04-ImageEnhancementSpatialDomain-session7

May 30, 2020

Table of Contents

[1]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
```

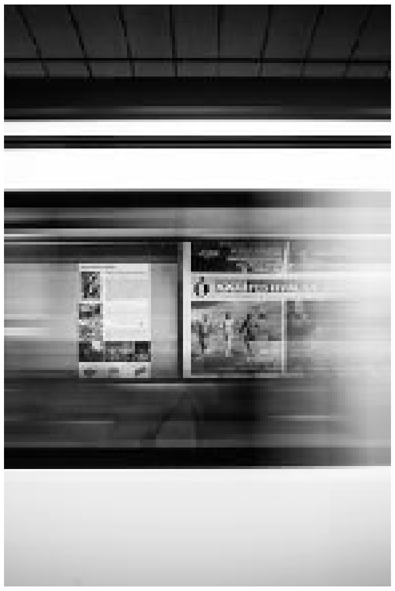
[2]:
```
# this block may raise an error in WINDOWS
!wget https://picsum.photos/200/300 -O sample.jpg
```

```
--2020-05-29 23:17:51--  https://picsum.photos/200/300
Resolving picsum.photos (picsum.photos)... 104.26.5.30, 104.26.4.30,
2606:4700:20::681a:41e, ...
Connecting to picsum.photos (picsum.photos)|104.26.5.30|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://i.picsum.photos/id/345/200/300.jpg [following]
--2020-05-29 23:17:59--  https://i.picsum.photos/id/345/200/300.jpg
Resolving i.picsum.photos (i.picsum.photos)... 104.26.5.30, 104.26.4.30,
2606:4700:20::681a:51e, ...
Connecting to i.picsum.photos (i.picsum.photos)|104.26.5.30|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8474 (8.3K) [image/jpeg]
Saving to: âĂŸsample.jpgâĂŹ

sample.jpg          100%[===================>]   8.28K  --.-KB/s    in 0.02s

2020-05-29 23:18:07 (395 KB/s) - âĂŸsample.jpgâĂŹ saved [8474/8474]
```

[38]:
```python
plt.imshow(cv2.imread("sample.jpg",0),cmap='gray')
plt.axis(False)
plt.show()
```



1 Image Negatives

$$s = L - 1 - r$$

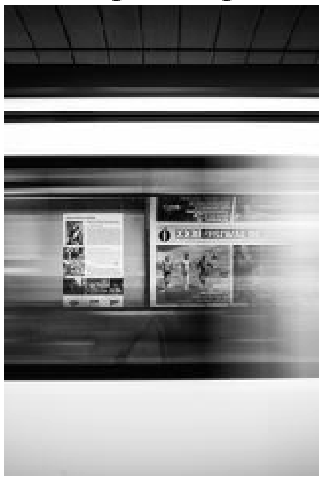
[40]:
```python
def Negative(img,L=None):
    if L==None:
        L=np.max(img)
    return L-1-img

testCases=['sample.jpg','neg0.png','neg1.png']
```

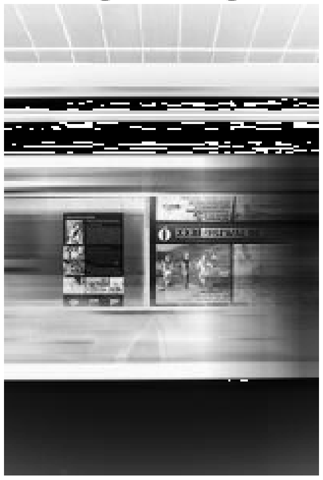
```python
plt.figure(figsize=(10,len(testCases)*5))

