



# Görüntü İşleme Filtreleme Teknikleri Matris İşlemleri

10 min read · Mar 23, 2025



Süleyman Meral

Share

More



Herkese Merhaba! Bu yazımızda görüntü işlemede kullanılan filtreleme işlemlerinin arkasındaki matematiksel işlemlerden bahsedeceğiz. Filtreleme işlemlerinin matrislerde nasıl gerçekleştiğine dair işlemleri anlatacağız. Aynı zamanda yapılan işlemleri Python'da göreceğiz.

# FILTERING OPERATIONS



Median Filtering



Midpoint Filtering



Maximum Filtering



Alpha Trimmed Mean Filtering



Minimum Filtering



Gauss Filtering



Mean Filtering

Temel olarak 7 Filtreleme işleminden bahsediyor olacağım. Bunlar görselde gördüğümüz işlemler olacak. Şimdifiltreleme tekniklerinden biraz teorik olarak bahsedelim.

## Görüntü İşlemede Filtreleme Tekniklerinin Arkasında Ne Var?

Görüntü işleme dünyasında,filtreleme teknikleri genellikle gürültü azaltma, detay iyileştirme veya çeşitli görsel efektlerin elde edilmesi amacıyla kullanılır. Ancak, bu tekniklerin arkasındaki matematiksel altyapı, aslında çok daha derin ve ilginçtir. Peki,filtreleme işlemleri arkaplanda nasıl gerçekleşir?

Temelde,filtreleme işlemleri, bir görüntü üzerinde belirli bir matris (genellikle çekirdek veya kernel olarak adlandırılır) ile işlem yapılarak gerçekleştirilir. Bu matris, her bir pikselin çevresindeki belirli bir bölgeyi dikkate alır ve bu bölgeyi işlemden geçirerek çıkış görüntüsünü oluşturur. Bu işlem, görüntü üzerinde yerel değişiklikler yaparak genel gürültüyü azaltmaya, kontrastı artırmaya veya belirli özelliklerini daha belirgin hale getirmeye yardımcı olur.

Matematiksel olarak,filtreleme genellikle konvolüsyon işlemi ile yapılır. Bu işlem, bir görüntü matrisi ile filtre matrisinin (kernel) her bir pikseli için çarpma ve toplama işlemi gerçekleştirir. Filtreleme sırasında, bu çekirdek matrisinin her bir elemanı, görüntüdeki karşılık gelen piksel değerleriyle çarpılır ve sonuçlar toplanarak yeni bir piksel değeri elde edilir.

Özetle, filtreleme tekniklerinin gücü, bu basit ama etkili matris hesaplamalarına dayanır. Bu teknikler sayesinde, bir görüntüdeki gürültüyü ortadan kaldırabilir, keskinlik ve kontrastı artırabilir veya çeşitli görsel efektler uygulayabilirsiniz. Filtreleme, dijital görüntü işleme dünyasında, matematiksel temelleriyle devrim yaratan, güçlü bir araçtır.

Bir görseli matrise çevirdiğimizde, her bir pikselin renk değerleri (genellikle RGB formatında) bir matris olarak temsil edilir. Filtreleme işlemi, bu matris üzerine yapılan matematiksel işlemlerle, görüntüyü işlemek (örneğin, gürültü azaltma veya keskinleştirme) için kullanılır.

İşlemlerimizi gerçekleştirmek için 6x6 'lık bir matrisi kullanacağız.

Süleyman Meral

100	120	130	29	32	90
110	120	130	37	75	58
14	71	21	1	57	41
71	74	52	63	21	91
60	87	1	59	88	59
20	99	87	20	48	79

**Filtreleme işlemlerimizi bu matrise göre yapacağız.**

(We are going to perform our filtering operations based on this matrix.)

İşlemlerimizi bu matrisi baz alarak kullanacağız. Bu matrisi piksel değerlerine çevrilmiş bir resim olarak düşünebiliriz. Değerler tamamen random olarak oluşturulmuştur. Gerçek olarak bir resime karşılık gelmemektedir.

Median Filtreleme İşlemi ile başlayabiliriz.

## Median Filtreleme Nedir?

Median filtreleme, görüntü işleme tekniklerinde gürültü azaltma amacıyla sıkça kullanılan bir yöntemdir. Bu filtre, her pikselin çevresindeki piksellerin değerlerinin medyanını alarak, o pikselin yeni değerini belirler. Özellikle “salt and pepper” (tuz ve biber) gürültüsünü etkili bir şekilde temizler. Bu işlem, piksellerin

aşırı uç değerlerden etkilenmesini engelleyerek, görüntüyü daha temiz ve doğal hale getirir.

Süleyman Meral

Median  
Filtering

## MEDİAN FİLTRELEME NEDİR? (WHAT IS THE MEDIAN FILTERING?)

Gürültü azaltma için kullanılır. Matris maskemizin elemanları büyükten küçüğe doğru sıralanır, ortadaki değer maskemizin ortasındaki değer ile güncellenir. (It is used for noise reduction. The elements of our matrix mask are sorted in descending order, and the middle value is updated with the center value of our mask.)

