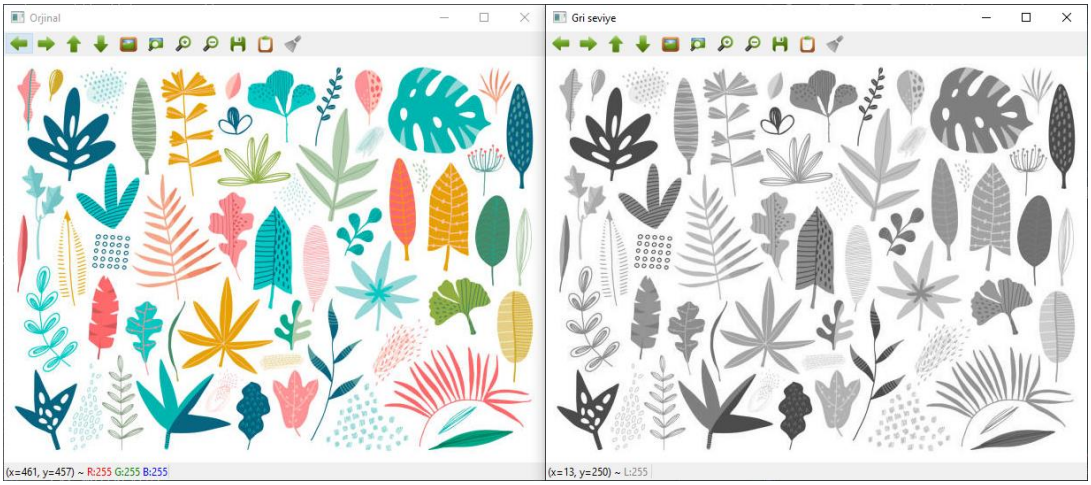
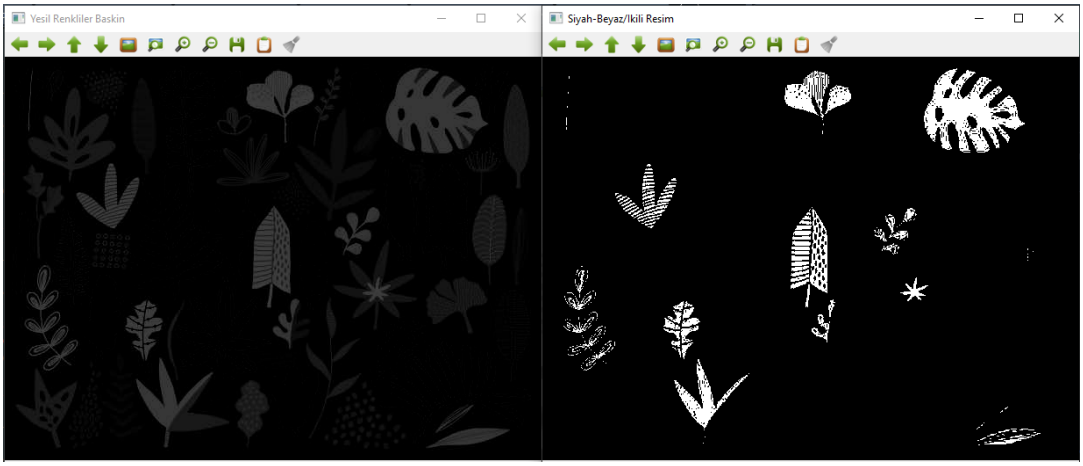
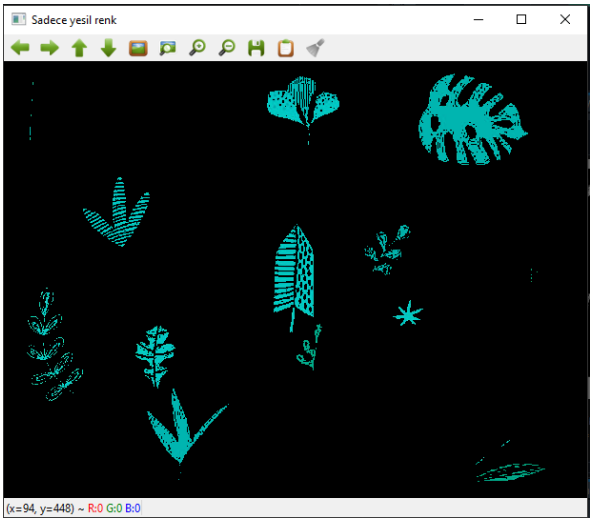
Bir resimde kırmızı renkli nesneleri bulma

```python
# -*- coding: utf-8 -*-
"""
Created on Sun Nov  6 14:05:30 2022

@author: Mekatronik
"""

import cv2
from matplotlib import pyplot as plt
x=cv2.imread("balon.jpg")
xGri=cv2.cvtColor(x,cv2.COLOR_BGR2GRAY)
xred=x[:,:,2]
kirmiziRenk=cv2.subtract(xred,xGri)
cv2.imshow('Orjinal',x)
cv2.imshow('Gri seviye',xGri)
cv2.imshow('Kirmizi Renkliler Baskin',kirmiziRenk)
z,xIkili=cv2.threshold(kirmiziRenk,60,255,cv2.THRESH_BINARY)
plt.figure(2)
plt.imshow(xIkili,'gray',vmin=0,vmax=255)
#Aşağıda xIkili tek katmanlı olduğu için 3 katmanlı olan x ile AND işlemine girmiyordu.
#Bu nedenle xIkili'yi 3 katmanlı hale getriyoruz.
xIkili_RGB=cv2.cvtColor(xIkili, cv2.COLOR_GRAY2RGB)
sadeceKirmizi=cv2.bitwise_and(xIkili_RGB,x)
cv2.imshow('Sadece kirmizi renk',sadeceKirmizi)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Benzer bir kodu **yeşil** renkli nesneler için yazalım: Yapılacak işlem: sadece 0. Satırdaki resim dosyasının adını değiştirmek (desenler.jpg) ve 2. Satırda yeşil bant için xRed=x[:,:,1] yapmaktır. Ayrıca 7. Satırdaki threshold eşiğini de bir miktar değiştirmek gerekebilir.







