PRACTICE 2: LABELLING

Subject: Artificial Intelligence

Gerard Josep Guarin Velez - 1605947
Max Galindo Poza - 1537514
Josías Micael Cueva Castro – 1603392

1. Introduction

The main objective, which make enthusiastic this project, is develop an efficient and accurate search that could be implement in a hypothetical app. This seeker must label an image's group depending in their colour and which clothing type it is.

The principal algorithms that we did implement in this project, were those which let us implement labels depending in the situation and the data that we dispose at specific time.

At first, we started coding Kmeans algorithm, which give us the possibility to labelling by a predominant colour of an image. In this algorithm we code some functions as *init_centroids, distance, get_labels, get_centroids, converges, withinClassDistance, find_bestK* and *get_color*.

All these functions have de main objective of identify all the data (colours) of a given images. This algorithm goes from the initialize of the data that we receive as number of colours, number of images and the position of each pixel. And then we take care of the process of this data in order to determinate important points of the function that will make successful work of the algorithm. Finally, we have the final data that is an Kmeans instance with all the process information that the algorithm did, that must be all the colours that we find and his proportion of each one.

The other algorithm is KNN, a simple and elegant one, which needs training before try to predict a new image's group.
In this case we train the KNN with all those types of clothing that we can take in consideration during a specific search and the possible applications for a hypothetical app.

All the functions that fulfil, and make work, this algorithm are: *init_train, get_k_neighbours, get_class* and *predict.*
The types of classes are the same that in Kmeans: the initialization of the data, the process and, in this case, a function to predict a new image's block that the algorithm must label by the type of clothing (in this project we only consider dresses, shirts, FlipFlops, Jeans, Sandals, Shirts, Shorts, Socks and Handbags)
Once we execute a KNN we should first train it, and then we could give it only a image's group that KNN should process and return the clothing type of each image.

We must remark that the use of numpy is present in all the areas that we can optimize, because not only does it improve the size's code, but it also does in time. At the beginning of this project, we try not to use this library because we weren't related with it enough. But once we realize that the time optimization was stunning, it forces us to understand and implement it in our code.

2. Implemented method of analysis

2.1. Qualitative methods

Once we check that our Kmeans and KNN pass all the tests and our program has been programmed in a correct way, we proceed to code those functions for seeing if the results shows if it really works in real situation that the program could be use (always making references to an app search).

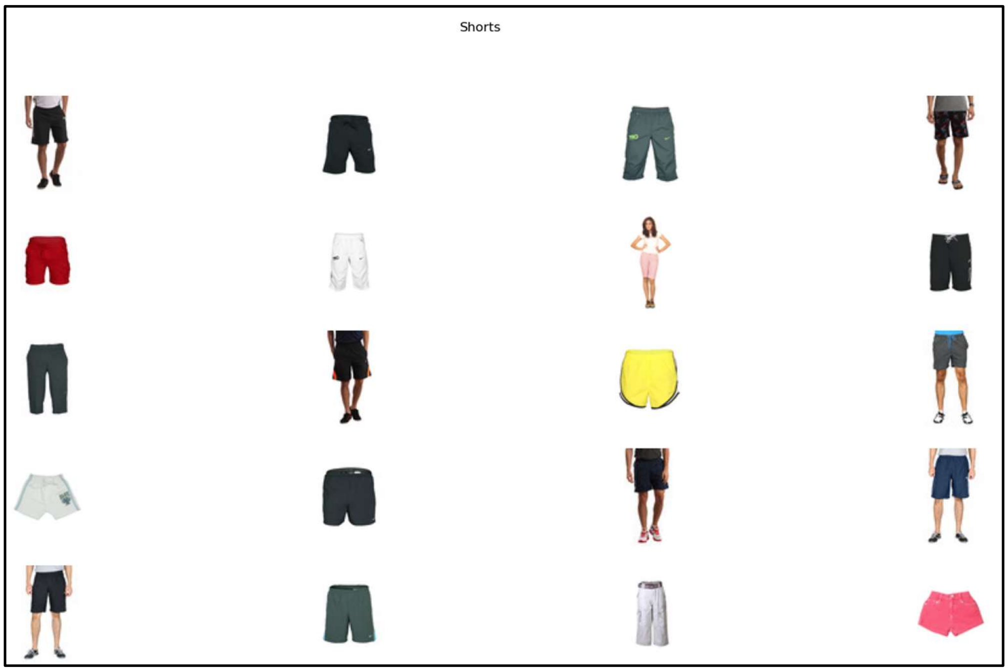
2.1.1. Retrieval by shape

As we said before, the KNN precises an image's group and their solution labels, in order that our algorithm could compiling enough information. Then we should give it a new group of images that should predict (by all the train that we did before) and returning all the images with the piece of clothing that might be.

First, we started initializing the KNN with all the *train_imgs* and *train_class_labels.* Once we ha done it, the program will receive a completely new group of images that the function *predict* must return which clothing type it is.

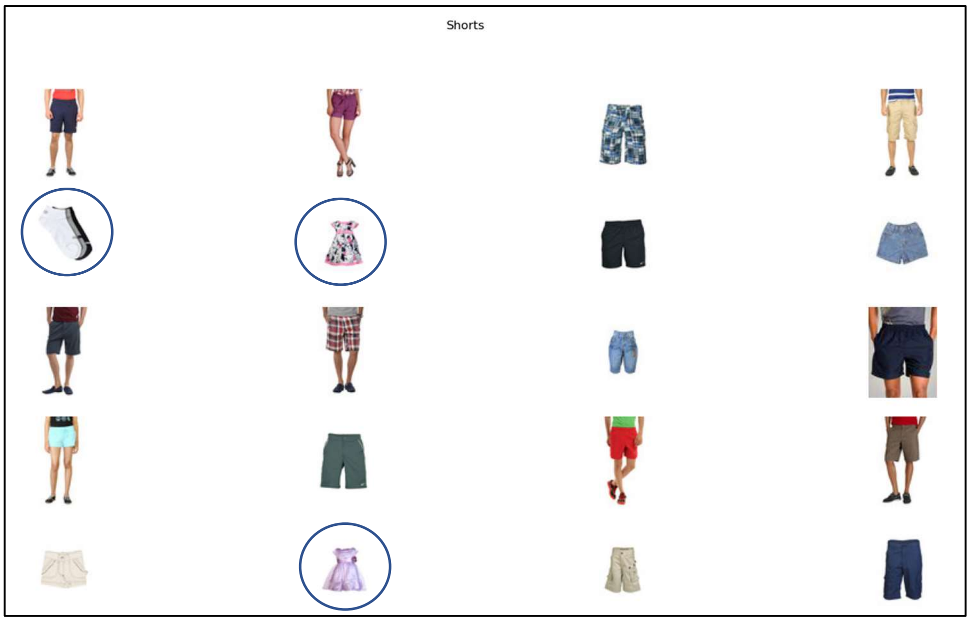
Since the point of coding view, the function receives all the images, predict result and an array of strings that contents our parameters of search (in our case we only put '*shorts*'). Then in the function we look for those elements that coincide with the elements that we put as parameters Finally, we only return the index where those images should be, due being recognize as that element that we look for.

