Computer Vision

1.- Introduction

1 Installation of OpenCV

1.1 Installing OpenCV in Google Colab

Google Colab is a powerful platform for running Python code in a collaborative and interactive environment. Follow these steps to install OpenCV in Google Colab:

1.- Open a new Colab notebook:
Go to Google Colab and create a new notebook by clicking on "New Notebook."

2.- Mount Google Drive (optional):
If your images or data are stored in Google Drive, you may want to mount Google Drive to your Colab notebook. Run the following code in a cell:

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

Follow the link provided to get the authorization code and paste it into the cell.

3.- Install OpenCV:
Run the following code in a cell to install OpenCV:

```
!pip install opencv-python
```

This command uses the pip package manager to install the OpenCV Python package.

3.- Import OpenCV:
In a new cell, import the OpenCV library:

```
import cv2
```

If there are no errors, OpenCV is successfully installed.

4.- Test OpenCV:
To test if OpenCV is working, you can load an image and display it. You can use an image from the web or upload one to your Colab environment.

```
import urllib.request
import numpy as np
from matplotlib import pyplot as plt

# Load an image from the web
img_url = "url of an image"
img_array = np.asarray(bytearray(urllib.request.urlopen(img_url).
                                 read()), dtype=np.uint8)
img = cv2.imdecode(img_array, -1)

# Display the image
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis('off')
plt.show()
```

1.2 Installation in PC/Mac

OpenCV is the most widely used library for image processing and can be installed on Windows, Linux, MacOS, iOS, and Android. It can also be integrated with ROS, making OpenCV widely used in the field of robotics. It is available for C++, Python, and Java.

In the Computer Vision course, we will use OpenCV 4.8 in Python 3. To conduct the practices in the labs, we will work on Linux (recommended), although you can also perform the exercises on MacOS or Windows.

There are two options for installing the necessary software for the course, both on Linux and MacOS or Windows: miniconda (recommended) or direct installation.

1.2.1 Installation with miniconda (recommended)

This type of installation ensures that the Python libraries required for the course do not interfere with the versions of other libraries already installed on the system.

For this option, you should install miniconda using this link: `https://docs.conda.io/en/latest/miniconda.html`.

Important: Download the version corresponding to your operating system that specifies Python 3.11. If you are using Linux, you need to give execution permissions to the downloaded file from a terminal, for example:

```
chmod +x ./Miniconda3-latest-Linux-x86_64.sh
```

Once you have the permissions, you can install conda:

```
./Miniconda3-latest-Linux-x86_64.sh
```

After installing conda, you need to restart the terminal. If everything goes well, you should see the word "(base)" at the beginning of the command line.

Next, you have to run (only once) from the terminal:

```
conda create -n vision python=3
```

This creates a Python 3 environment. Each time you start a new terminal to write Computer Vision code, you'll need to activate the environment with this command:

```
conda activate vision
```

Once inside the environment, you can install Python libraries (this only needs to be done once since they are installed for that environment) or Linux packages with apt-get. These will be installed only for the environment:

```
pip3 install opencv-contrib-python numpy matplotlib pandas scikit-
                                    image scikit-learn
```

You can exit the environment with the following command:

```
conda deactivate
```

1.2.2 Direct Installation

From the terminal (with Python 3 previously installed), you can execute directly:

```
pip3 install opencv-contrib-python numpy matplotlib pandas scikit-
                                    image scikit-learn
```

The advantage is that you won't have to change the environment every time you open a new terminal, but the disadvantage is that if you have other courses (or, in general, other software) that requires different versions of any of these libraries, there may be compatibility issues.

1.2.3 Editing Python Code

You can use any editor for Python, but it is recommended to install Visual Studio Code. This software is available for Linux, Mac, and Windows.

2 Introduction to OpenCV

In this section, we will explore the basic functionalities of OpenCV: loading an image or a video, displaying it on the screen, and saving files. All the code is used in Google Colab.

