Classification of images based on textures

Laura Bravo
Universidad de los Andes
Cra 1 #18a-12, Bogotá, Colombia
lm.bravo10@uniandes.edu.co

Juan Pérez
Universidad de los Andes
Cra 1 #18a-12, Bogotá, Colombia
jc.perez13@uniandes.edu.co

Abstract

Textons are the micro-structures found in natural images that are considered the basis of human perception. In this work a texton dictionary was generated and two classification algorithms were trained to classify images in one of 25 categories of textures. The first algorithm, a KNN classifier, managed to have an ACA of 6.8%, while the bagged trees algorithm had an ACA of 2.0%. The performance of both algorithms was poor, and the conjecture made around this is that the dictionary was too small to cover the variance of each class of textures, so that the algorithms were not well trained for the incoming data. As future work, larger datasets are going to be processed with no subsampling or cropping of the images, so that more stiff data is obtained and the algorithms can achieve better performance.

1. Introduction

Texture is a concept common to humans but, unlike color or borders, is not easily described by a single value, like an intensity. In the same way, it can not be described by the result of a simple mathematical function, such as a convolution with a kernel or the selection of a threshold. Nevertheless, texture is very important for computer vision, since textures are quite characteristic of the object in which they appear and, therefore, if one can obtain information about the object's texture, one could, in principle, use this information to segment the object and classify it.

Given that texture is not easily described, the approach to study it is more of the exhaustive-search type. This means that trial and error with different textures will be applied to it, and a search for the texture that is closest to a part of the image will be run. That part of the image will be said to have the same texture as the texture that was closest to it. Obviously, like all trial and error algorithms, it would actually need to have a large amount of textures so that almost every possible texture appears at least once; additionally, this will demand very high computational power. The idea is to breakdown these textures into their basic components, that is, the primary elements whose combination can make textures. This elements are called *textons*.

Textons refer to fundamental micro-structures in natural images and are considered as the atoms of pre-attentive human perception [1]. Again, unlike colors, the texture at a single pixel can not be determined just by the pixel's intensity, rather, it is a function of its neighborhood, regarding size, orientation, and color variation, among others.

In the present work a dictionary comprising different textons was created by computing the response a pixel, and its neighborhood, had to a given filter. Then, an unsupervised algorithm, namely *kmeans*, was run to generate clusters based on these responses. The image was transformed from pixels standing for grayscales, into pixels standing for the texton they 'belong' to. As is usual when classifying based on color, the histogram of the transformed image was taken and this representation, of a part of the dataset, was given to two classification algorithms: a *KNN* classifier and a *bagged trees* classifier. Whenever there was an incoming image, the response to each filter was computed, textons were assigned based on distance to previously computed textons and the histogram of this new image was given to each algorithm for classification.

2. Materials and methods

2.1. Dataset

The dataset for this exercise was obtained from the Ponce group [2]. It was divided into two sets, *test* and *train*. Both sets had many orientations for 25 different textures, comprising things like wood, stripes, rectangular patterns, rocks, bricks, carpets, fur, among others. The *train* set had 30 instances of each textures, while the *test* set had 10 instances.

2.2. Obtaining texture

In order to obtain the texture of each pixel, a filter bank was used, consisting of bars in different orientations, and it was then applied over all the image. Each pixel could then be represented as a point in $\mathbb{R}^n$, where n is the number

of filters applied to the image. In this space every point corresponds to the response that a pixel had to a particular filter.

2.3. Generation of the texton dictionary

First, a subsampling of the *train* set was performed by cropping the images, to reduce size, and randomly selecting 5 images of each texture. As stated before, all the 32 filters were applied to each image of the set and then *kmeans* with $k = 64$, was run over the pixels to obtain their labels as well as the centroids of the clusters, which are the textons. This process has a high computational complexity due to the amount of points in $\mathbb{R}^{32}$, and took several hours to run on the server.

2.4. Classification based on textons

Once the texton dictionary was assembled, the images from the *train* set can be expressed as a matrix of the same size as the original image, but instead of an intensity at each pixel, it has the texton each pixel corresponds to. Then, a histogram of this matrix is computed, with as much bins as the number of textons, 64 in this case. Given that the class of each one of these images is known, a relationship between the histogram of the image and its class can be generated, in principle, through a supervised classification algorithm, such as *KNN* or *bagged trees*.

With the models obtained from these algorithms new images can be classified. When there is a new image, the response to each of the filters in teh filter bank must be computed and, with these responses, the pixel can be given the same class as the texton closest to it, given some distance metric. That way, just like with the *train* set, the map of these labels is generated as a matrix and its histogram of textons can be computed. Now, this histogram can be given to the trained models, which will classify the new histogram into one of the classes of the *train* set.