For this prove, in the retrieval shape will select all those images that must be '*shorts'*, and this is the result:



[Result after execute retrieval_by_shape]

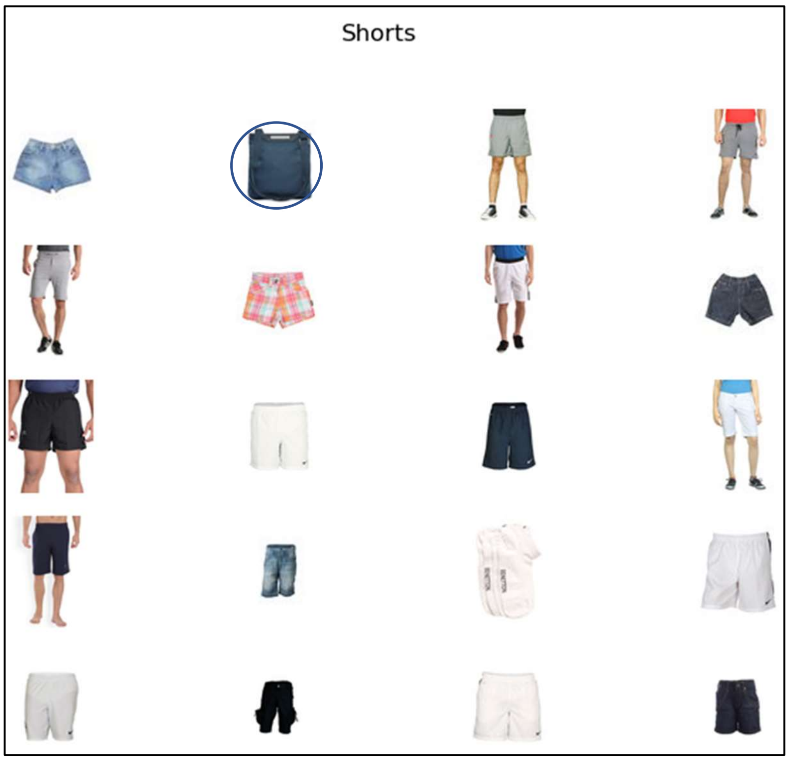
In the first 20 elements that the program visualize, we can apreciate that error range is 0, that makes looks that the algorithm works as we planned.
But let we see the next 20 following images.



[Result after execute retrieval_by_shape]

In this 20 images we can notice that our error range has increase, and we found different imatges that should not appear. For example, there is a pair of socks and 2 dresses that shouldn't be in this predict.

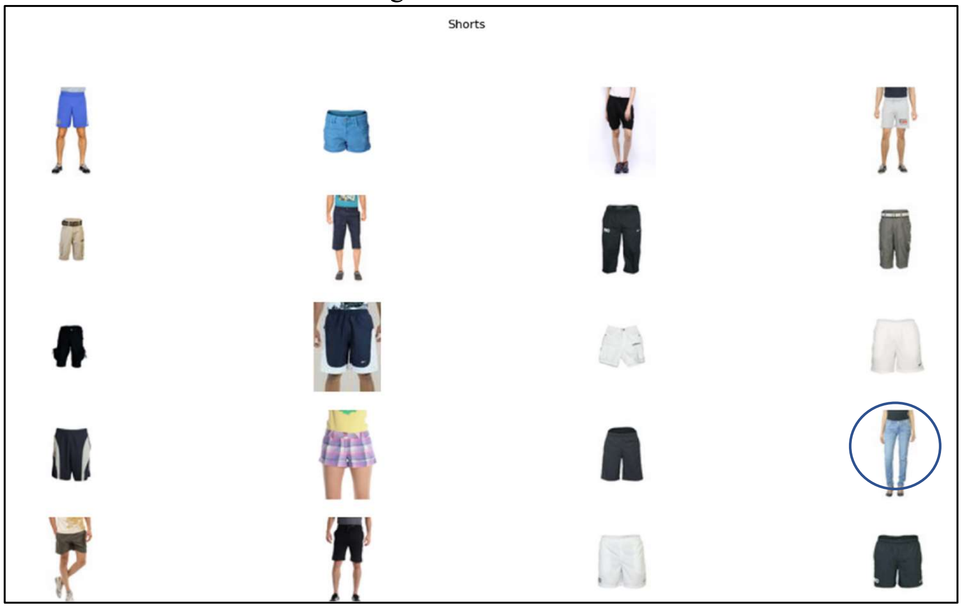
Now we continue with other 20 images:



[Result after execute retrieval_by_shape]

In this case a handbag has fool our KNN. However our error range does not increased as the previous case.

We continue with the last 20 images:



[Result after execute retrieval_by_shape]

In this last iteration we appreciate that only one error has appear and seems not to be strange as previous ones.

