# BAB 07

# BERBAGAI OPERASI DASAR PENGOLAHAN CI1RA

Pokok bahasan pada bab ini mencakup hal-hal berikut:

- pengaburan citra;
- penapisan citra;
- pengurangan derau.

# 7.1 Pengaburan Citra

Pengaburan citra kadang diperlukan dalam pengolahan citra. Sebagai contoh, pengaburan dimaksudkan agar ketajaman citra berkurang sehingga hal-hal yang terlalu detail akan mengganggu proses segmentasi citra. Namun, pengaburan citra juga bisa menghasilkan efek berupa pemulusan tekstur pada citra atau bahkan mengurangi derau pada citra.

Terdapat berbagai cara yang bisa dilakukan untuk melakukan pengaburan citra. Beberapa cara dibahas di sini.

## 7.1.1 Pengaburan dengan cvBlur()

Metode cv2. cvBlur () berguna untuk melakukan pengaburan citra dengan cara melakukan pererataan terhadap nilai-nilai di sekitar piksel yang nilainya akan diganti. Area yang digunakan untuk pererataan berupa jendela berukuran n x n dengan n berupa bilangan ganjil.

Filter pererataan dilakukan dengan menggunakan rumus:

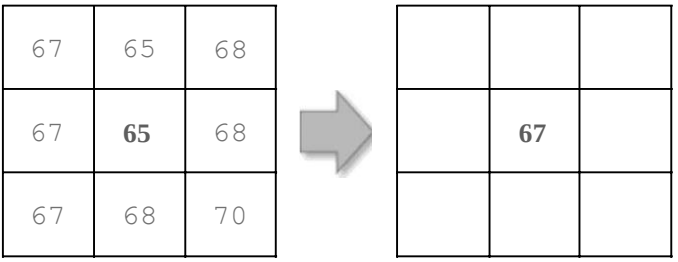
$$g(x,y) = \frac{1}{9} \sum_{p=-1}^{1} \sum_{q=-1}^{1} f(x+p, y+q)$$

Sebagai contoh, piksel pada f(x, y) dan delapan tetangganya memiliki nilai-nilai kecerahan seperti terlihat pada Gambar 7.1. Pada contoh ini, nilai 65 merupakan nilai pada f(x, y). Nilai rerata pengganti untuk g(x, y) dihitung dengan cara seperti berikut:

g(x,y) = 1/9 x (67+65+68+67+65+68+67+68+70) = 67,22

::67

Jadi, nilai 68 pada f(x, y) diubah menjadi 67 pada g(x, y).




***Gambar 7.1 Pererataan dihitung menurutjendela berukuran s x s***

Secara umum, kernel berukuran n x n yang digunakan untuk melakukan operasi pererataan adalah seperti berikut:

$$K = \frac{1}{n.n}\begin{vmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{vmatrix}$$

Kernel inilah yang digunakan pada cvBlur (). Adapun cara untuk menggunakan cvBlur () adalah seperti berikut:

cv2.cvBlur(citra, *(n,* n))

Argumen pertama berupa citra sumber dan argumen kedua berupa tupel dengan dua elemen, dengan *n* menyatakan ukuran jendela untuk pererataan.

erikut menunjukkan contoh penggunaan cvBlur ():

**Berkas** : **blur.py**

```
# Pengaburan citra dengan cvBlur

import cv2
import numpy as np
```

```
citra   cv2.imread('simba.png')

blurl   cv2.blur(citra, (5, 5))
blur2   cv2.blur(citra, (13, 13))

hasil   np.hstack((citra, blurl, blur2))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)
```

**Akhir berkas**

Skrip ini menggunakan dua kernel. Pada blur1,kernel yang digunakan berukuran 5 x 5. Pada blur2, kernel yang digunakan berukuran 9 x 9. Hasil pengenaan kedua kernel ini pada citra s imba. png ditunjukkan pada Gambar 7.2. Tampak bahwa semakin besar ukuran kernel yang digunakan, gambar semakin kabur.

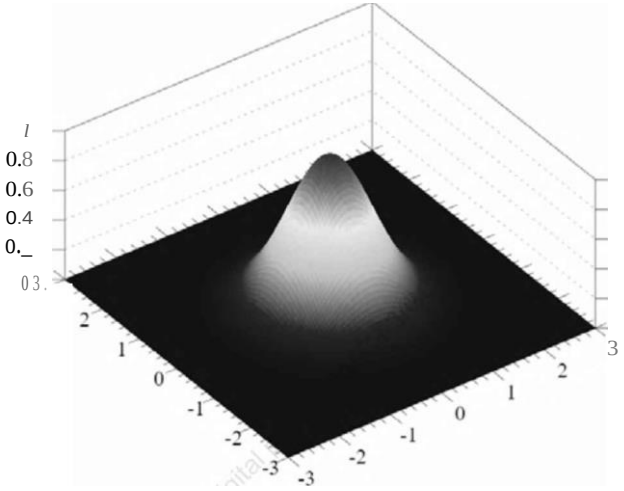

*Gambar 7. Citra asal dan citra yang dikaburkan*

### 7.1.2 Pengaburan dengan GaussianBlur()

Filter *Gaussian* didasarkan pada fungsi *Gaussian* untuk dua dimensi yang dirumuskan seperti berikut:

$$g(x,y)=e^{x2+y2}$$

Pada rumus ini, **ladalah** deviasi standar dan piksel pada pusat (x, y) mendapatkan bobot terbesar berupa 1. Gambar 7.3 memperlihatkan model fungsi ini.




***Gambar 7.3 Fungsi Gaussian untuk model dua dimensi***

Filter Gaussian paling tidak berukuran 5 x 5. Bobot-bobotnya dapat diperoleh dengan membuat I bernilai 1. Dengan demikian:

G(0,0)=e--o =1

G(1,o) = G(0,1) = G(-1,0) = G(0,-1) = $e^{-112}$ = 0,6065

G(l,1) = G(1,-1) = G(-1,1) = G(-1,-1) = e-[1] = 0,3679

G{2,1) = G{l,2) = G(-2,1) = G(-2,-1) = e[illegible]= 0,0821

G(2,0) = G(0,2) = G(0,-2) = G(-2,0) = e-[2] = 0.1353

G(2,2) = G(-2,-2) = G(-2,2) = G(2,-2) = e-[4] = 0,0183

Dengan mengatur nilai terkecil menjadi 1, maka setiap nilai di atas perlu dikalikan dengan 55 (diperoleh dari 1/0,0183 dan kemudian hasilnya dibulatkan ke atas). Hasilnya adalah seperti yang terlihat pada Gambar 7.4.

| | | | | |
|---|---|---|---|---|
| 1 | 5 | 7 | 5 | 1 |
| 5 | 20 | 33 | 20 | 5 |
| 7 | 33 | 55 | 33 | 7 |
| 5 | 20 | 33 | 20 | 5 |
| 1 | 5 | 7 | 5 | 1 |

*Gambar 7.4 Filter Gaussian*

Hasil setelah dinormalisasi ditunjukkan pada Gambar 7.5.

