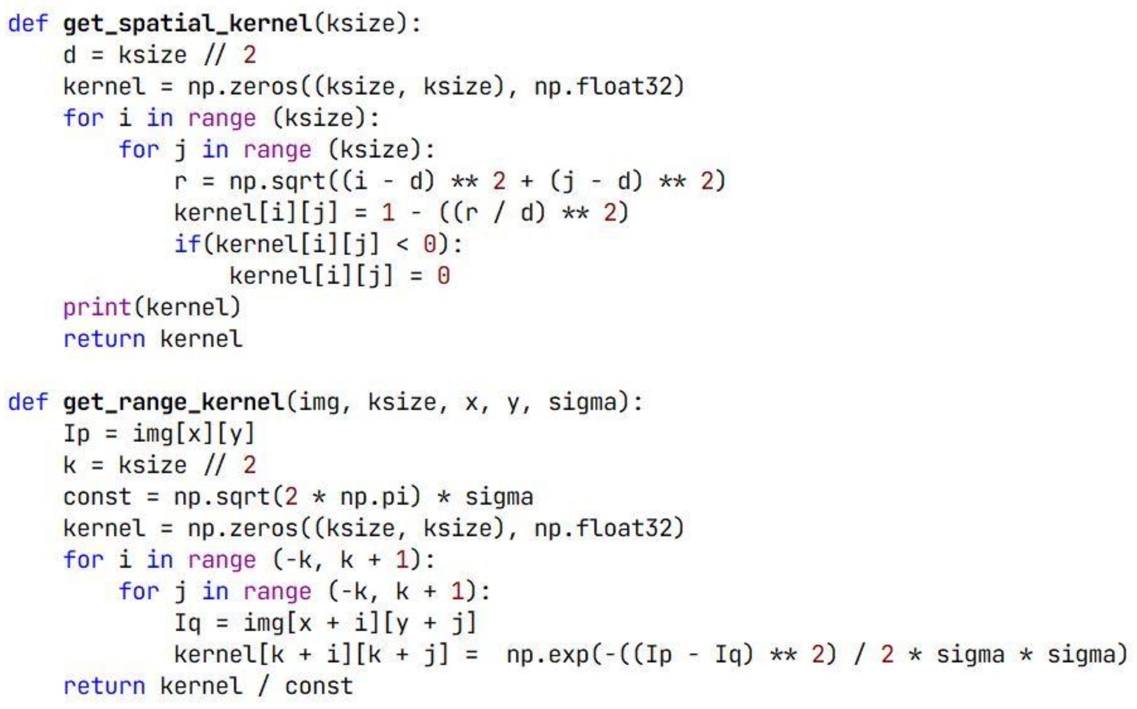
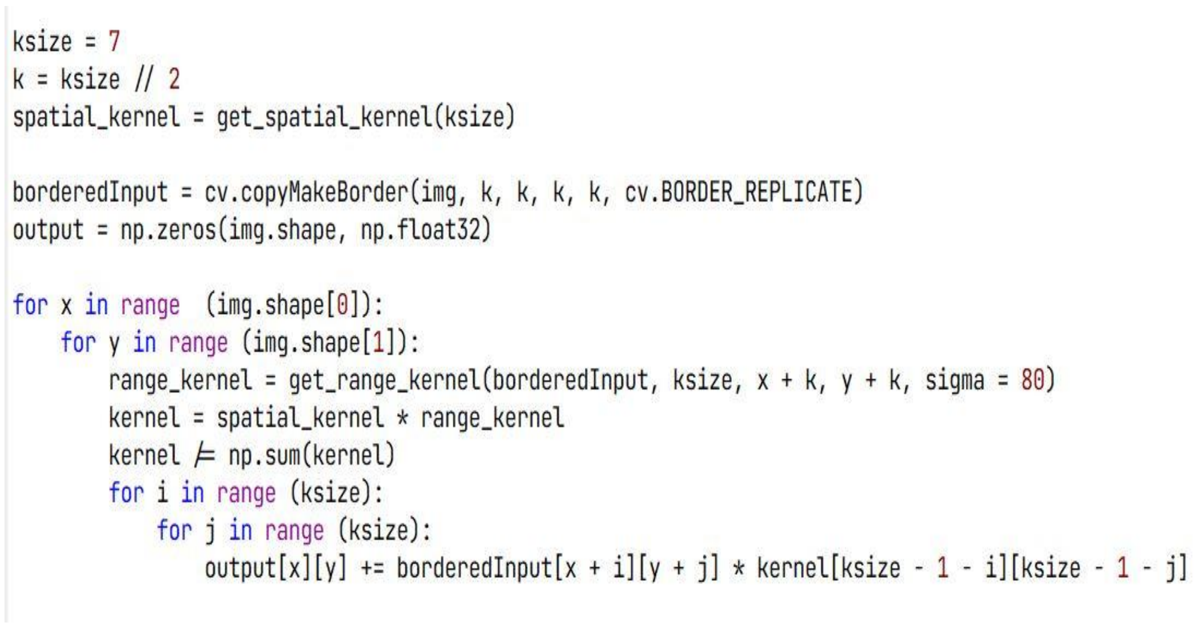
Code For Bilateral Filtering using Epanechnikov Distribution for Spatial Domain



```python
def get_spatial_kernel(ksize):
    d = ksize // 2
    kernel = np.zeros((ksize, ksize), np.float32)
    for i in range (ksize):
        for j in range (ksize):
            r = np.sqrt((i - d) ** 2 + (j - d) ** 2)
            kernel[i][j] = 1 - ((r / d) ** 2)
            if(kernel[i][j] < 0):
                kernel[i][j] = 0
    print(kernel)
    return kernel

def get_range_kernel(img, ksize, x, y, sigma):
    Ip = img[x][y]
    k = ksize // 2
    const = np.sqrt(2 * np.pi) * sigma
    kernel = np.zeros((ksize, ksize), np.float32)
    for i in range (-k, k + 1):
        for j in range (-k, k + 1):
            Iq = img[x + i][y + j]
            kernel[k + i][k + j] =  np.exp(-((Ip - Iq) ** 2) / 2 * sigma * sigma)
    return kernel / const
```



```python
ksize = 7
k = ksize // 2
spatial_kernel = get_spatial_kernel(ksize)

borderedInput = cv.copyMakeBorder(img, k, k, k, k, cv.BORDER_REPLICATE)
output = np.zeros(img.shape, np.float32)

for x in range  (img.shape[0]):
    for y in range (img.shape[1]):
        range_kernel = get_range_kernel(borderedInput, ksize, x + k, y + k, sigma = 80)
        kernel = spatial_kernel * range_kernel
        kernel /= np.sum(kernel)
        for i in range (ksize):
            for j in range (ksize):
                output[x][y] += borderedInput[x + i][y + j] * kernel[ksize - 1 - i][ksize - 1 - j]
```

