



GÖRÜNTÜ İŞLEME YARDIMCI NOTLARI -2023-

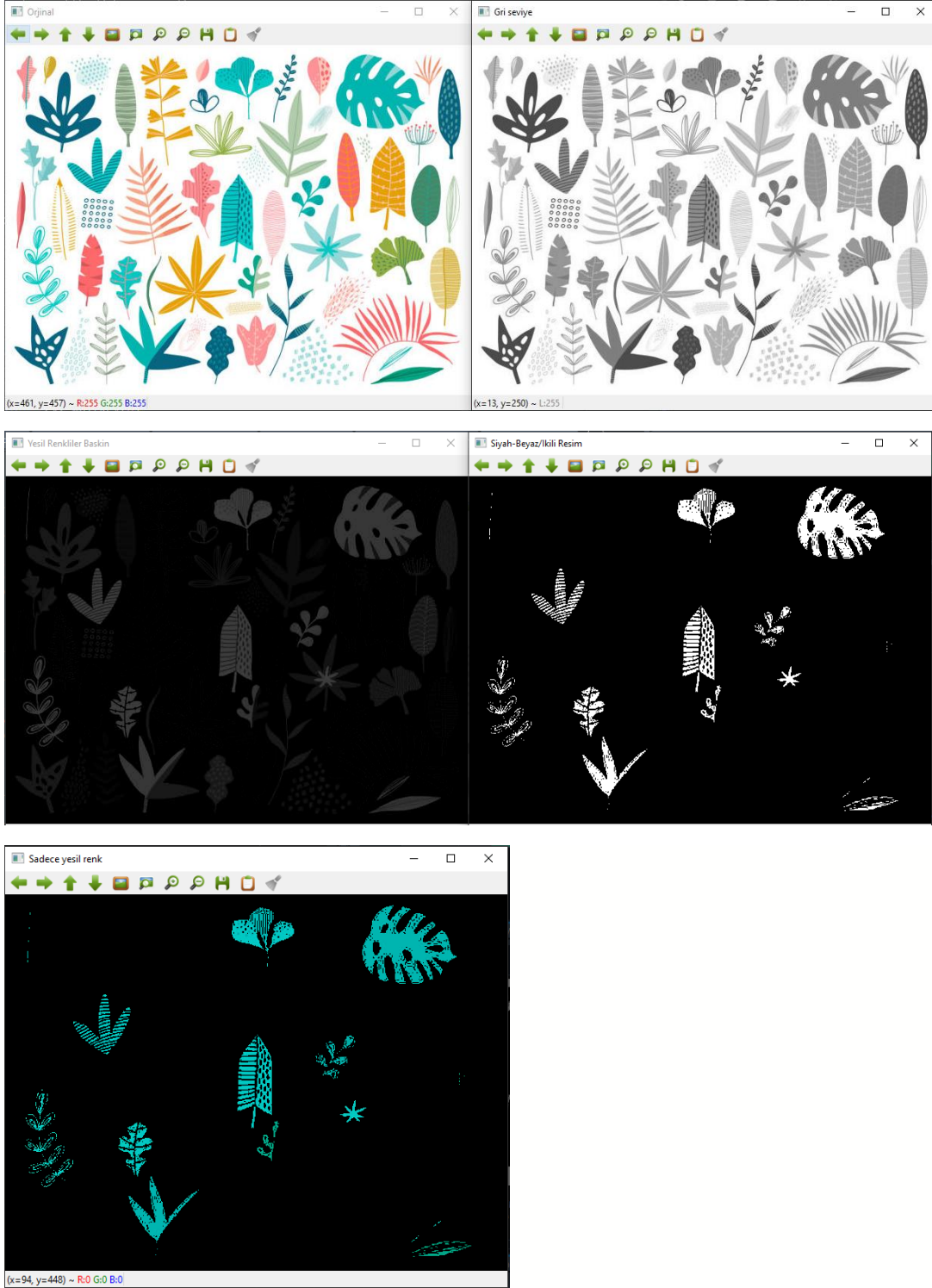


Bir resimde kırmızı renkli nesneleri bulma

```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Sun Nov 6 14:05:30 2022
4
5 @author: Mekatronik
6 """
7
8 import cv2
9 from matplotlib import pyplot as plt
10 x=cv2.imread("balon.jpg")
11 xGri=cv2.cvtColor(x,cv2.COLOR_BGR2GRAY)
12 xred=x[:, :,2]
13 kirmiziRenk=cv2.subtract(xred,xGri)
14 cv2.imshow('Orjinal',x)
15 cv2.imshow('Gri seviye',xGri)
16 cv2.imshow('Kirmizi Renkliler Baskin',kirmiziRenk)
17 z,xIkili=cv2.threshold(kirmiziRenk,60,255,cv2.THRESH_BINARY)
18 plt.figure(2)
19 plt.imshow(xIkili,'gray',vmin=0,vmax=255)
20 #Aşağıda xIkili tek katmanlı olduğu için 3 katmanlı olan x ile AND işlemine girmiyordu.
21 #Bu nedenle xIkili'yi 3 katmanlı hale getiriyoruz.
22 xIkili_RGB=cv2.cvtColor(xIkili, cv2.COLOR_GRAY2RGB)
23 sadeceKirmizi=cv2.bitwise_and(xIkili_RGB,x)
24 cv2.imshow('Sadece kırmızı renk',sadeceKirmizi)
25 cv2.waitKey(0)
26 cv2.destroyAllWindows()
```



Benzer bir kodu **yeşil** renkli nesnelere 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 eşliğini 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