Yukarıdaki örnekler ile aslında AND/OR/NOT gibi mantıksal işlemlerin önemini anlamış oluyoruz. Bu operatörler sayesinde siyah beyaza çevirdiğimiz nesnelerin orijinal görüntülerini resimden söküp alabiliyoruz.

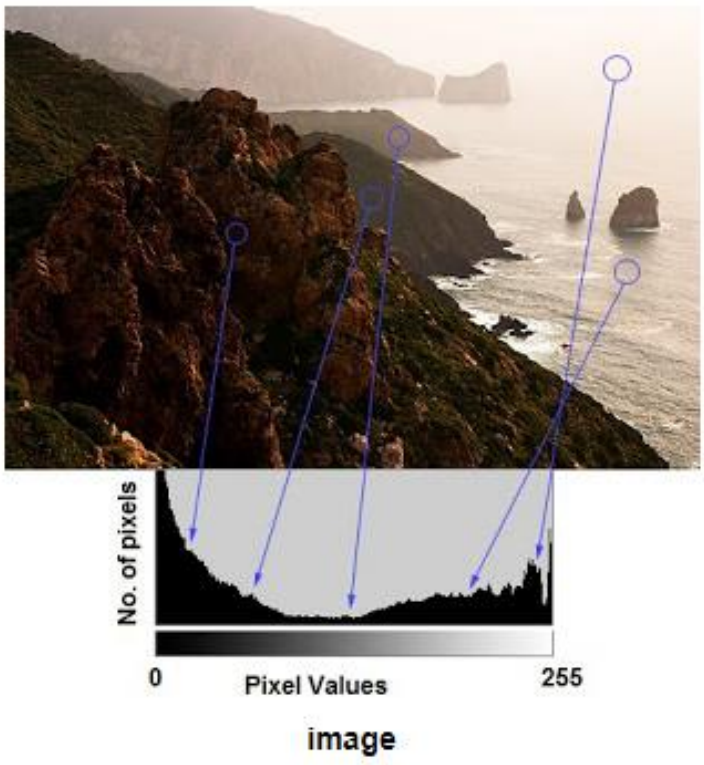
Histogram Kavramı ve Histogram Eşitleme

Histogram bir resimdeki renk değerlerinin sayısını gösteren grafiktir. Histogram dengeleme veya eşitleme de bir resimdeki renk değerlerinin belli bir yerde kümelenmiş olmasından kaynaklanan, renk dağılımı bozukluğunu gidermek için kullanılan bir yöntemdir.

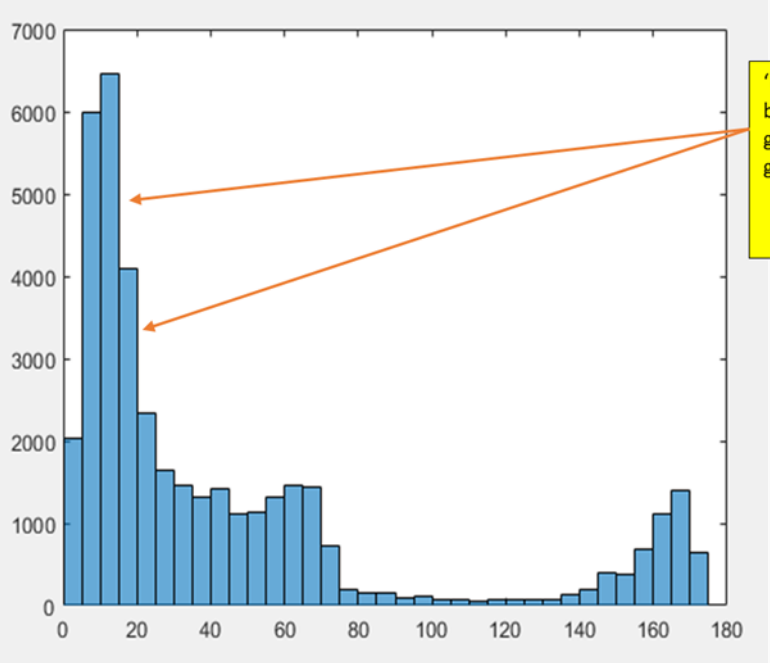
- Histogram matematiksel olarak aşağıdaki şekilde gösterilebilir.
- $h(r_k)=n_k$

r_k: k'nıncı parlaklık değeri
n_k: k nıncı parlaklık değerinin görüntüdeki sayısı

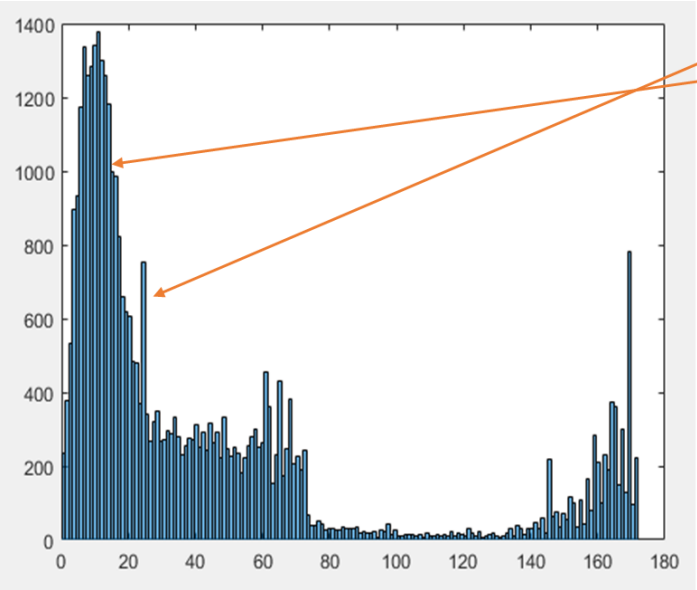
Bir Histogram Örneği:



“Bin” sayısı kavramı:



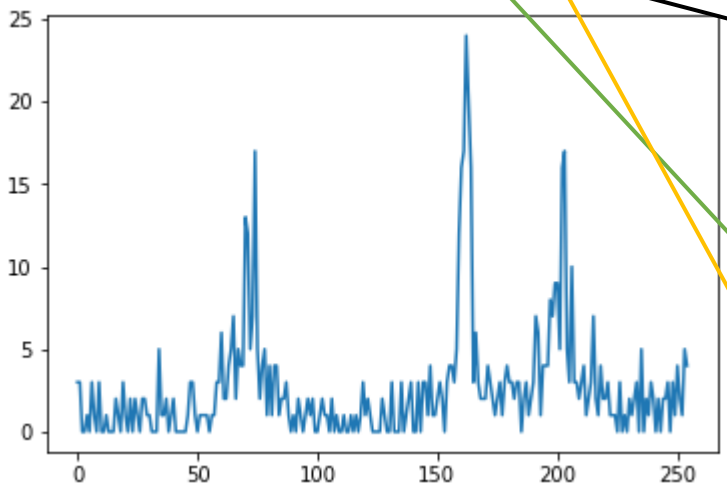
‘Bin’ olarak tabir edilen çubuk barlar. Bunların sayısı istenildiği gibi değiştirilebilir. Bu grafiktebin sayısı: 35’dir.



Bu grafikte bin sayısı 165’dir.

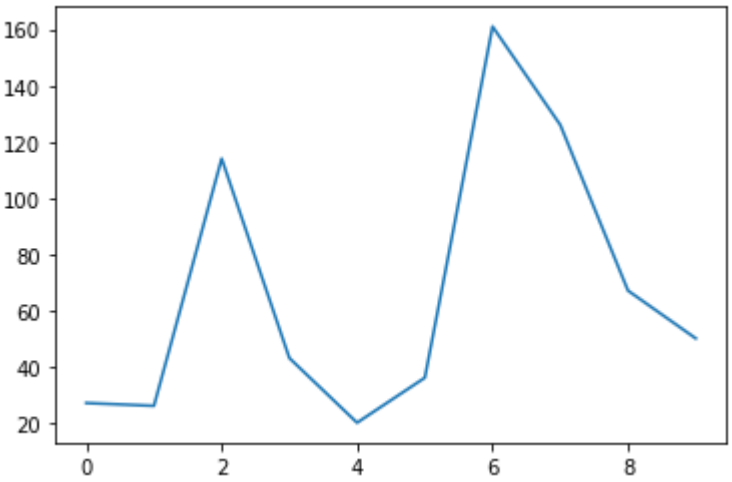
Opencv Örneği:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("balonlar2.jpg")
hist = cv2.calcHist([img],[0],None,[255],[0,255])
plt.plot(hist)
```



