BAB 4

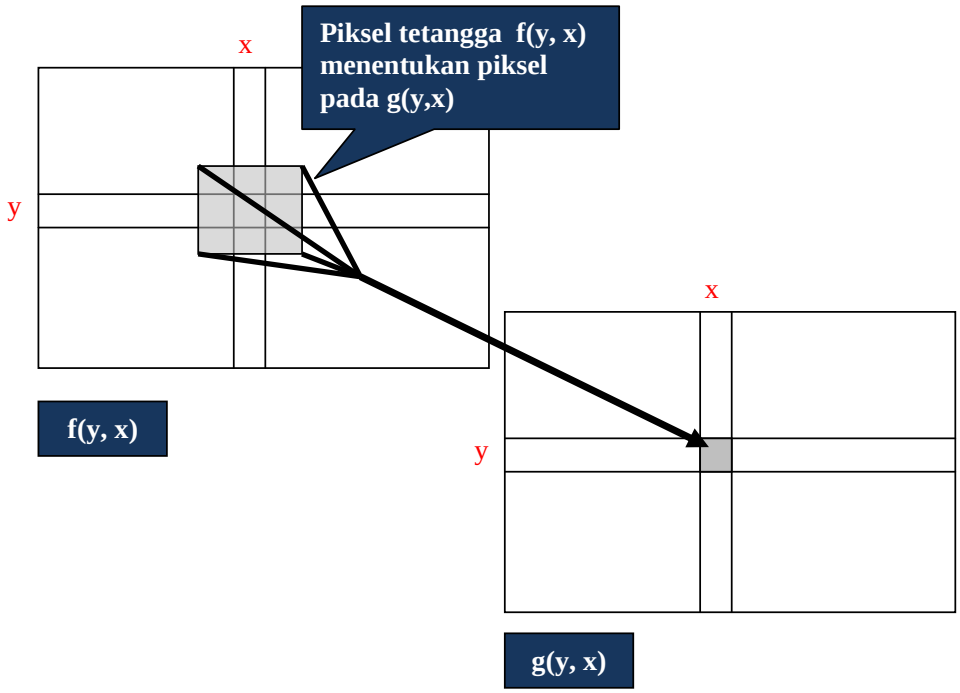
Operasi Ketetanggaan Piksel

Setelah bab ini berakhir, diharapkan pembaca mendapatkan pengetahuan mengenai hal-hal berikut dan cara mempraktikkannya.

- ✓ Pengertian operasi ketetanggaan piksel
- ✓ Pengertian ketetanggaan piksel
- ✓ Aplikasi ketetanggaan piksel pada filter batas
- ✓ Pengertian konvolusi
- ✓ Problem pada konvolusi
- ✓ Mempercepat komputasi pada konvolusi
- ✓ Pengertian frekuensi
- ✓ Filter lolos-rendah
- ✓ Filter lolos-tinggi
- ✓ Filter *high-boost*
- ✓ Efek *emboss*

4.1 Pengertian Operasi Ketetanggaan Piksel

Operasi ketetanggaan piksel adalah operasi pengolahan citra untuk mendapatkan nilai suatu piksel yang melibatkan nilai piksel-piksel tetangganya. Hal ini didasarkan kenyataan bahwa setiap piksel pada umumnya tidak berdiri sendiri, melainkan terkait dengan piksel tetangga, karena merupakan bagian suatu objek tertentu di dalam citra. Sifat inilah yang kemudian mendasari timbulnya algoritma untuk mengolah setiap piksel citra melalui piksel-piksel tetangga. Sebagai contoh, suatu citra yang berderau dapat dihaluskan melalui pererataan atas piksel-piksel tetangga. Gambar 4.1 memberikan ilustrasi operasi ketetanggaan piksel. Delapan piksel tetangga terdekat dengan piksel f(y,x) digunakan untuk memperbaikinya menjadi g(y,x) di tempat yang sama.

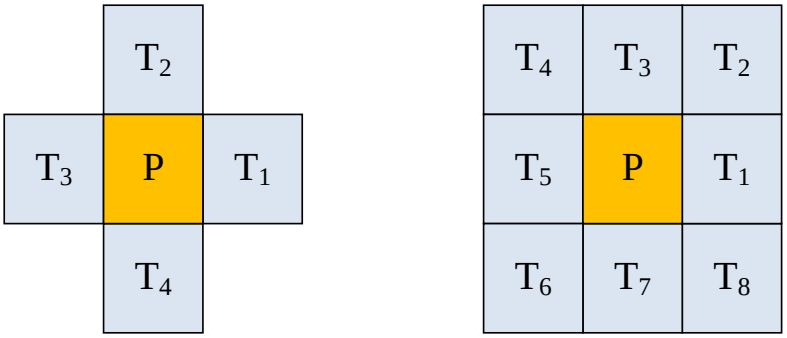


Gambar 4.1 Operasi ketetanggaan piksel.
Sejumlah tetangga menentukan nilai sebuah piksel

4.2 Pengertian Ketetanggaan Piksel

Pada pengolahan citra, ketetanggaan piksel banyak dipakai terutama pada analisis bentuk objek. Ketetanggaan piksel yang umum dipakai adalah 4-

ketetanggaan dan 8-ketetanggan. Untuk memahami dua jenis ketetanggaan piksel, lihat Gambar 4.2.



Gambar 4.2 Dua macam ketetanggaan piksel

Pada 4-ketetanggan, T_1, T_2, T_3, dan T_4 merupakan tetangga terdekat piksel P. Pada 8-ketetanggan, tetangga piksel P yaitu piksel-piksel yang berada di sekitar P. Totalnya sebanyak 8 buah. Bila P mempunyai koordinat (b, k) dengan b baris dan k kolom, hubungan piksel tetangga terhadap P sebagai berikut.

- Pada 4-ketetanggaan

$$T_1 = (b, k+1), T_2 = (b-1, k), T_3 = (b, k-1), T_4 = (b+1, k) \quad (4.1)$$

- Pada 8-ketetanggaan

$$\begin{aligned} &T_1 = (b, k+1), T_2 = (b-1, k-1), \\ &T_3 = (b, k-1), T_4 = (b-1, k-1) \\ &T_5 = (b, k-1), T_6 = (b+1, k-1), \\ &T_7 = (b+1, k-1), T_8 = (b+1, k+1) \end{aligned} \quad (4.2)$$

4.3 Aplikasi Ketetanggaan Piksel pada Filter

Ada tiga jenis filter yang menggunakan operasi ketetanggaan piksel yang akan dibahas sebagai pengantar pada bab ini. Ketiga filter tersebut adalah filter batas, filter pererataan, dan filter median. Sebagai filter atau tapis, operasi ketetanggaan piksel berfungsi untuk menyaring atau paling tidak mengurangi gangguan atau penyimpangan pada citra.

4.3.1 Filter Batas

Filter batas adalah filter yang dikemukakan dalam Davies (1990). Idenya adalah mencegah piksel yang intensitasnya di luar intensitas piksel-piksel tetangga. Algoritma yang digunakan untuk keperluan ini dapat dilihat berikut ini.

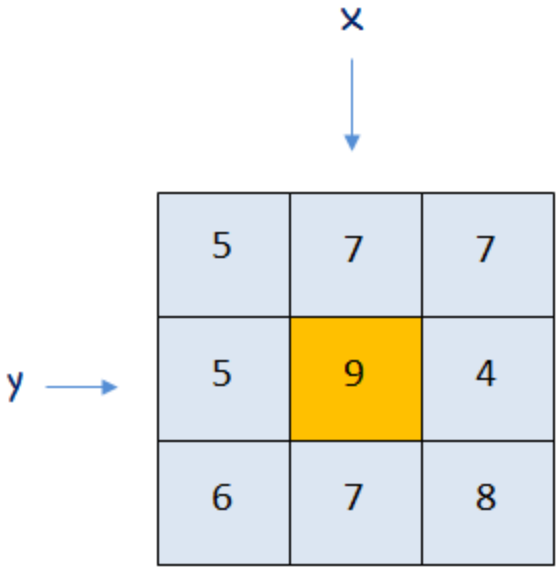
ALGORITMA 4.1 – Menghitung piksel dengan filter batas

Masukan:
- f(y, x) : Piksel pada posisi (y, x)

Keluaran:
- g(y, x) : Nilai intensitas untuk piksel pada citra g pada posisi (y, x)

1. Carilah nilai intensitas terkecil pada tetangga f(y, x) dengan menggunakan 8-ketetanggan dan simpan pada minInt.
2. Carilah nilai intensitas terbesar pada tetangga f(y, x) dengan menggunakan 8-ketetanggan dan simpan pada maksInt.
3. IF f(y, x) < minInt
   g(y, x) ← minInt
   ELSE
   IF f(y. x) > maksInt
   g(y, x) ← maksInt
   ELSE
   g(y, x) ← f(y, x)
   END-IF
   END-IF

Sebagai contoh, terdapat piksel seperti terlihat pada Gambar 4.3.



Gambar 4.3 *Contoh piksel dan tetangga*

Berdasarkan keadaan tersebut,

- minInt = minimum(5, 7, 7, 5, 4, 6, 7, 8) = 4;
- maksInt = maksimum(5, 7, 7, 5, 4, 6, 7, 8) = 8;
- mengingat f(y, x) bernilai 9 dan lebih besar daripada 8 (maksInt) maka g(y, x) bernilai 8;
- seandainya f(y, x) pada keadaan di atas bernilai 2 (bukan 9), g(y,x) akan bernilai 4.

Untuk melihat efek filter batas, cobalah program berikut.