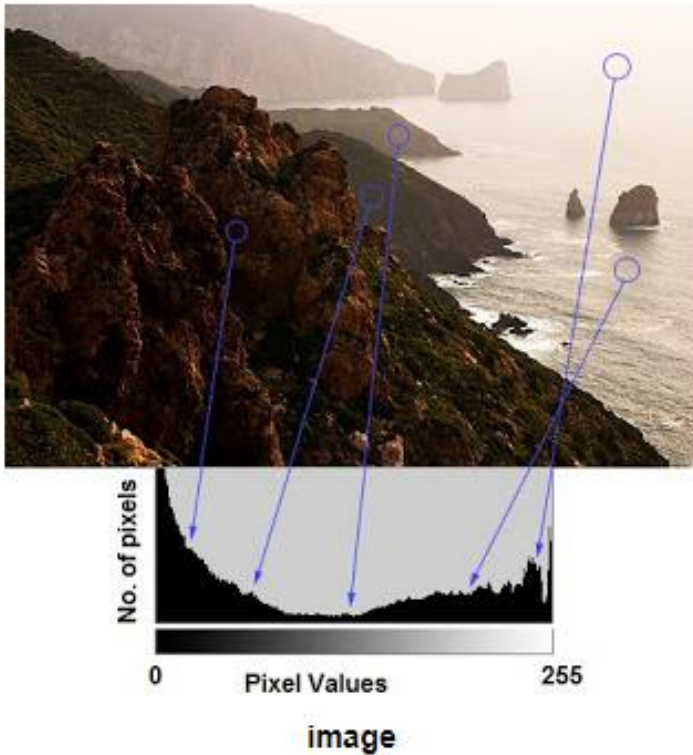
Histogram bir resimdeki renk değerlerinin sayısını gösteren grafikdir. Histogram dengeleme veya eşitleme de bir resimdeki renk değerlerinin belli bir yerde kümelenmiş olmasından kaynaklanan, renk dağılımı bozukluğunu gidermek için kullanılan bir yöntemdir.

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

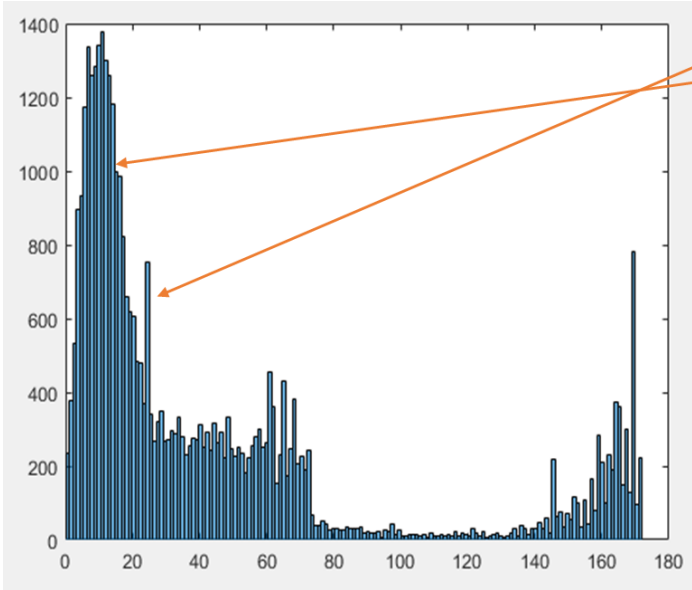
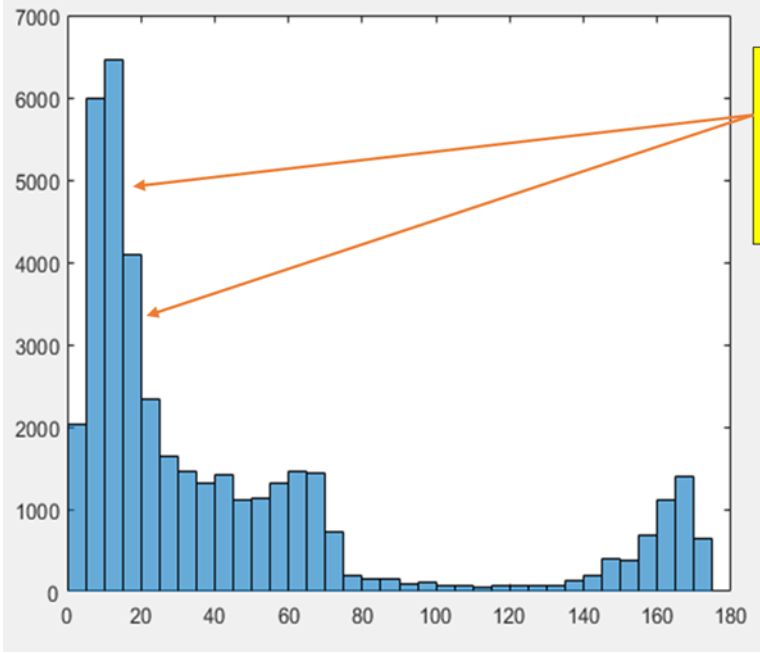
r_k : k'nıncı parlaklık değeri

n_k : k nıncı parlaklık değerinin görüntüdeki sayısı

Bir Histogram Örneği:

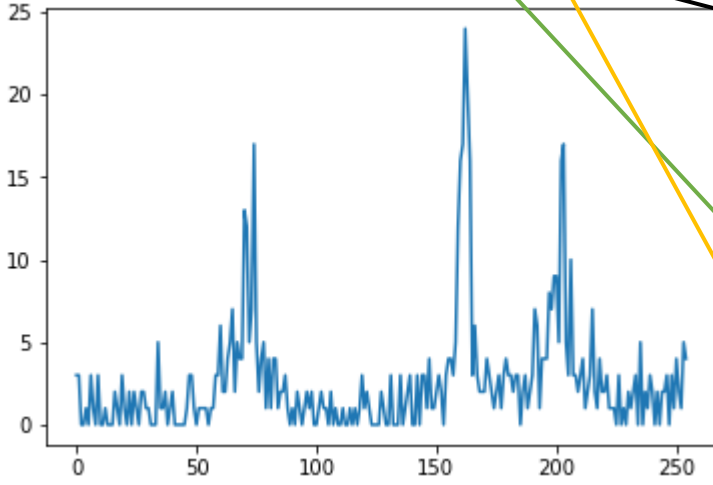


“Bin” sayısı kavramı:



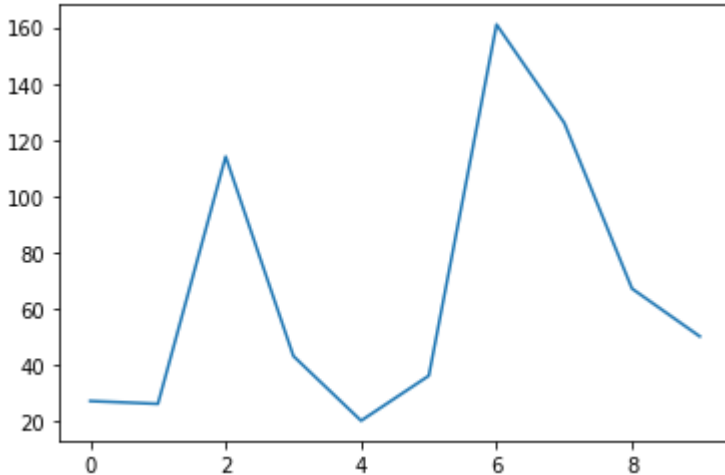
Opencv Örneği:

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



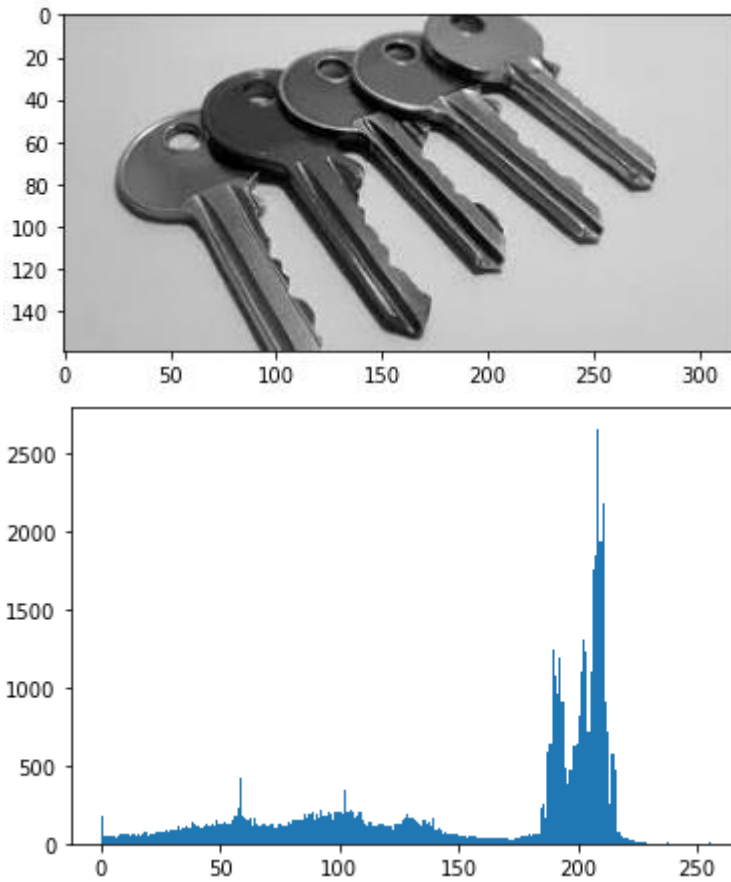
1. images : uint8 veya float32 türündeki kaynak görüntüdür. köşeli parantez içinde verilmelidir, yani "[img]".
2. channels : bu da köşeli parantez içinde verilir. Histogramını hesapladığımız kanalın indeksidir. Örneğin, girdi gri tonlamalı görüntü ise, değeri [0]'dır. Renkli görüntü için, sırasıyla mavi, yeşil veya kırmızı kanalın histogramını hesaplamak için [0], [1] veya [2] geçebilirsiniz.
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'imızdır. 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)
```



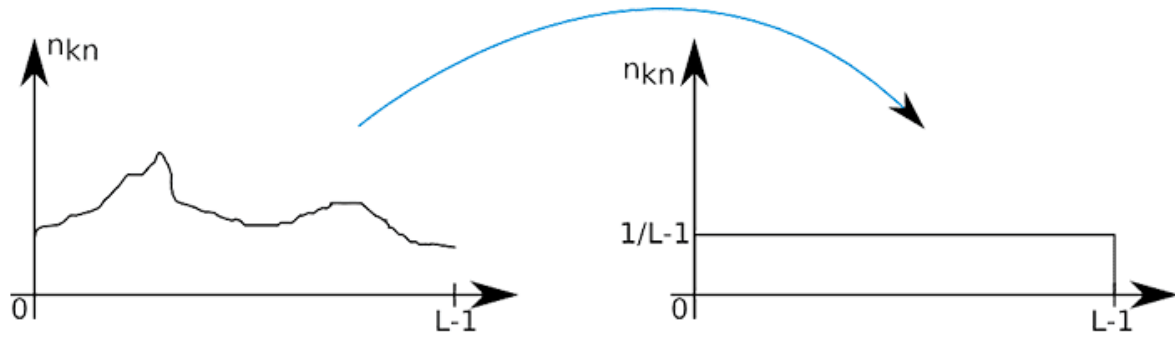
Doğrudan histogramı da çizdirebiliriz. Yukarıdaki histogramı önce hesaplayıp sonra çizdirmiştik.