for line,pic in enumerate(testCases):
    img=cv2.imread(pic,0)
    plt.subplot(len(testCases),2,2*line+1)
    plt.imshow(img,cmap='gray')
    plt.axis(False)
    plt.title("Original Image")
    plt.subplot(len(testCases),2,2*line+2)
    plt.imshow(Negative(img),cmap='gray')
    plt.axis(False)
    plt.title("negative Image")
plt.show()
```

Original Image

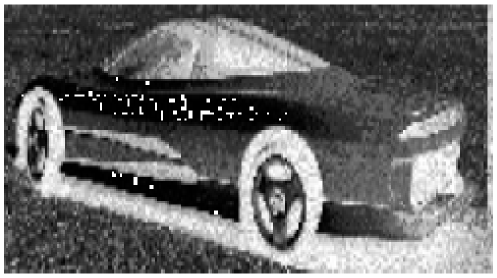


negative Image



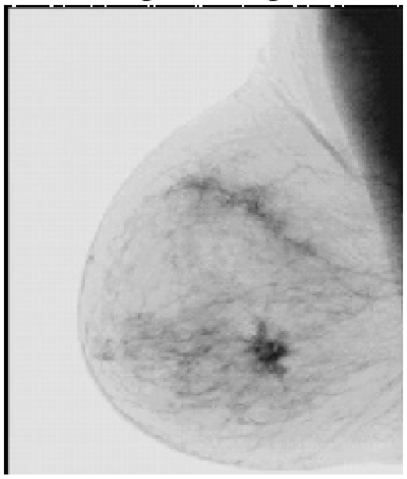
Original Image

negative Image



Original Image

negative Image



2 Log Transformations

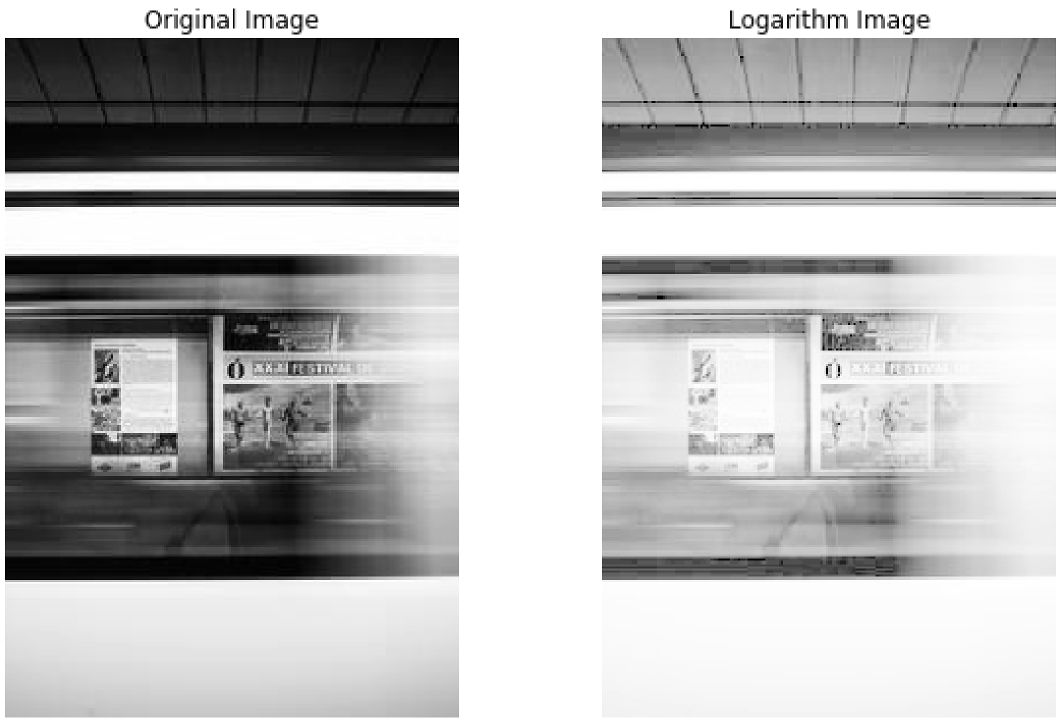
$$s = c * log(1 + r)$$

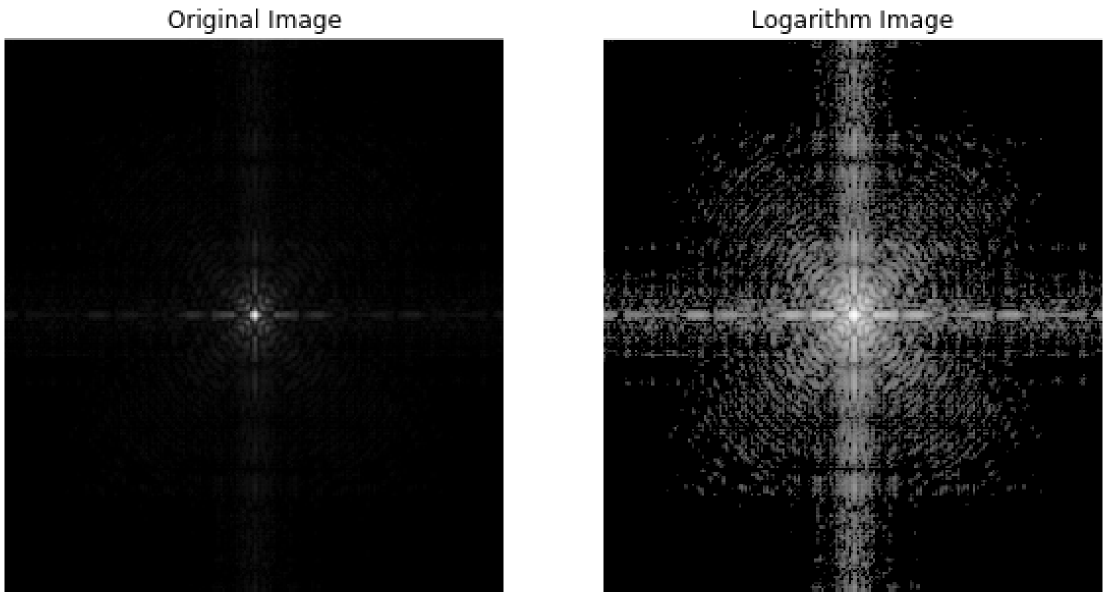
> c is a constant and r>=0

[44]:
```
def LogTransformations(img,c=1):
    img[img==0]=1
    img=img.astype(np.int64)
    return c*np.log10(img)

testCases=['sample.jpg','log0.png',]
plt.figure(figsize=(10,len(testCases)*7))

for line,pic in enumerate(testCases):
    img=cv2.imread(pic,0)
    plt.subplot(len(testCases),2,2*line+1)
    plt.imshow(img,cmap='gray')
    plt.axis(False)
    plt.title("Original Image")
    plt.subplot(len(testCases),2,2*line+2)
    plt.imshow(LogTransformations(img),cmap='gray')
    plt.axis(False)
    plt.title("Logarithm Image")
#     plt.imshow(img,cmap='gray')
plt.show()
```


Original Image
Logarithm Image


Original Image
Logarithm Image

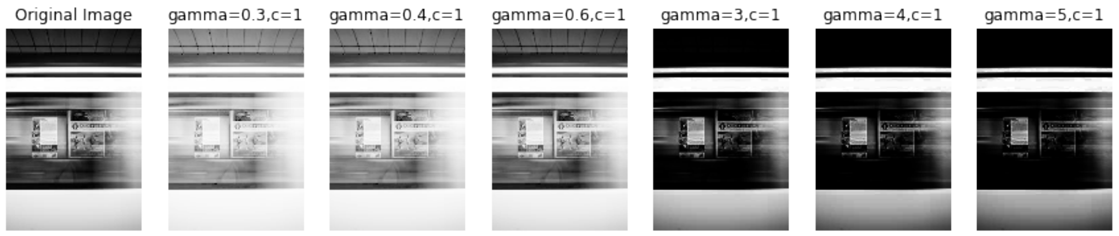
3 Power-law transformation

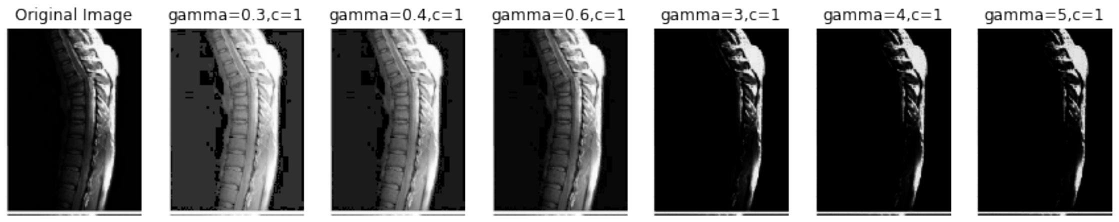
$$s = T(r) = c * r^{\gamma}$$

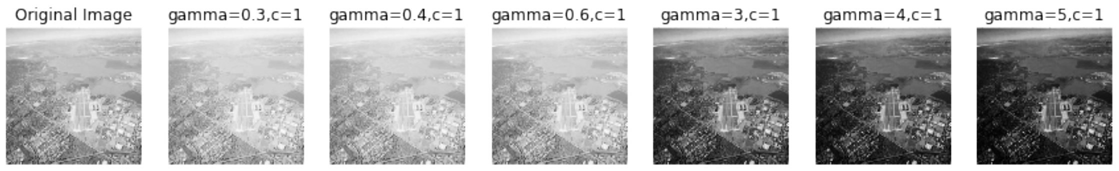
[51]:
```
def PowerLawTransformation(img,gamma,c=1):
    img=c*255*(img.astype(np.int64)/255)**gamma
    img[img>255]=255
    return img.astype(np.int64)