1/339

| | | | | |
|---|---|---|---|---|
| 1 | 5 | 7 | 5 | 1 |
| 5 | 20 | 33 | 20 | 5 |
| 7 | 33 | 55 | 33 | 7 |
| 5 | 20 | 33 | 20 | 5 |
| 1 | 5 | 7 | 5 | 1 |

*Gambar 7.5 Filter Gaussian yang telah mengalami nonnalisasi*

Dengan menggunakan metode cv2.GaussianBlur(), kita tidak perlu bersusah-payah memikirkan filter tersebut. Format penggunaannya seperti berikut:

*nilaiBalik* = cv2.GaussianBlur(citra, (n, n), *sigmaX)*

Dalam hal ini, argumen pertama berupa citra sumber. Argumen kedua berupa tupel yang menyatakan ukuran kernel. Argumen ketiga berupa $\sigma$ pada arah horizontal. Nilai balik berupa citra yang telah dikaburkan.

Contoh:

```
blur = cv2.GaussianBlur(citra,(5,5),0)
```

Pada contoh ini, kernel berukuran 5 x 5 dan $\sigma_x$ sebesar 0. Contoh ini dapat pada skrip berikut:

**Berkas : gaussblur.py**

```
# Pengaburan citra dengan GaussianBlur()

import cv2
import numpy as np

citra = cv2.imread('simba.png')

blur1 = cv2.GaussianBlur(citra, (5, 5), 0)
blur2 = cv2.GaussianBlur(citra, (45, 45), 0)

hasil = np.hstack((citra, blur1, blur2))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)

cv2.waitKey(0)
```

Akhir berkas

Hasilnya diperlihatkan pada Gambar 7.6. Tampak bahwa dengan menggunakan kernel berukuran 5 x 5, pengaburan tidak terlalu kentara.

Hasil yang jauh berbeda terlihat manakala kernel berukuran 45 x 45 digunakan.

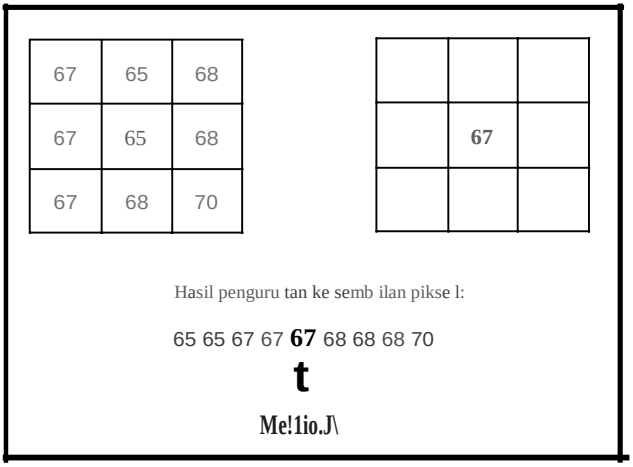

*Gambar 7.6 Hasil pengaburan citra dengan GaussBJur()*

### 7.1.3 Pengaburan dengan Tapis Median

Filter median sering juga dipakai untuk menghilangkan derau bintik-bintik. Sebagai contoh, dengan menggunakan kernel berukuran 3 x 3, perhitungan median dilakukan dengan memperhitungkan piksel itu sendiri dan kedelapan piksel tetangga. Secara matematis, filter dapat dinotasikan seperti berikut:

$$g(y,x) = median(f(x-1, y-1), f(x, y-1), f(x+1, y-1), f(x-1, y), f(x-1, y), f(x-1, y+1), f(x-1, y+1), f(x, y+1), f(x+1, y+1))$$

Supaya lebih jelas, perhatikan Gambar 7.7. Piksel yang berada di tengah bernilai 10. Maka, median dihitung melalui sembilan piksel. Setelah kesembilan nilai piksel diurutkan, didapatkan nilai median atau nilai yang terletak di tengah, yakni 12.




***Gambar 7. 7 Penentuan median***

Di OpenCV, pengaburan dengan tapis median dilakukan dengan menggunakan cv2.medianBlur (). Penggunaannya seperti berikut:

*median*= cv2.medianBlur(citra,*ukuran)*

Dalam hal ini, argumen pertama berupa citra sumber untuk pemrosesan dan argumen kedua berupa ukuran kernel. Ukuran kernel harus berupa bilangan ganjil. Nilai balik berupa median.

Skrip berikut menunjukkan contoh penggunaan cv2.medianBlur ():

**11!1** **Berkas** : **Illur.iey**

```
# Pengaburan citra dengan medianBlur()

import cv2
import numpy as np

citra   cv2.imread('simba.png')

blurl   cv2.medianBlur(citra, 5)
blur2   cv2.medianBlur(citra, 45)
```

```
hasil = np.hstack((citra, blurl, blur2))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)
```

**Akhir berkas**

Hasilnya ditunjukkan pada Gambar 7.8.


***Gambar 7.8 Hasil pengaburan dengan medianBJurO***

### 7.1.4 Penapisan Bilateral

Fungsi Gaussian membedakan pengaburan pada area di sekitar pusat piksel dan area yang jauh dari pusat piksel mengingat fungsi ini berbentuk menyerupai kurva bel. Kelemahannya, fungsi ini tidak memerhatikan kesamaan intensitas piksel sehingga bagian tepi yang tentu intensitasnya sangat bervariasi pun akan dikaburkan. Nah, kelemahan inilah yang diatasi pada penapisan bilateral sehingga bagian tepi tidak hilang.

Penapisan bilateral dilakukan dengan menggunakan `cv2.bilateralFilter()`. Bentuk pemakaiannya seperti berikut:

*nilaiBalik* = cv2.bilatera1Filter(citra, *diameter, sigmaWarna, sigmaRuang)*

**Argumen pertama berupa citra sumber untuk pemrosesan. Argumen kedua menyatakan diameter ketetanggaan piksel yang dipakai untuk penapisan. Argumen ketiga menyatakan nilai filter pada ruang warna. Nilai yang besar menyatakan rentang warna yang lebih panjang yang akan dianggap sama. Argumen keempat menyatakan nilai filter pada ruang koordinat. Nilai yang besar menyatakan rentang jarak yang lebih panjang yang akan saling mempengaruhi sepanjang memenuhi kriteria argumen ketiga.**

**Skrip berikut menunjukkan penggunaan** cv2.bilateralFilter()**:**

**11!1 Berkas : blur.jly**

```
# Pengaburan citra dengan bilateralFilter()

import cv2
import numpy as np

citra   cv2.imread('simba.png')

blur1   cv2.bilateralFilter(citra, 10, 75, 75)
blur2   cv2.bilateralFilter(citra, 10, 150, 150)

hasil   np.hstack((citra, blur1, blur2))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)

waitKey(0)
```

**Akhir berkas**

**Hasilnya diperlihatkan pada Gambar 7.9.**


***Gambar 7.9 Hasil pengaburan dengan bilateralFilterO***

## 7.2 Penapisan Citra

Penapisan citra dapat dilakukan untuk berbagai tujuan. Pengaburan citra, penghilangan derau pada citra, dan pemerolehan tepi pada citra adalah contoh-contoh operasi yang dapat dilakukan melalui penapisan.