### FİLTRELEME MASKESİ (KERNEL)

100	120	130
110	120	130
14	71	21



100	120	130
110	110	130
14	71	21

14 - 21 - 71 - 100 - 110 - 120 - 120 - 130 - 130

Filtreleme işlemlerinde genellikle 3x3'lük bir maske kullanılır. Bu maske, her bir hücrenin etrafındaki 3x3'lük bölgeyi alarak, ana matrise sırasıyla uygulanır ve her pikselin yeni değeri hesaplanır. Ancak, köşe hücrelerinde maske tam olarak uygulanamayacak şekilde bazı boş alanlar kalabilir. Bu durumda, boş kalan hücreler için farklı yöntemler kullanılabilir. Yaygın yöntemler arasında “0” ekleme veya en yakın komşuyu referans alarak değer atama bulunur. Biz bu işlemlerde, köşe hücrelerindeki boş alanları “0” ekleyerek doldurmayı tercih edeceğiz.

Maskemizin içindeki değerler küçükten büyüğe doğru sıralanır. Ardından ortanca değer belirlenir. Ortanca değer ana matrisimizde maskemizin ortasında bulunan hücredeki değer ile güncellenir. Sırasıyla tüm matrise bu işlem uygulanır.

100	120	130	29	32	90
110	120	130	37	75	58
14	71	21	1	57	41
71	74	52	63	21	91
60	87	1	59	88	59
20	99	87	20	48	79



0	110	37	32	32	0
74	110	74	37	41	41
71	74	63	52	57	41
60	60	59	52	59	41
60	71	63	52	59	48
0	20	20	20	48	0

Median filtreleme işleminden sonra matrisimiz bu şekilde güncellendi.  
(Our matrix updated like this after median filtering)

Köşe Değerleri belirlemek için boş gelen kısımlar “0” ekleme yöntemi ile güncellenmiştir.  
(To determine the corner values, the empty sections have been updated using the method of adding '0'.)

Teorik olarak işin matematiği bu şekilde gerçekleşmektedir. Aynı zamanda bu işlemi python kullanarak gerçekleştirelim ve sonuçları gözlemleyelim.

```
import numpy as np
from scipy.ndimage import median_filter

# 6x6'lık matris
matrix =np.array([[100, 120, 130, 29, 32, 90],
                  [110, 120, 130, 37, 75, 58],
                  [14, 71, 21, 1, 57, 41],
                  [71, 74, 52, 63, 21, 91],
                  [60, 87, 1, 59, 88, 59],
                  [20, 99, 87, 20, 48, 79]])

# 3x3 median filtre uygulama
median_filtered = median_filter(matrix, size=3, mode='constant', cval=0)

# Sonuçları yazdırma
print("Median Filtering\n",median_filtered)
```

```
[3]: import numpy as np
from scipy.ndimage import median_filter, maximum_filter, minimum_filter

# Matris
matrix = np.array([
    [100, 120, 130, 29, 32, 90],
    [110, 120, 130, 37, 75, 58],
    [14, 74, 21, 1, 57, 41],
    [71, 74, 52, 63, 21, 91],
    [60, 87, 1, 59, 88, 59],
    [20, 99, 87, 20, 48, 79]
])

# 3x3 Maskemiz, köse değerler 0 ile güncellenecek
median_filtered = median_filter(matrix, size=3, mode='constant', cval=0)

print("Median Filtering:\n", median_filtered)
```

Süleyman Meral

## Python Median Filtering

Görüldüğü üzere python ile aynı sonucu aldık.

Şimdi ise diğer bir filtreleme yöntemi olan Maximum filtrelemeye geçelim

### Maximum Filtreleme Nedir ?

Maximum filtreleme, görüntü işleme alanında genellikle nesnelerin kenarlarını belirginleştirmek veya gürültüyü gidermek amacıyla kullanılan bir tekniktir. Bu yöntemde, her pikselin etrafındaki belirli bir pencere (genellikle 3x3 boyutunda) alınarak, bu pencere içindeki en yüksek piksel değeri seçilir ve o pikselin yerine yerleştirilir. Bu işlem, özellikle gürültülü görüntülerde, küçük nesneleri ve ince detayları daha belirgin hale getirirken, düşük değere sahip gürültüyü ortadan kaldırır. Maximum filtreleme, özellikle “salt and pepper” gürültüsünü temizlemek için etkilidir. Piksel değeri küçük karanlığa yakın değerleri temizler.

## MAXIMUM FILTRELEME NEDİR? (WHAT IS THE MAXIMUM FILTERING?)

Genelde koyu pikselleri azaltmak için kullanılır. Matris maskemizin elemanları büyükten küçüğe doğru sıralanır, en büyük değer maskemizin ortasındaki değer ile güncellenir. (It is used for noise reduction. The elements of our matrix mask are sorted in descending order, and the maximum value is updated with the center value of our mask.)

### FILTRELEME MASKESİ (KERNEL)

100	120	130
110	120	130
14	71	21



100	120	130
110	130	130
14	71	21

14 - 21 - 71 - 100 - 110 - 120 - 120 - 130 - 130

Tüm filtreleme yöntemlerinde olduğu gibi maskemizdeki değerleri küçükten büyüğe doğru sıralıyoruz. Daha sonra en büyük değeri seçiyoruz ve ana matrisimizde maskemizin ortasına gelen değer ile güncelliyoruz.