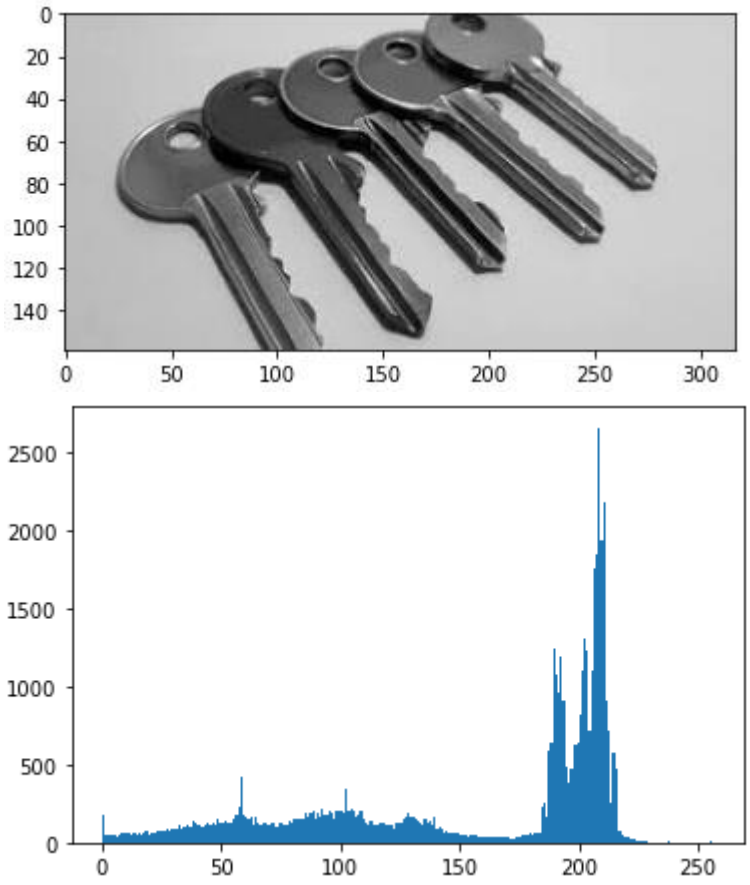
1. images : uint8 veya float32 türündeki kaynak görüntüdür. köşeli parantez içinde verilmelidir, yani "[img]".
2. channels : bu da köşeli parantez içinde verilir. Histogramını hesapladığımız kanalın indeksidir. Örneğin, girdi gri tonlamalı görüntü ise, değeri [0]'dır. Renkli görüntü için, sırasıyla mavi, yeşil veya kırmızı kanalın histogramını hesaplamak için [0], [1] veya [2] seçebilirsiniz.
3. maske : maske görüntüsü. Tam görüntünün histogramını bulmak için "None" olarak verilir. Ancak görüntünün belirli bir bölgesinin histogramını bulmak istiyorsanız, bunun için bir maske görüntüsü oluşturmanız ve bunu maske olarak vermeniz gerekir. (Daha sonra bir örnek göstereceğim.)
4. histSize : bu bizim BIN sayımızı temsil eder. Köşeli parantez içinde verilmesi gerekir. Tam ölçek için [256] geçiyoruz.
5. ranges : bu bizim RANGE'ımızdır. Normalde [0,255] şeklindedir.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("balonlar2.jpg")
hist = cv2.calcHist(img,[0],None,[10],[0,255])
plt.plot(hist)
```



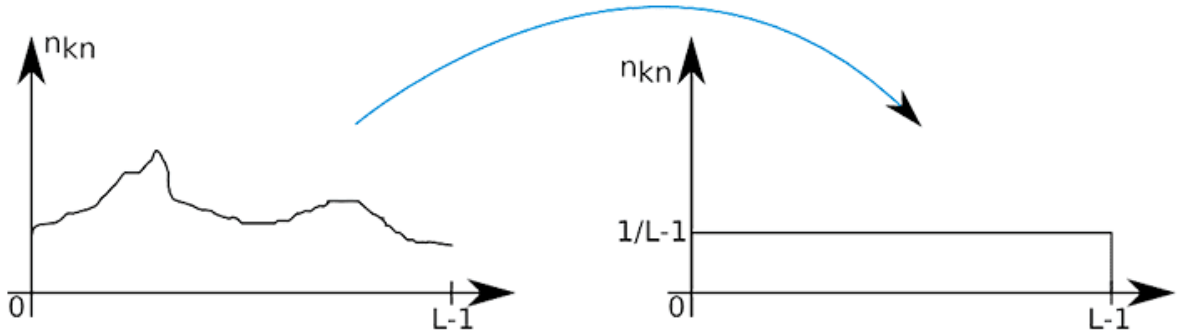
Doğrudan histogramı da çizdirebiliriz. Yukarıdaki histogramı önce hesaplayıp sonra çizdirmiştik.

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('anahtar.jpg',0)
plt.imshow(img, cmap='gray', vmin=0, vmax=255)
plt.figure()
plt.hist(img.ravel(),256,[0,256])
plt.show()
```