Program : filbatas.m

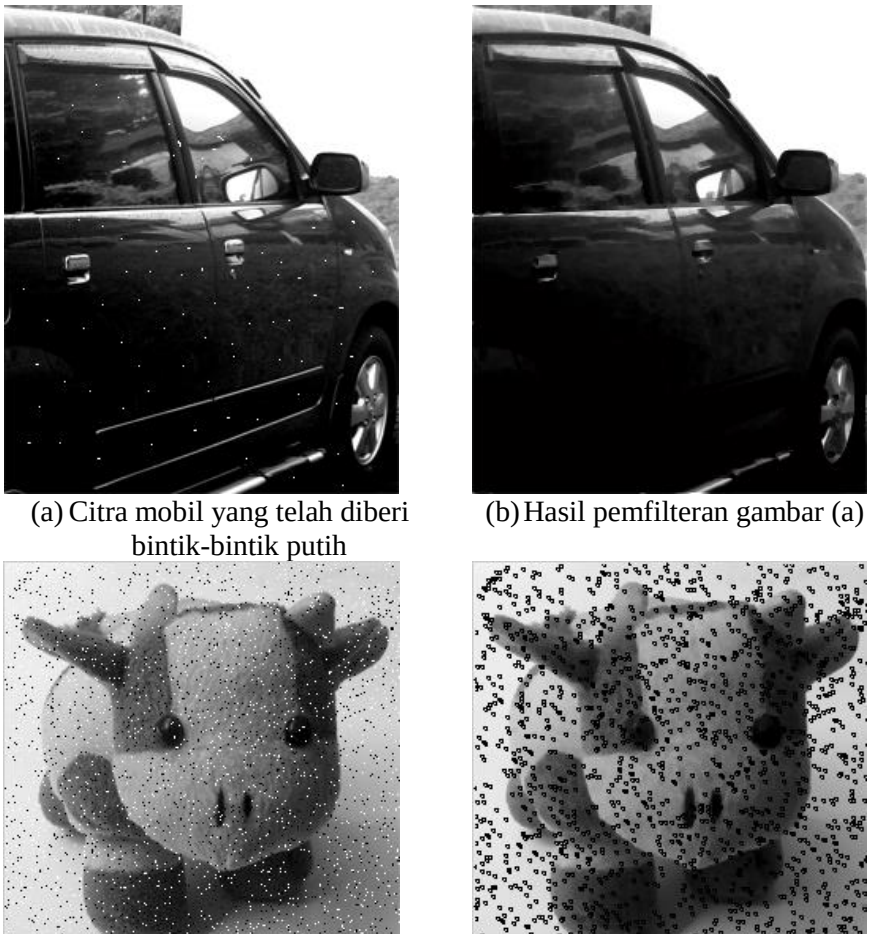
```
% FILBATAS Melakukan operasi ketetanggan piksel
%     menggunakan filter batas

F = imread('c:\Image\mobil.png');
Ukuran = size(F);
tinggi = Ukuran(1);
lebar = Ukuran(2);

G = F;
```

```
for baris=2 : tinggi-1
    for kolom=2 : lebar-1
        minPiksel = min([F(baris-1, kolom-1)       ...
            F(baris-1, kolom) F(baris, kolom+1)    ...
            F(baris, kolom-1)                      ...
            F(baris, kolom+1) F(baris+1, kolom-1)  ...
            F(baris+1, kolom) F(baris+1, kolom+1)]);
        maksPiksel = min([F(baris-1, kolom-1)       ...
            F(baris-1, kolom) F(baris, kolom+1)    ...
            F(baris, kolom-1)                      ...
            F(baris, kolom+1) F(baris+1, kolom-1)  ...
            F(baris+1, kolom) F(baris+1, kolom+1)]);

        if F(baris, kolom) < minPiksel
           G(baris, kolom) = minPiksel;
        else
            if F(baris, kolom) > maksPiksel
                G(baris, kolom) = maksPiksel;
            else
                G(baris, kolom) = F(baris, kolom);
            end
        end
    end
end

figure(1);
imshow(G);

clear;
```

Akhir Program

Perlu diketahui, pemrosesan hanya dilakukan selain baris pertama, baris terakhir, kolom pertama, dan kolom terakhir. Keempat area tersebut tidak diproses karena tidak mempunyai tetangga yang lengkap (sebanyak 8).

Untuk melihat efek filter batas, jalankan program di atas. Kemudian, bandingkan citra asli dan citra yang dihasilkan oleh program tersebut. Gambar 4.3 memperlihatkan perbedaannya.



(a) Citra mobil yang telah diberi bintik-bintik putih

(b) Hasil pemfilteran gambar (a)

Gambar 4.4 Efek filter batas terhadap citra yang mengadung derau

Terlihat bahwa bintik-bintik putih pada citra mobil.png dapat dihilangkan. Namun, kalau diperhatikan dengan saksama, operasi tersebut juga mengaburkan citra. Pada citra boneka2.png, derau malah diperkuat. Artinya, filter itu tidak cocok digunakan untuk menghilangkan jenis derau yang terdapat pada citra tersebut.

4.3.2 Filter Pererataan

Filter pererataan (Costa dan Cesar, 2001) dilakukan dengan menggunakan rumus:

$$g(y,x) = \frac{1}{9}\sum_{p=-1}^{1}\sum_{q=-1}^{1} f(y+p, x+q) \qquad (4.3)$$

Sebagai contoh, piksel pada f(y, x) dan kedelapan tetangganya memiliki nilai-nilai kecerahan seperti berikut.

65	50	55
78	68	60
60	60	62

Pada contoh di atas, yang diarsir (yaitu yang bernilai 68) merupakan nilai pada f(y, x). Nilai rerata pengganti untuk g(y, x) dihitung dengan cara seperti berikut:

$$g(y, x) = 1/9 \times (65+50+55+76+68+60+60+60+62) = 61{,}7778$$
$$\cong 62$$

Jadi, nilai 68 pada f(y, x) diubah menjadi 62 pada g(y, x).

Implementasi dalam program dapat dilihat berikut ini.

Program : pemerataan.m

```
% PEMERATAAN Melakukan operasi dengan filter pererataan

F = imread('C:\Image\mobil.png');
[tinggi, lebar] = size(F);

F2 = double(F);
for baris=2 : tinggi-1
    for kolom=2 : lebar-1
        jum = F2(baris-1, kolom-1)+ ...
              F2(baris-1, kolom) + ...
              F2(baris-1, kolom-1) + ...
              F2(baris, kolom-1) + ...
              F2(baris, kolom) + ...
              F2(baris, kolom+1) + ...
              F2(baris+1, kolom-1) + ...
              F2(baris+1, kolom) + ...
              F2(baris+1, kolom+1);

         G(baris, kolom) = uint8(1/9 * jum);
    end
end
```

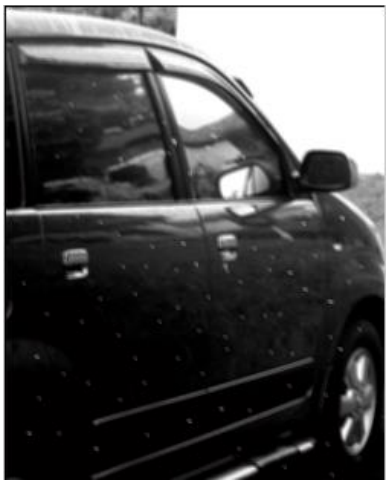
```
figure(1); imshow(G);
clear;
```

Akhir Program

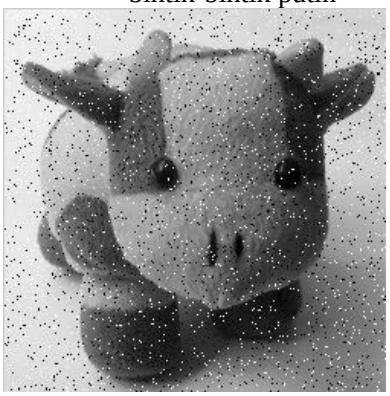
Pada program di atas baris dan kolom yang terletak di pinggir citra tidak ikut diproses.

Gambar 4.5 menunjukkan efek pemrosesan dengan filter pererataan. Dibandingkan dengan filter batas, hasil pemrosesan filter pererataan tidak menghilangkan bintik-bintik putih pada citra mobil, tetapi hanya agak menyamarkan. Pada citra boneka2.png, derau lebih dihaluskan.

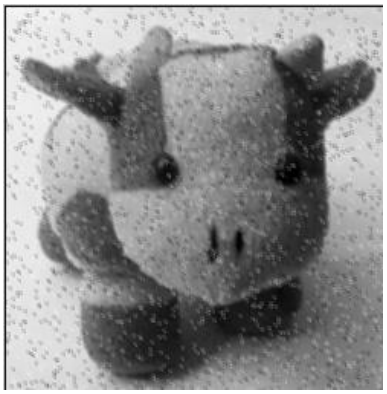
(a) Citra mobil dengan bintik-bintik putih



(b) Hasil pemrosesan mobil



(c) Citra boneka berbintik dengan derau



(d) Hasil pemrosesan boneka

Gambar 4.5 Contoh penerapan filter pererataan

Catatan

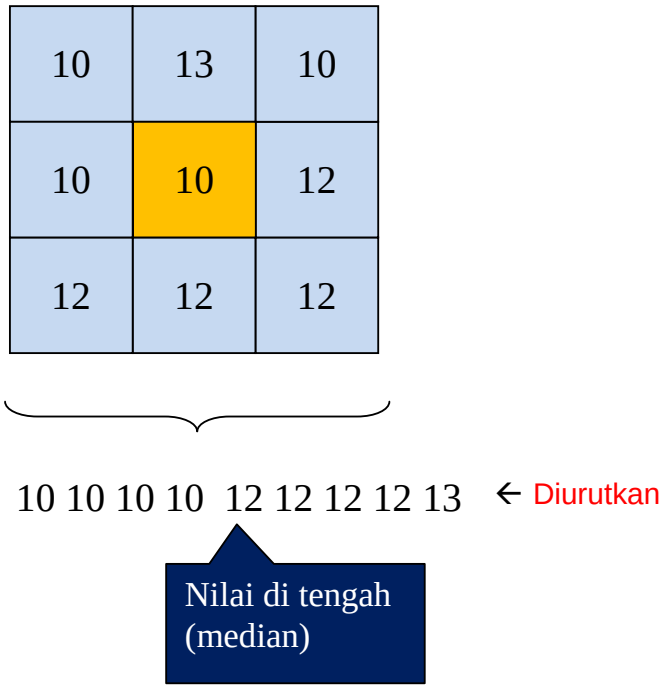
Perhatikan hasil pemrosesan filter pada Gambar 4.5(b) dan Gambar 4.5(d). Terlihat keberadaan garis pada kolom pertama dan baris pertama. Untuk menghindari efek seperti itu, baris pertama, kolom pertama, baris terakhir, dan kolom terakhir perlu dihilangkan. Jadi, efek "bingkai" dihilangkan dengan memperkecil ukuran citra menjadi (N-2) x (N-2) jika ukuran citra semula adalah N x N.

4.3.3 Filter Median

Filter median sangat populer dalam pengolahan citra. Filter ini dapat dipakai untuk menghilangkan derau bintik-bintik. Nilai yang lebih baik digunakan untuk suatu piksel ditentukan oleh nilai median dari setiap piksel dan kedelapan piksel tetangga pada 8-ketetanggaan. Secara matematis, filter dapat dinotasikan seperti berikut:

$$\begin{aligned} g(y,x) = median(& \\ & f(y-1,x-1), f(y-1,x), f(y-1,x+1), \\ & f(y,x-1), f(y,x), f(y,x+1), \\ & f(y+1,x-1), f(y+1,x), f(y+1,x+1)) \end{aligned} \qquad (4.4)$$

Contoh untuk satu piksel ditunjukkan pada Gambar 4.6.



Gambar 4.6 Gambaran operasi penggunaan filter median

Pada contoh di atas terlihat bahwa untuk mendapatkan median, diperlukan pengurutan (*sorting*) terlebih dulu.

Contoh berikut menunjukkan penggunaan filter median.