2.1 Loading and Displaying Images

Let's check if the installation has been done correctly by running the following example program.

```
import cv2
from matplotlib import pyplot as plt

# Load the image specified by the user (default is in color)
```

```
image_path = 'image.jpg'  # Change the path if your image is in a
                                different directory
img = cv2.imread(image_path)

# Check if the image was successfully loaded
if img is None:
    print("Error loading the image", image_path)
else:
    # Display the image using matplotlib instead of cv2.imshow
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.title('title of the image')
    plt.axis('off')
    plt.show()
```

More information about methods:

- `cv2.imread()`: This function reads an image from a file. The image is loaded in BGR format (Blue, Green, Red).
  - **Arguments:**
    - `image_path`: The path to the image file.
  - **Returns:** The loaded image as a NumPy array.
- `print()`: Outputs a message to the console. In this case, it prints an error message if the image loading fails.
- `plt.imshow()`: Displays an image using Matplotlib.
  - **Arguments:**
    - `cv2.cvtColor(img, cv2.COLOR_BGR2RGB)`: Converts the BGR image to RGB format, which is suitable for Matplotlib.
    - `plt.title()`: Sets the title of the displayed image.
    - `plt.axis('off')`: Turns off the axis labels and ticks.
  - `plt.show()`: Displays the image.
- **Additional Arguments:**
  - `Custom Image Path`: You can specify a different image path by changing the value of `image_path`.
  - `Display in Grayscale`: You can display the image in grayscale using `cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)`.
  - `Display with Different Colormap`: If working with a grayscale image, you can use a different colormap with `plt.imshow(img, cmap='gray')`.
  - `Custom Window Title`: You can set a custom window title using `plt.title('Custom Title')`.

2.1.1 Loading Images

The next instruction loads an image into a matrix using the filename passed as a parameter.

```
image = cv2.imread('image.jpg')
```

As you know, digital images are represented by matrices.
The main image formats supported by OpenCV are:

- Windows bitmaps (bmp, dib)
- Portable image formats (pbm, pgm, ppm)
- Sun rasters (sr, ras)

It also supports other formats through auxiliary libraries:

- JPEG (jpeg, jpg, jpe)
- JPEG 2000 (jp2)
- Portable Network Graphics (png)
- TIFF (tiff, tif)
- WebP (webp)

The 'imread' function has an optional parameter. When loading a grayscale image, we must use `'IMREAD_GRAYSCALE'`:

```
# Load the image (whether it's in color or not) in grayscale
img = cv2.imread('lena.jpg', cv2.IMREAD_GRAYSCALE)
```

This is because the default option is `'IMREAD_COLOR'`, so the image will be loaded with 3 channels regardless of whether it is grayscale or not. These are the three options that can be used with 'imread':

- `'cv.IMREAD_GRAYSCALE'`: Load the image in grayscale.
- `'cv.IMREAD_COLOR'`: Load the image in color. If it has an alpha channel (transparent), it is ignored.
- `'cv.IMREAD_UNCHANGED'`: Load the image as it is, including the alpha channel if it has one.

In general, when you need help on the syntax of any OpenCV function, you can write directly in Python: 'help(cv2.imshow)', replacing 'imshow' with the name of the function you want help with.

Note that in a typical OpenCV script, you would use the cv2.imshow() function to display an image. However, if you are working in a Jupyter Notebook or a similar environment, the display may not work as expected (you must use 'plt.imshow(img)')

2.1.2 Images in OpenCV

In Python, OpenCV uses NumPy arrays to store images. To use this library in your code, you have to import it at the beginning:

```
    import numpy as np
```

Once imported, you can create a matrix of any size, whether it contains image data or not:

```
img = np.full((100, 100, 3), (0, 0, 255), dtype=np.uint8)
print(img)
```

In this example, we created a 100x100x3 matrix (which could correspond to an image with 100 rows by 100 columns and 3 channels) initialized with the red value '(0, 0, 255)' and of type 'uint8' (8 bits). This data type is standard for creating 8-bit depth images in Python (in C++, it's 'uchar' instead of 'uint8'). With 8 bits, values ranging from 0 to 255 can be represented.

If you visualize this image with 'imshow', you will see that it is red. This is because the red value is represented in the last channel since OpenCV loads images in BGR mode instead of the standard RGB. In other words, channel 0 is blue, channel 1 is green, and channel 2 is red.

