REPORT OF THE LAB 4

1. THEORETICAL UNDERSTANDING OF THE ALGORITHMS USED

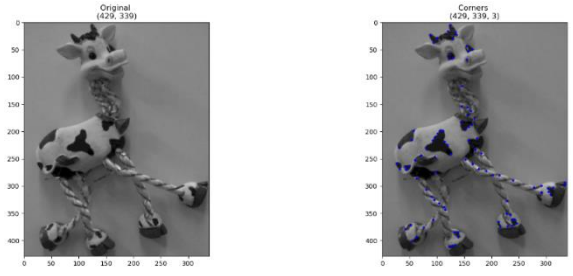
SECTION 1: Feature detection

In computer vision, feature detection refers to the process of identifying distinctive and relevant structures or patterns within an image. These structures, known as features, can be corners, edges, blobs, or other localized patterns that can be used to describe and represent certain characteristics of the visual content.

Edge detection refers to the process of identifying points in an image where sharp changes in intensity occur, typically on the boundary between two different regions in an image. On the other hand, corner detection is an approach used within computer vision systems to extract certain kinds of features and infer the contents of an image. 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 neighbourhood of the point. The main difference between edge detection and corner detection lies in the nature of the features they are designed to detect: edges are significant local changes in the image, while corners represent a lot of variation in the direction of the gradient in a small neighbourhood. Corners are considered more robust features because they are points stable over changes of viewpoint and illumination. They are also invariant to translation, rotation, and illumination, making them particularly useful for tasks such as motion detection, image registration, video tracking, image mosaicking, panorama stitching, 3D reconstruction, and object recognition. While edges can be sensitive to noise and may not be robust in the presence of occlusions or partial occlusions.

If we focus on Corner detectors, the principles behind them involve analyzing the local structure of the image and detecting areas where intensity changes occur in a way that is characteristic of corners. One widely used corner detector is the Harris corner detector. Here are the key principles behind corner detectors, using the Harris method as an example. The Harris corner detection algorithm uses the following principles:

- **Intensity Gradient:** It calculates the intensity gradients in both the x and y directions for each pixel in the image.
- **Structure Tensor:** The algorithm then constructs a 2x2 matrix known as the structure tensor for each pixel. This matrix summarizes the local intensity variations in the neighbourhood of the pixel.



Corner Detection

- **Corner Response Function:** The corner response function is defined using the eigenvalues of the structure tensor. The eigenvalues represent the principal curvatures of the local intensity distribution. If both eigenvalues are large, it indicates a corner.
- **Thresholding:** A threshold is applied to the corner response function to identify points where the response is significantly greater than the surrounding region.

Harris corner detection is effective in identifying corners because it considers both the intensity gradients and the second-order intensity variations. Corners are points where the image intensity changes in multiple directions, leading to high values in the corner response function.

The Harris corner detector, while effective in many cases, has some limitations, particularly in dealing with complex images or under certain conditions. Here are some of the key limitations of the Harris corner detector:

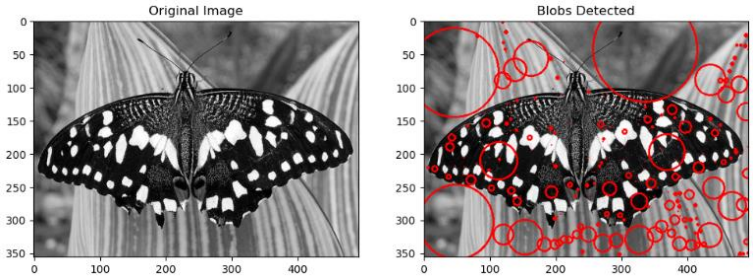
1. **Sensitivity to Image Noise:**
    - The Harris corner detector is sensitive to noise in the image. Since it relies on intensity gradients, the presence of noise can lead to false positives, where noise is mistaken for corners.
2. **Scale Sensitivity:**
    - The Harris corner detector does not inherently handle scale variations in the image. It may not perform well when dealing with corners at different scales. This limitation can be critical in scenarios where objects appear at varying distances from the camera.
3. **Rotation Sensitivity:**
    - The detector is sensitive to rotations in the image. If an image undergoes a significant rotation, the Harris corner detector may not perform as well, as it does not explicitly account for rotation invariance.
4. **Thresholding Challenges:**
    - Determining an appropriate threshold for the corner response function can be challenging. Setting the threshold too low may result in false positives, while setting it too high may cause the detector to miss some corners.
5. **Uniform Regions:**
    - In regions of the image with uniform intensity, the Harris corner detector may struggle to identify corners. It tends to work better in textured or high-contrast regions.

