REPORT LAB 4: Feature detection, Image descriptors and BoVW

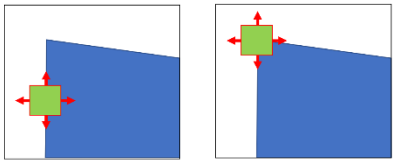
INTRODUCTION

In computer vision when we are working with images, we actually try to study different aspects to find out everything about an image. For example, we work with feature detection, image descriptors and Bag of Visual Words (BoVW). We use feature detection for checking the important features of a frame. Those features can be: edges, corners, ridges, and blobs. Moreover, we try to define some characteristics of the image such as: shape, color, texture or motion. Those are visual features that are considered while using Image descriptors. Last, BoVW participates in the image classification. It represents an image as a set of features consisting of key points (relevant points that you can get no matter how it changes an image: rotate, scale…) and descriptors, which are unique patterns that can be found in an image.

CONTENT

1. HARRIS CORNER DETECTOR

A corner can be defined as the intersection of two edges, or a point for which there are two dominant and different edge directions in a local neighborhood of the point. To identify those elements, we use corner detectors which highlight relevant parts (pixels) of an image, which help during the process of identifying objects in an image in computer vision. One of the most well-known methods is the Harris corner detection algorithm. It's quite effective as it takes the differential of the corner score into account with reference to direction directly, instead of using shifting patches for every 45-degree angle. This makes it more accurate in distinguishing between edges and corners. Besides, it provides good repeatability under changing illumination and rotation.



Edge
No change in edge direction

Corner
Significant changes in all directions

Nevertheless, it is important to know that we have coroner detention algorithms but also edge detectors. The last one referring to edges, is able to find the areas where there is a sharp change in color or intensity, which usually correspond to object boundaries.

Let's try to see the main difference between those detector algorithms.

While edge detection identifies boundaries of objects, corner detection identifies points where two boundaries intersect:

Furthermore, edges refer to the process of identifying points in a digital image where the image brightness changes sharply or has discontinuities and these points usually correspond to object boundaries. In the case of corners, the process actually obtains points in the image where the directions of the edges change. They are considered to be a subset of the edge points and are the intersection of two edges.

It is important to know that corners are considered more robust as they are invariant to translation, rotation, and illumination (even if the image is moved, rotated, or the lighting changes, the corners remain the same).

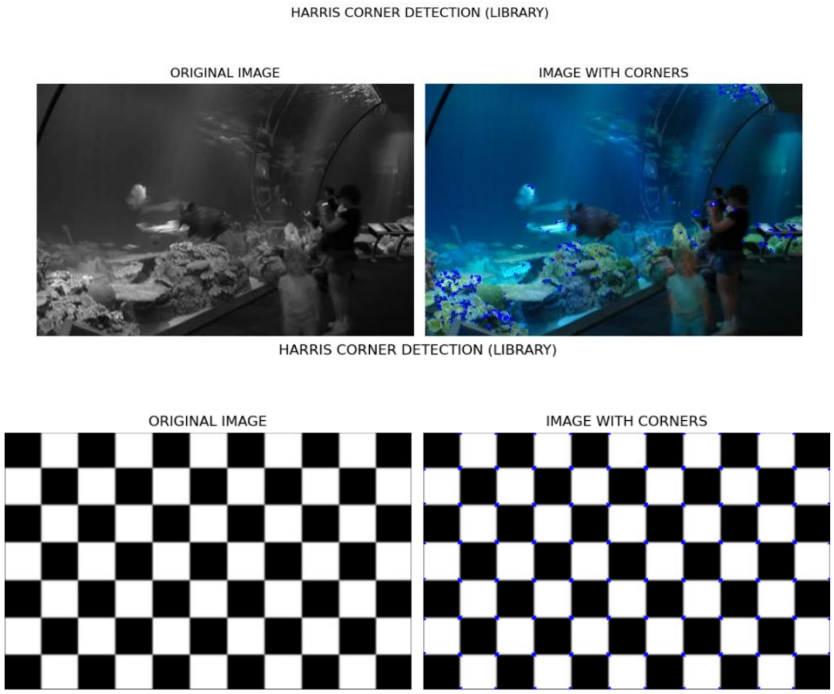
Nonetheless, the Harris corner detector has its limitation while processing complex images and can affect the accuracy and reliability of the Harris corner detector in complex image analysis tasks...

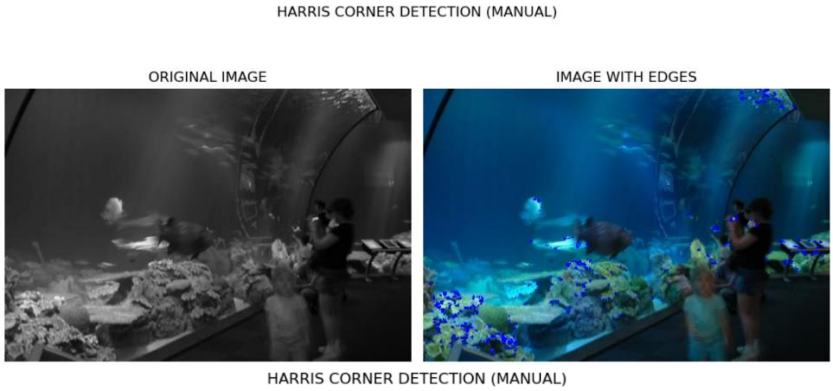
As this code relies on the strong invariance and local correlation if the image has low contrast and/or non-obvious edges it may not be able to identify the corners. Besides, having for example some areas with corners but some others with no corner found may mean that the feature points detected have been clustered due to the many non-local minima values. In addition, as we have known it has good outputs while playing with illumination and rotation, but it may struggle with more complex transformations. Lastly, if we talk about the codification as it works with a window size for the correlation function it may be influenced by this parameter selection.

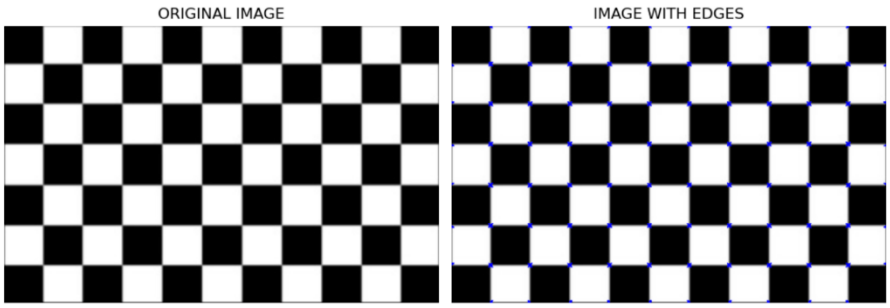
After getting some basic knowledge about Harris detector let's see how it works:

While elaborating our code we have had to consider some values to concrete parameters such as the kernel_size or the threshold. It is important to know that the last one determines how tolerant our algorithm has to be. A smaller number implies more tolerance and detectors as more numbers are accepted as corner values. In this project we have computed the manual and also implemented the library package about Harris in order to compare the results. However, while we were investigating how to compute the Harris using libraries, we saw that they tend to dilate the results to make it more visible. Nevertheless, to obtain the same results as with the manual we have not done it.

