IMPRO Lab 1

Jayotsana Sharma

March 6, 2025

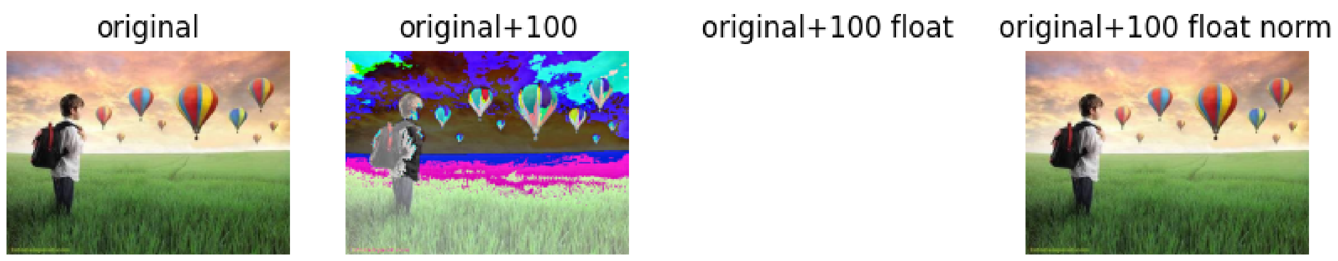
1. Working with Digital Images

- While working on the first section, we first uploaded an image and then tried different operations on that to see how these affect the image. Original stores the real array or pixels of the image.
- In the **first** condition, adding a number (in this case, 100) to the image increases all pixel values by that amount. However, when I checked the maximum and minimum pixel values, they remained the same as in the original image. This results in a change to all pixel values, potentially causing an overflow of intensity values.
- In the second condition, we convert the intensities to floating-point values. When checking the maximum and minimum values in this case, I observed different values that exceed the standard intensity range of [0, 255], leading to an overflow of intensities.
- To resolve the above situation, we need to normalize the image.

code

```
# Pipeline 1
wrongim = original + 100

# Pipeline 2
im = original
# floatim = im + 100
floatim=im.astype(float)+100

# Pipeline 3
floatimnorm = (floatim - floatim.min())/(floatim.max()-floatim.min())
```

- **Results :**



2. Color Images

(a) To convert a colored image to grayscale, we can use a specific function that adjusts the weights of the three color channels (red, green, and blue).
By assigning generalized weights to each channel, or by calculating the average weights from the combination of the three channels and then assigning the average to each channel, we can produce a grayscale image.
However, the average conversion tends to provide better contrast compared to the standard method.

code

```
def color_to_rgb(im, weights=[0.3086,0.6094,0.0820]):
    red_im=im[:,:,0]
    green_im=im[:,:,1]
    blue_im=im[:,:,2]

    L_I=weights[0]*red_im+weights[1]*green_im+weights[2]*blue_im
    return L_I

im_gray = color_to_rgb(original)
im_gray_avg = color_to_rgb(original, weights=[0.33,0.33,0.33])
```

- **Results :**

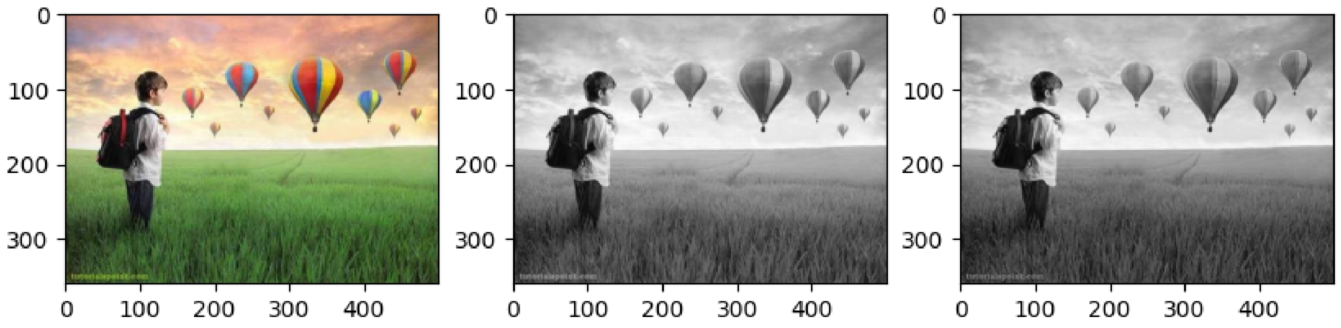


Figure 1: Color(1) to Gray Scale(2) and Average Gray Scale(3) Conversion

(b) To create a color blindness effect, we combine the intensities of the red and green color channels. This is done by calculating the average of the red and green channels and then assigning this average value to both channels. The blue channel, however, will remain unchanged from the original image.