Penapisan citra dapat menggunakan filter lolos-rendah *(/ow-pass* fi/ter/LPF), filter lolos-tinggi *(high-pass filter/HPF),* dll. Masing-masing digunakan untuk kepentingan khusus.

OpenCV menyediakan metode `cv2. fil ter2D ()` untuk kepentingan penerapan berbagai jenis filter. Metode ini melakukan suatu konvolusi dengan kernel yang dapat diatur sendiri oleh pemakai. Namun, karena terdapat istilah konvolusi dan kernel, pemakaian metode ini diterangkan belakangan

### 7.2.1 Pengertian Konvolusi dan Kernel

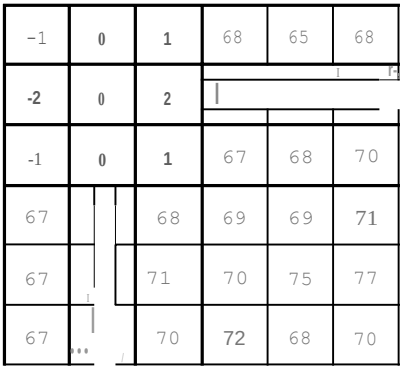
Konvolusi pada citra didefinisikan sebagai proses untuk memperoleh suatu nilai piksel didasarkan pada nilai piksel itu sendiri dan tetangganya, dengan melibatkan suatu matriks yang disebut kernel yang merepresentasikan bobot. Gambar 7.10 menunjukkan contoh kernel untuk konvolusi yang berukuran 3 x 3.

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

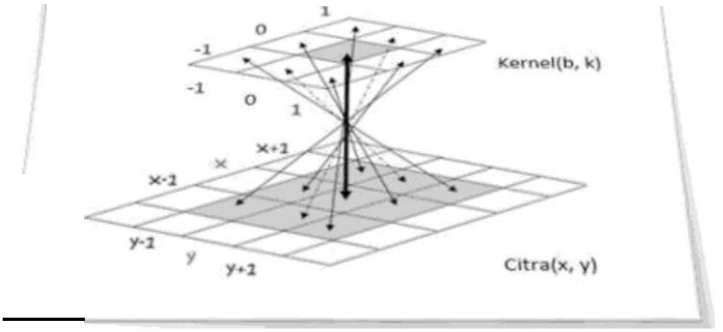
***Gambar 7.10 Contoh kernel untuk konvolusi***

Operasi konvolusi dilakukan dengan menumpangkan kernel yang berisi bobot pengali pada setiap piksel yang ditumpangi. Kemudian, nilai rerata diambil dari hasil-hasil perkalian tersebut. Apabila semua angka pengali tersebut bernilai 1, hasil yang didapat sama saja dengan filter pererataan.

Kernel digerakkan ke sepanjang baris dan kolom dalam citra seperti yang diilustrasikan pada Gambar 7.11 sehingga diperoleh nilai baru pada citra keluaran. Adapun nilai pada citra keluaran diperoleh melalui penjumlahan dari seluruh perkalian antara elemen di kernel dan di citra dengan mekanisme seperti yang diperlihatkan pada Gambar 7.12.




***Gambar 7.11 Konvolusi dilaksanakan dengan menggeser kernel di sepanjang kolom dan baris citra***




*Gambar 7.1 Ilustrasi konvolusi citra*

### 7.2.2 Pemakaian cv2.filter2D()

Bentuk pemakaian cv2. fil ter2D () yang digunakan untuk konvolusi citra adalah seperti berikut:

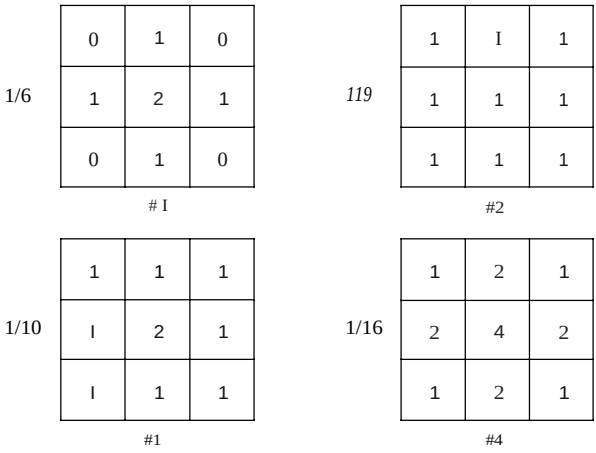
*nilaiBalik* = cv2.filter2D(citra, *jumBit, kernel)*

Argumen pertama berupa citra sumber yang hendak diproses. Argumen kedua menyatakan jumlah bit untuk citra keluaran, yang bisa berupa 8, 16, 32, dst. Apabila argumen ini diisi dengan -1, jumlah bit yang digunakan sama dengan jumlah bit yang digunakan pada argumen pertama. Argumen ketiga berupa kernel untuk konvolusi. Nilai balik metode berupa citra keluaran.

### 7.2.3 Filter Lolos-bawah

Filter lolos-bawah adalah filter yang digunakan untuk meloloskan bagian berfrekuensi rendah dan menghilangkan yang berfrekuensi tinggi. Hal ini membuat perubahan aras keabuan menjadi lebih lembut. Filter ini berguna untuk menghaluskan derau atau untuk kepentingan interpolasi tepi objek dalam citra.

**Contoh kernel yang bertindak sebagai filter lolos-bawah dapat dilihat pada Gambar 7.13.**




***Gambar 7.JS Contoh empat filter lolos-bawah***

**Contoh penerapan kernel #2 dapat dilihat pada skrip berikut:**

**11!1 Berkas : lolosbawah.py**

```
# Penerapan filter lolos-bawah

import cv2
import numpy as np

citra = cv2.imread('goldhill.png', 0)

kernel= np.float32([[1, 1, 1],
                    [1, 1, 1],
                    [1, 1, 1]]) / 9
terfilter = cv2.filter2D(citra, -1, kernel)

hasil = np.hstack((citra, terfilter))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)

.waitKey(O)
```

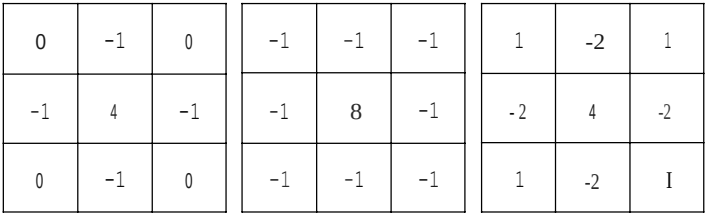
**Akhir berkas**

Skrip ini menerapkan kernel #2 pada Gambar 7.13 ke citra goldhill .png. Kernel ini tidak lain adalah filter pererataan yang dibahas pada Bagian 7.1.1. Hasilnya ditunjukkan pada Gambar 7.14. Tanda panah menekankan pada efek filter terhadap genting rumah. Detail genting menjadi hilang. Hal inilah yang menunjukkan bahwa filter lolos-bawah menghilangkan nilai-nilai yang sering muncul (frekuensi tinggi).


