ipcv

November 26, 2024

Import Libraries

[325]:
```
import PIL
from PIL import Image
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import cv2
```

Data

[326]:
```
train=pd.read_csv('mnist_test.csv')
train.head()
```

[326]:

	label	1x1	1x2	1x3	1x4	1x5	1x6	1x7	1x8	1x9	…	28x19	28x20	\
0	7	0	0	0	0	0	0	0	0	0	…	0	0	
1	2	0	0	0	0	0	0	0	0	0	…	0	0	
2	1	0	0	0	0	0	0	0	0	0	…	0	0	
3	0	0	0	0	0	0	0	0	0	0	…	0	0	
4	4	0	0	0	0	0	0	0	0	0	…	0	0	

	28x21	28x22	28x23	28x24	28x25	28x26	28x27	28x28
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0

[5 rows x 785 columns]

Image 1

[327]:
```
rownum=100
#change this value to get another example from the training dataset
if rownum>-1 and rownum<260715:
    pixels=train.iloc[rownum][1:].values.reshape(28, 28)
    array=np.array(pixels, dtype=np.uint8)
    img=Image.fromarray(array)
```

```
    img.save("check.jpg")
    # cv2.imwrite("check.jpg", array)
else:
    print("Row index out of bounds")
img=plt.imread('check.jpg')
plt.imshow(img, cmap='Greys_r')
plt.axis('off')
plt.show()
```

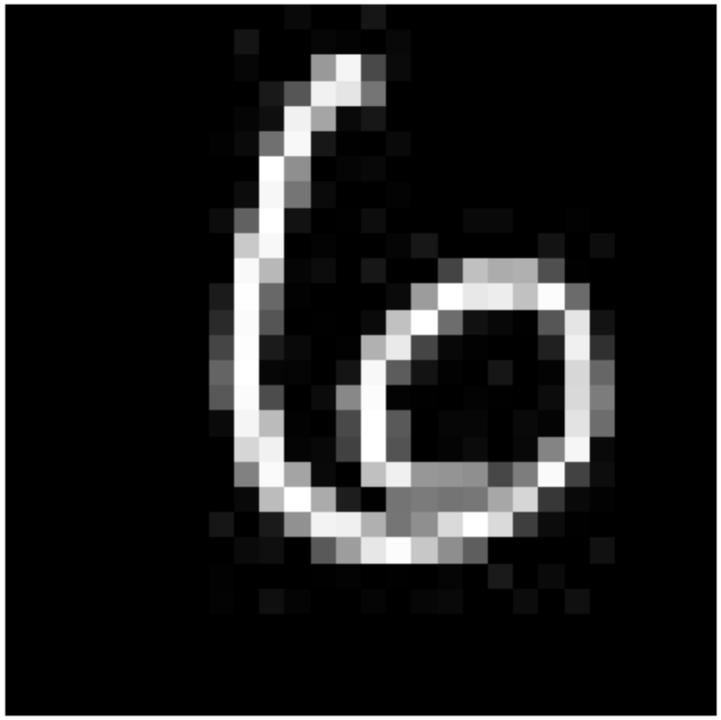
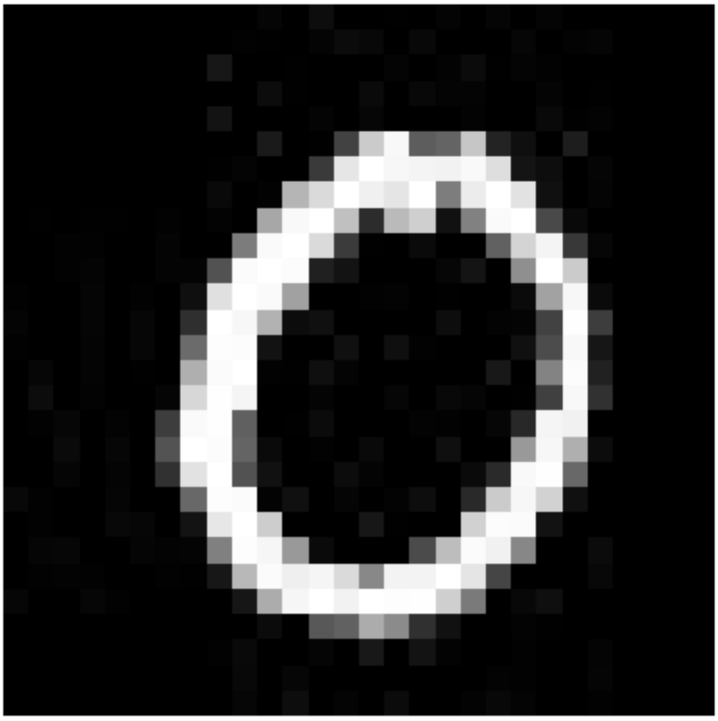


Image 2

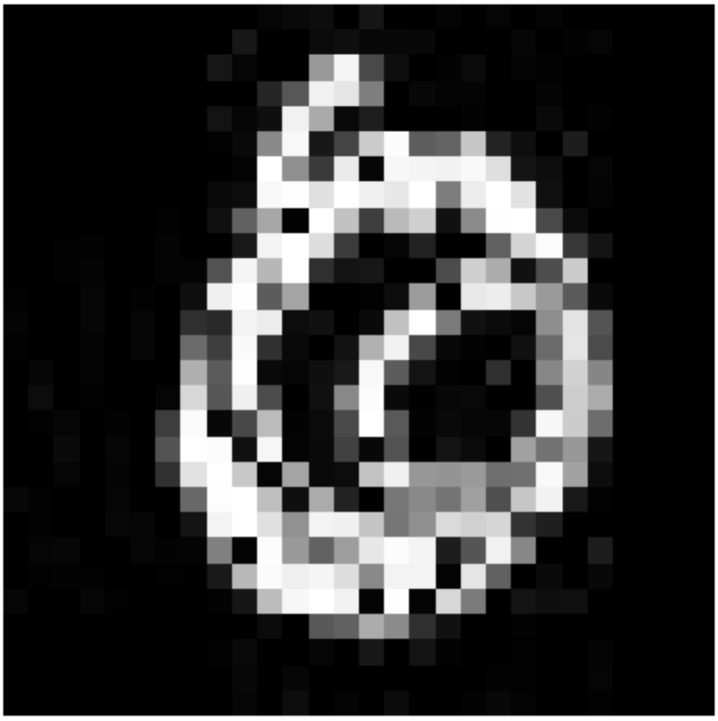
[328]:
```
rownum=101
#change this value to get another example from the training dataset
if rownum>-1 and rownum<260715:
    pixels=train.iloc[rownum][1:].values.reshape(28, 28)
    array=np.array(pixels, dtype=np.uint8)
    img1=Image.fromarray(array)
    img1.save("check1.jpg")
    # cv2.imwrite("check1.jpg", array)
else:
    print("Row index out of bounds")
img1=plt.imread('check1.jpg',)
plt.imshow(img1, cmap='Greys_r')
```

```
plt.axis('off')
plt.show()
```



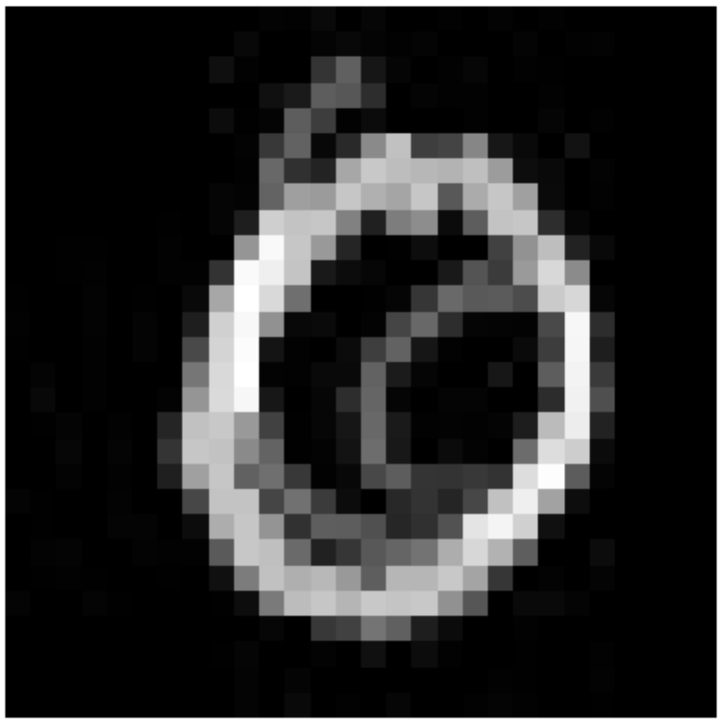
Addition

[329]:
```
plt.imshow(img1+img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



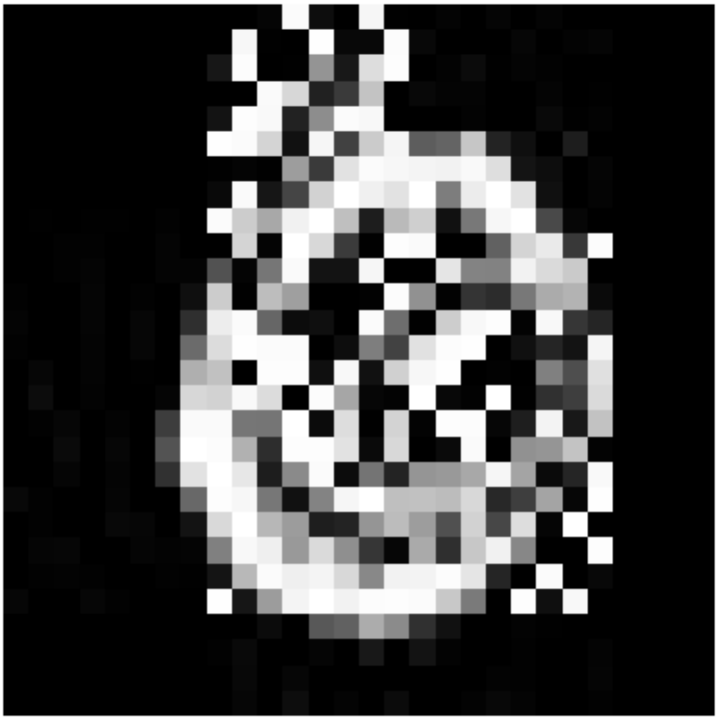
Weighted Addition

[330]:
```
plt.imshow(img1+img*0.5, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Subtraction