testCases=['sample.jpg','powerLaw0.png','powerLaw1.png']
plt.figure(figsize=(15,len(testCases)*4))

for line,pic in enumerate(testCases):
    img=cv2.imread(pic,0)
    plt.subplot(len(testCases),7,7*line+1)
    plt.imshow(img,cmap='gray')
    plt.axis(False)
    plt.title("Original Image")
    for i,gamma in enumerate([0.3,0.4,0.6,3,4,5]):
        plt.subplot(len(testCases),7,7*line+2+i)
        plt.imshow(PowerLawTransformation(img,gamma),cmap='gray')
        plt.axis(False)
        plt.title(f"gamma={gamma},c=1")
plt.show()
```





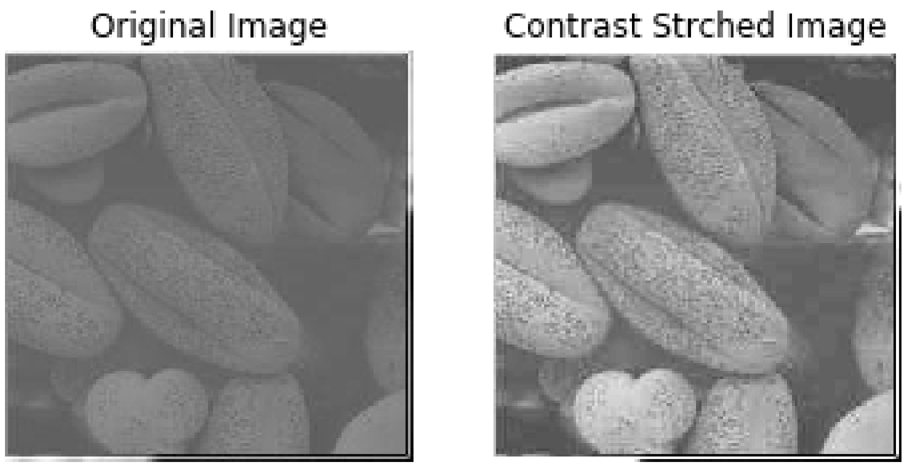


4 contrast stretching

$$s = \begin{cases} \frac{s_1}{r_1} * pix & 0 \leqslant pix \leqslant r_1 \\ \frac{s_2 - s_1}{r_2 - r_1} * (pix - r_1) + s_1 & r_1 < pix \leqslant r_2 \\ \frac{255 - s_2}{255 - r_2} * (pix - r_2) + s_2 & r_2 < pix \end{cases}$$

[52]:
```
def pixelVal_contrastStreching(pix, r1, s1, r2, s2):
    if 0<=pix<=r1:   return (s1/r1)*pix
    elif r1<pix<=r2: return (s2-s1)/(r2-r1)*(pix-r1)+s1
    else:            return (255-s2)/(255-r2)*(pix-r2)+s2
def contrastStreching(img):
    return np.vectorize(pixelVal_contrastStreching)(img, 80, 55, 150, 255)