*Gambar 7.14 Penerapan filter lolos-bawah pada citra goldhill.png*

### 7.2.4 Filter Lolos-Tinggi

Filter lolos-tinggi adalah filter yang dipakai untuk melewatkan frekuensi tinggi dan menghalangi yang berfrekuensi rendah. Filter ini biasa dipakai untuk mendapatkan tepi objek dalam citra atau menajamkan citra. Contoh filter lolos-tinggi diperlihatkan pada Gambar 7.15.




*Gambar 7.15 Contoh tiga kernel filter lolos-tinggi*

Contoh penerapan ketiga filter pada Gambar 7.15 dapat dilihat pada skrip berikut:

11!1 Elerkas : lch.:r.11v

```
# Penerapan filter lolos-tinggi

import cv2
import numpy as np

citra = cv2.i mread('goldhill.png', 0)
jumSaris, jumKolom = citra .shape[: 2]

citra = cv2.resize  (citra,
           (int(0.5 * jumSaris), int(0.5*  jumKolom)))

kernelA   np.float32([[0, -1, OJ,
                       [-1, 4, -1],
                       [O, -1, O]])
kernels   np.float32([[-l, -1, -1],
                       [-1, 8, -1],
                       [-1, -1, -ll l)
kernelC   np.float32([[1, -2, l],
                       [-2, 4, -2],
                       [l, -2, 1] l)

filterA   cv2.filter2D(citra, -1, kernelA)
filters   cv2.filter2D(citra, -1, kernels)
filterC   cv2.filter2D(citra, -1, kernelC)

barisl   np.hstack((citra, filterA))
baris2   np.hstack((filterS, filterC))
```

```
hasil = np.vstack((barisl, baris2))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)
```

**Akhir berkas**

Hasilnya diperlihatkan pada Gambar 7.16.


***Gambar 7.16 Hasil penerapan filter lolos-tinggi***

### 7.2.5 Filter High-Boost

Filter "high boost" berguna untuk menajamkan citra. Kernelnya mempunyai nilai yang tinggi pada titik tengahnya. Sebagai contoh,

**kernel yang dapat digunakan untuk kepentingan ini dapat dilihat pada Gambar 7.17.**

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | c | -1 |
| -1 | -1 | -1 |

c > 8; **misalnya 9**

***Gambar 7.17 Contoh filter high-boost***

**Contoh penerapan tiga macam filter *high-boost* diperlihatkan pada skrip berikut:**

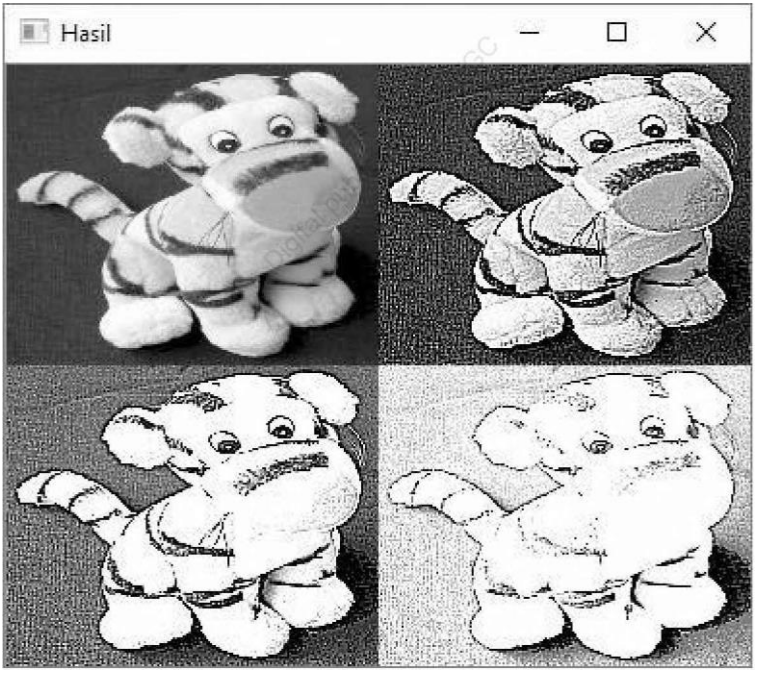
**11!1 Berkas : highboost.py**

```
# Penerapan filter high-boost

import cv2
import numpy as np

citra = cv2.imread('simba.png', 0)
jumBaris, jumKolom = citra.shape[:2]

citra =  cv2.resize(citra,
           (int(0.5 * jumBaris), int(0.5 * jumKolom)))

kernelA    np.float32([[-1, -1, -1],
                       [-1,  9, -1],
                       [-1, -1, -lJ  J)
kernelB    np.float32([[-1, -1, -1],
                       [-1, 10, -1],
                       [-1, -1, -lJ  J)
kernelC    np.float32([[-1, -1, -1],
                       [-1, 13, -1],
                       [-1, -1, -lJ  J)

filterA    cv2.filter2D(citra, -1, kernelA)
```

```
filters    cv2.filter2D(citra, -1, kernelB)
filterC    cv2.filter2D(citra, -1, kernelC)

barisl     np.hstack((citra, filterA))
baris2     np.hstack((filterB, filterC))

hasil = np.vstack((barisl, baris2))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)
```

**Akhir berkas**

**Efek penggunaan *C* dengan nilai 9, 10, dan 13 diperlihatkan pada Gambar 7.18.**




***Gambar 7.18 Contoh penerapan filter high-boost***

### 7.2.6 Filter Penonjolan

Gambar 7.19 menunjukkan contoh hasil penonjolan bagian tertentu dalam citra. Terlihat ada penebalan garis pada arah tertentu.

(a) Berdasar citra simba.png

(b) Berdasar citra lena.png

***Gambar 7.19 Hasil penonjolan pada arah tertentu***

Kernel yang digunakan untuk memperoleh efek seperti itu dapat dilihat pada Gambar 7.20.

| | | |
|---|---|---|
| -2 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 2 |

***Gambar 7. 0 Contoh tiga kernel filter lolos-tinggi***

**Skrip yang digunakan untuk menghasilkan citra seperti pada Gambar**

**m ng didasarkan pada** lena.png **adalah seperti berikut:**

**Berkas : emboss.py**

```
# Penonjolan pada arah tertentu

import cv2
import numpy as np

citra = cv2.imread('lena.png', 0

kernel= np.float32([[-2, 0, OJ,
                    [ 0,  0,  OJ,
                    [ 0, 0, 2J J l

terfilter = cv2.filter2D(citra, -1, kernel)

hasil = np.hstack((citra, terfilter))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)

waitKey(O)
```

**Akhir berkas**

**Nilai negatif dan positif yang berpasangan pada kernel untuk penonjolan menentukan perubahan kecerahan yang berefek pada penggambaran garis gelap atau terang. Gambar 7.21 memperlihatkan efek beberapa kernel dan hasilnya untuk citra** lena.png.

| | | |
|---|---|---|
| -1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 1 |

(a) Kernel #1

(b) Hasil untuk kernel #1

| | | |
|---|---|---|
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | -1 |