See in the following images:





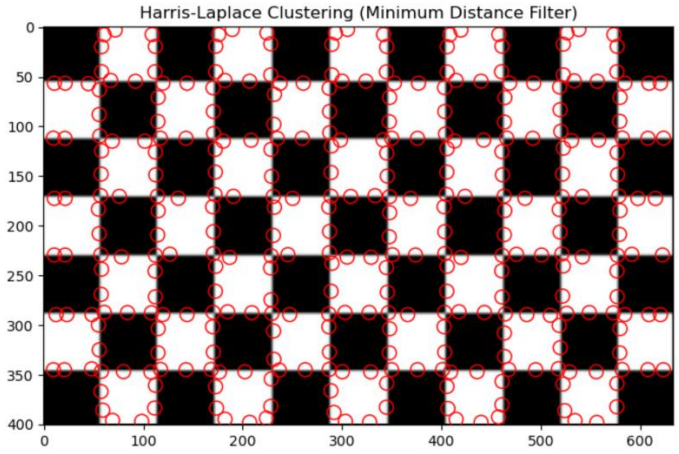


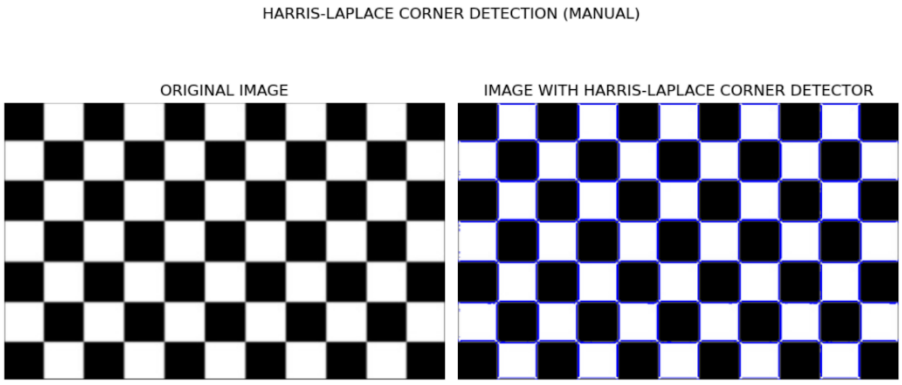
Besides from doing Harris we also have codified the manual Harris-Laplace which actually detects points of interest by combining the Harris detector and the Laplacian of Gauss. The Harris detector is used to locate points at each level of the scale-space representation. Then, the Harris-Laplace selects those points for the Laplacian of Gauss that reaches a maximum over the scale.

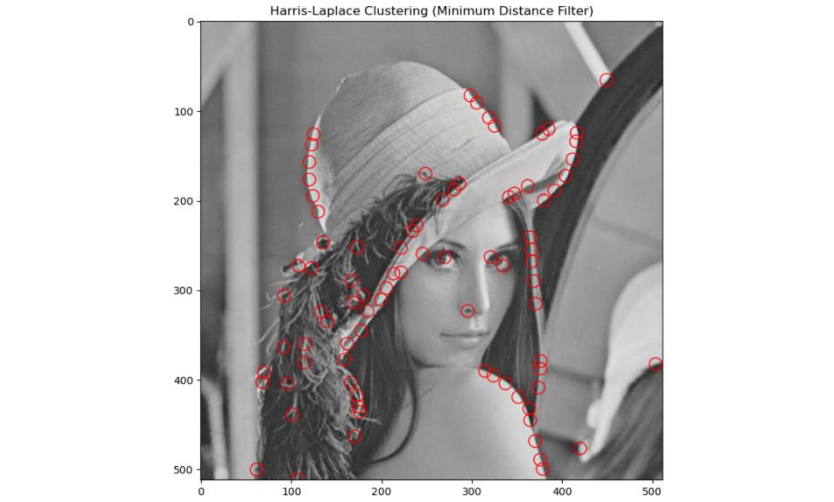
Nonetheless, after the execution we have found that although the Harris-Laplace detector can identify points of interest that are invariant to scale changes and affine transformations. So, sometimes if the threshold value is very little more points are considered of interest so that edges will be returned instead of points.

See it below:

We have decided to visualize it as a cluster also so that you get an idea of the “corners” (points of interest) that are selected.







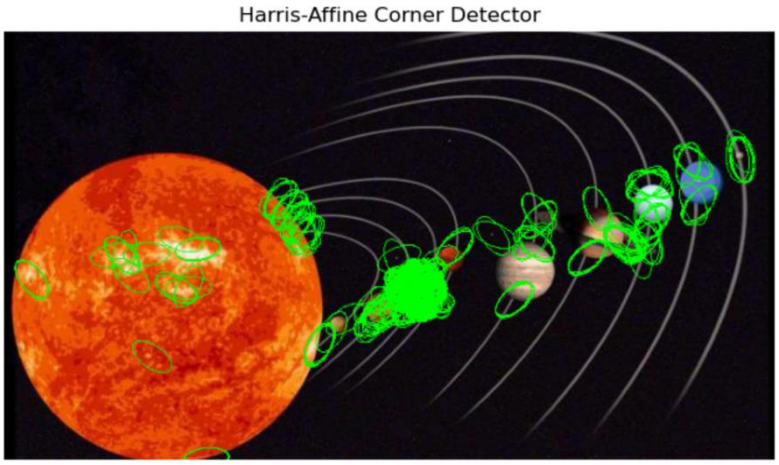
So, as it uses a blurry function before applying the Laplace it is normal that we detect some other structures as corners, as it is used to define contours.

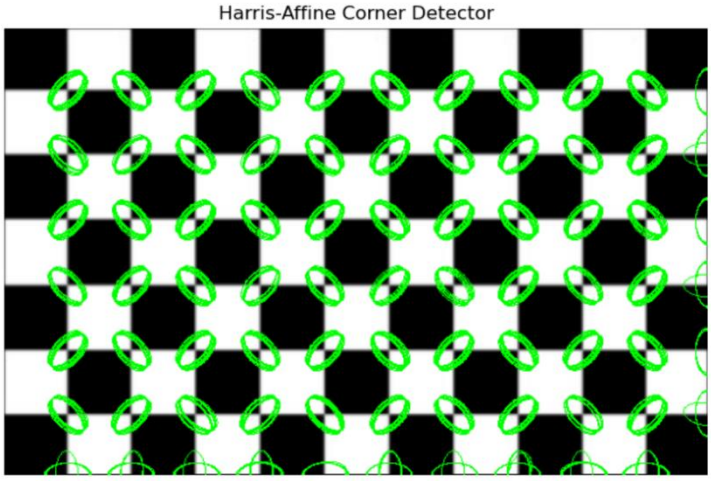
Finally, there is another algorithm for corner detectors which is an extension of Harris-Laplace, it is called the Harris-Affine corner detector. It is actually considered a feature detector that relies on the combination of corner points detected through Harris corner detection, multi-scale analysis through Gaussian scale space, and affine normalization using an iterative affine shape adaptation algorithm. Nonetheless, even if this method provides a good invariance, accuracy and efficiency, and even if it provides a good repeatability even if we change the illumination and rotation, it is actually quite difficult to encode due to its level of mathematical implications and considerations.

Here is what our code does using the OpenCV library:

- First it computes the Harris corner using cv2.cornerHarris by considering some parameters as; for corner detection (`blockSize`), the aperture parameter for the Sobel operator (`ksize`), and the Harris detector free parameter (`k`).
- Secondly, it finds those corners above a threshold which we actually choose (those points are converted to integer indices)
- Thirdly, compute LoG, considering the use of the cv2.GaussianBlur and then computing Laplacian by applying a cv2 function.
- Fourthly, by having the outputs of Laplacian and those points which are above the threshold we select those corners with positive results.
- Fifthly, for each corner we extract a local patch around the corner and compute the affine shape parameters using the moments of the patch and we compute the eigenvalues and eigenvectors of the moment's matrix. It then extracts the affine shape parameters: the lengths of the major and minor axes and the orientation. These parameters are stored as affine features.
- Lastly the results are visualized in the image using ellipses as the form that can define better the representation of those corners.

It's important to note that the choice of parameters like the threshold, sigma, k, and laplace_threshold can significantly affect the performance of the corner detector.





2. LOG BLOB DETECTOR ALGORITHM

In computer vision, we can define a blob as a region of an image in which some properties are constant or very similar (all the points in a blob can be considered similar). Blobs provide complementary information about regions, which is not obtained from edge detectors or corner detectors.

The first and one of the most common blob detectors is the Laplacian of Gaussian (LoG). It is particularly useful for detecting blobs that appear at various image scales or degrees of image focus.

In spite of all the good properties of blob detectors such as the LoG blob detector, it also has some drawbacks that we need to take into consideration. Firstly, it is sensitive to noise as it associates a bright (dark) blob with each local maximum (minimum) in the intensity landscape. Moreover, it may have some troubles to detect very small blobs because the detection speed is independent of the size of blobs as internally the implementation uses box filters instead of convolutions.

