Projecte 2: Tagging

Intel·ligència Artificial

Escola d'Enginyeria

Universitat Autònoma de Barcelona

Verd-fosc
Verd
Blau
Taronja
Taronja-groc
Blanc
Groc

Rubén Reyes Andrades – 1382357

Marc Espinosa Gil - 1495036

Índex

AI Practical Report – Tagging

Introduction

This is our Tagging Project Practical Report and Analysis. After being more than one month working on the implementation of the methods to make Tagging python project work, here we explain how we got it and what are the results of our complete analysis.

The purpose

This project has the aim to make a Python programme that can determinate which colours appear on an image using algorithms like K-Means. Tagging can read an array of 3-dimensions (RGB) of all the point of an image or using im.read(). After reading the image, we determinate some parameters like number of centroids and then we run K-Means until it converges. After this, the result is passed by different functions to convert it into 11-dimensional array of "ColorNaming" that depending on the percentage of the colour appearing and a single-value determinate which colours actually appear in the image.

We were told to implement some functions of K-Means algorithm, as well as, of Labels.py to make it work. Those exercises were divided into specific methods.

The first ones were about K-Means. Those were implementing the initialization of the algorithm, converting the 3-dimensional array into two dimensions, and making the algorithm, so we can iterate until it converges.
Then, we have to implement Labels to transform K-Means results into an 11-dimensional array (ColorNaming) and determinate which colours actually appear, the similarity between the ground-truth…

After having everything working successfully, the longest part of this project has been making the analysis of the 200 images that we were provided.

Brief explanation of our implementation

- _init_X:
    - In this function we initialize our working space X (data-set of points) from the image passed by parameters. If this image has 3 dimensions (RGB-like), we reshape it as the number of rows are the number of all the points and then the number of columns are the number of the dimensions, 3 in this case.

- _init_centroids():
    - We initialize the centroids array as the first different pixels of the image given, in case the selected option is “first”. Otherwise, if the selected option is “random”, we initialize the centroids array as random points of the image given.
- _cluster_points:
    - We calculate the distance array between the image pixels and centroids pixels. Then, we save the index of the minimum of all the distances array calculate for a point and all of the available centroids in the clusters.
- _get_centroids:
    - We recalculate the new centroids coordinates. Then we assign the centroids of the previous iteration to self.old_centroids. We clean the array of self.centroids and then, we group each centroid with clusters, and after that, we calculate the mean by columns to calculate the new centroids coordinates.
- _converges:
    - We create a new distance array that contains the distances between the centroids in this iteration and the previous one. Then we go comparing if any value is bigger than the maximum tolerance specified (“single-thr”) on options K-Means dictionary.

- bestK(), fitting(), plot results(), autoK() :
    - This point was optional. We use Fisher-Discriminant to test the efficiency of an execution of the K-Means algorithm. We decided to test for each image, K between 2 and 15. Then we calculate the Fisher value using fitting function. After having all the results, we pass them to the AutoK function to determinate automatically which K was the best. In this case, we used the elbow method. We defined that the maximum value of this equation was the bestK.
    - The equation is: for all the values we calculate the difference between previous fisher-K and actual fisher-K for all K values. Then we divided the previous difference by actual difference for all the values. After this, we chose the biggest value, that this means the change is "important" for us.
    - Once we get the bestK value, we plot the results and the K chosen with the plot_results function is marked in green colour. After this, we run again the K-Means algorithm for this K in bestK function to get the best results.
- Distance ():
    - In this function we calculate the difference between two arrays passed by parameters by columns. Then, we save those results in a new array that we return.
- ProcessImage():
    - This is the most important and complete function of the Tagging Project. Here we receive an image and the options dictionary. We change the image to the corresponding colour space for K-Means. Then we run K-Means with the options passed by parameters. If K value is lower than 2, bestK is ran, otherwise we run K-Means for the specified K. Then we get the name labels detected on the 11-dimensional space labelling the centroids. This is done by the next function.

- getLabels():
    - In this function we label all the centroids to their colour name depending on the "single_thr" value of options dictionary.

- SimilarityMetric() + Evaluate():
    - Those functions are used to determinate who successful our program and execution was compared to the Ground-Truth values. We compare the labels got with our execution with the ones of the Ground-Truth and we calculate a similarity value. Evaluate makes this for all the images ran in the execution, and it returns a global percentage of the success.
    - We want to mention that we think the equation given on the project guide was not correct, because the intersection of our colours and GT ones are divided by the length of our result, while we think it should be the opposite. The explanation says clear GT while equation says the other. We think we should have used what the explanation says, but we decided to keep as the equation result shows.

Resolution and problems of the exercises

In this project, exercises were divided in two parts. The implementation of the K-Means algorithm and the Labels algorithm. In K-Means the main problem we had to face was the complexity of working with different arrays and understanding what we were doing. After so many tries, we get everything working but the time it lasted for a single run was about 5 minutes. This was amazingly hard, so we started optimizing our code using a lot of functions of "numpy" library to solve the most part of the calculations. The programme reduced the needed time to 6,57 seconds (8,56s if we test also k=0). We know Teachers version lasted for 10 seconds so we thought it's good.

In other hand, Labels implementation was easier and had shorter functions to implement, but we need to know how "ColorNaming" library works. Once we understood completely all the points, everything became easier.

Other aspect that is relevant is that to execute the program completely we designed a script that test all the images given. We ran it twice, one with “single_thr” = 0,25 and the other equal to 0,75.

Each execution lasted around 5-6 hours to get completely. After all the results were saved on a .txt file, and then with Excel, we start analysing all the information we got.