img=cv2.imread("Low-contrast-image.png",0).astype(np.int64)
plt.subplot(1,2,1)
plt.imshow(img,cmap='gray')
plt.axis(False)
plt.title("Original Image")
plt.subplot(1,2,2)
plt.imshow(contrastStreching(img),cmap='gray')
plt.axis(False)
plt.title("Contrast Strched Image")
plt.show()
```



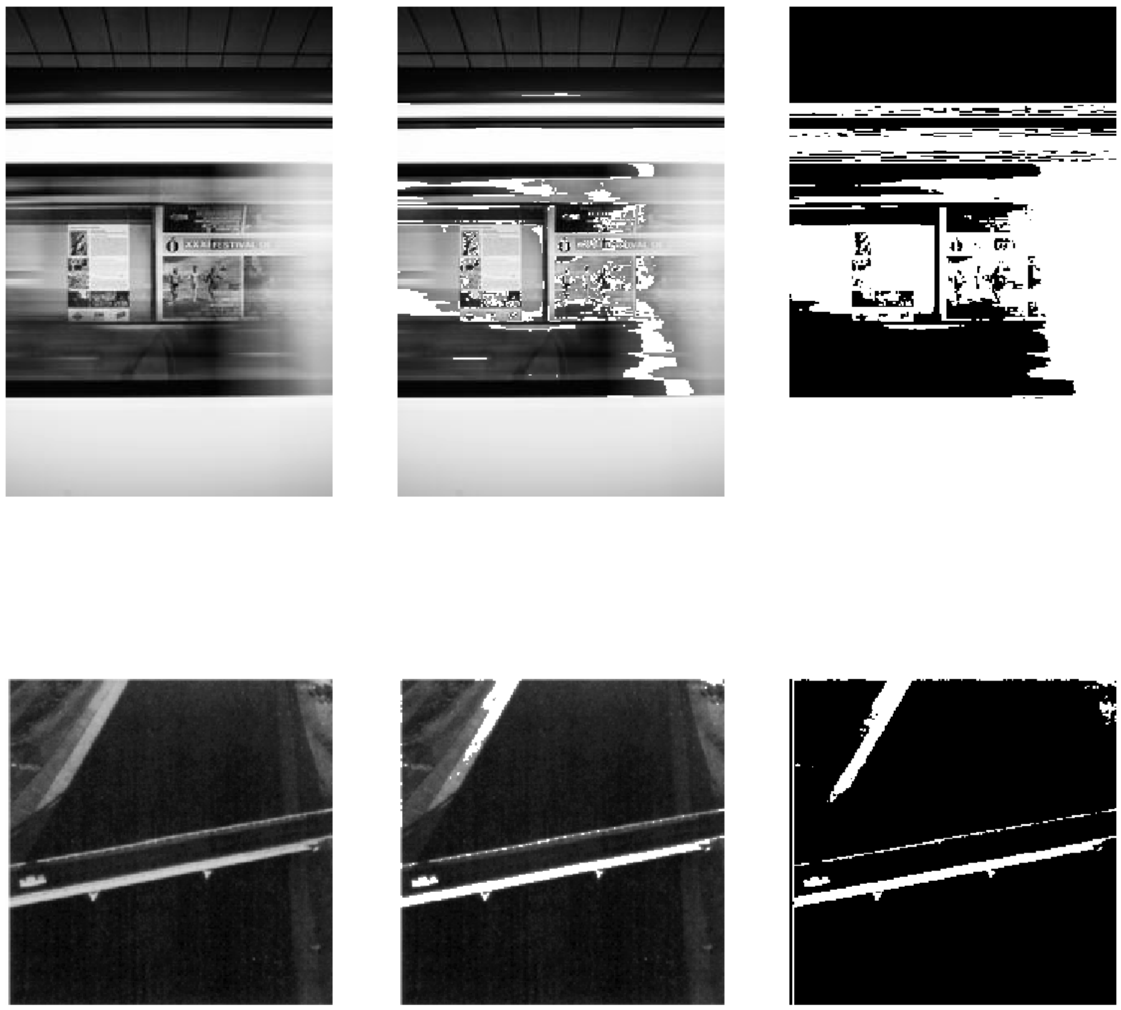
5 Gray-level slicing

Objective: Highlighting a specific range of gray levels in an image

[56]:
```
def GrayLevelSlicing(img,A,B,raisedValue,loweredValue=None):
    assert A<B
    renderdImg=img.copy()
    renderdImg[np.logical_and(img<=B,img>=A)]=raisedValue
    if loweredValue!=None:
        renderdImg[np.logical_or(img>=B,img<=A)]=loweredValue
    return renderdImg

plt.figure(figsize=(20,20))
img=cv2.imread("sample.jpg",0)
plt.subplot(2,3,1)
plt.imshow(img,cmap='gray')
plt.axis(False)
plt.subplot(2,3,2)
plt.imshow(GrayLevelSlicing(img,80,120,255),cmap='gray')
plt.axis(False)
plt.subplot(2,3,3)
plt.imshow(GrayLevelSlicing(img,100,255,255,100),cmap='gray')
plt.axis(False)
img=cv2.imread("GrayLevelSlicing.png",0)
```

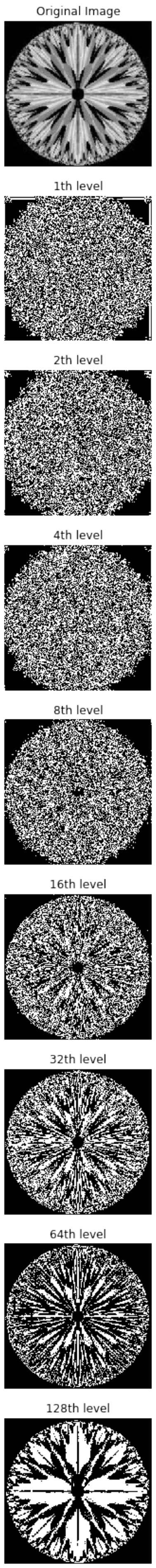
```
plt.subplot(2,3,4)
plt.imshow(img,cmap='gray')
plt.axis(False)
plt.subplot(2,3,5)
plt.imshow(GrayLevelSlicing(img,150,255,255),cmap='gray')
plt.axis(False)
plt.subplot(2,3,6)
plt.
 ↪imshow(GrayLevelSlicing(img,A=120,B=220,raisedValue=255,loweredValue=80),cmap='gray')
plt.axis(False)
plt.show()
```



6 Bit-plane slicing

[58]:
```
def BitPlaneSlicing(img):
     while np.any(img):
        yield img%2
        img//=2