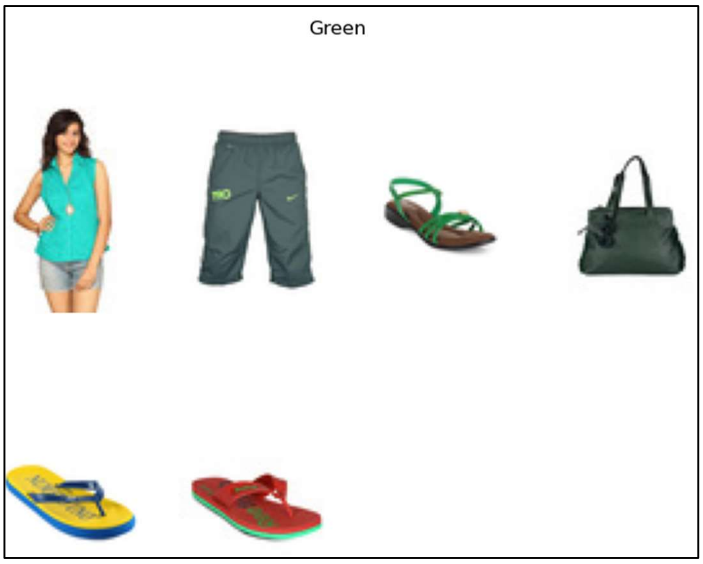
As conclusion, we can appreciate that our KNN doesn't work as accurent as should be. In instance, the occurency of our predict's results are pretty high, but we can improve as everyting in engineering field

2.1.2. Retrieval by colour

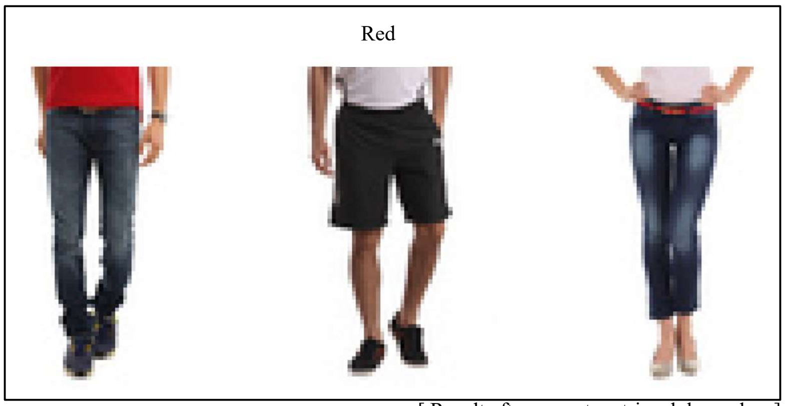
In this case our code just does as the retrieval by colour does, but in this case only receives values as the Kmeans instance, labels and a string (the colour that we want to do the retrieval).

The sequence of the orders are simple, we just take those indexes elements from array that are classified as the specific colour that we want, then we just visualize the index of those images.

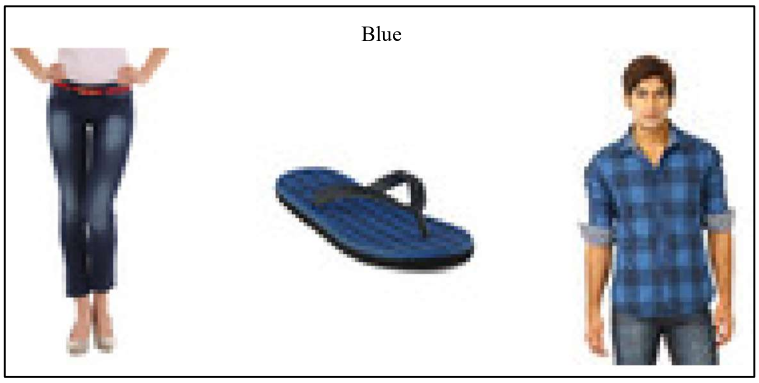
Here are some searches that our code does:



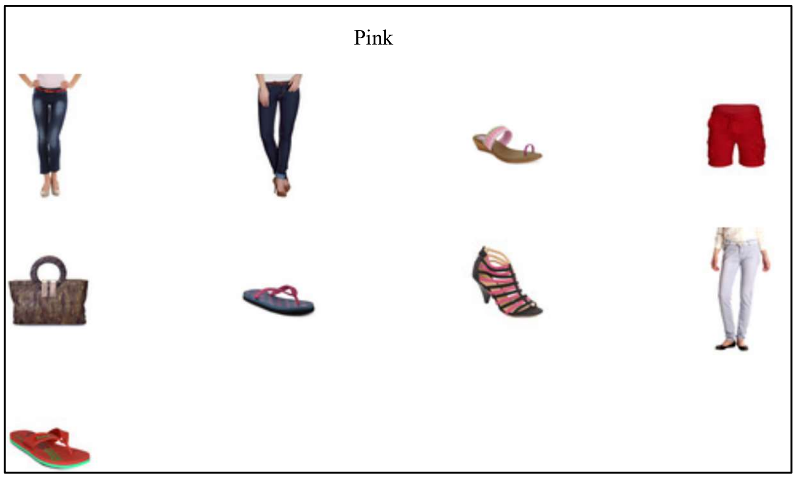
[Result after execute retrieval_by_colour]



[Result after execute retrieval_by_colour]



[Result after execute retrieval_by_colour]



[Result after execute retrieval_by_colour]

At this point we can see that there is a error range pretty small, in fact there is some colour as black and white that are present in all the images, due the background of the images and the difference between other colours.

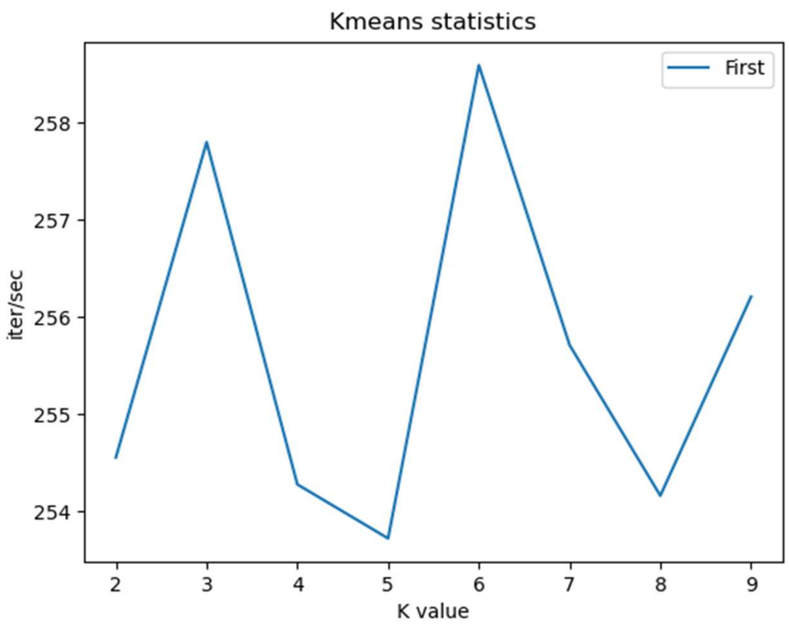
2.2. Quantitative methods

In this part we just use those functions that will show us how the Kmeans and KNN does work in an uncharted field.
In this case we just programmed *get shape accuracy* and *Kmean Statics.*

