LAB: Image Processing and Analysis in OpenCV with Python

https://www.coursera.org/learn/introduction-computer-vision-watson-opencv/ungradedLti/u1352/lab-image-processing-and-analysis-in-opencv-with-python

In this lab, you will learn about the OpenCV package in Python and will learn to perform various image processing techniques.

Ce cours utilise l'outil d'un tiers, LAB: Image Processing and Analysis in OpenCV with Python, pour améliorer votre expérience d'apprentissage. L'outil référence des informations de base comme votre nom, votre adresse e-mail et votre ID Coursera.

IBM Developer
SKILLS NETWORK

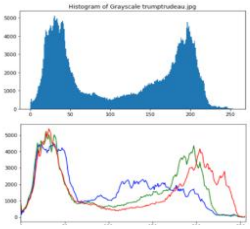
Lab - Image Processing and Analysis in Python with OpenCV

https://labs.cognitiveclass.ai/tools/jupyterlab/lab/tree/labs/CV0101EN/Image_Processing_with_OpenCV.ipynb

Welcome! After completing this lab you will:

1. Learn to download, read and display images using Python, OpenCV and Matplotlib
2. Learn to apply various Image Processing techniques using Python and OpenCV
3. Understand about the importance of various Image Processing techniques
4. Analyze images using Histograms

Table of Contents



OpenCV Homepage

- OpenCV stands for: Open Source Computer Vision Library
- includes C++, C, Python and Java interfaces

Import the OpenCV package

Did you know? The package is called `cv2` in Python. The `2` in `cv2` doesn't refer to a particular package version.

```
[ ]: import cv2
     print(cv2.__version__)
```

CODE :

```
import cv2
print(cv2.__version__)
```

Downloading and plotting images in Python using OpenCV and Matplotlib

Download an image

Let's first download an image, using the `urllib` package. We'll need to specify a `url` of an image and a `filename`.

We'll be downloading the following image of Donald Trump and Justin Trudeau, the President of the U.S. and Prime Minister of Canada, respectively.

```
[ ]: import urllib.request

     #Set the url and filename
     trumptrudeau_url = "https://upload.wikimedia.org/wikipedia/commons/7/76/Donald_Trump_Justin_Truc
     trumptrudeau_filename = "trumptrudeau.jpg"


     urllib.request.urlretrieve(trumptrudeau_url, trumptrudeau_filename) # downloads file as "trumptr
```

CODE :

```
import urllib.request

#Set the url and filename
trumptrudeau_url =
"https://upload.wikimedia.org/wikipedia/commons/7/76/Donald_Trump_Justin_Trudeau_2017-02-13_02.jpg"
trumptrudeau_filename = "trumptrudeau.jpg"

urllib.request.urlretrieve(trumptrudeau_url, trumptrudeau_filename) # downloads file as "trumptrudeau.jpg"
```

Verify that the image has downloaded

At this point, you should now see the file listed in the files directory in the left-sidebar of the JupyterLab environment. If this side menu is hidden, you can go to `View` > `View Left-Sidebar`.

Alternatively, you can run the command below to check the files in your current directory:

```
[ ]: import os
os.listdir(os.curdir) #shows all files in current directory
```

CODE
import os
os.listdir(os.curdir) #shows all files in current directory

```
[ ]: #Is trump_filename in your directory?
print(trumptrudeau_filename in os.listdir(os.curdir))
```

CODE
#Is trump_filename in your directory?
print(trumptrudeau_filename in os.listdir(os.curdir))

Plotting images in Jupyter Notebooks

Next, let's display the image into this notebook, using OpenCV. We'll also be borrowing from the plotting library, `matplotlib`, to help display the images.

```
[ ]: from matplotlib import pyplot as plt
%matplotlib inline

trumptrudeau = cv2.imread(trumptrudeau_filename)

plt.imshow(trumptrudeau)
```

CODE
from matplotlib import pyplot as plt
%matplotlib inline

trumptrudeau = cv2.imread(trumptrudeau_filename)

plt.imshow(trumptrudeau)

Image Processing using OpenCV

Hm... the Presidents are looking a little *blue* in the above photo, wouldn't you say?