img=cv2.imread("BitPlaneSlicing.png",0)
plt.figure(figsize=(3,30))
plt.subplot(9,1,1)
plt.imshow(img,cmap='gray')
plt.axis(False)
plt.title("Original Image")
for i,slicedImg in enumerate(BitPlaneSlicing(img)):
    plt.subplot(9,1,i+2)
    plt.imshow(slicedImg*2**i,cmap='gray')
    plt.axis(False)
    plt.title(f"{2**i}th level")
plt.show()
```

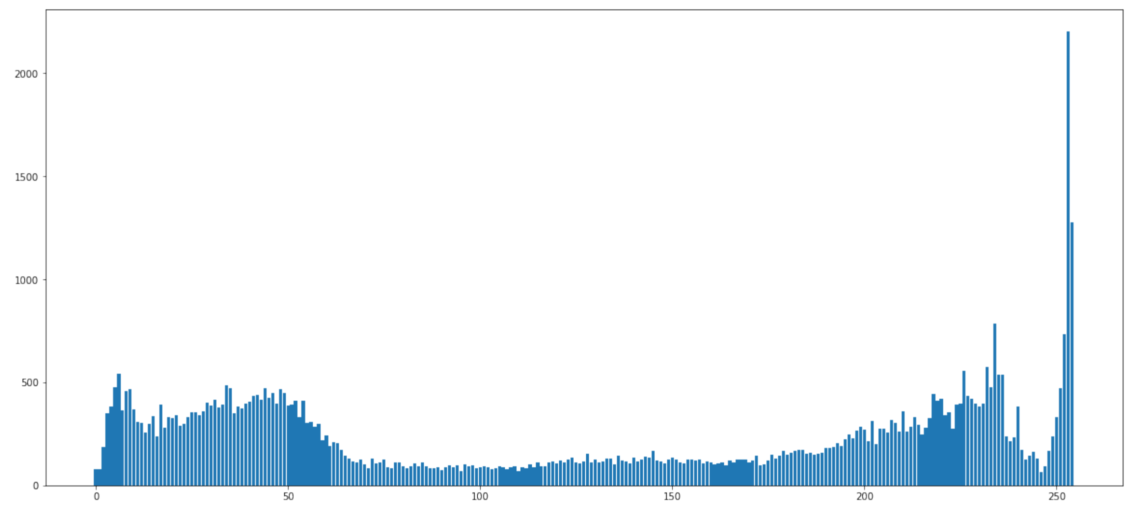

Original Image
1th level
2th level
4th level
8th level
16th level
32th level
64th level
128th level

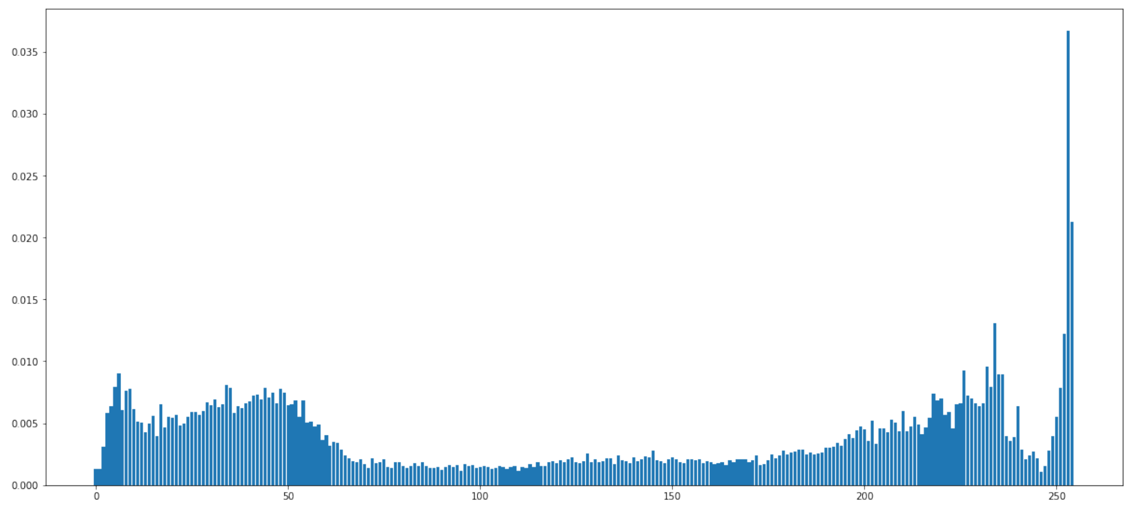
7 Histogram Processing

```
[10]: def Histogram(img):
          grayLevelRepitition=np.zeros(255)
          for line in img:
              for cell in line:
                  grayLevelRepitition[cell-1]+=1
          return grayLevelRepitition

      img=cv2.imread("sample.jpg",0)
      histogram=Histogram(img)
      normalizedHistogram=histogram/(img.shape[0]*img.shape[1])
      print("sum of normalized histogram: ", np.sum(normalizedHistogram))
      plt.figure(figsize=(20,20))
      plt.subplot(2,1,1)
      plt.bar(range(255),histogram)
      plt.subplot(2,1,2)
      plt.bar(range(255),normalizedHistogram)
      plt.show()
```

