Computer Vision lab 1 (2023)

Gayatri Sharma (079MSISE03), Krishna Bahadur Ojha (079MSISE04)

***Abstract*—This document is a lab report on the lab session of Computer Vision on the study of Computer Vision on the process of visualising the image properties and image filtering. Additionally, different functions of OpenCV and other libraries are used under the python programming language.**

***Index Terms*—grayscale, pixel, matrix, Region of interest (ROI)**

I. INTRODUCTION

Image processing basically means performing processes on an image with the help of software. The goal of applying processes like smoothing, sharpening, contrasting, stretching etc on an image can be to increase its readability or to enhance its quality or even transform the image. That being said, image processing is a part of computer vision. Computer vision is an extraordinarily powerful field in the world of artificial intelligence and it has got enormous amounts of applications in real time. These include detecting license plates, scanning whiteboard contents, detecting text in still images, re-scaling images, detecting templates in images, image recognition, image retrieval, image restoration etc and so many more.

Open-CV is an open-source computer vision library developed by Intel for real-time image & video analysis and processing. Primarily written in C++, this library has bindings for Python, Java, MATLAB, Octave etc. Open-CV combined with python makes image/video analysis and processing astonishingly simple and for many, it can also be the first step in the world of Computer Vision.

II. IMAGE PROPERTIES

1) Pixel

Pixel is a small square or rectangular element that contains a specific colour value and combines to form an entire image. Each pixel represents one information point and contributes to the overall image composition.

2) grayscale image

A grayscale image is a digital image in which each pixel is represented by a single sample value indicating the level of brightness or intensity at a given location. The most common display is to use 8 bits per pixel, resulting in 256 possible Gray levels from 0 (black) to 255 (white).

Grayscale images reduces the memory requirements and computational complexity for image storage, transmission, and processing, making them particularly suitable for applications with limited resources or where colour is not relevant, such as medical imaging, document analysis, and computing. visual tasks.

3) colour channels

colour channels refer to the separate components that make up a colour image. Most colour images are represented using the RGB colour model, which stands for red, green, and blue. In this model, each pixel in the image is composed of three-colour channels: red, green, and blue.

4) rows and column

Rows and Columns refer to the organization and arrangement of pixels in an image. The image is divided into a grid pattern with rows running horizontally and columns running vertically.

The number of lines in an image corresponds to its height or vertical dimension. Represents the total number of rows of pixels from the top to the bottom of the image. Each row consists of a series of pixels aligned horizontally.

5) image intensity

Image intensity refers to the brightness or grayscale value associated with each pixel in the image. Represents the level of light or dark at a specific location within the image.

The intensity value of each pixel typically ranges from 0 to 255, with 0 representing black (minimum intensity) and 255 representing white (maximum intensity). Mid-intensity values between 0 and 255 represent different shades of grey, with lower values indicating darker shades and higher values indicating lighter shades.

6) image size

Image size refers to the dimensions or resolution of an image, usually measured in pixels. Represents the width and height of the image in terms of the number of pixels along each dimension.

Image size is usually specified using two numbers, such as "width x height" or "horizontal pixels x vertical pixels". For example, a 1024x768 image means it is 1024 pixels wide and 768 pixels tall.

III. REGION OF INTEREST

Region of interest (ROI) refers to a specific area or part of an image or video that is selected for further analysis or processing. It represents a subset of the overall image or video that is considered relevant or interesting for a particular task or application.

ROI selection is a fundamental step in many computer vision tasks, such as object detection, object tracking, and image segmentation. By defining a region of interest, a computer vision algorithm can focus its processing efforts on a specific area, which can help improve efficiency and accuracy.

There are several ways to find the region of interest in computer vision, some popular methods are listed below.

1) Manual Selection

Users can manually draw a bounding box or polygon around the desired area using interactive tools. This approach allows for precise control, but can be time-consuming for large data sets.

2) Object localization

Object localization techniques, such as the use of object detectors or key point detectors, can automatically identify and locate objects within an image. These algorithms output bounding boxes or key point coordinates that represent the region of interest around the detected objects.