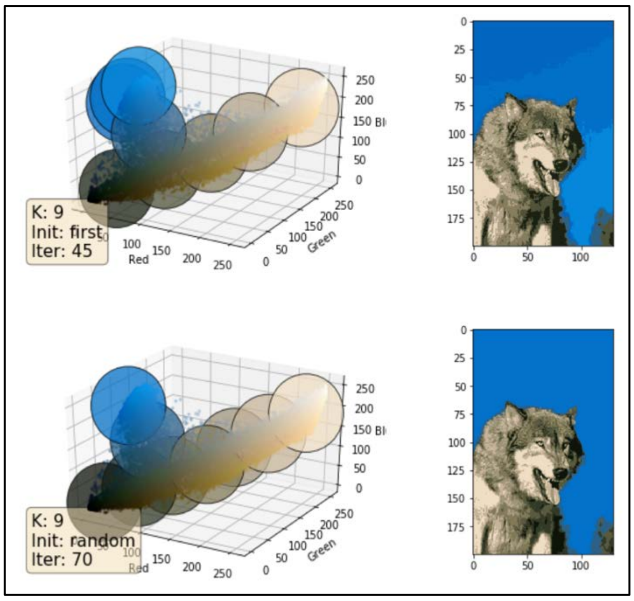
Analysis of 200 images

It is impossible to analyse completely all the details for the 200 images, but we tried to do it, so we ran the script explained before to have all the information ready.

To get ourselves understood, we think it’s better to explain deeply some examples before, of how the analysis has been done.

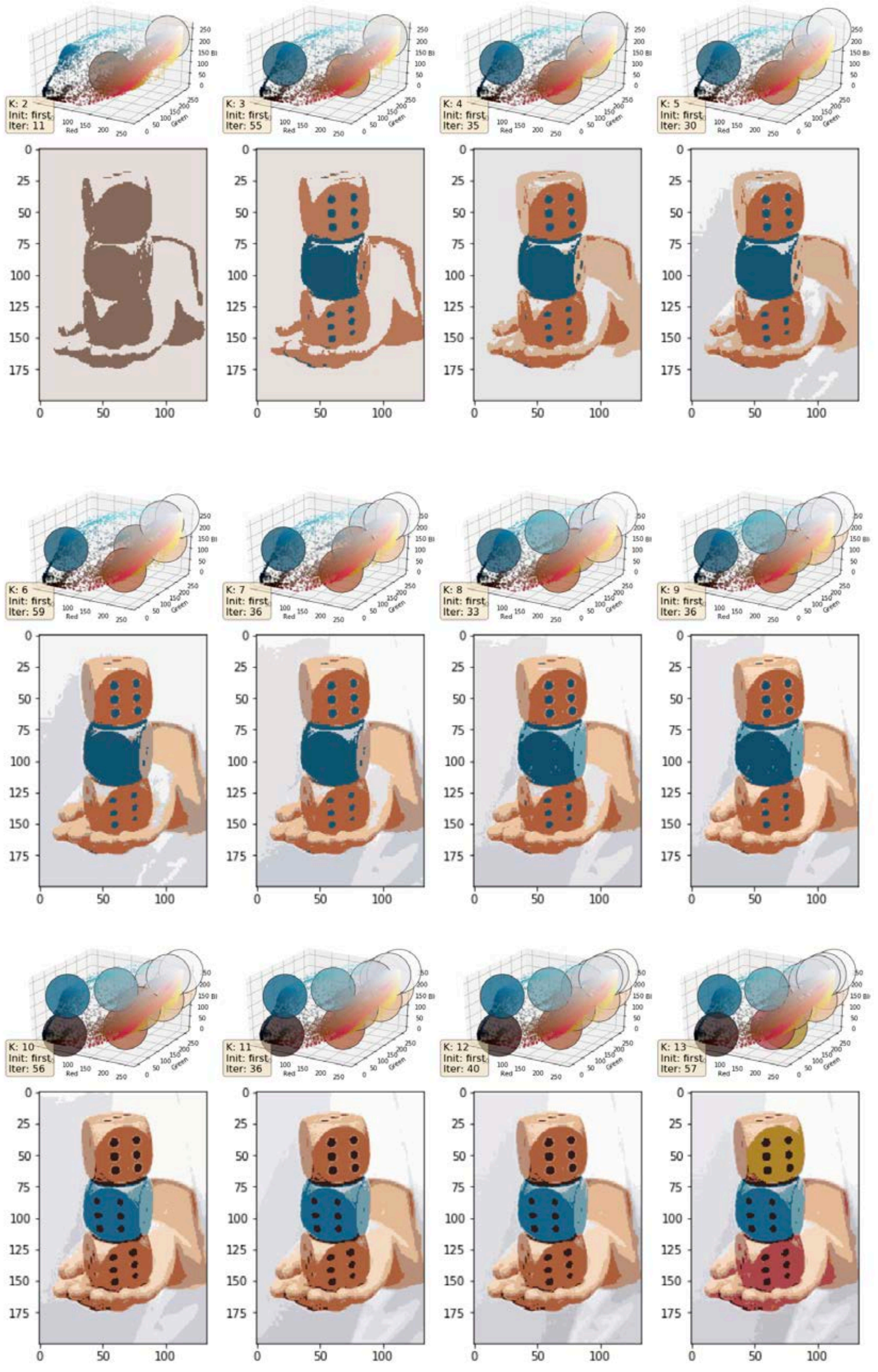
The first parameter we tested was the initialization of the centroids. For the complete analysis we only used the “first” initialization and RGB mode because random results were worse and differ between different executions of the program. In spite of this, we’ve done a smaller analysis. With “0187.png” we show the results of K=K_GT and the single_thr = 0,5; and with “0138.png” we show the results of the bestK and the same single_thr.