2.2.1. Kmean statistics

This function just receives a Kmeans instance, an image's group and the Kmax that we want to use.
The way that it works is quite simple, we just star a chronometer when we execute the *fit* and *withinClassIntance* then we just stop the chronometer, and we calculate the proportion of iterations divided by execution time.

Here we can see how it works:



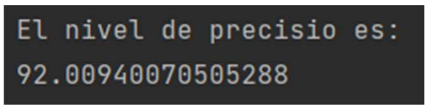
[Result after execute kmean statistics]

We can appreciate that depending in which Kmax we give, our time in iteration can variate in time, which means also in compute time.
This graphic can change not only by the Kmax, in fact, using a different centroid's initialization also change dramatically.

2.2.2. Shape Accuracy

In this part we just compare the predict accuracy in order to determinate if it really works properly as we planned at the beginning of this project.

For instance, the function receives the KNN predict and the labels solution that KNN should have process by a image group. Then the function just counts how many mistakes has the algorithm did. Finally, we just print the accuracy:



```
El nivel de precisio es:
92.00940070505288
```

In our case we just not only find that the precision is quite high, but also it is a prove that in *labelling by shape* reflects the statistics functions.

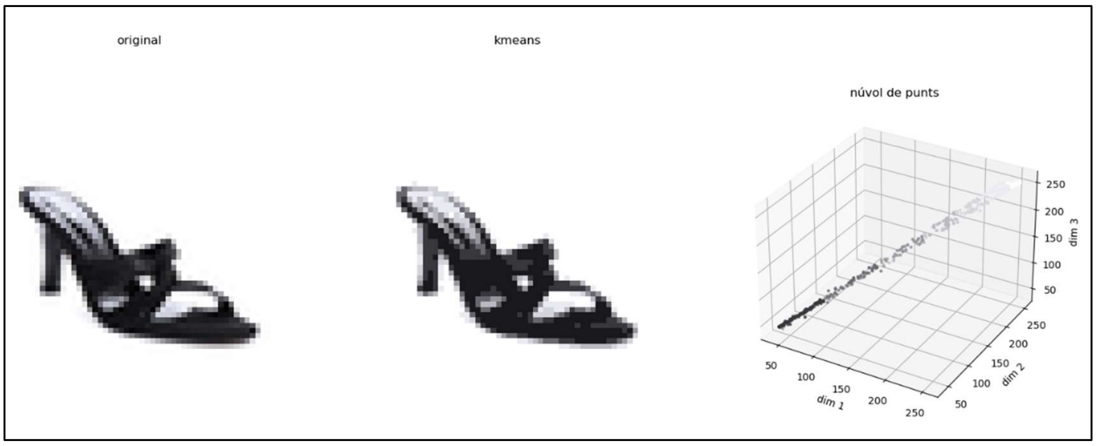
3. IMPROVEMENTS ON KMEANS AND KNN

3.1. KMEANS INITIALIZATIONS

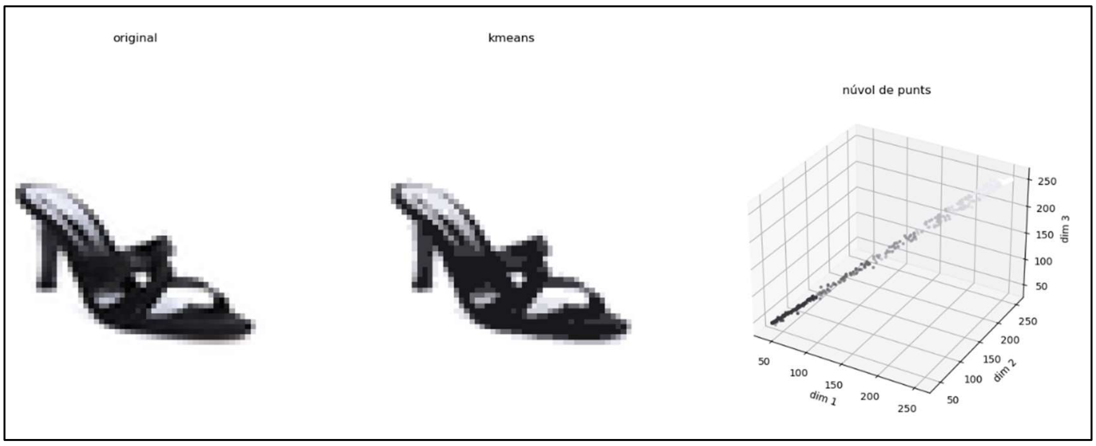
In this key modification we just change in how we recap information from the image that we receive in the time that the constructor is working. Moreover, we added two different initialization mode: random and costume.
In random mode we just select random and non-repeated points of the images and then cataloguing as new centroids that will be as reference in the following instructions that the algorithm will do.
Costume mode has been programmed in a way that takes the diagonal of the images as centroids. In this mode is curious how easy was the numpy function to take a diagonal from a 3d dimension array.

The test that we implement in this improvement is executing *visualize_kmeans,* that shows how Kmean draw an image by how the initialization centroid mode was.