3) Fixed or Predefined Region

In some cases, the area of interest may be predetermined based on prior knowledge of the scene or task. For example, in face detection, the region of interest may be a predefined region where faces are expected to occur.

4) Motion based selection

In applications such as video surveillance or surveillance, the region of interest can be determined dynamically based on motion analysis. Moving objects or areas of significant change can be considered areas of interest.

IV. COLOUR QUANTIZATION

Colour quantization in computer vision refers to the process of reducing the number of distinct colours in an image while preserving its visual appearance. It is commonly used to reduce the memory requirements or computational

complexity of image processing tasks such as compression, indexing, or analysis.

The goal of colour quantization is to represent an image using a reduced colour palette, also known as a colour palette or colour map. By replacing similar colours with representative colours from the palette, the image can be efficiently compressed or processed with fewer bits per pixel. This reduction in colour information can lead to more efficient storage and faster processing without significant loss of visual quality.

Some commonly used quantization techniques are listed below.

1) Uniform Quantization

This method divides the colour space into a fixed number of regions or bins and assigns each pixel the closest representative colour in the colour palette. This is a simple and quick technique, but can result in visible artifacts if the palette size is too small.

2) K-means Clustering

K-means is a popular clustering algorithm that can be used for colour quantization. Iteratively divides the colours into k clusters, where k represents the desired number of colours in the palette. Cluster centroids serve as representative colours and pixels are assigned to the nearest centroid.

3) Median Cut

The mid-slice algorithm recursively divides the colour space into smaller regions based on the colour distribution. The colours in each area are then represented by the average colour or median colour. This approach can produce better results than uniform quantization by adaptively partitioning the colour space.

4) Octree Quantization

The octree algorithm organizes colours into a hierarchical tree structure. Starting with the entire colour space, the tree is successively divided into octants until the specified number of colours is reached. Each leaf node represents a colour and similar colours are grouped together. Octree quantization can yield good results with reduced memory requirements, but can be computationally intensive.

V. LITERATURE REVIEW

1) Image Colour to grayscale conversion

Image Colour to grayscale conversion [1] have designed a algorithm to perform RGB approximation, reduction, and addition of chrominance and luminance to preserve the features of the colour image such as contracts, sharpness, shadow, and image structure. The algorithm calculates the luminance and chrominance values of the source colour image and then a approximation value is generated by using RGB values of the Image Channel. The average of RGB component is used to represent the resulted grey scale image. They have found that the algorithm is very helpful in those application where high quality grey scale image is required.

2) Comparison and optimization of different method of colour quantization

Comparison and optimization of different methods of colour image quantization [2], have compared different quantization methods to test the performance. They have introduced a new colour space called H1 H2 H3, which improves the quantization heuristics. They have found that the Look-up table approach method of quantization is faster while comparing among Taylor expansion, Newton-Raphson and Look-up tables method.

VI. METHODOLOGY

A. LIBRARY USED

1) *Open Computer Vision (OpenCV)*

OpenCV is a library of programming functions mainly used for real-time computer vision problems. It provides a high level of tools to customize and analyse the image and video properties.

2) *Numpy*

It is a fundamental package for scientific computing with python. It is used as a multidimensional container of generic data.it is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

B. FUNCTIONS USED

1) ***imread:*** is used to load the image from a specific file..

Syntax: cv2.imread(filename,flag)

Parameter:

- **filename:** The path to the image file.
- **flag:** The flag specifies the way how the image should be read.

❖ **cv2.IMREAD_COLOUR:** - It loads a colour image and transparency is neglected. We can pass integer value 1 for this flag.

❖ **cv2.IMREAD_GRAYSCALE**- it loads an image in grayscale mode. We can pass integer value 0 for this flag.

❖ **cv2.IMREAD_UNCHANGED** - It load an image including alpha channel. We can pass integer value -1 for this flag.

Return Value: This function returns a NumPy array if the image is loaded successfully.

2) ***imshow*** : it loads an image in the BGR (Blue-Green-Red) format.