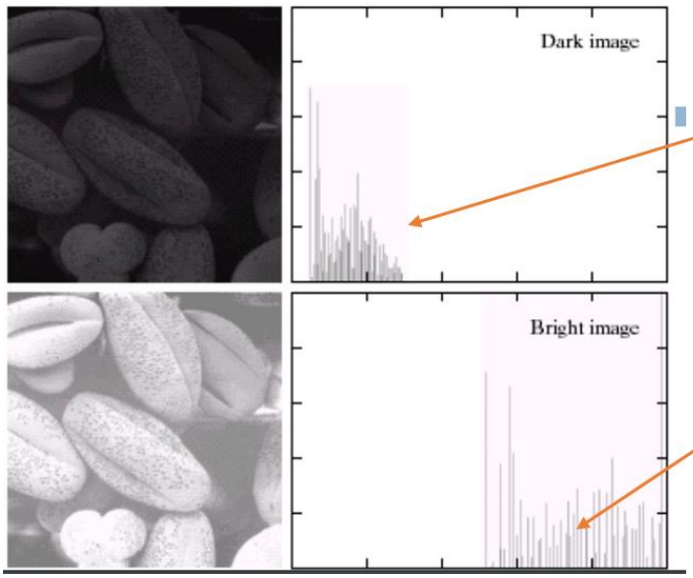
Histogramımızın nasıl çok sayıda tepe noktasına sahip olduğuna dikkat edin, bu da ilgili aralıklara çok sayıda pikselin yerleştirildiğini gösterir. Histogram eşitleme ile amacımız bu pikselleri, kendilerine çok fazla piksel atanmamış aralıklara yaymaktır.

