

Vision – IF6083

Tugas Homework 2 (HW2)

Reza Budiawan

33221040



Program Studi S3 Teknik Elektro & Informatika

INSTITUT TEKNOLOGI BANDUNG

Desember 2022

Indeks contents

Indeks contents.....	2
Link Terkait.....	3
Deskripsi Umum Tugas 2.....	3
2.1: Create your box filter	3
2.2: Write a convolution function	5
2.2 Make some more filters and try them out!	7
2.3: Implement a Gaussian kernel	9
2.4: Hybrid images	10
2.5: Sobel filters	11
2.5.1: Make the filter	12
2.5.2: One more normalization... ..	12
2.5.1: Calculate gradient magnitude and direction	13
2.5.1: Make colorized representation.....	14
Hasil Skenario Test	15

Link Terkait

- Laporan Tugas
- Video Demo
- Link source code If6083ComputerVision

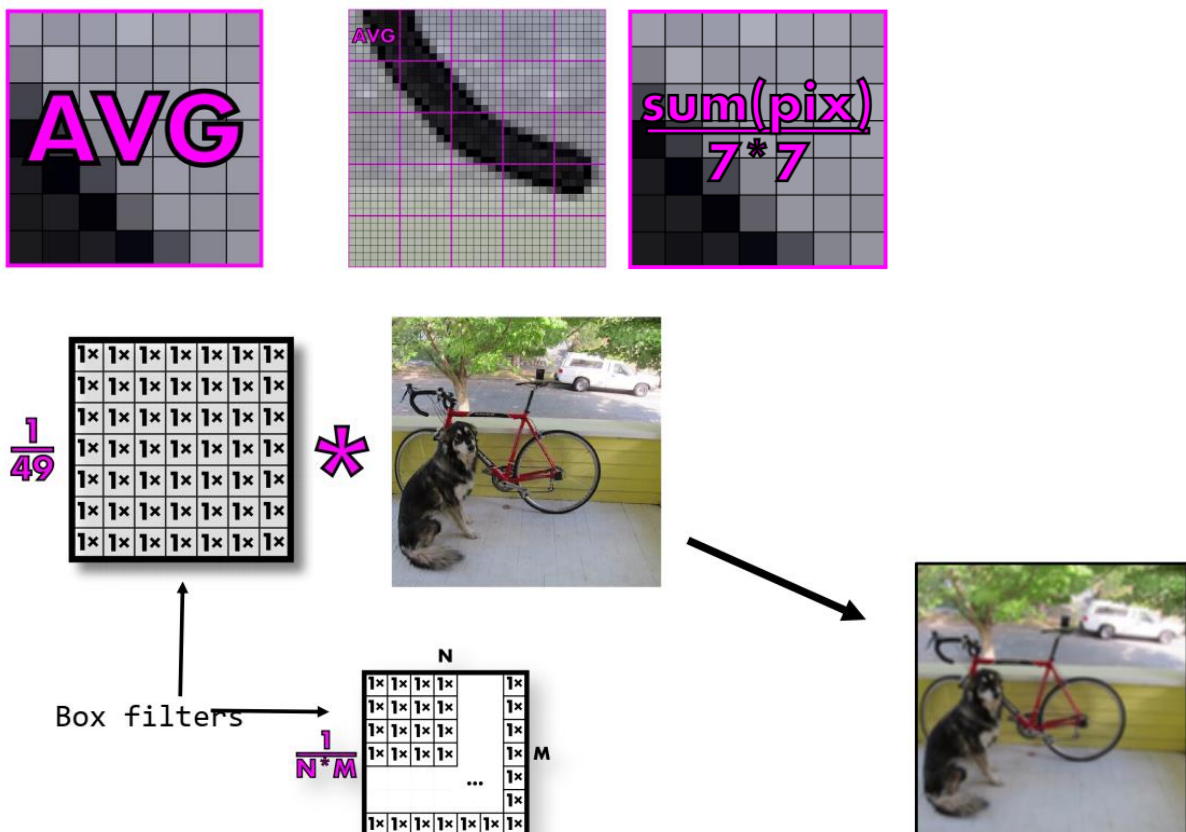
Deskripsi Umum Tugas 2

Operasi yang dilakukan merupakan operasi filter setelah melakukan resizing image. Hal ini dilakukan untuk menghasilkan citra yang smooth. Cara yang dilakukan yaitu dengan mengimplementasikan konvolusi.