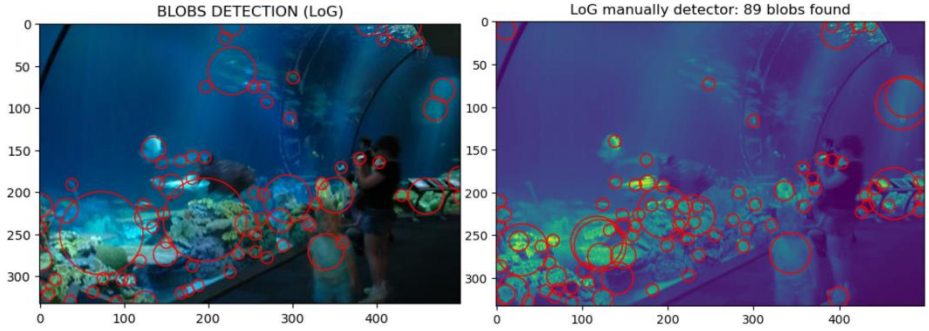
To solve some of these inconveniences we may try to either before using LoG detection you apply a reduction noise step (e.g. smoothing filter) or for small blobs you may want to work with other corner blob detectors (Determinant of Hessian).

Notwithstanding, it is imperative to know that there exist some differences between blob detection and edge detection which lies in the features they target. While edge detection looks for edges in the input image and identifies points where the brightness of the image changes sharply or has discontinuities, blob detection methods are aimed at detecting regions in a digital image where some properties are constant.

As far as we have gone, we have detected interest points so we can now compute a descriptor for every one of them. Descriptors are algorithms that take an image and output feature vectors, encode interesting information into a series of numbers and act as a sort of numerical unique pattern (as a fingerprint) and can be used to differentiate one feature from another. Descriptors can be categorized into two classes:

- Local Descriptor: It is a compact representation of a point's local neighborhood. They try to resemble shape and appearance only in a local neighborhood around a point (suitable for representing it in terms of matching).
- Global Descriptor: A global descriptor describes the whole image. They are not very robust as a change in part of the image may change the resulting descriptor.

Now let's see the result obtained by codifying the LoG manually (left image) comparing it with the computation of the one using library:



While creating the manual code it is relevant to emphasize that we have used the minimum distance idea to reduce the number of clusters that were referring to a similar zone, but instead of helping they were dirtying the quality and the theoretical ideas of the output image.

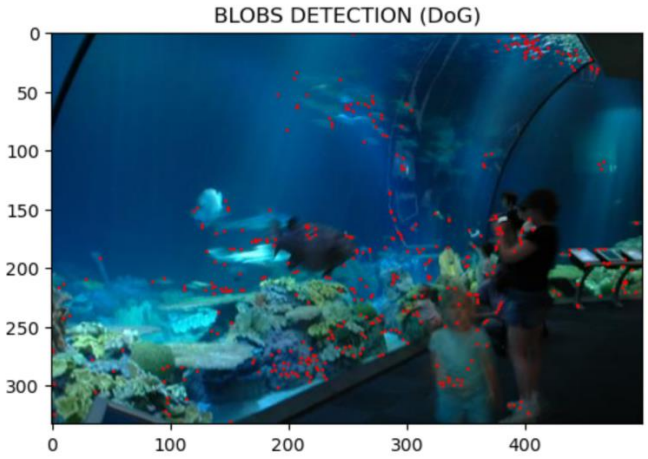
Lastly, using OpenCV library we have actually computed the DoG (difference of Gaussian) and the DoH (determinant of Hessian).

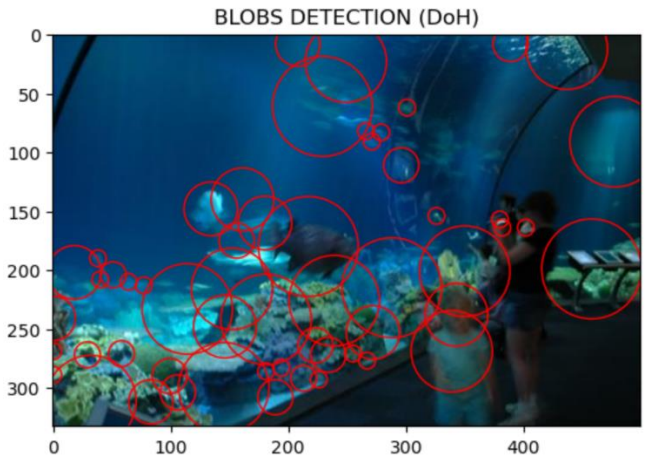
DoG is a feature enhancement algorithm that involves the subtraction of one Gaussian blurred version of an original image from another. It suppresses high-frequency spatial information in an image. Thus, the DoG is a spatial band-pass filter that can increase the visibility of edges and other details present in a digital image (effective with a high degree of noise).

DoH is used to detect blob-like structures in an image. It calculates the determinant of the Hessian matrix for each pixel in the image, and then finds local maxima in the resulting image. These local maxima correspond to blob-like structures in the original image.

In conclusion, while the DoG algorithm is a good choice for edge detection, the DoH algorithm is ideal for detecting blob-like structures.

Let's see the results in an image:





3. NORMALIZED COLOR HISTOGRAM AND COLOR SPACE CHANGES

When we are working with images, we usually compute the color histogram as it is useful for image feature extraction because they capture the frequency distribution of color in an image. At the end, they are able to capture the image from a different perspective, counting similar pixels and storing them. It is a procedure computationally efficient, easy to implement, and invariant to rotation and small changes in viewing position.

Let's see how it is codified:

1. First you compute the histogram. You count the occurrence of each pixel intensity value.
2. Lastly, as you are interested in being able to compare histograms you need to normalize the histogram to adjust it on a fair scale. This is done by dividing each bin in the histogram by the total number of pixels in the image.
3. The obtained result is a histogram that represents the relative frequency of different color intensities in the image (useful for comparing images or extracting features).

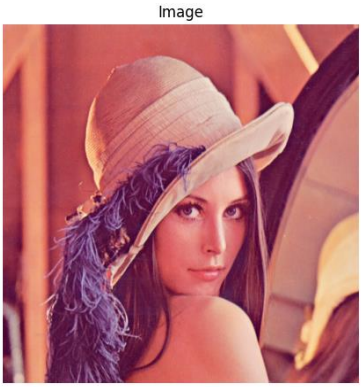
```python
def normalized_histogram(image, bins=32):
    hist = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0, 256, 0, 256, 0, 256])
    hist = hist / hist.sum()  #Normalize the histogram
    return hist.flatten()
```

Yet it is actually relevant to know that when we compute these histograms, we are working in a concrete color space. The choice of color should suit the specific requirements as it can affect the type and quality of features extracted from an image, the dimensionality of the feature space, and the performance of subsequent tasks like image classification or segmentation.

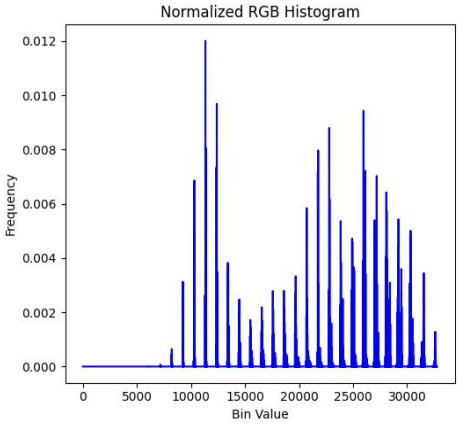
Many difficulties of color image processing may be resolved using specific color spaces. The problematic when discussing image databases is the same: in which color space a method will be the most effective. So, the process will be the following: presenting classical color spaces and represent images in these spaces to analyze which color space is the most relevant on the studied images. Then, we will introduce hybrid color spaces. The basic idea of hybrid color spaces is to combine several color components from different color spaces with the objective of increasing the effectiveness of color components to discriminate color data, and to reduce correlation rate between color components. At the end, the idea is to use a set of images as a unique image, and to realize statistical computation on this new image (the one created).