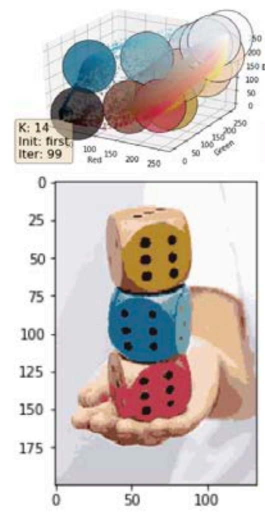
As we see in this example, the first initialization needs 45 iterations to converge and the random initialization needs 70. However, we can see that the difference between both images are light. With the same K value, the first initialization does not show the sky as well as the random one. That’s because first, select K centroids as the first different pixels of the image and random is completely random.

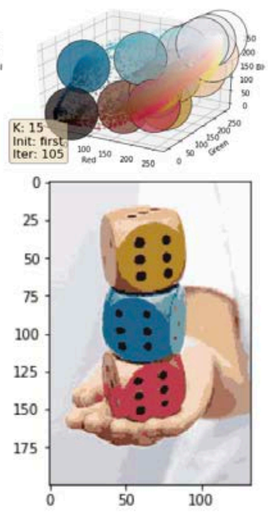


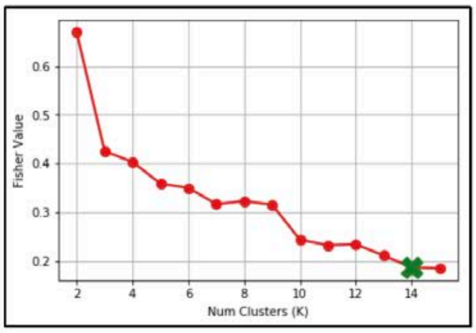
To explain the best_K execution, we show the following details:

<u>“First” Initialization</u>

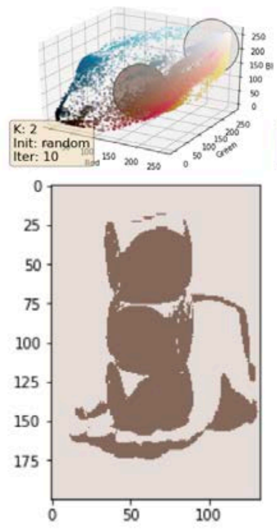


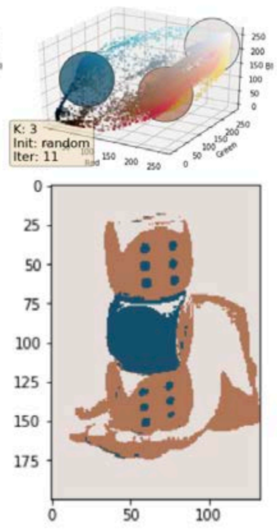


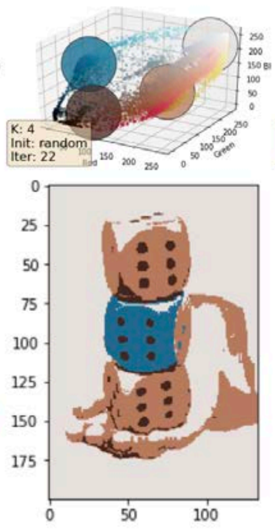


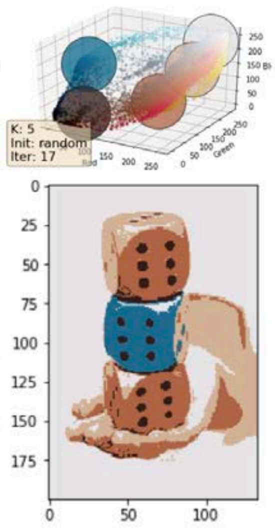


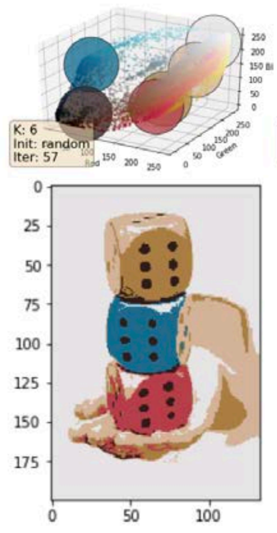
“Random” Initialization

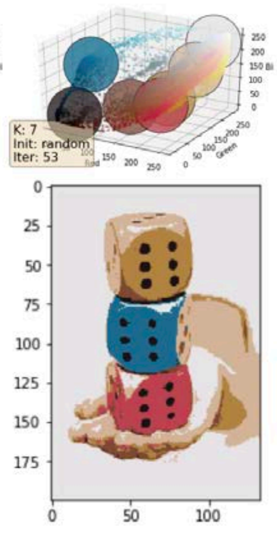


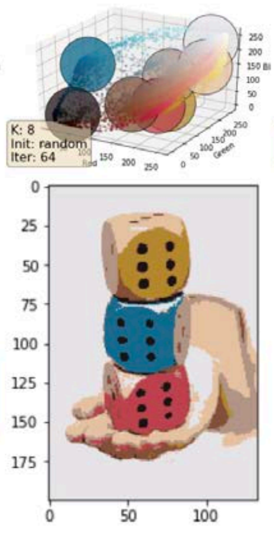


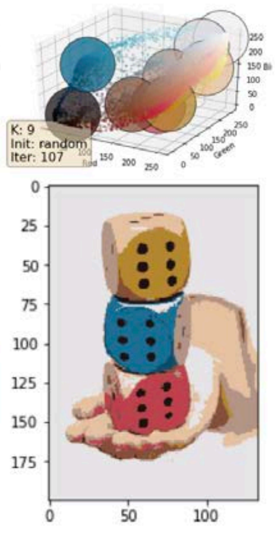


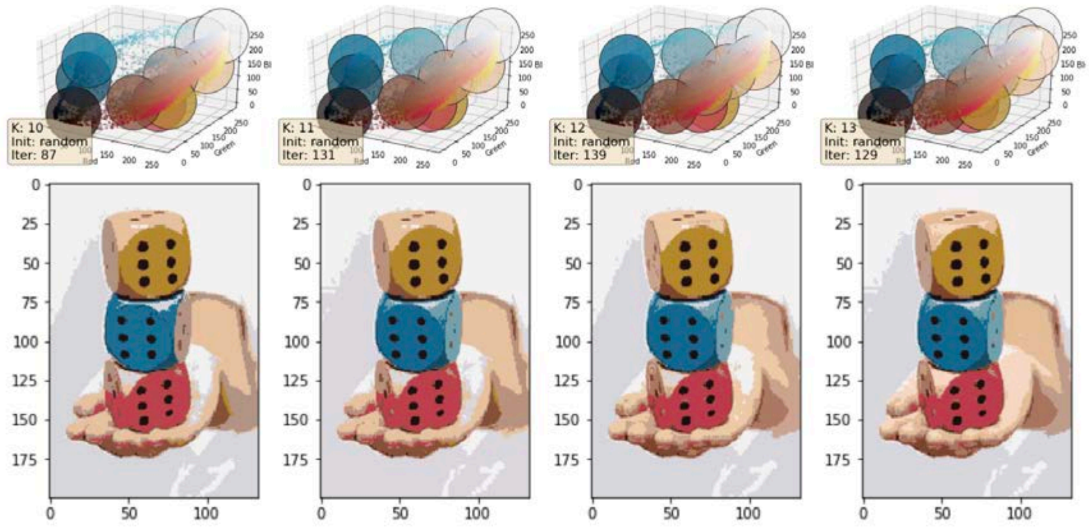


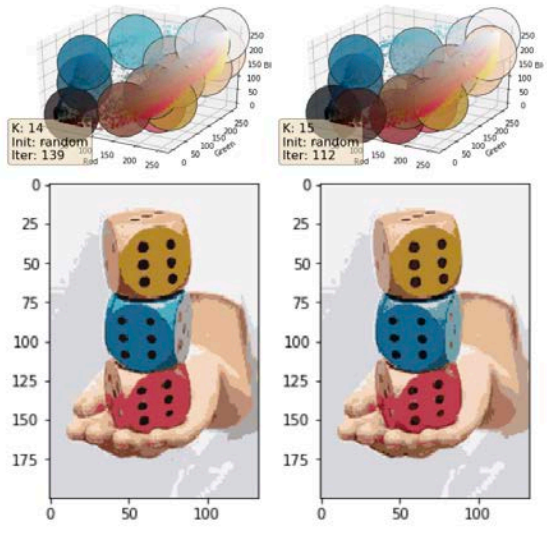


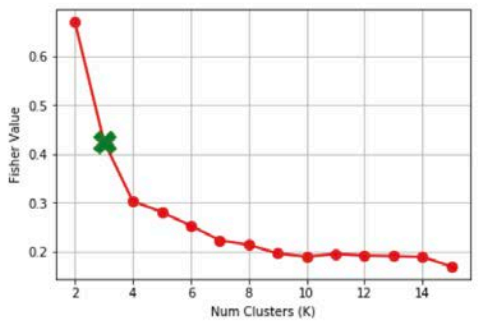








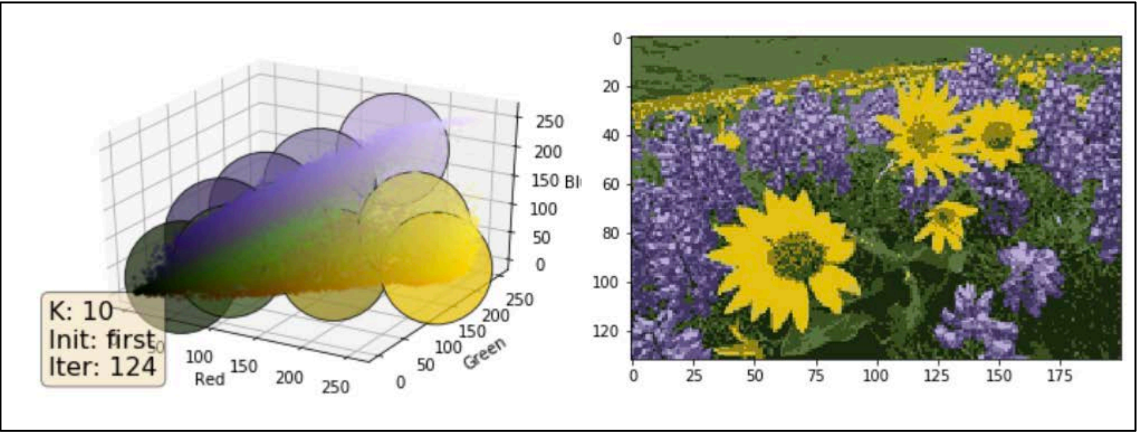




Here we can see that with “first” initialization we got that the best K was 13, although we got with “random” a K equal to 3. If we see the details on the evolution of the images given, we observe a great difference. For instance, in “random”, the yellow dice appears with K=6 instead of K=13 of “first”. This is because of the centroids initialization as we explained in the previous example. So, we conclude that in this case, “random” is more accurate and better than “first”.

To test which K is the best, we tested all K from 2 to 15, and with Fisher discriminant we determinate which one would be the best.
Another details that is important to mention is the difference of the new centroids created. While in “first” the new centroids when changing K value were set on white colour, “random” are set randomly. And this is very useful, when your image has a lot of pixels representing the same or similar colour.

Now, we are going to explain the results of the execution of the program with different “single_thr” (0,25 and 0,75), using the K=K_GT. To do this and show the results, we chose “0088.png” image.