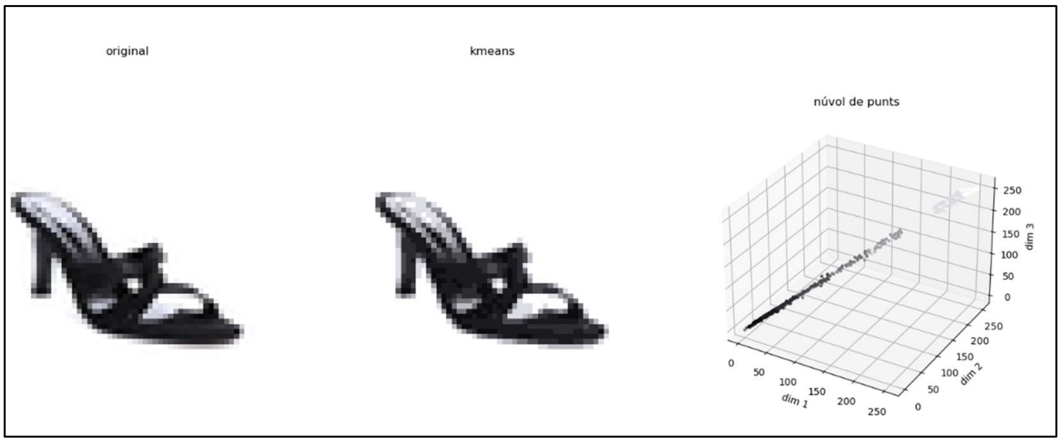
We will start with the first:



[Result after execute kmean visualitzation in first mode]

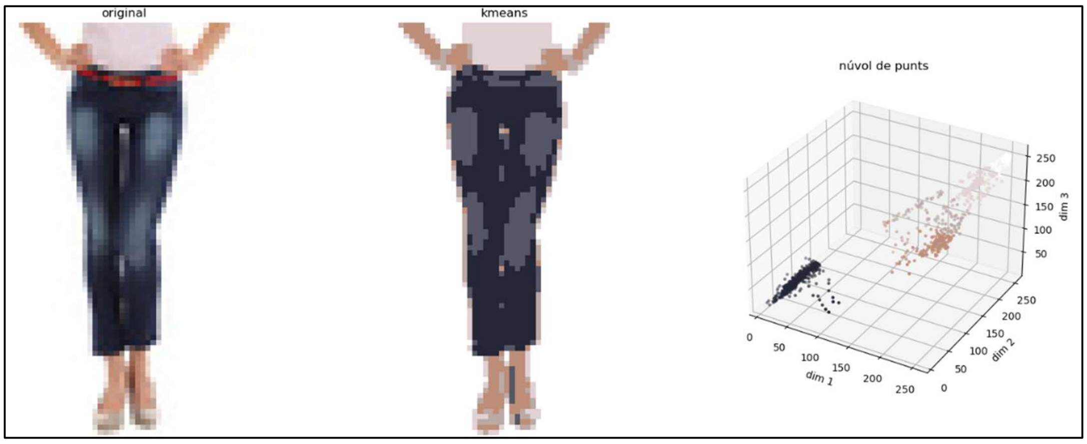


[Result after execute kmean visualitzation in random mode]

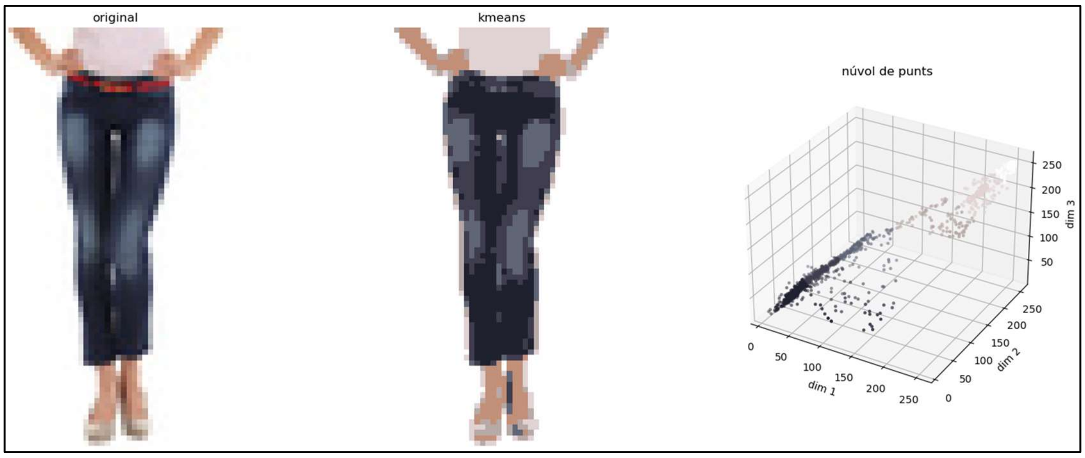


[Result after execute kmean visualitzation in custom mode]

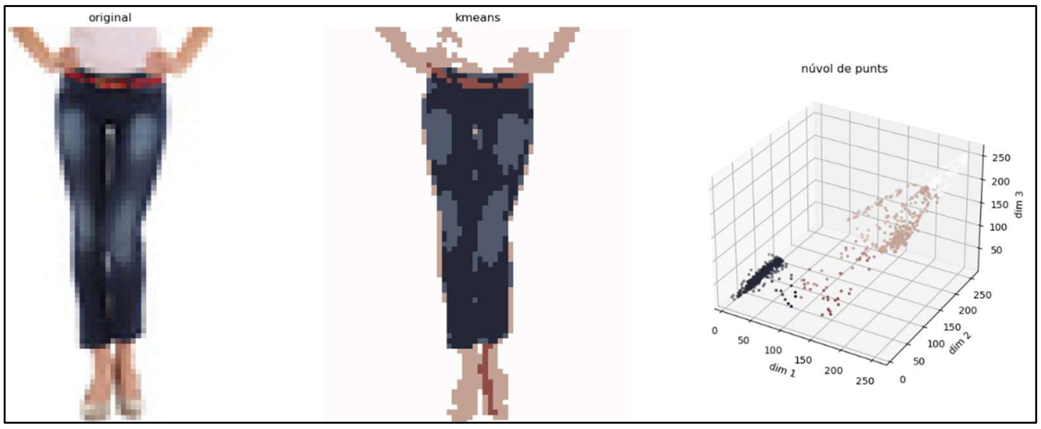
In this first image the difference can be barely appreciate, we will prove in other image that the difference is huge.



