CourseName: Computer Vision Lab **Course Code:** CSP-422

Experiment:1.3

Aim:

Write a program to analyze the impact of refining feature detection for image segmentation.

Software Required:

Any IDE (Jupyter Notebook, Pycharm, Google Colab).

Relevance Of the Experiment:

The experiment is relevant for understanding the impact of refining feature detection for image segmentation is crucial in various fields such as computer vision, medical imaging, and object recognition. By refining feature detection, we can enhance the precision of segmenting objects in images, leading to improved analysis, object recognition, and decision-making processes.

Description:

Refining feature detection for image segmentation involves enhancing the quality and accuracy of the detected features to improve the subsequent segmentation process. Some techniques commonly used for refining feature detection in the context of image segmentation are:

- Multi-scale Analysis: Image segmentation often requires detecting features at different scales to capture objects of various sizes. Multi-scale analysis involves applying feature detection algorithms (e.g., SIFT, SURF) at multiple scales by using image pyramids or scale-space representations. This helps in capturing features across different levels of detail and scale.

- Non-Maximum Suppression: Feature detection algorithms such as corner detectors or blob detectors may produce multiple responses for a single feature due to noise or image variations. Non-maximum suppression techniques help in selecting the most salient or distinct features by suppressing the less significant responses in their vicinity.
- Adaptive Thresholding: Feature detection algorithms often rely on setting a threshold to determine the presence of features. However, a fixed threshold may not be optimal for all images, especially in cases with varying lighting conditions or contrast. Adaptive thresholding techniques dynamically adjust the threshold based on local image statistics to improve feature detection accuracy.
- Edge Refinement: Edges are important cues for many segmentation tasks. After detecting edges using techniques such as the Canny edge detector or gradient-based methods, refining the edges can enhance feature detection. This can involve edge thinning, edge linking, or edge contour enhancement techniques to improve the accuracy and connectivity of detected edges.
- Feature Filtering and Selection: Feature detection algorithms may produce a large number of features, including false positives or irrelevant features. Filtering and selecting the most informative and relevant features can improve the subsequent segmentation process. This can be done based on feature quality measures (e.g., response strength, repeatability), spatial distribution, or contextual information.
- Feature Fusion: Combining multiple types of features or feature descriptors can enhance the discriminative power of feature detection. Feature fusion techniques involve integrating information from different feature extraction methods or channels (e.g., color, texture, shape) to capture a more comprehensive representation of the

image content. This can be achieved through concatenation, weighted averaging, or more advanced fusion strategies.

- Contextual Information: Utilizing contextual information can improve the accuracy of feature detection and segmentation. Context-aware feature detection methods take into account the relationships between neighboring pixels or features to refine the detection results. This can involve incorporating spatial constraints, semantic priors, or contextual cues from surrounding regions.
- Deep Learning-based Approaches: Deep learning models, such as convolutional neural networks (CNNs), can be trained to directly detect and refine features for segmentation tasks. These models learn feature representations from large amounts of data and can capture complex patterns and contextual information. Fine-tuning or incorporating pre-trained CNN models can improve feature detection performance.