This is the result of the execution. We iterated 124 until it converged for K=10. The image next to the plot shows the image created and analysed.

For “single_thr” = 0,25 we got:

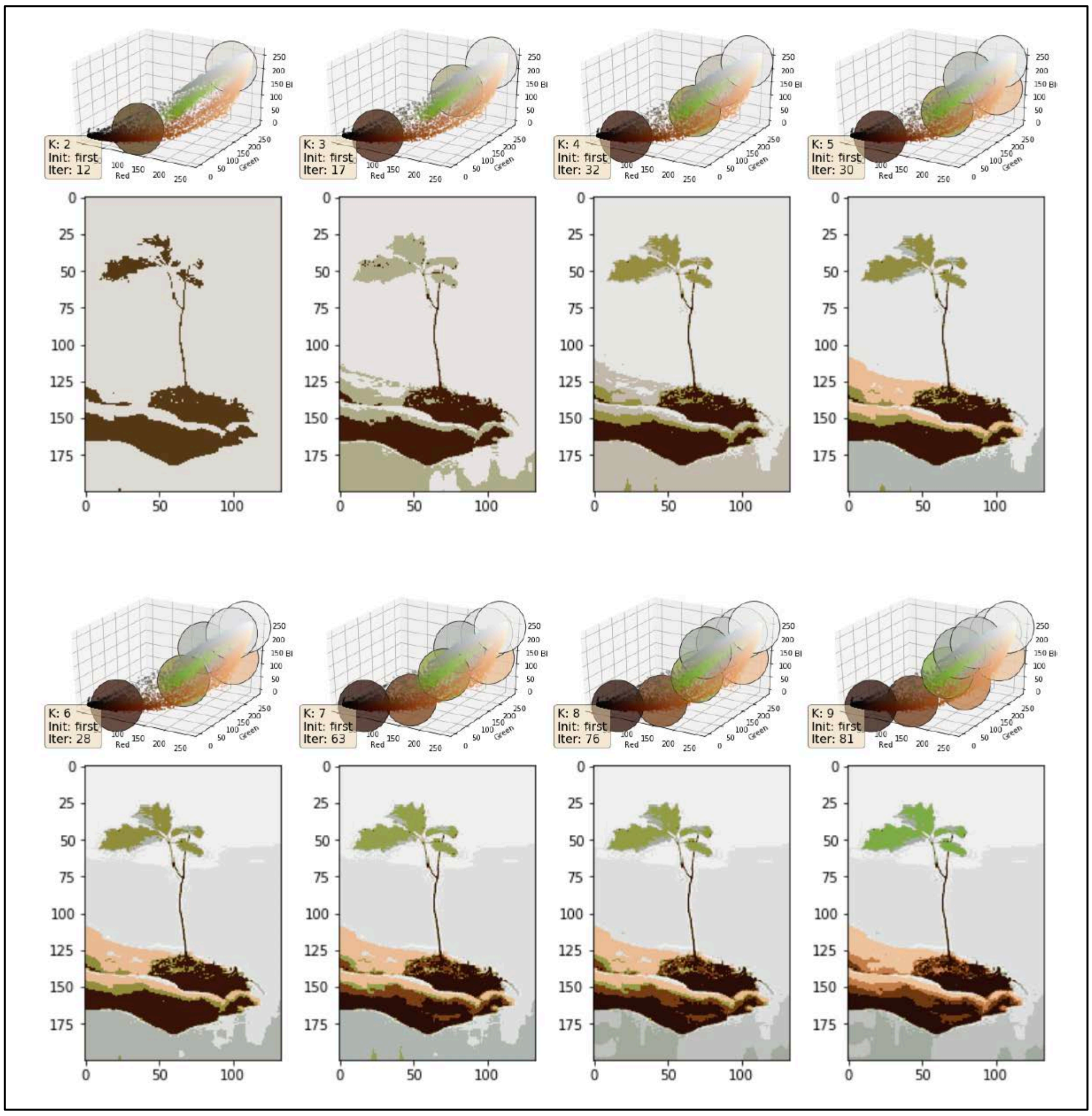
Image Name	K-GT	Colour List	K-Best_K	Colour List	% Similarity
0088.png	10	['Black', 'BlackGreen', 'BluePurple', 'Brown', 'Green', 'OrangeYellow', 'Purple', 'PurpleWhite', 'White', 'Yellow']	3	['Purple', 'Green', 'Yellow']	100%