Program : filmedian.m

```
% FILMEDIAN Melakukan operasi dengan filter median

F = imread('C:\Image\mobil.png');
[tinggi, lebar] = size(F);

for baris=2 : tinggi-1
    for kolom=2 : lebar-1
        data = [F(baris-1, kolom-1) ...
              F(baris-1, kolom)  ...
              F(baris-1, kolom+1)  ...
              F(baris, kolom-1)  ...
              F(baris, kolom)  ...
              F(baris, kolom+1)  ...
              F(baris+1, kolom-1)  ...
              F(baris+1, kolom)  ...
              F(baris+1, kolom+1)];
```

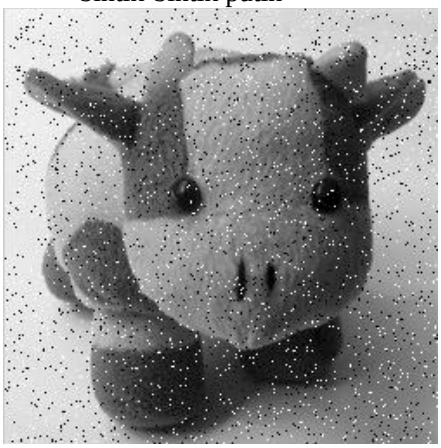
```
            % Urutkan
            for i=1 : 8
                for j=i+1 : 9
                    if data(i) > data(j)
                        tmp = data(i);
                        data(i) = data(j);
                        data(j) = tmp;
                    end
                end
            end

            % Ambil nilai median
            G(baris, kolom) = data(5);
        end
    end

figure(1); imshow(G);

clear;
```

Akhir Program

Contoh hasil penggunaan filter median dapat dilihat pada Gambar 4.7.

(a) Citra mobil dengan bintik-bintik putih

(b) Hasil pemrosesan terhadap gambar (a)



(c) Citra boneka dengan derau

(d) Hasil pemrosesan terhadap gambar (c)

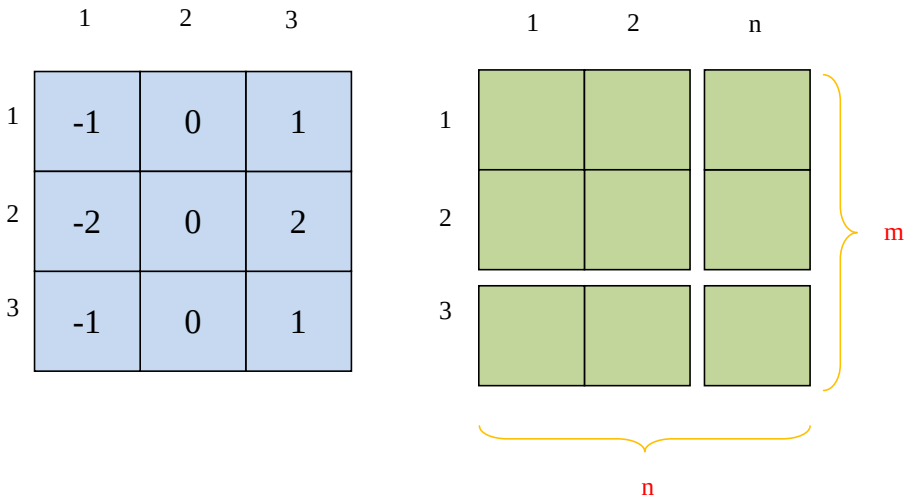
Gambar 4.7 Contoh penerapan filter median

Hasilnya terlihat bahwa derau dapat dihilangkan, tetapi detail pada citra tetap dipertahankan. Namun, hal ini tentu saja didapat dengan tambahan beban komputasi "pengurutan".

4.4 Pengertian Konvolusi

Konvolusi seringkali dilibatkan dalam operasi ketetanggaan piksel. Konvolusi pada citra sering disebut sebagai konvolusi dua-dimensi (konvolusi 2D). Konvolusi 2D didefinisikan sebagai proses untuk memperoleh suatu piksel didasarkan pada nilai piksel itu sendiri dan tetangganya, dengan melibatkan suatu matriks yang disebut kernel yang merepresentasikan pembobotan. Wujud kernel

umumnya bujur sangkar, tetapi dapat pula berbentuk persegi panjang. Gambar 4.8 menunjukkan contoh kernel untuk konvolusi.



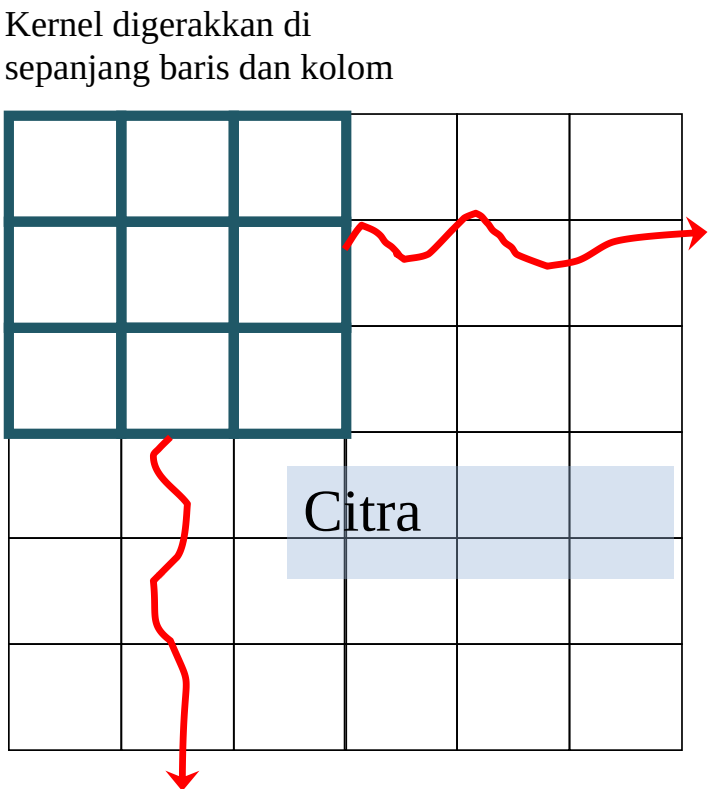
Gambar 4.8 Contoh kernel untuk konvolusi berukuran 3 x 3 dan m x n

Catatan

Kernel konvolusi terkadang disebut dengan istilah cadar, cadar konvolusi, atau cadar spasial.

Secara umum, proses penapisan di kawasan ruang (*space domain*), sebagai alternatif di kawasan frekuensi, dilaksanakan melalui operasi konvolusi. Operasi ini dilakukan dengan menumpangkan suatu jendela (kernel) yang berisi angka-angka pengali pada setiap piksel yang ditimpali. Kemudian, nilai rerata diambil dari hasil-hasil kali tersebut. Khusus bila angka-angka pengali tersebut semua adalah 1, hasil yang didapat sama saja dengan filter pererataan.

Pada pelaksanaan konvolusi, kernel digeser sepanjang baris dan kolom dalam citra (lihat Gambar 4.9) sehingga diperoleh nilai yang baru pada citra keluaran.



Gambar 4.9 Konvolusi dilakukan dengan melakukan proses di sepanjang kolom dan baris pada citra

Bagaimana konvolusi dilakukan? Prosesnya dirumuskan sebagai berikut:

$$g(y,x) = \sum_{p=-m2}^{m2} \sum_{q=-n2}^{n2} h(p+m2+1, q+n2+1) f(y-p, x-q) \quad (4.5)$$

Dalam hal ini,

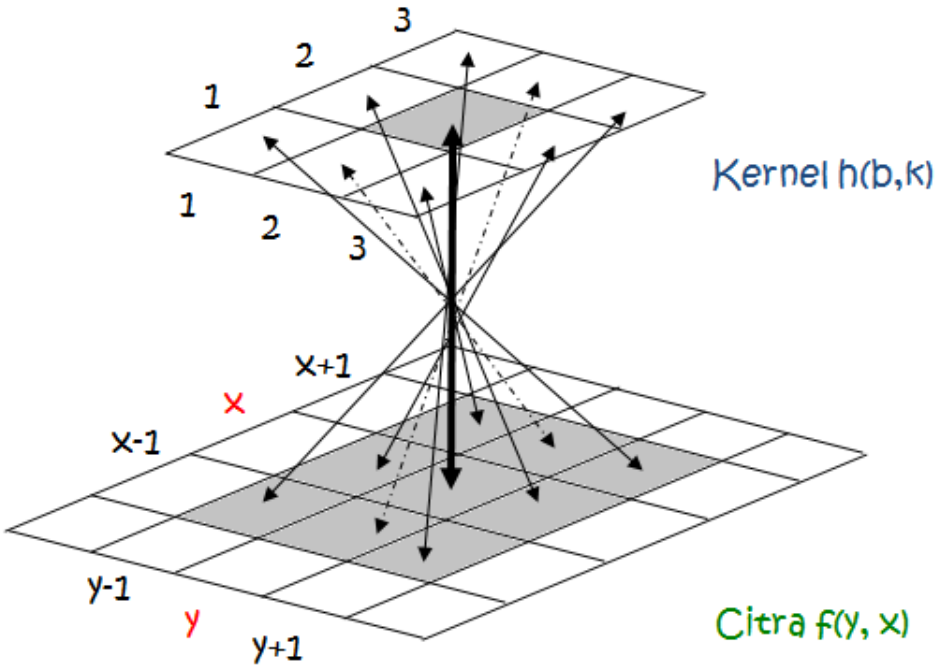
- m2 adalah separuh dari tinggi kernel (m2 = floor(m/2)),
- n2 adalah separuh dari lebar kernel (n2 = floor(n/2)),
- **floor** menyatakan pembulatan ke bawah, dan
- h menyatakan kernel, dengan indeks dimulai dari 1.

Catatan

Apabila kernel h diputar sebesar 180° (dapat dilaksanakan dengan perintah k = rot90(h)), perhitungan g(y,x) dapat diperoleh melalui:

$$g(y,x) = \sum_{p=-m2}^{m2} \sum_{q=-n2}^{n2} k(p,q) f(y+p, x+q)$$

Ilustrasi konvolusi dijelaskan melalui contoh pada Gambar 4.10.



Gambar 4.10 Contoh konvolusi

Berdasarkan Gambar 4.10, apabila citra yang berada pada jendela kernel berupa

65	50	55
78	68	60
60	60	62

dan kernel berupa

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

maka nilai piksel hasil konvolusi berupa:

$$
\begin{aligned}
g(x, y) = \; & -1 \times 62 + 0 \times 60 + 1 \times 60 + \\
& -2 \times 60 + 0 \times 68 + 2 \times 78 + \\
& -1 \times 55 + 0 \times 50 + 1 \times 65 \\
= \; & -62 + 0 + 60 - 120 + 0 + 152 - 55 + 0 + 65 \\
= \; & 40
\end{aligned}
$$

Dengan demikian, nilai 68 akan diubah menjadi 40 pada citra keluaran.

Catatan

- Apabila kernel h diputar sebesar 180° (k = rot180(h)), perhitungan g(y,x) dapat diperoleh melalui

$$g(y,x) = \sum_{p=-m2}^{m2} \sum_{q=-n2}^{n2} h(p+m2+1, q+n2+1) f(y+p, x+q)$$

- Untuk kernel yang simetrik (k(y,x)=k(x,y)), proses konvolusi sama dengan tidak melalui konvolusi (yaitu korelasi).