code

```
def color_to_colorblind(im):
    im_norm=im.astype(float)/255
    new_img=np.zeros_like(im_norm)
    red_im=im_norm[:,:,0]
    green_im=im_norm[:,:,1]
    blue_im=im_norm[:,:,2]
    mean_rg=(red_im+green_im)/2
```

```
        new_img[:,:,0]=mean_rg
        new_img[:,:,1]=mean_rg
        new_img[:,:,2]=blue_im
        #new_im=np.array([red_imnorm,green_imnorm,blue_im]).reshape(im.shape)
        return new_img

    img=color_to_colorblind(original)
```

- **Results :**

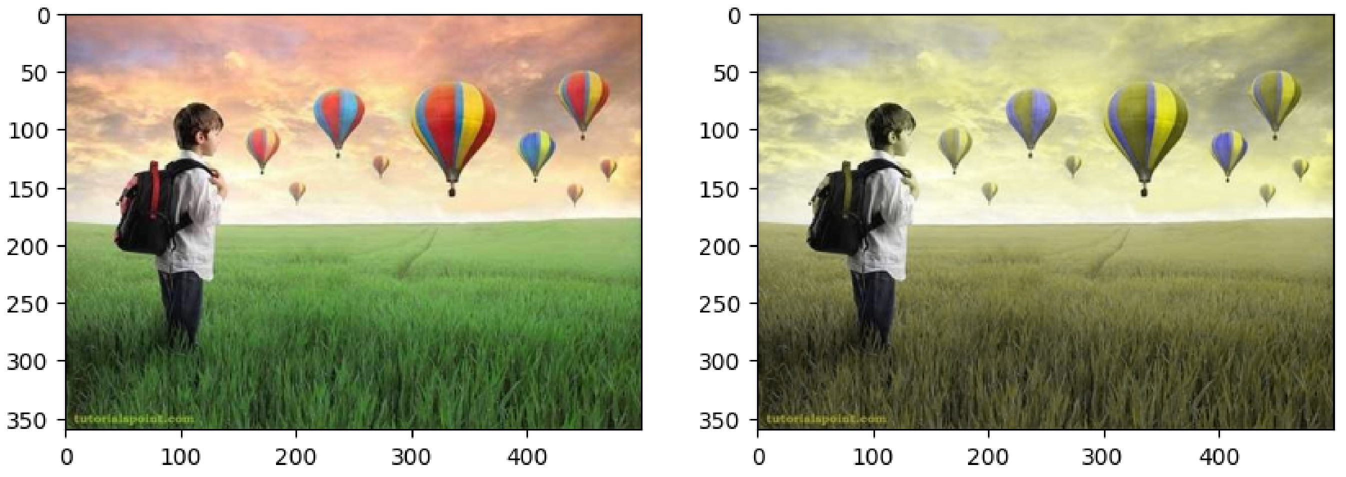


Figure 2: Original v/s Colorblind Picture

3. Global Measures

(a) Function to compute and visualize the histogram :

code

```
    def image_histogram(input_image, plot_me=True):
        hist, bin_limits = np.histogram(input_image, bins=256)
        bin_centers = 0.5*(bin_limits[:-1] + bin_limits[1:])
        return bin_centers,hist
```

(b) Function to store all the properties of an image as a dictionary :

code

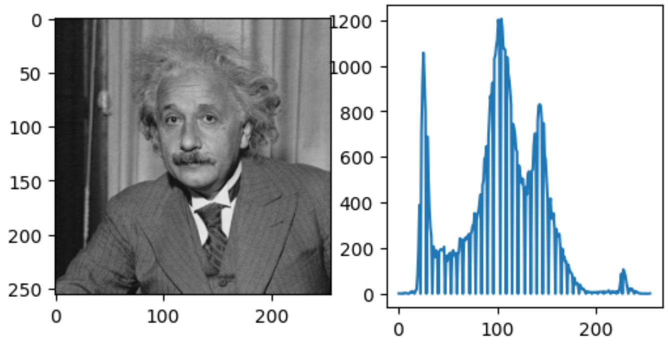
```
    def global_measures(input_image, display=True):
        properties = {}
        properties['shape'] =input_image.shape
        properties['min'] = input_image.min()
        properties['max'] = input_image.max()
        properties['luminance']=input_image.mean()
        properties['contrast']=input_image.std()
        properties['type'] = input_image.dtype
        properties['snr'] = 20*(np.log10(input_image.mean()/input_image.std()))
        properties['histo'] = image_histogram(input_image)

        if display:
            for name,val in properties.items():
                if name != 'histo':
                    print(name,'-->',val)
        return properties
```