Output For Bilateral Filtering

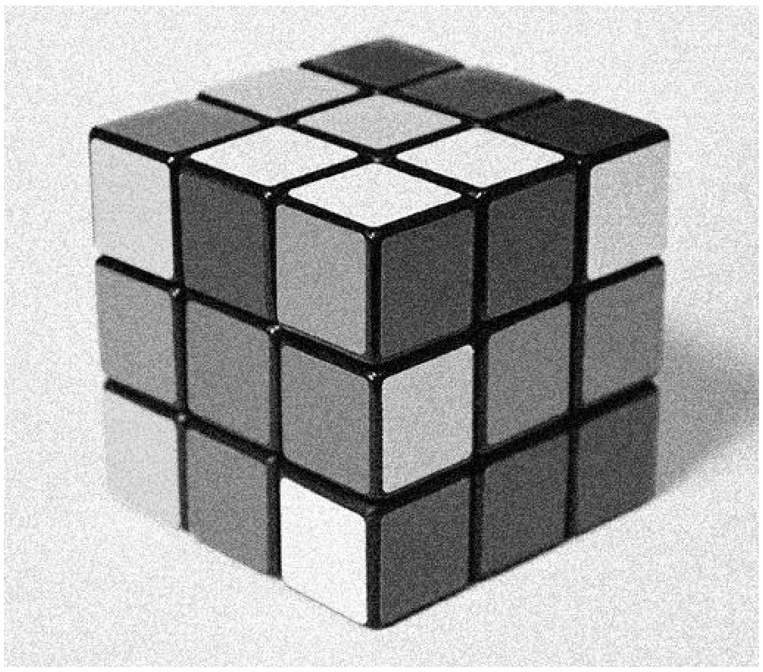


Fig 2.1: Input Image

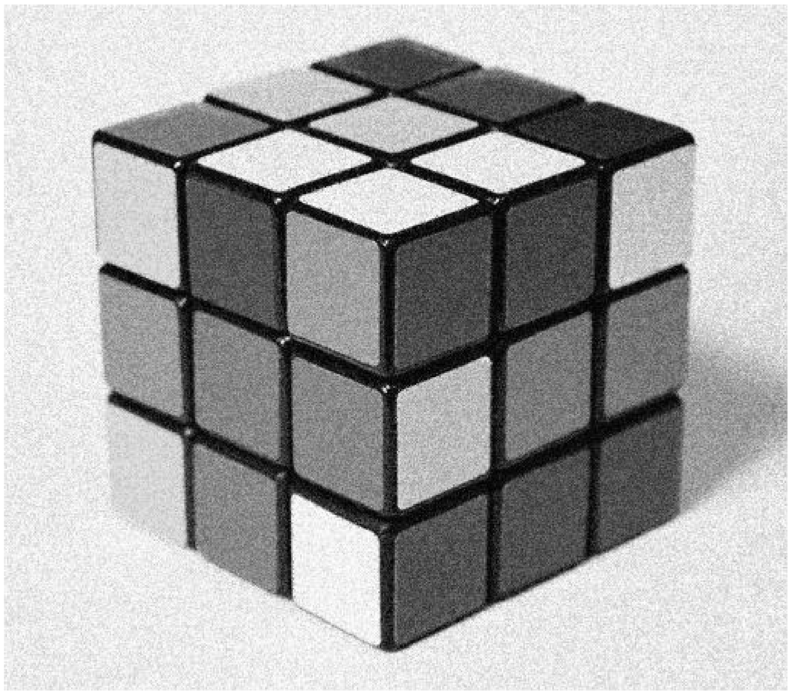


Fig 2.2: Bilateral Filtering Output