Fix colors when displaying images

```
[ ]: img_corrected = cv2.cvtColor(trumptrudeau, cv2.COLOR_BGR2RGB)

plt.imshow(img_corrected)
```

CODE
img_corrected = cv2.cvtColor(trumptrudeau, cv2.COLOR_BGR2RGB)

plt.imshow(img_corrected)

Remove the axes around the image

```
[ ]: plt.axis("off") #remove axes ticks
     plt.imshow(img_corrected)
```

CODE
plt.axis("off") #remove axes ticks
plt.imshow(img_corrected)

Change the size of the displayed image in the notebook

Note that this simply changes the display size of the image, not the actual image dimensions

```
[ ]: from pylab import rcParams

     rcParams['figure.figsize'] = 10, 12

     plt.axis("off") #remove axes ticks
     plt.imshow(img_corrected)
```

CODE
from pylab import rcParams

rcParams['figure.figsize'] = 10, 12

plt.axis("off") #remove axes ticks
plt.imshow(img_corrected)

Convert to Grayscale

```
[ ]: gray_trumptrudeau = cv2.cvtColor(trumptrudeau, cv2.COLOR_BGR2GRAY)

     plt.imshow(gray_trumptrudeau, cmap = 'gray')
     plt.axis("off") #remove axes ticks
     plt.title('Grayscale Image')
```

CODE
gray_trumptrudeau = cv2.cvtColor(trumptrudeau, cv2.COLOR_BGR2GRAY)

plt.imshow(gray_trumptrudeau, cmap = 'gray')
plt.axis("off") #remove axes ticks
plt.title('Grayscale Image')

Canny Edge Detection

Canny Edge Detection is an algorithm used to detect edges in an image, and was developed by John F. Canny in 1986.
Full OpenCV documentation on Canny Edge Detection: here

```
[ ]: rcParams['figure.figsize'] = 10, 12


     edges = cv2.Canny(img_corrected,
                       threshold1=100,
                       threshold2=200)

     plt.imshow(edges,cmap = 'gray')
     plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
```

CODE

```
rcParams['figure.figsize'] = 10, 12


edges = cv2.Canny(img_corrected,
            threshold1=100,
            threshold2=200)

plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
```

Try different levels of threshold:

```
[ ]: rcParams['figure.figsize'] = 10, 12

edges = cv2.Canny(img_corrected,
                  threshold1=1,   ## try different values here
                  threshold2=200) ## try different values here

plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
```

CODE

```
rcParams['figure.figsize'] = 10, 12

edges = cv2.Canny(img_corrected,
            threshold1=1,   ## try different values here
            threshold2=200) ## try different values here

plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
```

```
[ ]: rcParams['figure.figsize'] = 10, 12

edges = cv2.Canny(img_corrected,
                  threshold1=100,   ## try different values here
                  threshold2=500) ## try different values here

plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
```

CODE

```
rcParams['figure.figsize'] = 10, 12

edges = cv2.Canny(img_corrected,
            threshold1=100,   ## try different values here
            threshold2=500) ## try different values here

plt.imshow(edges,cmap = 'gray')
plt.title('Edge Image'), plt.xticks([]), plt.yticks([])
```

Want to know the parameters of a function?

Try using a question mark `?` before a method to display the help doc for the method.

```
[ ]: ?cv2.Canny
```

CODE; ?cv2.Canny

Analyzing Images using Histograms

Histograms - Grayscale

Histograms allow you to quickly determine the number of pixels that are on a scale "black pixel" (0) to "white pixel" (255).

Execute the code cell below to see the histogram of the image of President Trump and President Trudeau:

[ ]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt

rcParams['figure.figsize'] = 8,4

plt.hist(gray_trumptrudeau.ravel(),256,[0,256])
plt.title('Histogram of Grayscale trumptrudeau.jpg')
plt.show()
```

CODE

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

rcParams['figure.figsize'] = 8,4

plt.hist(gray_trumptrudeau.ravel(),256,[0,256])
plt.title('Histogram of Grayscale trumptrudeau.jpg')
plt.show()

What's in the histogram?

Like any other histogram, this graph shows the **count** of something. In this case, it's the *number of pixels in the image* that are either *dark* (towards the left of the graph) or *lighter* (towards the right of the graph).

Why are histograms important?

- If you're concerned about whether the image is properly exposed (not too dark and not too bright), then histograms are a great way of checking the distribution of your lighter and darker pixels!
- If your images are overexposed (too bright) or underexposed (too dark), you can fix exposure with OpenCV in Python (although you may lose some image quality).
- This might be important if you're developing an app where users are taking photos and your app is trying to classify the objects inside the image. If it's too dark or too light, the objects may not be recognizable.

How do you read the histogram?

- On the **x-axis**, the values normally range from 0 (black) to 255 (white). So darker pixels are on the left, and whiter pixels are to the right. In the `matplotlib` chart above, the x-axis goes from 0 to 256, since each pixel is represented as a bin start from 0 to 0.99, and ends with 255 to 255.99.
- The **y-axis** shows the number of pixels found in the image on the scale of black to white.

Observations: The grayscale histogram above shows that:

- lots of pixels around `x = 30`, which are **darker pixels**.
- lots of pixels around `x = 200`, which are **lighter pixels**.

Interpretation:

- the image appears to have lots of very dark, and very light pixels, but not a lot of pixels in between. Looking at the image of the Presidents, this is readily apparent -- the darker pixels seem to come from the black suits and the fireplace. A lot of the lighter pixels come from the furniture and walls.

Histograms - Color scale (RGB)

Interpreting RGB histograms

Like grayscale histograms, RGB also shows the count distribution of pixels from black (`x=0`) to full color (`x=255`).

Here is a color image of the presidents:

```
[ ]: rcParams['figure.figsize'] = 10, 12

plt.axis("off") #remove axes ticks
plt.imshow(img_corrected)
```

CODE

```
rcParams['figure.figsize'] = 10, 12

plt.axis("off") #remove axes ticks
plt.imshow(img_corrected)
```

And the following is an RGB histogram for the above image

RGB histogram

```
[ ]: rcParams['figure.figsize'] = 8, 4

color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([trumptrudeau],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```

CODE

```
rcParams['figure.figsize'] = 8, 4

color = ('b','g','r')
for i,col in enumerate(color):
    histr = cv2.calcHist([trumptrudeau],[i],None,[256],[0,256])
    plt.plot(histr,color = col)
    plt.xlim([0,256])
plt.show()
```

In the histogram above, you can observe that:

- of the lighter pixels (around `x >= 200`), there are more red pixels than green or blue pixels. This is likely because of the redness in the faces of the presidents.
- there seems to be a similar amount of dark red, dark blue, and dark green pixels (around `x<50`)
- in terms of middle-tones, there seems to be more blue pixels than green or red. This is likely because of the neckties worn by the presidents.

Exercises

Exercise 1

1.1. Download an image of your choosing, and display it

1.2. Convert the image to grayscale, and display it

```
[ ]: # 1.1. Download an image of your choosing and display it

     # Write your code below:
```

```
Double-click <font color="red"><b><u>here</b></u></font> for the solution.

<!-- The answer is below:

You should specify the download_image_url and download_image_filename


download_image_url = ""
download_image_filename = ""
urllib.request.urlretrieve(download_image_url, download_image_filename)

img = cv2.imread(download_image_filename)
plt.imshow(img)
```

CODE
Double-click <u>here</u> for the solution.

<!-- The answer is below:

You should specify the download_image_url and download_image_filename

download_image_url = ""
download_image_filename = ""
urllib.request.urlretrieve(download_image_url, download_image_filename)

img = cv2.imread(download_image_filename)
plt.imshow(img)

--->

```
[ ]: # 1.2. Convert the image to grayscale and display it

     # Write your code below:
```

```
Double-click <font color="red"><b><u>here</b></u></font> for the solution.

<!-- The answer is below:

gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(gray_img, cmap = "gray")
plt.axis("off") #remove axes ticks
plt.title('Grayscale Image')

--->
```

CODE
Double-click <u>here</u> for the solution.

<!-- The answer is below:

gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
plt.imshow(gray_img, cmap = "gray")
plt.axis("off") #remove axes ticks
plt.title('Grayscale Image')

--->

Exercise 2

2.1. Using your grayscale image from Exercise 1, display its grayscale histogram
2.2. Using your color image from Exercise 1, display its color histogram
2.3. Look at your histogram and think about how its pixels are distributed. What are some observations that you can make?