To address some of these limitations, researchers have developed extensions or alternative corner detection methods that incorporate scale invariance, rotation invariance, and other improvements. Examples include the Scale-Invariant Feature Transform (SIFT) and the Speeded Up Robust Features (SURF), which aim to enhance the robustness and performance of corner detection in more complex scenarios.

Also, **hybrid feature detectors**, combining properties of algorithms like SIFT and Harris, offer benefits in terms of robustness and speed. SIFT provides distinctive features, while Harris is computationally efficient. Evaluating their performance involves assessing the trade-off between accuracy and speed. Hybrid detectors can be particularly advantageous in real-time applications, where a balance between feature distinctiveness and computational efficiency is crucial.

Besides corner detector and edge detectors, there are other types of detectors.

The Laplacian of Gaussian (LoG) blob detector is a feature detection method used in image processing. It's designed to identify regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. These regions are often referred to as blobs. The LoG blob detector works by convolving the image with a Laplacian of Gaussian function, effectively combining Gaussian smoothing with the Laplacian operator. The Gaussian function serves to smooth the image and reduce noise, while the Laplacian operator is used to identify regions of rapid intensity change.



Edge Detection

The LoG blob detector differs from edge detection in several key ways. While both methods are used to identify important features in an image, they look for different types of features. Edge detection methods, such as the Canny edge detector, are designed to identify points in an image where the image intensities changes distinctly. These points usually represent the boundaries of an object within the image. On the other hand, the LoG blob detector is designed to identify regions in an image where some properties are constant or approximately constant. These regions, or blobs, can signal the presence of objects or parts of objects in the image.

However, Laplacian of Gaussian (LoG) blob detector still has some weaknesses. It is sensitive to noise and may produce false positives, especially in real-world scenarios with varying noise levels. It is also computationally expensive due to the convolution with a Gaussian kernel. To improve its performance, techniques such as scale-space analysis and thresholding can be applied. Employing an adaptive thresholding mechanism and incorporating machine learning algorithms to refine blob detection results could enhance accuracy and reduce false positives.

In feature detection, features are often selected or designed to be invariant to certain transformations, such as rotation, scaling, or changes in illumination. This makes them robust and suitable for various applications.

Scale and rotation invariance in feature detection refers to the ability of an algorithm to identify features in an image regardless of the size (scale) or orientation (rotation) of the image. Here's how SIFT, SURF, and ORB handle these variations:

- **SIFT (Scale-Invariant Feature Transform)**: SIFT achieves scale invariance by identifying key points (extremal points) in the image at different scales (using Gaussian convolutional filters). Also, the role of orientation assignment in the SIFT algorithm is crucial for achieving rotation invariance. SIFT aims to detect and describe key features in images that are robust to variations in scale, rotation, and illumination. The orientation assignment step ensures that the computed feature descriptors are invariant to the orientation of the local image patch around each key point. This is done by computing

an orientation histogram for each key point, which is done by considering a 16x16 pixel neighbourhood for each key point.