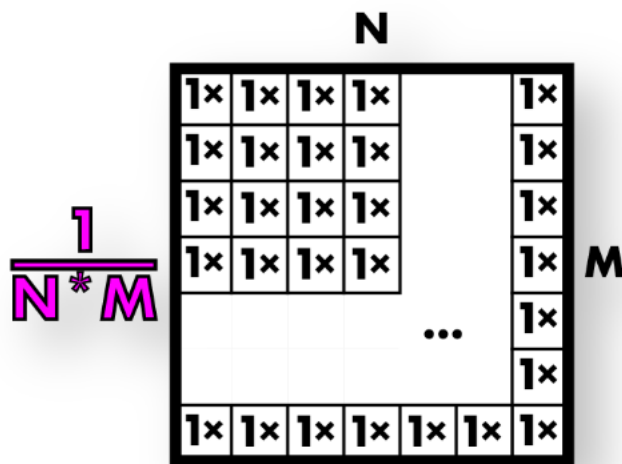
2.1: Create your box filter

Deskripsi

Pada proses shrinking, dapat dilakukan operasi averaging untuk menghasilkan citra yang lebih smooth setelah melakukan shrinking. Hal ini dilakukan dengan membuat sebuah box filter/kernel, dan melakukan operasi konvolusi terhadap citra asli yang di-shrink.



Tahap awal dari proses filter, diperlukan pembuatan box filter sebagai berikut:



Untuk membuat box filter di atas, buat sebuah image & isi setiap piksel dengan nilai 1, lalu lakukan normalisasi. Normalisasi ini dilakukan dengan membuat piksel dari citra memiliki jumlah total bernilai 1. Hal ini dituliskan pada method “l1_normalize(image im)”. Setelah mengisi method l1_normalize, isi method “make_box_filter(int w)” untuk membuat filter-nya.

Solusi

Pada method “void l1_normalize(image im)”, hal yang dilakukan adalah melakukan normalisasi nilai piksel dari citra. Pastikan total keseluruhan piksel dari citra memiliki jumlah 1. Untuk itu, terdapat 3 blok program pada method ini:

- Menghitung jumlah (sum) dari piksel citra
- Memastikan jumlah (sum) tidak bernilai 0 (citra kosong)
- Melakukan pembagian piksel citra dengan hasil penjumlahan sebelumnya (sum)

```
void l1_normalize(image im)
{
    int i;
    float sum = 0;
    for (i = 0; i < im.c * im.h * im.w; ++i){
        sum += im.data[i];
    }
    if (!sum)
        return;
    for (i = 0; i < im.c * im.h * im.w; ++i){
        im.data[i] /= sum;
    }
}
```

Berikutnya, untuk membuat box filter kodenya dituliskan pada “image make_box_filter(int w)”. Box filter ini merupakan image berukuran $w \times w$, dengan 1 channel & jumlah total pikselnya bernilai 1.

```

image make_box_filter(int w)
{
    image box_filter = make_image(w, w, 1);
    int i;
    for (i = 0; i < w * w; ++i){
        box_filter.data[i] = 1.0 / (w * w);
    }
    return box_filter;
}

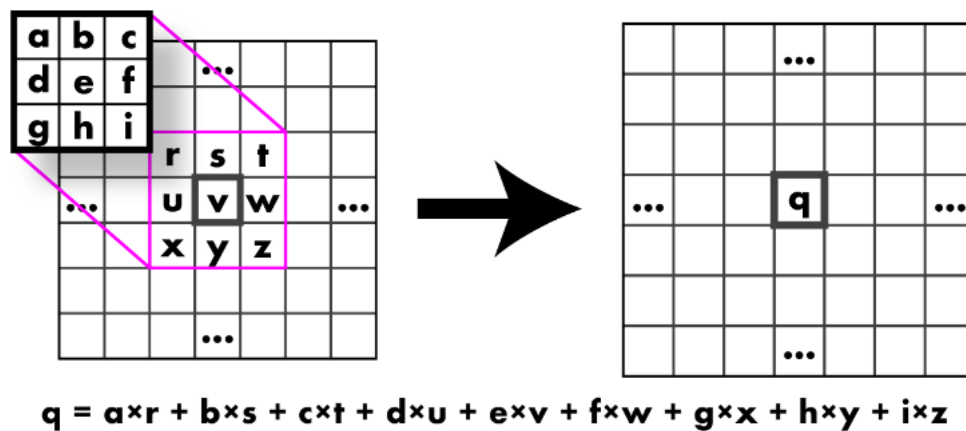
```