[331]:
```
plt.imshow(img1-img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



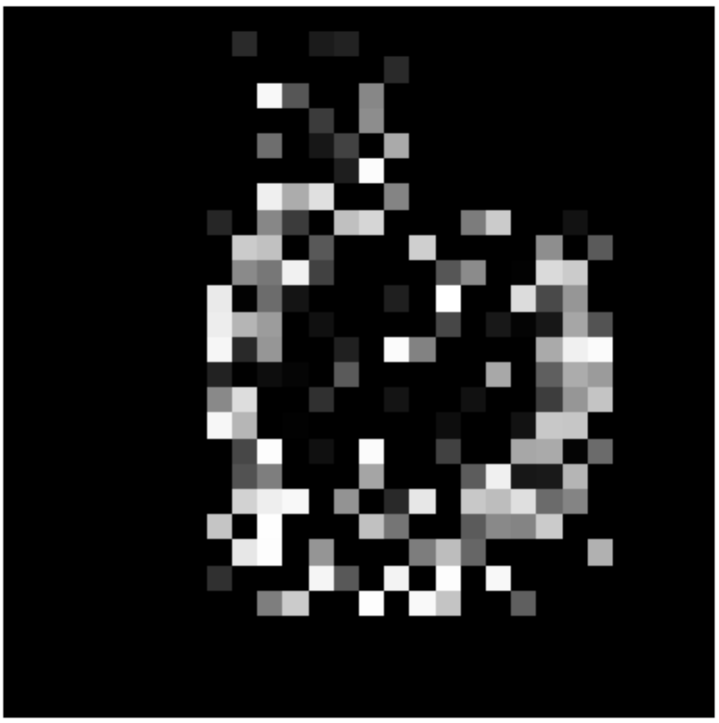
Division

[332]:
```
plt.imshow(img1/img, cmap='Greys_r')
plt.axis('off')
plt.show()
```

```
/var/folders/nx/qjk1687x5lq067mvl83vlw_c0000gn/T/ipykernel_97881/3009222443.py:1
: RuntimeWarning: divide by zero encountered in divide
  plt.imshow(img1/img, cmap='Greys_r')
/var/folders/nx/qjk1687x5lq067mvl83vlw_c0000gn/T/ipykernel_97881/3009222443.py:1
: RuntimeWarning: invalid value encountered in divide
  plt.imshow(img1/img, cmap='Greys_r')
```

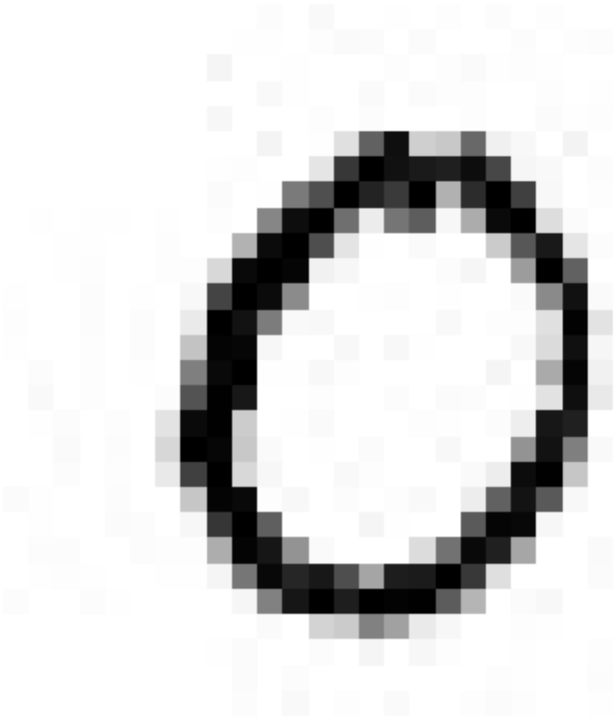
Multiplication

[333]:
```
plt.imshow(img1*img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



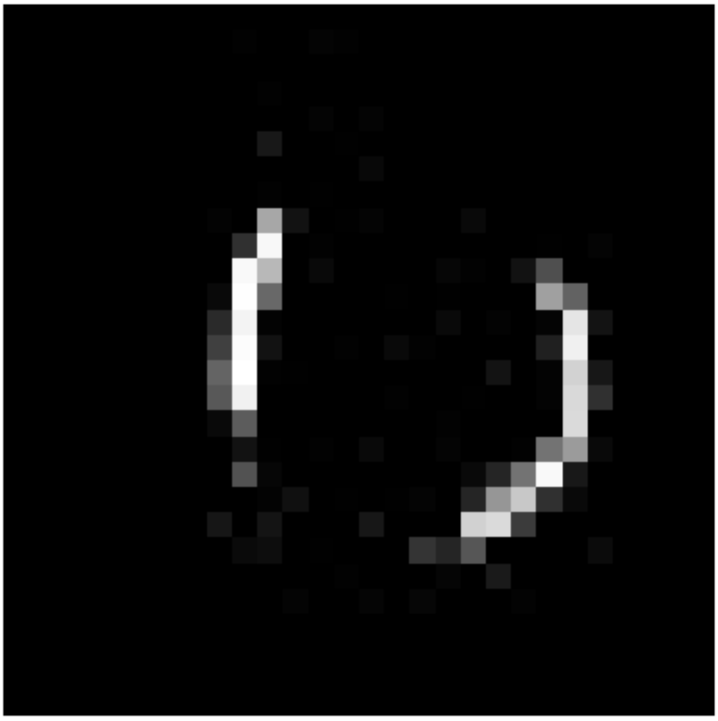
Inverse

[334]:
```
plt.imshow(~img1, cmap='Greys_r')
plt.axis('off')
plt.show()
```



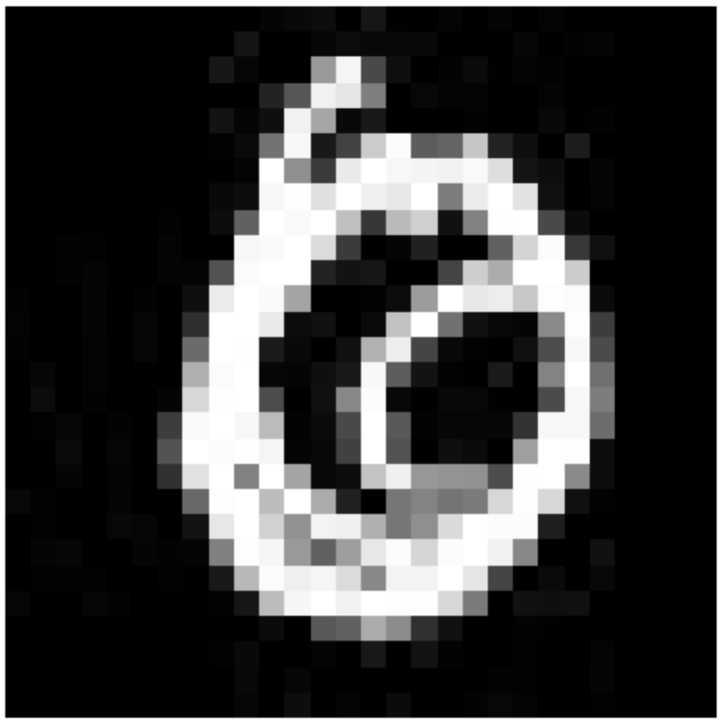
Bitwise AND

[335]:
```
plt.imshow(img1&img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



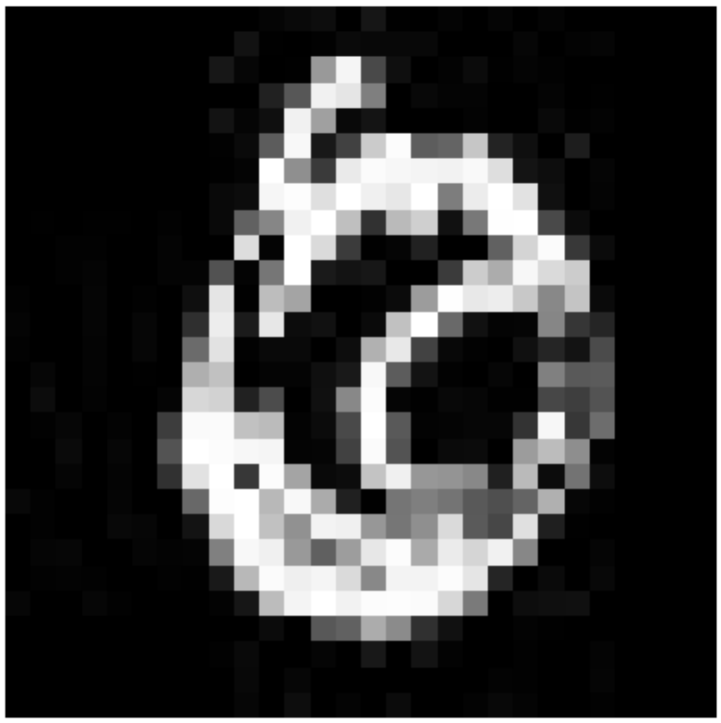
Bitwise OR

[336]:
```
plt.imshow(img1|img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



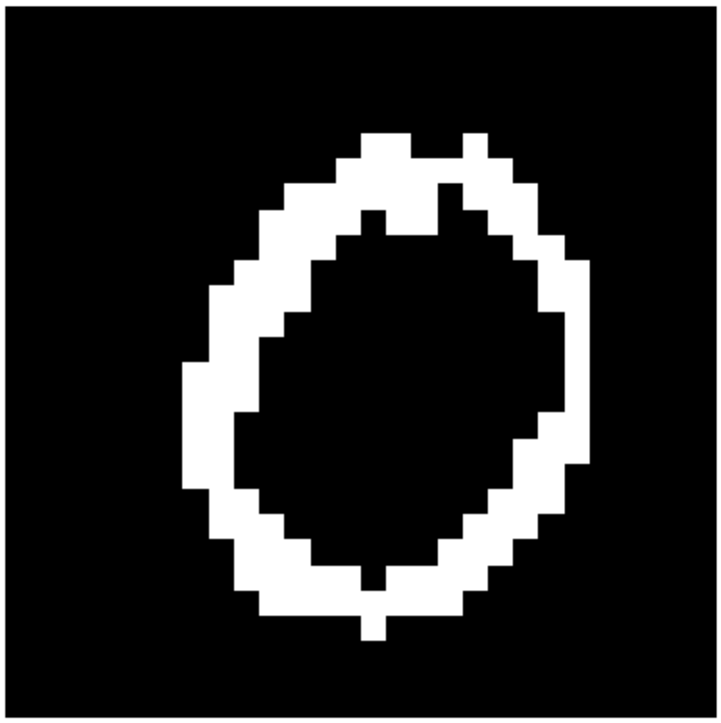
Bitwise XOR

[337]:
```
plt.imshow(img1^img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Bitwise Right Shift

[338]:
```
plt.imshow(img1>>7, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Bitwise Left Shift