```
sum of normalized histogram:  1.0
```

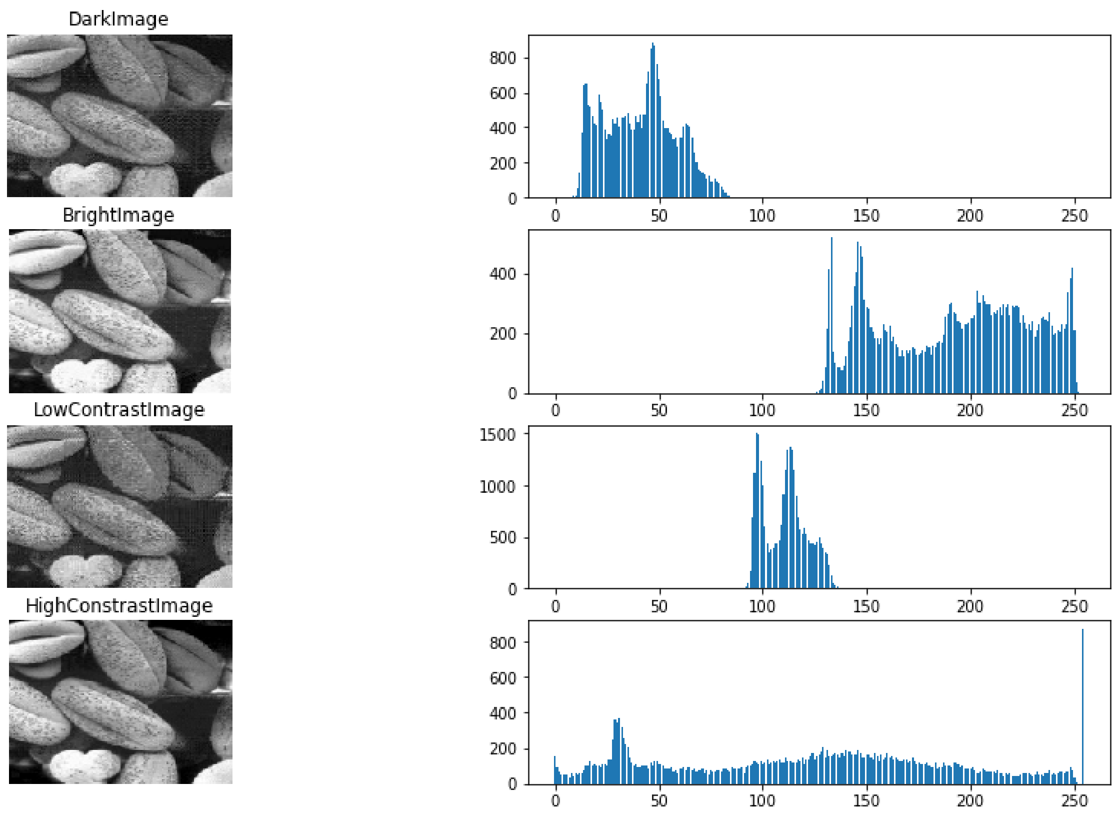




7.1 calculating histogram for different images

```
[11]: testCases=["DarkImage-forHistogram.png",
                 "BrightImage-forHistogram.png",
                 "LowContrastImage-forHistogram.png",
                 "HighConstrastImage-forHistogram.png"]
      i=0
      plt.figure(figsize=(15,9))
      for pic in testCases:
          i+=1
          plt.subplot(len(testCases),2,i)
          img=cv2.imread(pic,0)
          plt.imshow(img,cmap='gray')
```

```
    plt.title(pic.split("-")[0])
    plt.axis(False)
    i+=1
    plt.subplot(len(testCases),2,i)
    plt.bar(range(255),Histogram(img))
```



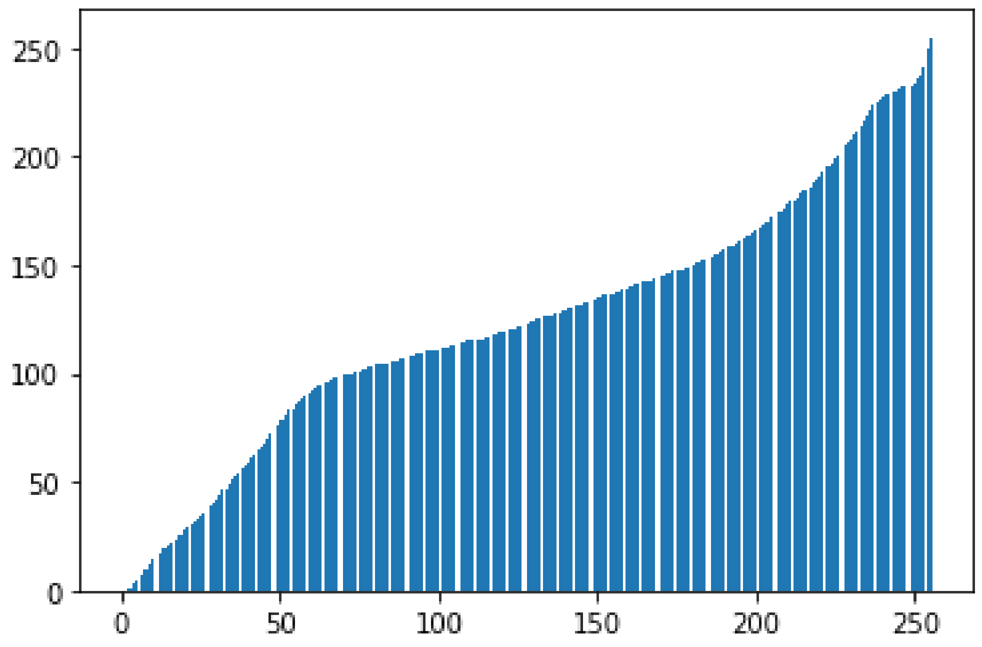
7.2 Histogram Equalization

$$h(v) = \text{round}\left(\frac{cdf(v) - cdf_{min}}{(M \times N) - cdf_{min}} \times (L-1)\right)$$

Where: cdf stands for *Cumulative Distribution Function*
Obviuosly: cdf_{min} is the minimum non-zero value of the cumulative distribution function
Also: MN gives the image's number of pixels (where M is width and N the height)
And: L is the number of grey levels used (in most cases 256).

[12]:
```
def HistogramEqualization(img):
    histogram=np.histogram(img.ravel(),256,[0,256])[0]
    cdf=np.cumsum(histogram)
    cdf_min=np.min(cdf[cdf!=0])
    size=img.shape[0]*img.shape[1]
    return np.round((cdf-cdf_min)/(size-cdf_min)*255)

img=cv2.imread("sample.jpg",0)
# print(HistogramEqualization(img))
plt.bar(range(256),HistogramEqualization(img))
plt.show()
```



[13]:
```
def Image_of_EqulizedHistogram(img):
    histogram=np.histogram(img.ravel(),256,[0,256])[0]
    cdf=np.cumsum(histogram)
    cdf_min=np.min(cdf[cdf!=0])
    size=img.shape[0]*img.shape[1]
    EqulizedHistogram=np.round((cdf-cdf_min)/(size-cdf_min)*255)
    newImg=img.copy()
    for i in range(256):
        newImg[img==i]=EqulizedHistogram[i]
    return newImg