(c) While calling the functions of global measures and histogram we can observe the properties of the input image and the intensity distribution pattern in the image(as in figure(a)). For example:

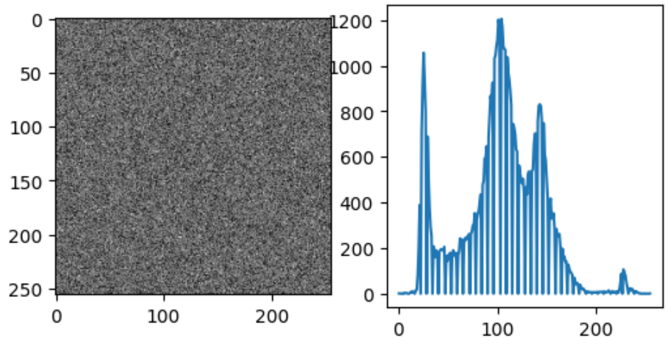
code

```
    im = io.imread(os.path.join(IMDIR,'einstein.png'),as_gray=True)
    im_ref = 255*im/im.max()
    properties = global_measures(im_ref,display=True)
    bins_ref,histo_ref = image_histogram(im_ref)
```

- **Results :**
- **shape** : (256, 256), **min** : 0.0, **max** : 255.0, **luminance** :102.8033447265625, **contrast** : 41.65809713229708, **type** : float64, **snr** : 7.84615632954702



(a) Input Image and Intensity Distribution



(b) Shuffled Image and Intensity Distribution

(d) By applying random shuffling to the input image and then computing the global measures and histogram (as shown in figure (b)), we can observe that there is no change in the global measures compared to before. Additionally, the intensity distribution in the histogram remains the same as it was previously.
Even though the image itself does not look the same due to the random arrangement of all the pixels, the overall statistical properties are unchanged.

code

```
    def shuffle_image(im_gray):
        im_shuffle = im_gray.ravel().copy()
        np.random.shuffle(im_shuffle)
        im_shuffle=np.reshape(im_shuffle, im_gray.shape)
        return im_shuffle

        im_shuffled = shuffle_image(im_ref)
```

```
new_measures=global_measures(im_shuffled)
bins_shuf, histo_shuf = image_histogram(im_shuffled)
```

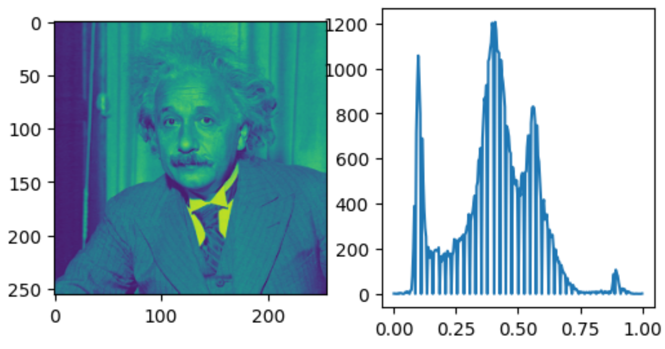
4. Intensity transformations, transfer functions

(a) Function to invert images :

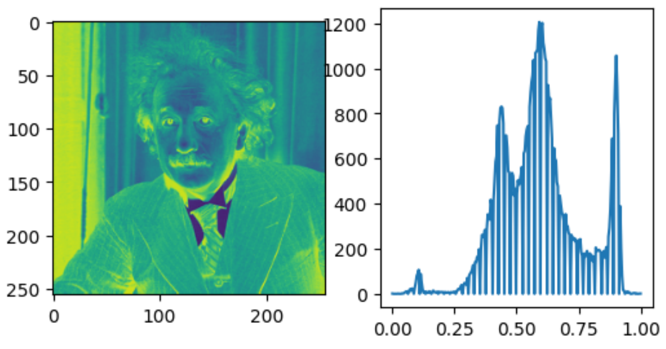
code

```
def invert (input_im):
    im_ref = input_im/input_im.max()
    im_inv=sk.util.invert(im_ref, signed_float=False)
    return im_inv
```