100	120	130	29	32	90
110	120	130	37	75	58
14	71	21	1	57	41
71	74	52	63	21	91
60	87	1	59	88	59
20	99	87	20	48	79



120	130	130	130	90	90
120	130	130	130	90	90
120	130	130	130	91	91
87	87	87	88	91	91
99	99	99	88	91	91
99	99	99	88	88	88

Maximum filtreleme işleminden sonra matrisimiz bu şekilde güncellendi.  
(Our matrix updated like this after maximum filtering)

Köşe Değerleri belirlemek için boş gelen kısımlar “0” ekleme yöntemi ile güncellenmiştir.  
(To determine the corner values, the empty sections have been updated using the method of adding '0'.)

Python kodumuz aşağıdaki gibidir.

```

import numpy as np
from scipy.ndimage import median_filter,maximum_filter

# 6x6'luk matris
matrix =np.array([[100, 120, 130, 29, 32, 90],
                  [110, 120, 130, 37, 75, 58],
                  [14, 74, 21, 1, 57, 41],
                  [71, 74, 52, 63, 21, 91],
                  [60, 87, 1, 59, 88, 59],
                  [20, 99, 87, 20, 48, 79]])

# 3x3 maximum filtre uygulama
maximum_filtered = maximum_filter(matrix, size=3, mode='constant', cval=0)

# Sonuçları yazdırma
print("Maximum Filtering\n",maximum_filtered)

```

```

[3]: import numpy as np
      from scipy.ndimage import median_filter, maximum_filter, minimum_filter

      # Matris
      matrix = np.array([
          [100, 120, 130, 29, 32, 90],
          [110, 120, 130, 37, 75, 58],
          [14, 74, 21, 1, 57, 41],
          [71, 74, 52, 63, 21, 91],
          [60, 87, 1, 59, 88, 59],
          [20, 99, 87, 20, 48, 79]
      ])

      # 3x3 Maskemiz, köşe değerler 0 ile güncellenecek
      maximum_filtered = maximum_filter(matrix, size=3, mode='constant', cval=0)

      print("Maximum Filtering:\n", maximum_filtered)

```

```

Maximum Filtering:
[[120 130 130 130 90 90]
 [120 130 130 130 90 90]
 [120 130 130 130 91 91]
 [ 87  87  87  88  91  91]
 [ 99  99  99  88  91  91]
 [ 99  99  99  88  88  88]]

```

## Python Maximum Filtering

Şimdi maximum filtrelemeye çok benzer olan minimum filtrelemeye geçelim.

### Minimum Filtreleme Nedir ?

Minimum filtreleme, görüntü işleme tekniklerinden biri olup genellikle görüntülerdeki parlak noktalardan (örneğin, "salt" gürültüsünden) arındırma amacıyla kullanılır. Bu yöntemde, her pikselin etrafındaki belirli bir pencere (genellikle 3x3 boyutunda) alınarak, bu pencere içindeki en düşük piksel değeri

seçilir ve o pikselin yerine yerleştirilir. Minimum filtreleme, özellikle “salt and pepper” gürültüsünün temizlenmesinde etkilidir çünkü parlak noktalar (yüksek değerli pikseller) yerine, çevresindeki düşük değerli pikselleri tercih eder. Bu sayede, görüntüdeki aşırı parlak noktalar ortadan kaldırılarak daha düzgün bir sonuç elde edilir.

Süleyman Meral

Minimum Filtering

## MINIMUM FILTRELEME NEDİR? (WHAT IS THE MINIMUM FILTERING?)

Genelde parlak pikselleri azaltmak için kullanılır. Matris maskemizin elemanları büyükten küçüğe doğru sıralanır, en küçük değer maskemizin ortasındaki değer ile güncellenir. (It is used for noise reduction. The elements of our matrix mask are sorted in descending order, and the minimum value is updated with the center value of our mask.)

### FİLTRELEME MASKESİ (KERNEL)

100	120	130
110	120	130
14	71	21



100	120	130
110	14	130
14	71	21

14

- 21 - 71 - 100 - 110 - 120 - 120 - 130 - 130

Maskemizdeki değerleri küçükten büyüğe sıralayıp, en küçük değer ile ana matrisimizde maskenin ortasına gelen değeri güncelliyoruz.

Süleyman Meral

100	120	130	29	32	90
110	120	130	37	75	58
14	71	21	1	57	41
71	74	52	63	21	91
60	87	1	59	88	59
20	99	87	20	48	79



0	0	0	0	0	0
0	14	1	1	1	0
0	14	1	1	1	0
0	1	1	1	1	0
0	1	1	1	20	0
0	0	0	0	0	0

Minimum filtreleme işleminden sonra matrisimiz bu şekilde güncellendi.  
(Our matrix updated like this after minimum filtering)

Köşe Değerleri belirlemek için boş gelen kısımlar “0” ekleme yöntemi ile güncellenmiştir.  
(To determine the corner values, the empty sections have been updated using the method of adding '0'.)