Besides all of that is relevant to know that as we have the possibility to play with colors, we are also able to change the image resolution. Higher resolution images provide more detailed information, which can enhance the performance of feature descriptors. However, they also require more processing power and memory to train deep Convolutional Neural Networks (CNNs) for instance.

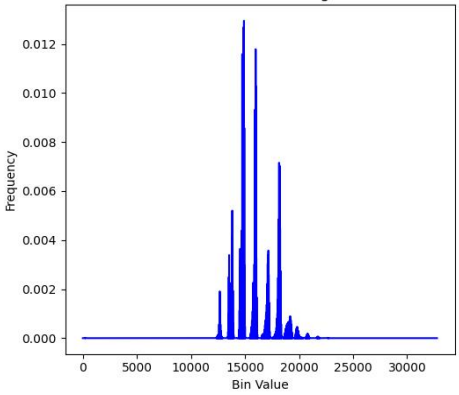
After changing an image to different color spaces and computing its color histograms, these are some of the results:



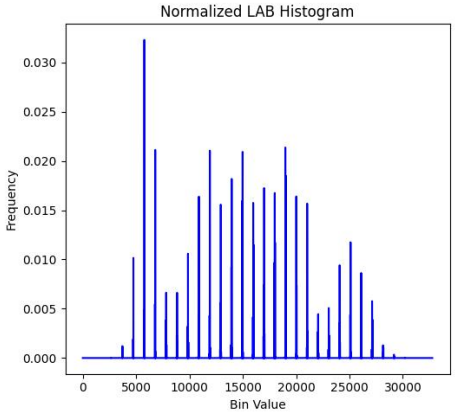
Image

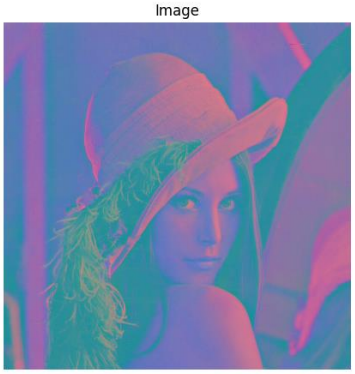

Normalized RGB Histogram
Frequency
Bin Value
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
5000
10000
15000
20000
25000
30000


Image

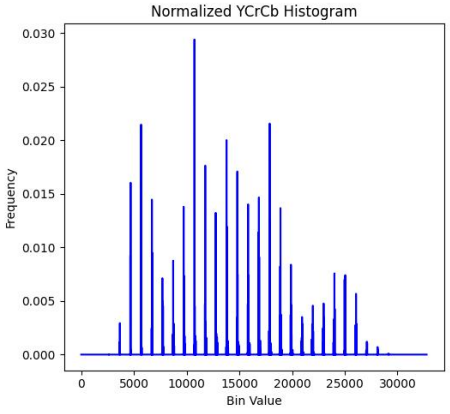

Normalized HSV Histogram
Frequency
Bin Value
0.012
0.010
0.008
0.006
0.004
0.002
0.000
0
5000
10000
15000
20000
25000
30000


Image

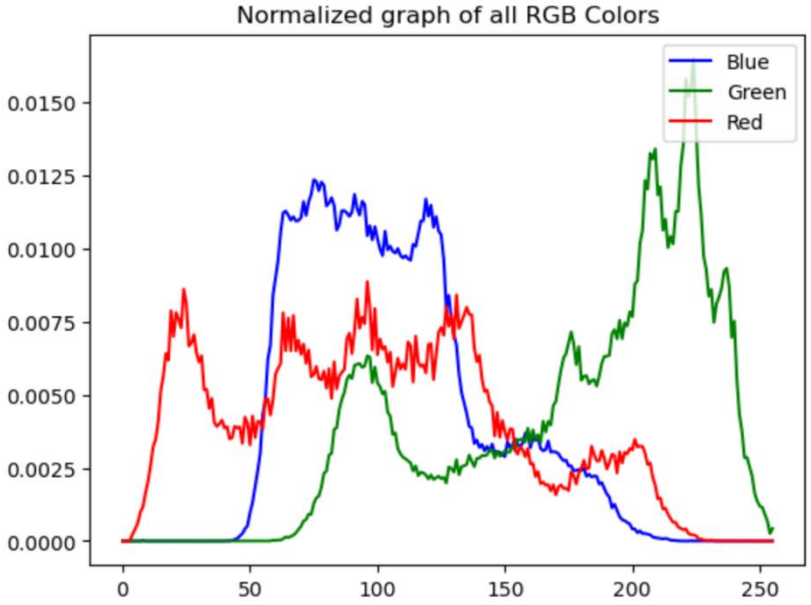

Normalized LAB Histogram
Frequency
Bin Value
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
5000
10000
15000
20000
25000
30000



Image


Normalized YCrCb Histogram
Frequency
Bin Value
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
5000
10000
15000
20000
25000
30000

Besides we codify to see how in RGB are represented the colors (red, green and blue) in an image. Just as the following:



4. IMPLEMENTATION AND ANALYSIS OF SIFT, SURF AND ORB

Feature detection is a low-level image processing operation that examines every pixel to see if there is a feature present at that pixel. Features may be specific structures in the image such as points, edges, or objects. The resulting features will be subsets of the image domain, often in the form of isolated points, continuous curves, or connected regions.

But besides knowing how we might get important information from an image is also relevant to know its properties. Scale and rotation invariance are crucial properties in feature detection. While scale invariance ensures that the algorithm can detect the same features regardless of the scale of the image, the rotation invariance is related to the fact that we can detect the same features regardless of the orientation of the image.

Howbeit, when we are talking about feature detection there is an aspect which is directly directed: the feature descriptors. As we have identified those points of relevance, we want to classify them. This is achieved by creating a vector of values that somehow encapsulates the characteristics of the image patch. The descriptor could be as simple as the raw pixel values, or it could be more complex, such as a histogram of gradient orientations. The problem is that by varying some aspects of the image such as the light can really influence our results. Some well-known detectors actually fail when they have non or really bad light conditions. Aside from this problem we actually need to consider other factors such as: illumination, occlusion, viewing angle, or camera rotation. To mitigate some of those disadvantages we could think about applying some form of post-processing after image capture can help mitigate changes in illumination or training a deep convolutional neural network model to predictively adjust camera gain and exposure time parameters can help maintain a number of feature matches.

Moreover, we have some algorithms that identify the same features across different images (feature matching) and handle some specific conditions or variations than an image may have. For instance, SIFT (Scale-Invariant Feature Transform) achieves scale invariance by constructing a pyramid of scales of the image. In case of rotation invariance,

it computes the dominant gradient at the feature area (image patch). In this algorithm, the role of orientation assignment plays a crucial role as it provides the rotation invariance. This assignment ensures that the description of the key point is relative to this orientation. Additionally, we have SURF (Speeded Up Robust Features) that is quite similar to SIFT but faster and more robust against the variations. To achieve scale invariance, it uses an image pyramid just as SIFT. However, for rotation invariance, it assigns the orientation to each key point based on the values obtained while computing the Haar wavelets in its neighborhood. Lastly, the ORB (Oriented FAST and Rotated BRIEF) is quite different in comparison with the previous two algorithms, it uses an image pyramid to achieve the scale invariance and for rotation invariance, it calculates the intensity-weighted centroid of the patch with the corner located at the center (direction of the vector from this corner point to the centroid gives the orientation).

As far as we have explained we know how they work but not which one is more efficient or accurate. Be as it may seem at the end, we will use one or another depending on what our priorities are and the requirements of our task. Nevertheless, SIFT and SURF are known for their accuracy. SIFT produces high-quality features based on gradients and so it does SURF which actually is only a faster implementation. On the other hand, ORB stands out for its computational efficiency (it uses the FAST corner detector).