Syntax: cv2.imshow(window_name,image)

Parameters:

window_name: A String representing the name of the window in which image to be displayed.

image: It is the image that is to be displayed.

Return Value: It doesn't return anything.

3) ***waitKey :*** Display image with a time limit and if 0 (zero) is passed as a parameter image will display until the key is pressed.

4) ***destroyWindow:*** It is used to close a particular window.

Syntax: cv2.destroyWindow(window_name)

Parameter:

- **window_name:** name of the window which you want to destroy

Return: This function doesn't return anything

5) ***Imwrite:*** It is used to save an image to any storage device.

Syntax: cv2.imwrite(filename, image)

Parameters:

filename: A string representing the file name. The filename must include image formats like .jpg, .png, etc.

image: It is the image that is to be saved.

Return Value: It returns true if the image is saved successfully.

C. OBSERVATION

OpenCV library is used to observe the different features of the image and perform different modification on it. The observations are listed below.

1) Load Image

A imread method of OpenCV library is used to load the colour image in coloured format and grayscale format.

eg:

```
import cv2
cv2.imread(imagePath, imageFormat)
```

In the above pseudo code, the imagePath is relative path of the original image and imageFormat is the loaded image format. cv2.IMREAD_GRAYSCALE is passed in imagePath to get gray scale image else the output is coloured image.

2) Display image in new window

A imshow method of OpenCv is used to show the loaded original image in the form of colourfull and grayscale format on the new window.

eg.

```
import cv2
image = cv2.imread(imagePath, imageFormat)
cv2.imshow("image", image)
cv2.waitKey(0)
```

In above code, the first argument of imshow method is a new window name and the second parameter is image original image object. The method waitKey is used to display the window for a certain time. It takes time in millisecond as a parameter. 0 is passed as an argument in waitKey to display the window until user press the any key.

3) print rows, columns, channel of image

Rows, Columns and channel of the image is obtained from the object of the original image that is loaded using OpenCV. The Shape of the image is displayed in Tuple.

eg:

```
import cv2
image = cv2.imread(imagePath)
imageSize = image.shape
imageDimention = shape[0]*shape[1]*shape[2]
print("Image size:", imageSize)
print("Image Dimention", imageDimention)
```

In above pseudo code, the shape of the image is obtained in the form of tuple. The dimension of the image of different channels are calculated by multiplying width, height and the channels of the image.

4) print intensity of the image at (100, 100) coordinate

The intensity of the image is obtained from the object of the image that is generated by loading the original image. The intensity of the image at point 100, 100 for red, green and blue is obtained by adding the coordinate in the image matrix.

eg:

```
import cv2
image = cv2.imread(imagePath)
print('Intensity at (100, 100)', image[100, 100])
```

In the above pseudo code, the intensity value of red, green, blue of the pixel at 100, 100 is displayed in array format. Where the first, second and the third elements are intensity of red, green, blue respectively.

5) modifying the colour at certain region and save the image

The original image of size width 342 and height 548 is modified to show green coloured square at the top left and red coloured square at top right corner by suppressing the remaining colour code. The green colour square is generated by suppressing the red and blue colour and green is set to 255. i.e (0, 255, 0). The red colour square is generated by suppressing the red and green colour. i.e (0, 0, 255).

eg:

```
import cv2
image = cv2.imread(imagePath)
image[0:100, 0:100] = (0, 255, 0)
image[0:100, 448:548] = (0, 0, 255)
cv2.imshow('modified_image', image)
cv2.imwrite('modified_image.jpg', image)
cv2.waitKey(0)
```

In above pseudo code, the first 100 pixels from left to right and top to bottom colour value of red and blue is suppressed to zero and the green is raised to 255 to produce green square. Similarly the green and blue colour is suppressed and red colour is raised to 255 for last 100 pixel at right to generate red square. The modified image is saved in the local device as a jpg format.

6) image channel suppression

The original image of size 342 and 548 is loaded in it's original form. The each channel of coloured image is suppressed by using available methods of OpenCV library and numpy.

