

Morphological Dilation and Erosion Filter Visualization Step-by-Step Explaination

Nama : Faiz Noor Adhytia

NIM : 22.K3.0006

Dilation

Berikut merupakan visualisasi Step-by-Step mengenai proses pembentukan Morphological Distilation Visualization, cross-shaped kernel.

```
def manual_dilate_stepwise(binary, kernel):
    """
    Slide kernel over binary image and yield frames showing the effect.
    The binary image is 0/255. Kernel is 0/1 (uint8).
    Yields tuples: (center_y, center_x, output_before_cropped,
    output_after_cropped)
    """

    # Ensure proper formats
    bin_img = (binary > 127).astype(np.uint8) * 255
    k = (kernel != 0).astype(np.uint8)
    h, w = bin_img.shape
    kh, kw = k.shape
    pad_h, pad_w = kh // 2, kw // 2

    # pad input and output
    padded = np.pad(bin_img, ((pad_h, pad_h), (pad_w, pad_w)), constant_values=0)
    out = padded.copy() # we'll update this in padded coordinates

    # iterate over all centers (padded coords)
    for y in range(pad_h, pad_h + h):
        for x in range(pad_w, pad_w + w):

            # window under kernel
            win = padded[y-pad_h:y+pad_h+1, x-pad_w:x+pad_w+1]
            before = out.copy()

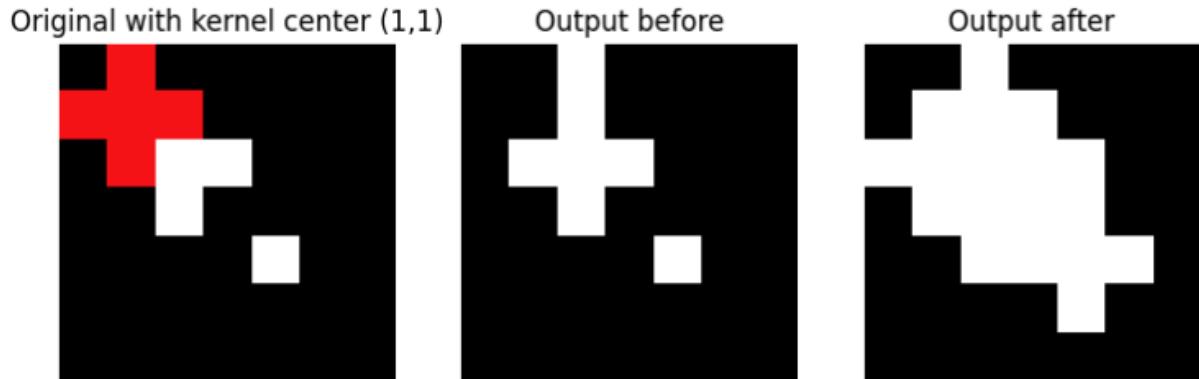
            # dilation rule: if any input pixel under kernel==255 where kernel==1 => set
            # center to 255
            if np.any((win == 255) & (k == 1)):
                out[y, x] = 255
            # yield only if changed (useful for large images)
            if before[y, x] != out[y, x]:
                # return cropped (original) images (without padding)
                out_cropped_before = before[pad_h:pad_h+h, pad_w:pad_w+w]
                out_cropped_after = out[pad_h:pad_h+h, pad_w:pad_w+w]
                center_y = y - pad_h
                center_x = x - pad_w
                yield (center_y, center_x, out_cropped_before, out_cropped_after)
```

https://github.com/notyouriiz/Image_Processing/blob/main/Morphological%20Operations/Dilation.ipynb

Berikut merupakan persamaan untuk proses *Eroton* yang dijabarkan pada Persamaan (1)

$$A \oplus B = \{z \mid (\hat{B})_z \cap A \neq \emptyset\} \quad (1)$$

Saat elemen struktur B ditempatkan di posisi tertentu dalam citra (setelah refleksi dan pergeseran), jika **setidaknya satu piksel elemen struktur tumpang tindih dengan foreground (piksel bernilai 1)**, maka pixel output pada posisi pusat kernel akan diset menjadi 1 (foreground). Proses ini mengakibatkan perluasan (expansion) area foreground pada citra.[1].