(c) Kernel #2

(d) Hasil untuk kernel #2

| | | |
|---|---|---|
| 0 | 0 | 0 |
| -4 | 0 | 4 |
| 0 | 0 | 0 |

(e) Kernel #3

(d) Hasil untuk kernel #3

***Gambar 7.1 Efek penonjolan untuk berbagai kernel***

Beberapa contoh lain kernel untuk penonjolan citra dapat dilihat pada Gambar 7.22. Anda bisa mempraktikkan sendiri untuk melihat efeknya.

| | | |
|---|---|---|
| -4 | -4 | 0 |
| -4 | 1 | 4 |
| 0 | 4 | 4 |

(a) Embossing dari arah kiri atas

| | | |
|---|---|---|
| -6 | 0 | 6 |
| -6 | 1 | 6 |
| -6 | 0 | 6 |

(b) Embossing dari arah kiri

| | | |
|---|---|---|
| 4 | 4 | 0 |
| 4 | 1 | -4 |
| 0 | -4 | -4 |

(c) Embossing dari arah kanan bawah

| | | |
|---|---|---|
| 6 | 0 | -6 |
| 6 | 1 | -6 |
| 6 | 0 | -6 |

( d) Embossing dari arah kanan

***Gambar** 7.f.lf.l **Berbagai kernel untuk penonjolan***

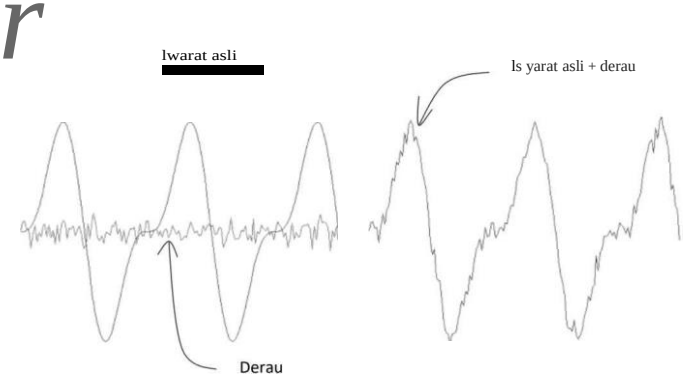
## 7.3 Pengurangan Derau

Target utama dalam subbab ini adalah mengenalkan beberapa cara untuk mengurangi derau yang muncul dalam suatu citra. Namun, sebelumnya, akan dibahas terlebih dahulu mengenai pengertian derau dan juga cara menambahkan derau pada citra.

### 7.3.1 Pengertian Derau

Derau adalah suatu di citra yang kehadirannya tidak dikehendaki, tetapi justru terkadang muncul. Dalam praktik, kehadiran derau tidak dapat dihindari. Sebagai contoh, derau *Gaussian* biasa muncul pada sebarang isyarat. Derau putih *(white noise)* biasa menyertai

pada siaran televisi yang berasal dari stasiun pemancar yang lemah. Derau butiran biasa muncul dalam film-film fotografi. Gambar 7.23 menunjukkan efek derau dalam isyarat satu-dimensi. Pada citra, derau muncul pada kolom dan baris.




***Gambar 7. s Derau pada isyarat satu dimensi***

## 7.3.2 Cara Menambahkan Derau

Untuk kepentingan percobaan dalam mengurangi derau, diperlukan cara untuk menambahkan derau ke citra. Kemudian, citra yang berderau ini dikenai operasi pengurangan derau. Hasilnya bisa diamati dengan membandingkannya dengan citra semula untuk melihat seberapa jauh cara pengurangan derau yang diterapkan memberikan efek.

**Derau Garam dan Merica**

Derau yang dinamakan 'garam dan merica' sering mewarnai citra. Bentuknya berwarna hitam dan putih. Disebut derau garam karena derau berwarna putih dan dinamakan derau garam karena berwarna hitam. Derau garam dan merica, sering muncul pada citra yang diperoleh melalui kamera.

Cara untuk membangkitkan derau garam dan merica dapat dilihat pada algoritma berikut.

**ALGORITMA 1.1- Membangkitkan derau garam dan merica**

Masukan:

- f: Citra berskala keabuan berukuran **M x** N
- p : Probabilitas pembangkitan derau (O s/d 1)

Keluaran:

- g : Citra yang telah ditambahi dengan derau

**derauGaramDanMrica(f, p):**

1. Salin citra **f keg**
2. **UNTUKy 1TOM**
   **UNTUKx** 1TON
   nilai_acak pembangkit_random
   **JIKA** nilai_acak p / 2
   **g(y, x)** 0 // merica (berwarna hitam)
   **LAINNYA**
   **JIKA** ni l ai_acak > p / 2 AND nilai_acak <= p
   **g(y, x)** 255 // Garam (berwarna putih)
   **AKHIR-JIKA**
   **AKHIR-JIKA**
   **AKHIR-JIKA**
   **AKHIR-JIKA**
3. **NILAI-BALIK** g

Untuk membuat derau garam dan merica adalah seperti berikut:

**Berkas** : **garamdanmerica.py**

```
# Pembentukan derau garam dan merica

import cv2
import random
import sys
```

```
if len(sys.argv) != 3:
    print('Pemakaian: python ' +
          'garamdanmerica.py namaberkas probabilitas')
    sys.exit()

berkas = sys.argv[l]
prob= float(sys.argv[2])
if prob> 1: prop= 0.1

citra = cv2.imread(berkas, 0)
if citra is None:
    print('Tidak dapat membaca berkas', berkas)
    sys.exit()

hasil = citra.copy()  # Salin isi citra
jumBaris, jumKolom = hasil.shape[:2]

for baris in  range(jumBaris):
    for kolom in range(jumKolom):
        nilaiAcak = random.random()
        if nilaiAcak <prob/ 2:
            hasil[baris, kolom]   0 # merica
        elif nilaiAcak > prob / 2 and\
             nilaiAcak <= prob:
            hasil[baris, kolom]   255 # Garam

cv2.imshow('Hasil rotasi', hasil)
cv2.waitKey()

# Simpan citra
    imwrite('simbagdm.png', hasil)
```

**Akhir berkas**

**Skrip ini melibatkan tiga modul. Modul** random **digunakan untuk memanfaatkan metode** random() **yang berguna untuk mendapatkan bilangan acak antara O dan 1. Modul** sys **dipakai untuk menangani baris perintah.**

**Perintah berikut digunakan untuk memastikan bahwa baris perintah menyertakan argumen untuk nama berkas citra dan probabilitas:**

```
if len(sys.argv) != 3:

    print('Pemakaian: python ' +
          'garamdanmerica.py namaberkas probabilitas')
    sys.exit()
```

**Apabila jumlah argumen baris perintah tidak sama dengan 3, pesan mengenai cara memanggil skrip ditampilkan dan eksekusi skrip dihentikan melalui** `sys.exit()`.