- **SURF (Speeded Up Robust Features)**: SURF approximates the Difference of Gaussian (DoG) with box filters for scale invariance. For rotation invariance, it uses wavelet responses in both horizontal and vertical directions by applying adequate Gaussian weights.
- **ORB (Oriented FAST and Rotated BRIEF)**: ORB uses the FAST key point detector for scale invariance. For rotation invariance, it computes the intensity-weighted centroid of the patch with the located corner at the center. The direction of the vector from this corner point to the centroid gives the orientation.

In terms of **computational efficiency and accuracy**:

- **SIFT**: It is highly accurate and robust to scale and rotation changes, but it requires a large computational complexity.
- **SURF**: It performs faster than SIFT without reducing the quality of the detected points.
- **ORB**: It is the most efficient among the three and provides a good trade-off between speed and performance.

The **ORB algorithm** differs from SIFT and SURF in terms of feature matching in the following ways:

- ORB is a fusion of the FAST keypoint detector and BRIEF descriptor with some modifications.
- ORB computes the intensity-weighted centroid of the patch with the located corner at the center. The direction of the vector from this corner point to the centroid gives the orientation.
- In case of matching, the features are compared only if they have the same type of contrast (based on sign), which allows faster matching.

SECTION 2: Feature description

Feature descriptors are an algorithm that extracts features in an image and describes it as a vector. To evolve a robust and efficient system for object detection and learning using feature descriptors, key point based local Feature descriptors like SIFT, SURF, ORB, etc are deeply studied.

Feature descriptors play a crucial role in object recognition within the field of computer vision. Object recognition involves identifying and categorizing objects or scenes within images or video streams. Feature descriptors are used to represent distinctive information about local regions of an image, enabling robust matching and recognition. Here are key aspects of the role of feature descriptors in object recognition:

1. **Localization of Key Points:** Feature descriptors are often associated with key points or interest points in an image. These key points represent regions with distinctive visual characteristics, such as corners, edges, or blobs.
2. **Representation of Local Image Information:** Feature descriptors encode information about the local appearance of a region around a key point. This representation is designed to be distinctive and invariant to certain transformations.
3. **Matching and Correspondence:** Feature descriptors enable matching between corresponding key points in different images. When recognizing an object, the system compares the feature descriptors of the key points in the query image with those in a database or a reference image. Matching enables establishing correspondences between points and, consequently, recognizing objects.

Feature descriptors extract information from images in terms of numerical values that are difficult for humans to understand and correlate. They reduce the dimensionality of the original image, which helps to decrease the overheads of processing large numbers of images.

There are two main types of feature descriptors: global and local.

- **Global descriptors** are used in image retrieval, object detection, and classification. They describe the image as a whole to generalize the entire object. Examples of global descriptors include contour representations, shape descriptors, and texture features.
- **Local descriptors**, on the other hand, are used for object recognition or identification. They describe image patches or key points in the image. Examples of local descriptors include Scale Invariant Feature Transform (SIFT), Speeded Up Robust Feature (SURF), and Local Binary Patterns (LBP).

The choice between global and local descriptors depends on the specific application. For low-level applications such as object detection and classification, global features are typically used. For higher-level applications such as object recognition, local features are generally preferred.

Current feature descriptors struggle with occlusions as they focus on local patterns and may fail when significant parts of an object are obscured. Improved methods could involve learning context-aware representations that consider the global context of features. Integrating depth information or using 3D feature descriptors can enhance the ability to handle occlusions and provide more robust feature representations.

On the other hand, feature descriptors play a crucial role in the process of image stitching, particularly for creating panorama images. Image stitching is the process of combining multiple photographic images with overlapping fields of view to produce a segmented panorama or high-resolution image. Varying image resolutions affect the effectiveness of feature descriptors, particularly in matching and recognition tasks. *Lower resolutions may lead to loss of fine details, impacting descriptor distinctiveness. Higher resolutions increase computational complexity and memory requirements. Adaptive scaling mechanisms and hierarchical feature extraction approaches, such as image pyramids, can address this challenge by capturing features at multiple resolutions.*

Feature descriptors represent the image patch surrounding a feature point. They provide a significant amount of key information that is essential for image stitching. This is because accurate extraction of these features can decrease misalignment flaws in the final stitched image. These features are then matched across different images. The matched features provide the basis for aligning and blending the seams of the images automatically to create a seamless panoramic image.