In the first observation, the red colour channel is suppressed by assigning the value of read channel to zero and the output is displayed on the window.

eg:

```
import cv2
import numpy as np
image = cv2.imread(imagePath)
shape = image.shape
image[:,  :, 2] = np.zeros([shape[0], shape[1])
cv2.imshow('redSuppressed', image)
cv2.waitKey(0)
```

In the second observation, the blue colour channel is suppressed by assigning the value of blue channel to zero and the output is again displayed to a new window.

eg:

```
import cv2
import numpy as np
image = cv2.imread(imagePath)
shape = image.shape
image[:,  :, 0] = np.zeros([shape[0], shape[1])
cv2.imshow('blueSuppressed', image)
cv2.waitKey(0)
```

In the third observation of channel suppression, the green channel is suppressed by assigning the value of green channel to zeros.

eg:

```
import cv2
import numpy as np
image = cv2.imread(imagePath)
shape = image.shape
image[:,  :, 1] = np.zeros([shape[0], shape[1])
cv2.imshow('greenSuppressed', image)
cv2.waitKey(0)
```

In the last observation the channel of blue and green is suppressed and displayed the changes of image on a new window.

eg:

```
import cv2
import numpy as np
image = cv2.imread(imagePath)
shape = image.shape
image[:,  :, 0] = np.zeros([shape[0], shape[1])
image[:,  :, 1] = np.zeros([shape[0], shape[1])
cv2.imshow('blueGreenSuppressed', image)
cv2.waitKey(0)
```

7) Extract the region of interest of image and place at different location

The image of messi with football is used to perform the extraction of region of interest. The ball of the image is pointed as a region of interest and the goal is to place the ball just above the head of the messi. The observation is performed by loading the original image of messi by using imread method of OpenCv, then the region of extracted. Multiple random experiment the coordinate of region of interest is found that the region of interest is at 285 pixel form top to at 340

pixel bottom in x - axis and 335 pixel at the left to the 390 pixel in y - axis. The region of interested is extracted from the above coordinate and replaced the image pixel of destination place by the pixels of the region of interest. The final modified image is then displayed to observe the modification.

eg:

```
import cv2
image = cv2.imread(imagePath)
roi = image[285:340, 335:390]
Image[10:65, 210: 265] = roi
cv2. imshow('roi', image)
cv2.waitKey(0)
```

In the above pseudo code, the image is loaded by using imread method of OpenCV, then the region of interest is extracted. In our case the region of interest is football. The extracted region of interest is placed just above the head of the messi. The modified image of the messi with ball is now changed with two ball one at the top of the head of the messi and another near foot. The image is displayed a new window.

8) colour quantization of image

The colour quantization experiment is performed for two level, four level and eight level.

The two level colour quantization of image is performed by dividing the image pixel intensity into two category. The intensity value of pixel below 128 is assigned to zero and the intensity value of pixel above 128 is set to 128 for all three channel. Then quantized image is displayed in a new window to observe the modification due to two level quantization.

eg:

```
import cv2
img = cv2.imread(imagePath)
for k in range(img.shape[2]):
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if img[i][j][k] < 128:
                img[i][j][k] = 0
            else:
                img[i][j][k] = 128
cv2.imshow('twoLevelQuantization, img)
cv2.waitKey(0)
```

In the second experiment of colour quantization, the intensity of image between 0 to 255 is divided into four section. The first section includes the intensity of pixels between 0 to 63, the second section inclues the intensity of pixels between 64 to 127, the third section consists the intensity of pixels between 128 to 191 and the remaining intensity of pixel lies on the section 4. The intensity of the pixel of each section is set to 0, 64, 128 and 192 respectively. The operation is performed by using 3 for loop to quantized the all three channels.

eg:

```
 import cv2
img = cv2.imread(imagePath)
for k in range(img.shape[2]):
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if 0 < img[i][j][k] <= 63:
                img[i][j][k] = 0
            elif 64 < img[i][j][k] <= 127:
                img[i][j][k] = 64
            elif 128 < img[i][j][k] <= 191:
                img[i][j][k] = 128
            else:
                img[i][j][k] = 192
cv2.imshow('fourLevelQuantization, img)
cv2.waitKey(0)
```