Besides, we have some other limitations such as variation in occlusion. We want to avoid having these cases as it can greatly vary, making it challenging for feature descriptors to accurately capture and represent the features of the occluded object. This means a higher degree of complexity while executing feature descriptions. Moreover, we have issues while trying to locate where the occlusion happens. This adds another layer of complexity. Despite these challenges, there are several potential improvements which could enhance the performance of feature descriptors. One would be exploiting the specificities of motion propagation. Secondly, is the use of some integration of advanced techniques such as Hopfield networks, Deep Belief Networks (DBN), and Lanczos interpolation. These techniques can help develop a robust framework that effectively handles occlusions and improves recognition accuracy. Lanczos interpolation preserves image quality during resizing, the Hopfield network is ideal for feature extraction, and the DBN is employed for representation learning.

Surprisingly, when we talk about feature matching (identify the same features across different images), we may want to produce a segmented panorama or high-resolution image, this is where image stitching (photo stitching) might appear. It actually uses multiple images with overlapping fields so that by identifying some features (feature matching) it gets consistent results.

Such as image stitching means getting better results by overlapping different images, we actually have hybrid features that by combining the properties of multiple algorithms, such as SIFT (Scale-Invariant Feature Transform) and Harris, aim to leverage the strengths of each individual algorithm to improve overall performance.

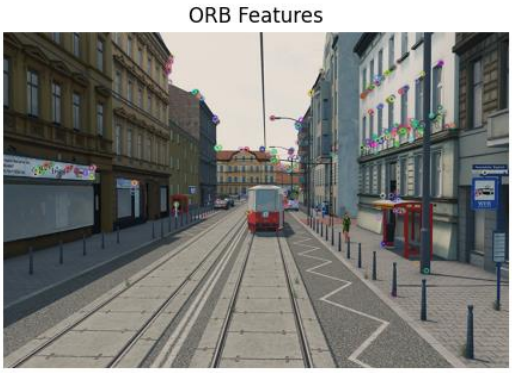
Let’s analyze it: On one hand, these hybrid detectors are known for their speed and efficiency. By integrating the Harris corner detector with the SIFT descriptor, we significantly increased the speed process which is beneficial for time-critical applications. On other hand, they offer improved stability in comparison with the individual algorithms. SIFT and SURF are effective in scale-space feature detection, but they lack stability and reliability during the detection process. Finally, by combining Harris and SIFT we gain robustness and invariance. This hybrid method is reliable in extracting

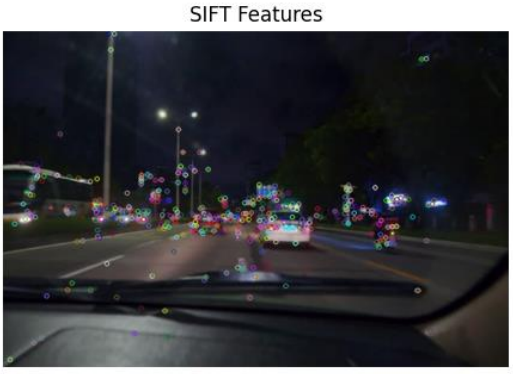
features that are invariant to translation, scale, and rotation. In consequence, it is less affected by changes in illumination, which enhances its robustness and makes it more resilient in various conditions.

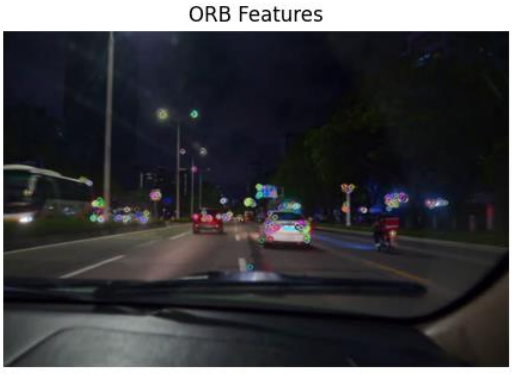
To sum up, we have explained about the feature detectors, descriptors and feature matching. Nonetheless, there is also another role involved in the study of an image content. This is the feature extraction. This method involves automatically computing a compact representation of some characteristic of digital images, which is used to derive information about the image contents just as a case of dimensionality reduction. Besides, this process also greatly affects the efficiency and accuracy of the systems (e.g. content-based image retrieval (CBIR)) as it helps in the reduction of the dimensionality of data, which is needed to process the data effectively. In other words, it means creating new characteristics that still capture the essential information from the original data but in a more efficient way.

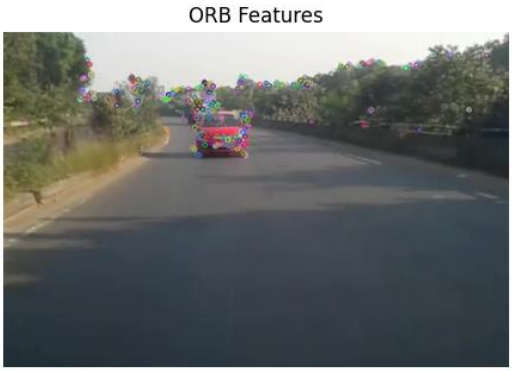
Lastly is relevant to know that feature matching also involves having some difficulties which can be faced using feature quantization (evolving dictionary which is consistent with feature statistics of the recent distribution history)

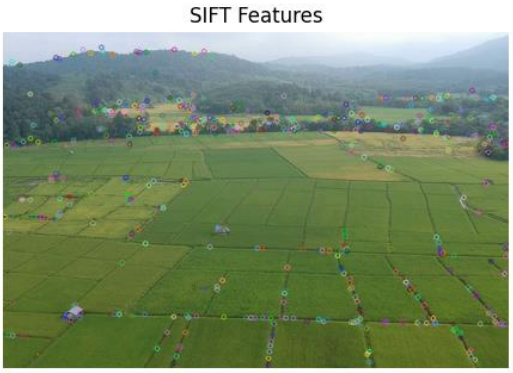
After implementing SIFT and ORB algorithms using cv2 implementations, we selected a dataset of 5 images and applied these methods. We couldn't include SURF because it is patented. These are the results:

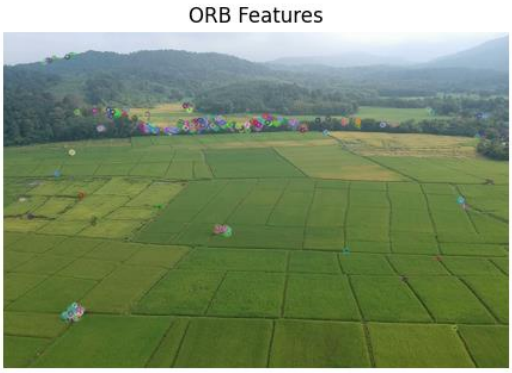






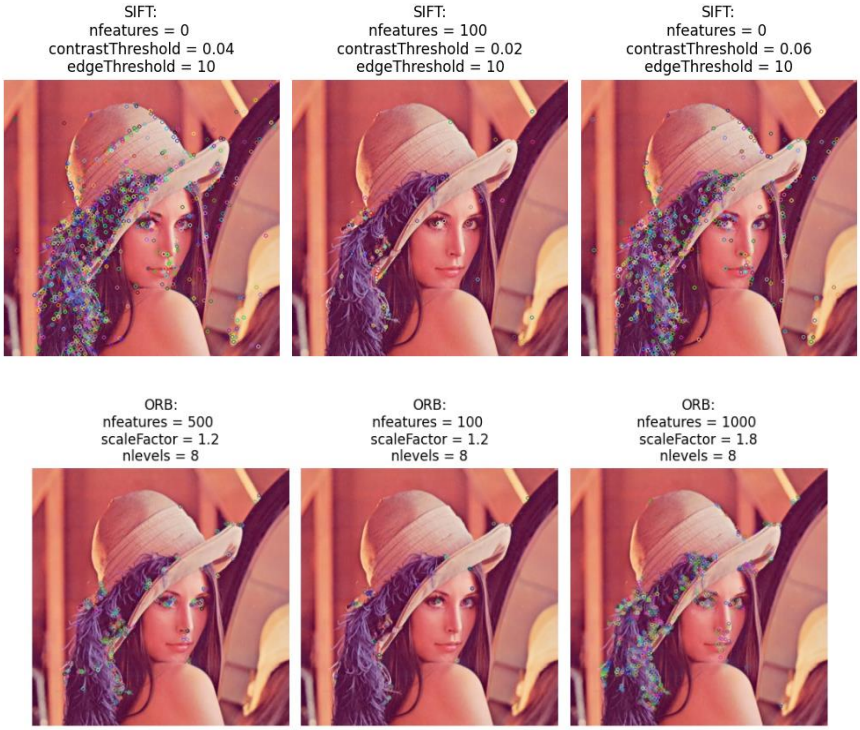






As we can see, SIFT detected many more features than ORB, since ORB has limited rotation invariance (it is not as robust in handling large rotations) and is less distinctive than SIFT.