2.2: Write a convolution function

Deskripsi

Pada tahapan ini, isi method “convolve_image(image im, image filter, int preserve)”. Proses dari method ini yaitu menjumlahkan nilai piksel citra & filter (weighted sum). Terdapat beberapa kondisi yang harus di-handle, yaitu penyesuaian jumlah channel terhadap citra input. Filter sebaiknya memiliki jumlah channel yang sama dengan citra input “im” atau punya nilai 1 untuk channelnya. Secara kode hal ini diatur pada parameter input “preserve” (jika bernilai 1, channel sesuai jumlah citra “im”).

Operasi yang dilakukan merupakan cross-correlation (tapi disebut convolution—walaupun tanpa operasi flip). Operasi convolution tersebut diilustrasikan sebagai berikut:



Solusi

Pada body method convolve_image(image im, image filter, int preserve) dapat dituliskan beberapa langkah, yaitu memastikan image & filter tidak kosong. Berikutnya, buat sebuah citra kosong (im_out) untuk menampung citra hasil operasi konvolusi. Hal ini tergantung nilai “preserve” yang diberikan. Begitu juga proses konvolusi yang dilakukan selanjutnya.

```

image convolve_image(image im, image filter, int preserve)
{
    assert(filter.c == 1 || filter.c == im.c);

    image im_out;
}

```

```

if (preserve == 1){
    im_out = make_image(im.w, im.h, im.c);
}
else{
    im_out = make_image(im.w, im.h, 1);
}

int h_offset = filter.h / 2;
int w_offset = filter.w / 2;

for (int i = 0; i < im.c; ++i){
    for (int j = 0 - h_offset; j < im.h - h_offset; ++j){
        for (int k = 0 - w_offset; k < im.w - w_offset; ++k){

            float v;
            if (preserve == 1){
                v = 0;
            }else{
                v = get_pixel(im_out, k + w_offset, j + h_offset, 0);
            }

            for (int m = 0; m < filter.h; ++m){
                for (int n = 0; n < filter.w; ++n){
                    if (filter.c == 1){
                        v += get_pixel(im, k + n, j + m, i) *
                            get_pixel(filter, n, m, 0);
                    }else{
                        v += get_pixel(im, k + n, j + m, i) *
                            get_pixel(filter, n, m, i);
                    }
                }
            }

            if (preserve == 1){
                set_pixel(im_out, k + w_offset, j + h_offset, i, v);
            }else{
                set_pixel(im_out, k + w_offset, j + h_offset, 0, v);
            }
        }
    }
}
return im_out;
}

```

Hasil

Tahap ini memastikan proses resize menghasilkan image output yang lebih baik. Hal ini dapat dilihat dengan menjalankan script “tryhw2.py”.

```

from uwing import *
im = load_image("data/dog.jpg")
f = make_box_filter(7)
blur = convolve_image(im, f, 1)
save_image(blur, "dog-box7")

```

Berikut adalah gambar yang dihasilkan dari operasi tersebut:



Pada hasil di samping terlihat hasil resize & blur yang dilakukan dengan averaging (proses konvolusi) menghasilkan citra luaran yang diharapkan.

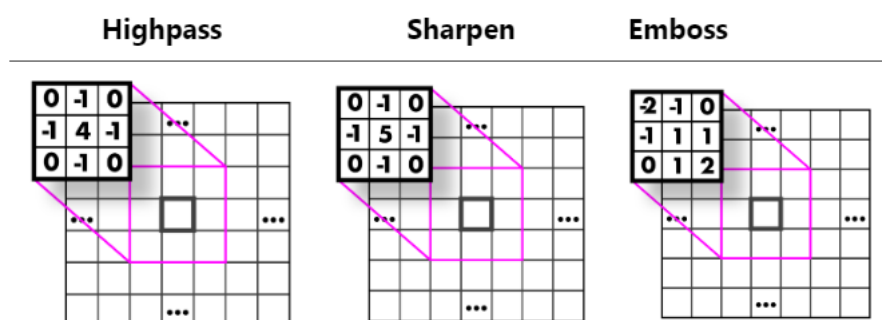
2.2 Make some more filters and try them out!