Minimum Filtreleme Python kodumuz aşağıdaki gibidir.

```
import numpy as np
from scipy.ndimage import median_filter,maximum_filter,minimum_filter

# 6x6'lık matris
matrix =np.array([[100, 120, 130, 29, 32, 90],
                  [110, 120, 130, 37, 75, 58],
                  [14, 74, 21, 1, 57, 41],
                  [71, 74, 52, 63, 21, 91],
                  [60, 87, 1, 59, 88, 59],
                  [20, 99, 87, 20, 48, 79]])

# 3x3 minimum filtre uygulama
minimum_filtered = minimum_filter(matrix, size=3, mode='constant', cval=0)

# Sonuçları yazdırma
print("Minimum Filtering\n",minimum_filtered)
```

```
[5]: import numpy as np
      from scipy.ndimage import median_filter, maximum_filter, minimum_filter

      # Matriş
      matrix = np.array([
          [100, 120, 130, 29, 32, 90],
          [110, 120, 130, 37, 75, 58],
          [14, 74, 21, 1, 57, 41],
          [71, 74, 52, 63, 21, 91],
          [60, 87, 1, 59, 88, 59],
          [20, 99, 87, 20, 48, 79]
      ])

      # 3x3 Maskemiz, köşe değerler 0 ile güncellenecek
      minimum_filtered = minimum_filter(matrix, size=3, mode='constant', cval=0)

      print("Minimum Filtering:\n", minimum_filtered)
```

```
Minimum Filtering:
[[ 0  0  0  0  0  0]
 [ 0 14  1  1  1  0]
 [ 0 14  1  1  1  0]
 [ 0  1  1  1  1  0]
 [ 0  1  1  1  20  0]
 [ 0  0  0  0  0  0]]
```

## Python Minimum Filtering

Şimdi diğer bir yöntemimiz olan Midpoint Filtrelemeye geçelim.

### Midpoint Filtreleme Nedir ?

Midpoint filtrelere, görüntü işleme alanında, özellikle gürültü azaltma ve görüntüdeki ayrıntıları koruma amacıyla kullanılan bir tekniktir. Bu yöntemde, her

pikselin etrafındaki belirli bir pencere (genellikle 3x3 boyutunda) alınır ve bu pencere içindeki en yüksek ve en düşük piksel değerleri arasındaki ortalama (midpoint) değer hesaplanır. Bu ortalama, o pikselle değiştirilir.

Midpoint filtreleme, özellikle “salt and pepper” (tuz ve biber) gürültüsünü temizlemek için etkilidir ve aynı zamanda görüntüdeki kenarları yumusatmadan gürültüyü azaltmaya yardımcı olur. Bu filtreleme türü, görüntüdeki aşırı uç değerleri ortadan kaldırarak, daha düzgün bir görüntü sağlar.

Süleyman Meral

Midpoint Filtering

## MİDPOINT FILTRELEME NEDİR? (WHAT IS THE MIDPOINT FILTERING?)

Matris maskemizin elemanları büyükten küçüğe doğru sıralanır, en büyük değer ve en küçük değer toplanıp 2'ye bölünür, maskemizin ortasındaki değer ile güncellenir. (It is used for noise reduction. The elements of our matrix mask are sorted in descending order, and the maximum value and minimum value added each other. New value is updated with the center value of our mask.)

### FILTRELEME MASKESİ (KERNEL)

100	120	130
110	120	130
14	71	21



100	120	130
110	72	130
14	71	21

**14 - 21 - 71 - 100 - 110 - 120 - 120 - 130 - 130**

$$(130 + 14) / 2 = 72$$

Sıralanmış değerlerimizden en büyük ve en küçük değerler seçilir ve 2'ye bölünür. Bu değer ana matrisimizde olan maskemizin ortasına karşılık gelen değer ile güncellenir. Bir anlamda maximum filtreleme ve minimum filtreleme işlemlerini kombine etmiş oluyoruz.

100	120	130	29	32	90
110	120	130	37	75	58
14	71	21	1	57	41
71	74	52	63	21	91
60	87	1	59	88	59
20	99	87	20	48	79



60	65	65	65	45	45
60	72	65,5	65,5	45,5	45
60	72	65,5	65,5	46	45,5
43,5	44	44	44,5	46	45,5
49,5	50	50	44,5	55,5	45,5
49,5	49,5	49,5	44	44	44

Midpoint filtreleme işleminden sonra matrisimiz bu şekilde güncellendi.  
(Our matrix updated like this after midpoint filtering)

Köşe Değerleri belirlemek için boş gelen kısımlar “0” ekleme yöntemi ile güncellenmiştir.  
(To determine the corner values, the empty sections have been updated using the method of adding '0'.)