- Effect of Inversion on Images :
  In this comparison of the input image and its inverted version, we can observe that the intensities of the pixels have been reversed. Brighter pixels in the original image are transformed into darker ones, while darker pixels become brighter.
  The histograms also illustrate this inversion of intensities; in the original histogram, there is a higher frequency of darker pixels, whereas in the inverted histogram, the frequency of brighter pixels is significantly increased.



(a) Original Image



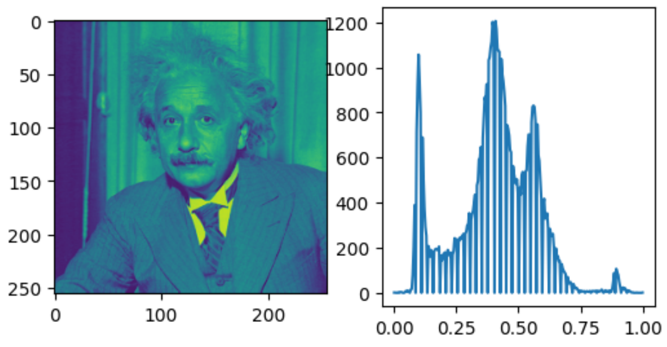
(b) Inverted Image

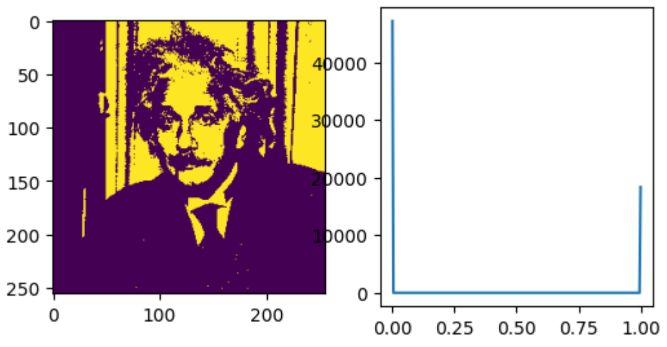
(b) Function to apply thresholding on images :
Thresholding is a process used to segment images by converting them into binary images. It involves selecting a threshold intensity, which is applied to each pixel in the image. If a pixel's intensity is greater than the threshold, it is set to one; otherwise, it is set to zero.

code

```
def threshold(input_im, th=0.5):
    im_ref = input_im/input_im.max()
    row, col = im_ref.shape
    for i in range(row):
        for j in range(col):
            if im_ref[i][j]> th:
                im_ref[i][j]=1
             else:
                im_ref[i][j]=0
    return im_ref
```

- Effect of Thresholding on Images :
  As a result, a full-color image is transformed into a two-color picture that only displays the darkest and brightest colors. In the histogram, this transition is evident as the intensity distribution changes, presenting only two intensity values: zero and one.


(a) Original Image

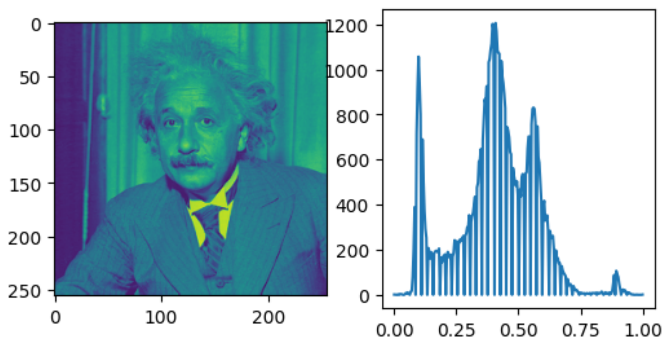

(b) Thresholded Image

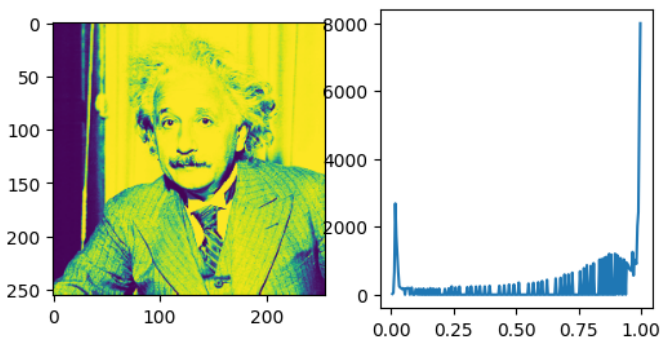
(c) Function to apply contrast stretching on images :
Contrast stretching improve the contrast in an image by 'stretching' the range of intensity values to fill the entire dynamic range.