Deskripsi

Implementasikan ketiga method berikut:

- image make_highpass_filter(),
- image make_sharpen_filter(),
- image make_emboss_filter()

dengan membuat ketiga filter berikut:



Selain itu jawablah ketiga pertanyaan berikut:

1. Question 2.2.1: Which of these filters should we use preserve when we run our convolution and which ones should we not? Why?
2. Question 2.2.2: Do we have to do any post-processing for the above filters? Which ones and why?

Solusi

Ketiga filter yang dikembalikan merupakan array 9 elemen dengan nilai yang sudah ditentukan. Berikut kode yang dapat dituliskan untuk membuat masing-masing filter.

```
image make_highpass_filter()
{
    image highpass_filter = make_image(3, 3, 1);
    float weight[9] = {0, -1, 0,
                       -1, 4, -1,
                       0, -1, 0};
    memcpy(highpass_filter.data, weight, sizeof(weight));
    return highpass_filter;
}

image make_sharpen_filter()
{
    image sharpen_filter = make_image(3, 3, 1);
    float weight[9] = {0, -1, 0,
                       -1, 5, -1,
                       0, -1, 0};
    memcpy(sharpen_filter.data, weight, sizeof(weight));
    return sharpen_filter;
}

image make_emboss_filter()
{
    image emboss_filter = make_image(3, 3, 1);
    float weight[9] = {-2, -1, 0,
                       -1, 1, 1,
                       0, 1, 2};
    memcpy(emboss_filter.data, weight, sizeof(weight));
    return emboss_filter;
}
```

Jawaban pertanyaan:

1. Pertanyaan 2.2.1: Filter apa yang harus menggunakan nilai 1 untuk atribut preserve saat menjalankan operasi konvolusi (convolve_image)?

Jawaban 2.2.1: Filter yang harus menggunakan nilai 1 untuk atribut preserve adalah filter yang menghasilkan channel sesuai dengan image yang dilakukan operasi konvolusi. Jika menghasilkan 1 channel saja, maka atribut preserve tidak diperlukan (bernilai 0). Dikarenakan box filter berfungsi untuk melakukan smoothing, preserve diperlukan. Sharpen filter & emboss filter juga memerlukan hal ini. Sedangkan pada highpass filter, atribut ini tidak diperlukan (bernilai 0).

2. Pertanyaan 2.2.2: Apakah diperlukan post-processing untuk filter di atas (highpass, sharpen, emboss).

Jawaban 2.2.2: Post-processing diperlukan untuk filter tersebut. Hal ini dilakukan untuk memastikan nilai hasil konvolusi bernilai 0-1. Untuk itu diperlukan pemrosesan lanjut (post-processing) seperti `feature_normalize`.

2.3: Implement a Gaussian kernel

Deskripsi

Isilah method “`image make_gaussian_filter(float sigma)`” yang menghasilkan filter untuk proses smoothing menggunakan gaussian dari nilai standar deviasi & input parameter “sigma”. Kernel memiliki ukuran berupa nilai integer setelah 6x dari nilai sigma. Hal ini dikarenakan probability mass untuk Gaussian ± 3 , dan filter memiliki nilai ganji.

Solusi

Berikut kode dari “`image make_gaussian_filter(float sigma)`”:

```
image make_gaussian_filter(float sigma)
{
    int filter_size = (int)((int)ceil(sigma * 6) % 2 == 0) ? (int)ceil(sigma * 6) + 1 :
                                                                (int)ceil(sigma * 6);

    image gs_filter = make_image(filter_size, filter_size, 1);
    for (int j = 0; j < gs_filter.h; j++){
        for (int i = 0; i < gs_filter.w; i++){
            int x = i - (filter_size - 1) / 2;
            int y = j - (filter_size - 1) / 2;
            float val = exp(-(pow(x, 2) + pow(y, 2)) / (2 * pow(sigma, 2))) /
                                                                (TWOPI * pow(sigma, 2));

            set_pixel(gs_filter, i, j, 0, val);
        }
    }
    l1_normalize(gs_filter);
    return gs_filter;
}
```

Hasil

Tahap ini memastikan gaussian filter berjalan dengan baik. Hal ini dapat dilihat dengan menambahkan & menjalankan script berikut pada “`tryhw2.py`”:

```
im = load_image("data/dog.jpg")
f = make_gaussian_filter(2)
blur = convolve_image(im, f, 1)
save_image(blur, "dog-gauss2")
```

Setelah menjalankan script di atas, akan terbentuk sebuah file “dog-gauss2”. Kode ini akan menghasilkan image dog (dog-gauss2.jpg) blurry seperti penggunaan box filter di tahap sebelumnya (2.2). Gambar di bawah ini merupakan hasil bentukan dari gaussian filter.