Midpoint Filtreleme Python kodumuz aşağıdaki gibidir.

```
import numpy as np
from scipy.ndimage import median_filter,maximum_filter,minimum_filter

# 6x6'lık matris
matrix =np.array([[100, 120, 130, 29, 32, 90],
                  [110, 120, 130, 37, 75, 58],
                  [14, 71, 21, 1, 57, 41],
                  [71, 74, 52, 63, 21, 91],
                  [60, 87, 1, 59, 88, 59],
                  [20, 99, 87, 20, 48, 79]])

# 3x3 midpoint filtre uygulama
midpoint_filtered = (minimum_filter(matrix, size=3, mode='constant', cval=0) +
                      # Sonuçları yazdırma
print("Midpoint Filtering\n",midpoint_filtered)
```

```
[9]: import numpy as np
from scipy.ndimage import median_filter, maximum_filter, minimum_filter

# Matris
matrix = np.array([
    [100, 120, 130, 29, 32, 90],
    [110, 120, 130, 37, 75, 58],
    [14, 74, 21, 1, 57, 41],
    [71, 74, 52, 63, 21, 91],
    [60, 87, 1, 59, 88, 59],
    [20, 99, 87, 20, 48, 79]
])

# 3x3 Maskemiz, köşe değerler 0 ile güncellenecek
minimum_filtered = (minimum_filter(matrix, size=3, mode='constant', cval=0)+maximum_filter(matrix, size=3, mode='constant', cval=0))/2

print("Minimum Filtering:\n", minimum_filtered)
```

Minimum Filtering:  
[[60. 65. 65. 45. 45. 45.]  
[60. 72. 65.5 65.5 45.5 45. ]  
[60. 72. 65.5 65.5 46. 45.5]  
[43.5 44. 44. 44.5 46. 45.5]  
[49.5 50. 50. 44.5 55.5 45.5]  
[49.5 49.5 49.5 44. 44. 44. ]]]

## Python Midpoint Filtering

Şimdi ise diğer bir yöntemimiz olan Alfa kesilmiş ortalama Filtrelemeye geçelim.

### Alfa Kesilmiş Ortalama Filtreleme Nedir?

Alfa kesilmiş ortalama (alpha-trimmed mean), görüntü işleme ve sinyal işleme gibi alanlarda, özellikle gürültü azaltma amacıyla kullanılan birfiltreleme yöntemidir. Bu teknik, belirli bir pencere (örneğin, 3x3) içinde yer alan piksel değerlerinden, en yüksek ve en düşük alfa kadar olan değerlerin kesilmesi (yani çıkarılması) ve kalan değerlerin ortalamasının alınması işlemiyle çalışır.

Bu yöntem, sıradan ortalamafiltrelemesinin aksine, üç değerlerin (gürültü) etkisini azaltır. Böylece, gürültülü piksellerin görüntüye etkisi minimize edilir ve daha temiz bir sonuç elde edilir.

### Alfa Kesilmiş Ortalama İşlemi:

1. İlk olarak, pencere içindeki piksel değerleri sıralanır.
2. Ardından, pencere boyutuna göre belirlenen alfa sayıda en küçük ve en büyük değer çıkarılır.
3. Kalan piksellerin ortalaması alınarak yeni piksel değeri hesaplanır.

Örneğin, 3x3'lük bir pencerede 9 piksel olduğunda ve alfa=2 seçildiğinde, en küçük 2 ve en büyük 2 piksel değerleri çıkarılır, ve geriye kalan 5 değerin ortalaması alınarak yeni piksel değeri elde edilir.

Alfa kesilmiş ortalama, özellikle “salt and pepper” gürültüsünü etkili bir şekilde temizler ve görüntüdeki kenarları yumuşatmadan gürültü azaltma sağlar.

Süleyman Meral

ATM Filtering

## ALFA KESİLMİŞ ORTALAMA FILTRELEME NEDİR? (ALPHA TRIMMED MEAN FILTERING)

Matris maskemizin elemanları büyükten küçüğe doğru sıralanır. Bir alpha değeri belirlenir. Bu alpha değeri maske matrisin boyutu ile çarpılır. Çıkan sonuç en yakın çift sayıya yuvarlanır. Bu sayı kadar en büyük ve en küçük değer silinir. Kalan sayıların ortalaması alınır. Yeni değer maskemizin ortanca değeri ile güncellenir. Alpha değerini 0.2 olarak belirleyelim.  $9 \times 0.2 = 1.8$  Yuvarladığımızda 2 yapar. Başta ve sonda olan 1'ler değer silinir.

### FILTRELEME MASKESİ (KERNEL)

100	120	130
110	120	130
14	71	21

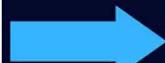


100	120	130
110	96	130
14	71	21

$$14 - 21 - 71 - 100 - 110 - 120 - 120 - 130 - 130 \\ (21+71+100+110+120+120+130)/7=96$$

Süleyman Meral

100	120	130	29	32	90
110	120	130	37	75	58
14	71	21	1	57	41
71	74	52	63	21	91
60	87	1	59	88	59
20	99	87	20	48	79



47	82	62	43	33	23
59	96	75	54	47	34
49	74	63	46	50	36
41	52	49	39	55	38
44	64	63	50	59	42
23	36	36	30	37	26

Alpha Trimmed Mean filtreleme işleminden sonra matrisimiz bu şekilde güncellendi.  
(Our matrix updated like this after Alpha Trimmed Mean filtering)

Köşe Değerleri belirlemek için boş gelen kısımlar “0” ekleme yöntemi ile güncellenmiştir.  
(To determine the corner values, the empty sections have been updated using the method of adding '0'.)