code

```
def contrast_stretch(img,t=0.3,w=0.05):
    img = img/img.max()
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            a= np.exp(-(img[i][j]-t)/w)
            s=1/(1+a)
            img[i][j]=s

    return img
```

- Effect of contrast stretching on Image :
  By applying contrast stretching to an image, we can effectively reveal the variations in intensities, allowing for a more even distribution of these intensities.
  This is significantly influenced by the parameters t and w, ultimately enhancing the overall visual quality and detail of the image.


(a) Original Image


(b) Contrast Stretched Image

(d) **Plotting of Transfer Functions:**
These three are the plots of transfer functions between the transformed image and the original image.

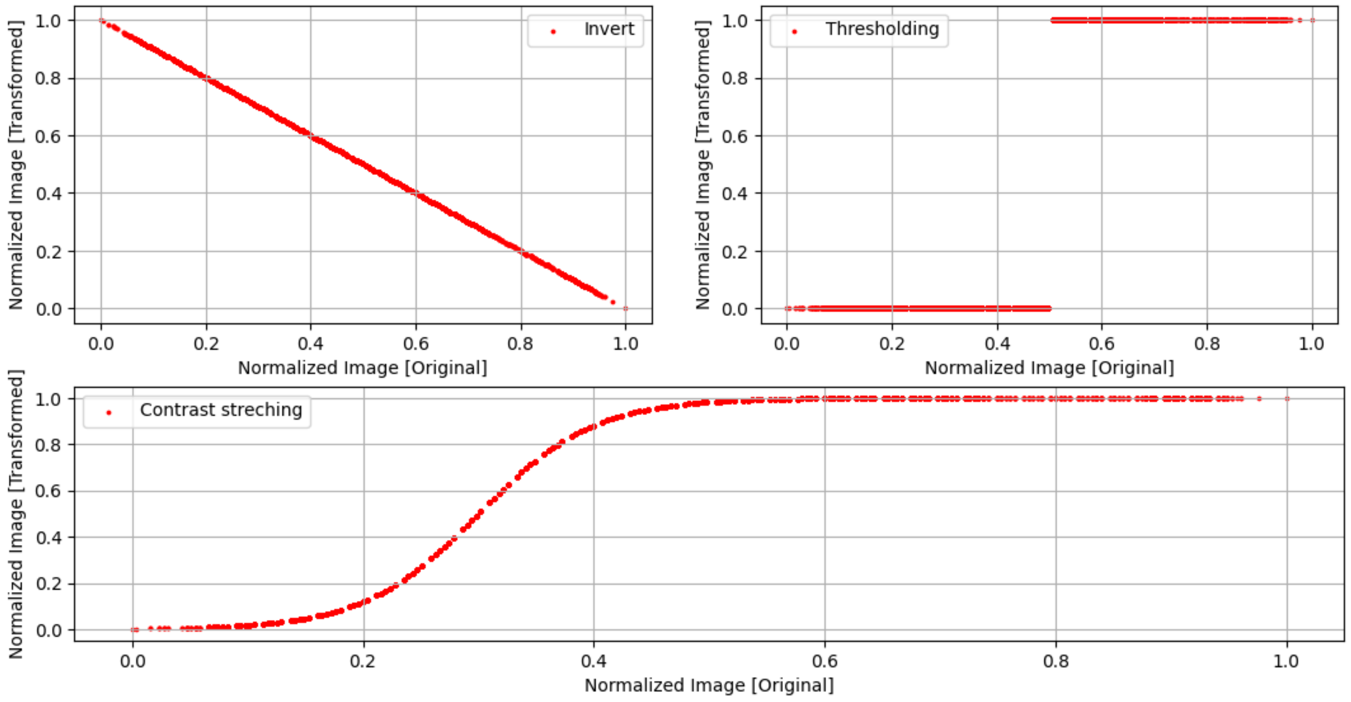


Figure 7: Intensity Transformation Plots