In essence, feature descriptors contribute to the image stitching process by enabling the detection, description, and matching of features across multiple images. This facilitates the alignment and blending of images, ultimately leading to the creation of a seamless panoramic image.

SECTION 3: Feature extraction: importance of colour

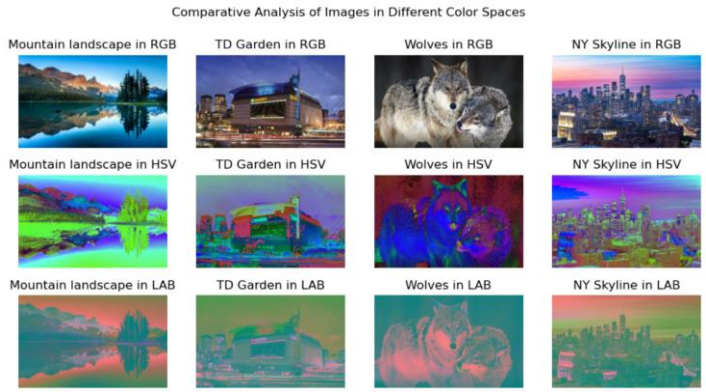
Feature extraction is a process in which relevant information or distinctive patterns are extracted from raw data, such as images, to create a more compact and informative representation. In the context of computer vision, feature extraction involves identifying and capturing important characteristics of an image that are relevant for a particular task, such as object recognition, classification, or analysis. Colour and handling varying lighting conditions are two aspects that can be considered during feature extraction.

One common approach for colour feature extraction is to use colour histograms. A histogram represents the distribution of colour intensities in an image. So, by quantizing the colour space and counting the occurrences of different colour bins, a colour histogram provides a concise description of the image's colour content. This makes them computationally efficient, easy to implement, and invariant to rotation and small changes in viewing position. They represent the image from a different perspective, counting similar pixels and storing it.

The process of creating a normalized colour histogram involves the following steps:

- Collect your data: This involves gathering the pixel values from your image.
- Decide on the number and size of bins: These bins will categorize your data.
- Count the number of data points (pixel values) that fall within each bin: This forms the basis of your histogram.
- Normalize the histogram: This involves dividing each bin count by the total count of pixels, which scales the histogram values so that the total sums to 1. This step is crucial when comparing histograms of images with different pixel counts.

A change in colour space can significantly affect feature extraction in images. Different colour spaces or colour models have been created depending on how each of the trichromatic colours is defined. The most commonly used colour space is the RGB colour space, where each of the colours is defined by adding three primary colours in the visible light spectrum (red, green, and blue) with various proportions. Other commonly used colour spaces include LUV, HSV/HSL/HSI, YCrCb. Each colour space has its own pros and cons, and the choice of colour space can influence the effectiveness of the feature extraction.



Hybrid colour spaces, formed by combining attributes from different colour models, offer improved robustness in feature extraction across diverse lighting and environmental conditions. Integrating luminance information from grayscale models like YUV with chromatic information from RGB or HSV models can provide a more comprehensive representation. Utilizing machine learning techniques to dynamically adapt colour space transformations based on the scene characteristics further enhances adaptability to varying conditions.

Changes in lighting conditions can significantly impact object appearance, making accurate detection and tracking challenging. So, algorithms should exhibit resilience to variations in illumination, ensuring consistency in object recognition.

For instance, in many robotic applications, especially long-term outdoor deployments, the success or failure of feature-based image registration is largely determined by changes in lighting. Similarly, existing computer vision and object detection methods, which strongly rely on neural networks and deep learning, are vulnerable to multiple factors such as illumination.