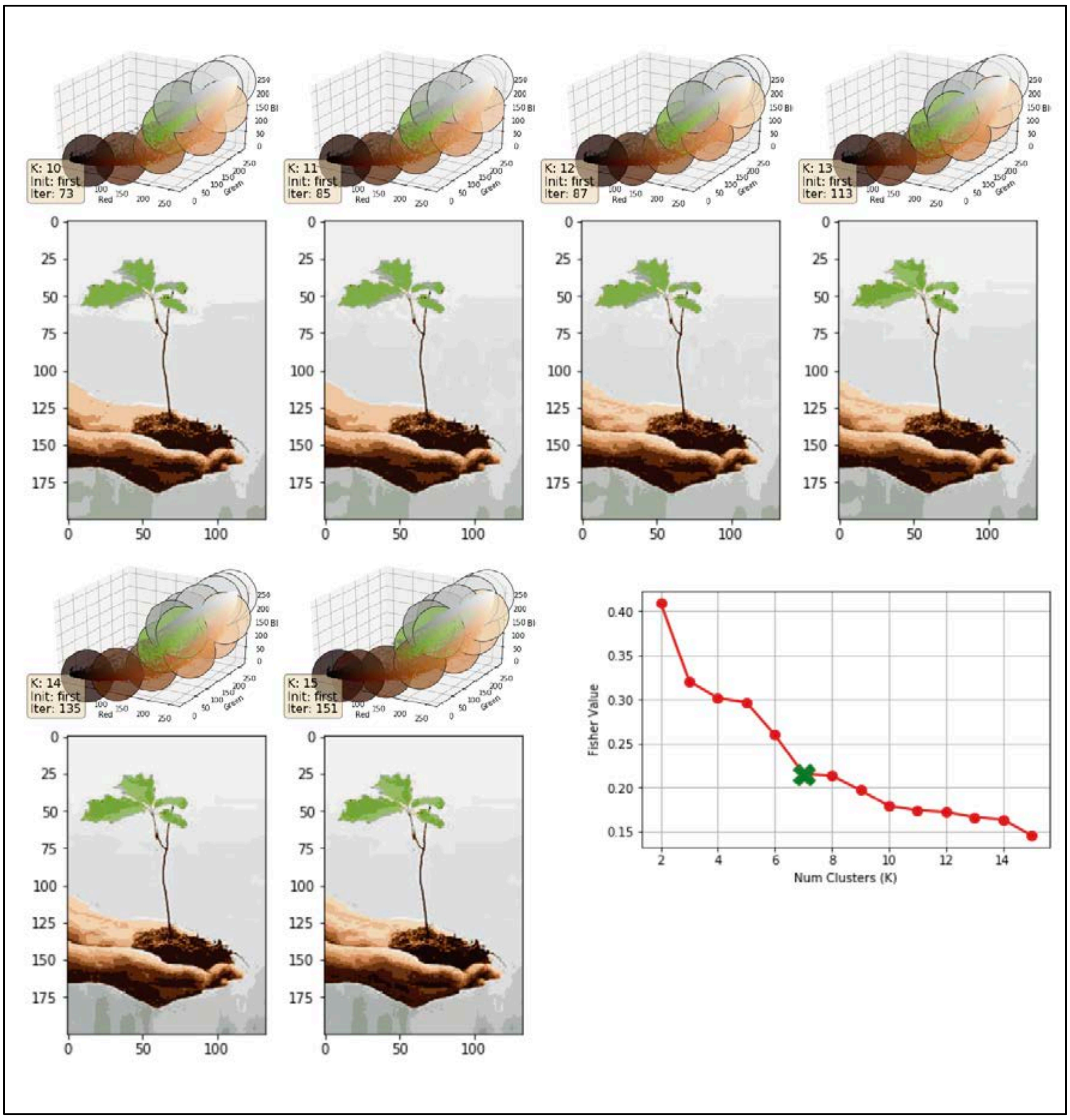
And for “single_thr” = 0,75 we got:

Image Name	K-GT	Colour List	K-Best_K	Colour List	% Similarity
0088.png	10	['Black', 'BlackGreen', 'BluePurple', 'Brown', 'Green', 'OrangeYellow', 'Purple', 'PurpleWhite', 'White', 'Yellow']	3	['BluePurple', 'Green', 'Yellow']	100%

We can see that the difference of the number of centroids (K) is considerable. Our bestK function select K=3 and depending of the “single_thr” parameter we got different result for labelling. For “0,25”, we got the same the results less “Purple” that with “0,75” is combined with “Blue”. This is because, with 0,75 the “exigence” is bigger and we have to combine labels to reach this value, while with 0,25 the “exigence” is lower and we don’t have to. Both executions marked as 100% of similarity because the colours detected by our program are included in GT solution.

As we have programmed and tested our implementation of BestK function and algorithm, we want to show the results of one image. With both different “single_thr” (0,25 and 0,75), using the K=BestK. To do this, we chose “0111.png” image.





This is the result for the execution of the K-Means for K=BestK. Our test has determinated that the BestK was 7, while K value of GT is 8. So, we conclude that our test is working almost fine. Now, we analyse the results of the colours detected.

For "single_thr" = 0,25 we got:

Image Name	K-GT	Colour List	K-Best_K	Colour List	% Similarity
0111.png	8	['Black', 'Brown', 'BrownWhite', 'Green', 'Grey', 'GreyWhite', 'Orange', 'White']	7	['Brown', 'Orange', 'White', 'Grey', 'Green']	100%

And for “single_thr” = 0,75 we got:

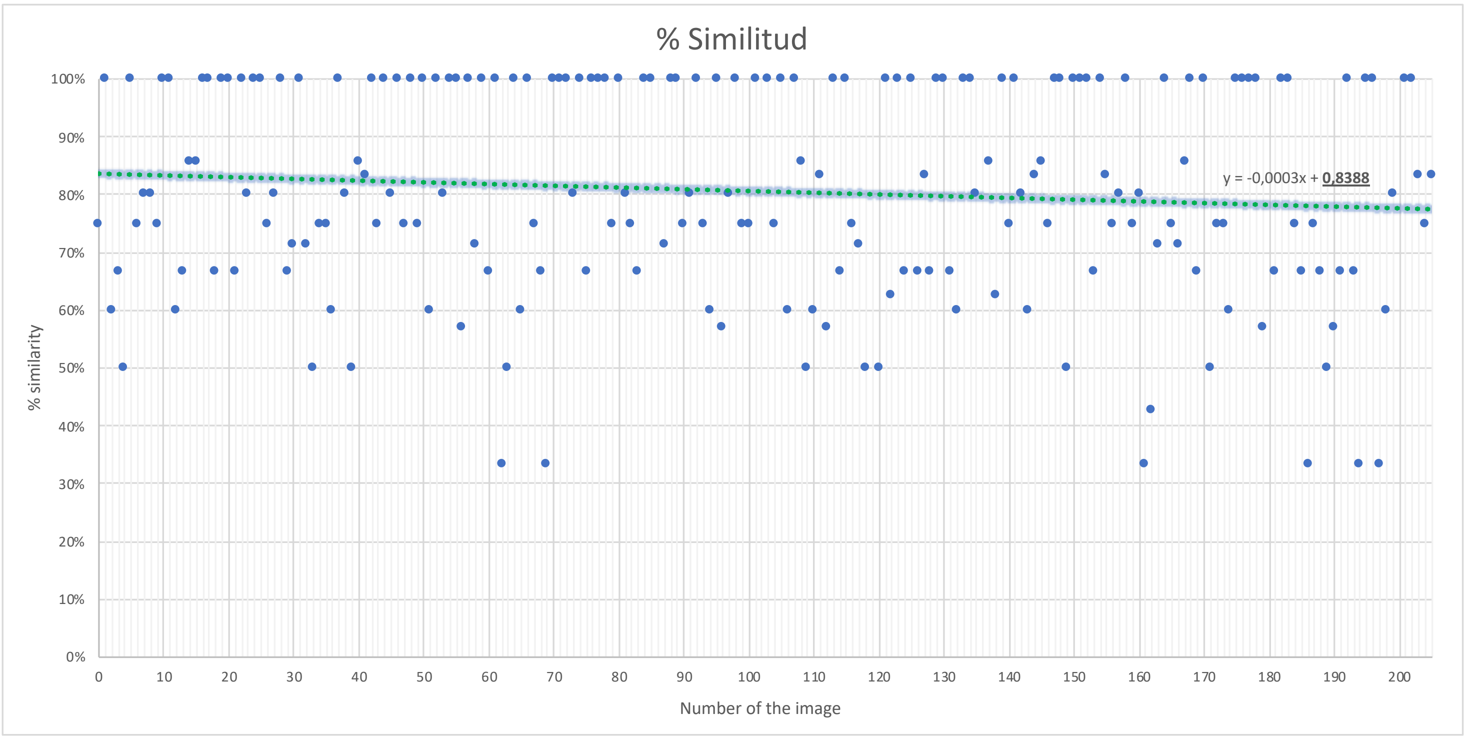
Image Name	K-GT	Colour List	K-Best_K	Colour List	% Similarity
0111.png	8	['Black', 'Brown', 'BrownWhite', 'Green', 'Grey', 'GreyWhite', 'Orange', 'White']	7	['Brown', 'Orange', 'White', 'Grey', 'Green', 'BrownRed']	83%

Here we can see a clear example of the difference between testing with “0,25” and “0,75”. As explained before, “0,25” is less “exigence” than “0,75” so we got the different results.

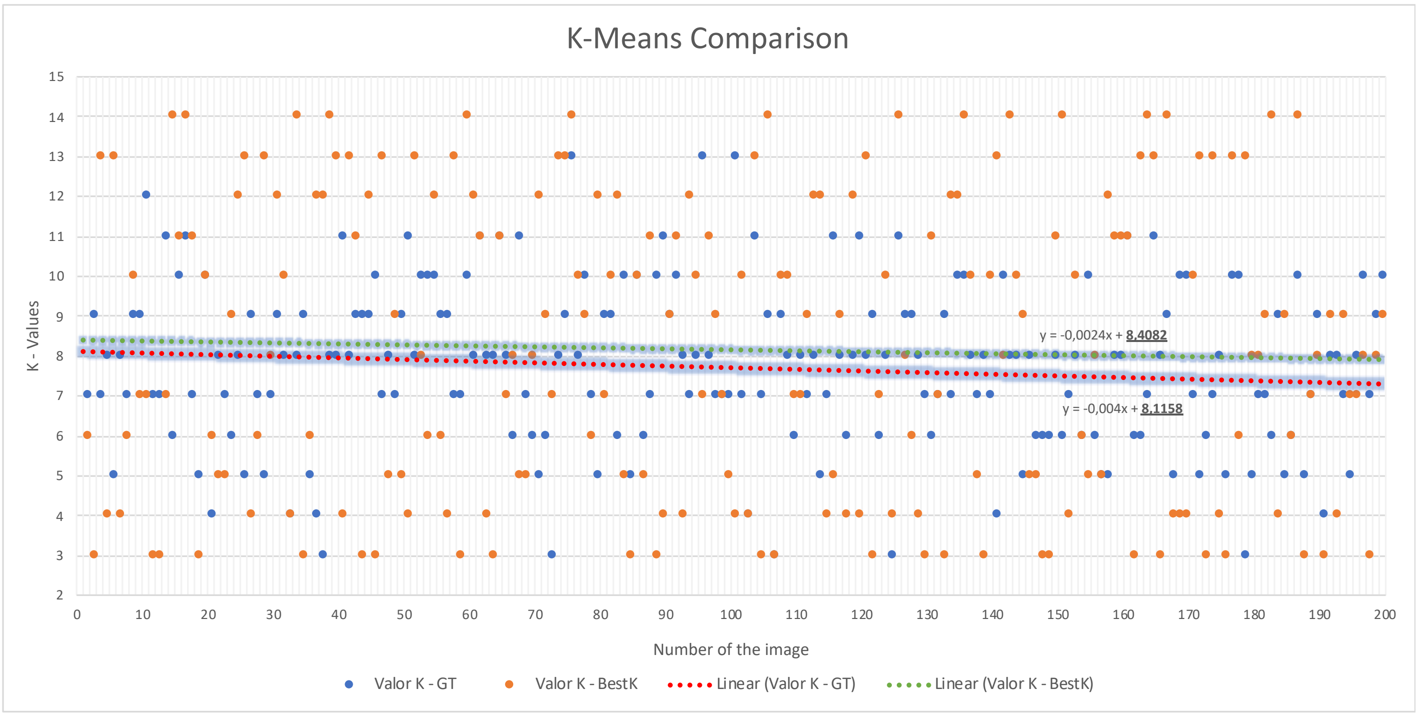
With “0,25” the colours detected were all on the GT solution so the similarity was 100%. In contrast, with “0,75” not all the colours detected were on the GT solution. It was the same solution less “Brown-Red”. Why? We know that combined labels are more likely to happen when “single-thr” is higher and there we think that the part of the hand that is brown and darker has been detected as “Brown-Red”. For sure, if we reduce this value, this combined labelled colour will disappear.

