1.2.1 Getting Started with Images

Goals

- Here, you will learn how to read an image, how to display it and how to save it back
- You will learn these functions : **cv2.imread()**, **cv2.imshow()** , **cv2.imwrite()**
- Optionally, you will learn how to display images with Matplotlib

Using OpenCV

Read an image

Use the function **cv2.imread()** to read an image. The image should be in the working directory or a full path of image should be given.

Second argument is a flag which specifies the way image should be read.

- cv2.IMREAD_COLOR : Loads a color image. Any transparency of image will be neglected. It is the default flag.
- cv2.IMREAD_GRAYSCALE : Loads image in grayscale mode
- cv2.IMREAD_UNCHANGED : Loads image as such including alpha channel

---

Note: Instead of these three flags, you can simply pass integers 1, 0 or -1 respectively.

See the code below:

```
import numpy as np
import cv2

# Load an color image in grayscale
img = cv2.imread('messi5.jpg',0)
```

Warning: Even if the image path is wrong, it won't throw any error, but `print img` will give you `None`

Display an image

Use the function **cv2.imshow()** to display an image in a window. The window automatically fits to the image size.

First argument is a window name which is a string. second argument is our image. You can create as many windows as you wish, but with different window names.

```
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

A screenshot of the window will look like this (in Fedora-Gnome machine):

cv2.waitKey() is a keyboard binding function. Its argument is the time in milliseconds. The function waits for specified milliseconds for any keyboard event. If you press any key in that time, the program continues. If **0** is passed, it waits indefinitely for a key stroke. It can also be set to detect specific key strokes like, if key *a* is pressed etc which we will discuss below.

cv2.destroyAllWindows() simply destroys all the windows we created. If you want to destroy any specific window, use the function **cv2.destroyWindow()** where you pass the exact window name as the argument.

Note: There is a special case where you can already create a window and load image to it later. In that case, you can specify whether window is resizable or not. It is done with the function **cv2.namedWindow()**. By default, the flag is `cv2.WINDOW_AUTOSIZE`. But if you specify flag to be `cv2.WINDOW_NORMAL`, you can resize window. It will be helpful when image is too large in dimension and adding track bar to windows.

See the code below:

```
cv2.namedWindow('image', cv2.WINDOW_NORMAL)
cv2.imshow('image',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Write an image

Use the function **cv2.imwrite()** to save an image.

First argument is the file name, second argument is the image you want to save.

```
cv2.imwrite('messigray.png',img)
```

This will save the image in PNG format in the working directory.

Sum it up

Below program loads an image in grayscale, displays it, save the image if you press 's' and exit, or simply exit without saving if you press *ESC* key.

```
import numpy as np
import cv2

img = cv2.imread('messi5.jpg',0)
cv2.imshow('image',img)
k = cv2.waitKey(0)
if k == 27:         # wait for ESC key to exit
    cv2.destroyAllWindows()
elif k == ord('s'): # wait for 's' key to save and exit
    cv2.imwrite('messigray.png',img)
    cv2.destroyAllWindows()
```

Warning: If you are using a 64-bit machine, you will have to modify `k = cv2.waitKey(0)` line as follows : `k = cv2.waitKey(0) & 0xFF`

Using Matplotlib

Matplotlib is a plotting library for Python which gives you wide variety of plotting methods. You will see them in coming articles. Here, you will learn how to display image with Matplotlib. You can zoom images, save it etc using Matplotlib.

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('messi5.jpg',0)
plt.imshow(img, cmap = 'gray', interpolation = 'bicubic')
plt.xticks([]), plt.yticks([])  # to hide tick values on X and Y axis
plt.show()
```

A screen-shot of the window will look like this :

See also:

Plenty of plotting options are available in Matplotlib. Please refer to Matplotlib docs for more details. Some, we will see on the way.

Warning: Color image loaded by OpenCV is in BGR mode. But Matplotlib displays in RGB mode. So color images will not be displayed correctly in Matplotlib if image is read with OpenCV. Please see the exercises for more details.

Additional Resources

1. Matplotlib Plotting Styles and Features

Exercises

1. There is some problem when you try to load color image in OpenCV and display it in Matplotlib. Read this discussion and understand it.

1.2.2 Getting Started with Videos

Goal

- Learn to read video, display video and save video.
- Learn to capture from Camera and display it.
- You will learn these functions : **cv2.VideoCapture()**, **cv2.VideoWriter()**

Capture Video from Camera

Often, we have to capture live stream with camera. OpenCV provides a very simple interface to this. Let's capture a video from the camera (I am using the in-built webcam of my laptop), convert it into grayscale video and display it. Just a simple task to get started.

To capture a video, you need to create a **VideoCapture** object. Its argument can be either the device index or the name of a video file. Device index is just the number to specify which camera. Normally one camera will be connected (as in my case). So I simply pass 0 (or -1). You can select the second camera by passing 1 and so on. After that, you can capture frame-by-frame. But at the end, don't forget to release the capture.

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

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

    # Our operations on the frame come here
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Display the resulting frame
    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# When everything done, release the capture
cap.release()
cv2.destroyAllWindows()
```