[ ]:
```
# 2.1. Using your grayscale image from Exercise 1, display its grayscale histogram

# Write your code below:
```

```
Double-click <font color="red"><b><u>here</b></u></font> for the solution.

<!-- The answer is below:

rcParams['figure.figsize'] = 8,4
plt.hist(gray_img.ravel(),256,[0,256])
plt.title('Histogram of Grayscale Image')
plt.show()
```

CODE
Double-click <u>here</u> for the solution.

<!-- The answer is below:

rcParams['figure.figsize'] = 8,4
plt.hist(gray_img.ravel(),256,[0,256])
plt.title('Histogram of Grayscale Image')
plt.show()

--->

[ ]:
```
# 2.2. Using your color image from Exercise 1, display its color histogram

# Write your code below:
```

```
Double-click <font color="red"><b><u>here</b></u></font> for the solution.

<!-- The answer is below:

colors = ('b','g','r')
for i,col in enumerate(colors):
    histr_color = cv2.calcHist([download_image_filename],[i],None,[256],[0,256])
    plt.plot(histr_color,color = col)
    plt.xlim([0,256])
plt.show()

--->
```

CODE

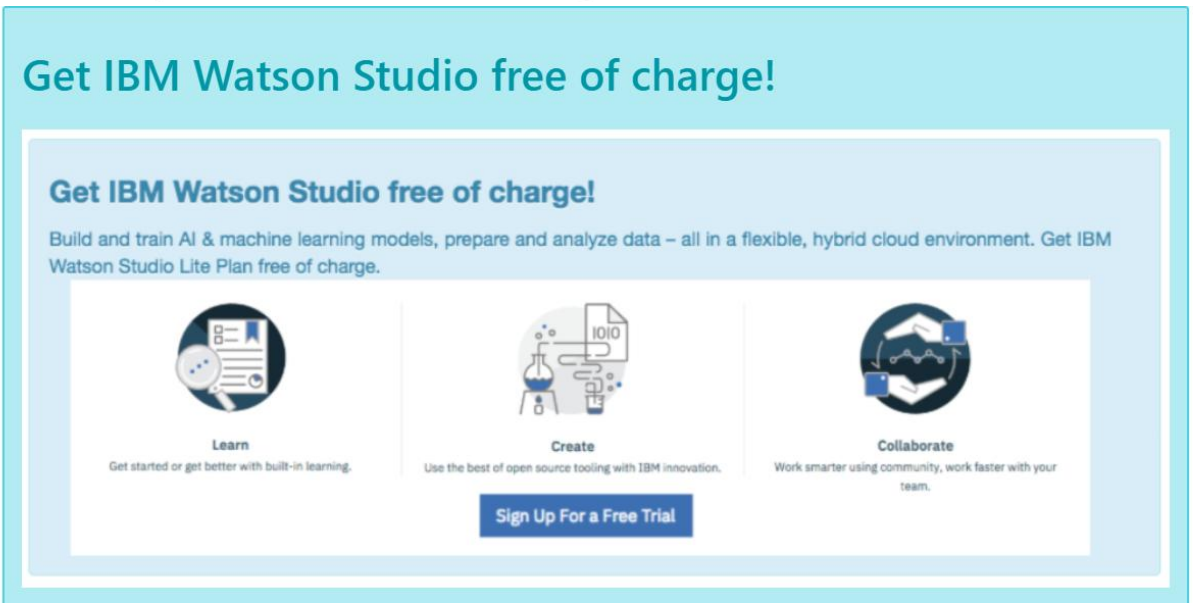
Double-click <u>here</u> for the solution.

<!-- The answer is below:

```
colors = ('b','g','r')
for i,col in enumerate(colors):
    histr_color = cv2.calcHist([download_image_filename],[i],None,[256],[0,256])
    plt.plot(histr_color,color = col)
    plt.xlim([0,256])
plt.show()
```

--->

Thank you for completing this lab!



About the Authors:

This lab was written by Sacchit Chadha and revised by Nayef Abou Tayoun

Sacchit Chadha is a Software Engineer at IBM, and is a rising senior pursuing a Bachelors Degree in Computer Science from the University of Waterloo. His work at IBM is focused on Computer Vision, Cloud Computing and Blockchain.

Nayef Abou Tayoun is a Cognitive Data Scientist at IBM, and is pursuing a Master's Degree in Artificial Intelligence.

Copyright © 2019 IBM Developer Skills Network. This notebook and its source code are released under the terms of the MIT License.