Laboratory 1. Working with images in OpenCV

The **OpenCV** library offers a large set of functions useful in Computer Vision applications, originally written in C++. This laboratory uses an OpenCV binding in **Python** and is based on **PyCharm** integrated development environment. A digital image is the equivalent of a matrix. OpenCV allows pixels or groups of pixels manipulations, displaying and saving images. In order to use the OpenCV functionalities and also, to easily operate with large arrays, the fallowing libraries should be imported at the beginning of the program:

```
import cv2
import numpy as np
```

NumPy is a Python optimized library for large multi-dimensional arrays, allowing the use of high-level functions for those arrays.

1.1 Basic image operations

1.1.1 Reading an image

There are several functions of interest for reading, writing, and displaying images. OpenCV allows reading different types of images (JPG, PNG, etc), grayscale or color images, also images with alpha channel. The function used for reading images is `imread` which has the following syntax:

```
retval  = cv2.imread( filename[, flags] )
```

The function's arguments are:

- `retval` is the image if it is successfully read, otherwise it is None (if the filename is wrong or the file is corrupt).
- `Path of the image file`: this can be an absolute or relative path. This is a mandatory argument.
- `Flags`: needed for reading an image in a particular format (for example, grayscale/color/with alpha channel). This argument is optional, with a default value of `cv2.IMREAD_COLOR` or `1` which loads the image as a color image. The set of flags available are:
  `cv2.IMREAD_GRAYSCALE` or `0`: reads the image in grayscale mode
  `cv2.IMREAD_COLOR` or `1`: read a color image (having 3 matrices for 3 color components). Any transparency of that image will be neglected. It is the default flag.
  `cv2.IMREAD_UNCHANGED` or `-1`: reads the image as such, including the alpha channel.

Example: `test_image = cv2.imread("C:\Student\images\test_image.jpg",0)`

1.1.2 Displaying an image

The format of the image to be displayed should be:

1. for an image in ***float*** data type, then the range of pixels values should be between 0 and 1
2. for an image in ***int*** data type, the range of values should be between 0 and 255.

The 1st option to display an image is Matplotlib's `imshow` function. `matplotlib.pyplot` is a collection of command style functions that make **Matplotlib** work like MATLAB, available by importing the library at the beginning of the program:

```
import matplotlib.pyplot as plt
```

The function syntax is:

```
none  =  plt.imshow( mat )
```

with only 1 parameter `mat` that represents the image to be displayed. When Matplotlib is used in a Terminal or in a script, as the case is for PyCharm, then it is also required to use the function `plt.show()` for the figures to be displayed.

Example:

```
testImage = cv2.imread("…test_image.jpg",0)
plt.imshow(testImage)
plt.show()
```

The 2nd option to display an image is OpenCV's `cv2.imshow()` function, that will be used while running the Python script from command line. This function's syntax is:

```
none    =  cv2.imshow( winname, mat )
```

with the parameters `winname` for the name of the window, and `mat` for the image to be displayed.

Example:

```
testImage = cv2.imread("…test_image.jpg",0)
cv2.imshow("Test image", testImage)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **E1. Task**

In the previous example there are 2 new functions (`cv2.waitKey()` and `cv2.destroyAllWindows()`). Search for their syntax and parameters, then explain their utility! Also analyze the syntax / parameters for the following functions: `cv2.namedWindow()` and `cv2.destroyWindow()`.

1.1.3 Saving an image

In cases where the image output of a program is needed to be saved, we will write that image into a file using the `cv2.imwrite()` function:

```
retval = cv2.imwrite( filename, img [, params] )
```

The function parameters are: `filename` – representing a string with the absolute or relative path where the image should be saved; `img` – the image matrix to be saved; `params` – additional information, as the JPEG compression quality and others.

1.2 Types of images

1.2.1 Binary and Grayscale images

In a binary image, each pixel is represented either black, value equal to 0, either white, with a value of 1 or 255 (depending on the data type). Such binary images can be obtained after segmentations, being used as masks.

Some images consist of only one 2D matrix and the values in that matrix are the intensity values of each pixel. A value of `0` means the pixel is black and as the value increases, the gray shade moves towards white. A value of `255` is a white pixel (true for images of type `uint8`).

1.2.2 Color images

The third dimension from `image.shape` indicates the number of channels in an image. For most images, this number is 3 (namely R,G,B for red, green and blue). In some cases, there may be an additional channel (called *alpha channel*) which contains the transparency information of the pixels. Each channel

itself is an intensity image (or grayscale). The combination of intensity values from the 3 channels gives the color that is displayed on the screen. The typical color space is RGB, but other color spaces may be used (HSV, YUV, Lab, etc).

In OpenCV, the order of the channels R, G and B is reversed: the Blue channel is indexed first, followed by the Green Channel and finally the Red Channel. So OpenCV uses BGR format, while Matplotlib assumes the image to be in RGB format. There are 2 possible fixes:

- convert the image to RGB color-space using `cv2.cvtColor(img, cv2.COLOR_BGR2RGB)` or
- reverse the order of channels as in `plt.imshow(img[:,:,::-1])` (swaps the 1st and 3rd channel).

Each separate channel can be accessed separately using the function `cv2.split()`, while `cv2.merge()` allows merging back the color components into an image.

Example:

```
b,g,r = cv2.split(image)
imgMerged = cv2.merge((b,g,r))
plt.figure()
plt.subplot(141);plt.imshow(b,cmap='gray');plt.title("Blue channel")
plt.subplot(142);plt.imshow(g,cmap='gray');plt.title("Green channel")
plt.subplot(143);plt.imshow(r,cmap='gray');plt.title("Red channel")
plt.subplot(144);plt.imshow(imgMerged[:,:,::-1]);plt.title("Merged
Output")
```

In images with an ***alpha channel***, each pixel has a color value given by the 3 amounts of red, green and blue, and also a *numerical transparency value* (between 0 to 255) that defines what will happen when the pixel is placed over another pixel. The 4th channel is called alpha channel and indicates the transparency. The alpha mask is a very accurate segmentation of the image (foreground / background) and it is useful for creating overlays (as in Augmented Reality type of applications).

- **E2. Application**

Use the alpha-channel from the image *Moustache.png* to put the moustache on top of the nose-lip region on the face in *trump.jpg* ! Try to change the code slightly so the original skin region in the nose-lip area is partially visible through the moustache! Use the transparency characteristic of the alpha channel.

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

faceImage = cv2.imread('trump.jpg')
# Load the moustache image with Alpha channel
moustache = cv2.imread('Moustache.png',-1)
print("image Dimension ={}".format(moustache.shape))
# Separate the Color and alpha channels
moustacheBGR = moustache[:,:,0:3]
moustacheMask1 = moustache[:,:,3]
```

```
# faceImage is a 3-channel color image, we need a 3 channel image for the mask
moustacheMask = cv2.merge((moustacheMask1,moustacheMask1,moustacheMask1))
# Make the values [0,1] since we are using arithmetic operations
moustacheMask = np.uint8(moustacheMask/255)

face = faceImage.copy()
# Get the nose-lip region from the face image
roi = face[65:110,20:100]

# Use the mask to create the masked roi region
maskedRoi = cv2.multiply(roi,(1-  moustacheMask ))
# Use the mask to create the masked moustache region
maskedMoustache = cv2.multiply(moustacheBGR,moustacheMask)
# Combine the moustache in the nose-lip region to get the enhanced image
nose_lip_final = cv2.add(maskedRoi, maskedMoustache)
face[65:110,20:100] = nose_lip_final

plt.figure()
plt.subplot(121); plt.imshow(faceImage[:,:,::-1])
plt.subplot(122); plt.imshow(face[:,:,::-1])
plt.show()
```

1.2.3 Image properties

The datatype of a loaded image can be obtained by '`image.dtype`', and the result can be `uint8`, `float32`, etc.

Image properties include the number of rows, columns, and channels. The shape of an image is accessed by `image.shape`. If an image is grayscale, the tuple returned contains only the number of rows and columns, since there is only one matrix. For a color image, the values returned will indicate the number of rows, number of columns and 3 channels (corresponding to red, green, blue).

Total number of pixels is accessed by `image.size`.

- **E3. Application**

- Read the input color image "*shoes.jpg*" in variable `bgrImg`, convert it into a grayscale image `grayImg` and display it.

- Resize `grayImg` to have half pixels per horizontal and vertical coordinates, then display the resized image. Use the function `cv2.resize`.

- Display the transposed matrix of `grayImg`. Use the function `cv2.transpose`.