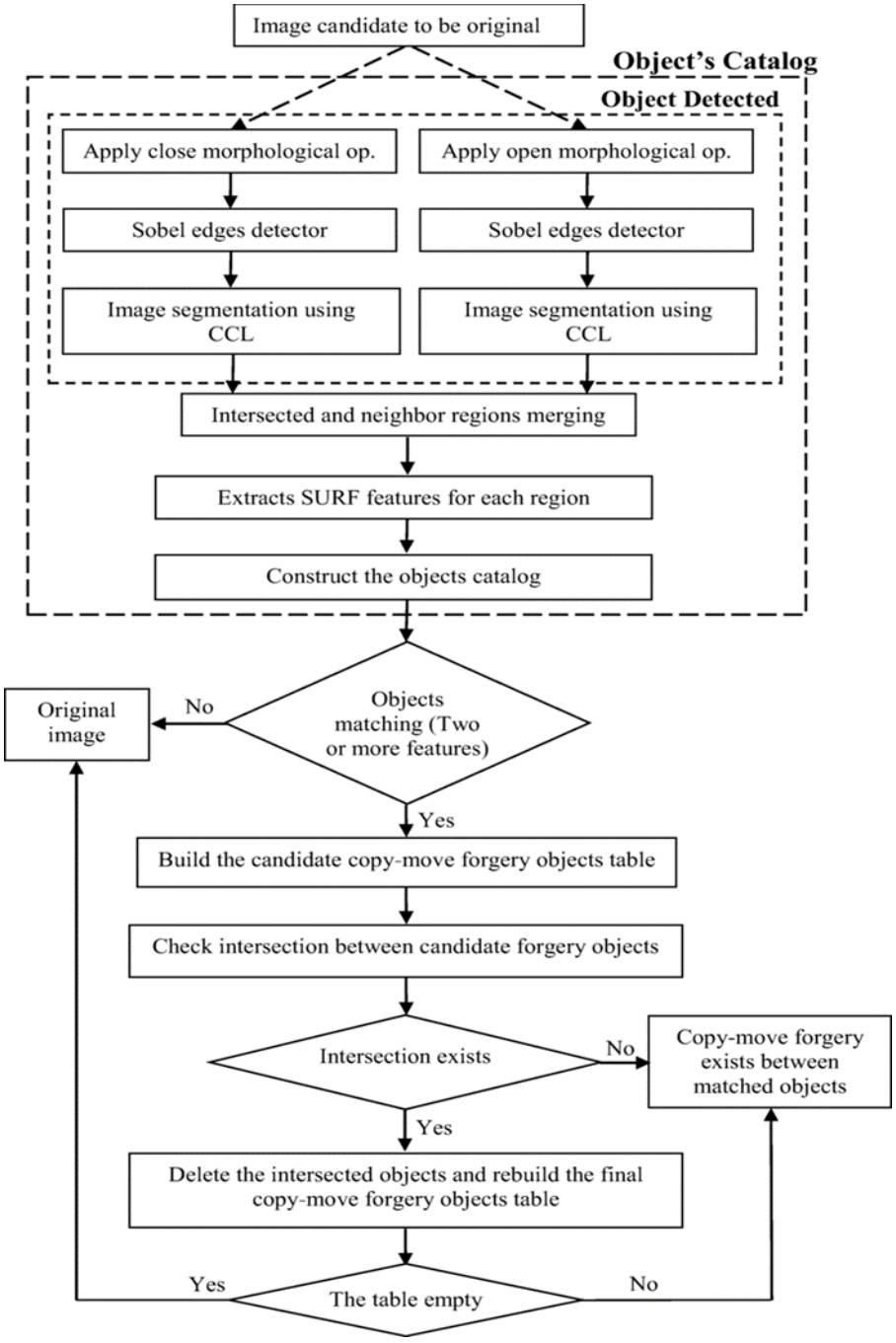
Pseudo code/Algorithms:

1. Import necessary libraries and modules.
2. Load image dataset.
3. Preprocess images if needed.
4. Select target object image.
5. Extract features from target object.
6. Loop through dataset images:
   a) Extract features from each image.
   b) Match features between target object and image.
   c) Calculate matching score.
   d) Store matching score and image.

CourseName: Computer Vision Lab **Course Code:** CSP-422

7. Sort images based on matching scores.
8. Display top-ranked images and scores.

Flowchart/Steps:



CourseName: Computer Vision Lab **Course Code:** CSP-422

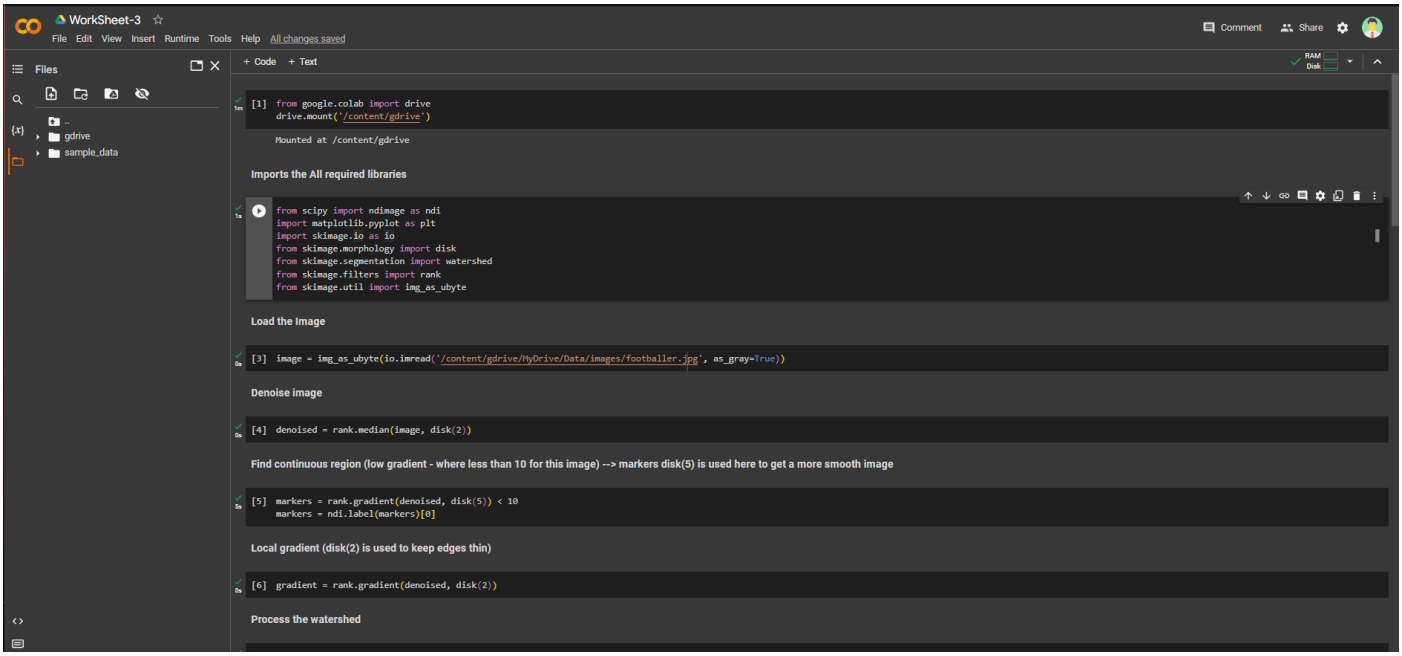
Implementation:

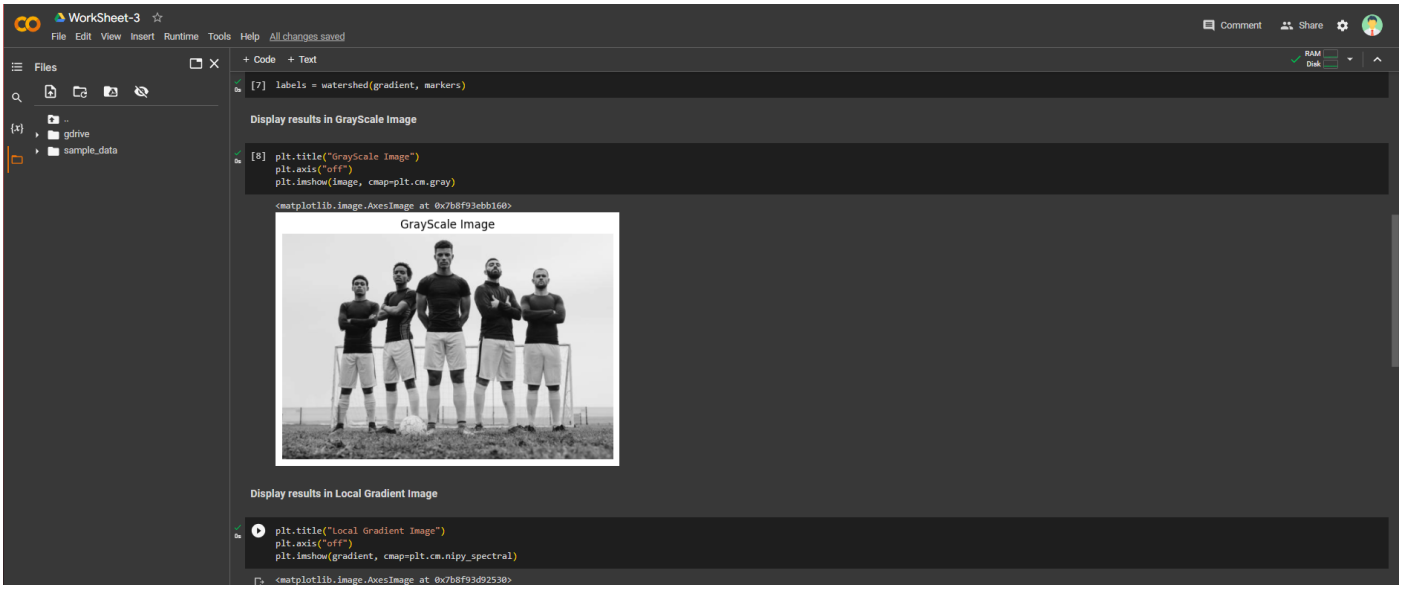
```
from google.colab import drive
drive.mount('/content/gdrive')

"""**Imports the All required libraries**"""

from scipy import ndimage as ndi
import matplotlib.pyplot as plt
import skimage.io as io
from skimage.morphology import disk
from skimage.segmentation import watershed
from skimage.filters import rank
from skimage.util import img_as_ubyte

"""**Load the Image**"""
image = img_as_ubyte(io.imread('/content/gdrive/MyDrive/Data/images/footballer.jpg',
as_gray=True))

"""**Denoise image**"""
denoised = rank.median(image, disk(2))

"""**Find continuous region (low gradient - where less than 10 for this image) --> markers
disk(5) is used here to get a more smooth image**"""
markers = rank.gradient(denoised, disk(5)) < 10
```

```
markers = ndi.label(markers)[0]

"""**Local gradient (disk(2) is used to keep edges thin)**"""
gradient = rank.gradient(denoised, disk(2))

"""**Process the watershed**"""
labels = watershed(gradient, markers)

"""**Display results in GrayScale Image**"""
plt.title("GrayScale Image")
plt.axis("off")
plt.imshow(image, cmap=plt.cm.gray)

"""**Display results in Local Gradient Image**"""
plt.title("Local Gradient Image")
plt.axis("off")
plt.imshow(gradient, cmap=plt.cm.nipy_spectral)

"""**Display results in Marker Image**"""
plt.title("Marker Image")
plt.axis("off")
plt.imshow(markers, cmap=plt.cm.nipy_spectral)

"""**Display results in Segmented Image**"""
plt.title("Segmented Image")
```

```
plt.imshow(image, cmap=plt.cm.gray)
plt.imshow(labels, cmap=plt.cm.nipy_spectral, alpha=.5)
```

Output:





CourseName: Computer Vision Lab **Course Code:** CSP-422

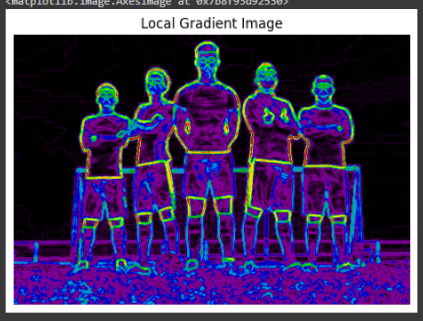
Help All changes saved

Comment Share

+ Code + Text

RAM Disk

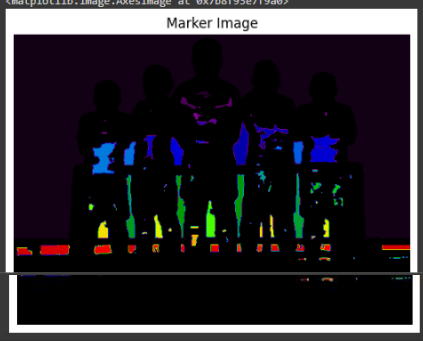
```
<matplotlib.image.AxesImage at 0x7b8f93d92530>
```



Display results in Marker Image

```
[10] plt.title("Marker Image")
     plt.axis("off")
     plt.imshow(markers, cmap=plt.cm.nipy_spectral)
```

```
<matplotlib.image.AxesImage at 0x7b8f93e7f9a0>
```



Display results in Segmented Image

```
[11] plt.title("Segmented Image")
     plt.imshow(image, cmap=plt.cm.gray)
     plt.imshow(labels, cmap=plt.cm.nipy_spectral, alpha=.5)
```

```
<matplotlib.image.AxesImage at 0x7b8f93df1420>
```