In addition to the 'uint8' type, which is the most common for images, a NumPy array can be of any of these types.

There are several alternatives to assign values to an already created matrix, for example:

```
img.fill(255)  # only if the image is 1 channel
img[::] = (255, 0, 0)  # To change all values to (255, 0, 0)
```

For more information on the NumPy array access syntax, you can refer to the documentation.

If we wanted to initialize all values of the matrix to 0, 1, or any other value, we could indicate it:

```
img = np.zeros((100, 100, 3), dtype=np.uint8)  # Initialization
                                    with zeros
img = np.ones((100, 100, 3), dtype=np.uint8)  # Initialization with
                                     ones
img = np.array([[[255, 0, 0], [255, 0, 0]],
                [[255, 0, 0], [255, 0, 0]],
                [[255, 0, 0], [255, 0, 0]]], dtype=np.uint8)  #
                                                    Initialization of
                                                     a 3x2x3 matrix
                                                    with all pixels
                                                    in blue
```

In Python, to copy a variable to another, you must be careful when using the equal symbol:

```
x = img
y = np.copy(img)
```

If we modify 'img' after executing this code, the value of 'x' will also change, but not that of 'y'. This is because the assignment operator ('=') does not make a copy of the matrix but creates a pointer pointing to the variable. To make a copy, the 'copy' method is necessary.

To access individual values of a matrix, you can use the following options:

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])  # Create a 2x3 matrix
print(matrix[0, 0], matrix[0, 1], matrix[1, 0])  # Prints "1 2 4"
matrix[0, 0] = 2  # Changes the value of position 0,0
```

If you want information about the matrix structure, you can use the following instruction:

```
print(matrix.dtype)  # Prints the matrix type
print(matrix.shape)  # Prints the dimensions "(2, 3)"
print(matrix.ndim)   # Prints the total number of dimensions (2)
print(matrix.size)   # Prints the number of elements in the array (
                                6)
```

Many operations can be performed with NumPy arrays, such as inverting matrices, transposing them, etc.

In NumPy, it is easy to select part of an array, as shown in the following example taken from the NumPy documentation:

```
data = np.array([1, 2, 3])
print(data[1])      # 2
print(data[0:2])     # (1,2)
print(data[1:])      # (2,3)
print(data[-2:])     # (2, 3)
```

To access all elements of an array iteratively, you can use the following loop:

```
for x in data:
    print(x)
```

In the case of an image for which we want to iterate element by element:

```
for x in img:
    for y in x:
        print(y)
```

If we want to access an image using indices instead of iterators, we can use 'range':

```
rows, cols, channels = img.shape if len(img.shape) == 3 else (*img.
                                shape, 1)
for i in range(rows):
    for j in range(cols):
        print(img[i, j])
```

It is also possible to create a matrix that stores a region of interest (a rectangular area) from another image:

```
r = img[y1:y2, x1:x2]
```

Where '(x1, y1)' are the coordinates of the upper-left corner of the rectangle to be cut, and '(x2, y2)' are the coordinates of the lower-right corner.

You can try this example program to see how a subimage is extracted, this time using the 'selectROI' function that allows us to select a region of interest using the OpenCV interface:

```
img = cv2.imread('lena.jpg')
r = cv2.selectROI(img)
img_crop = img[int(r[1]):int(r[1] + r[3]), int(r[0]):int(r[0] + r[2
                                        ])]
cv2.imshow('Crop', img_crop)
cv2.waitKey(0)
```

Sometimes it is necessary to change the data type of an array or matrix. You can do this easily using NumPy types:

```
dst = np.float32(src)  # Conversion to float
dst = np.intc(src)     # Conversion to int
dst = np.uint8(src)    # Conversion to uint8
```

There are times when the type conversion cannot be done directly, for example, when converting a 'float32' matrix to 'uint8' because it may go out of range (in the first case, the variable is represented with 32 bits, and in the second with 8). To avoid this, normalization is usually applied. For example, if we have a matrix 'm' of type 'float32', we can convert it as follows:

```
# Normalize values between 0 and 255
m = m - m.min()
m = m / m.max() * 255

# Now it can be converted to uint8
dst = np.uint8(m)
```

2.1.3 Saving Images

To save an image to disk, use the 'imwrite' function of OpenCV. For example:

```
cv2.imwrite('output.jpg', img)
```

This function determines the output file format from the extension provided in its name (in this case, JPG). There is a third optional parameter where we can specify an array with writing options. For example:

```
cv2.imwrite('compress.png', img,  [cv2.IMWRITE_PNG_COMPRESSION, 9])
                                    # PNG compression level 9
```

As seen, images can be saved with 'imwrite', but there are cases where this operation may fail (for example, when trying to access a directory without permissions). If this happens, the method will return 'False'. If you want to know if the image has been saved correctly, you need to check it (it is advisable to do so always):

```
write_status = cv.imwrite('img.jpg', img)
if write_status:
    print('Image saved')
else:
    print('Error saving the image')  # Exception or other writing
                                      problem
```

2.2 Persistence

In addition to the specific functions for reading and writing images and video, OpenCV provides another generic way to save or load data. This is known as data persistence. The values of objects and variables in the program can be saved (serialized) to disk, which is useful for storing results and loading configuration data.

These data are usually saved in an XML file using a dictionary (in some programming languages like C++, dictionaries are also called maps) using key/-value pairs. For example, if we wanted to save a variable containing the number of objects detected in an image:

```
fs = cv2.FileStorage('config.xml', cv2.FileStorage_WRITE)
# Open the file for writing
fs.write('number_of_objects', num_objects) # Save the number of
                                           objects
fs.release() # Close the file
```

Assuming our variable contains the value 10, it will be stored on disk in the following config.xml file:

```
<?xml version="1.0"?>
<opencv_storage>
<number_of_objects>10</number_of_objects>
</opencv_storage>
```

If we later want to load this information from the file, we can use the following code:

```
fs = cv2.FileStorage('config.xml', cv2.FileStorage_READ)
num_objects = fs.getNode('number_of_objects')
print(num_objects.real())
```

Note: In Google Colab, the file system is different from a local environment.

2.3 Visual Elements

As we saw at the beginning, we can create a window to display an image using the 'namedWindow' function. The second parameter it takes can be:

- `'cv.WINDOW_NORMAL'`: The user can resize the window once it is displayed on the screen.

- `'cv.WINDOW_AUTOSIZE'`: The window size adjusts to the size of the image, and the user cannot resize it. It is the default option.
- `'cv.WINDOW_OPENGL'`: The window is created with OpenGL support (not necessary in this course).

Within the OpenCV window where we display the image, we can add trackbars, buttons, capture mouse position, etc. In this link: `https://docs.opencv.org/4.x/gui_components.html`, you can see methods and constants related to the management of the standard visual environment.

To capture the mouse's position, we can use the 'setMouseCallback' method, which takes three parameters:

1. The window name where the mouse is captured.
2. The name of the function to be invoked when any mouse event occurs (hovering, clicking, etc.).
3. A pointer (optional) to any object we want to pass to our function.

The callback function we created receives four parameters: the event code, the values x and y, some options (flags), and the pointer to the element passed to the function.

```
import cv2 as cv
import matplotlib.pyplot as plt

# Function invoked when the mouse is used
def mouse_click(event, x, y, flags, param):
    # If the left button is pressed
    if event == cv.EVENT_LBUTTONDOWN:
        message = 'Left Button (' + str(x) + ',' + str(y) + ')'
        # Show text on the image
        cv.putText(img, message, (x, y), cv.FONT_HERSHEY_TRIPLEX, 0
                                            .5, (255, 255, 255), 1)
        plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
        plt.title('Image with Mouse Click')
        plt.axis('off')
        plt.show()

# Load image
img = cv.imread('lena.jpg')

# Display the image using Matplotlib
plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
plt.title('Image with Mouse Click')
plt.axis('off')
plt.show()

# Create a window before setting the mouse callback
cv.namedWindow('image')

# Specify the function to call when the mouse is clicked on the
                                    window
cv.setMouseCallback('image', mouse_click)
```

More information about the parameters of 'putText' can be found in this link: `https://docs.opencv.org/4.x/da/d6e/tutorial_py_geometric_transformations.html`.

Through the 'createTrackbar' method, we can create a trackbar (also called slider) to adjust some value in the window interactively. Similar to the method that manages the mouse, it can receive a reference to a function as the last parameter (in the following example, 'onChange').