Python kodumuz aşağıdaki gibidir.

```
import numpy as np  
from scipy.ndimage import generic_filter
```

```

from scipy import stats

def alpha_trimmed_mean_filter(values, proportion=0.2):
    return stats.trim_mean(values, proportion)

# Matris
matrix = np.array([
    [100, 120, 130, 29, 32, 90],
    [110, 120, 130, 37, 75, 58],
    [14, 74, 21, 1, 57, 41],
    [71, 74, 52, 63, 21, 91],
    [60, 87, 1, 59, 88, 59],
    [20, 99, 87, 20, 48, 79]

])

# Alpha-trimmed mean filtering uygula (3x3 pencere, %20 kesme oranı)
filtered_matrix = generic_filter(matrix, alpha_trimmed_mean_filter, size=3, mode='constant', cval=0)

print("Alpha-Trimmed Mean Filtering:\n", filtered_matrix)

```

```

[29]: import numpy as np
from scipy.ndimage import generic_filter
from scipy import stats

def alpha_trimmed_mean_filter(values, proportion=0.2):
    return stats.trim_mean(values, proportion)

# Matris
matrix = np.array([
    [100, 120, 130, 29, 32, 90],
    [110, 120, 130, 37, 75, 58],
    [14, 74, 21, 1, 57, 41],
    [71, 74, 52, 63, 21, 91],
    [60, 87, 1, 59, 88, 59],
    [20, 99, 87, 20, 48, 79]
])

filtered_matrix = generic_filter(matrix, alpha_trimmed_mean_filter, size=3, mode='constant', cval=0)

print("Alpha-Trimmed Mean Filtering:\n", filtered_matrix)

```

Alpha-Trimmed Mean Filtering:  
[[47 82 62 43 33 23]  
[59 96 75 54 47 37]  
[49 74 63 46 50 36]  
[41 52 49 39 55 38]  
[44 64 63 50 59 42]  
[23 36 36 30 37 26]]

## Python Alpha Trimmed Mean Filtering

Şimdi diğer bir yöntemimiz olan Gauss Filtrelemeye geçelim.

### Gauss Filtreleme Nedir?

Gauss filtreleme, görüntü işleme ve sinyal işleme alanlarında gürültüyü azaltmak, pürüzsüzleştirme (blurring) yapmak ve kenarları yumusatmak için kullanılan bir yöntemdir.

Bu filtre, Gauss fonksiyonuna dayalı bir çekirdek (kernel) kullanarak görüntüdeki her pikselin çevresindekilerle ağırlıklı ortalamasını alır.

Süleyman Meral

Gauss Filtering

## GAUSS FILTRELEME NEDİR? (WHAT IS THE GAUSS FILTERING?)

Gauss Filtreleme yönteminde Gauss formülü ile bir kernel oluşturulur. Daha sonra bu kernel ana matrisimizdeki maske ile çarpılır. Çarpılan değerler toplanır. Bu değer maskemizin ortanca değeri ile güncellenir. Sırasıyla tüm matrise bu işlem uygulanır. Kernelimizin boyutunu 3x3 seçelim ve formülde kullanacağımız sigma değerini 1 olarak seçelim.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

**X=[-1,0,1]**  
**Y=[-1,0,1]**

**KERNEL**

G(-1,-1)	G(-1,0)	G(-1,1)
G(0,-1)	G(0,0)	G(0,1)
G(1,-1)	G(1,0)	G(1,1)

**KERNEL**

0,0585	0,0965	0,0585
0,0965	0,1592	0,0965
0,0585	0,0965	0,0585



The diagram illustrates the mapping of a 3x3 kernel from raw values to normalized Gaussian values. On the left, a 3x3 grid of labels G(x,y) is shown. A large blue arrow points to the right, where the same grid is filled with numerical values representing the Gaussian function  $G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$  for  $\sigma=1$ .

3x3'lük seçtiğimiz için x ve y değerlerimiz -1,0,1 olarak oluştı. Eğer 5x5 gibi bir maske seçseydik -2,-1,0,1,2 olarak oluşacaktı. Bu değerler kernelimize görselde olduğu gibi yerleştirilip gauss fonksiyonu uygulanıyor. Formülümüzde bir adet de sigma değeri bulunuyor. İşlemimiz için bu değeri 1 seçik.  $\sigma$  (Standart Sapma): Filtrenin bulanıklık miktarını belirler.

Formüle uygulandıktan değerler görselde görüldüğü gibi oluşuyor.

$$0,0585+0,0965+0,0585+0,0965+0,1592+0,0965+0,0585+0,0965+0,0585=0,779$$

KERNEL

0,0585	0,0965	0,0585
0,0965	0,1592	0,0965
0,0585	0,0965	0,0585



$$\times \ 1/0,779$$

KERNEL

0,0751	0,1238	0,0751
0,1238	0,2042	0,1238
0,0751	0,1238	0,0751

Matrisin ortalamasını alıp yeni kernelimizi elde ediyoruz. Bu kernel ile işlem yapacağız.  
 (We average the matrix and obtain our new kernel. We will process with this kernel)

Daha sonra bu değerlerin hepsi toplanıp, bu toplam matrisimize bölünüyor. İşlem yapacağımız kernel şekilde görüldüğü gibi oluşuyor.