**Nama berkas dan probabilitas diperoleh melalui:**

```
    berkas = sys.argv[1]
    prob= float(sys.argv[2])
```

**Adapun perintah berikut digunakan untuk memberikan nilai probabilitas sebesar 0,01 sekiranya nilai** `prob` **lebih besar daripada 1:**

```
    if prob> 1: prop= 0.1
```

**Citra dibaca:**

```
    citra = cv2.imread(berkas, 0)
```

**Argumen kedua yang berupa 0 menyatakan bahwa citra akan dikonversi ke citra berskala keabu-abuan.**

**Perintah berikut digunakan untuk menyalin data di** `citra` **ke** `hasil`**:**

```
    hasil = citra.copy()
```

**Selanjutnya, jumlah baris dan kolom pada citra** `hasil` **diperoleh melalui:**

```
    jumBaris, jumKolom = hasil.shape[:2]
```

**Adapun perintah berikut digunakan untuk mewujudkan algoritma yang dipaparkan di depan:**

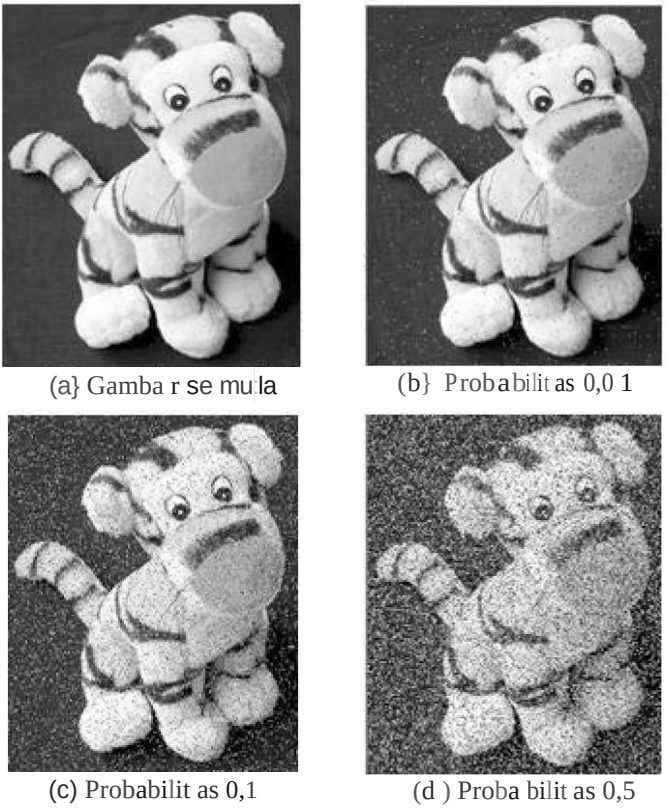
```
    for baris in  range(jumBaris):
        for kolom in range(jumKolom):
            nilaiAcak = random.random()
```

```
        if nilaiAcak <prob/ 2:
            hasil[baris, kolom]   0 # merica
         elif nilaiAcak >prob/ 2 and\
            nilaiAcak <= prob:
            hasil[baris, kolom]   255 # Garam
```

Untuk mencoba skrip ini, perintah seperti berikut bisa diberikan:

```
python garamdanmerica.py simba.png 0.01
```

Gambar 7.24 memberikan contoh citra semula dan citra setelah diberi derau untuk tiga macam nilai probabilitas.



(a} Gamba r se mu la (b} Proba bilit as 0,0 1

(c) Probabilit as 0,1 (d ) Proba bilit as 0,5

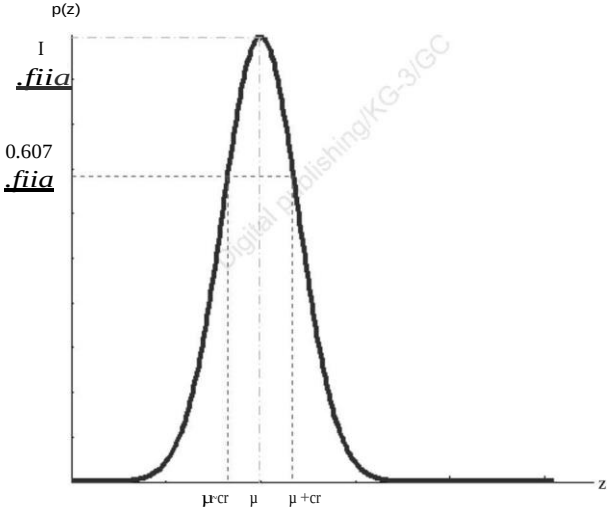
*Gambar 7.24 Pemberian derau garam dan merica*

**Derau Gaussian**

Derau *Gaussian* adalah model derau yang memiliki fungsi kerapatan probabilitas *(probability density function/PDF)* yang diberikan oleh kurva *Gaussian.* PDF yang mewakili sifat paling acak dalam bentuk satu dimensi seperti berikut:

$$p(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

Dalam hal ini, μ adalah nilai rerata dan cr adalah deviasi standar (atau akar varians) variabel random. PDF fungsi ini ditunjukkan pada Gambar 7.25.




*Gambar 7. 5 Fungsi kepadatan probabilitas derau Gaussian*

Derau *Gaussian* dapat dilakukan dengan menggunakan fungsi pembangkit bilangan acak random. gauss (). Bentuk penggunaannya seperti berikut:

random.gauss(mu, *sigma)*

**Dalam hal ini, argumen pertama menyatakan μ dan argumen kedua menyatakan *a*.**

**Contoh penambahan derau pada citra berskala keabu-abuan [illegible]ut:**

**11!1** **Berkas : cmhess.y**

```
# Pembentukan derau Gaussian

import cv2
import random
import sys

if len(sys.argv) != 3:
    print('Pemakaian: python ' +
          'drgaussian.py namaberkas sigma')
    sys.exit()

berkas = sys.argv[1]
sigma= float(sys.argv[2])
mu=  0

citra = cv2.imread(berkas, 0)
if citra is None:
    print('Tidak dapat membaca berkas', berkas)
    sys.exit()

hasil = citra.copy()  # Salin isi citra
jumBaris, jumKolom = hasil.shape[:2]

for baris in  range(jumBaris):
    for kolom in range(jumKolom):
        nilaiBaru = hasil[baris, kolom] + \
                    random.gauss(mu, sigma)
        if nilaiBaru > 255:
            nilaiBaru = 255
        else:
            if nilaiBaru < 0:
                nilaiBaru = 0

        hasil[baris, kolom]   int(nilaiBaru)

cv2.imshow('Hasil rotasi', hasil)
cv2.waitKey()
```

```
# Simpan citra
    imwrite('simbagauss.png', hasil)

    Akhir berkas
```

**Skrip ini melibatkan tiga modul. Modul** random **digunakan untuk memanfaatkan metode** random() **yang berguna untuk mendapatkan bilangan acak antara O dan 1. Modul** sys **dipakai untuk menangani baris perintah.**

**Perintah berikut digunakan untuk memastikan bahwa baris perintah menyertakan argumen untuk nama berkas citra dan probabilitas:**

```
if len(sys.argv) != 3:
    print('Pemakaian: python ' +
          'drgaussian.py namaberkas sigma')
    sys.exit()
```

**Apabila jumlah argumen baris perintah tidak sama dengan 3, pesan mengenai cara memanggil skrip ditampilkan dan eksekusi skrip dihentikan melalui** sys.exit ().