```
from google.colab import drive
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Constant to indicate the maximum value of the slider
alpha_slider_max = 100

# Function that creates the trackbar
def onChange(val):
    alpha = val / alpha_slider_max
    beta = 1.0 - alpha
    # The 'addWeighted' method blends the images
    dst = cv2.addWeighted(img1, alpha, img2, beta, 0.0)
    # Display the blended image
    plt.imshow(cv2.cvtColor(dst, cv2.COLOR_BGR2RGB))
    plt.title('Linear Blend')
    plt.axis('off')
    plt.show()

# Load images
img1 = cv2.imread('image1.jpg')
img2 = cv2.imread('image2.jpg')

if img1 is None:
    print('Could not open the image 1.jpg')
    quit()
if img2 is None:
    print('Could not open the image 2.jpg')
    quit()

# Create the window
cv2.namedWindow('Linear Blend')

# Create the trackbar
cv2.createTrackbar('Alpha', 'Linear Blend' , 0, alpha_slider_max,
                                    onChange)

# Call the function that manages what happens when the trackbar is
                                    modified
onChange(0)
```

As an alternative to using the native visual elements of the OpenCV interface, you can use more powerful libraries like imgui (`https://imgui-datascience.readthedocs.io/en/latest/`), although it may not be necessary for this course.

2.4 Video

OpenCV allows loading video files or using a webcam for real-time processing. Let's look at an example of edge detection using a webcam (it will give an error if the laboratory is not equipped with cameras, but if you have a laptop, you can try it):

```
import cv2
from google.colab import drive

# Mount Google Drive
drive.mount('/content/gdrive')

# Create a VideoCapture object for the webcam
cap = cv2.VideoCapture(0)

# Get the width and height of the frames
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Define the codec and create a VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
output_path = '/content/gdrive/My Drive/output.avi'
out = cv2.VideoWriter(output_path, fourcc, 20.0, (width, height))

while True:
    # Capture frame by frame
    ret, frame = cap.read()

    # Process the frames here
    if ret:
        # Example processing: convert to grayscale
        gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Write the frame to the output video file
        out.write(gray_frame)
    else:
        break

    # Break the loop when the user presses 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the VideoWriter and VideoCapture objects
out.release()
cap.release()
```

As seen, the code is straightforward. We just need to initialize a video capture variable, and with 'read', we can get the frames to process. If 'ret' is True, it means the frame has been read correctly.

If you want to load a video file (for example, this one), you just need to change a couple of lines:

```
cap = cv.VideoCapture('Megamind.avi')
```

To save a video file, you need to call the 'VideoWriter' function specifying the format, fps (frames per second), and dimensions. For example:

```
out = cv.VideoWriter('output.avi', fourcc, 20.0, (640, 480))  # AVI
                                    , 20fps, 640x480
```

If you try to save directly the resulting video from the previous program with 'VideoWriter', it won't work because the edges are in grayscale, and all supported video formats require color frames.

These are some of the accepted formats, although there are many more:

```
fourcc = cv.VideoWriter_fourcc('m', 'j', 'p', 'g')  # AVI,
                                    recommended in the course
fourcc = cv.VideoWriter_fourcc('d', 'i', 'v', '3')  # DivX MPEG-4
                                    codec
fourcc = cv.VideoWriter_fourcc('m', 'p', 'e', 'g')  # MPEG-1 codec
fourcc = cv.VideoWriter_fourcc('m', 'p', 'g', '4')  # MPEG-4 codec
fourcc = cv.VideoWriter_fourcc('d', 'i', 'v', 'x')  # DivX codec
```

To write a video frame, you can use the 'write' method:

```
out.write(frame)
```