```
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()
```



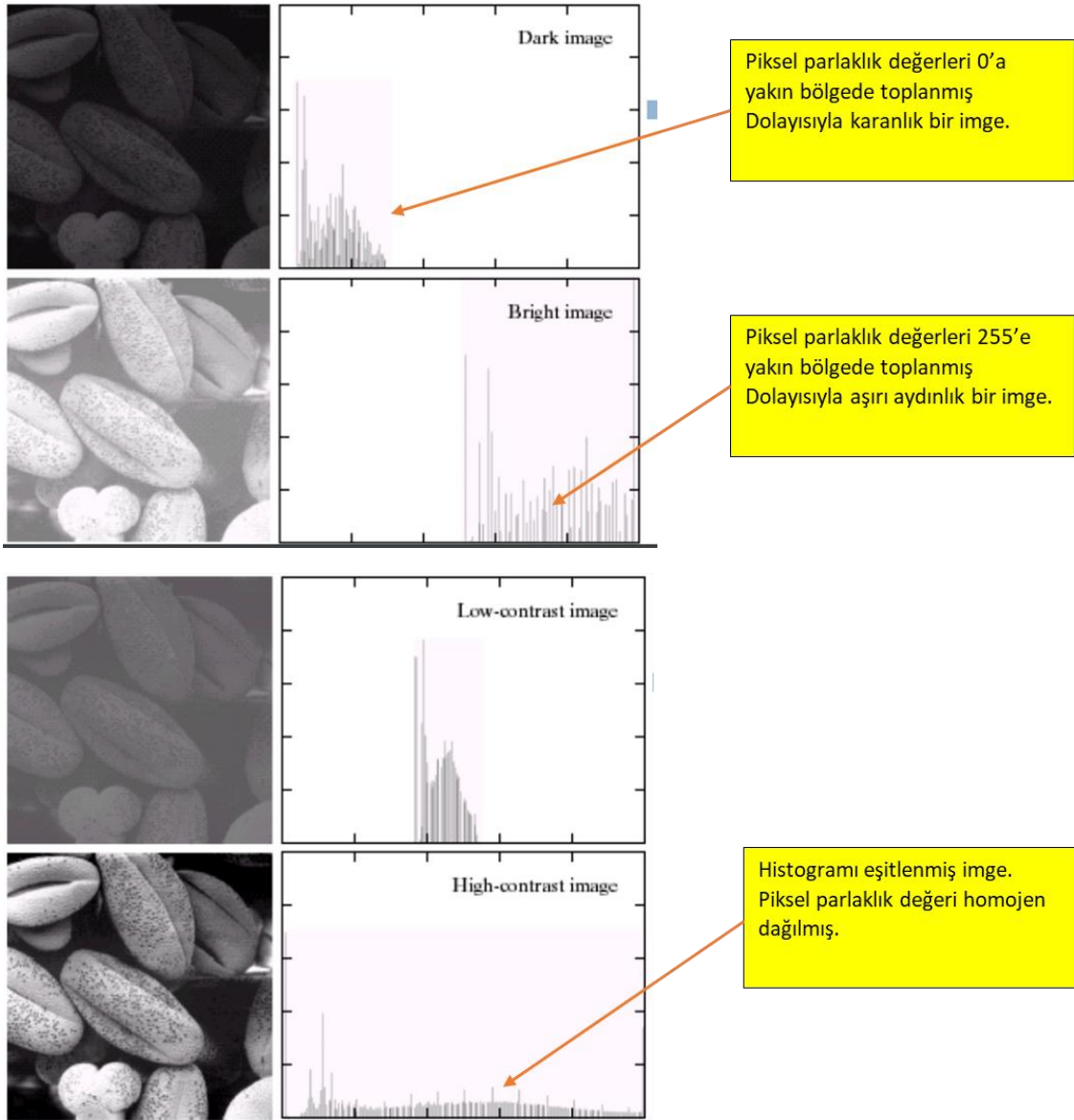
Histogramımızın nasıl çok sayıda tepe noktasına sahip olduğuna dikkat edin, bu da ilgili aralıklara çok sayıda pikselin yerleştirildiğini gösterir. Histogram eşitleme ile amacımız bu pikselleri, kendilerine çok fazla piksel atanmamış aralıklara yaymaktır.

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/>

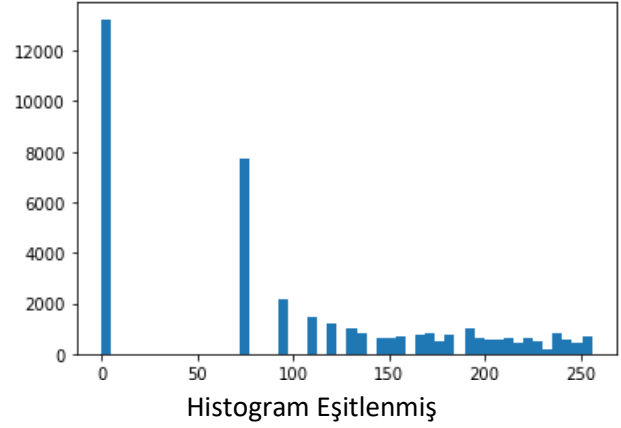
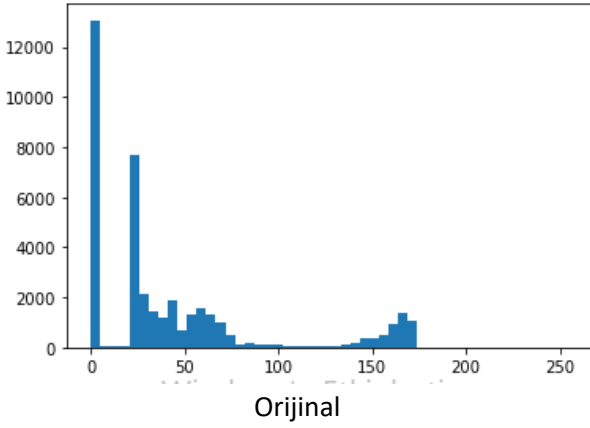
Bir başka örnek:



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()
```



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.

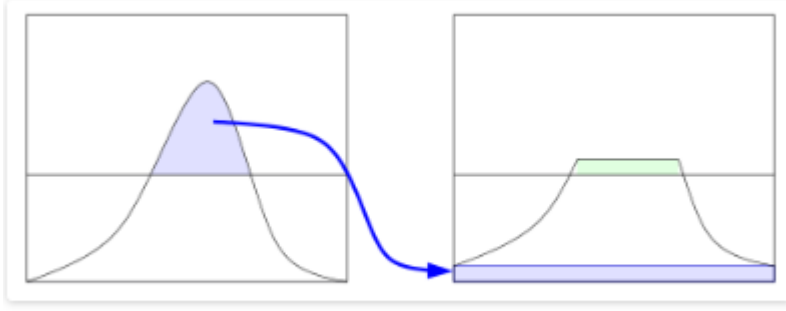
Uyarlanabilir Histogram Eşitleme'de (AHE) görüntü "karo" adı verilen küçük bloklara bölünür (örneğin 64 karo (8x8) yaygın bir seçimdir).