Matematiksel olarak bunun anlamı, kümülatif dağılım fonksiyonumuza (CDF) doğrusal bir eğilim uygulamaya çalıştığımızdır:



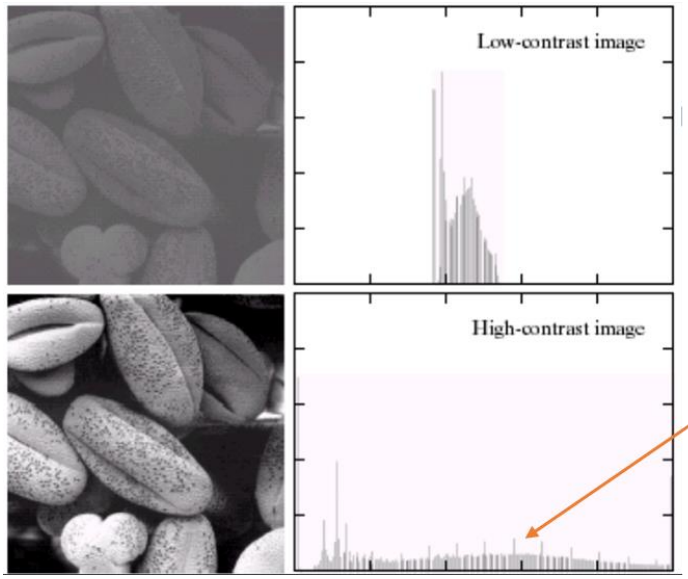
https://pyimagesearch.com/2021/02/01/opencv-histogram-equalization-and-adaptive-histogram-equalization-clahe/