Gambar 1. Visualisasi *Dilation*

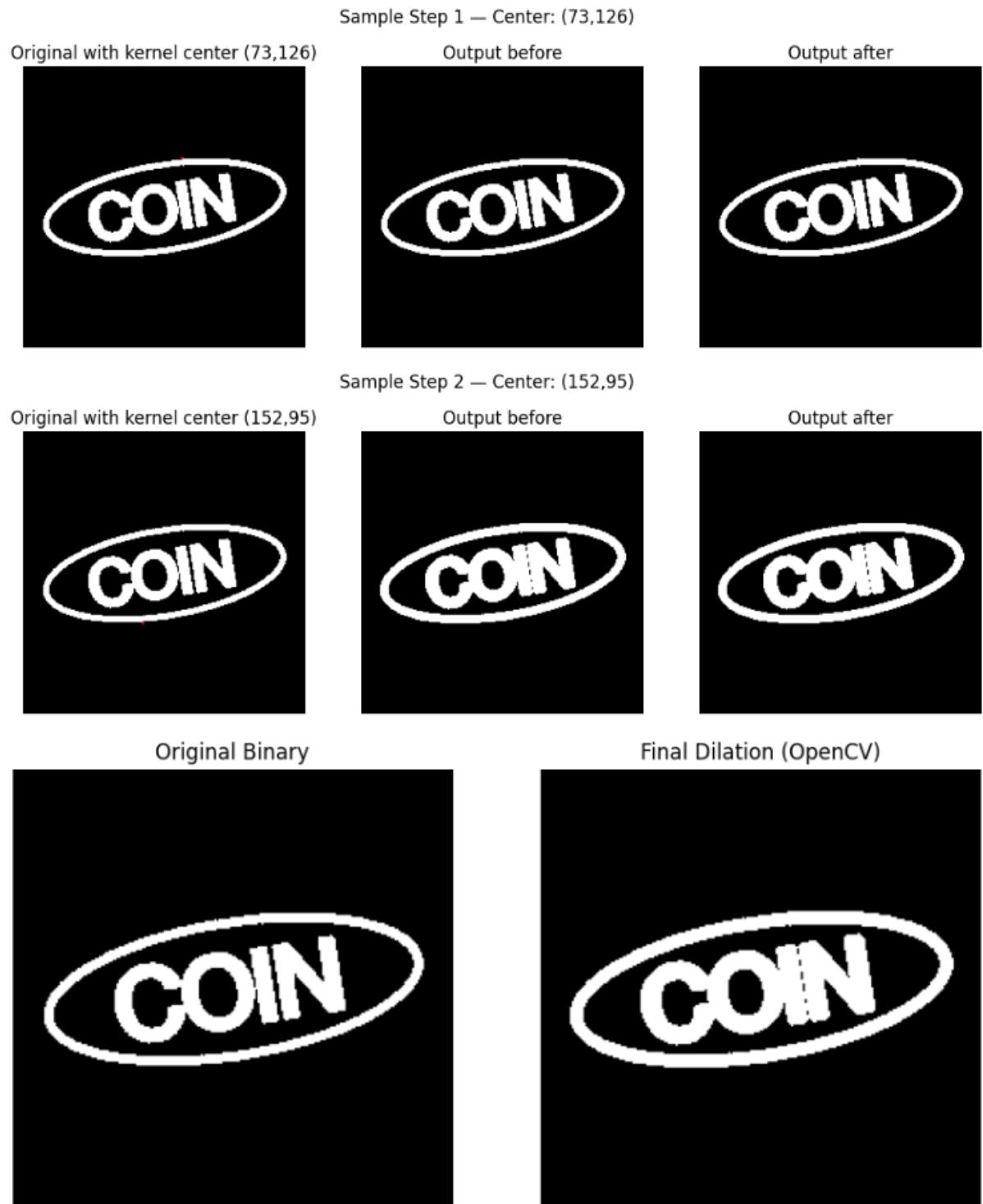
Tabel 1. Keterangan Proses Implementasi *Dilation*