However, we tried to adjust the parameters of the functions to see if it would make any change:



By increasing the "nfeatures" parameter in ORB we could make it detect more features.

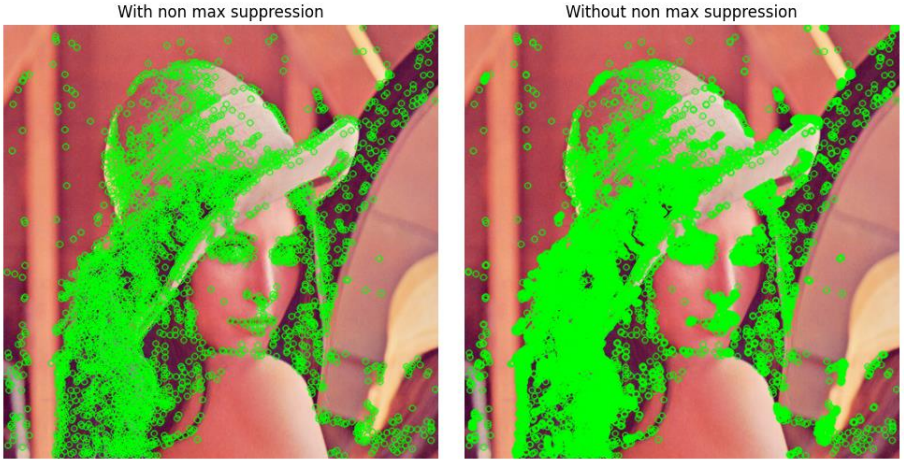
4.1. OPTIONAL EXERCISE: Implement other image descriptor methods.

We decided to implement 2 more feature detection algorithms: FAST and BRIEF. Even though BRIEF is also patented, we could use it by downloading an upgrade of the OpenCV contrib python library:

```
pip install --user --upgrade opencv-contrib-python
```

4.1.1. FAST (Features from Accelerated Segment Image)

FAST is a corner detection method which, as the name says, is very computational efficient. It is suitable for real-time video processing because of its high-speed performance. We implemented it using the "cv2.FastFeatureDetector_create()" OpenCV implementation in 2 ways: first we applied FAST with Non-Max Suppression, and then we disabled the nonMax Suppression to apply FAST again. Here are the results:



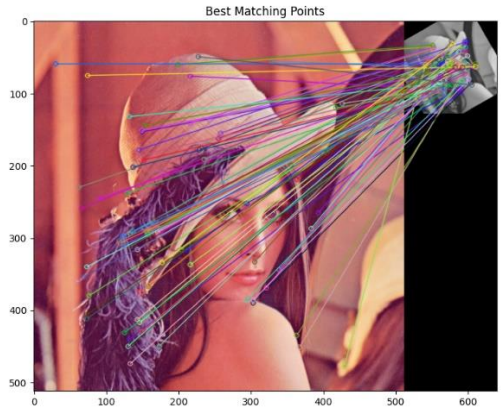
When we don't apply Non-Max Suppression, FAST detects many more features.

4.1.2. BRIEF (Binary Robust Independent Elementary Features)

The other algorithm we included was BRIEF, which, also as the name says, it is very fast to build and to match features and it easily outperforms other fast descriptors such as SIFT and SURF in terms of speech and recognition rate.

We implemented it using a training image and a testing image. The testing image was simply the original image but with scale invariance and rotation invariance added using image pyramid down sampling and affine rotation transformations. Then, FAST was applied to detect key points in both the training and the test images, and later we used BRIEF to compute binary feature descriptors for the detected key points. Finally, we draw the key points on the training image, both with and without size information:

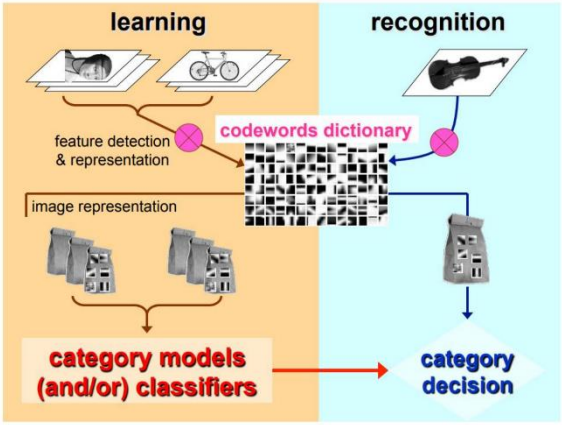
After that, we created a brute-force matcher to match BRIEF descriptors of key points from the training and the test image. The matches are sorted based on their distances, and the best matches are drawn on a new image:



5. BAG OF VISUAL WORDS PIPELINE AND VOC DATASET ANALYSIS

The Bag of Visual Words (BoVW) model is a popular technique used in image classification, where images are represented as a collection of visual words. Moreover, the BoVW model simplifies image analysis by reducing images to histograms of visual words, which can be easily compared for image classification or retrieval tasks. Nevertheless, it is relevant to know its key steps (some of them previously explained) to understand its execution process.

1. First, feature extraction. Identify unique points, or key points, in an image and describe them. These descriptions, known as descriptors, represent the local appearance of the key points.
2. Second, Codebook Construction. Using the extracted features from the images we clustered using a clustering algorithm like K-means. The center of each cluster forms a "visual word". The collection of these visual words forms a "codebook" or "dictionary" that represents a specific pattern found in different images.
3. Third, Vector Quantization. Each image is represented as a frequency histogram of visual words. It serves as the new, more compact representation of the image.
4. Fourth, Weighting with Term Frequency-Inverse Document Frequency (TF-IDF). Evaluate the importance of a word to a document in a collection or corpus. In this case. "Term frequency" refers to the number of times a visual word appears in an image, while "inverse document frequency" is the measure of whether the visual word is common or rare across all images. With this model we can highlight visual words that are more distinctive.



5. Lastly, Similarity Measurement. To classify or find similar images, the histogram of the new image is compared with the histograms of images in the database. This comparison can be done using similarity or for instance, the Euclidean distance.

Though there are still some improvements, mostly in the accuracy of the classification. Maybe one option would be incorporating high-level semantic information into the model. This would refine image semantics by introducing user conceptions for extracting semantic vocabularies and reducing the size of the vocabulary. Another option could be using more advanced feature extraction techniques or optimizing the existing ones. The better the features that the model can extract, the more accurate the classification results will be.