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 kırıyoruz 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)
```

FİLTRELEME İŞLEMLERİ

Görüntü filtreleme, piksellerin renklerini/parlaklık değerlerini değiştirerek görünümünü değiştirmektir. Kontrastı artırmanın yanı sıra görüntülere çeşitli özel efektler eklemek de filtre uygulamanın sonuçlarından bazılarıdır.

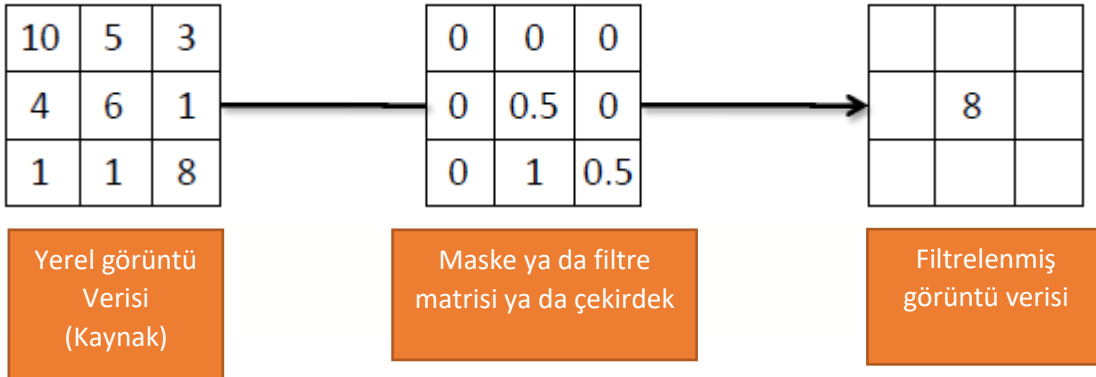
Filtreler görüntü zenginleştirme amacı ile de uygulanan, adından da anlaşılacağı gibi görüntüde belirli ayrıntıların ayıklanması ya da daha belirgin hale getirilmesi vb. gibi operasyonları gerçekleştiren operatörlerdir.

Farklı amaçlar için farklı filtreleme operatörleri vardır. Bunlara:

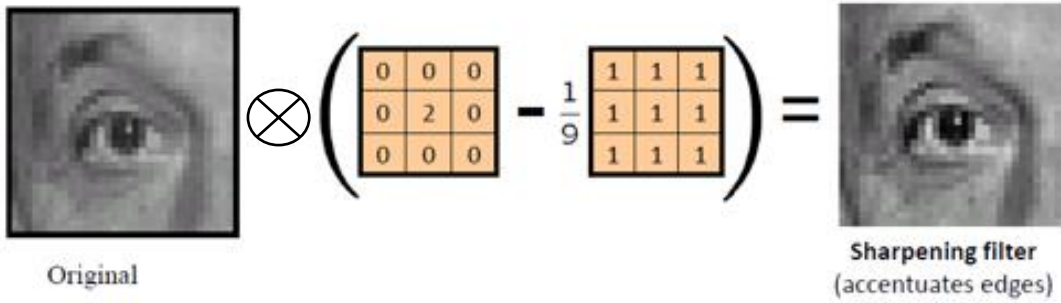
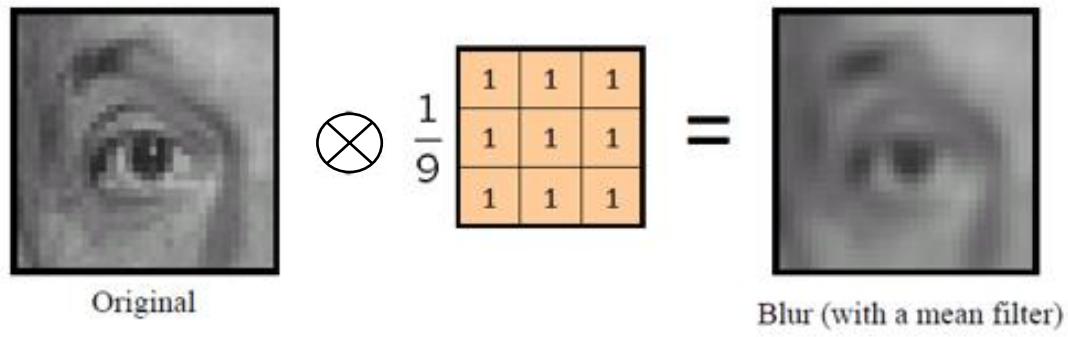
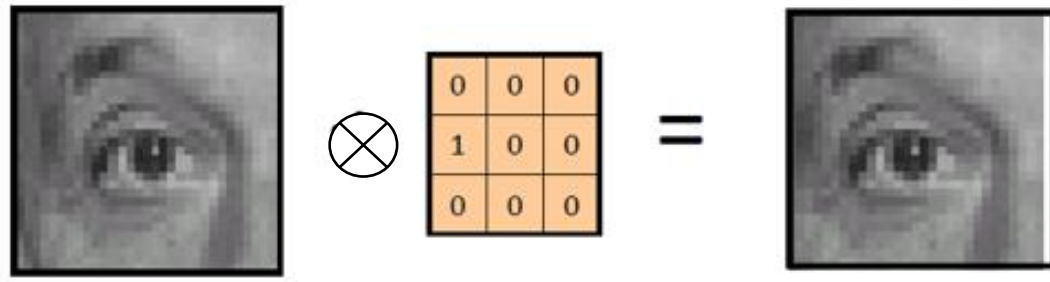
- Gürültü Yok etme
- Kenar keskinleştirme
- Kenar yakalama
- Görüntü yumuşatma ve bunun gibi daha birçok amaçla kullanılan filtreler örnek verilebilir.

Sürecin en önemli parçalarından birisi kernel, çekirdek vs. gibi isimlerle anılan filtre matrisleridir.

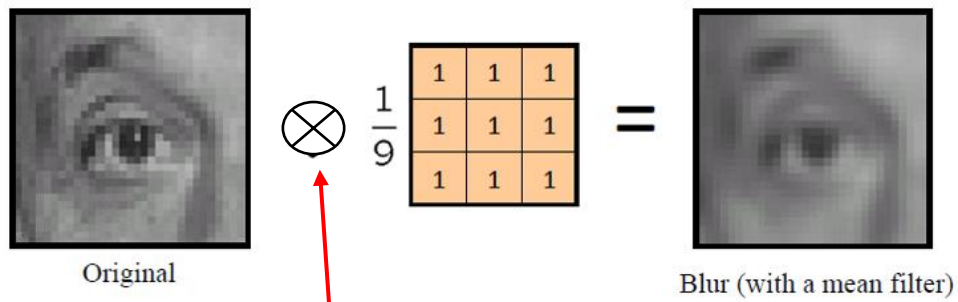
Filtreler, kaynak görüntülerle aşağıdaki resimlerde verildiği şekilde işleme tabi tutulur ve ortaya filtrelenmiş görüntü çıkar.



Bir kısım farklı filtrelerle gerçekleştirilmiş işlemler:



Filtre matrisleri ile orijinal görüntüler arasındaki etkileşim konvolüsyon olarak tanımlanmıştır.

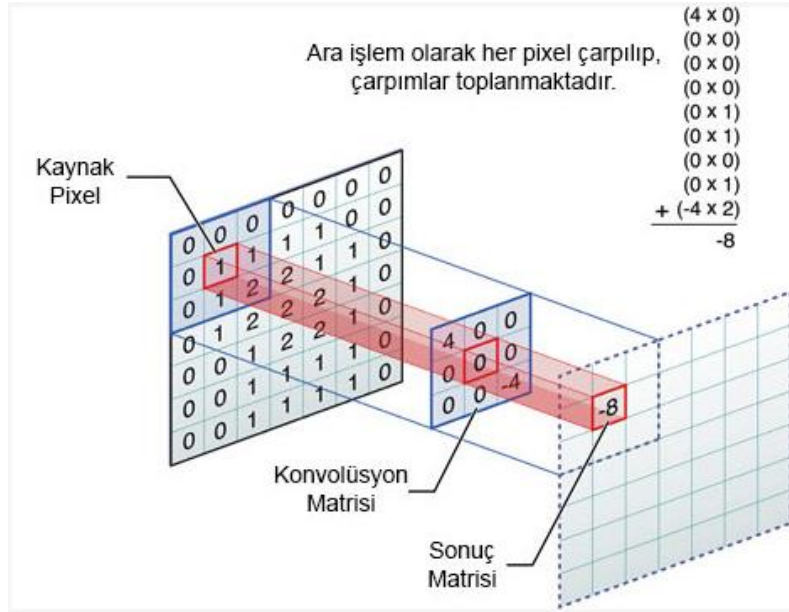


Konvolüsyon işlemi

İki boyutlu korelasyon ve konvolüsyon:

İki boyutlu korelasyon ve konvolüsyon prensip olarak bir boyutluyla aynıdır. Konvolüsyon işleminde kullanılan filtre korelasyon işleminde kullanılanın 180° döndürülmüş halidir.

2 Boyutlu konvolüsyon işlemini gösteren şekil aşağıda gösterilmiştir. Bu defa filtre (maske, şablon, çekirdek) matrisinin tam merkezindeki piksel kaynak görüntü matrisinin sol üst köşesindeki piksele karşılık gelecek şekilde işlem yapılır. Filtre matrisinde boşta kalan pikseller için kaynak matriste sıfırlardan oluşan satır ve sütunlar eklenir (zero padding).

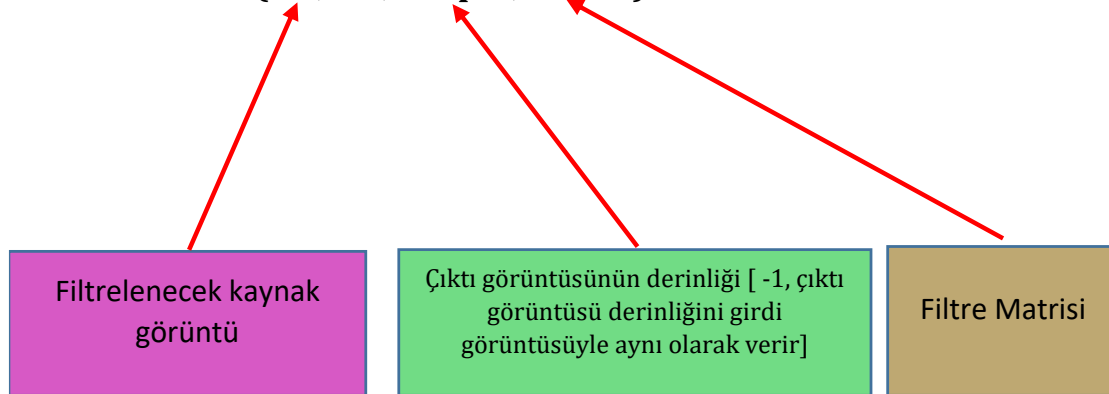


Sonuç olarak korelasyon ve konvolüsyonun matematiksel eşitliği:

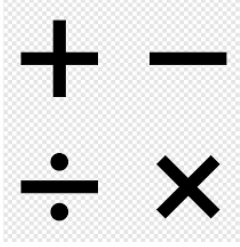
$$w(x, y) \star f(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s, t) f(x + s, y + t)$$

Filtreleme İşlemleri:

cv2.filter2D (src, dst, ddepth, kernel)



Örnek:



Yukarıdaki resimde yatay, dikey ve diagonal doğru parçaları mevcuttur. Amacımız doğru filtreler seçerek konvolüsyon işlemi yapmak ve resimdeki istediğimiz kısmı (yatay, dikey ve diagonal) parlatmak, istemediğimiz kısımları ise sönmölemektedir.

Bunun için filtrelerimizi oluşturalım.

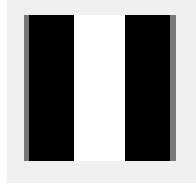
`yatayFiltre =`