[Result after execute kmean visualitzation in first mode]

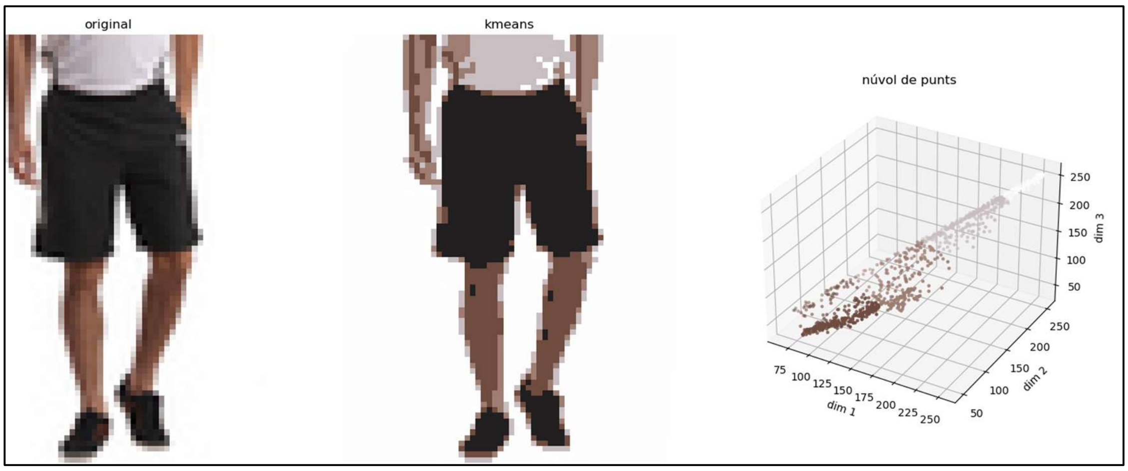


[Result after execute kmean visualitzation in random mode]

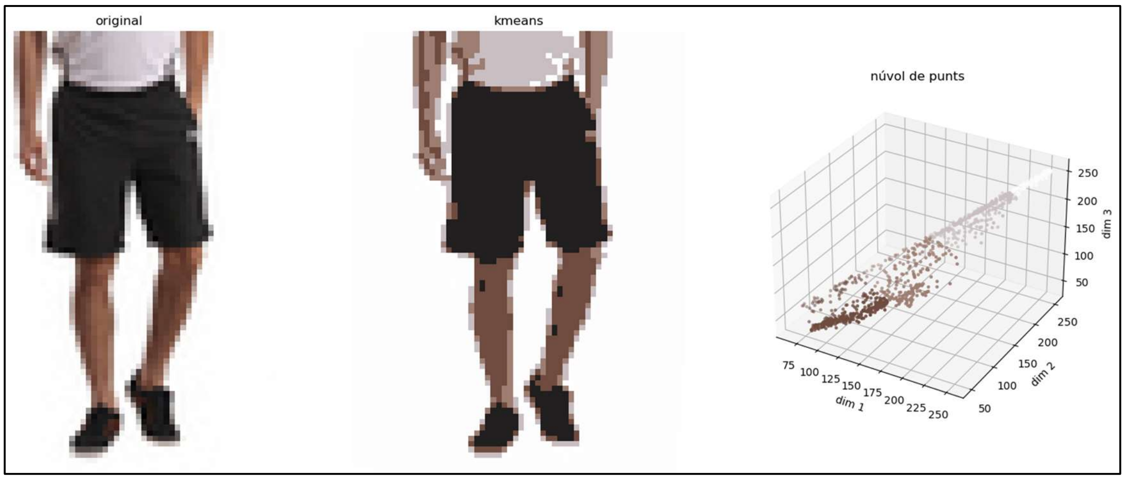


[Result after execute kmean visualitzation in custom mode]

We can appreciate that the difference between first mode and random are pretty small, the only appreciable difference between them is the cloud points. In random case we find out that the colours homogeny is higher than the first, and the first one has the representation of a brown, while in the second not.
And if we compare the two previous one with the costume the improvement is enormous. The diagonal can distinct red belt, while the other does not. However, seems that it losses homogeny in the cloud point, which reflects in the representation with a poor definition of the image.
Moreover, we can relate the poor definition of the image with the '*llindar'*, but this point will be seen in the following improvement. However, we let some other examples:



[Result after execute kmean visualitzation in first mode]



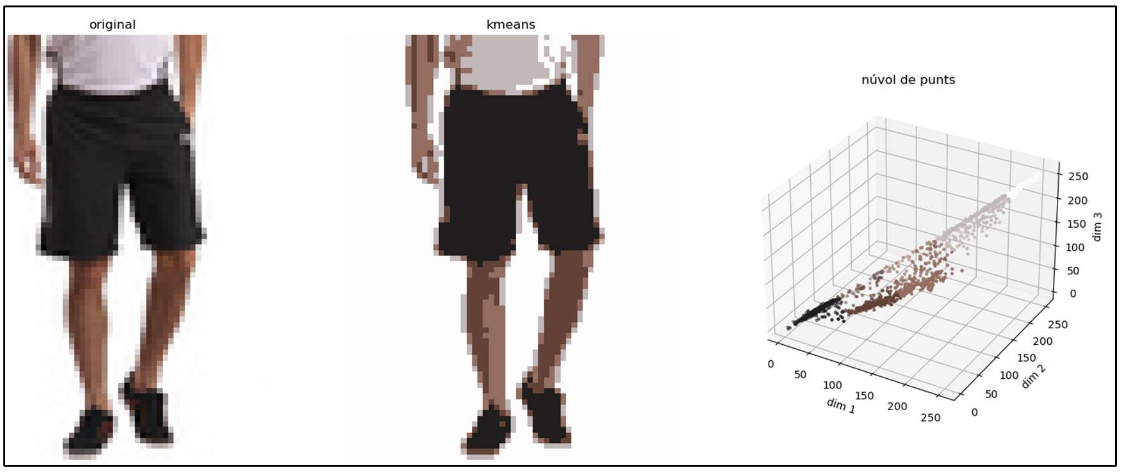
[Result after execute kmean visualitzation in random mode]