- Increase by 50 the intensity in `grayImg`, then decrease the intensity by 50. Display both output images and comment on the results.

- Generate and display the negative of `grayImg`.

- Save one of the previous output images in JPEG format with a quality factor of 80.

```
cv2.imwrite('output.jpg', im, [cv2.IMWRITE_JPEG_QUALITY, 80])
```

1.3 Color spaces

1.3.1 RGB color space

RGB is an additive color space in which Red, Green and Blue light waves are added in various proportions to produce different colors. It is the most commonly used color space in image processing and computer vision.

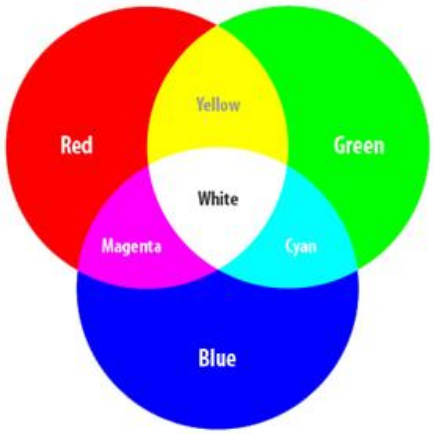

Figure 1. Primary colors in RGB color space and resulting color combinations

As already mentioned, in OpenCV the image is loaded into BGR format by default, so it is stored in reverse order. For images with 8-bit per channel, the intensity values for a given channel can range from 0 to 255. Brighter pixels signify higher intensity values in a particular channel and vice-versa. In RGB color space, all 3 channels contain information about the color as well as brightness. It is better for some applications if we can separate the color component, also known as *Chrominance* , from the lightness or brightness component also known as *Luminance*. This separation is present in the following color spaces.

1.3.2 HSV color space

The 3 components in HSV color space are:

- **Hue** - indicates the color or tint of the pixel, ranging from 0 to 180 in OpenCV. It is represented as an angle where a hue of 0 is red, green is 120 degrees (60 in OpenCV), and blue is at 240 degrees(120 in OpenCV). Because hue is an angle, the red color is contained for both H = 0 and H = 360 (or 180 in OpenCV's representation).
- **Saturation** - indicates the purity or richness of the color. Different shades of a color correspond to different saturation levels. Saturation of 0 corresponds to white color which indicates that the color shade is at the lowest or the color is simply absent.
- **Value** - indicates the amount of brightness or luminance of the pixel. When the value is 0, the image is black and when it is close to 255, the image is white.

HSV is more intuitive than RGB color space because it separates the color and brig–htness into different axes. This makes it easier to describe any color directly.

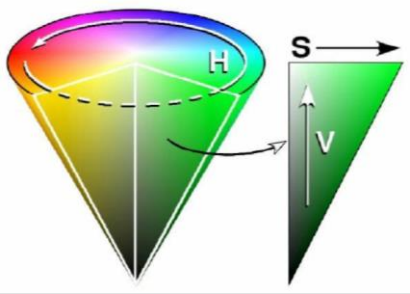

Figure 2. Illustration of Hue, Saturation and Value components in HSV color model

In order to convert an image from BGR to HSV format, we will use OpenCV's `cv2.cvtColor` function:

```
dst = cv2.cvtColor( src, code[, dst[, dstCn]] )
```

The parameters of this function are:

`src` - input image: 8-bit unsigned, 16-bit unsigned (CV_16UC...), or single-precision floating-point.

`dst` - output image of the same size and depth as `src`.

`code` - color space conversion code (`cv2.COLOR_BGR2HSV`, `cv2.COLOR_HSV2RGB`, `v2.COLOR_BGR2YCrCb`, `etc.`).

`dstCn` - number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from `src` and `code`.

1.3.3 YCbCr color space

The YCbCr color space is obtained from the RGB color model. YCbCr is mainly used in digital television and image and video compression (JPEG, H.264/AVC, H.265/HEVC). Its main components are:

Y – luminance channel derived from the RGB values, leads to an imitation of the grayscale image

Cb = B - Y - indicates the amount of Blue in the image

Cr = R – Y - indicates the amount of Red in the image

Converting an image from BGR to YCbCr color model can be performed using the same `cv2.cvtColor` function, with the parameter `code` set to `cv2.COLOR_BGR2YCrCb`.