testCases=['sample.jpg','HistogramEqualization.png','Low-contrast-image.png']
i=0
plt.figure(figsize=(20,25))
for pic in testCases:
    img=cv2.imread(pic,0)
    i+=1
    plt.subplot(len(testCases),2,i)
    plt.imshow(img,cmap='gray')
    plt.axis(False)
    i+=1
    plt.subplot(len(testCases),2,i)
    plt.imshow(Image_of_EqulizedHistogram(img),cmap='gray')
    plt.axis(False)
plt.show()
```

8 Enhancement using Arithmetic/Logic Operations

8.1 Arithmetic Operations

$$AND \begin{cases} x.0 = 0 \\ x.1 = x \end{cases}$$

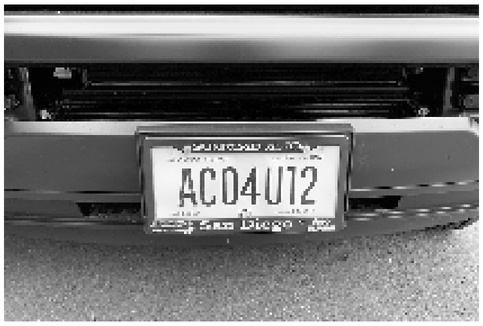
$$OR \begin{cases} x + 0 = x \\ x + 1 = 1 \end{cases}$$

$$XOR \begin{cases} x \oplus 1 = \overline{x} \\ x \oplus 0 = x \end{cases}$$

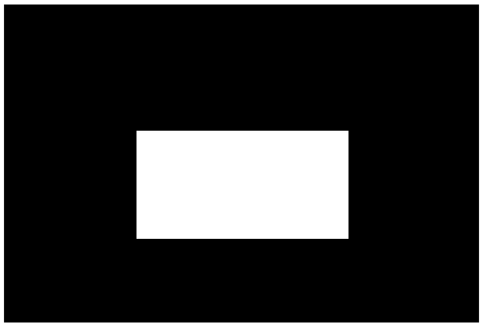
[14]:
```
def LogicOperations(orignImg,originalMask,operation):
    if operation not in ['AND','OR','XOR']:
        raise ValueError(f"the operation must be either AND, OR, XOR\n!!!")
    img=orignImg.copy()
    mask=originalMask.copy()
    minRow=min(mask.shape[0],img.shape[0])
    minCol=min(mask.shape[1],img.shape[1])
    mask=mask[:minRow,:minCol]
    img=img[:minRow,:minCol]
    mask[mask<124]=0
    mask[mask>124]=1
    if operation=='AND':
        img[mask==0]=0
    elif operation=='OR':
        img[mask==1]=255
    elif operation=='XOR':
        notImg=255-img
        img[mask==1]=notImg[mask==1]
    return img