```
-1  -1  -1
 2   2   2
-1  -1  -1
```



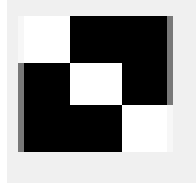
`dikeyFiltre =`

```
-1   2  -1
-1   2  -1
-1   2  -1
```



`diagFiltre =`

```
 2  -1  -1
-1   2  -1
-1  -1   2
```



```

import cv2
import numpy as np
from matplotlib import pyplot as plt
# img=cv2.imread("headquarters-2.jpg",0)
img=cv2.imread("dortIslem.png",0)
np.array([[ -1, 2, -1],
          [ -1, 2, -1],
          [ -1, 2, -1]])
yatayFiltre=np.array([[ -1, -1, -1],
                     [ 2, 2, 2],
                     [ -1, -1, -1]])

dikeyFiltre=np.array([[ -1, 2, -1],
                     [ -1, 2, -1],
                     [ -1, 2, -1]])

diagFiltre=np.array( [[2, -1, -1],
                    [-1, 2, -1],
                    [-1, -1, 2]])

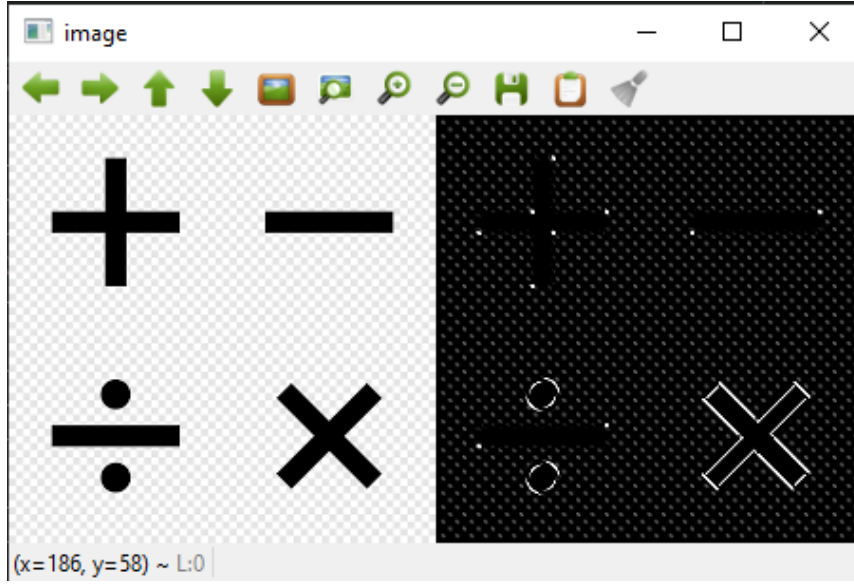
kernel2 = np.array([[ -1, -1, -1],
                    [ -1, 8, -1],
                    [ -1, -1, -1]])

filtre=cv2.filter2D(src=img, ddepth=-1, kernel=diagFiltre)
cv2.imshow('image', np.hstack((img, filtre)))
cv2.waitKey(0);
cv2.destroyAllWindows();

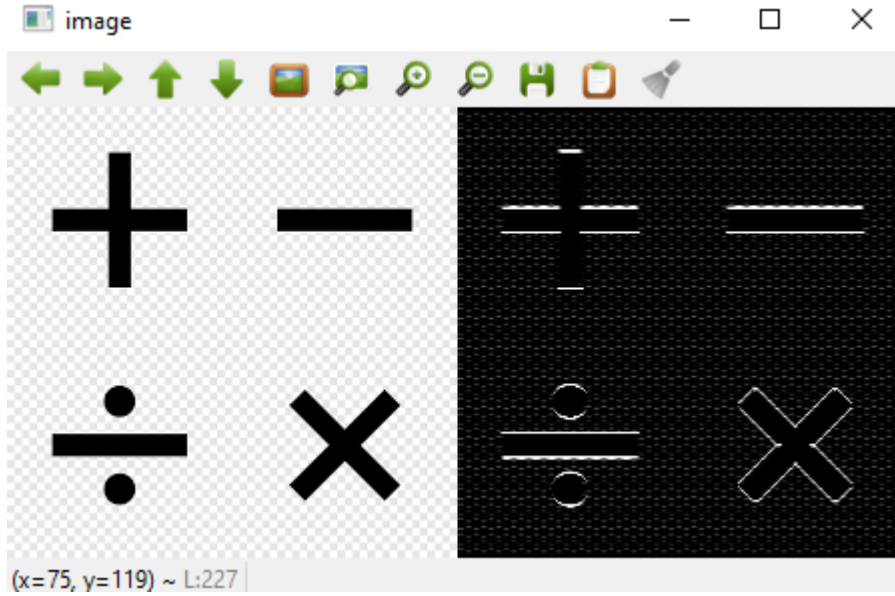
```

Filtre fonksiyonu, filtre:
diagonal

Sonuç:



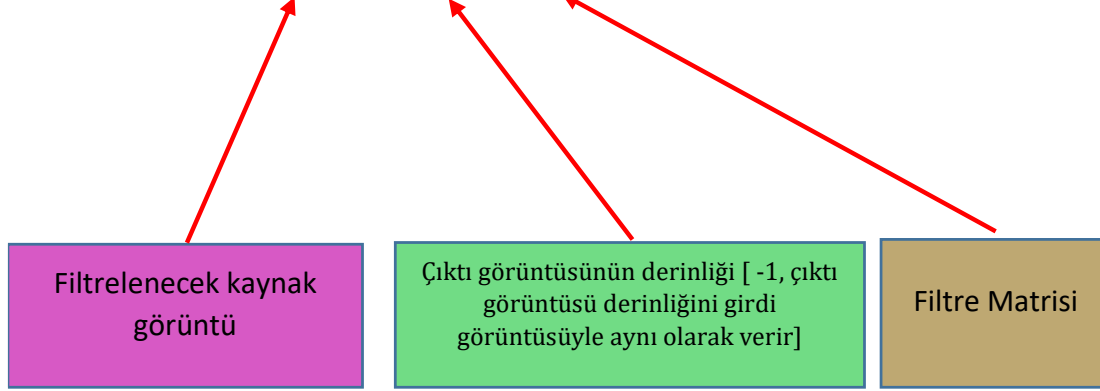
Yukarıda dikkat edilirse diagonal kenarlar yakalandı, yatay ve dikey olanlar absorbe oldu. Aynı fonksiyonu yatay filtre ile çalıştıralım.



Görüldüğü üzere yatay kenarlar yakalanırken dikey kenarlar absorbe oldu. Diagonal kenarlar ise az oranda yakalandı.

Filtreleme İşlemleri:

cv2.filter2D (src, dst, ddepth, kernel)



Filtreleme için başka fonksiyonlar da kullanılmaktadır.

Blur() ve BoxFilter() Filtreleri

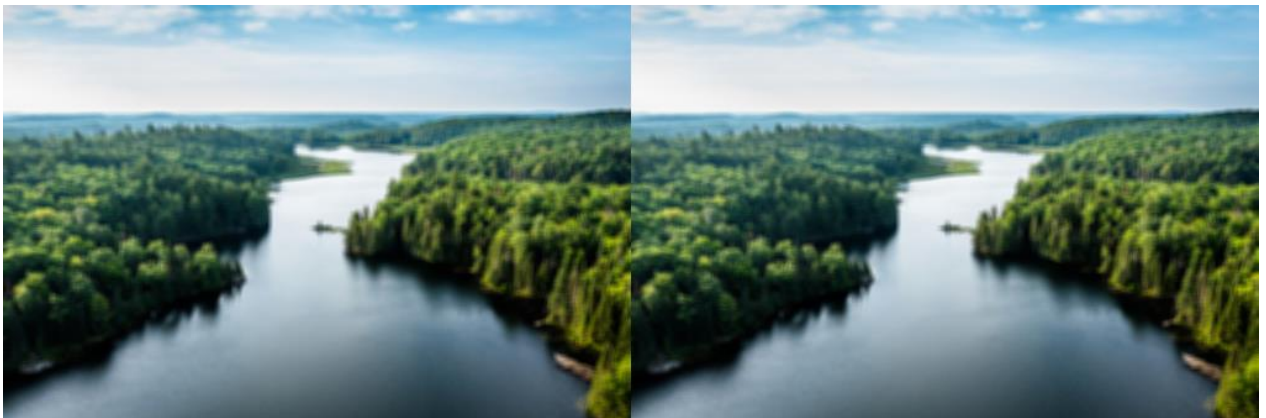
Örnek:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("nehir2.jpg")
im1 = cv2.blur(img,(5,5))
im2 = cv2.boxFilter(img, -1, (5, 5), normalize=True)

# plt.figure(),plt.imshow(img)
# plt.figure(),plt.imshow(im1)

cv2.imshow('image', np.hstack((im1, im2)))
cv2.waitKey(0);
cv2.destroyAllWindows();
```