Berikut adalah salah satu algoritma yang dipakai untuk mengimplementasikan konvolusi pada citra, dengan asumsi kernel mempunyai jumlah baris dan kolom bernilai ganjil.

ALGORITMA 4.2 – Konvolusi pada citra dengan mengabaikan bagian tepi

Masukan:
- f : Citra yang akan dikonvolusi
- h : kernel konvolusi

Keluaran:
- g : Citra hasil konvolusi

```
1. m2 ← floor(jumlah_baris_kernel h)
2. n2 ← floor(jumlah_lebar_kernel h)
3. FOR y ← m2+1 TO tinggi_citra_f – m2
       FOR x ←n2+1 TO lebar_citra_f – n2
          // Lakukan konvolusi
          jum ← 0;
          FOR p ← -m2 TO m2
             FOR q ← -n2 TO n2
                jum ← jum * h(p+m2+1, q+n2+1) * f(y-p, x-p)
             END-FOR
          END-FOR
          g2(y, x) ← jum
       END-FOR
   END-FOR
4. // Salin posisi g2 ke g dengan membuang yang tidak dikonvolusi
```

```
5.  FOR y ← m2+1 TO tinggi_citra_f – m2
        FOR x ← n2+1 TO lebar_citra_f – n2
            g(y-m2, x-n2) ← g2(y, x)
        END-FOR
    END-FOR
```

Berdasarkan algoritma di atas, maka citra hasil akan kehilangan sebesar:

- 2 * m2 baris atau sama dengan jumlah baris kernel dikurangi 1
- 2 * n2 kolom atau sama dengan jumlah kolom kernel dikurangi 1

Baris dan kolom yang dihilangkan adalah yang berada di tepi citra.

Fungsi yang digunakan untuk melakukan konvolusi dapat dilihat berikut ini.

Program : konvolusi.m

```
function [G] = konvolusi(F, H)
% KONVOLUSI Melakukan konvolusi kernel H dengan citra F
%     H harus mempunyai tinggi dan lebar ganjil
%     Hasil: citra G

[tinggi_f, lebar_f] = size(F);
[tinggi_h, lebar_h] = size(H);

m2 = floor(tinggi_h/2);
n2 = floor(lebar_h/2);

F2=double(F);
for y=m2+1 : tinggi_f-m2
    for x=n2+1 : lebar_f-n2
        % Pelaksanaan konvolusi F(baris, kolom)
        jum = 0;
        for p=-m2 : m2
            for q=-n2 : n2
                jum = jum + H(p+m2+1,q+n2+1) * ...
                      F2(y-p, x-q);
            end
        end

        G(y-m2, x-n2) = jum;
    end
end
```

Akhir Program

Contoh pemakaian fungsi `konvolusi` ditunjukkan berikut ini.

```
>> H = [-1 0 -1; 0 4 0; -1 0 -1]; ↵
>> F = imread('C:\Image\gedung.png'); ↵
>> K = konvolusi (F, H); ↵
```

Pertama-tama, kernel H ditentukan melalui

```
H=[-1 0 -1; 0 4 0; -1 0 -1];
```

Kernel di atas dinamakan "Quick Mask" (Phillips, 2000) dan berguna untuk deteksi tepi.

Selanjutnya, citra gedung.png dibaca dan diletakkan di F. Lalu, konvolusi dilaksanakan dengan memanggil fungsi konvolusi. Dengan cara seperti itu, K berisi hasil konvolusi citra F dan kernel H. Nilai K dapat dilihat secara sekilas dengan mengetikkan

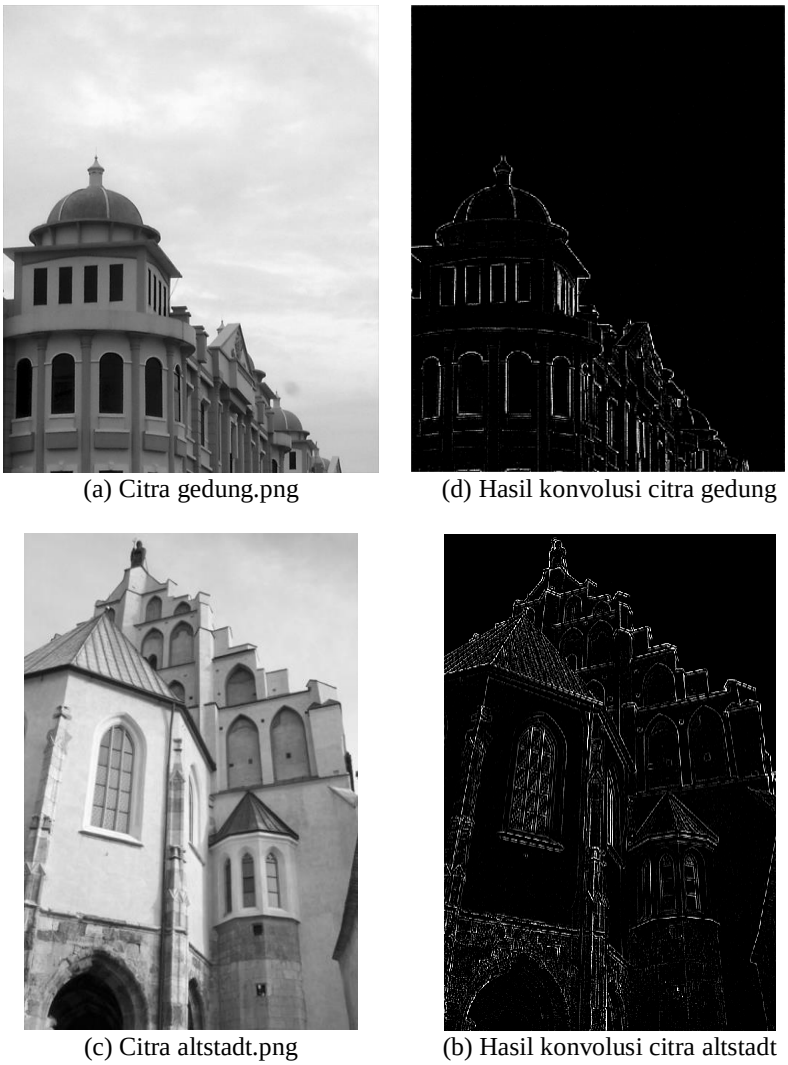
```
>> K
```

Nilai yang dihasilkan dengan konvolusi dapat bernilai negatif dan bahkan dapat melebihi nilai 255. Oleh karena itu, pemrosesan konvolusi harus dilaksanakan dengan menggunakan presisi ganda (bukan bilangan bulat). Lalu, setelah semua citra diproses dengan konvolusi, perlu dilakukan pengaturan nilai piksel agar berada pada jangkauan [0, 255]. Nilai yang kurang dari 0 diubah menjadi 0 dan yang melebihi 255 diubah menjadi 255. Fungsi **uint8** dapat digunakan untuk kepentingan tersebut Contoh:

```
>> K2 = uint8(K); ↵
```

Dengan cara seperti itu, nilai pada K2 berada pada jangkauan [0, 255]. Citra K2 dapat ditampilkan dengan menggunakan **imshow**.

Gambar 4.11 memperlihatkan contoh citra asli (tersimpan dalam F) dan citra yang telah mengalami konvolusi dan telah diatur agar bernilai dalam jangkauan [0, 255] (tersimpan dalam K2).



(a) Citra gedung.png (d) Hasil konvolusi citra gedung

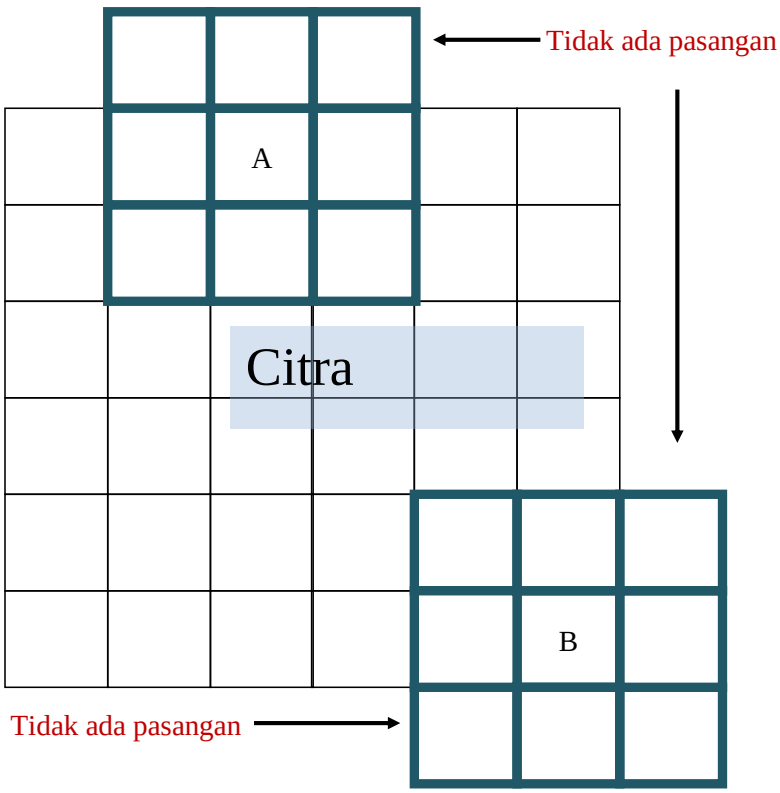
(c) Citra altstadt.png (b) Hasil konvolusi citra altstadt

Gambar 4.11 Contoh penerapan konvolusi

Contoh di atas menunjukkan aplikasi konvolusi yang dapat digunakan untuk mendapatkan tepi objek. Namun, tentu saja aplikasi konvolusi tidak hanya untuk kepentingan seperti itu. Untuk memperlihatkan hasil deteksi tepi objek, nilai-nilai piksel hasil perlu dinaikkan yaitu ditambah 127.

4.5 Problem pada Konvolusi

Pada Algoritma 4.2, terlihat bahwa tidak semua piksel dikenai konvolusi, yaitu baris dan kolom yang terletak di tepi citra. Hal ini disebabkan piksel yang berada pada tepi tidak memiliki tetangga yang lengkap sehingga tentu saja rumus konvolusi tidak berlaku pada piksel seperti itu. Gambar 4.12 menjelaskan contoh tentang hal ini. Sebagai contoh, konvolusi tidak mungkin dilakukan pada posisi A dan B.



Gambar 4.12 Problem pada konvolusi.
Ada bagian dari kernel yang tidak punya pasangan dengan piksel

Problem konvolusi pada piksel yang tidak mempunyai tetangga lengkap dibahas pada beberapa literatur (Efford, 2000 dan Heijden, 2007; Burger dan Burge, 2008). Untuk mengatasi keadaan seperti itu, terdapat beberapa solusi.

1. **Abaikan piksel pada bagian tepi.**

Cara ini yang dilakukan pada Algoritma 4.2. Karena pada bagian tepi citra, tetangga tidak lengkap maka piksel pada posisi tersebut tidak dikenai konvolusi. Sebagai konsekuensinya, citra yang tidak mengalami konvolusi maka diisi dengan nol atau diisi sesuai nilai pada citra asal. Alternatif lain (seperti pada contoh program konvolusi.m), bagian yang tidak diproses tidak diikutkan dalam citra hasil. Akibatnya, ukuran citra hasil mengecil.