Original with Kernel Center (1,1)	Output Before	Output After
a) Citra biner menunjukkan objek berwarna putih (foreground) pada latar belakang hitam.	a) Menampilkan citra output sebelum langkah dilasi ini diterapkan.	Jika ada piksel putih yang tumpang tindih dengan kernel, maka piksel di pusat kernel menjadi putih. Hasilnya, area putih berkembang, mengisi piksel tetangga sesuai bentuk kernel merah.
b) Silang merah adalah elemen struktur (kernel) yang dipusatkan pada posisi piksel (1,1).	b) Bentuk objek tampak sama seperti input karena <i>Dilation</i> belum dilakukan pada posisi kernel ini.	
c) Kernel ini menentukan tetangga piksel yang memengaruhi proses <i>Dilation</i> .		

Tabel 2. Keterangan Interpretasi *Dilation*

Interpretasi	
Red Cross (Kernel Cross-Shaped)	menentukan tetangga yang diperiksa
Sebelum vs Sesudah	terlihat bagaimana kernel menambahkan piksel baru di sekitar objek sehingga objek menjadi lebih tebal.
Efek	<i>Dilation</i> memperbesar area <i>Foreground</i> atau dengan kata lain mengisi celah serta menghubungkan bagian yang terputus

Contoh Implementasi Dilation



Gambar 2. Implementasi *Dilation*

Eroton

Berikut merupakan visualisasi Step-by-Step mengenai proses pembentukan Morphological Distilation Visualization, cross-shaped kernel dengan konfigurasi 3×3 .

```
def manual_erode_stepwise(img, kernel):
    """
    Simulates erosion step by step.
    For each pixel, check if the kernel fits entirely inside white pixels.
    """
    h, w = img.shape
    kh, kw = kernel.shape
    pad_y, pad_x = kh // 2, kw // 2
    padded = np.pad(img, ((pad_y, pad_y), (pad_x, pad_x)),
mode='constant', constant_values=0)
    output = img.copy()

    steps = []
    for y in range(h):
        for x in range(w):
            region = padded[y:y+kh, x:x+kw]

    # erosion condition: all kernel '1' must cover white pixels
    if np.all(region[kernel==1] == 255):
        new_val = 255
    else:
        new_val = 0

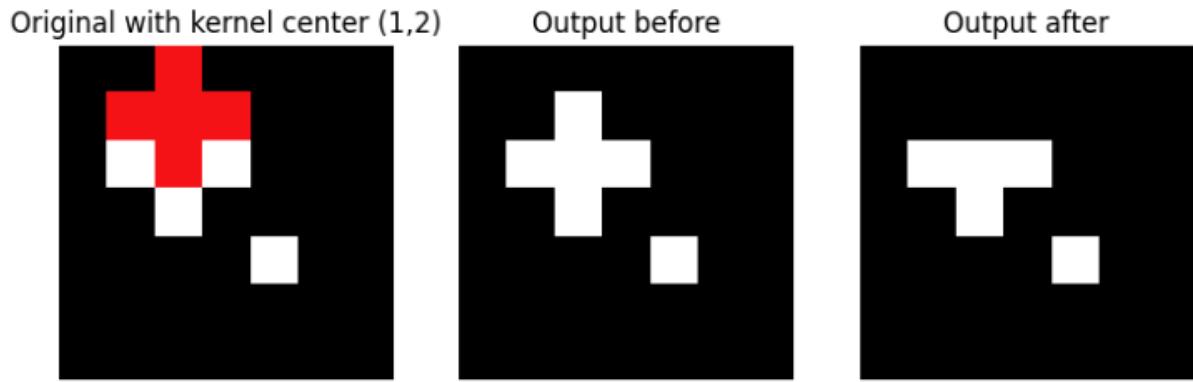
    if new_val != output[y, x]:
        before = output.copy()
        output[y, x] = new_val
        steps.append((y, x, before, output.copy()))
    return steps
```

https://github.com/notyouriiz/Image_Processing/blob/main/Morphological%20Operations/Eroton.ipynb

Berikut merupakan persamaan untuk proses *Eroton* yang dijabarkan pada Persamaan (1)

$$A \ominus B = \{z \mid B_z \subseteq A\} \quad (1)$$

di mana B_z adalah elemen struktur B yang digeser oleh vektor z . Artinya, piksel output pada posisi z akan bernilai 1 jika seluruh piksel elemen struktur B yang sudah digeser (translasi) berada di dalam foreground A . Jika tidak, piksel output menjadi 0 [1].