[339]:
```
plt.imshow(img1<<3, cmap='Greys_r')
plt.axis('off')
plt.show()
```

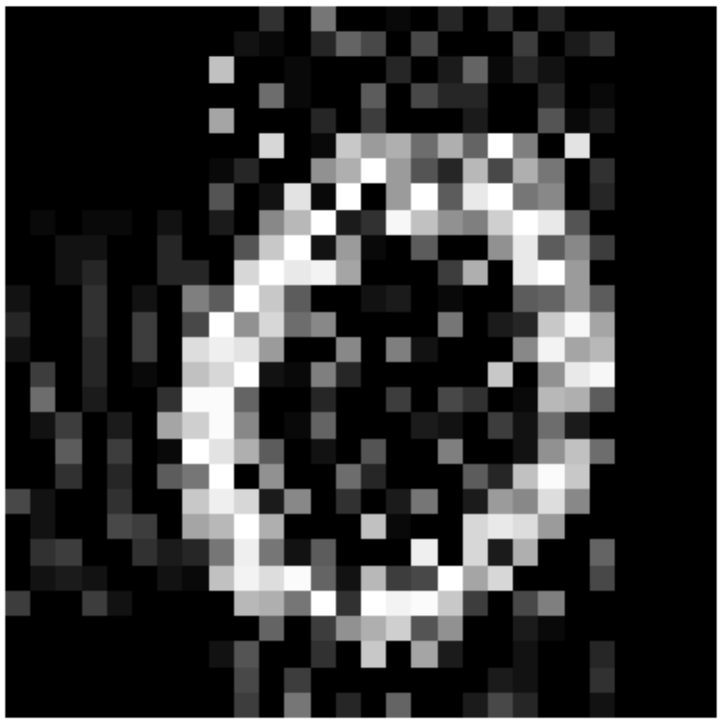
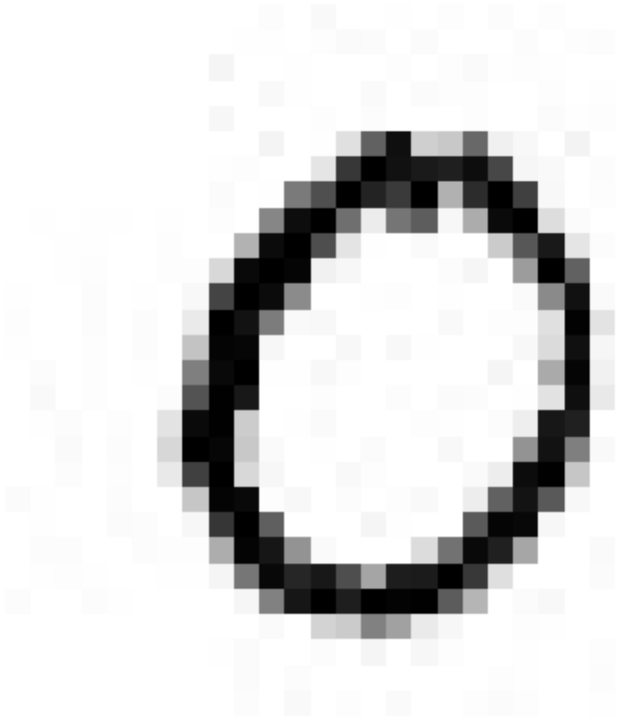


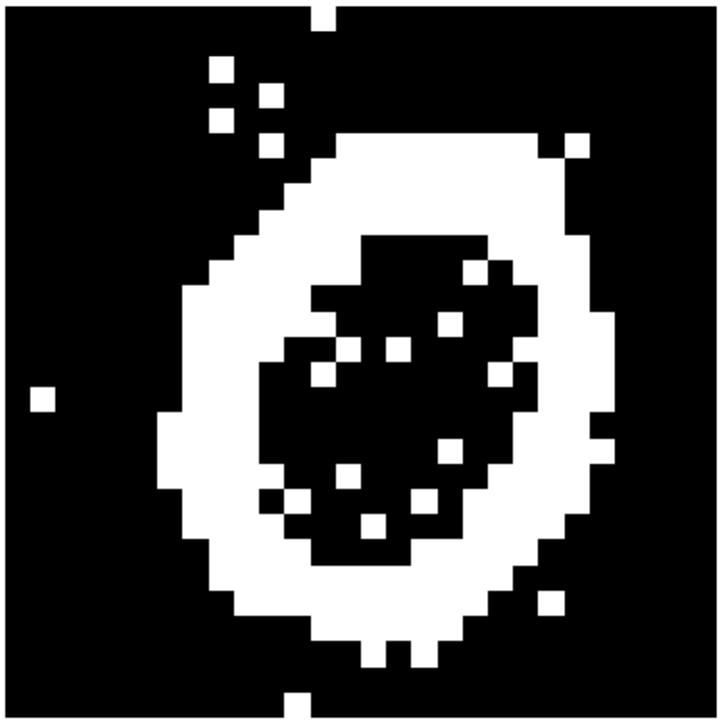
Image Negative

[340]:
```
plt.imshow(~img1, cmap='Greys_r')
plt.axis('off')
plt.show()
```



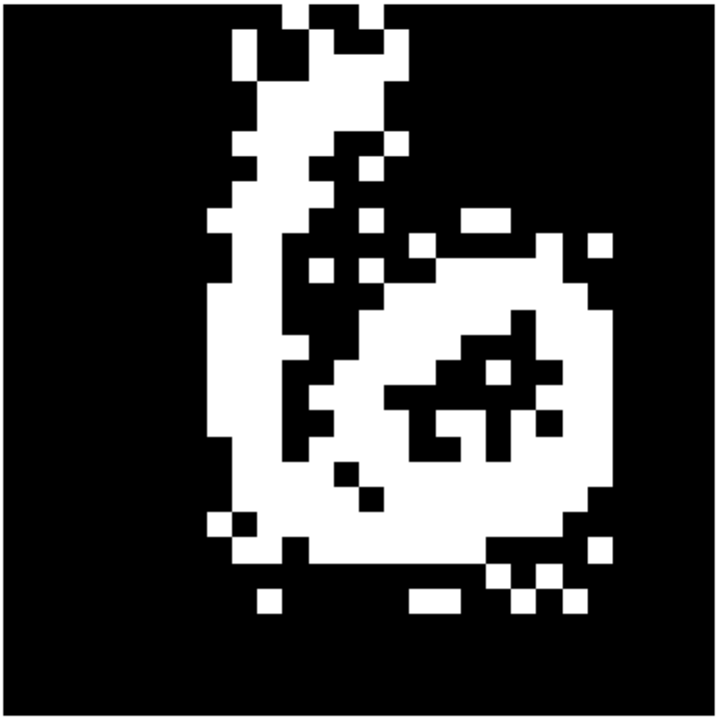
Thresholding

[341]:
```
plt.imshow(img1>10, cmap='Greys_r')
plt.axis('off')
plt.show()
```



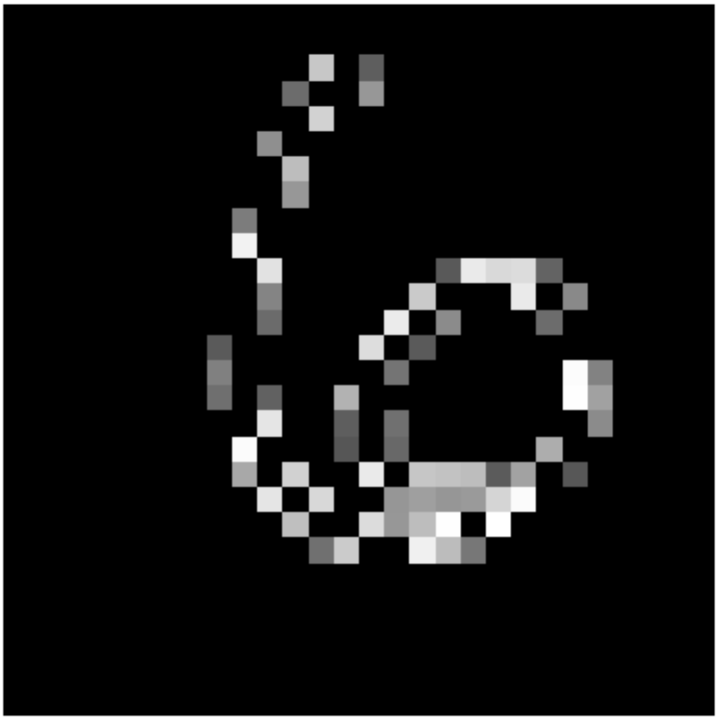
Grey Level Slicing without Background

[342]:
```
img_glswo=img.copy()
n=len(img)
threshold=5
for i in range(n):
    for j in range(n):
        if img_glswo[i][j]>threshold:
            img_glswo[i][j]=255
        else:
            img_glswo[i][j]=0
plt.imshow(img_glswo, cmap='Greys_r')
plt.axis('off')
plt.show()
```



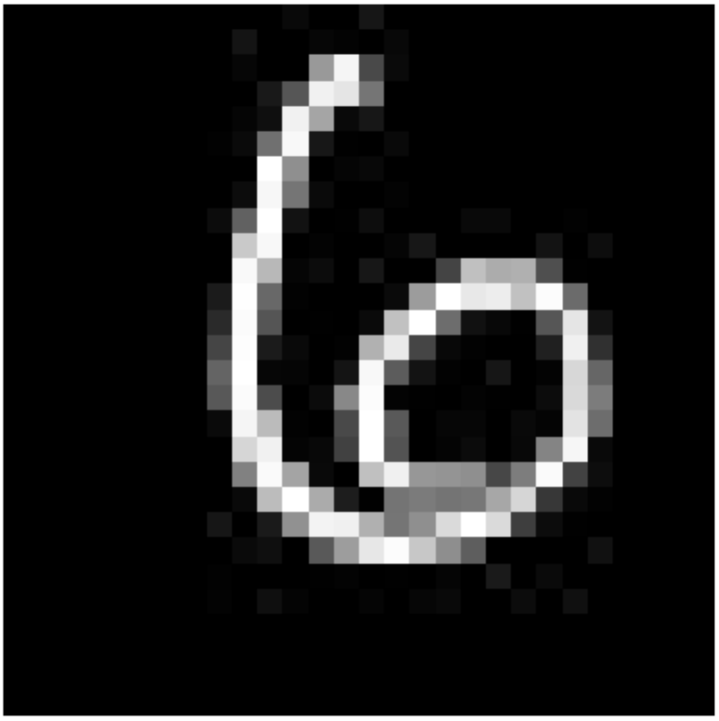
Grey Level Slicing with Background

[343]:
```
img_gls=img.copy()
n=len(img)
threshold1=50
threshold2=200
for i in range(n):
    for j in range(n):
        if threshold2>img_gls[i][j]>threshold1:
            img_gls[i][j]=img_gls[i][j]
        else:
            img_gls[i][j]=0
plt.imshow(img_gls, cmap='Greys_r')
plt.axis('off')
plt.show()
```