Bu kerneli sırasıyla ana matrisimizdeki hücrelere uygulayacağız .

120	130	29
120	130	37
74	21	21



0,0751	0,1238	0,0751
0,1238	0,2042	0,1238
0,0751	0,1238	0,0751

$$120 \cdot 0,0751 + 130 \cdot 0,1238 + 29 \cdot 0,0751 + 120 \cdot 0,1238 + 130 \cdot 0,2042 + 37 \cdot 0,1238 + 74 \cdot 0,0751 + 21 \cdot 0,1238 + 21 \cdot 0,0751 = 81,51$$



120	130	29
120	81,51	37
74	21	21

Ana matrisimizde bulunan maske ile kernelimiz çarpılıyor. Bu değerlerin hepsi toplanıyor. Ve şekilde gördüğümüz 130 değeri 81.51 değeri ile güncelleniyor. Bu değerler yuvarlanıp uygulanabilir.

100	120	130	29	32	90
110	120	130	37	75	58
14	71	21	1	57	41
71	74	52	63	21	91
60	87	1	59	88	59
20	99	87	20	48	79



57	85	72	45	37	35
66	98	81	54	50	44
49	70	58	43	47	41
44	57	46	41	53	44
47	64	54	48	60	49
30	48	43	34	41	35

Gauss filtreleme işleminden sonra matrisimiz bu şekilde güncellendi.  
(Our matrix updated like this after Gauss filtering)

Köşe Değerleri belirlemek için boş gelen kısımlar “0” ekleme yöntemi ile güncellenmiştir.  
(To determine the corner values, the empty sections have been updated using the method of adding '0'.)

Python kodumuz aşağıdaki gibidir.

```
import numpy as np
import scipy.ndimage
import matplotlib.pyplot as plt
def gaussian(x, y, sigma=1.0):
    return (1 / (2 * np.pi * sigma**2)) * np.exp(-(x**2 + y**2) / (2 * sigma**2))

sigma = 1.0
kernel_size = 3
half_size = kernel_size // 2
gaussian_kernel = np.zeros((kernel_size, kernel_size))
for i in range(-half_size, half_size + 1):
    for j in range(-half_size, half_size + 1):
        gaussian_kernel[i + half_size, j + half_size] = gaussian(i, j, sigma)

gaussian_kernel /= np.sum(gaussian_kernel)
matrix = np.array([
    [100, 120, 130, 29, 32, 90],
    [110, 120, 130, 37, 75, 58],
    [14, 71, 21, 1, 57, 41],
    [71, 74, 52, 63, 21, 91],
    [60, 87, 1, 59, 88, 59],
    [20, 99, 87, 20, 48, 79]
])

filtered_matrix = scipy.ndimage.convolve(matrix, gaussian_kernel, mode='constant')
```

```
print("3x3 Gaussian Kernel:\n", gaussian_kernel)
print("\nGaussian Filtre Uygulandıktan Sonra:\n", filtered_matrix)
```

Süleyman Meral

```
[13]: import numpy as np
import scipy.ndimage
import matplotlib.pyplot as plt
# Gaussian Fonksiyonu
def gaussian(x, y, sigma=1.0):
    return 1 / (2 * np.pi * sigma**2) * np.exp(-(x**2 + y**2) / (2 * sigma**2))
x = 3
y = 3 # 3x3'den kerneli oluştur
sigma = 1.0
kernel_size = 3
half_size = kernel_size // 2
gaussian_kernel = np.zeros((kernel_size, kernel_size))
for i in range(-half_size, half_size + 1):
    for j in range(-half_size, half_size + 1):
        gaussian_kernel[i + half_size, j + half_size] = gaussian(i, j, sigma)
gaussian_kernel /= np.sum(gaussian_kernel)

matrix = np.array([
    [100, 110, 130, 20, 32, 90],
    [110, 120, 130, 37, 75, 50],
    [14, 76, 21, 1, 57, 41],
    [17, 73, 54, 34, 59, 37],
    [66, 67, 34, 95, 68, 59],
    [20, 99, 87, 20, 48, 79]
])
# Gaussian Filtresini uygula
filtered_matrix = scipy.ndimage.convolve(matrix, gaussian_kernel, mode='constant', cval=0)
print("3x3 Gaussian Kernel:\n", gaussian_kernel)
print("Orijinal Matris:\n", matrix)
print("\nGaussian Filtre Uygulandıktan Sonra:\n", filtered_matrix)

3x3 Gaussian Kernel:
[[0.07511361 0.1238414 0.07511361]
 [0.1238414 0.20417996 0.1238414]
 [0.07511361 0.1238414 0.07511361]

Orijinal Matris:
[[100 110 130 20 32 90]
 [110 120 130 37 75 50]
 [14 76 21 1 57 41]
 [17 73 54 34 59 37]
 [66 67 34 95 68 59]
 [20 99 87 20 48 79]]

Gaussian Filtre Uygulandıktan Sonra:
[[57 85 73 45 37 35]
 [66 98 81 54 58 44]
 [49 70 58 43 47 41]
 [54 64 54 41 41 41]
 [47 64 54 41 48 49]
 [30 48 43 34 41 35]]
```

Python Gauss Filtering

Şimdi son yöntemimiz olan mean filtreleme işlemine geçelim.