2.4: Hybrid images

Deskripsi

Pada task ini, implementasikan operasi penambahan & pengurangan terhadap dua citra. Hal ini dilakukan untuk mengekstraksi citra high frequency & low frequency. Untuk mendapatkan low frequency, dapat digunakan gaussian filter. Sedangkan untuk mendapatkan citra high frequency, lakukan operasi pengurangan antara citra asal dengan citra low frequency:

$$citra_{highfreq} = citra_{original} - citra_{lowfreq}$$

Berdasarkan operasi di atas, didapatkan operasi konstruksi yang dapat dilakukan dengan menambahkan citra high frequency dengan citra low frequency. Kedua operasi ini dituliskan pada method “image add_image(image a, image b)” & “image sub_image(image a, image b)”.

Solusi

```
image add_image(image a, image b)
{
    assert(a.w == b.w && a.h == b.h && a.c == b.c);
    image im_out = make_image(a.w, a.h, a.c);
    for (int i = 0; i < a.c * a.h * a.w; ++i){
        im_out.data[i] = a.data[i] + b.data[i];
    }
    return im_out;
}

image sub_image(image a, image b)
{
    assert(a.w == b.w && a.h == b.h && a.c == b.c);
    image im_out = make_image(a.w, a.h, a.c);
    for (int i = 0; i < a.c * a.h * a.w; ++i){
        im_out.data[i] = a.data[i] - b.data[i];
    }
    return im_out;
}
```

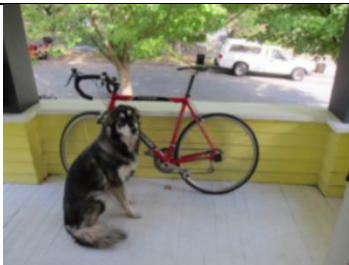
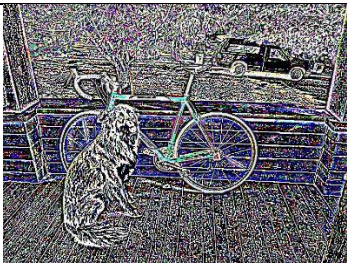

Pada method ini dilakukan perulangan sebanyak jumlah piksel dari citra, dan melakukan penambahan atau pengurangan nilai piksel untuk masing-masing piksel terkait.

Hasil

Tahap ini memastikan citra low frequency, high frequency, & construction berhasil didapatkan. Hal ini dapat dilihat dengan menambahkan script berikut pada “tryhw2.py”:

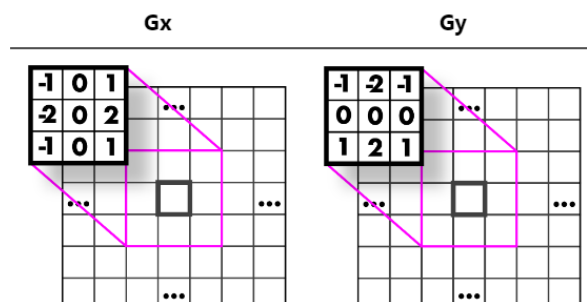
```
im = load_image("data/dog.jpg")
f = make_gaussian_filter(2)
lfreq = convolve_image(im, f, 1)
hfreq = im - lfreq
reconstruct = lfreq + hfreq
save_image(lfreq, "low-frequency")
save_image(hfreq, "high-frequency")
save_image(reconstruct, "reconstruct")
```

Berikutnya, jalankan script dengan command “python tryhw2.py”. Hasilnya, akan terbentuk 3 buah file yang memperlihatkan citra low frequency, high frequency, dan citra rekonstruksi.

Citra low frequency	Citra high Frequency	Citra rekonstruksi
		

2.5: Sobel filters

Sobel filter banyak digunakan untuk image processing & computer vision, khususnya edge detection yang menegaskan “tepi” dari sebuah citra. Sobel filter meng-estimasi gradient & direction dari sebuah citra. Operator yang digunakan berupa 2 kernel 3x3 G_x & G_y .