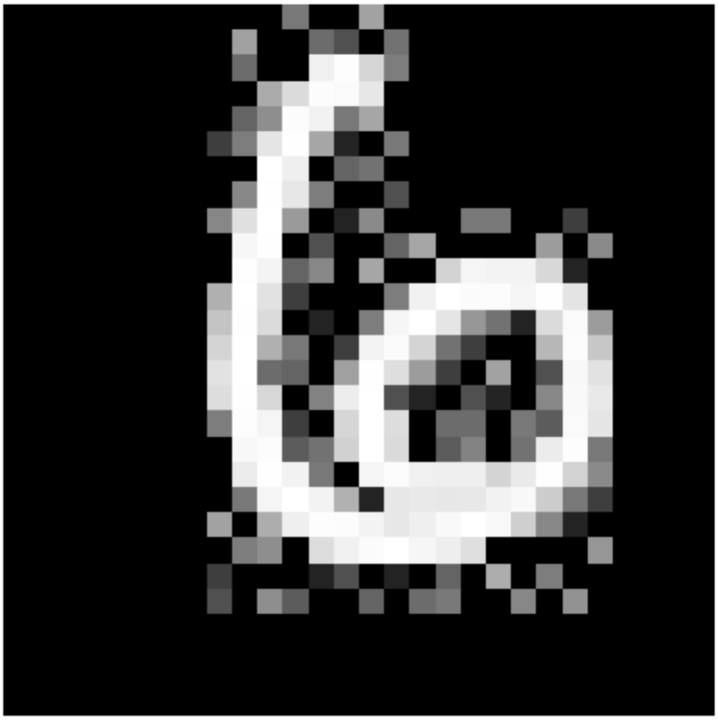
Min Max Stretching

[344]:
```
min_intensity=np.min(img)
max_intensity=np.max(img)
stretched_img=((img-min_intensity)/(max_intensity-min_intensity)*255).astype(np.
↪uint8)
plt.imshow(stretched_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



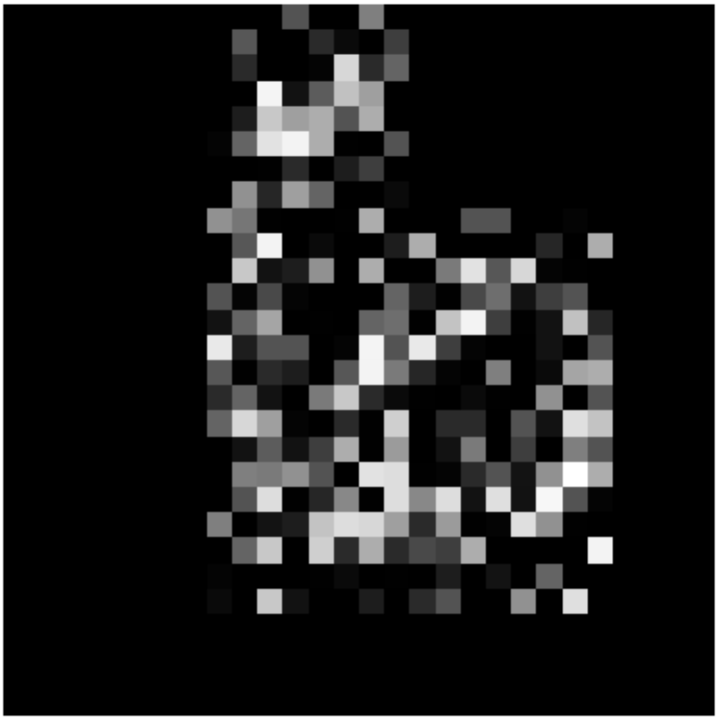
Log Transformation

[345]:
```
c=1.0
log_img=c*np.log1p(img)
log_img=np.uint8(255*(log_img-np.min(log_img))/(np.max(log_img)-np.
  ↪min(log_img)))
plt.imshow(log_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Power Law Transformation

[346]:
```
c=1.5
gamma=2
power_img=c*np.power(img,gamma)
power_img=np.uint8(255*(power_img-np.min(power_img))/(np.max(power_img)-np.
  ↪min(power_img)))
plt.imshow(power_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



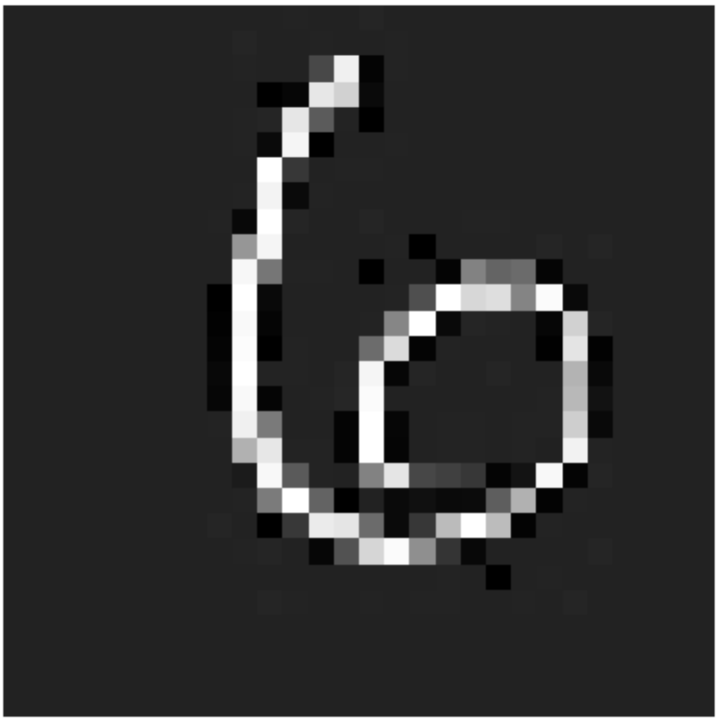
Contrast Stretching

[347]:
```
n=len(img)
r1=20
r2=100
s1=10
s2=20
L=255
contrast_img=img.copy()
for i in range(n):
    for j in range(n):
        pixel = img[i][j]
        if pixel <= r1:
            contrast_img[i, j] = s1 + (pixel - r1) * ((s2 - s1) / (r2 - r1))
        elif pixel <= r2:
            contrast_img[i, j] = s1 + (pixel - r1) * ((s2 - s1) / (r2 - r1))
        else:
            contrast_img[i, j] = s2 + (pixel - r2) * ((L - 1 - s2) / (L - 1 -␣
 ↪r2))

contrast_img = np.clip(contrast_img, 0, L - 1).astype(np.uint8)
plt.imshow(contrast_img, cmap='Greys_r')
plt.axis('off')
```

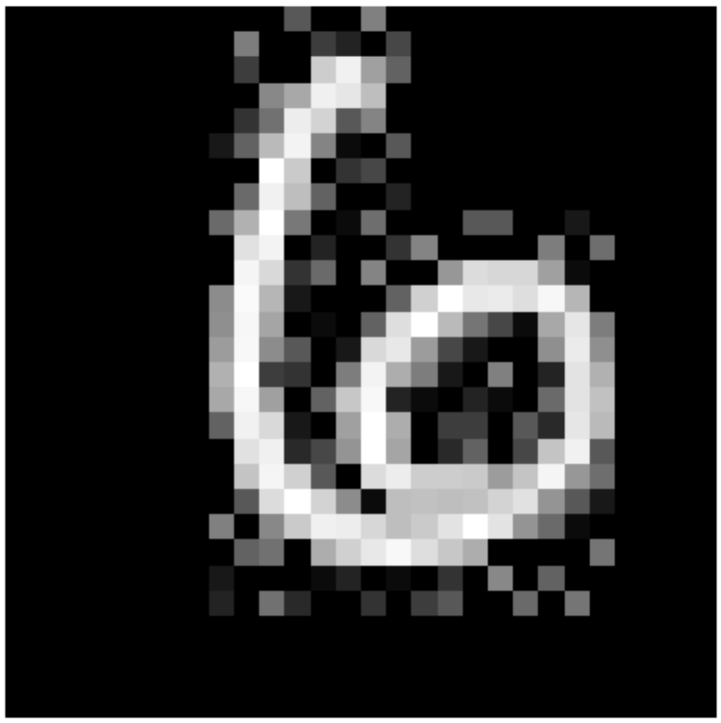
```
plt.show()
```