**Nama berkas dan** nilai **sigma diperoleh melalui:**

```
berkas = sys.argv[1]
sigma=  float(sys.argv[2])
```

**Adapun nilai** mu **ditetapkan sebesar 0.**

**Citra dibaca:**

```
citra = cv2.imread(berkas, 0)
```

**Argumen kedua yang berupa O menyatakan bahwa citra akan dikonversi ke citra berskala keabu-abuan.**

**Perintah berikut digunakan untuk menyalin data di** ci tra **ke** hasil:

```
hasil = citra.copy()
```

**Selanjutnya, jumlah baris dan kolom pada citra hasil diperoleh melalui:**

```
jumBaris, jumKolom = hasil.shape[:2]
```

**Adapun perintah berikut digunakan untuk menambahkan derau Gaussian:**

```
for baris in  range(jumBaris):
    for kolom in range(jumKolom):
        nilaiBaru = hasil[baris, kolom] + \
                    random.gauss(mu, sigma)
        if nilaiBaru > 255:
            nilaiBaru  = 255
        else:
            if nilaiBaru < 0:
                nilaiBaru = 0

        hasil[baris, kolom]   int(nilaiBaru)
```

**Dalam hal ini, mula-mula nilaiBaru diisi dengan penjumlahan nilai piksel semula dan derau Gaussian yang diperoleh melalui** random. gauss (mu, sigma). **Ada kemungkinan nilaiBaru ini memiliki nilai yang melebihi 255 atau kurang dari 0. Nilai di luar jangkauan O dan 255, harus diatur kembali agar berada di antara O dan 255. Itulah sebabnya, terdapat perintah:**

```
if nilaiBaru > 255:
    nilaiBaru = 255
else:
    if nilaiBaru < 0:
        nilaiBaru = 0
```

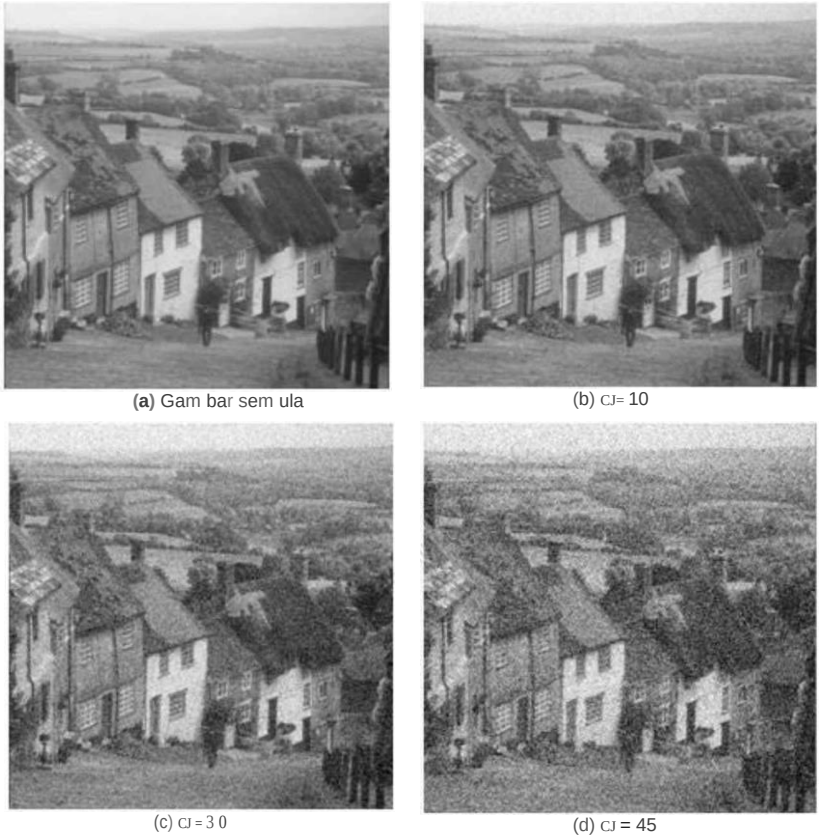
**Selanjutnya, nilai di nilaiBaru ini dikembalikan ke hasil melalui:**

```
hasil[baris, kolom] = int(nilaiBaru)
```

**Untuk mencoba skrip ini, perintah seperti berikut bisa diberikan:**

```
python drgaussian.py goldhill.png 10
```

Gambar 7.26 memberikan contoh citra semula dan citra setelah diberi derau untuk tiga macam nilai sigma.



(a) Gambar semula (b) CJ= 10

(c) CJ = 30 (d) CJ = 45

***Gambar 7.6 Hasil pengujian penambahan derau Gaussian***

### 7.3.3 Percobaan untuk Mengurangi Derau pada Citra

Derau pada citra dapat dikurangi dengan menggunakan cara yang digunakan untuk mengaburkan citra. Contoh berikut menunjukkan penggunaan filter median untuk mengurangi derau garam dan merica pada s imba. png:

Berkas : deraumedian.py

```
# Pengurangan derau pada citra menggunakan
#    filter median

import cv2
import numpy as np
import sys

citra = cv2.imread('simbagdm.png', 0)
if citra is None:
    print('Berkas citra tak dapat dibaca')
    sys.exit()

terfilter = cv2.medianBlur(citra, 5)

hasil = np.hstack((citra, terfilter))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)

waitKey(0)                    0
```

**Akhir berkas**

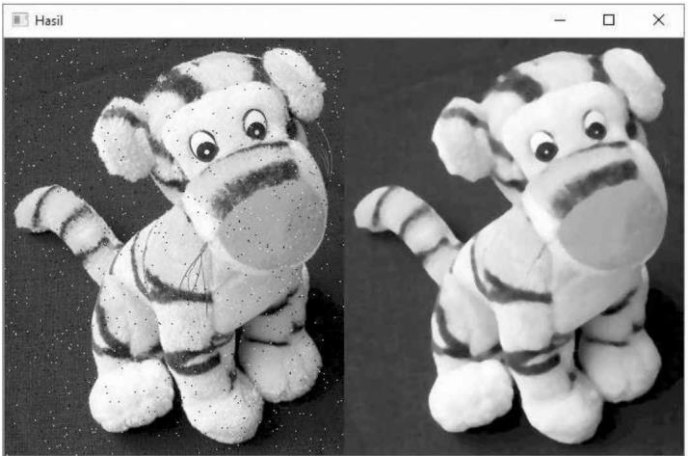
Untuk menguji skrip ini, mula-mula berikan perintah berikut:

```
python garamdanmerica.py simba.png 0.01
```

Perintah ini digunakan untuk membentuk berkas simbagdm. png dengan probabilitas sebesar 0.01. Selanjutnya, pengujian skrip deraumedian. py dilakukan dengan cara memberikan perintah seperti berikut:

```
python detaumedian.py
```

Gambar 7.27 menunjukkan gambar semula dan hasil pengurangan derau. Terlihat bahwa warna titik-titik pada Simba dan juga latar belakang menghilang.