Bir başka örnek:



Piksel parlaklık değerleri 0'a yakın bölgede toplanmış
Dolayısıyla karanlık bir imge.

Piksel parlaklık değerleri 255'e yakın bölgede toplanmış
Dolayısıyla aşırı aydınlık bir imge.

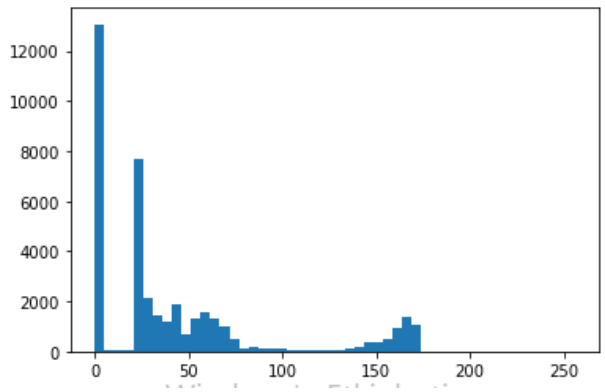


Histogramı eşitlenmiş imge.
Piksel parlaklık değeri homojen dağılmış.

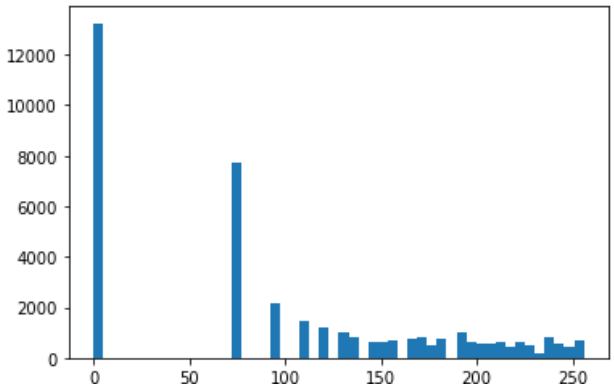
Histogram Eşitleme

Bir görüntüyü önce karartalım, sonrada histogramını eşitleyerek kalitesini arttıralım:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
gri=cv2.imread("headquarters-2K.png",0)
#griKaranlik=gri-30;
equ = cv2.equalizeHist(gri)
z=cv2.hconcat([gri, equ])
cv2.imshow("resimler",z)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Orijinal