```
/var/folders/nx/qjk1687x5lq067mvl83vlw_c0000gn/T/ipykernel_97881/1591491351.py:1
2: RuntimeWarning: overflow encountered in scalar subtract
  contrast_img[i, j] = s1 + (pixel - r1) * ((s2 - s1) / (r2 - r1))
```

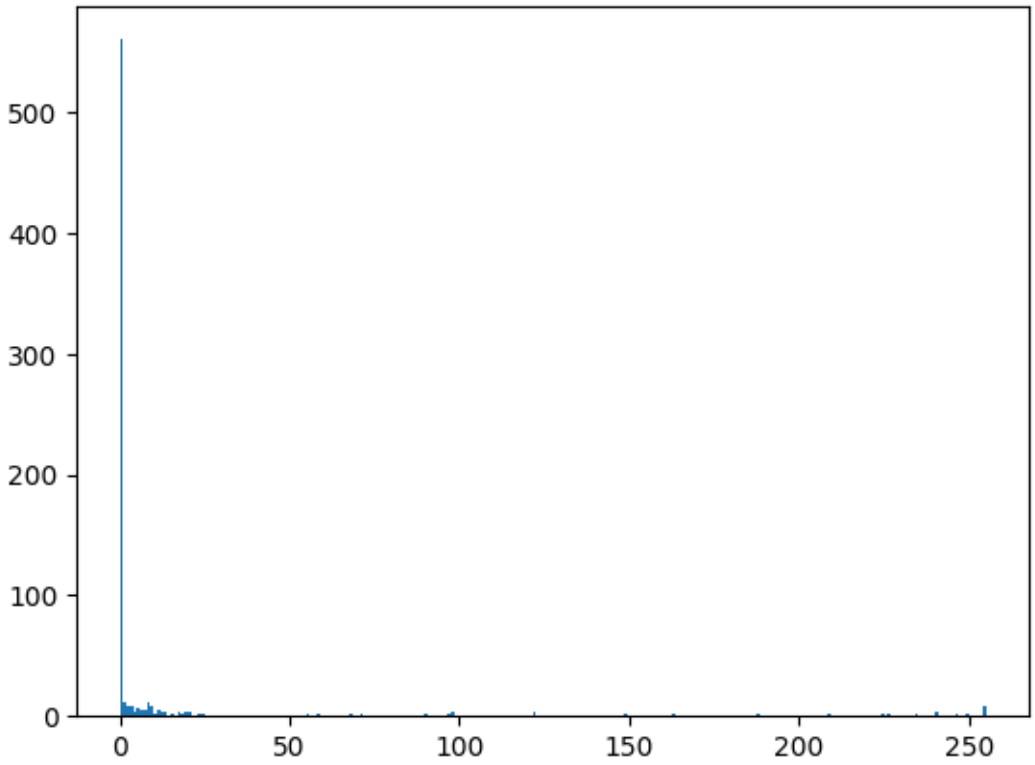


Histogram Equalization

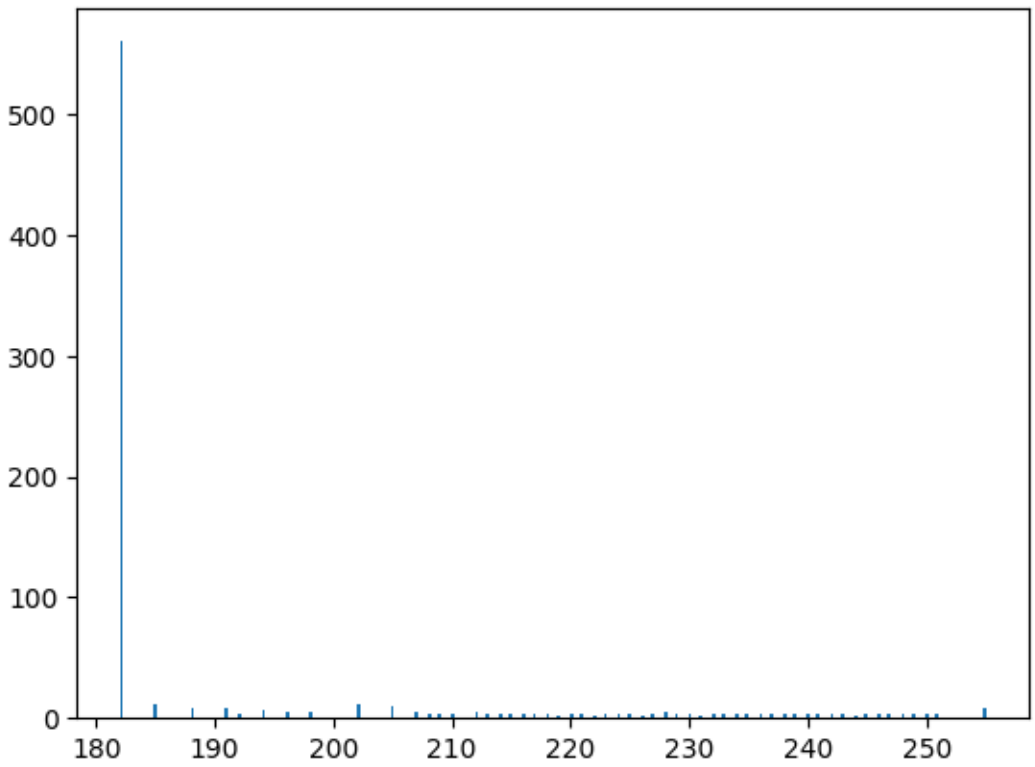
[348]:
```
histogram, bins = np.histogram(img.flatten(), bins=256)
cdf = histogram.cumsum()
cdf_normalized = cdf / cdf[-1]  # Normalize to range [0, 1]
cdf_scaled = (cdf_normalized * 255).astype(np.uint8)  # Scale the CDF to [0,␣
↪255]
equalized_image = cdf_scaled[img]
plt.imshow(equalized_image, cmap='Greys_r')
plt.axis('off')
plt.show()
```



[349]:
```
plt.hist(img.flatten(),bins=256)
plt.show()
```

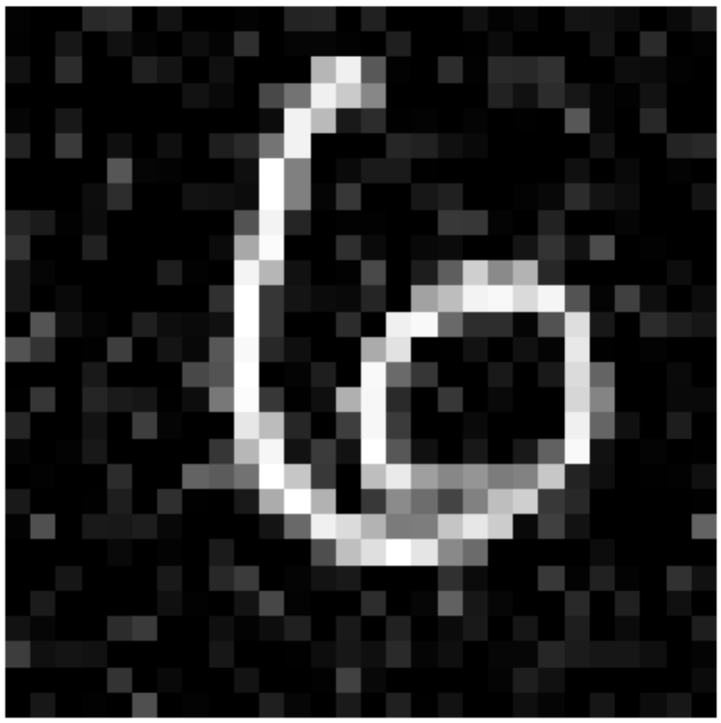


[350]:
```
plt.hist(equalized_image.flatten(),bins=256)
plt.show()
```



Gaussian Noise

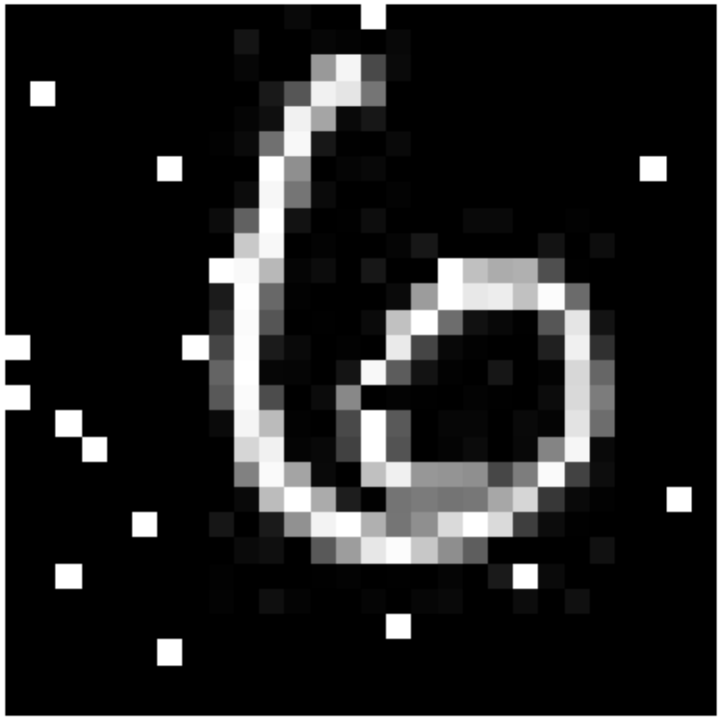
[351]:
```
mean=0
std=25
noise = np.random.normal(mean, std, img.shape)
noisy_image = img + noise
noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)
plt.imshow(noisy_image, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Salt and Pepper Noise

[352]:
```
salt_prob=0.02
pepper_prob=0.02
sp_noisy_image = np.copy(img)
random_matrix = np.random.rand(*img.shape)

sp_noisy_image[random_matrix < salt_prob] = 255
sp_noisy_image[random_matrix > 1 - pepper_prob] = 0
plt.imshow(sp_noisy_image, cmap='Greys_r')
plt.axis('off')
plt.show()
```

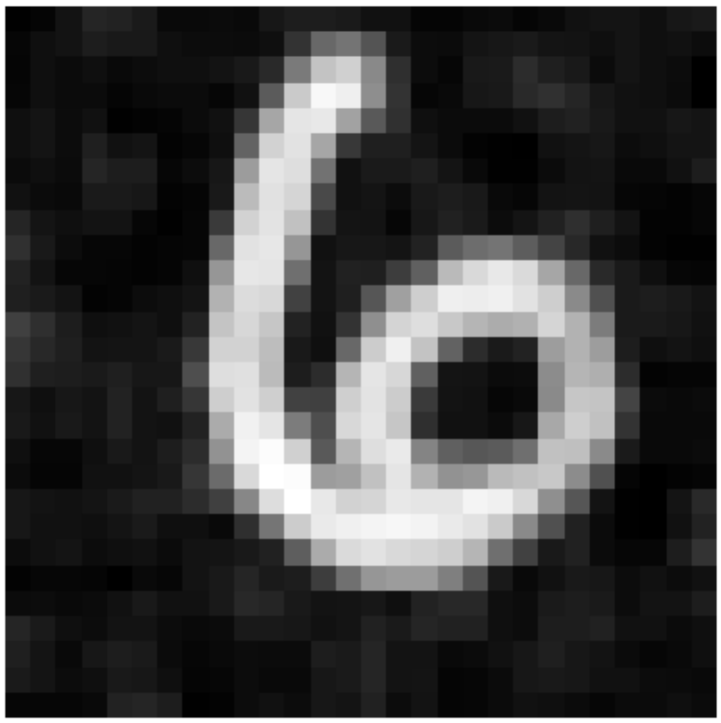


1 Apply Filter

[353]:
```
from scipy.ndimage import convolve
def apply_filter_scipy(img, kernel):
    filtered_img = convolve(img, kernel)
    return filtered_img
```

Averaging Filter

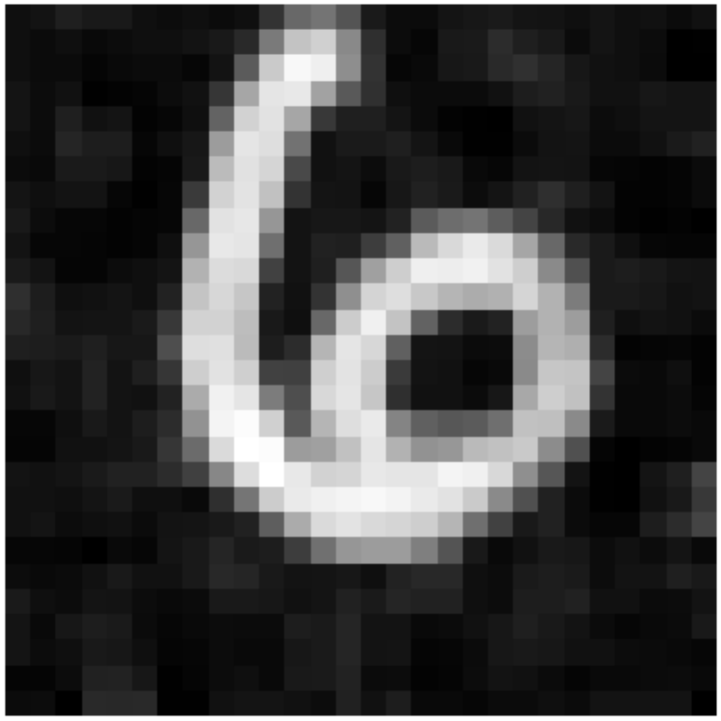
[354]:
```
averaging_kernel = np.ones((3, 3)) / 9
average_img = apply_filter_scipy(noisy_image, averaging_kernel)
plt.imshow(average_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Average Scratch

[355]:
```
def average_filter(image):
    kernelh=3
    kernelw=3
    n=len(image)
    new_image=image.copy()
    padded_image=np.
 ↪pad(image,pad_width=((1,1),(1,1)),mode='constant',constant_values=0)
    for i in range(n):
        for j in range(n):
            neighbourhood=image[i:i+kernelh,j:j+kernelw]
            new_image[i][j]=np.mean(neighbourhood)
    return new_image
```

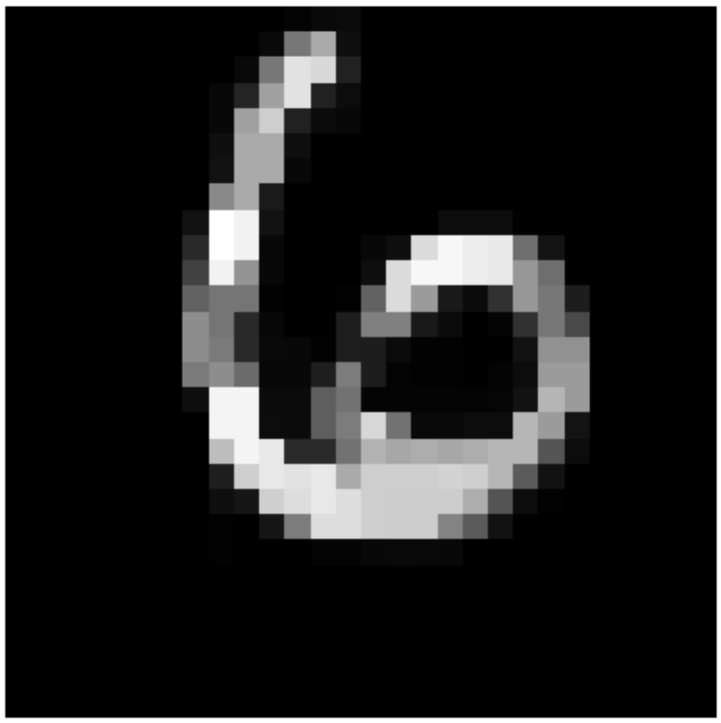
[356]:
```
average_imgs = average_filter(noisy_image)
plt.imshow(average_imgs, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Median Filter