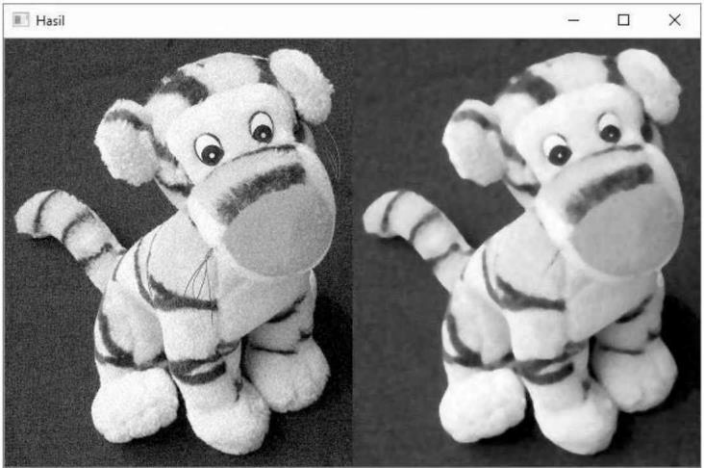



***Gambar** 7. 7**Hasil pengurangan derau garam dan merica melalui filter median***

Adapun Gambar 7.28 memperlihatkan efek filter median pada citra sirnbagauss. png yang mengandung derau Gaussian dengan sigma sebesar 10. Perlu diketahui, berkas ini diciptakan dengan cara memberikan perintah:

```
python drgaussian.py goldhill.png 10
```

Kemudian, pengurangan derau dilakukan dengan mengganti berkas sirnbagdrn.png pada skrip deraumedian.py menjadi sirnbagauss .png.




*Gambar 7. 8 Hasil pengurangan derau Gaussian melalui filter median*

### 7.3.4 Penghilangan derau Menggunakan fastNIMeansDenoising() dan fastNIMeansDenoisingColored()

OpenCV menyediakan `fastNlMeansDenoising()` yang secara khusus ditujukan untuk mengurangi derau pada citra berskala keabu-abuan. Metode ini menggunakan algoritma *Non-local Means Denoising,* yang dijabarkan dalam tulisan di `http://www.ipol.im/pub/art/2011/bcm_nlm/`. Adapun bentuk pemakaiannya seperti berikut:

cv2.fastNIMeansDenoising(sumber[, *hasil* [,
*h* [, *templateWindowSize* [, *searchWindowSize]]]])*

Berikut adalah penjelasan untuk argumen masing-masing.

- Argumen pertama menyatakan citra sumber yang berupa citra yang mengandung derau.

- Argumen kedua menyatakan citra hasil, yakni citra berdasar argumen pertama yang telah mengalami proses reduksi derau. Umumnya, argumen ini diisi dengan `None`, yang menyatakan tidak perlu ada penyebutan nama citra hasil.
- Argumen ketiga menyatakan kekuatan dalam mereduksi derau. Semakin besar nilainya, semakin baik dalam mengurangi derau. Akan tetap, detail citra juga ikut tereduksi. Nilai bawaannya adalah 3.
- Argumen keempat harus bernilai ganjil dengan nilai bawaannya adalah 7. Argumen ini menentukan ukuran piksel yang digunakan untuk menghitung bobot *template* untuk mereduksi derau.
- Argumen kelima harus bernilai ganjil dengan nilai bawaannya adalah 21. Argumen ini menentukan ukuran piksel jendela yang digunakan untuk menghitung rerata berbobot dari suatu piksel.

Nilai balik berupa citra yang komponen deraunya telah tereduksi.

**Skrip** berikut menunjukkan contoh penggunaan metode fastNlMeansDenoising():

**11!1** Berkas : **llero1umeci.m.11y**

```
# Pengurangan derau menggunakan
#     fastNlMeansDenoising()

import cv2
import numpy as np
import sys

citra = cv2.imread('simbagdm.png', 0)
if citra is None:
    print('Berkas citra tak dapat dibaca')
    sys.exit()
```

```
terfilter = cv2.fastN1MeansDenoising(citra,    None, 30)

hasil = np.hstack((citra, terfilter))

# Tampilkan citra asal dan hasilnya
cv2.imshow{'Hasil', hasil)

    waitKey(O)
```

**Akhir berkas**

Hasil skrip ini dapat dilihat pada Gambar 7.29.


***Gambar 7.f!9 Hasil pengurangan derau garam dan merica melalui lastNJMeansDenoisingO***

Untuk citra berwarna, pengurangan derau ditangani melalui `fastNlMeansDenoisingColored()`. Adapun bentuk pemakaiannya seperti berikut:

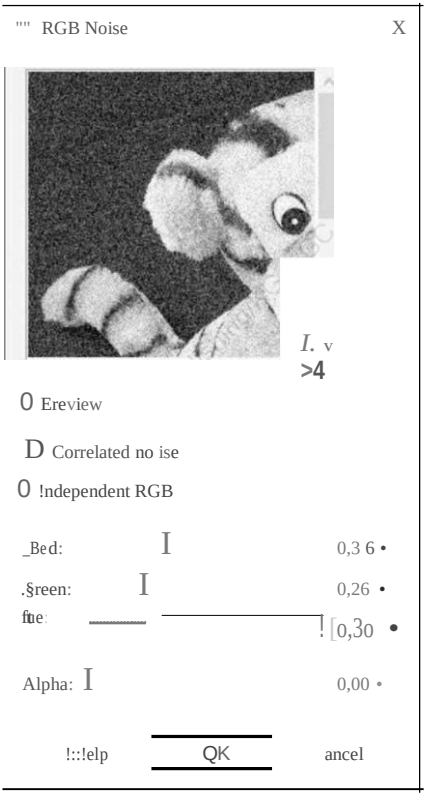
cv2.fastNIMeansDenoising(sumber[, *hasil* [,
*h* [,hForColorComponents *[,templateWindowSize*
[, *searchWindowSize]]]]])

Tambahan argumen `hForColorComponents` digunakan untuk mengatur kekuatan dalam mereduksi derau berdasarkan warna.

Perlu diketahui, jika Anda menggunakan aplikasi GIMP, penambahan derau berwarna dapat dilakukan melalui:

Filters | Noise | RGB Noises...

Contoh tampilannya dapat dilihat pada Gambar 7.30.




***Gambar 7.30 Pengaturan derau pada citra berwarna***

Skrip berikut menunjukkan contoh penggunaan metode `fastNlMeansDenoisingColored()`:

**11!1 Berkas : Iler,mmellian.:-y**

```
# Pengurangan derau menggunakan
#     fastNlMeansDenoising{)

import cv2
import numpy as np
import sys

citra = cv2.imread('simbagdm.png', 0)
if citra is None:
    print('Berkas citra tak dapat dibaca')
    sys.exit{)

terfilter = cv2.fastN1MeansDenoising(citra,    None, 30)

hasil = np.hstack((citra, terfilter))

# Tampilkan citra asal dan hasilnya
cv2.imshow('Hasil', hasil)

    waitKey(0)
```

**Akhir berkas**

**Hasilnya diperlihatkan pada Gambar 7.31.**


***Gambar 7.31 Hasil pengurangan derau RGB melalui lastNJMeansDenoisingColoredO***