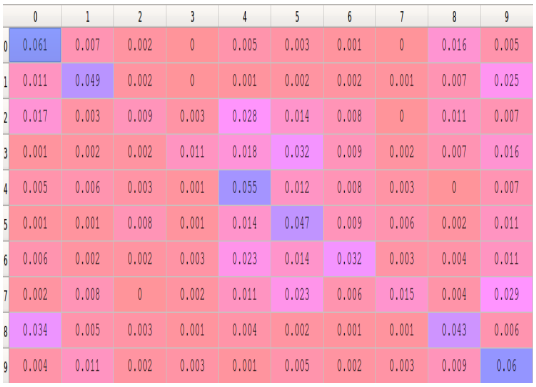
Figure 2. ACA vs k in the test set

3. Results

After all the experiments iterating the algorithm with a different number of clusters when computing the texton library, The results for this experiments can be seen in figures 1 and 2. Analyzing the results obtained, the best option was the Random Forest classifier, using a k of 96, with which we achieved an ACA of 0.3066 in the test set. The behavior of the ACA along the two different classifiers was odd, as in the train results, KNN had a higher ACA, however when the models were used on the test set, Random forest was clearly superior. This could be a case of over-adjustment of the model. The normalized confusion matrices obtained for this experiment can be seen in figure 4 and 3. In this matrices we can see how the algorithm is better at classifying category 1 and 9 (trucks and airplanes) and fairly good at

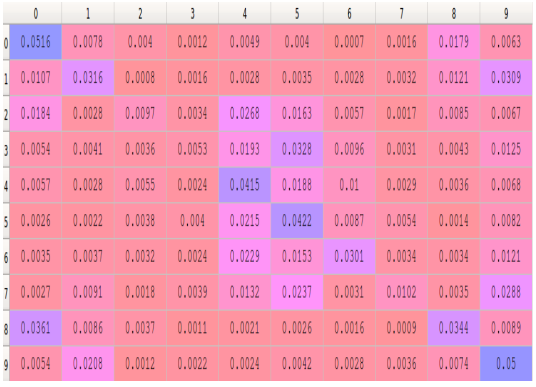
4. Conclusion

5. Annexes



	0	1	2	3	4	5	6	7	8	9
0	0.061	0.007	0.002	0	0.005	0.003	0.001	0	0.016	0.005
1	0.011	0.049	0.002	0	0.001	0.002	0.002	0.001	0.007	0.025
2	0.017	0.003	0.009	0.003	0.028	0.014	0.008	0	0.011	0.007
3	0.001	0.002	0.002	0.011	0.018	0.032	0.009	0.002	0.007	0.016
4	0.005	0.006	0.003	0.001	0.055	0.012	0.008	0.003	0	0.007
5	0.001	0.001	0.008	0.001	0.014	0.047	0.009	0.006	0.002	0.011
6	0.006	0.002	0.002	0.003	0.023	0.014	0.032	0.003	0.004	0.011
7	0.002	0.008	0	0.002	0.011	0.023	0.006	0.015	0.004	0.029
8	0.034	0.005	0.003	0.001	0.004	0.002	0.001	0.001	0.043	0.006
9	0.004	0.011	0.002	0.003	0.001	0.005	0.002	0.003	0.009	0.06

Figure 3. Confusion Matrix for the Train set



	0	1	2	3	4	5	6	7	8	9
0	0.0516	0.0078	0.004	0.0012	0.0049	0.004	0.0007	0.0016	0.0179	0.0063
1	0.0107	0.0316	0.0008	0.0016	0.0028	0.0035	0.0028	0.0032	0.0121	0.0309
2	0.0184	0.0028	0.0097	0.0034	0.0268	0.0163	0.0057	0.0017	0.0085	0.0067
3	0.0054	0.0041	0.0036	0.0053	0.0193	0.0328	0.0096	0.0031	0.0043	0.0125
4	0.0057	0.0028	0.0055	0.0024	0.0415	0.0188	0.01	0.0029	0.0036	0.0068
5	0.0026	0.0022	0.0038	0.004	0.0215	0.0422	0.0087	0.0054	0.0014	0.0082
6	0.0035	0.0037	0.0032	0.0024	0.0229	0.0153	0.0301	0.0034	0.0034	0.0121
7	0.0027	0.0091	0.0018	0.0039	0.0132	0.0237	0.0031	0.0102	0.0035	0.0288
8	0.0361	0.0086	0.0037	0.0011	0.0021	0.0026	0.0016	0.0009	0.0344	0.0089
9	0.0054	0.0208	0.0012	0.0022	0.0024	0.0042	0.0028	0.0036	0.0074	0.05

Figure 4. Confusion Matrix for the Test set