`cap.read()` returns a bool (True/False). If frame is read correctly, it will be True. So you can check end of the video by checking this return value.

Sometimes, `cap` may not have initialized the capture. In that case, this code shows error. You can check whether it is initialized or not by the method **cap.isOpened()**. If it is True, OK. Otherwise open it using **cap.open()**.

You can also access some of the features of this video using **cap.get(propId)** method where propId is a number from 0 to 18. Each number denotes a property of the video (if it is applicable to that video) and full details can be seen here: Property Identifier. Some of these values can be modified using **cap.set(propId, value)**. Value is the new value you want.

For example, I can check the frame width and height by `cap.get(3)` and `cap.get(4)`. It gives me 640x480 by default. But I want to modify it to 320x240. Just use `ret = cap.set(3,320)` and `ret = cap.set(4,240)`.

Note: If you are getting error, make sure camera is working fine using any other camera application (like Cheese in Linux).

Playing Video from file

It is same as capturing from Camera, just change camera index with video file name. Also while displaying the frame, use appropriate time for `cv2.waitKey()`. If it is too less, video will be very fast and if it is too high, video will be slow (Well, that is how you can display videos in slow motion). 25 milliseconds will be OK in normal cases.

```
import numpy as np
import cv2

cap = cv2.VideoCapture('vtest.avi')

while(cap.isOpened()):
    ret, frame = cap.read()

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cv2.imshow('frame',gray)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Note: Make sure proper versions of ffmpeg or gstreamer is installed. Sometimes, it is a headache to work with Video Capture mostly due to wrong installation of ffmpeg/gstreamer.

Saving a Video

So we capture a video, process it frame-by-frame and we want to save that video. For images, it is very simple, just use `cv2.imwrite()`. Here a little more work is required.

This time we create a **VideoWriter** object. We should specify the output file name (eg: output.avi). Then we should specify the **FourCC** code (details in next paragraph). Then number of frames per second (fps) and frame size should be passed. And last one is **isColor** flag. If it is True, encoder expect color frame, otherwise it works with grayscale frame.

FourCC is a 4-byte code used to specify the video codec. The list of available codes can be found in fourcc.org. It is platform dependent. Following codecs works fine for me.

- In Fedora: DIVX, XVID, MJPG, X264, WMV1, WMV2. (XVID is more preferable. MJPG results in high size video. X264 gives very small size video)
- In Windows: DIVX (More to be tested and added)
- In OSX : *(I don't have access to OSX. Can some one fill this?)*

FourCC code is passed as `cv2.VideoWriter_fourcc('M','J','P','G')` or `cv2.VideoWriter_fourcc(*'MJPG)` for MJPG.

Below code capture from a Camera, flip every frame in vertical direction and saves it.

```
import numpy as np
import cv2

cap = cv2.VideoCapture(0)

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))

while(cap.isOpened()):
    ret, frame = cap.read()
    if ret==True:
        frame = cv2.flip(frame,0)

        # write the flipped frame
        out.write(frame)

        cv2.imshow('frame',frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break

# Release everything if job is finished
cap.release()
out.release()
cv2.destroyAllWindows()
```

Additional Resources

Exercises

1.2.3 Drawing Functions in OpenCV

Goal

- Learn to draw different geometric shapes with OpenCV
- You will learn these functions : **cv2.line()**, **cv2.circle()** , **cv2.rectangle()**, **cv2.ellipse()**, **cv2.putText()** etc.

Code

In all the above functions, you will see some common arguments as given below:

- img : The image where you want to draw the shapes
- color : Color of the shape. for BGR, pass it as a tuple, eg: `(255,0,0)` for blue. For grayscale, just pass the scalar value.
- thickness : Thickness of the line or circle etc. If **-1** is passed for closed figures like circles, it will fill the shape. *default thickness = 1*
- lineType : Type of line, whether 8-connected, anti-aliased line etc. *By default, it is 8-connected.* `cv2.LINE_AA` gives anti-aliased line which looks great for curves.