The first parameter to be chosen is the number of textons. This must be selected taking into account that textons, as entities found in nature, must be continuous and that in the algorithm, naturally, this is discretized; therefore, textures found in pictures are probably some combination of the filters generated here, so that the number of textons are not likely the same number of filters. Here we supposed that one of the filters was an option (one out of 16) and that it could be combined with one of the filters from the other side of orientations and the other scale (one out of four), accordingly, the total number of textons must be $16 \times 4 = 64$. Regarding the machine learning algorithms, the parameter k, for the *KNN* algorithm, which is the number of neighbors to take into account, was chosen to be 10, and the distance metric was *euclidean*. The number of trees for the *bagged trees* algorithm was also 10.

When writing the algorithm, some differences to the responses of different filters was observed, namely, the bigger filters are more discriminative than the smaller ones. This is because bigger filters need bigger structures in the image so that some response occurs, while small ones are more likely to find structures to respond to. Nevertheless, all of this depends on the shapes found in the image and can not be generalized.

2.5. Evaluation

The previous process, of predicting the textures of the test images, requires a quantification of the classifier's ability and the steps before it have to do this correctly. This assessment is obtained by organizing the predicted labels (textures) into a matrix (confusion matrix) that for each test image identifies in what category (texture) it was classified. The diagonal of the matrix corresponds to the amount of images that were correctly classified according to the ground truth. Once normalized, the matrix can be resumed with the ACA, the average of the diagonal.

3. Results

From the two classifiers we obtained a prediction of the texture categories of the test images. According to the values of the ACAs, *KNN* was a better texture classifier with 6.8%, while for *bagged trees* it was 2.0%. Neither of these methods are what would be desired, that is, for them to classify correctly most images an fail in few of them, which would correspond to an ACA in the double digits. Nevertheless, the *KNN* classifier probably did classify, because it surpassed what would have been possibly classified by chance. This did not happen with *bagged trees*.

By examining closer the results, it can be said that the categories that caused more confusion were firstly different for each classification method as was expected. But particularly, for *KNN* it consistently classified most of the images of the categories from 14 to 25 as the first type of carpet. This could be because of the filters used, particularly their scale. The last textures are characterized by being composed of small elements, with the filters being small also, they could have created textons that responded to this similarity, thus confusing them amongst themselves. Additionally, given that the result of *kmeans*, that is to say, the textons, depends on the initialization, the carpet as a confusing texture could have been any of the other textures composed of small patterns. On the other hand, for *KNN*, a similar confusion occurred, the second type of wood was confused with the final textures. This could have been caused by the easily identifiable orientation of the textures in the images. The confused images are clearly orientated similarly to the filters used, this response could have been what inclined the *KNN* classifier to assign them all the same texture.

After attempting to execute the identification of textons for the dictionary on several occasions, it was particularly

important to choose a proper k for *kmeans*. That is, not only because the result of *kmeans* depends on it, but because as the number of clusters increases, so does the computational complexity, and thus the time in which the texton dictionary can be obtained. This is explained by the algorithm of *kmeans* in which there is a constant re-organization of the points, that leads to a recalculation of the clusters, until conversion or a maximum number of iterations is reached.

Training both classifiers took almost no time -less than 10 seconds- even in regular computers, not servers. Applying the classifiers took not more than one second per instance in both classifiers. Nevertheless the problem with applying classifiers is the preprocessing needed to feed the algorithm, since it requires passing various filters (16 in this case), assigning a texton to each pixel in the image, and computing the resulting histogram. All of this adds up to a very large amount of computations which, for the entire *test* set, had to be run on the server.

There is a gigantic limitation of this method and it is because it relies on the output of *kmeans* for the rest of the conjectures that are made. First, the input k will determine the amount of possible textons, if it is below the 'actual' number of textons it will cause textons to be merged as one, so that differentiation will not be possible; if, on the other side, k is above that number, textons will be broken even when they are the 'same'. Second, the inherent randomness coming from *kmeans* may cause small errors when assigning the labels for each pixel that will then be considered to be the *truth* when classifying incoming images. Third, there is no order between textons when creating the histograms, as opposed to histograms generated from gray intensities; this represents a problem because the way *kmeans* generates the seeds with different labels is random, therefore, there is no sense of *closeness* between, say, cluster 5 and 6, and there will be nothing that can fix the possible misclassification between adjacent bins. In the representation of each image in $\mathbb{R}^n$ this can lead to mis-classification from the generated models. Fourth, there is a limitation coming from the 'discretization' made on textures, since there is an assumption being made on the fact that textons can be represented as some sort of linear combination of the filters generated here, which may not be true, or may be partially true if the discretization is not enough for this to be a good model. Finally, scale becomes an issue that could partially be solved by generating a lot of filters looking out for these, but this will increase significantly the computational complexity, so that there is probably an optimum to be found but it will depend on the images being processed, so that the only way to find it is trial and error.