## Mean Filtreleme Nedir?

Mean (Ortalama) filtreleme, görüntü veya sinyal işleme alanında gürültüyü azaltmak, bulanıklaştırmak (blurring) ve pikselleri yumusatmak için kullanılan temel bir filtreleme yöntemidir.

Bu filtre, belirli bir pencere boyutundaki tüm piksellerin aritmetik ortalamasını alarak merkezi pikseli günceller.

## ORTALAMA FILTRELEME NEDİR? ( MEAN FILTERING)

Matris maskemizin elemanlarının ortalaması alınır ve bu değer ile maskenin ortanca değeri güncellenir. (The average of the elements in our matrix mask is taken, and this value is used to update the median of the mask.)

### FILTRELEME MASKESİ (KERNEL)

100	120	130
110	120	130
14	71	21



100	120	130
110	73.55	130
14	71	21

**14 - 21 - 71 - 100 - 110 - 120 - 120 - 130 - 130**

$$(14+21+71+100+110+120+120+130+130)/9=73.55$$

Maskemizdeki tüm değerler toplanıp ortalaması alınır. Bu değer ana matrisimizdeki maskedeki ortanca değer ile güncelleniyor.

100	120	130	29	32	90
110	120	130	37	75	58
14	71	21	1	57	41
71	74	52	63	21	91
60	87	1	59	88	59
20	99	87	20	48	79



108.88	117.77	93.88	69.33	52.44	68.33
84.66	91.00	73.55	56.88	46.66	60.22
73.11	74.00	63.55	50.77	49.33	59.22
58.33	50.44	48.00	40.33	53.33	60.88
62.44	61.22	60.22	48.77	58.66	68.33
53.88	32.22	62.11	50.88	55.55	68.66

Mean filtreleme işleminden sonra matrisimiz bu şekilde güncellendi.  
(Our matrix updated like this after Mean filtering)

Köşe Değerleri belirlemek için boş gelen kısımlar “0” ekleme yöntemi ile güncellenmiştir.  
(To determine the corner values, the empty sections have been updated using the method of adding '0'.)

Python kodumuz aşağıdaki gibidir.

```
import numpy as np
import scipy.ndimage
```

```

# Orijinal Matris
matrix = np.array([
    [100, 120, 130, 29, 32, 90],
    [110, 120, 130, 37, 75, 58],
    [14, 74, 21, 1, 57, 41],
    [71, 74, 52, 63, 21, 91],
    [60, 87, 1, 59, 88, 59],
    [20, 99, 87, 20, 48, 79]
], dtype=float)

# 3x3 Ortalama Filtresi
filtered_matrix = scipy.ndimage.uniform_filter(matrix, size=3, mode='constant', cval=0)

filtered_matrix_rounded = np.around(filtered_matrix, decimals=2)

print("Mean Filtered Matrix (Rounded to 2 Decimal Places):\n", filtered_matrix)

```

```

[19]: import numpy as np
import scipy.ndimage

# Orijinal Matris
matrix = np.array([
    [100, 120, 130, 29, 32, 90],
    [110, 120, 130, 37, 75, 58],
    [14, 74, 21, 1, 57, 41],
    [71, 74, 52, 63, 21, 91],
    [60, 87, 1, 59, 88, 59],
    [20, 99, 87, 20, 48, 79]
], dtype=float)

filtered_matrix = scipy.ndimage.uniform_filter(matrix, size=3, mode='constant', cval=0)

print("Mean Filtered Matrix (Without Rounding):\n", filtered_matrix)

```

Mean Filtered Matrix (Without Rounding):

[50.	78.88888889	62.88888889	48.11111111	35.66666667	28.33333333]
[59.77777778	91.	73.55555556	56.88888889	46.66666667	39.22222222]
[51.44444444	74.	63.55555556	50.77777778	49.33333333	38.11111111]
[42.22222222	50.44444444	48.	40.33333333	53.33333333	39.66666667]
[45.66666667	61.22222222	60.22222222	48.77777778	58.66666667	42.88888889]
[29.55555556	39.33333333	39.22222222	33.66666667	39.22222222	30.44444444]]

## Python Mean Filtering

Filtreleme işlemlerimizin arkasındaki matematiksel süreci anlatmaya çalıştım. Diğer bir yazımmda bu işlemleri gerçek bir resim üzerinden gösteremiyi düşünüyorum. Okuduğunuz için teşekkür ederim. Ayrıca bu işlemleri youtube kanalımda da anlatacağım linkten kanalıma ulaşabilirsiniz. Herkese kolay gelsin.

<https://www.youtube.com/@suleymanmeral53/featured>

[Edit profile](#)

## Written by Süleyman Meral

6 followers · 6 following

Backend Developer(.NET Core)

---

No responses yet



...



Süleyman Meral

What are your thoughts?

More from Süleyman Meral