In the third experiment of colour quantization, the intensity of image between 0 to 255 is divided into eight section. The first section includes the intensity of pixels between 0 to 33, the second section inclues the intensity of pixels between 34 to 63, the third section consists the intensity of pixels between 64 to 95, the fourth section includes the intensity of pixels between 96 to 127, the fifth section includes the intensity of pixels between 128 to 159, the sixth section includes the intensity of pixels between 160 to 191, the seventh section includes the intensity of pixels between 192 to 223 and the remaining intensity of pixel lies on the section eight. The intensity of the pixel of each section is set to 0, 32, 64, 96, 128, 160 and 192 respectively. The operation is performed by using 3 for loop to quantized the all three channels.

eg:

```
import cv2
img = cv2.imread(imagePath)
 for k in range(img.shape[2]):
   for i in range(img.shape[0]):
      for j in range(img.shape[1]):
         if 0 < img[i][j][k] <= 33:
            img[i][j][k] = 0
         elif 34 < img[i][j][k] <= 63:
            img[i][j][k] = 32
         elif 64 < img[i][j][k] <= 95:
            img[i][j][k] = 64
         elif 96 < img[i][j][k] <= 127:
            img[i][j][k] = 96
         elif 128 < img[i][j][k] <= 159:
            img[i][j][k] = 128
         elif 160 < img[i][j][k] <= 191:
            img[i][j][k] = 160
         elif 192 < img[i][j][k] <= 223:
            img[i][j][k] = 192
         else:
            img[i][j][k] = 224
cv2.imshow('eightLevelQuantization, img)
cv2.waitKey(0)
```

VIII. DISCUSSION

From this lab we observed that the OpenCV library is a great tool to deal with image and video format and its analysis. It showcased their versatility and effectiveness in various image processing tasks. By using these tools we were able to manipulate images, enhance their quality and extract the valuable features. This discussion section provides an overview of our findings and highlights some potential applications and limitations of OpenCV functions.

VII. RESULT

Throughout the lab experiment, we successfully implemented various functions provided by OpenCV and observed their impact on image processing tasks. We obtained visually appealing results by applying different functions to image.

- Feature detection and description: Feature detection like size, number of rows, columns, and channels of image; datatype used to represent the pixel intensities and finding the pixel intensities at particular coordinates.
- Image thresholding : The thresholding functions provided by OpenCV enabled us to segment images by separating objects or regions of interest from the image. Thresholding is an essential step in many computer vision applications, such as object detection and image segmentation
- Image Manipulation: The image manipulation function, such as suppressing certain colour channels and the channel intensity in the pixel.

The lab experiment focused on analyzing the size, rows, columns, channels of image also focused on accessing and printing pixel intensities at a specific

coordinate , extracting a region of interest (ROI) of an image and performing a two-level quantization on the image using OpenCV. The discussion section will cover the significance of these image attributes and their practical applications.

Understanding the size of an image is essential in image processing tasks as it provides information about the memory requirements and the amount of data being processed.

The number of rows and columns in an image represents its spatial dimensions. The rows indicate the vertical resolution or height of the image, while the columns indicate the horizontal resolution or width. These attributes are crucial for operations that require traversing or accessing individual pixels of an image, such as image manipulation, filtering, and feature extraction. Knowing the dimensions of an image allows us to define regions of interest, apply transformations, or perform calculations based on specific image regions.

The number of channels in an image refers to the colour channels or colour depth. Commonly encountered channel configurations are grayscale images (1 channel) and RGB images (3 channels). Grayscale images represent pixel intensity values ranging from black to white, while RGB images store pixel values for red, green, and blue channels, enabling the representation of a wide range of colours. The knowledge of the number of channels is crucial for tasks that involve colour analysis, object recognition, or applying colour-based filters to images.