Gambar 3. Visualisasi *Eroton*

Tabel 3. Keterangan Proses Implementasi *Eroton*

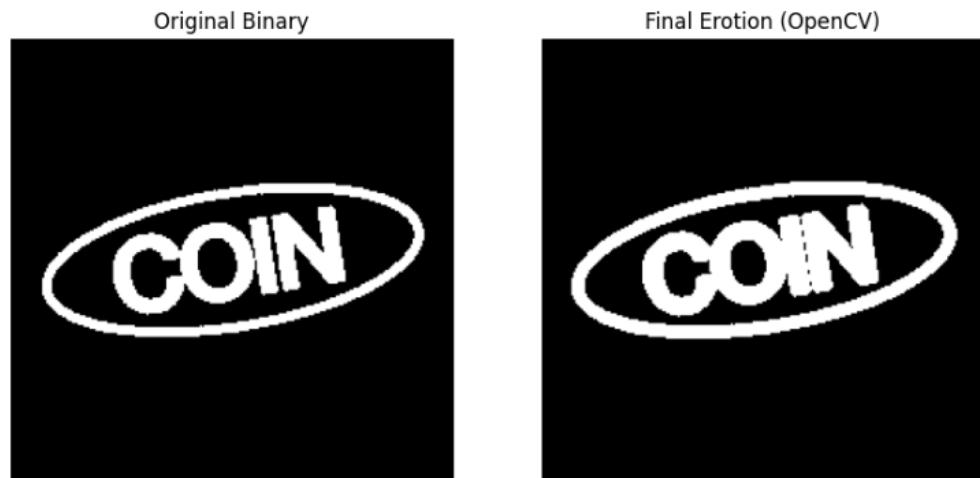
Original with Kernel Center (1, 2)	Output Before	Output After
a) Citra biner menampilkan objek putih pada latar belakang hitam.	a) Menampilkan kondisi citra output sebelum <i>Eroton</i> diterapkan pada posisi kernel ini.	Algoritma memeriksa apakah kernel sepenuhnya berada di dalam objek putih. Jika ada bagian kernel yang tumpang tindih dengan latar belakang hitam, piksel di pusat kernel diubah menjadi hitam.
b) Silang merah adalah elemen struktur (kernel), dipusatkan pada koordinat piksel (1,2).	b) Pada tahap ini, output masih terlihat seperti objek putih asli.	Akibatnya, bentuk putih menyusut dan bagian tipis hilang, dan piksel kecil yang tersisa bisa menghilang sepenuhnya.
c) Untuk operasi <i>Eroton</i> , kernel memeriksa apakah semua piksel di bawah kernel sesuai dengan <i>foreground</i> (putih).		

Tabel 4. Keterangan Interpretasi *Eroton*

Interpretasi	
Red Cross (Kernel Cross-Shaped)	menentukan tetangga yang diperiksa
Sebelum vs Sesudah	objek putih menjadi lebih tipis dan kecil karena <i>Eroton</i> menghilangkan piksel di tepi.
Efek	<i>Eroton</i> mengecilkan area foreground, menghilangkan noise, dan memisahkan objek yang tersambung lemah

Contoh Implementasi *Eroton*

Total added pixels (cross kernel): 1417



Gambar 4. Implementasi *Eroton*

References

- [1] BASIC MORPHOLOGICAL IMAGE PROCESSING OPERATIONS: A TUTORIAL,
https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/GASTERATOS/SOFT/2.htm (accessed 9 September 2025).