[357]:
```
def median_filter(image):
    kernelh=3
    kernelw=3
    n=len(image)
    new_image=image.copy()
    padded_image=np.
 ↪pad(image,pad_width=((1,1),(1,1)),mode='constant',constant_values=0)
    for i in range(n):
        for j in range(n):
            neighbourhood=image[i:i+kernelh,j:j+kernelw]
            new_image[i][j]=np.median(neighbourhood)
    return new_image
```

[358]:
```
median_img=median_filter(sp_noisy_image)
plt.imshow(median_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```

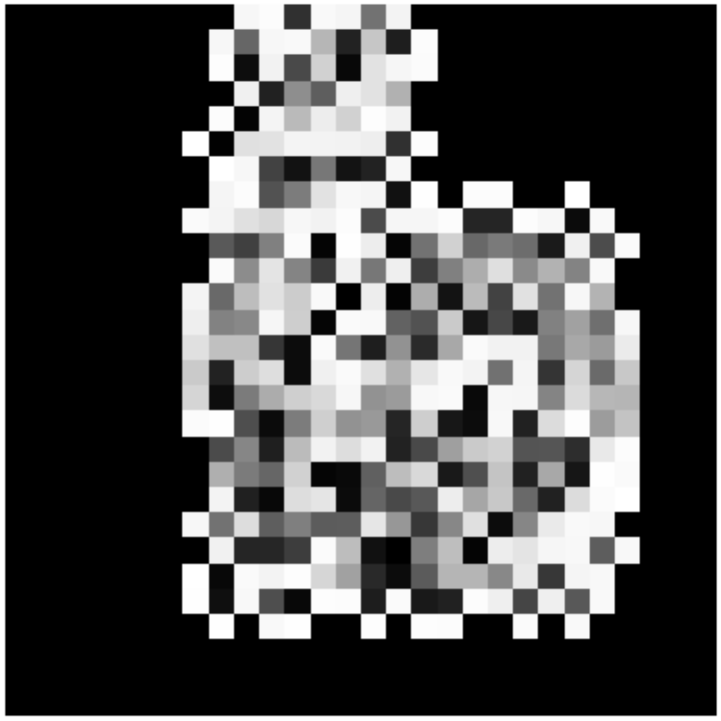


Apply Filter Scratch

[359]:
```
def apply_filter(image, kernel):
    kernelh, kernelw = kernel.shape
    n, m = image.shape
    new_image = image.copy()
    padded_image = np.pad(image, pad_width=((kernelh//2, kernelh//2), (kernelw//
 ↪2, kernelw//2)), mode='constant', constant_values=0)
    for i in range(n):
        for j in range(m):
            neighbourhood = padded_image[i:i+kernelh, j:j+kernelw]
            new_image[i, j] = np.sum(neighbourhood * kernel)
    return new_image
```

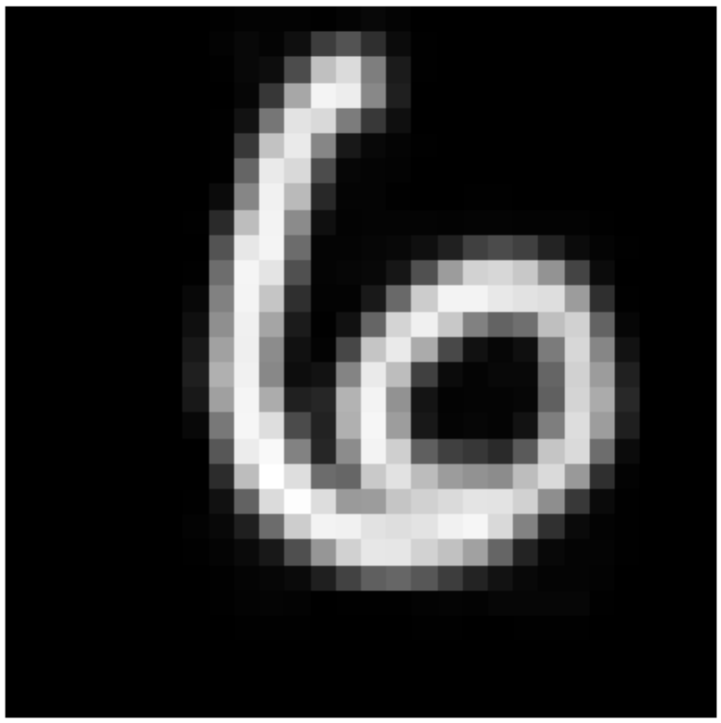
High Pass Filter

[360]:
```
hpkernel=np.array([[0,-1,0],
                   [-1,5,-1],
                   [0,-1,0]])
hp_img=apply_filter(img,hpkernel)
plt.imshow(hp_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



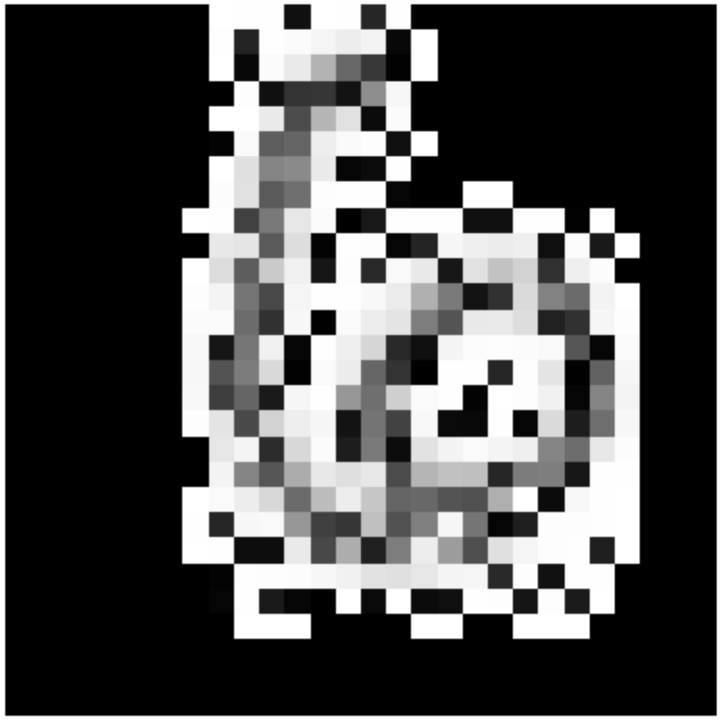
Low Pass Filter

[361]:
```
lpkernel=np.array([[1/16,2/16,1/16],
                   [2/16,4/16,2/16],
                   [1/16,2/16,1/16]])
lp_img=apply_filter(img,lpkernel)
plt.imshow(lp_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



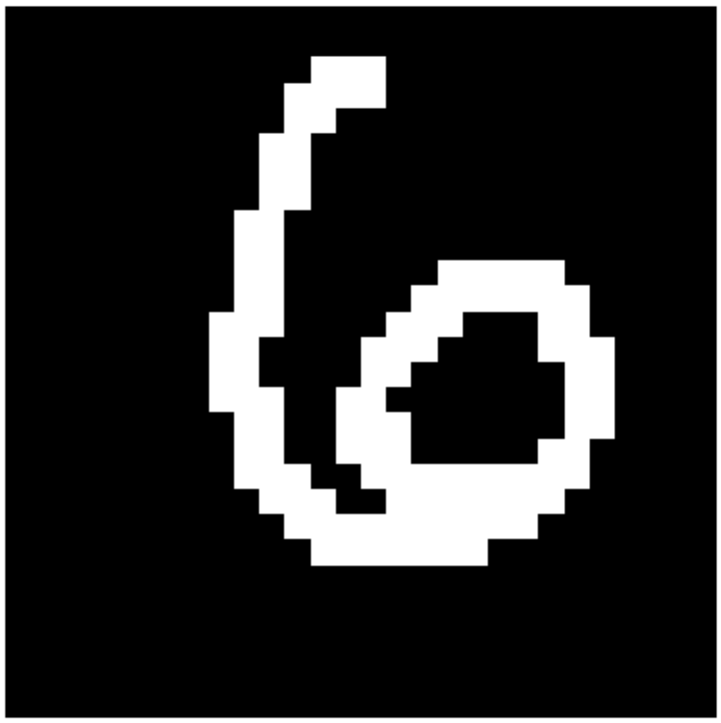
High Boost Filter

[362]:
```
A=2
low_passed=apply_filter(img,lpkernel)
high_passed=img-low_passed
boosted=img+(A-1)*high_passed
boosted_img=np.clip(boosted,0,255)
plt.imshow(boosted_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Binary Mask

[363]:
```
binary_mask = (img > 25).astype(np.uint8)
plt.imshow(binary_mask, cmap='Greys_r')
plt.axis('off')
plt.show()
```



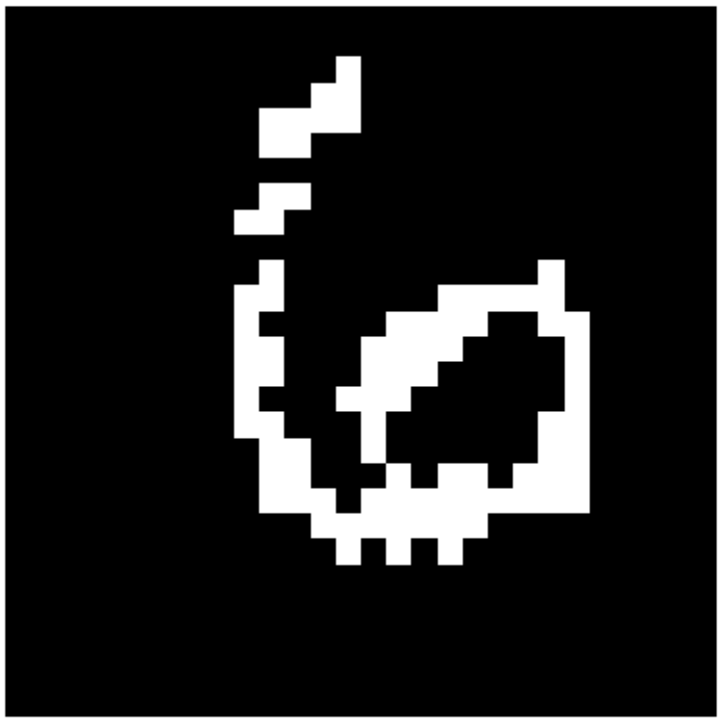
Erosion

[364]:
```
s=np.array([[0,1,0],
            [1,1,1],
            [0,1,0]])
```

[365]:
```
def erosion(image, s):
    image = (image >= 1).astype(np.uint8)
    new_image = np.zeros_like(image)
    n, m = image.shape
    elem_h, elem_w = s.shape
    pad_h, pad_w = elem_h // 2, elem_w // 2
    padded_img = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)),␣
 ↪mode='constant', constant_values=0)
    for i in range(n):
        for j in range(m):
            neighbourhood = padded_img[i:i + elem_h, j:j + elem_w]
            if np.all(neighbourhood[s == 1] == 1):
                new_image[i, j] = 1
    return new_image
```