2.5.1: Make the filter

Deskripsi

Implementasikan kernel/filter G_x & G_y pada method “image make_gx_filter” dan “image make_gy_filter”.

Solusi

```
image make_gx_filter()
{
    image gx_filter = make_image(3, 3, 1);
    float weight[9] = {-1, 0, 1,
                       -2, 0, 2,
                       -1, 0, 1};
    memcpy(gx_filter.data, weight, sizeof(weight));
    return gx_filter;
}

image make_gy_filter()
{
    image gy_filter = make_image(3, 3, 1);
    float weight[9] = {-1, -2, -1,
                       0, 0, 0,
                       1, 2, 1};
    memcpy(gy_filter.data, weight, sizeof(weight));
    return gy_filter;
}
```

2.5.2: One more normalization...

Deskripsi

Buatlah body method untuk “feature_normalization” untuk kepentingan visualisasi dari Sobel operator. Normalisasi dilakukan agar nilai piksel berada dalam rentang 0-1. Hal yang dilakukan adalah melakukan rescaling, mengurangi nilai minimum dari semua nilai & membagi dengan range data. Jika nilai range 0, maka set nilai piksel dengan 0.

Solusi

```
void feature_normalize(image im)
{
    float min = 1, max = 0;

    for (int i = 0; i < im.c * im.h * im.w; ++i){
        if (im.data[i] > max)
            max = im.data[i];
        if (im.data[i] < min)
            min = im.data[i];
    }
}
```

```

float range = max - min;
if (!range){
    for (int i = 0; i < im.c * im.h * im.w; ++i){
        im.data[i] = 0;
    }
}
else{
    for (int i = 0; i < im.c * im.h * im.w; ++i){
        im.data[i] = (im.data[i] - min) / range;
    }
}
}

```

2.5.1: Calculate gradient magnitude and direction

Deskripsi

Isilah body method untuk “image *sobel_image(image im)”. Strategi yang dilakukan adalah perkalian antara G_x & G_y dengan image. Lakukan operasi konvolusi. Berikutnya, hitung nilai G (gradient magnitude) sebagai akar dari kuadrat penjumlahan G_x & G_y .

$$G = \sqrt{G_x^2 + G_y^2}$$

Selanjutnya, dapat ditentukan gradient direction menggunakan arc tangent dari G_x & G_y (atan2).

Solusi

```

image *sobel_image(image im)
{
    image *res = calloc(2, sizeof(image));
    image gx_filter = make_gx_filter();
    image gy_filter = make_gy_filter();
    image gx = convolve_image(im, gx_filter, 0);
    image gy = convolve_image(im, gy_filter, 0);
    image mag = make_image(gx.w, gx.h, 1);
    image theta = make_image(gx.w, gx.h, 1);
    for (int i = 0; i < im.h * im.w; ++i){
        mag.data[i] = sqrtf(gx.data[i] * gx.data[i] + gy.data[i] * gy.data[i]);
        theta.data[i] = atan2f(gy.data[i], gx.data[i]);
    }
    res[0] = mag;
    res[1] = theta;
    free_image(gx_filter);
    free_image(gy_filter);
    free_image(gx);
    free_image(gy);
}

```

```
    return res;
}
```

Hasil

Luaran yang diharapkan dapat dilihat dengan menambahkan kode berikut pada tryhw2.py:

```
im = load_image("data/dog.jpg")
res = sobel_image(im)
mag = res[0]
feature_normalize(mag)
save_image(mag, "magnitude")
```

Hasilnya adalah sebuah citra magnitude.jpg dengan gambar di bawah ini.



2.5.1: Make colorized representation

Deskripsi

Tuliskan isi/body method dari “image_colorize_sobel(image im)”. Gunakan magnitude untuk menentukan nilai saturation & value, dan angle untuk menentukan nilai hue.