To mitigate these challenges, several strategies can be employed:

- **Learning Visual Feature Descriptors**: One method is to learn visual feature point descriptors that are more robust to changes in scene lighting than standard hand-designed features. By tracking feature points in time-lapse videos, one can easily generate training data that captures how the visual appearance of interest points changes with lighting over time.
- **Using High Dynamic Range (HDR) Imagery**: HDR imagery can capture and process a wide range of scene dynamic range similar to the human eye. HDR trained models are therefore able to extract more salient features from extreme lighting conditions leading to more accurate detections.
- **Deep Learning**: Deep learning-based object recognition can significantly increase the recognition speed and compatibility with external interference. Convolutional neural networks (CNNs), for example, have the advantages of end-to-end, sparse relation, and sharing weights.

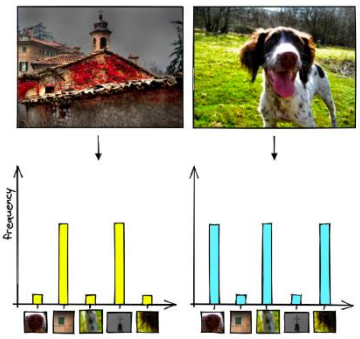
SECTION 5: Bag of Visual Words

Bag of words is a Natural Language Processing technique of text modelling. In computer vision, bag of visual words (BoVW) is one of the pre-deep learning methods used for building image embeddings. We can use BoVW for **content-based image retrieval**, object detection, and image classification.

Constructing a Bag of Visual Words (BoVW) model involves several key steps:

1. **Extract Visual Features:**
    - The process begins by extracting distinctive visual features from images. Common methods include using feature detectors like SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features), or ORB (Oriented FAST and Rotated BRIEF).
    - These features capture key information about the image, such as corners, edges, and textures, and provide a representation that is invariant to scale, rotation, and illumination changes.
2. **Create Visual Words:**
    - Once visual features are extracted, a clustering algorithm (typically k-means) is applied to group similar features into clusters or "visual words." The number of clusters determines the vocabulary size of the model.

    - Each cluster centroid becomes a visual word, representing a group of similar features. The visual words serve as a compact and descriptive representation of the original features.
3. **Build Sparse Frequency Vectors:**
    - For each image in the dataset, the next step involves representing it as a bag of visual words. This is done by counting the occurrences of each visual word in the image.
    - The result is a sparse frequency vector where each element corresponds to the count of a specific visual word in the image. The order of the words is usually determined by the order of the centroids in the clustering step.



4. **Adjust Frequency Vectors with TF-IDF:**
    - To address the issue of some visual words being more informative than others, term frequency-inverse document frequency (TF-IDF) weighting is applied to the frequency vectors.

- TF-IDF gives more weight to words that are rare across the entire dataset but common within a specific image. This helps to emphasize the discriminative power of certain visual words.

5. **Compare Vectors with Similarity or Distance Metrics:**
    - Once the TF-IDF-weighted vectors are obtained for all images, they can be compared using similarity or distance metrics, such as cosine similarity or Euclidean distance.
    - Similar images will have similar BoVW representations, making it possible to perform tasks like content-based image retrieval or image classification based on the nearest neighbors in the BoVW space.

Feature quantization in the context of the Bag of Visual Words (BoVW) model refers to the process of converting continuous visual features into discrete visual words or codes. This step is crucial in forming the vocabulary of the model and plays a significant role in subsequent steps. Feature quantization is typically done during the creation of visual words (Step 2). In the feature quantization step, a clustering algorithm, often k-means, is applied to group similar visual features into clusters. Each cluster center becomes a visual word, and the visual features are quantized by assigning them to the nearest cluster centroid. This mapping from continuous features to discrete visual words is the essence of feature quantization.