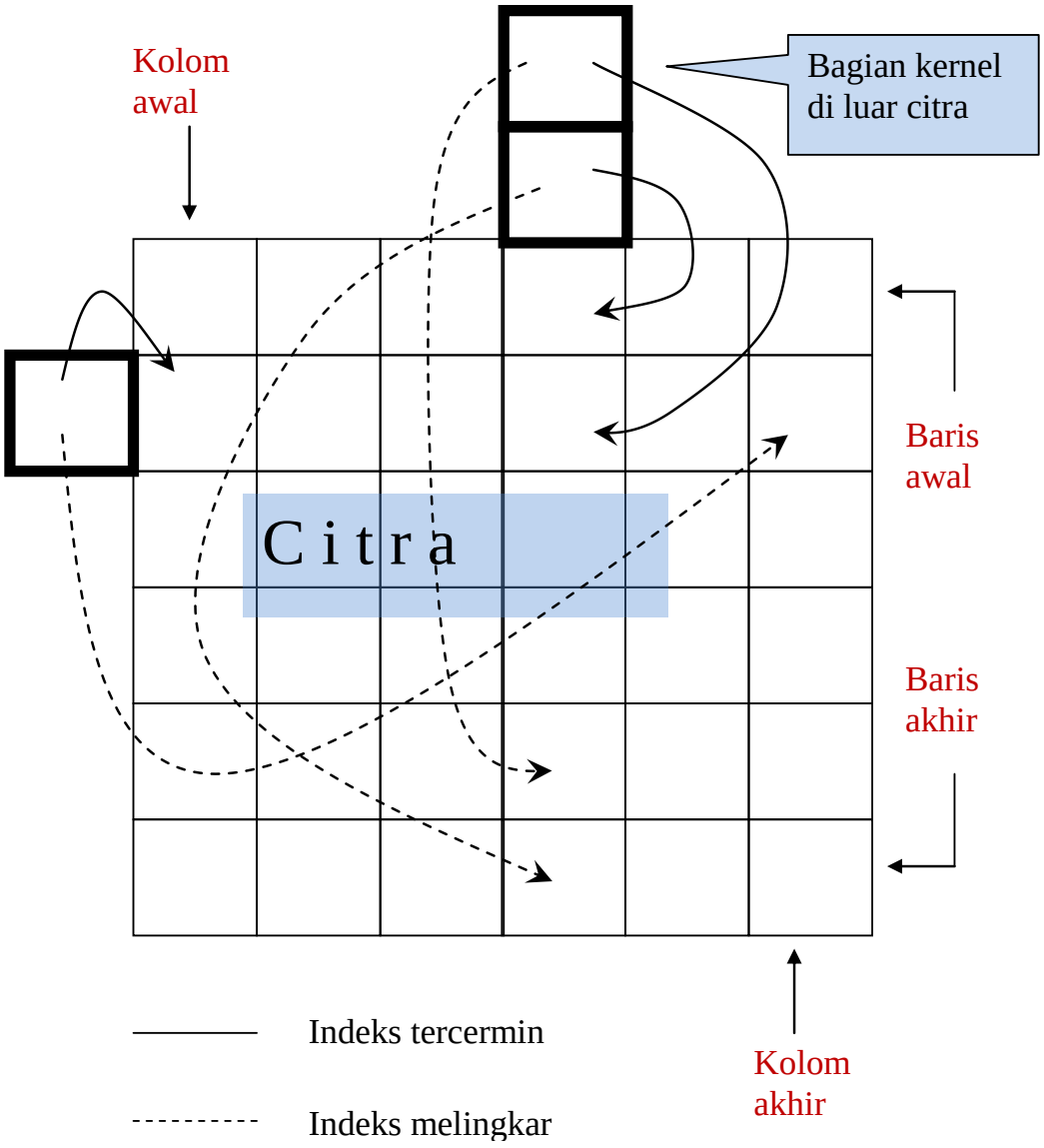
2. **Buat baris tambahan pada bagian tepi.**

   Baris dan kolom ditambahkan pada bagian tepi sehingga proses konvolusi dapat dilaksanakan. Dalam hal ini, baris dan kolom baru diisi dengan nilai 0.

3. **Ambil bagian yang tidak punya pasangan dengan bagian lain dari citra.**

   Ada beberapa pendekatan yang dapat dilakukan. Dua diantara cara-cara yang dapat digunakan dijelaskan dalam Gambar 4.12. Indeks melingkar dilaksanakan dengan mengambil data pada posisi di seberang citra, sedangkan indeks tercermin diambilkan dari baris/kolom yang ada di dekatnya. Dua cara yang lain yang diilustrasikan pada Gambar 4.14:

   - mengisi dengan citra pada bagian tepi (baik baris tepi maupun kolom tepi);
   - melakukan penggulungan secara periodis.



Gambar 4.13 Penentuan indeks untuk mengambil data untuk posisi kernel di luar area citra

(a) Citra asli

(b) Bagian tepi diberi nilai nol

(c) Pengisian dari baris atau kolom terpinggir

(d) Pengisian dengan pencerminan

(e) Pengulangan dari tepi yang berseberangan

Gambar 4.14 Cara menangani bagian tepi citra

Algoritma berikut menunjukkan cara menggunakan indeks tercermin.

ALGORITMA 4.3 – Konvolusi pada citra memakai indeks tercermin

Masukan:
- f : Citra yang akan dikonvolusi
- h : kernel konvolusi

Keluaran:
- g(y, x) : Citra hasil konvolusi

```
1. m2 ← floor(jumlah baris kernel h)
2. n2 ← floor(jumlah lebar_kernel h)
3. FOR y ← 1 TO tinggi_citra_f
     FOR x ← 1 TO lebar_citra_f
       // Lakukan konvolusi
       jum ← 0
       FOR p ← -m2 TO m2
         FOR q ← -n2 TO n2
           // Penanganan pada x
           x2 ← x-p
           IF x2 < 1
             x2 ← -x2 + 1
           ELSE
             IF x2 > lebar_citra_f
               x2 ←2 lebar_citra_f – x2 + 1
             END-IF
           END-IF
           // Penanganan pada y
           y2 ← y-p
           IF y2 < 1
             y2 ← -y2 + 1
           ELSE
             IF y2 > tinggi_citra_f
               x2 ← 2 tinggi_citra_f – x2 + 1
             END-IF
           END-IF
           jum ← jum * h(p+m2+1, q+n2+1) * f(y2, x2)
         END-FOR
       END-FOR
       g(y, x) ← jum
     END-FOR
   END-FOR
```

Implementasi dari Algoritma 4.3 dapat dilihat pada program berikut.

Program : konvolusi2.m

```
function [G] = konvolusi2(F, H)
% KONVOLUSI2 Melakukan konvolusi kernel H dengan citra F
%      (Versi Algoritma 4.3)
%      H harus mempunyai tinggi dan lebar ganjil
%      Hasil: citra G

[tinggi_f, lebar_f] = size(F);
[tinggi_h, lebar_h] = size(H);

m2 = floor(tinggi_h/2);
n2 = floor(lebar_h/2);

F2=double(F);
for y=1 : tinggi_f
    for x=1 : lebar_f
        % Pelaksanaan konvolusi F(baris, kolom)
        jum = 0;
        for p=-m2 : m2
            for q=-n2 : n2
                % Penanganan x
                x2 = x-q;
                if x2 < 1
                    x2 = -x2 + 1;
                else
                    if x2 > lebar_f
                       x2 = 2 * lebar_f - x2 + 1;
                   end
                end

                % Penanganan y
                y2 = y-p;
                if y2 < 1
                    y2 = -y2 + 1;
                else
                    if y2 > tinggi_f
                       y2 = 2 * tinggi_f - y2 + 1;
                   end
                end


                jum = jum + H(p+m2+1,q+m2+1) * ...
                      F2(y2, x2);
            end
        end

        G(y, x) = jum;
    end
end
```

Akhir Program

Penggunaan fungsi `konvolusi2` secara prinsip sama dengan pemakaian fungsi konvolusi. Perbedaannya, `konvolusi2` menghasilkan citra berukuran sama dengan ukuran citra pada argumennya.

4.6 Mempercepat Komputasi pada Konvolusi

Komputasi pada konvolusi dapat menjadi lama jika ukuran kernel membesar. Untuk kernel dengan ukuran n x n, proses konvolusi akan dilakukan n x n kali. Kalau dinyatakan dengan ukuran Big O, prosesnya memerlukan $O(n^2)$. Untuk mempercepat komputasi, perlu dicari solusi yang proses komputasinya kurang dari $O(n^2)$. Hal ini dapat dilakukan dengan memecah kernel yang berupa matriks menjadi dua buah vektor.

Misalnya, h adalah matriks kernel. Untuk kondisi tertentu, h dapat dipecah menjadi dua buah vektor seperti berikut:

$$\boldsymbol{h} = \boldsymbol{h_k}\ x\ \boldsymbol{h_b}$$

Dalam hal ini, $\mathbf{h_k}$ adalah vektor kolom dan $\mathbf{h_b}$ adalah vektor baris. Contoh:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ -1 \end{bmatrix} x \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

Nah, melalui vektor $\mathbf{h_b}$ dan $\mathbf{h_k}$ inilah konvolusi terhadap citra dilakukan. Dalam hal ini, kedua vektor dijadikan sebagai vektor mendatar.

Catatan

Suatu kernel dapat diperiksa dengan mudah untuk menentukan dapat tidaknya matriks diubah ke bentuk perkalian dua vektor. Hal ini bisa dilakukan dengan menggunakan fungsi **rank**. Hasil fungsi ini berupa 1 kalau matriks dapat didekomposisi menjadi dua buah vektor. Contoh:

```
>> H = [-1 0 1; -2 0 2; -1 0 1]; ↵
>> rank(H) ↵
ans =

     1
>>
```

Hasil 1 di atas menyatakan bahwa H dapat didekomposisi menjadi perkalian dua vektor.

Catatan

Suatu kernel yang mempunyai *rank* dengan nilai 1 dapat didekomposisi menjadi dua vektor dengan menggunakan fungsi **svd**. Misal, H adalah matriks kernel, maka perintah seperti berikut dapat diberikan:

```
>> [U,S,V]=svd(H); ↵
>> hkol = U(:,1) * sqrt(S(1)) ↵
>> hbrs = conj(V(:,1)) * sqrt(S(1)); ↵
```

Nah, berdasarkan hasil `hkol` dan `hbrs`, dapat dicoba perkalian seperti berikut:

```
>> hkol * hbrs' ↵
```

Tanda ‘ berarti tranpos. Hasilnya akan berupa matriks kernel.

Bagaimana konvolusi dilakukan melalui kedua vektor hasil dekomposisi kernel? Algoritma berikut menjelaskannya.