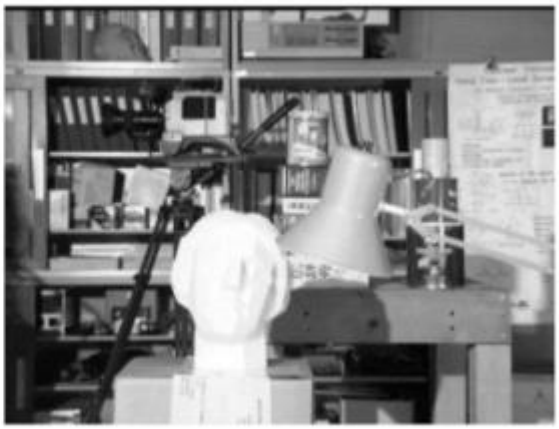


Histogram Eşitlenmiş

Adaptif Histogram Eşitleme

Bir önceki blogda, bir görüntünün global kontrastını dikkate alan Histogram Eşitleme'yi tartıştık. Bu, tüm görüntü piksellerini dönüştürmek için aynı dönüşüm fonksiyonunun kullanıldığı anlamına gelir. Bu yaklaşım çoğu durumda iyi çalışır ancak görüntü, görüntünün çoğundan önemli ölçüde daha açık veya daha koyu olan bölgeler içerdiğinde, bu bölgelerdeki kontrast yeterince geliştirilmeyecektir. Aşağıdaki resimde heykelin yüzüne bakın

Orijinal İmge



Histogram Eşitlenmiş İmge

Bazen de görüntünün tamamı yerine küçük alanlardaki ayrıntıları geliştirmek isteriz. Bu sorun, görüntüdeki her pikselin komşuluğundan türetilen bir dönüşüm fonksiyonu kullanırsak çözülebilir.

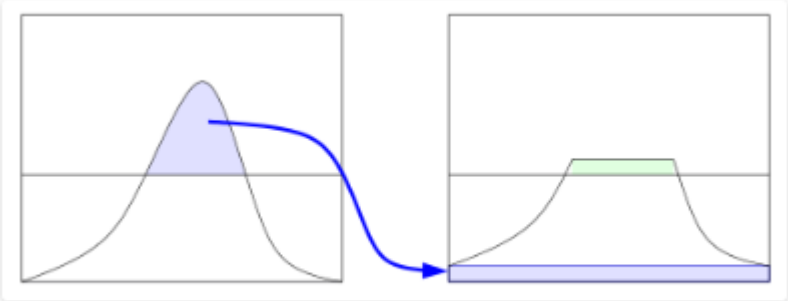
Uyarlanabilir Histogram Eşitleme'de (AHE) görüntü "karo" adı verilen küçük bloklara bölünür (örneğin 64 karo (8x8) yaygın bir seçimdir).

Daha sonra bu blokların her biri daha önce yaptığımız gibi histogram eşitlenir.

Son olarak, bu blokları bilineer enterpolasyon kullanarak birleştiriyoruz.

Ancak bu yöntemin bir sorunu vardır, eğer piksel değerleri bir miktar gürültü içeren bir blokta aşağı yukarı sabitse, AHE gürültüyü aşırı yükseltme eğilimindedir. Bunu önlemek için kontrast sınırlaması uygulanır ve yöntem Kontrast Sınırlı Uyarlanabilir Histogram Eşitleme (CLAHE) olarak bilinir.

CLAHE'de CDF'yi hesaplamadan önce histogramı önceden tanımlanmış bir değerde kırpıyoruz ve aşağıdaki şekilde gösterildiği gibi histogram eşitleme uygulamadan önce diğer kutulara tek biçimli olarak dağıtıyoruz.



Source: Wikipedia

Histogram Eşitlemede kullanılan dönüştürme fonksiyonu CDF ile orantılı olduğundan, kırpma CDF'nin ve dolayısıyla dönüştürme fonksiyonunun eğiminin sınırlandırılmasıyla sonuçlanır. Bu şekilde gürültünün aşırı yükselmesi önlenir. Her piksel için kendi komşuluğundan dönüşüm fonksiyonunu hesapladığımızdan, bu hesaplama açısından yoğun bir işlemdir.

Örnek Kodlar:

```
import numpy as np
import cv2

# Gri seviye imge yükleniyor.
img = cv2.imread('headquarters-2K.png',0)

# CLAHE nesnesi oluşturuyoruz
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
img_AHE = clahe.apply(img)

# İmgeleri yan yana görüntüleyelim
son = cv2.hconcat([img, img_AHE])
cv2.imshow('a',son)
cv2.waitKey(0)
```