Significance of Feature Quantization:

- Reduced Dimensionality: Feature quantization reduces the dimensionality of the visual feature space. Instead of dealing with the continuous values of visual features, the model works with a limited set of discrete visual words. This reduces the complexity of subsequent computations and memory requirements.
- Semantic Representation: Quantized visual words serve as a more semantically meaningful representation of the image content. By grouping similar visual features into clusters, the model captures the underlying structure and patterns in the data. This is essential for creating a meaningful visual vocabulary.
- Robustness to Variations: Feature quantization helps in making the BoVW model more robust to variations in the input data. Since visual words represent clusters of similar features, small variations in the appearance of an object may not significantly impact the final representation, making the model more tolerant to changes in scale, rotation, or lighting conditions.

How can the BoVW model can be improved? It can be improved in several ways to enhance its classification accuracy.

One approach is through **Feature Selection**. This involves searching for the best feature detector that avoids locating a large number of key points which do not contribute to the classification process. By focusing on the most relevant features, the model can make more accurate classifications.

Another strategy is the incorporation of **High Discriminative Words**. The traditional word bag model can be improved by combining it with contrast words and Hamming embedding technology, thereby creating a high discriminative Bag of Words. This can then be combined into the SVM image classification model, resulting in a model with higher discriminative power and improved classification accuracy.

Lastly, **Contextual Information** can be incorporated into the BoVW model. This involves computing the relative conjunction matrix in order to preserve the spatial order of the data by coding the relationships among visual word. Incorporating patch-level contextual information into the BoVW can reduce ambiguity in patch encoding by visual words, which is often caused by information loss due to vector quantization.

In feature-based image classification, context is crucial as it provides additional information that can help distinguish between different classes. One way to incorporate this contextual information is by constructing a hierarchical codebook. In this codebook, visual words in the upper hierarchy contain contextual information of visual words in the lower hierarchy. Another approach is to use a novel representation called *Contextual Bag-of-Words (CBOW)*. This representation models two kinds of typical contextual relations between local patches, namely a semantic conceptual relation and a spatial neighbouring relation. Yet another method is to use word groups with high relevance and incorporate their statistics into the BoVW representation.

On the other side, feature extraction plays a pivotal role in Content-Based Image Retrieval (CBIR) systems, influencing the efficiency and accuracy of searches. Effective feature representations enhance the system's ability to capture semantic information, allowing for more meaningful comparisons between images. Advanced feature descriptors, such as deep learning-based representations, can significantly improve the discriminative power of CBIR systems, leading to better retrieval results. Additionally, incorporating relevance feedback mechanisms based on user interactions can further refine the accuracy of image searches.

2. PRACTICUM: RESULTS AND DISCUSSION

Harris Corner Detection Implementation.

We have taken 3 images. We have also taken the convolution function from Lab 1.

Harris Corner Detection: We take the gradient from one direction and another (gradient_x(img) and gradient_y(img)). Then do the matrix. And after that, from each pixel, we do the sum of products of the derivatives, apply the formula, and normalize. Then we start an empty list for the corners; and for every window_size, we compare with the threshold, and if it is bigger than the threshold, we add it to the list *'corners'*. Then we have another function called draw_corners. Because, once we have this list with all the corners, where there is the coordinate of each one; when we return it and pass it to the draw_corners function, it will be visualized in the resulting image. All this, as previously said, has been done with 3 images:

- The cow image. We have stated a threshold of, after many trials, 0,10.
- Chessboard image. Same process, same threshold of 0,10.
- Objects image. This image is a little bit blurred, that's why some corners are not detected. We have stated a threshold of 0,45.

Changing the threshold, we change how many of them we count as corners. This needs to be changed according to each image's properties as seen in these 3 examples.