ALGORITMA 4.4 – Konvolusi pada citra menggunakan vektor

Masukan:
- f : Citra
- hy: Kernel baris
- hx: Kernel kolom
- hx dan hy ditulis dengan bentuk vektor mendatar dan berukuran sama

Keluaran:
- g : Citra hasil konvolusi

```
1. m2 ← floor(jumlah_kolom_kernel_hx)
2. t ← f
3. // Proses y
4. FOR y ← m2+1 TO tinggi_citra_f – m2
       FOR x ← 1 TO lebar_citra_f
       // Lakukan konvolusi dengan hy
       jum ← 0;
       FOR p ← -m2 TO m2
           jum ← jum * hy(p+m2+1) * f(y-p, x)
       END-FOR
       t(y, x) ← jum
   END-FOR
5. // Proses x
   FOR y ← 1 TO tinggi_citra_f
       FOR x ← m2+1 TO lebar_citra_f –m2
       // Lakukan konvolusi dengan hx
       jum ← 0;
       FOR p ← -m2 TO m2
           jum ← jum * hy(p+m2+1) * t(y, x-p)
       END-FOR
       g(y, x) ← jum
   END-FOR
```

Implementasi algoritma di atas ditunjukkan pada program berikut.

Program : konvolusi3.m

```
function [G] = konvolusi3(F, Hkol, Hbrs)
% KONVOLUSI3 Melakukan konvolusi kernel Hkol dan Hbrs dengan citra
F
%      (Versi Algoritma 4.4)
%      Hkol dan Hbrs harus mempunyai tinggi dan lebar ganjil
%      dan ukurannnya sama
%      Hkol dan Hbrs berupa vektor mendatar
%      Hasil: citra G

[tinggi_f, lebar_f] = size(F);
[tinggi_h, lebar_h] = size(Hbrs);

m2 = floor(lebar_h/2);

F2=double(F);
T = F2;
for y=m2+1 : tinggi_f-m2
    for x=1 : lebar_f
        jum = 0;
        for p=-m2 : m2
            jum = jum + Hkol(p+m2+1) * F2(y-p, x);
        end

        T(y, x) = jum;
    end
end

for y=1 : tinggi_f
    for x=m2+1 : lebar_f-m2
        jum = 0;
        for p=-m2 : m2
            jum = jum + Hbrs(p+m2+1) * T(y, x-p);
        end

        G(y, x) = jum;
    end
end
```

Akhir Program

Contoh berikut menunjukkan program yang menggunakan konvolusi3.m.

Program : teskonv.m

```
% TESKONV Menguji fungsi konvolusi3
```

```
Img = imread('C:\Image\gedung.png');
Hkol = [-1 -2 -1];
Hbrs = [1 0 -1];
K = konvolusi3(Img, Hkol, Hbrs);
K2 = uint8(K);
imshow(K2);
```

Akhir Program

Gambar 4.15 memperlihatkan hasil pemrosesan gedung.png menggunakan konvolusi.m dan konvolusi3.m. Pemrosesan dengan kedua algoritma tersebut memberikan hasil visual yang sedikit berbeda.

(a) Hasil pemrosesan dengan konvolusi.m

(b) Hasil pemrosesan dengan konvolusi3.m

Gambar 4.15 Contoh penerapan konvolusi menggunakan matriks dan vektor

Sebagai perbandingan, Tabel 4.1 menunjukkan waktu yang diperlukan untuk melakukan konvolusi dengan menggunakan konvolusi2.m, konvolusi3.m, dan **conv2** (milik *MATLAB*) dalam satuan detik.

Tabel 4.1 Perbandingan waktu komputasi konvolusi untuk berbagai ukuran kernel

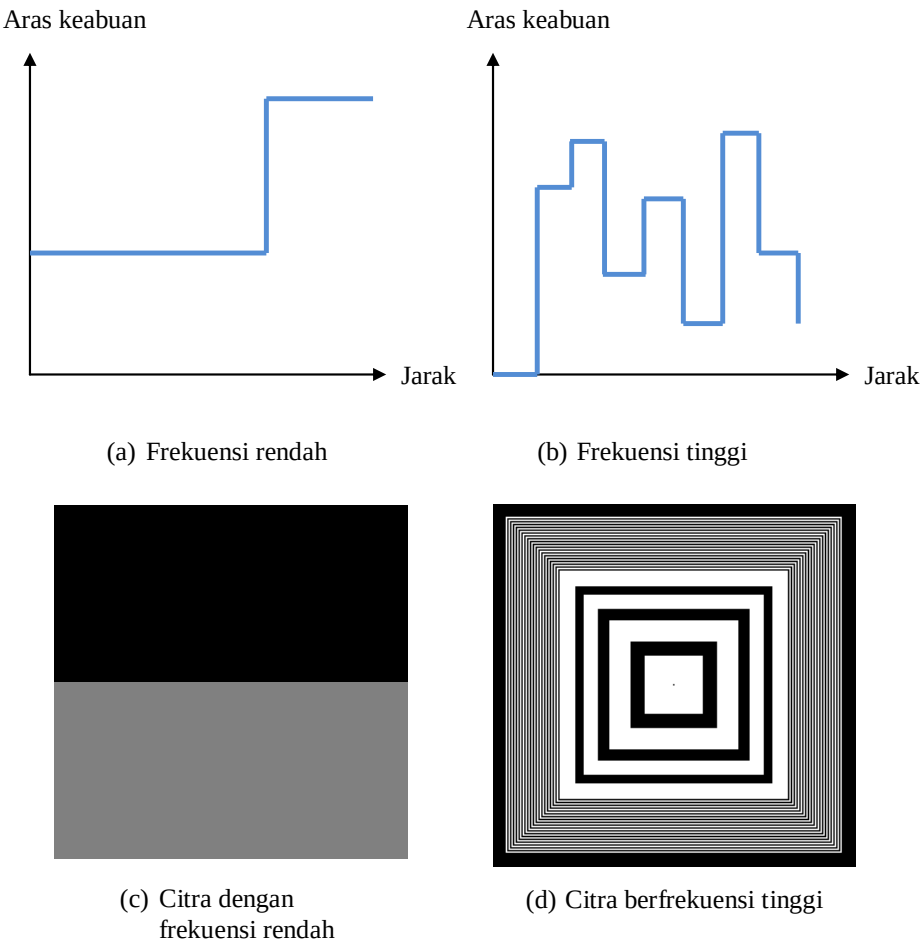
Fungsi	3x3	5x5	7x7	9x9	11x11	13x13
konvolusi2.m	3,74	4,20	4,86	5,56	6,37	7,71
konvolusi3.m	3.53	3,61	3,62	3,62	3,62	3,71
conv2.m	0.02	0,03	0,04	0,18	0,08	0,10

Dapat dilihat bahwa konvolusi3.m (yang menggunakan vektor) lebih cepat daripada konvolusi2.m (yang menggunakan matriks). Namun, dibandingkan dengan fungsi **conv2** yang tersedia dalam *MATLAB*, kecepatan kedua konvolusi yang dibuat sendiri jauh lebih rendah.

4.7 Pengertian Frekuensi

Istilah frekuensi berkonotasi punya kaitan dengan waktu. Sebagai contoh, isyarat listrik AC pada sistem kelistrikan di Indonesia mempunyai frekuensi sebesar 50 Hz. Makna 50 Hz di sini menyatakan bahwa terdapat 50 siklus sinus yang utuh pada setiap detik. Pada citra, istilah frekuensi tidak berhubungan dengan waktu, melainkan berkaitan dengan keruangan atau spasial. Oleh karena itu, citra dikatakan memiliki frekuensi spasial. Definisi di Wikipedia menyatakan bahwa frekuensi spasial adalah karakteristik sebarang struktur yang bersifat periodis sepanjang posisi dalam ruang. Frekuensi spasial adalah ukuran seberapa sering struktur muncul berulang dalam satu satuan jarak.

Frekuensi spasial pada citra menunjukkan seberapa sering suatu perubahan aras keabuan terjadi dari suatu posisi ke posisi berikutnya. Gambar 4.16 menunjukkan secara visual perbedaan antara frekuensi rendah dan frekuensi tinggi. Pada citra berfrekuensi tinggi, perubahan aras sering terjadi seiring dengan pergeseran jarak.



(a) Frekuensi rendah (b) Frekuensi tinggi

(c) Citra dengan frekuensi rendah (d) Citra berfrekuensi tinggi

Gambar 4.16 Perbedaan frekuensi rendah dan frekuensi tinggi pada citra

Pada Gambar 4.16(a), perubahan aras keabuan terjadi sekali saja, sedangkan pada Gambar 4.16(b) terlihat bahwa perubahan aras keabuan sering terjadi. Itulah sebabnya, Gambar 4.16(a) menyatakan contoh frekuensi rendah dan Gambar 4.16(b) menunjukkan contoh frekuensi tinggi.

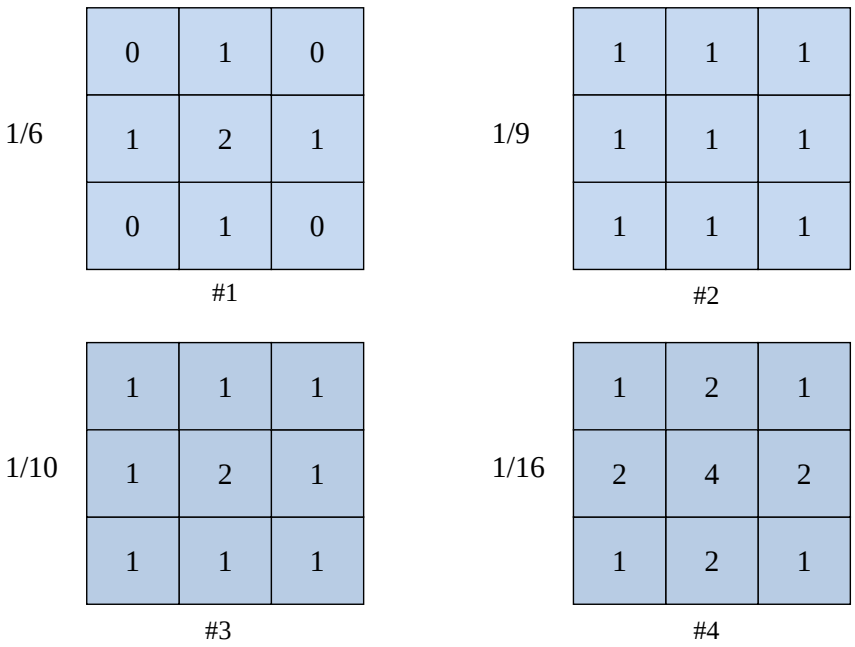
Pengertian frekuensi dalam citra perlu dipahami terlebih dulu. Pada beberapa pembicaraan di belakang, istilah frekuensi akan sering disebut.

4.8 Filter Lolos-Rendah

Filter lolos-bawah (*low-pass filter*) adalah filter yang mempunyai sifat dapat meloloskan yang berfrekuensi rendah dan menghilangkan yang berfrekuensi tinggi. Efek filter ini membuat perubahan aras keabuan menjadi lebih lembut.