Accessing and printing pixel intensities at specific coordinates provide valuable insights into the colour composition of an image. Understanding these intensities enables colour analysis, object detection, image processing, and various computer vision applications. By leveraging this information effectively, we can develop robust algorithms, enhance image processing techniques, and gain a deeper understanding of the visual content within images.

analyzing the size, rows, columns, and channels of an image provides valuable information for efficient memory management, image resizing, ROI selection, and colour-based analysis. These attributes are essential for various image processing tasks and play a crucial role in developing robust and accurate computer vision algorithms and applications. Understanding and utilizing this information effectively enables us to perform targeted operations, extract meaningful features, and gain valuable insights from images.

Extracting a region of interest (ROI) from an image is a crucial operation in image processing and computer vision. ROI extraction allows us to isolate relevant regions, focus on specific areas of interest, and perform targeted analysis or processing. OpenCV's efficient algorithms and flexible functions make ROI extraction a straightforward and effective task, enabling various practical applications in object recognition, image segmentation, region-specific analysis, and data augmentation.

Performing two-level quantization on an image offers numerous benefits and practical applications. Quantization simplifies image representation, reduces memory requirements, enhances image segmentation, and can be used for image compression and noise reduction. OpenCV's optimized algorithms, flexibility, and seamless integration with other image processing functions make it a powerful tool for performing quantization effectively and efficiently.

IX. CONCLUSION

This lab report aimed to explore the functions used in OpenCV Python, a powerful computer vision library widely used for image processing and video processing tasks. Throughout the experiment, we learned about the various essential functions provided by the OpenCV that enable us to perform a range of operations, such as image manipulation, filtering, feature detection and object recognition.
We began to understand the fundamental concept of an image and its representation as a matrix of pixels. We then delved into basic functions of loading, displaying, and saving images using OpenCV. This knowledge formed the foundation for our subsequent exploration of more advanced functionalities.

We explored the image size, rows, columns and channel of the image. We learned about the datatype used to represent the pixel intensity of the image and loaded them in a particular coordinate of colour image. Also, learned to modify the colour of certain region of the image.
Moreover, we learned about the importance of the thresholding in image segmentation which involve separating objects or region of interest from the image. OpenCV offers various thresholding functions that allows us to binarize images based on different criteria, enabling us to extract desired information effectively.

As we conclude this lab report, we acknowledge the vast potential that OpenCV Python offers for further exploration and experimentation. It will undoubtedly contribute to advancements in computer vision and its diverse applications in various industries.

Figure 1: Original Image

Figure 2: Modified Image

Figure 3: Blue Channel suppressed Image

X. APPENDIX

The source code of the observation is attached in the below link:

https://github.com/krishnabojha/Computer_Vision_lab/tree/master/Lab1

Figure 4: Green Channel Suppressed

Figure 5: Green and Blue Channel Suppressed

Figure 6: Two level quantization

Figure 7: Forth level quantization

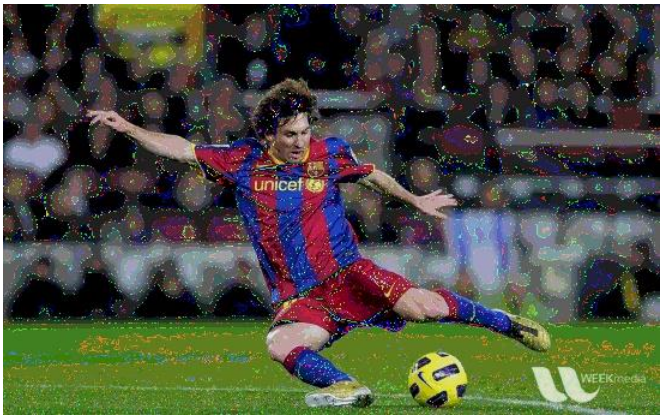

Figure 8: Eight level quantization

REFERENCES

[1] C. Saravanan, Ph.D., Assistant Professor, National Institute of Technology, "Computer Engineering and Applications"

[2] Jean-Pierre Braquelaire, Luc Brun, "Comparison and Optimization of Methods of Colour Image Quantization" in IEEE Transactions on Image Processing, 1997

[3] https://docs.opencv.org/3.4/index.html