LoG Blob Detector Implementation

- We apply the detect_blobs(image, sigma=3) function. Apply Laplace of Gaussian filter.
- And then, we do a binary image, where blobs are white and everything is black (similar to a mask). A binary thresholding operation is performed on the absolute values of the filtered image. Thresholding is done at a value of 1, and the result is inverted (cv2.THRESH_BINARY_INV).
- After that, we find the boundaries of the blobs. And, similarly to before, we now draw circles where there are contours, instead of drawing points where there are corners.
- Then, we have to calculate the area and the center (moments) of each blob (contour).

  ```
  area = cv2.contourArea(contour)
  moments = cv2.moments(contour)
  ```

  The bigger the blob, the bigger the contour.
- The final result is the image with red circles drawn around detected blobs.

Normalized Colour Histogram and Colour Space Changes

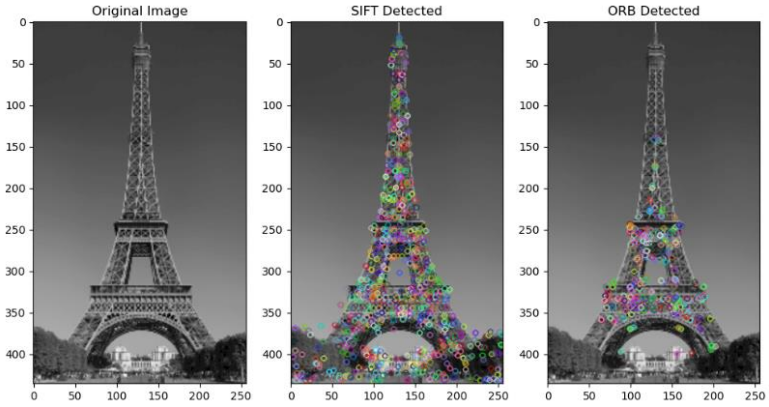
Describe what happens. We take 4 different images, normalize them, and we pass it to histograms with different colour spaces. The different colour spaces we use are: RGB, HSV, LAB. All images will be seen in the 3 colour spaces so that we can have a comparative analysis of the images in different colour spaces.

Implementation and Analysis of SIFT, SURF, and ORB

We do the descriptor, take the key points and descriptions, and draw with circles. Looking more in deep:

- **SIFT:**
  - Create a SIFT object using 'cv2.SIFT_create()'.
  - Detect keypoints and compute descriptors using 'sift.detectAndCompute'.
  - 'keypoints': Points of interest in the image.
  - 'descriptors': Descriptors associated with each keypoint
  - Use 'cv2.drawKeypoints' to draw the keypoints on the original image, resulting in 'image_with_keypoints'.

- **ORB:**
  - Create an ORB object using 'cv2.ORB_create()'.
  - Detect keypoints and compute descriptors using 'orb.detectAndCompute'.
  - 'keypoints2': ORB keypoints.
  - 'descriptors2': ORB descriptors.
  - Use 'cv2.drawKeypoints' again to draw the ORB keypoints on the original image, resulting in 'image_with_keypoints2'.

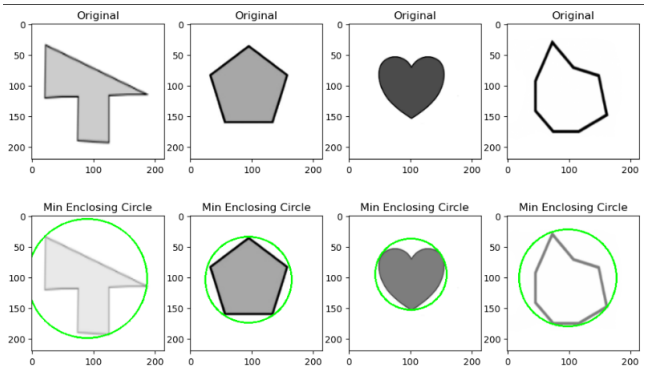


All this is what we have done with the Tour Eiffel. And we have plotted the visualisation for SIFT algorithm and the ORB one.

Then, we analysed the hyperparameters of each algorithm.