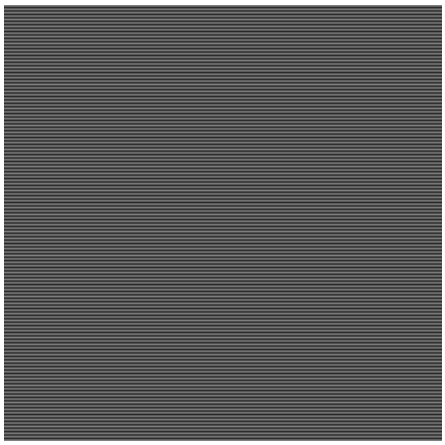
Filter ini berguna untuk menghaluskan derau atau untuk kepentingan interpolasi tepi objek dalam citra.

Operasi penapisan lolos-bawah dilaksanakan melalui konvolusi atau tanpa konvolusi. Contoh yang tidak memakai konvolusi dapat dilihat pada filter median (filter median termasuk dalam filter lolos-bawah). Adapun yang melibatkan konvolusi menggunakan kernel antara lain berupa seperti yang terlihat pada Gambar 4.17 (Phillips, 2000).



Gambar 4.17 Contoh kernel untuk filter lolos-bawah

Sebagai contoh, terdapat citra berfrekuensi rendah dan berfrekuensi tinggi dengan komposisi data seperti berikut.



40	40	40	40	40
40	40	40	40	40
40	40	40	40	40
40	40	40	40	40
40	40	40	40	40
40	40	40	40	40
40	40	40	40	40
40	40	40	40	40
40	40	40	40	40
128	128	128	128	128
128	128	128	128	128
128	128	128	128	128
128	128	128	128	128
128	128	128	128	128
128	128	128	128	128
128	128	128	128	128
128	128	128	128	128
128	128	128	128	128
128	128	128	128	128
128	128	128	128	128

(a) Citra dengan frekuensi rendah

128	128	128	128	128
40	40	40	40	40
128	128	128	128	128
40	40	40	40	40
128	128	128	128	128
40	40	40	40	40
128	128	128	128	128
40	40	40	40	40
128	128	128	128	128
40	40	40	40	40
128	128	128	128	128
40	40	40	40	40
128	128	128	128	128
40	40	40	40	40
128	128	128	128	128
40	40	40	40	40
128	128	128	128	128
40	40	40	40	40
128	128	128	128	128
40	40	40	40	40

(b) Citra dengan frekuensi tinggi

(c) Citra dengan frekuensi rendah

(d) Citra dengan frekuensi tinggi

Gambar 4.18 Nilai-nilai intensitas/kecerahan citra dengan frekuensi rendah dan frekuensi tinggi pada arah vertikal

Dengan menggunakan kernel

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

terhadap kedua citra tersebut dan menggunakan konvolusi.m maka didapatkan hasil seperti yang terdapat pada Gambar 4.19.

40	40	40
40	40	40
40	40	40
40	40	40
40	40	40
40	40	40
40	40	40
69	69	69
99	99	99
128	128	128
128	128	128
128	128	128
128	128	128
128	128	128
128	128	128
128	128	128
128	128	128
128	128	128

(a) Citra dengan frekuensi rendah

99	99	99
69	69	69
99	99	99
69	69	69
99	99	99
69	69	69
99	99	99
69	69	69
99	99	99
69	69	69
99	99	99
69	69	69
99	99	99
69	69	69
99	99	99
69	69	69
99	99	99
69	69	69

(b) Citra dengan frekuensi tinggi

Gambar 4.19 Hasil penapisan dengan filter lolos-rendah

Perhatikan Gambar 4.19 (a). Secara prinsip, filter tidak membuat perubahan yang sangat berarti pada citra kecuali perubahan pada baris yang berisi 69 dan 99. Adapun pada Gambar 4.19(b), frekuensi memang tidak berubah, tetapi terjadi penghalusan perubahan aras (128 menjadi 99 dan 40 menjadi 69). Apa pengaruhnya secara visual pada citra? Melalui filter lolos-rendah, hal-hal yang menyatakan frekuensi tinggi akan diredupkan, sedangkan bagian berfrekuensi rendah hampir tidak berubah.

Program berikut dapat dipakai untuk mengamati efek filter lolos rendah terhadap citra.

Program : tapis.m

```
function [G] = tapis(berkas, H)
% TAPIS Menerapkan filter H dengan citra F
%     H harus mempunyai tinggi dan lebar ganjil
%     Hasil: citra G

F = imread(berkas);
K = konvolusi(F, H);
G = uint8(K);

figure(1); imshow(F);
figure(2); imshow(G);
```

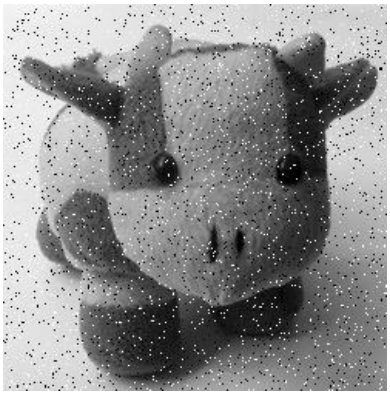
Akhir Program

Contoh penguji program di atas:

```
>> H = [1 1 1; 1 1 1; 1 1 1] / 9;
>> tapis('C:\Image\mobil.png', H);
```

Gambar asal dan hasil penapisan akan ditampilkan pada jendela yang terpisah.

Gambar 4.20 menunjukkan hasil penapisan dengan filter #2 pada Gambar 4.17. Contoh tersebut menunjukkan bahwa filter lolos-rendah mampu menghaluskan perubahan-perubahan yang drastis. Perhatikan ketajaman genting pada *goldhill* menjadi diperhalus setelah melalui penapisan. Begitu pula derau pada boneka.



(a) Citra boneka yang dilengkapi derau

(b) Hasil penapisan citra boneka

(c) Citra goldhill

(b) Hasil penapisan citra goldhill

Gambar 4.20 Contoh penerapan konvolusi menggunakan kernel #2

Gambar 4.21 menunjukkan hasil penggunaan kernel #1, #2, #3, dan #4 untuk menapis citra boneka yang telah diberi derau. Adapun Gambar 4.22 memperlihatkan contoh penerapan kernel berukuran 3x3, 5x3, dan 7x7 dengan nilai koefisien pada kernel bernilai sama.

(a) Citra boneka yang dilengkapi derau

(b) Hasil dengan kernel #1

(c) Hasil dengan kernel #2

(b) Hasil dengan kernel #4

Gambar 4.21 Efek pemakaian tiga macam filter lolos-rendah pada boneka

(a) Citra lena256

(b) Hasil dengan kernel 3x3

(c) Hasil dengan kernel 5x5

(b) Hasil dengan kernel 13x13

Gambar 4.22 Efek pemakaian filter lolos-rendah dengan berbagai ukuran kernel. Semua bobot bernilai sama

Catatan

- Efek pengaburan citra dapat ditingkatkan dengan menaikkan ukuran kernel.
- Rahasia kernel yang digunakan untuk keperluan mengaburkan citra seperti berikut.

1. Tinggi dan lebar kernel ganjil.
2. Bobot dalam kernel bersifat simetris terhadap piksel pusat.
3. Semua bobot bernilai positif.
4. Jumlah keseluruhan bobot sebesar satu.

4.9 Filter Lolos-Tinggi

Filter lolos-tinggi adalah filter yang ditujukan untuk melewatkan frekuensi tinggi dan menghalangi yang berfrekuensi rendah. Hal ini biasa dipakai untuk mendapatkan tepi objek dalam citra atau menajamkan citra. Contoh filter lolos-tinggi dapat dilihat pada Gambar 4.23.

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

#1

-1	-1	-1
-1	8	-1
-1	-1	-1

#2

1	-2	1
-2	4	-2
1	-2	1

#3

Gambar 4.23 Contoh tiga kernel filter lolos-tinggi

Filter lolos-tinggi mempunyai sifat yaitu jumlah seluruh koefisien adalah nol. Selain itu terdapat sifat sebagai berikut (Efford, 2000).

1. Apabila dikenakan pada area dengan perubahan aras keabuan yang lambat (frekuensi rendah), hasil berupa nol atau nilai yang sangat kecil.
2. Apabila dikenakan pada area yang perubahan aras keabuannya cepat (frekuensi tinggi), hasil konvolusi bernilai sangat besar.

Jika kernel seperti berikut

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

dikenakan pada data dalam Gambar 4.18, akan diperoleh hasil seperti berikut.

```
  0    0    0
  0    0    0
  0    0    0
  0    0    0
  0    0    0
  0    0    0
  0    0    0
  0    0    0
 88   88   88
  0    0    0
  0    0    0
  0    0    0
  0    0    0
  0    0    0
  0    0    0
  0    0    0
  0    0    0
  0    0    0
```

(a) Citra dengan frekuensi rendah

```
  0    0    0
176  176  176
  0    0    0
176  176  176
  0    0    0
176  176  176
  0    0    0
176  176  176
  0    0    0
176  176  176
  0    0    0
176  176  176
  0    0    0
176  176  176
  0    0    0
176  176  176
  0    0    0
176  176  176
```

(b) Citra dengan frekuensi tinggi

Gambar 4.24 Hasil penapisan dengan filter lolos-tinggi

Hasil pada Gambar 4.24(a) menunjukkan bahwa hanya pada perbatasan antara perubahan aras keabuan yang ditonjolkan (baris berisi 88) dan nilai yang lain bernilai rendah (nol). Dengan demikian, akan muncul garis putih. Hasil pada Gambar 4.24(b) menunjukkan bahwa citra yang berfrekuensi tinggi hampir tidak mengalami perubahan, kecuali nilainya saja yang berefek pada penajaman perbedaan aras keabuan (nilai 150 menjadi 176 dan nilai 40 menjadi 0).

Gambar 4.25 menunjukkan penggunaan filter lolos-tinggi yang terdapat pada Gambar 4.23 terhadap citra boneka.png. Adapun Gambar 4.26 memperlihatkan hasil pemrosesan pada citra bulat.png.