3.2. FIND BEST K

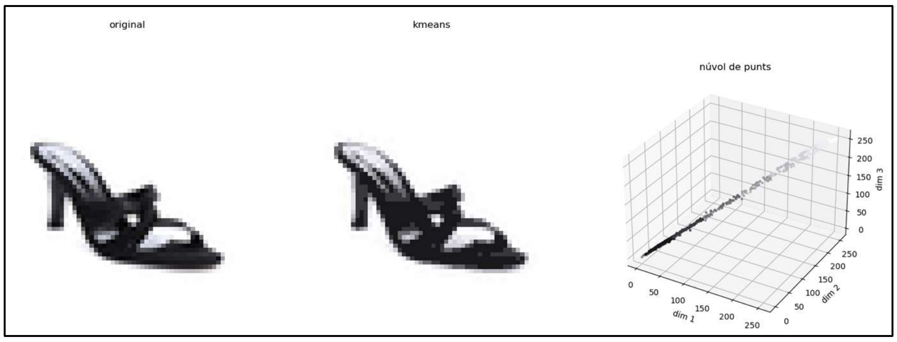
During this function we change the '*llindar*' value, this value sets when we will accept a centroid depending in the distance applicated in the function *withinClassDistance* and the grade of improvement. So, if me modified this value we set a different criteria in the moment to decide to execute again the process or we accept the result that we obtain, otherwise we don't improve and we are only wasting time in useless iterations.

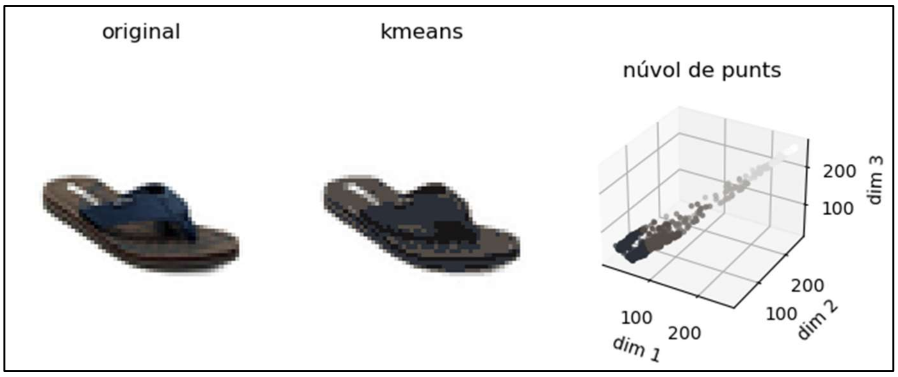
In our test we prove differents '*llindars'* and we obtain these results:

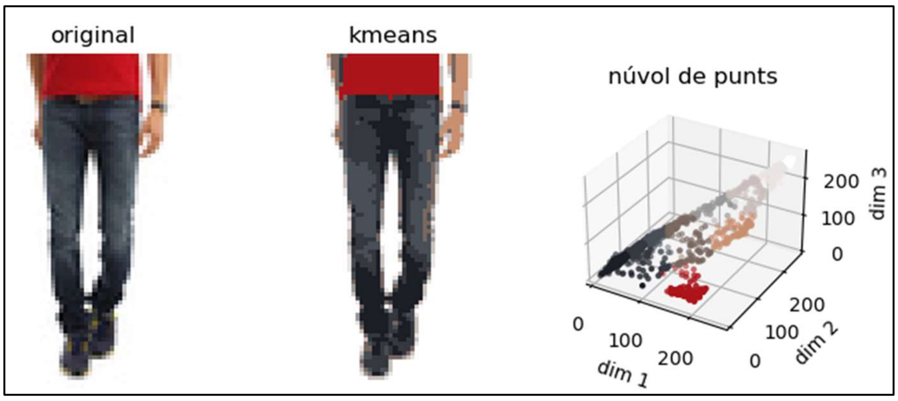
- '*Llindar'* = 1

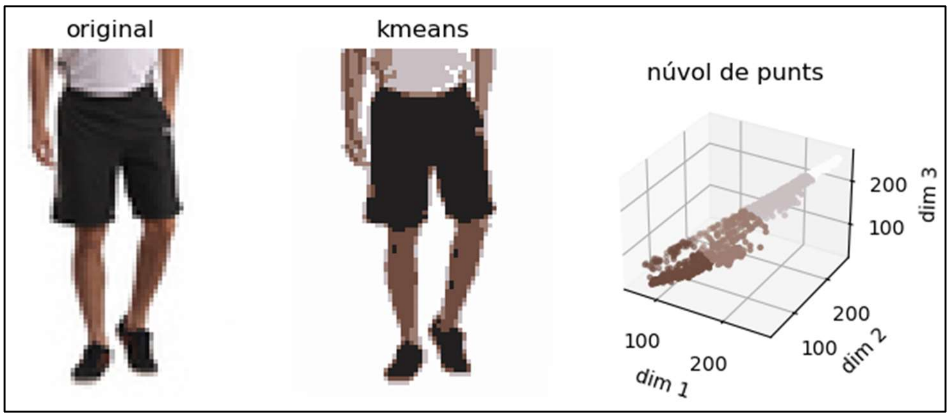


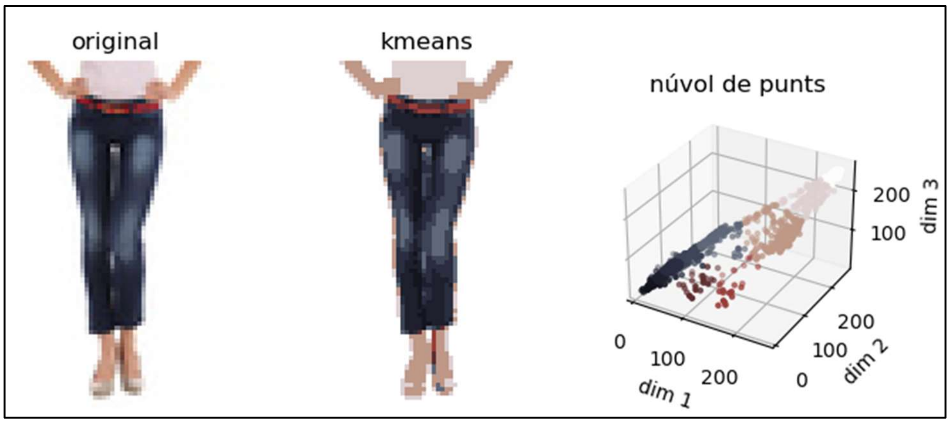
[Result after execute kmean visualitzation in custom mode]