Aside from all of those methods it is beneficial if we consider the contextual information to capture more complex patterns in the data and so, potentially improving the model's

performance. However, it also makes the model more complex and may increase the computational cost. We have different ways to incorporate the context:

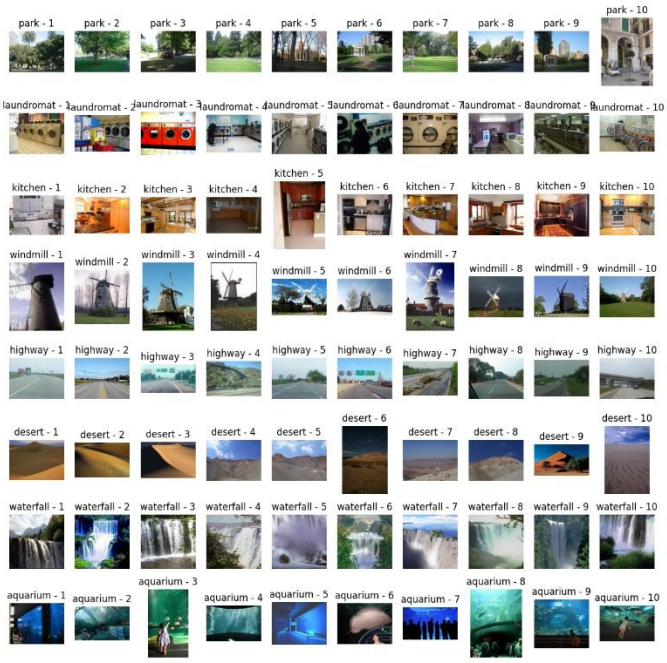
Firstly, we can include the spatial information model. By considering such spatial relationships, the BoVW model can potentially improve its performance instead of only taking into account the traditional 'bag' model (implies ignoring the spatial layout). Secondly, the semantic context, which by identifying patterns it can learn to recognize higher-level features. This approach allows the model to capture more complex patterns in the data, thereby enhancing its ability to classify images accurately. Thirdly, the scale and orientation of visual words also provide valuable contextual information which can potentially improve its ability to recognize objects in images. Lastly, temporal context is considered in video, comparing the previous and sequent frames and capturing temporal patterns in the data.

Here are the steps we followed in our code to apply the BoVW model:

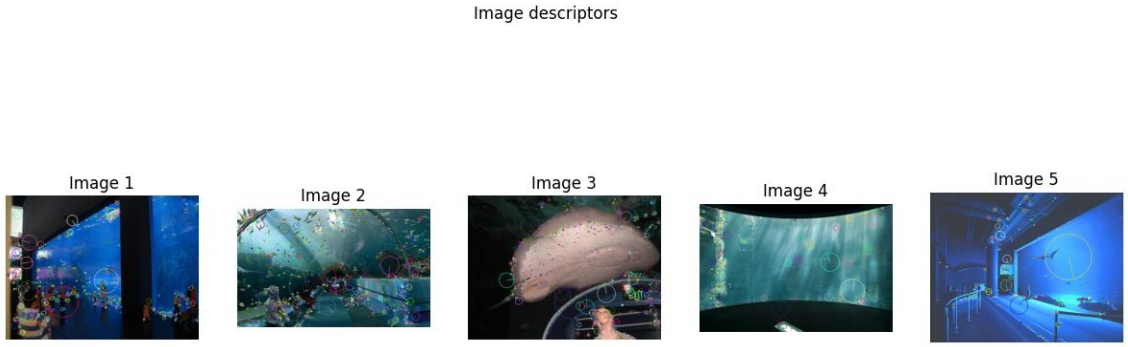
1. We first loaded the dataset in two different lists: one for the training images and another for the test images. We also defined some global variables, like the number of clusters we were going to use, the number of images, the number of classes and the samples per class. These global variables will help us later in other functions.

```
Number of images in the TRAIN dataset: 640
Labels: {'park', 'laundromat', 'kitchen', 'windmill', 'highway', 'desert', 'waterfall', 'aquarium'}
Number of images in the TEST dataset: 160
Labels: {'park', 'laundromat', 'kitchen', 'windmill', 'highway', 'desert', 'waterfall', 'aquarium'}
NUMBER OF CLUSTERS: 80
NUMBER OF CLASSES: 8
NUMBER OF SAMPLES X CLASS [80, 80, 80, 80, 80, 80, 80, 80]
```

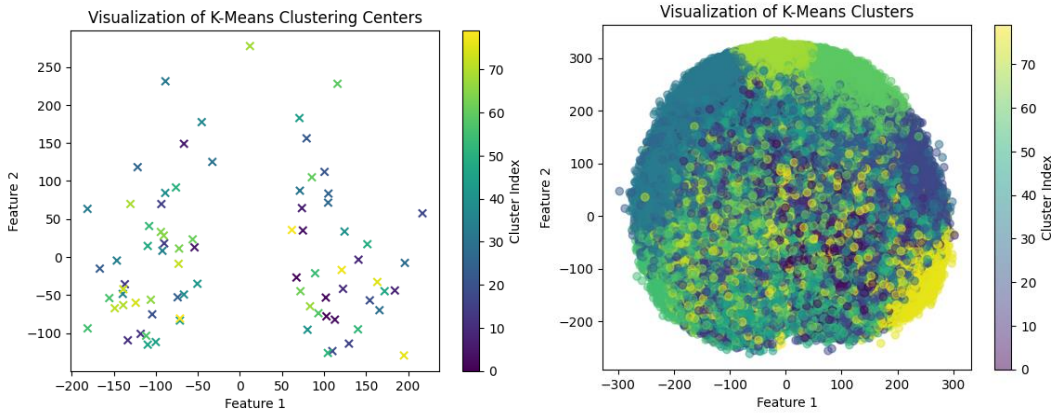
We also printed a visual version of the dataset to get an idea of what type of images we were working with:

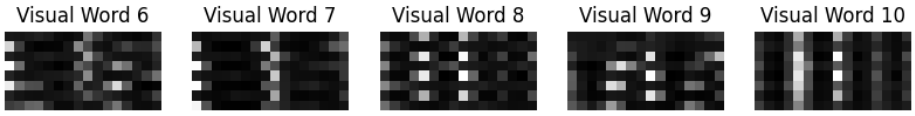


2. Now it's time to obtain the descriptors of the images. To do so, we used the SIFT features extractor, and for each image we extracted their descriptors and added them to a list. Let's visualize some of the image's descriptors:

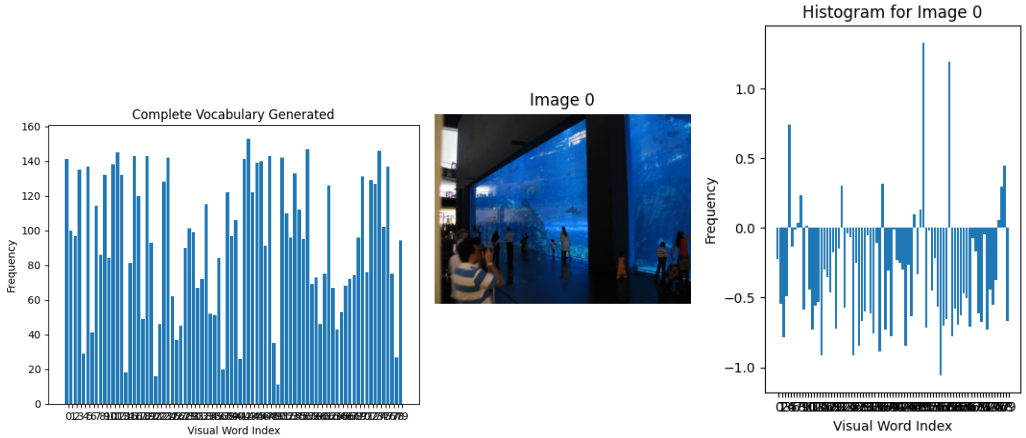


3. For building the codebook we used Minibatch KMeans instead of KMeans because it was more computational efficient. Even though KMeans would be a better option, it was taking hours to execute since our dataset wasn't small, that's why we decided to stick with MiniBatchKMeans. To get the codebook, we first created the model "kmeans" using the number of clusters we defined at the beginning, which is 80 (since we had 8 categories and we multiplied it by 10). We couldn't choose a much bigger number since it would also affect its computation time and would take too long to execute. Then we fitted the descriptors in the kmeans model and finally, to obtain the codebook, we just got the kmeans cluster centers. We also implemented some visualizations to see the Kmeans clusters and some words from the codebook:



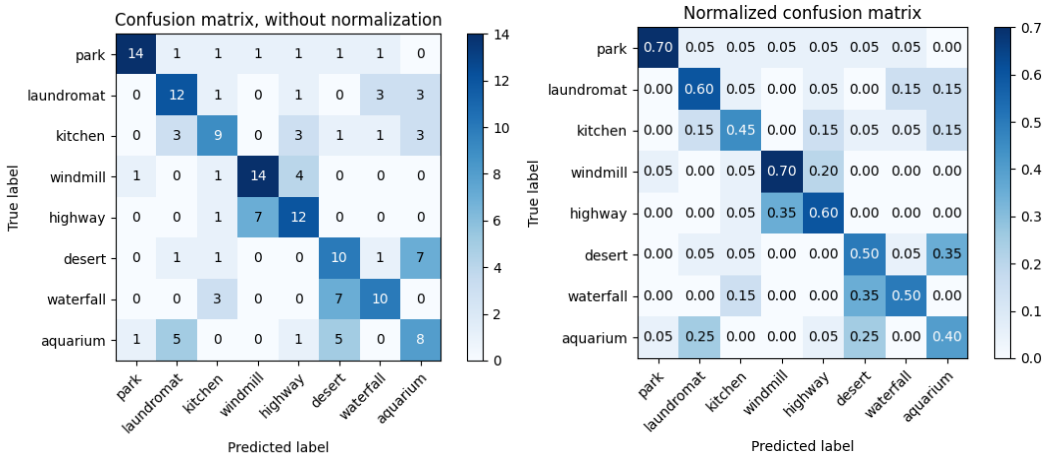


4. To encode each image as a frequency histogram of visual words we got the features of each image and implemented a visualization of the histogram. We also did a complete vocabulary visualization. For the individual images, we only computed 2 histogram examples, but with our function plotImageHistogram, just by changing the image_index we could obtain any image's histogram.



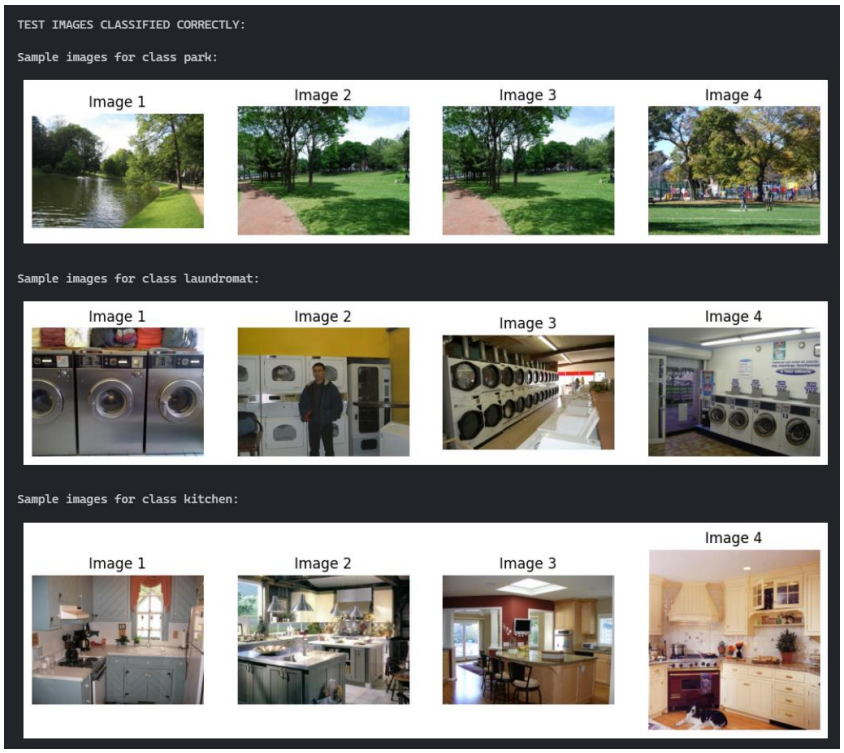
5. To train the classifier we used the Support Vector Machine model, using different weight for each class depending on the samples per class. We used GridSearchCV to obtain the most optimal parameters for SVM.

6. Finally, it's time to evaluate our model with the testing dataset. These are the results:

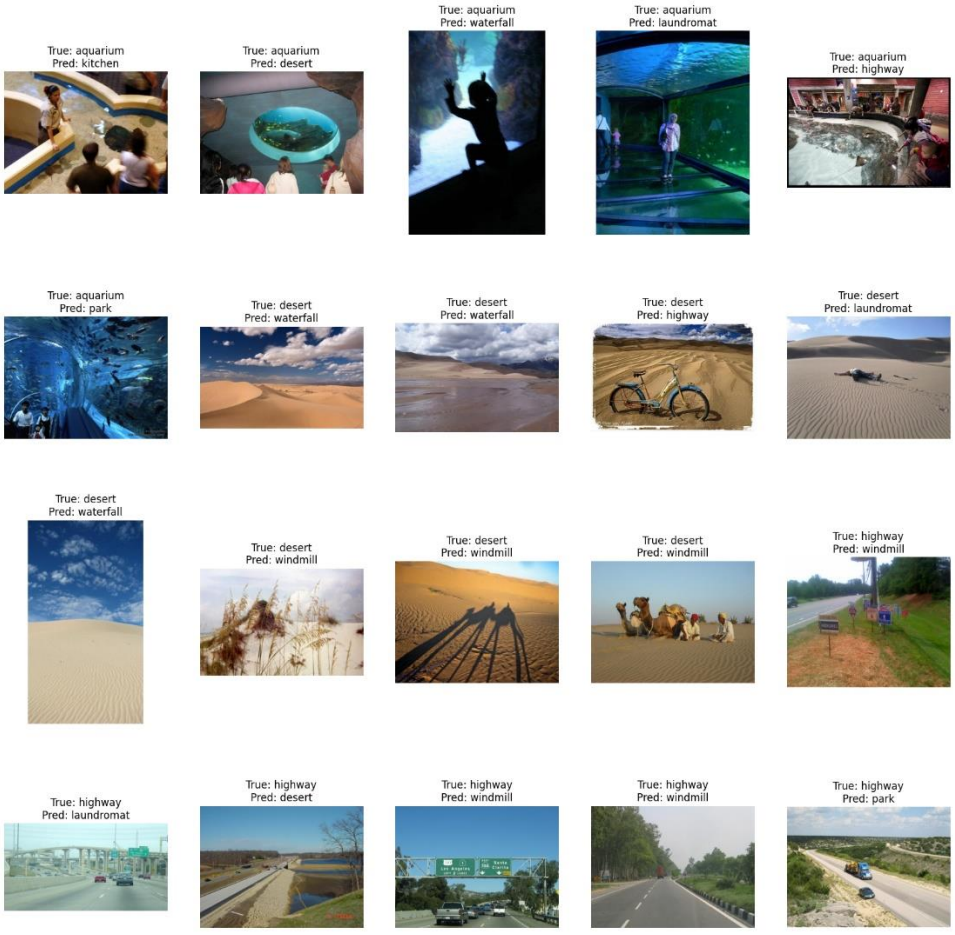


We obtain a pretty low accuracy score, but we think it's because we used MiniBatch KMeans instead of KMeans. Here are some of the images that were correctly classified:

```
Confusion matrixes plotted.
Accuracy score: 0.556
Accuracy calculated.
Correctly classified images: 89
Incorrectly classified images: 71
```



And these are some of the images classified incorrectly:



CONCLUSION

In conclusion, this project has provided us a comprehensive exploration of various feature extraction methods and image feature descriptors.
For instance, the implementation of the Harris Corner Detector and the Laplacian of Gaussian Blob Detector has increased our understanding of how algorithms can identify and extract meaningful features from images. These corner and blob features serve as the fundamental building blocks for more complex image analysis tasks.
Moreover, as we have studied colors spaces and features through the implementation of Normalized Color Histograms, it has revealed the importance of color as a distinguishing feature in image recognition tasks.
What's more, the use of feature descriptors such as SIFT, SURF, and ORB has allowed us to compare and contrast the strengths and weaknesses of these methods and decide how we might want to apply them in different scenarios. Finally, the development of a Bag of Visual Words model has demonstrated the practical utility of these feature extraction and descriptor methods in reality.
To sum up, this project has given us the chance to strengthen and see how well we know the theoretical explanations and to expand our understanding of these concepts. Besides, we have had the opportunity to practice not only the theory but also our programming and searching skills. All that we have done will be useful not only in computer vision aspects but in other future projects that we have to deal with; Patience and teamworking has been determinant to accomplish this project.