img=cv2.imread("ArithmaticOperations.jpg",0)
plt.figure(figsize=(10,15))
plt.subplot(5,1,1)
plt.imshow(img,cmap='gray')
plt.axis(False)
plt.title("Original Image")
mask=cv2.imread('maskForArithmaticOperation.jpg',0)
plt.subplot(5,1,2)
plt.imshow(mask,cmap='gray')
plt.axis(False)
plt.title("Mask")
i=2
for op in ['AND','OR','XOR']:
    i+=1
    plt.subplot(5,1,i)
    plt.imshow(LogicOperations(img,mask,op),cmap='gray')
    plt.axis(False)
    plt.title(f"{op} operation")
plt.show()
```

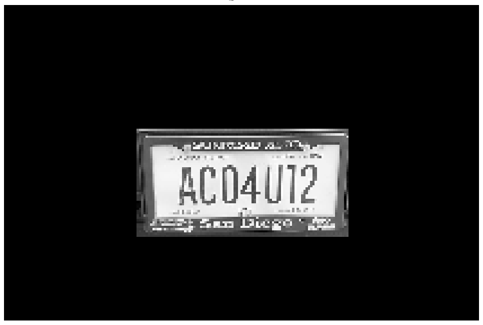
Original Image



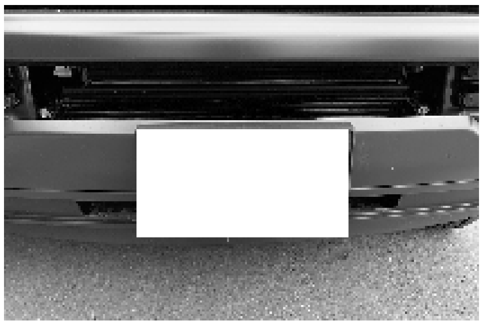
Mask



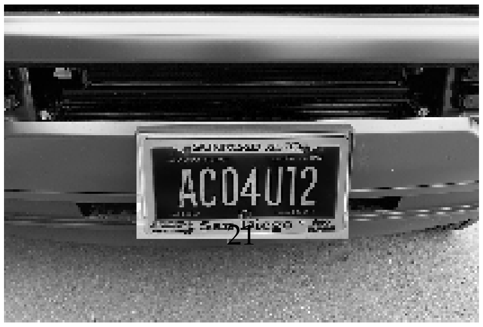
AND operation



OR operation



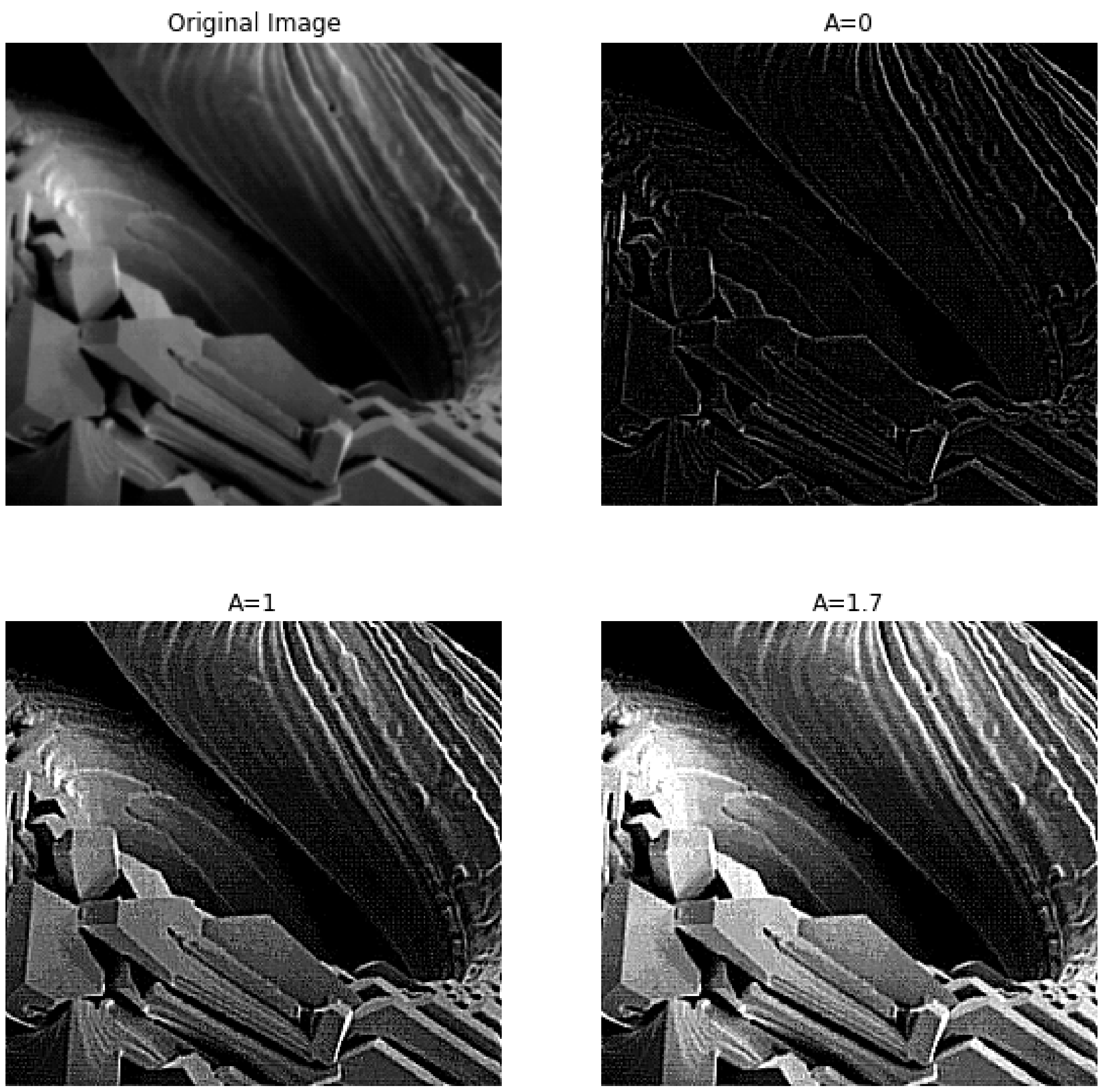
XOR operation



9 High-boost Filtering

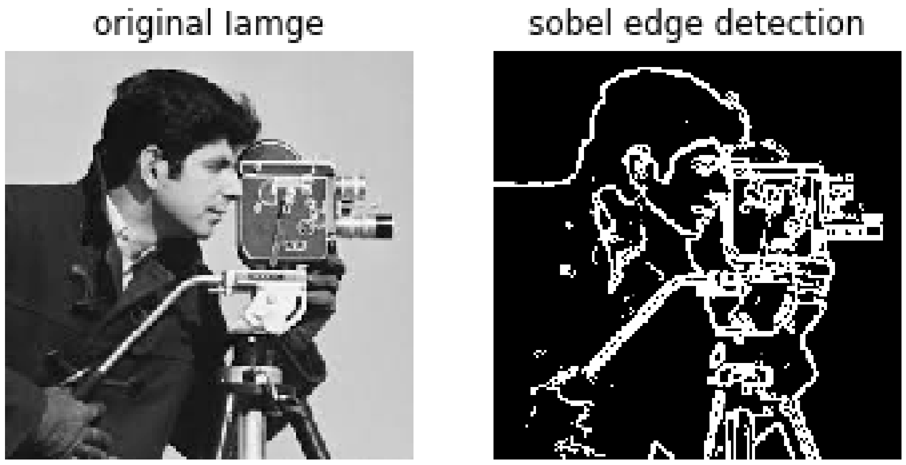
```
[59]: def HighBoostFiltering(originImg,A):
          HBFilter=np.array([[-1,-1  ,-1],
                             [-1,A+8 ,-1],
                             [-1,-1  ,-1]])
          return cv2.filter2D(originImg,-1,HBFilter)

      img=cv2.imread("HighBoostFilter.png",0)
      plt.figure(figsize=(10,10))
      plt.subplot(2,2,1)
      plt.imshow(img,cmap='gray')
      plt.axis(False)
      plt.title("Original Image")
      i=1
      for A in [0,1,1.7]:
          i+=1
          plt.subplot(2,2,i)
          plt.imshow(HighBoostFiltering(img,A),cmap='gray')
          plt.axis(False)
          plt.title(f"A={A}")
      plt.show()
```



10 Sobel Edge Detector

[37]:
```
img=cv2.imread("sobel.jpeg",0)
grad_x = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=3)
grad_y = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=3)
abs_grad_x = cv2.convertScaleAbs(grad_x)
abs_grad_y = cv2.convertScaleAbs(grad_y)
sobel = cv2.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
sobel[sobel<100]=0
sobel[sobel>100]=255
plt.subplot(1,2,1)
plt.imshow(img,cmap='gray')
plt.axis(False)
plt.title("original Iamge")
plt.subplot(1,2,2)
plt.imshow(sobel,cmap='gray')
plt.axis(False)
plt.title("sobel edge detection")
plt.show()
```



```
[60]: print("Finished! :)")
```

```
Finished! :)
```