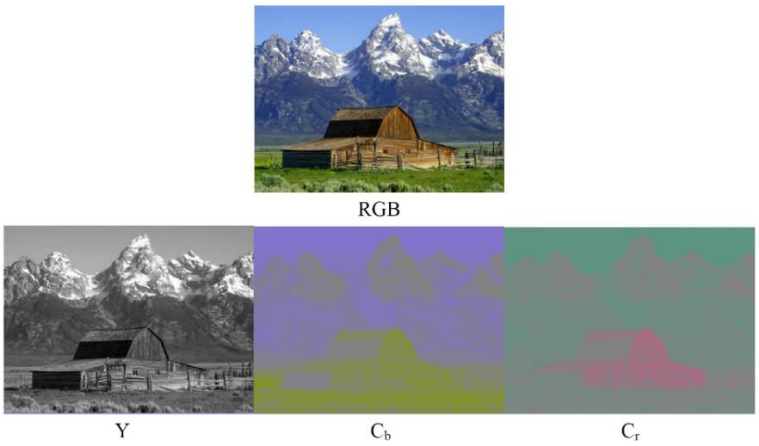

Figure 3. Illustration of the 3 channel components in the YCbCr color space

1.4 Image histogram

A histogram is a graphical representation of the distribution of data. An image histogram gives a graphical representation of the tonal distribution in a digital image. The x-axis indicates the range of intensity values the variable can take, which lie between 0 and 255. This range can be divided into a series of intervals called *bins*. The y-axis shows the count of how many values fall within that interval or bin, so the actual number of times a particular intensity value occurs in the image.

The function `plt.hist()` available in the `matplotlib` library is designed for drawing the histogram of an image. It can be used with the following syntax:

```
hist, bins, patches = plt.hist( x, bins=None, range=None, density=None, weights=None, cumulative=False, bottom=None, histtype='bar', align='mid', orientation='vertical', rwidth=None, log=False, color=None, label=None, stacked=False, normed=None )
```

Several common parameters are already assigned, while the main input parameters are: `x` - source image as an array, `bins` - number of bins, `color` - the color for plotting the histogram. The main output parameters are: `hist` - histogram array, `bins` - edges of bins.

Observation: the input to the histogram function is an array, not a complete image matrix. Hence, it is necessary to flatten the matrix into an array before passing it to the function.

- **E4. Application**

Imagine building a visual search engine for shoes in an online store where people can search for shoes by color. Every item in that store will be tagged with the dominant color(s). In this application, you will identify the dominant color using the hue component in a photo and the image histogram.

- import libraries `cv2`, `matplotlib`, and `numpy`

- read the color image "*shoes.jpg*" in `bgrImage` and display it

Figure 4. Test image "shoes.jpg"

```
import matplotlib
matplotlib.rcParams['figure.figsize'] = (10.0, 10.0)
matplotlib.rcParams['image.interpolation']='bilinear'



bgrImage = cv2.imread("shoes.jpg")
plt.figure(figsize=[20,10])
plt.imshow(bgrImage [...,::-1])
plt.axis('off')
```

- convert the image to HSV color space using the `cvtColor` function and split it into H, S and V channels

```
hsvImage = cv2.cvtColor(bgrImage,cv2.COLOR_BGR2HSV)
H, S, V = cv2.split(hsvImage)
print(H.shape)
```

- remove all background pixels with white / light gray values from the Hue array. This can be easily done by not considering all pixels below a certain saturation level. Use the `numpy` array method `flatten` to transform the foreground significant pixels in the Hue array into a vector

```
H_array = H[S > 10].flatten()
print(H_array.shape)
```

- display the histogram of the Hue vector and identify the dominant color in the original image. Justify your choice!

```
plt.figure()
plt.subplot(121);plt.imshow(bgrImage [...,::-1]);
plt.title("Image"); plt.axis('off')
plt.subplot(122);plt.hist(H_array, bins=180, color='r');plt.title("Histogram")
```

1.4.1 Histogram equalization

Histogram equalization is a non-linear contrast enhancement technique. For an image with a tonal distribution of intensity concentrated into a narrow interval of gray shades (a dark image for example), by redistributing the histogram bins over the entire range of gray values (from black to white), the histogram will be equalized and the image contrast improved. The function `equalizeHist()` performs histogram equalization on a grayscale image. The function syntax is:

```
dst = cv2.equalizeHist( src[, dst] )
```

with the parameters `src` - source 8-bit single channel image, and `dst` - destination image of the same size and type as `src`.