After some deep analysis of concrete images, now we show the results of our complete analysis. In this analysis, we ran our programme twice. One with “single-th” equal to “0,25”, and the other equal to “0,75”. All executions have been done using our implementation of BestK. To show the results, we decided to make two graphics. The first one shows the K-values that bestK algorithm ran in comparison with Ground-Truth ones, and the second graphic shows the similarity we got on this execution.

TEST 1: Single-thr = 0.75

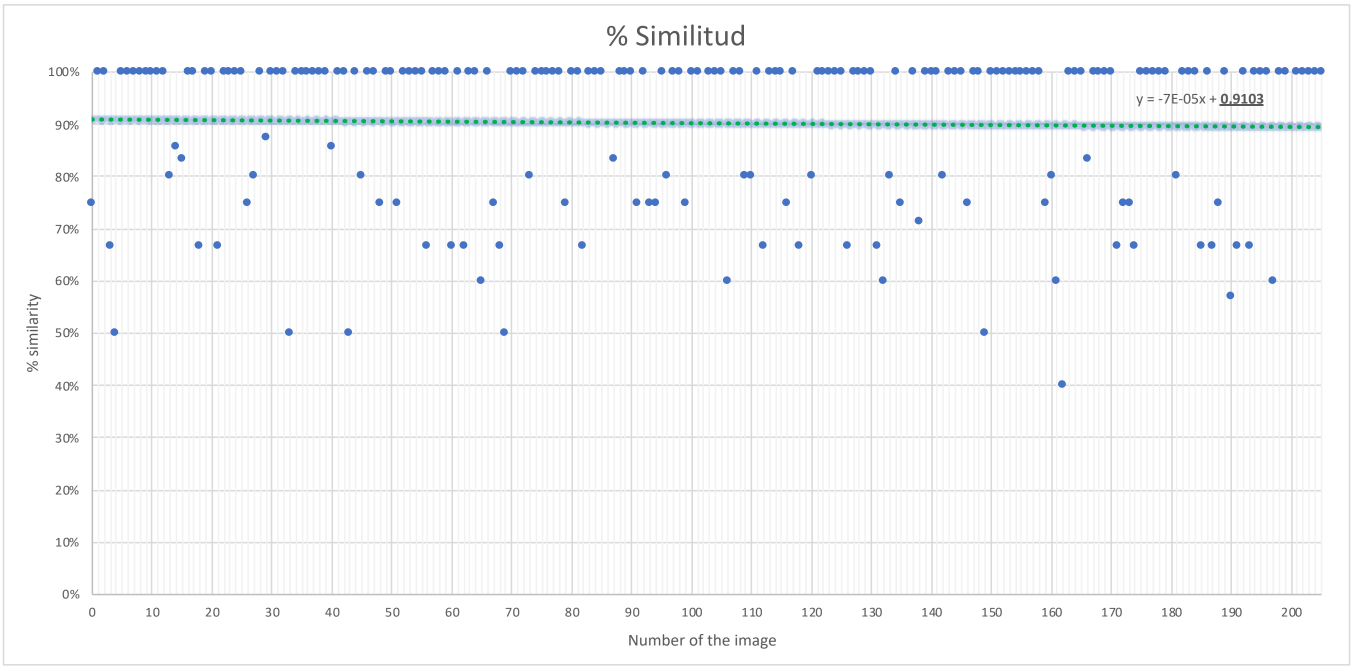


It is difficult to represent all the information we got it. We chose this way. In this graphic we can see as points the percentage of similarity got on the execution. We can see some different points near 30%, but the vast majority is 100%. This can be images with many different colours that our algorithm does not detect very well. The average efficiency is 83,88%. So, we have more than an 80% percent of success when we compare it to the Ground-Truth solution.



In this other graphic is very difficult to see which K is used in each case. Our aim has been testing the difference between the K selected with our algorithm and the K of the Ground Truth. Blue points are K of GT, orange ours. Even orange ones are more spread, and had very high values, the average is very similar. Both are between 8,1-8,4. So the medium K used was 8. For all the tests, the K have been the same, because we only have modified the "single-th" value, so we got the same results in the next test for the analysis of K.

TEST 2: Single-thr = 0.25



As we said before, the way to represent this information is difficult, but this second run of the execution with "0,25" as the value for "single-thr" gives us these results. In this graphic, we can see as points the percentage of similarity got on the execution. In this case, as opposed to the previous execution, there are no points near 30%, only one on 40% and the rest are on 50% and above. With this value, the similarity is better than the previous case because we reduced the value that restrict the number that determinates which colours are or not in the image. Now, the efficiency is increased by 8,52%, and the new value for similarity is 91,03%. So, we have more than an 90% percent of success in comparison with the Ground-Truth, and we "delete" those cases that were below 50% of success.

Conclusions

After finishing all the project, we can say we are satisfied with the work we have been working on. We've invested a lot of hours in total, both of us working and making tries to get it. We have been working on the faculty, and also in our homes, individually.

Although programming all the algorithms was quite long because we had to test that everything was working properly, making the analysis was quite long but very useful to get everything understood and be able to analyse correctly what we have been doing.

We used a Git-Hub repository to share our content and have all the documents and code modifications organised.

This project has given us a vision of what AI means, how to implement an algorithm to learn how to group elements in different clusters and apply them to a specific problem like colours, and to see what's behind.