Solusi

Manfaatkan function hsv_to_rgb dari data magnitude & angle/direction.

```
image_colorize_sobel(image im)
{
    image colored = make_image(im.w, im.h, 3);
    image *res = sobel_image(im);
    feature_normalize(res[0]);
    feature_normalize(res[1]);
    memcpy(colored.data, res[1].data, im.h * im.w * sizeof(float));
    memcpy(colored.data + im.h * im.w, res[0].data, im.h * im.w * sizeof(float));
    memcpy(colored.data + 2 * im.h * im.w, res[0].data, im.h * im.w * sizeof(float));
    hsv_to_rgb(colored);
    free_image(res[0]);
    free_image(res[1]);
}
```

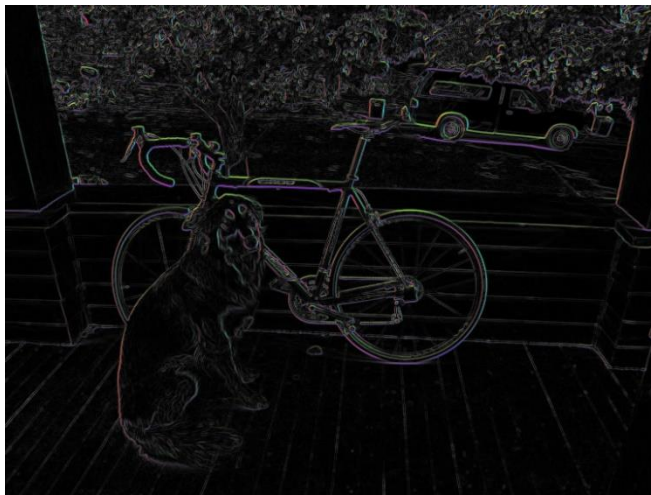
```
    return colored;
}
```

Hasil

Untuk melihat hasil dari operasi ini, tambahkan kode berikut pada tryhw2.py:

```
im = load_image("data/dog.jpg")
colorize_image = colorize_sobel(im)
save_image(colorize_image, "colorize_dog")
```

Berikutnya, seperti biasa, jalankan tryhw2.py dengan command “python tryhw2.py”. Hasilnya, akan terbentuk colorized_dog.jpg dengan gambar seperti di bawah ini.



Hasil Skenario Test

Untuk memastikan setiap method yang diimplementasikan/ditulisakan body method-nya sesuai harapan, jalankan command “uwimg test hw2” di command prompt/terminal. Testing ini terdiri dari 15 skenario test, dalam pengujian 9 method berikut:

- test_gaussian_filter();
- test_sharpen_filter();
- test_emboss_filter();
- test_highpass_filter();
- test_convolution();
- test_gaussian_blur();
- test_hybrid_image();
- test_frequency_image();
- test_sobel();

Jika sudah dikodekan dengan benar, maka hasil test menunjukkan status passed untuk setiap kasus uji.

```
C:\Users\rezab\Documents\Studi Lanjut\Sem 3\Vision - IF6083\If6083ComputerVision>uwimg test hw2
passed: [test_gaussian_filter] testing [same_image(f, gt, EPS)] in ./src/test.c, line 344
passed: [test_sharpen_filter] testing [same_image(blur, gt, EPS)] in ./src/test.c, line 314
passed: [test_emboss_filter] testing [same_image(blur, gt, EPS)] in ./src/test.c, line 299
passed: [test_highpass_filter] testing [same_image(blur, gt, EPS)] in ./src/test.c, line 284
passed: [test_convolution] testing [same_image(blur, gt, EPS)] in ./src/test.c, line 328
passed: [test_gaussian_blur] testing [same_image(blur, gt, EPS)] in ./src/test.c, line 356
passed: [test_hybrid_image] testing [same_image(reconstruct, gt, EPS)] in ./src/test.c, line 373
passed: [test_frequency_image] testing [same_image(lfreq, low_freq, EPS)] in ./src/test.c, line 396
passed: [test_frequency_image] testing [same_image(hfreq, high_freq, EPS)] in ./src/test.c, line 397
passed: [test_frequency_image] testing [same_image(reconstruct, im, EPS)] in ./src/test.c, line 398
passed: [test_sobel] testing [gt_mag.w == mag.w && gt_theta.w == theta.w] in ./src/test.c, line 418
passed: [test_sobel] testing [gt_mag.h == mag.h && gt_theta.h == theta.h] in ./src/test.c, line 419
passed: [test_sobel] testing [gt_mag.c == mag.c && gt_theta.c == theta.c] in ./src/test.c, line 420
passed: [test_sobel] testing [same_image(mag, gt_mag, EPS)] in ./src/test.c, line 436
passed: [test_sobel] testing [same_image(theta, gt_theta, EPS)] in ./src/test.c, line 437
15 tests, 15 passed, 0 failed
```