Additionally to the proposed evaluation of the classification used here (ACA and confusion matrix), the evaluation of multiclass classification problems can be interpreted as a binary classification among the classes *class X* and *not*

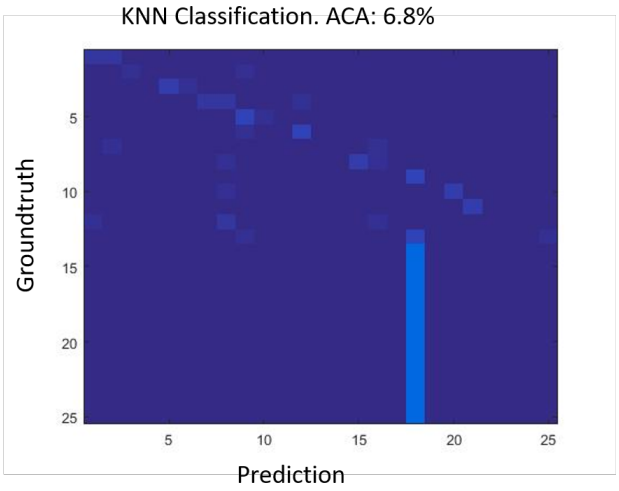

Figure 1. Confusion matrix resulting from the classification by KNN. The rows are the groundtruth and the columns the predictions. Normalizing this matrix gives an ACA of 6.8%

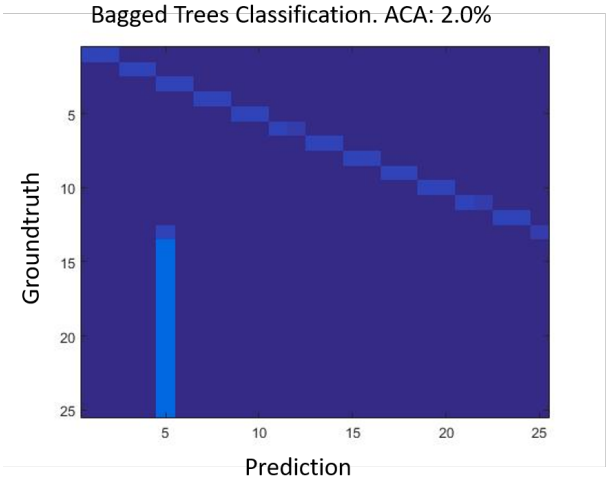

Figure 2. Confusion matrix resulting from the classification by Bagged Trees. The rows are the groundtruth and the columns the predictions. Normalizing this matrix gives an ACA of 2.0%

class X, hence, precision and recall can be computed for each class and be a way of evaluating the algorithm's performance. This computation is equivalent to making sums over the confusion matrix [3]:

$$Precision_i = \frac{M_{i,i}}{\sum_j M_{j,i}}$$

$$Recall_i = \frac{M_{i,i}}{\sum_j M_{i,j}}$$

4. Conclusions

The classifier that worked best was KNN. This could be due to its intrinsic simplicity that may adjust better to this type of problem. But additionally, its functionality is analogous to a mean filter, each pixel it takes into account the results of its neighbors to decide on its value. This process could help to reduce the amount of noise existing from the output of k means, the calculation of the textons.

The simplest way to improve the algorithm would be to generate a gigantic texton dictionary, based on a large dataset, and a really high k that contemplates all the combinations between the filters that will likely generate the textons. This, obviously, will increase computational complexity greatly, since it is close to a brute-force search. Another process that could be added to the algorithm would be to not assign textons to incoming pixels based only on the textons themselves, the centroids, but on all the data, that is, train a classifier, so that this classification is more accurate than just computing distances to single points one of each class. But, it is relevant to mention that the algorithms used here, the classifiers, kmeans and also the filters, were not optimized properly as can be seen by the classification accuracy.

References

[1] S. Zhu, C. Guo, Y. Wang and Z. Xu, "What are Textons?", International Journal of Computer Vision, vol. 62, no. 1-2, pp. 121-143, 2005.

[2] "Ponce Research Group: Datasets", Www-cvr.ai.uiuc.edu, 2017. [Online]. Available: http://www-cvr.ai.uiuc.edu/ponce_grp/data/. [Accessed: 02- Mar- 2017].

[3] "How do I compute precision and recall for a multi-class classification problem? Can I only compute accuracy?", Quora, 2017. [Online]. Available: https://www.quora.com/How-do-I-compute-precision-and-recall-for-a-multi-class-classification-problem-Can-I-only-compute-accuracy. [Accessed: 03- Mar- 2017].