original
kmeans
núvol de punts
dim 1
dim 2
dim 3
100
200

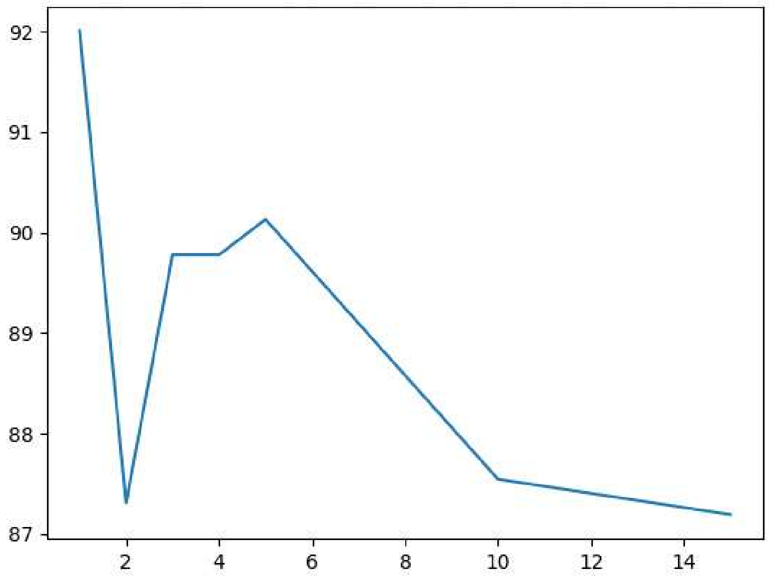

original
kmeans
núvol de punts
dim 1
dim 2
dim 3
0
100
200


original
kmeans
núvol de punts
dim 1
dim 2
dim 3
100
200



Using a '*llindar'* the last picture we can really appreciate that the definition of the image has improved, and the cloud point has more homogenic than in the first improvement.

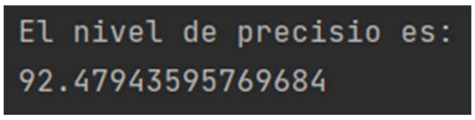
In the following graphic we can observe how the accuracy changes depending on the Kmax labels.



3.3 FEATURES FOR KNN

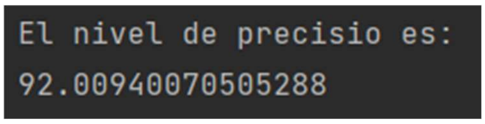
In this part we implement the idea to pass the RGB to Grey-Scale, in this form we can optimize and reduce the excess of information of RGB and making the process of the data more efficient than before.

We use some function and formula that let us change the scalar of colours into a grey scale. When we execute the function KNN using the grey-scale option, and then we start the shape accuracy we received this result:



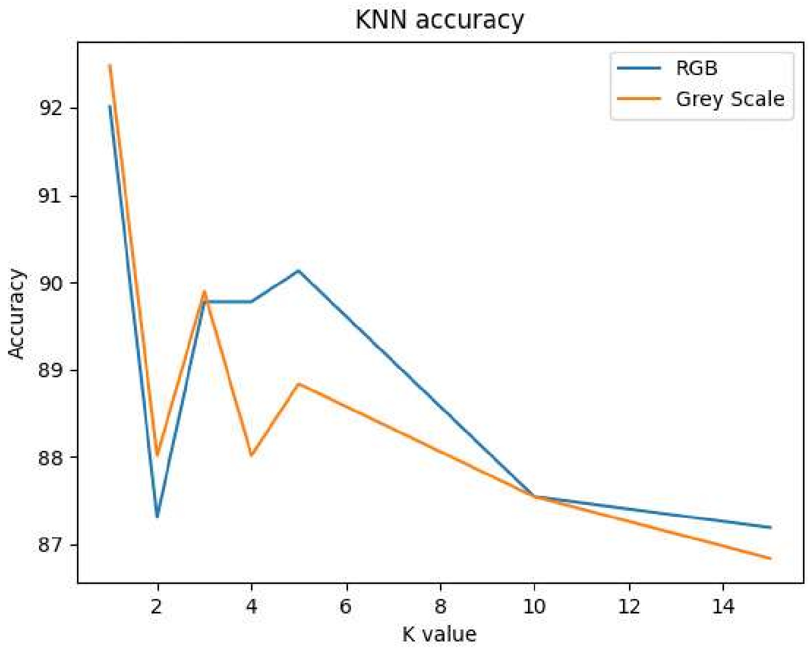
```
El nivel de precisio es:
92.47943595769684
```

This result has a small improvement when we compare it with non grey-scale, but it improves by little:



```
El nivel de precisio es:
92.00940070505288
```

In this graphic we can see how does improve the Grey Scale over the RBG when we change the K-labels:



4. GLOBAL CONCLUSSION

To sum up we can announce that we learn how useful it is the numpy. This library has improved in time and space our code. However, our poorest knowledge about that has made difficult to understand the practice at the first sessions, by this moment, we can affirm that we have a plenty control of it.
Moreover, we can understand and synthetize all the theory seen in class and use it in a practical situation that can be in our future jobs, for instance, most of the information that have been given should have been more explained.

In our experience we found this practice stunning because their uses in real projects can be useful, not only in labelling clothing, but also can be implemented in financial fields, educative, etc.
But the most important is that we really understand how our daily technologies works and how we should design a new version of them.

Since of our point of view we can affirm that we learn a lot with this practice. However, we think that the teachers should have present more initiative at the moment of the presentations of each practice. Also the documentation can be ambiguous and confusing sometimes, moreover, we were surprise how the teachers were available to give us advices of how we should go on with the practice.