- **E5. Application**

Apply histogram equalization for the grayscale image "rose.jpg" and compare the images and their histograms before and after the grayscale transformation. The original image is quite dark, so the result should be an image with improved contrast (with gray shades covering the entire range from 0-black to 255-white).

- import libraries `cv2`, `matplotlib`, `matplotlib.pyplot`, and `numpy`

- read the image in grayscale format

```
im = cv2.imread("rose.jpg", cv2.IMREAD_GRAYSCALE)
```

- apply histogram equalization and save the output image as `imOut`

```
imOut = cv2.equalizeHist(im)
```

- display the original image and the image with enhanced contrast, in the same figure with `subplot`, as shown in Figure 5. Then, display their histograms in the same figure, using subplot. Use the `numpy` method `ravel()` that reshapes the original image matrix into a flatten one-dimensional array.

Figure 5. Original 'rose.jpg' (left) and image after histogram equalization (right)

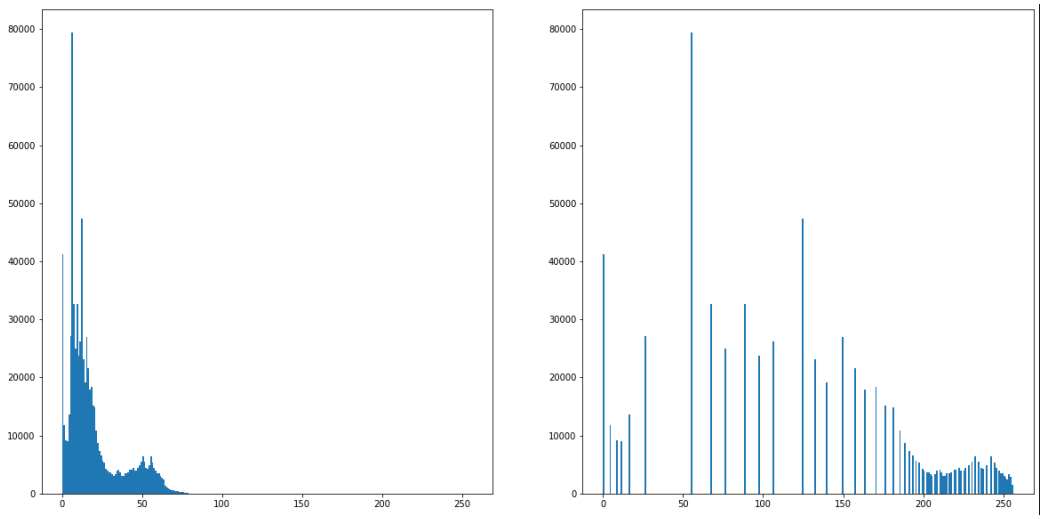

Figure 6. Original dark image histogram (left) and equalized histogram (right)

```
plt.figure()
ax = plt.subplot(1,2,1); plt.imshow(im, vmin=0, vmax=255);
ax.set_title("Original Image");ax.axis('off')
ax = plt.subplot(1,2,2); plt.imshow(imOut, vmin=0, vmax=255);
ax.set_title("Histogram Equalized")
ax.axis('off')

plt.figure()
plt.subplot(1,2,1); plt.hist(im.ravel(),256,[0,256]);
plt.subplot(1,2,2); plt.hist(imOut.ravel(),256,[0,256]);
plt.show()
```

- **E6. Application**

Apply histogram equalization for a color image! Do not simply perform histogram equalization for each channel separately. When each color channel is non-linearly transformed independently, the results are completely new and unrelated colors. A better solution is to transform the images to a space like HSV color space, where colors/hue/tint is separated from the intensity, and then perform histogram equalization.

Perform the following steps:

- read the image "*flowers.jpg*"
- tranform the image to HSV color space
- perform histogram equalization only on the V channel
- transform the image back to RGB color space
- convert the equalized HSV image back to BGR format
- display both images, original and output, in the same figure, using `subplot`
- display both histograms, original V channel and equalized V channel, in the same figure, using `subplot`

1.4.2 Contrast Limited Adaptive Histogram Equalization

Histogram equalization uses all the pixels of the image to improve contrast. In many cases the results are good, but sometimes it is necessary to enhance the contrast locally! In such situations, applying histogram equalization to the entire image will make it appear more natural and less dramatic. Contrast Limited Adaptive Histogram Equalization (CLAHE) has the advantage of improving the local contrast, by allowing the user to specify the area considered "local".

