תרגיל 3 – Bilateral Filter

שם המגיש – עומר גבאי

ת.ז – 302390984

בתרגיל זה נממש Bilateral Filter, פילטר זה מאוד שימושי משום שהוא מוריד רעש מהתמונה וגם שומר על הקווי מתאר של התמונה (Edges). הפילטר משתמש ב non-linear kernel שלמעשה אומר שבזמן הפעלת הקונבולוציה הוא מחשב עבור כל פיקסל את הקרנל המתאים עבורו תוך התחשבות ב dynamic range של הפיקסלים בסביבתו (העוצמה של הפיקסלים), וגם מתחשב במרחקן האוקלידי של הפיקסלים בקרנל מהפיקסל שעובר עיבוד.

רוב הפעמים נרצה שלפיקסלים רחוקים תהיה השפעה פחותה על הפיקסל המעובד אבל ניתן לשנות זאת אם נרצה להחליק שטחים גדולים בתמונה (נקבל תמונה יותר ציורית..).

שני הפרמטרים שהזכרתי שקובעים את מידת החלקה מסומנים במשוואה על ידי -

- σs - משפיע על מידת ההשפעה של פיקסלים רחוקים אוקלידית מהפיקסל
- σr - משפיע על מידת ההשפעה של פיקסלים עם עוצמה שונה, ככל שנגדיל פרמטר זה נקבל יותר טשטוש של קווי מתאר והפילטר יהיה יותר דומה להחלקה גאוסיאינית רגילה.

מימוש

```
def bilateral_one_pixel(source, x, y, d, sigma_r, sigma_s):
    # === init vars
    filtered_pix = 0
    Wp = 0
    px_intensity = source[x][y]

    # TODO:
    # 1. run on all neighboors (~3 lines)
    # 2. if neighboor out of matrix indices - don't count him in your computation
    # 3. find filtered_pix (~6 lines)
    for row in range(x-d//2,x+1+d//2):
        for col in range(y-d//2 , y+1+d//2):
            if row < 0 or row >= source.shape[0] or col < 0 or col >= source.shape[1]:
                continue
            curr_intensity = source[row][col]
            gs = unnormalized_gaussian(distance(row, col , x, y),sigma_s)
            fr = unnormalized_gaussian(abs(curr_intensity - px_intensity), sigma_r)
            filtered_pix += curr_intensity*gs*fr
            Wp += gs*fr
    # make result uint8
    filtered_pix /= Wp
```

Parameters –

filtered_pix - will hold the value of the processed pixel

px_intensity – the pixel intensity of the pre-processed pixel

d – kernel size (our window) we will add to the calculation only pixels at distance **d** from processed pixel both horizontally and vertically , boundary check added for pixel outside of the original image.

gs – spatial kernel determined by sigma_s , how distant pixels will affect the current pixel

fr – range kernel determined by sigma_r - how intensity differences will be significant

Wp – Normalization factor, after iterating over all pixels in kernel we will divide by this factor

Helper functions

```
def unnormalized_gaussian(x, sigma):
    # TODO: [V] helper function for calculating exponent of (-x) divided by
two sigma (one line)
    y = x/sigma
    return np.exp(-y**2/2)


def distance(x, y, I, j):
    # TODO: [V] get L2 distance function between two points – helper function
for calculating “g_s” (one line)
    delta_x = x-I
    delta_y = y-j
    return np.sqrt(delta_x**2+delta_y**2)
```