Optional Exercise 4.5: Implementation of another image descriptor method

We have the function min_enclosing_circle, which takes a grayscale image as input and identifies the **minimum enclosing circle around the largest contour in the binary version of the image**. The code then reads four images, resizes them, displays the original grayscale images on the top row of a subplot, and on the bottom row, it shows the same images with the minimum enclosing circle drawn around the prominent shape in each image. The images and their corresponding minimum enclosing circles are displayed using Matplotlib. The purpose is to demonstrate the functionality of the min_enclosing_circle function on different shapes. If no contours are found in the image, the function returns the original image; otherwise, it returns the image with the minimum enclosing circle drawn on it, the center coordinates, and the radius of the circle.



Bag Of Visual Words Pipeline and VOC Dataset Analysis

Step 1:

- Takes the images that we have of the folder path that we introduce. Then, takes the training. And then, for each folder,
    - Dataset → Train and test → 8 elements → 8 classes → in each class we have 80 image to do the training → in each folder, we take every image, and classify in classes.

Step 2:

- In the feature extraction, we have implemented both ORB and SIFT, but we only use ORB because SIFT is not giving us good results.
- extract_features_orb function → we take the ORB descriptor and do the same as we did on the Tour Eiffel implementation; then, return the 'descriptors'.
- extract_features_parallel function (we found this function online) → executes the function of extract_features_orb but for every image. Instead of executing sequentially it does parallelly, which makes us win more time of execution.

Step 3: Building vocabulary:

- We implement K Means, where k=46, which means there are 46 clusters. We tried with a lot of k values and found that 46 was the more optimal one.
- We also have "mini baches" (also found online), which also accelerates the process of execution.

Step 4: Feature Encoding:

- We take the features and do the histogram.

Step 5: Training Classifier:

- We give the encoded features and the train labels, and do the pipeline and the fit (which is basically training).

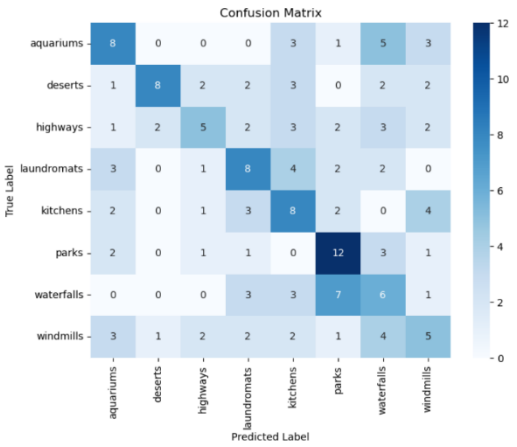
Step 6: Model Evaluation:

- We take the test, and evaluate. Here is where we have the confusion matrix.
  - Accuracy is low
  - Specificity for each class is quite high
  - Sensitivity/Recall is low. Most of the classes are mistakenly classified.

3. PROBLEMS / EXPLANATIONS + ALTERNATIVES

In the **Confusion Matrix** of the **Bag Of Visual Words**, the metrics were quite bad (still are); so we tried many things to improve the implementation.

- We tried changing the k of the clusters.
- We tried changing the gamma of the svc (can be found in step 5).
- There is a moment in step 1, when uploading image, when we do a resize; since we computationally save a lot of time. And also, because since the images are of different sizes, we kind of standardized. That could affect the accuracy. But, we removed that resizing to check if that was true, and the accuracy became even lower, that's why we kept that step.
- We also looked if by removing mini baches and parallel it would work better, but that was not the case.
- Finally, we modified the number of features taken by ORB, since if it had 256 instead of the predetermined 128, it could embrace more information in order to classify better. We tried with different values and found out the most optimal was **200**. The accuracy incremented. So that has helped.



However, it is easy for the classifier to misinterpret the images and misclassify due to the **similar colours**, etc. If a waterfall is misclassified as an aquarium, it would be "normal". If the waterfall is misclassified as a kitchen, then is when we have the real problem.

The main misclassification is found in waterfalls classified as parks, which could be because of the intensity, and, in this case, the green colour.