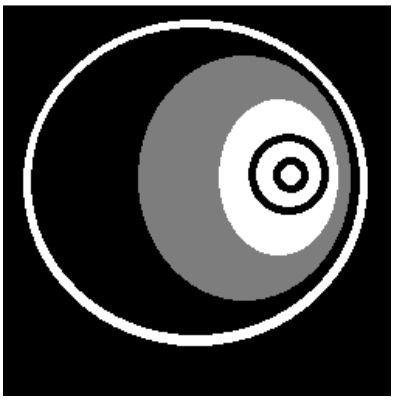
(a) Citra boneka.png

(b) Hasil dengan kernel #1

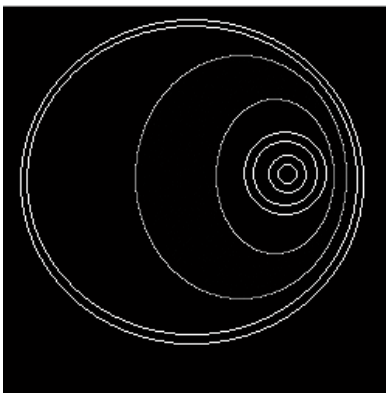
(c) Hasil dengan kernel #2

(b) Hasil dengan kernel #3

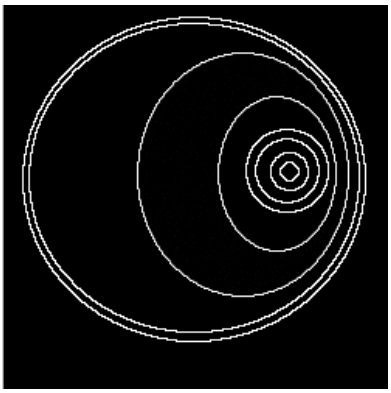
Gambar 4.25 Hasil pemrosesan dengan filter lolos-tinggi pada citra boneka



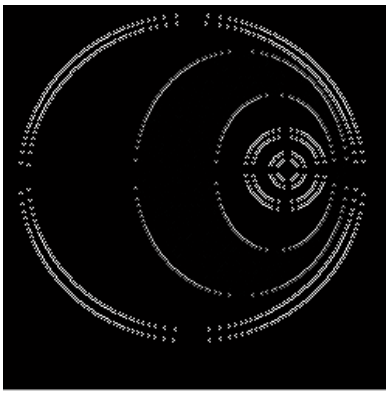
(a) Citra bulat.png



(b) Hasil dengan kernel #1



(c) Hasil dengan kernel #2



(b) Hasil dengan kernel #3

Gambar 4.26 Hasil pemrosesan dengan filter lolos-tinggi pada citra bulat

Catatan

Rahasia kernel yang digunakan untuk keperluan mendeteksi tepi seperti berikut (Oliver, dkk., 1993).

1. Tinggi dan lebar kernel ganjil.
2. Bobot dalam kernel bersifat simetris terhadap piksel pusat.
3. Bobot pusat kernel bernilai positif.
4. Bobot tetangga pusat kernel bernilai negatif (dapat menggunakan 4-ketetanggan atau 8 ketetanggaan).
5. Jumlah keseluruhan bobot sebesar satu.

4.10 Filter High-Boost

Filter "high boost" (Efford, 2000) dapat digunakan untuk menajamkan citra melalui konvolusi. Kernel yang dapat dipakai adalah kernel filter lolos-tinggi dengan nilai di pusat diisi dengan nilai yang lebih besar daripada nilai pada posisi tersebut untuk filter lolos-tinggi. Sebagai contoh, dapat digunakan kernel seperti berikut.

-1	-1	-1
-1	c	-1
-1	-1	-1

c > 8; misalnya 9

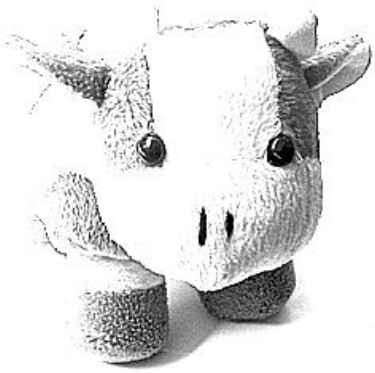
Gambar 4.27 Contoh filter *high boost*

Gambar berikut menunjukkan efek saat c diisi dengan 9, 10, dan 11.

(a) Citra boneka

(b) Hasil untuk c=9

(c) Hasil untuk c=10



(d) Hasil untuk c=11

Gambar 4.28 Hasil pemrosesan dengan filter *high boost*

Tampak bahwa dengan menggunakan filter *high boost* bernilai tengah tertentu (pada contoh di atas berupa 9), diperoleh hasil berupa penajaman citra.

Catatan

Rahasia kernel yang digunakan untuk keperluan menajamkan citra seperti berikut.

1. Tinggi dan lebar kernel gasal.
2. Bobot dalam kernel bersifat simetris terhadap piksel pusat.
3. Bobot pusat kernel bernilai positif.
4. Bobot di sekeliling pusat kernel bernilai negatif (dapat menggunakan 4-ketetanggaan atau 8 ketetanggaan).
5. Jumlah keseluruhan bobot lebih besar satu.
6. Bobot terbesar terletak di pusat kernel.

4.11 Efek *Emboss*

Gambar 4.29 menunjukkan contoh hasil *embossing*. Terlihat ada penebalan garis pada arah tertentu.

(a) Berdasar citra boneka2

(b) Berdasar citra lena256

Gambar 4.29 Efek *emboss*

Kernel yang digunakan seperti berikut:

-2	0	0
0	0	0
0	0	2

Nilai negatif dan positif yang berpasangan menentukan perubahan kecerahan yang berefek pada penggambaran garis gelap atau terang, Gambar 4.30 memperlihatkan efek beberapa kernel dan hasil yang didapatkan untuk citra lena256.png.

-1	0	0
0	0	0
0	0	1

(a) Kernel #1

(b) Hasil untuk kernel #1

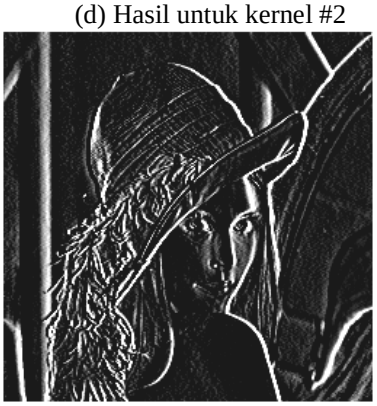
1	0	0
0	0	0
0	0	-1

(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 4.30 Efek emboss untuk berbagai kernel

Rahasia pembuatan *emboss* terletak pada kernel konvolusi dengan sifat seperti berikut (Oliver, dkk., 1993).

1. Tinggi dan lebar kernel gasal.
2. Bobot dalam kernel bersifat tidak simetris terhadap piksel pusat.
3. Bobot pusat kernel bernilai nol.
4. Jumlah keseluruhan bobot bernilai nol.

Nilai negatif pada kernel *emboss* menentukan arah penebalan garis. Beberapa contoh yang dapat dicoba ditunjukkan pada gambar berikut.

-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 4.31 Berbagai kernel untuk *embossing*

4.12 Pengklasifikasian Filter Linear dan Nonlinear

Filter disebut sebagai filter linear jika dalam melakukan penapisan melibatkan piksel dengan cara linear. Contoh filter linear yaitu filter pererataan. Filter-filter linear yang lain:

- filter *Gaussian*
- filter topi *Mexico* (*Laplacian*)

Kelemahan filter linear, terutama ketika dipakai untuk konvolusi citra atau penghilangan derau, yaitu membuat struktur citra yang meliputi titik, tepi, dan garis ikut terkaburkan dan kualitas citra keseluruhan menurun (Burger dan Burge, 2008). Kelemahan seperti ini dapat diatasi menggunakan filter nonlinear.

Filter nonlinear adalah filter yang bekerja tidak memakai fungsi linear. Filter batas dan filter median merupakan contoh filter nonlinear.

4.13 Filter *Gaussian*

Filter *Gaussian* tergolong sebagai filter lolos-rendah yang didasarkan pada fungsi *Gaussian*. Model dua dimensinya berupa:

$$G(y.x) = e^{-\frac{x^2+y^2}{2\sigma^2}} \qquad (4.6)$$

Dalam hal ini, σ adalah deviasi standar dan piksel pada pusat (y, x) mendapatkan bobot terbesar berupa 1.

Filter *Gaussian* paling tidak berukuran 5x5. Sebagai contoh, bobot-bobotnya dapat diperoleh dengan membuat σ^2 bernilai 1. Dengan demikian:

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

$G(1,0) = G(0,1) = G(-1,0) = G(0,-1) = e^{-1/2} = 0{,}6065$

$G(1,1) = G(1,-1) = G(-1,1) = G(-1,-1) = e^{-1} = 0{,}3679$

$G(2,1) = G(1,2) = G(-2,1) = G(-2,-1) = e^{-5/2} = 0{,}0821$

$G(2,0) = G(0,2) = G(0,-2) = G(-2,0) = e^{-2} = 0.1353$

$G(2,2) = G(-2,-2) = G(-2,2) = G(2,-2) = e^{-4} = 0{,}0183$

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). Dengan demikian, diperoleh hasil seperti berikut, yang diperoleh dengan mengalikan nilai G(x,y) di depan dengan 55.

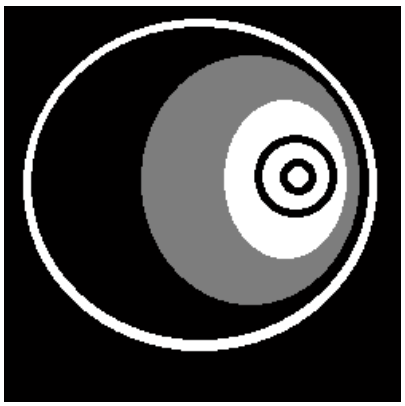
1	5	7	5	1
5	20	33	20	5
7	33	55	33	7
5	20	33	20	5
1	5	7	5	1

Setelah dinormalisasi diperoleh filter seperti berikut:

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 4.32 memberikan contoh penerapan filter *Gaussian* pada dua buah citra.



(a) Citra bulat.png

(b) Hasil konvolusi bulat.png

(c) Citra boneka.png

(d) Hasil konvolusi boneka.png

Gambar 4.32 Efek filter *Gaussian*

Hasilnya, terjadi sedikit penghalusan pada daerah yang intensitasnya berbeda jauh.

Latihan

1. Apa maksud 4-ketetanggan dan 8-ketetanggan?
2. Menurut pengamatan Anda, apa yang membedakan pemrosesan berikut kalau dilihat hasilnya secara visual?

   (a) Filter batas
   (b) Filter pererataan
   (c) Filter median

3. Jelaskan bahwa konvolusi dengan cadar

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

sesungguhnya sama dengan penggunaan filter pererataan.

4. Bagaimana bentuk kernel yang berguna untuk filter pererataan yang berukuran 5x5, 7 x 7, dan 9x9?
5. Jelaskan pengertian konvolusi.
6. Apa kegunaan konvolusi yang memecah kernel h menjadi dua buah vektor?
7. Terdapat kernel seperti berikut.

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Jika dikenakan pada citra yang berisi data seperti berikut, berapa hasil pada posisi yang diarsir abu-abu?

10	20	10
20	10	20
10	20	10

8. Bagaimana caranya agar citra menjadi kabur?

9. Bagaimana caranya kalau yang ingin didapatkan adalah tepi objek?
10. Apa yang dimaksud dengan frekuensi spasial?
11. Apa kegunaan filter lolos-rendah?
12. Bagaimana halnya dengan filter lolos-tinggi.
13. Berapa nilai c pada kernel berikut agar dapat bertindak sebagai filter *high boost*?

1	-2	1
-2	c	-2
1	-2	1

14. Cobalah untuk menguji tiga kernel yang digunakan dalam filter lolos-tinggi terhadap sejumlah gambar.
15. Jelaskan pengertian filter linear dan nonlinear. Berikan contoh masing-masing.
16. Dengan menggunakan pendekatan σ^2 bernilai 1, buatlah filter i berukuran 7x7.