In this adaptive histogram equalization, the image is divided into small blocks called "`tiles`" (`tileSize` is `8x8` by default in OpenCV). Then each of these blocks are histogram equalized as usual. So in a small area, the histogram would confine to a small region (unless there is noise). If noise is there, it will be amplified. CLAHE is a variant of adaptive histogram equalization in which the contrast amplification is limited, so as to reduce this problem of noise amplification. In simple words, CLAHE does histogram equalization in small patches or in small tiles with high accuracy and contrast limiting. After equalization, to remove artifacts in tile borders, bilinear interpolation is applied.

- **E7. Application**

Apply the adaptive histogram equalization algorithm called CLAHE to the original image "*night_sky.jpg*" and compare the result with usual histogram equalization. Perform the following steps:

- read the color image "*night_sky.jpg*"

- convert it to HSV color space and save it in `hsvImg`. Make a copy of `hsvImg` and denote it `hsvImgCopy`, using the method `copy()` provided by `NumPy` arrays.

- perform histogram equalization only on the V channel in `hsvImg`, convert back to BGR format and store the image with normal equalized histogram in `normalEqImg`

- create a CLAHE object using

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
```

- use the `apply()` method of the `clahe` object to perform adaptive histogram equalization for the V channel of the `hsvImgCopy` (`hsvImgCopy[:,:,0]` represents the Hue channel, `hsvImgCopy[:,:,1]` is the Saturation channel)

```
imhsvCLAHE[:,:,2] = clahe.apply(hsvImgCopy[:,:,2])
```

- convert also `hsvImgCopy` to BGR format

- display in the same figure (using `subplot`) the original image, the normal equalized image, and the image adaptively equalized with CLAHE. The result should resemble Figure 7, where the final image at the right is clearly more faithful to the original intent of the photographer.

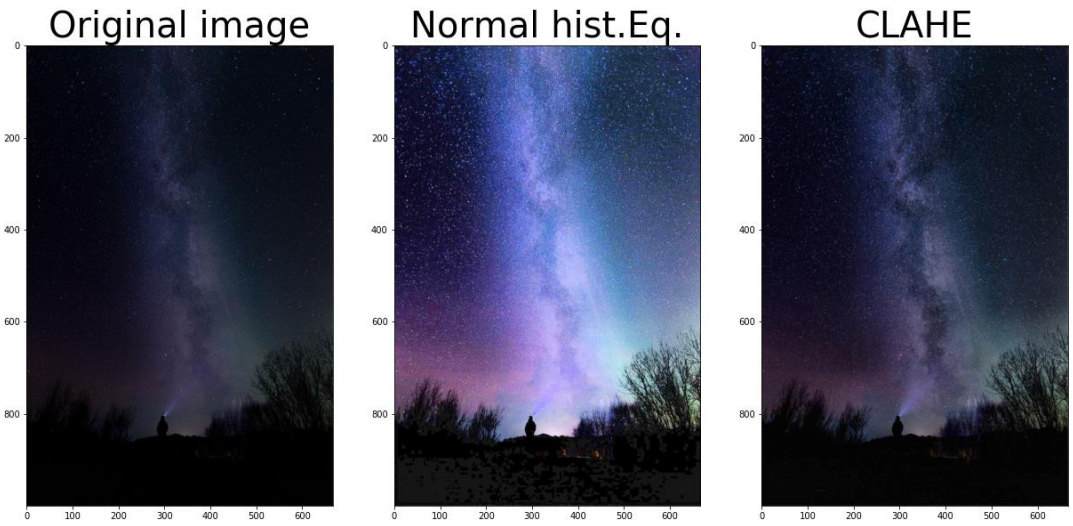

Figure 7. Adaptive histogram equalization compared to usual histogram equalization

- **E8. Task**

Knowing the meaning of the histogram, search for the definitions of Probability Mass Function (PMF) and Cumulative Distribution Function (CDF). Write their definitions in your report.