[366]:
```
eroded_img=erosion(img,s)
plt.imshow(eroded_img, cmap='Greys_r')
plt.axis('off')
```

```
plt.show()
```

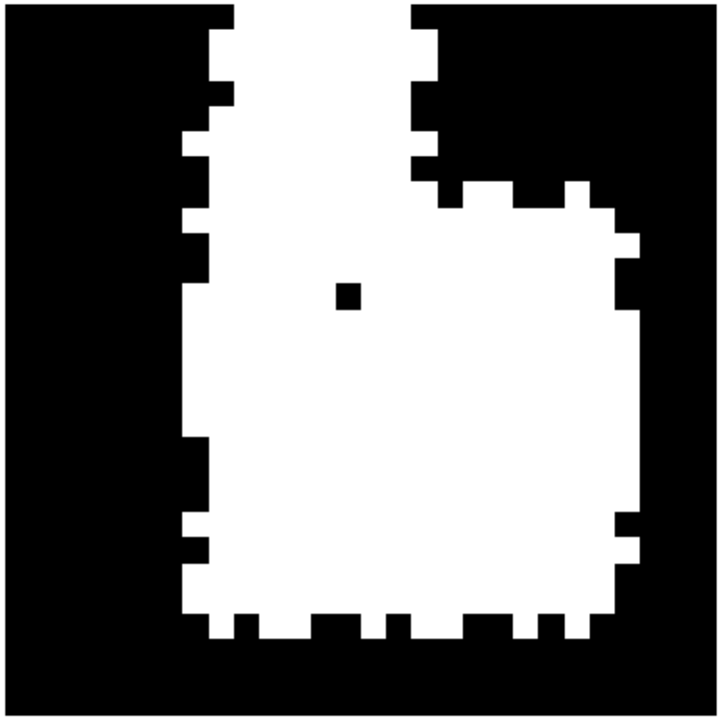


Dilation

[367]:
```
s=np.array([[0,1,0],
            [1,1,1],
            [0,1,0]])
```

[368]:
```
def dilation(image, s):
    image = (image >= 1).astype(np.uint8)
    new_image = np.zeros_like(image)
    n, m = image.shape
    elem_h, elem_w = s.shape
    pad_h, pad_w = elem_h // 2, elem_w // 2
    padded_img = np.pad(image, ((pad_h, pad_h), (pad_w, pad_w)),␣
 ↪mode='constant', constant_values=0)
    for i in range(n):
        for j in range(m):
            neighbourhood = padded_img[i:i + elem_h, j:j + elem_w]
            if np.any(neighbourhood[s == 1] == 1):
                new_image[i, j] = 1
    return new_image
```

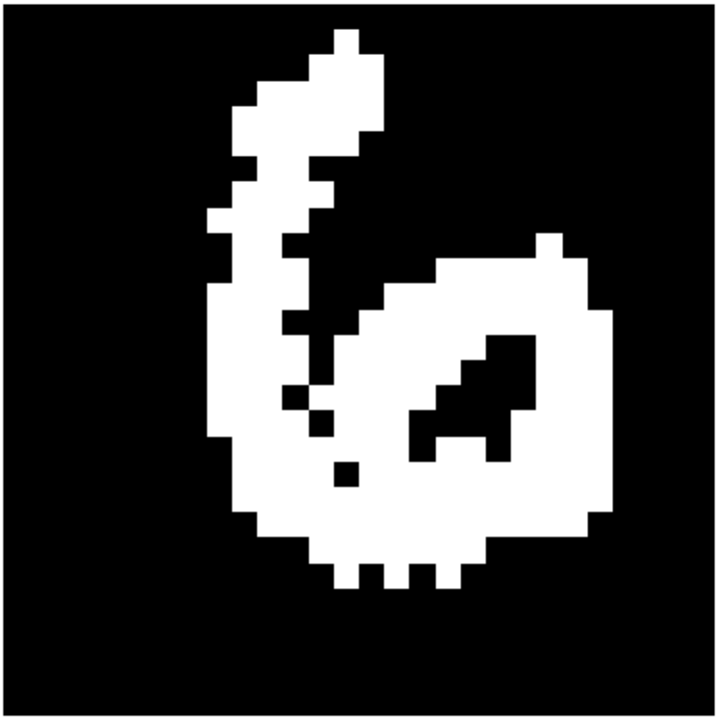
[369]:
```
dilated_img=dilation(img,s)
plt.imshow(dilated_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Opening

[370]:
```
def opening(img,s):
    return dilation(erosion(img,s),s)
```

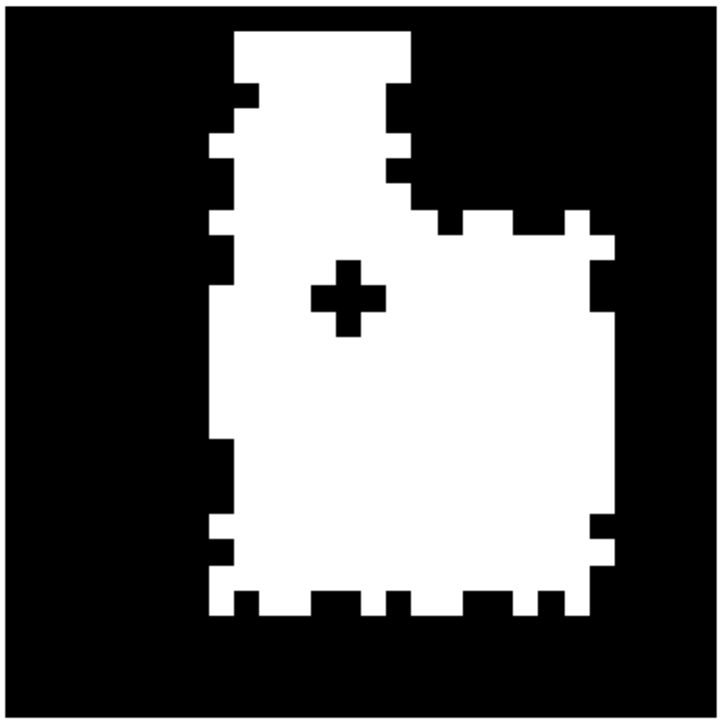
[371]:
```
open_img=opening(img,s)
plt.imshow(open_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```



Closing

[372]:
```
def closing(img,s):
    return erosion(dilation(img,s),s)
```

[373]:
```
close_img=closing(img,s)
plt.imshow(close_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```

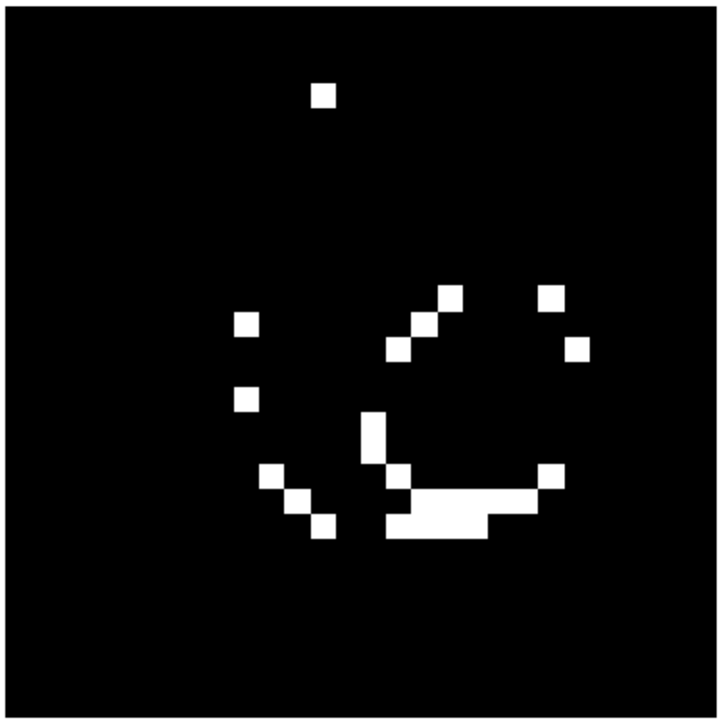


Hit and Miss

[374]:
```
s1=np.array([[0,1,0],
             [1,1,1],
             [0,1,0]])
s2=np.array([[1,0,1],
             [0,1,1],
             [0,0,0]])
```

[389]:
```
def hit_and_miss(img,s1,s2):
    cimg=~img
    e1=erosion(img,s1)
    e2=erosion(cimg,s2)
    new_img=e1&e2
    return new_img
```

[390]:
```
hm_img=hit_and_miss(binary_mask,s1,s2)
plt.imshow(hm_img, cmap='Greys_r')
plt.axis('off')
plt.show()
```

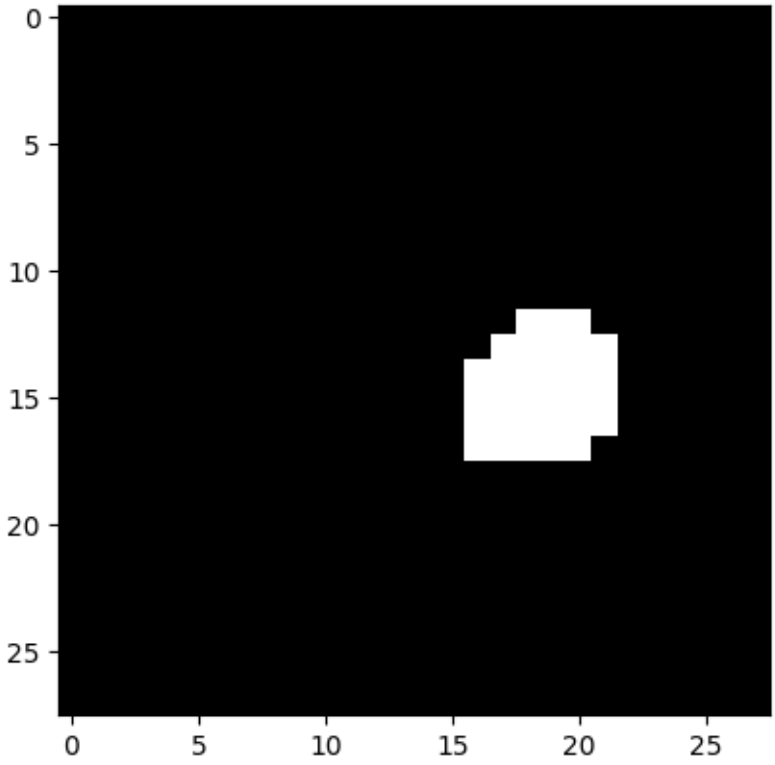


Region Growing

[377]:
```
def region_growing(image, seed, threshold):
    m, m= image.shape
    region_mean=float (image [seed])
    region_size = 1
    output_image = np.zeros((m, m), dtype=np.uint8)
    region_points = [seed]
    processed_points = set(region_points)
    while region_points:
        new_points = []
        for point in region_points:
            x, y = point
            for dx, dy in [(-1, 0), (1, 0), (0, 1), (0, 1)]:
                nx, ny=x+dx, y+dy
                if 0 <= nx < m and 0 <= ny < m and (nx, ny) not in␣
 ↪processed_points:
                    processed_points.add((nx, ny))
                    pixel_value=image[nx, ny]
                    if abs(pixel_value-region_mean) < threshold:
                        new_points.append((nx, ny))
                        region_mean = ((region_mean*region_size + pixel_value) /
 ↪ (region_size + 1))
```