Filtrenin boyutu
yani kernel size
(ksize)



Blur

BoxFilter

Yukarıda görüldüğü üzere görüntü bulanıklaşmıştır. Bulanıklaştırma çoğu zaman görüntüyü eşiklemek istediğimizde ya da çok keskin görüntüleri yumuşatmak istediğimizde kullanırız.

Blurs an image using the normalized box filter.

The function smooths an image using the kernel:

$$K = \frac{1}{\text{ksize.width} * \text{ksize.height}} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ 1 & 1 & 1 & \dots & 1 & 1 \\ \dots & & & & & \\ 1 & 1 & 1 & \dots & 1 & 1 \end{bmatrix}$$

Filtrenin boyutu
yani kernel size
(ksize)

The call `blur(src, dst, ksize, anchor, borderType)` is equivalent to `boxFilter(src, dst, src.type(), ksize, anchor, true, borderType)`.

Parameters

- src** input image; it can have any number of channels, which are processed independently, but the depth should be CV_8U, CV_16U, CV_16S, CV_32F or CV_64F.
- dst** output image of the same size and type as src.
- ksize** blurring kernel size.
- anchor** anchor point; default value Point(-1,-1) means that the anchor is at the kernel center.
- borderType** border mode used to extrapolate pixels outside of the image, see **BorderTypes**. **BORDER_WRAP** is not supported.

Medyan filtre

cv2.medianBlur(src, dst, ksize)

src - Bu işlem için kaynağı (girdi görüntüsü) temsil eden bir Mat nesnesi.

dst - Bu işlem için hedefi (çıktı görüntüsü) temsil eden bir Mat nesnesi.

ksize - Çekirdeğin boyutunu temsil eden bir Boyut nesnesi.

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import random
orj=cv2.imread("nehir2.jpg")
img=cv2.imread("nehir2.jpg")

#Resime 200 adet gürültü ekliyoruz.
for k in range(0,200):
    i=random.randint(0, img.shape[0]-1)
    j=random.randint(0, img.shape[1]-1)
    img[i,j]=255

filtre=cv2.medianBlur(img,3)
cv2.imshow('image', np.hstack((orj, img, filtre)))
cv2.waitKey(0);
cv2.destroyAllWindows();
```



Orjinal



Gürültülü



Medyan Filtreli

Kenar Bulma Filtreleri:

cv2.Canny(image, T_lower, T_upper, aperture_size, L2Gradient)

Görüntü: Canny filtresinin uygulanacağı girdi görüntüsü

T_lower: Histerezis Eşiklemeye alt eşik değeri

T_upper: Histerezis Eşiklemeye üst eşik değeri

aperture_size: Sobel filtresinin açıklık boyutu.

L2Gradient: Kenar Gradyanı hesaplamasında daha fazla hassasiyet için kullanılan Boolean parametresi.

Örnek:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread("desenler.jpg") # Resim Oku

# Parametre Değerlerini Ayarlama
t_lower = 50 # Lower Threshold
t_upper = 150 # Upper threshold

# Canny kenar filtresini uygulama
edge = cv2.Canny(img, t_lower, t_upper)

cv2.imshow('orjinal', img)
cv2.imshow('kenarlar', edge)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