Applying the filter and results

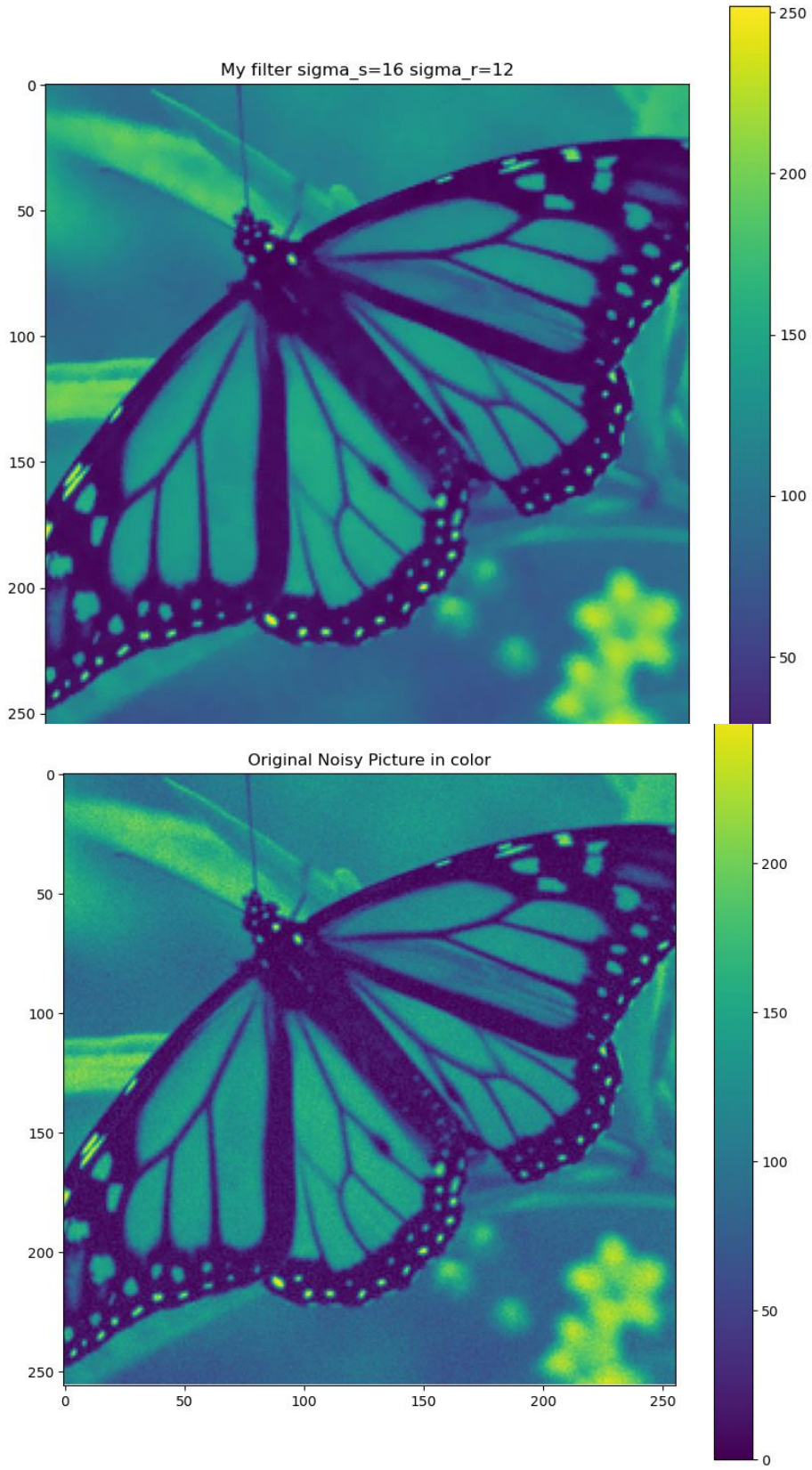
```
def bilateral_filter(source, d, sigma_r, sigma_s):

    # build empty filtered_image
    filtered_image = np.zeros(source.shape, np.uint8)
    # make input float
    source = source.astype(float)
    # d must be odd!
    assert d % 2 == 1, "d input must be odd"

    # TODO: run on all pixels with bilateral_one_pixel
    for x in range(source.shape[0]):
        for y in range(source.shape[1]):
            px = source[x][y]
            filtered_image[x][y] = bilateral_one_pixel(source,x,y,d,sigma_r,sigma_s)
```

Here you can see a comparison of the filtered picture with the original noisy picture.

The noise was reduced significantly and the edges are still preserved unlike gaussian filter (more pictures in the Jupyter notebook).



Canny filter on filtered image

As we can see edges are preserved like we wanted.