```
                        region_size += 1
                        output_image [nx, ny] = 255
        region_points = new_points
    return output_image
```

[378]:
```
seed_point=(15,15)
threshold=50
grown_region=region_growing(img,seed_point,threshold)
plt.imshow(grown_region, cmap='Greys_r')
plt.show()
```



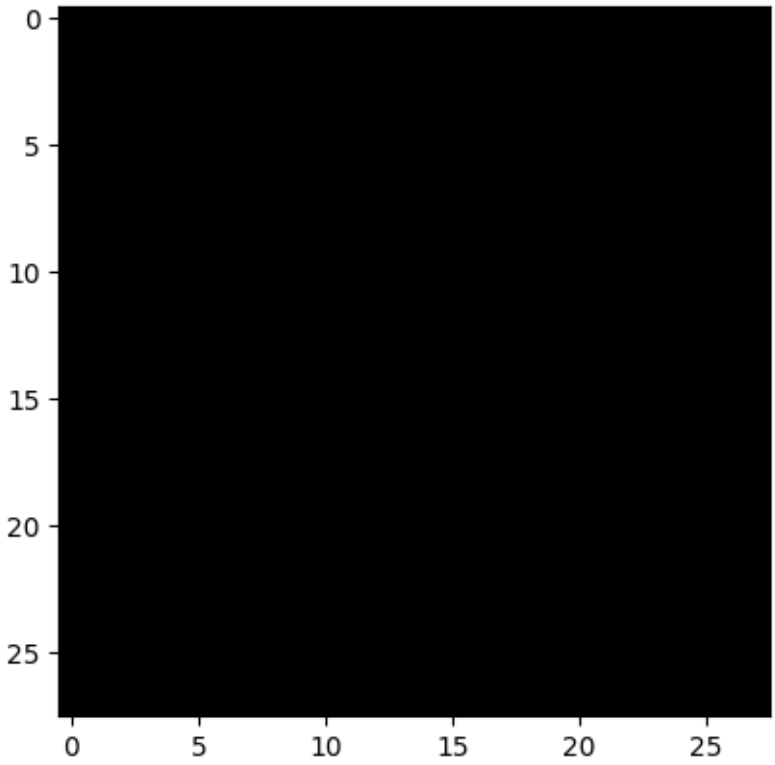
Splitting and Merging

[379]:
```
def merge(regions):
    while True:
        merged=False
        new_regions=[]
        while regions:
            p=regions.pop()
            was_merged=False
            for i in range(len(new_regions)):
                if abs(np.mean(new_regions[i])-np.mean(p))<0:
```

```
                    new_regions[i]=np.vstack([new_regions[i],p])
                    was_merged=True
                    merged=True
                    break
                if not was_merged:
                    new_regions.append(p)
        regions=new_regions
        if not merged:
            break
    return regions
```

[380]:
```
def split_and_merge(image,num_regions):
    m,n=image.shape
    step=m//num_regions
    regions=[np.arange(i,min(i+step,m)) for i in range(0,m,step)]
    regions=merge(regions)
    new_image=np.zeros_like(image)
    for region in regions:
        for row in region:
            new_image[i,:]=((np.mean(image[region,:])-image[row,:])<20)*255
    return new_image
```

[381]:
```
sm_img=split_and_merge(img,10)
plt.imshow(sm_img, cmap='Greys_r')
plt.show()
```



Apply Filter

[382]:
```
from numpy.fft import fft2, ifft2, fftshift, ifftshift
```

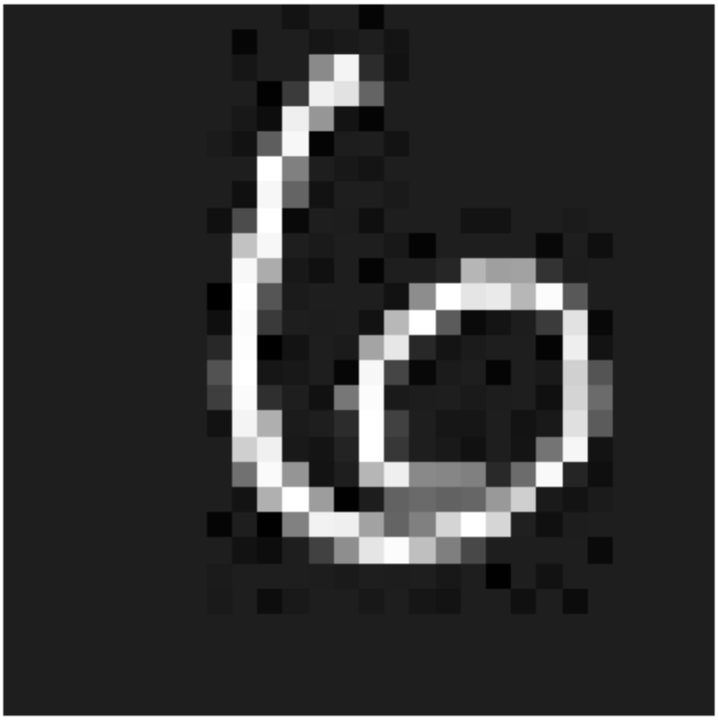
[383]:
```
def apply_filter(image,H):
    fourier_transform = fft2(image)
    f_transform_shifted = fftshift(fourier_transform)
    f_filtered = f_transform_shifted * H
    f_filtered_shifted_back = ifftshift(f_filtered)
    filtered_image = np.abs(ifft2(f_filtered_shifted_back))
    return filtered_image
```

D Formula

[384]:
```
rows, cols = img.shape
u = np.fft.fftfreq(cols, 1.0)
v = np.fft.fftfreq(rows, 1.0)
U, V = np.meshgrid(u, v)
D = np.sqrt(U**2 + V**2)
```

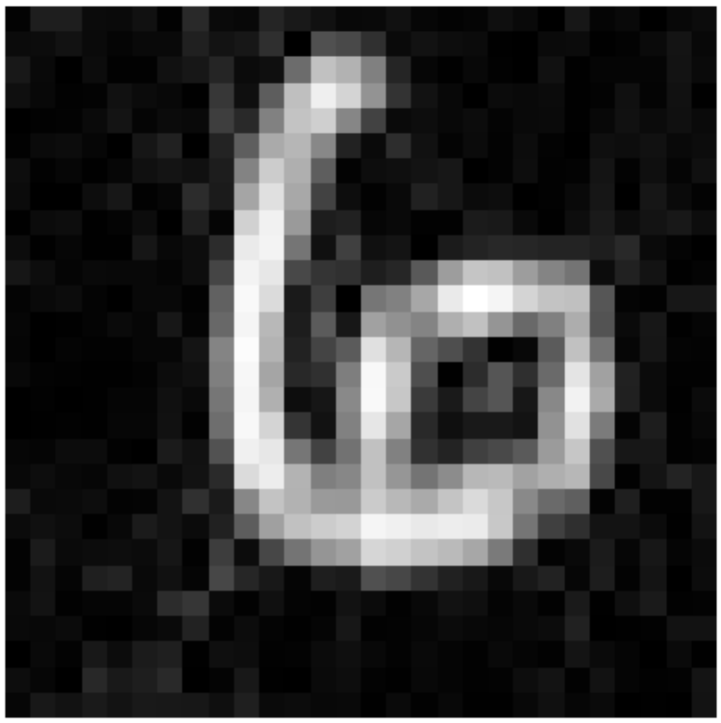
ILPF

[385]:
```
cutoff=0.69
H = np.zeros_like(D)
H[D <= cutoff] = 1
ilpf=apply_filter(img,H)
plt.imshow(ilpf, cmap='Greys_r')
plt.axis('off')
plt.show()
```



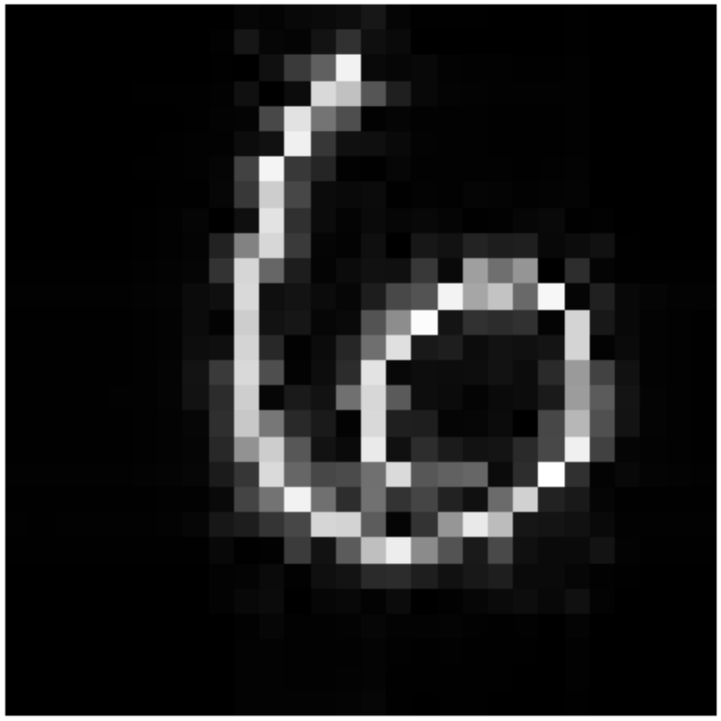
IHPF

[386]:
```
cutoff=0.5
H = np.ones_like(D)
H[D <= cutoff] = 0
ihpf=apply_filter(img,H)
plt.imshow(ihpf, cmap='Greys_r')
plt.axis('off')
plt.show()
```



GLPF

[387]:
```
sigma=0.4
H = np.exp(-(D**2) / (2 * (sigma**2)))
glpf=apply_filter(img,H)
plt.imshow(glpf, cmap='Greys_r')
plt.axis('off')
plt.show()
```



[388]:
```
sigma=0.4
H = 1-np.exp(-(D**2) / (2 * (sigma**2)))
ghpf=apply_filter(img,H)
plt.imshow(ghpf, cmap='Greys_r')
plt.axis('off')
plt.show()
```