

תרגיל בית 1 עיבוד תמונה

מגשים:

אור דינר - 207035809

איתמר - 207931296

שאלה 1 + בונס

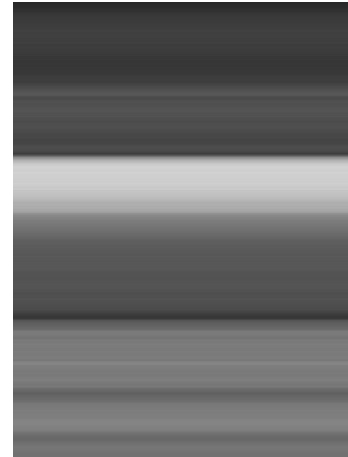
עבור תמונה 1:



כל פיקסל קיבל את
הערך הממוצע של
השורה שלו

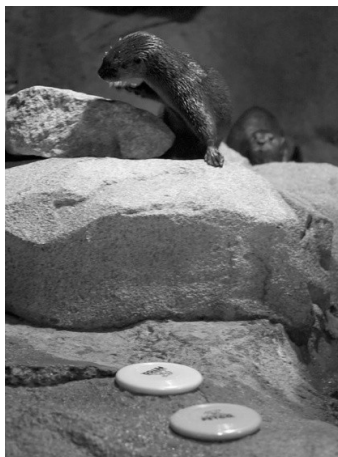


$$MSE = 0.518$$



ראינו בתמונה שלכל שורה יש ערך קבוע לכל השורה לאחר כמה בדיקות מימשנו את הפילטר וראינו באמת שהתמונה מתקבלת מחישוב הממוצע לכל שורה.

עבור תמונה 2:



טשטוש גיאוסייני
`GaussianBlur(image0,
ksize=(11, 11),
sigmaX=15)`

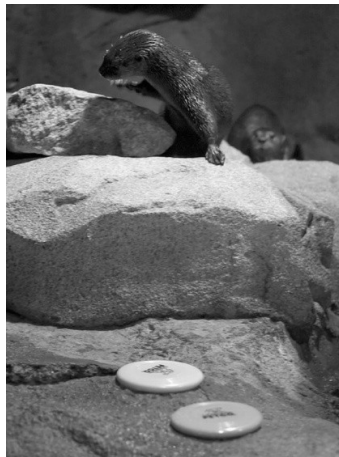


$$MSE = 9.074$$



ראינו שהתמונה מטושטשת בצורה חלקה (מה שמאפיין טשטוש גאוסייני) ולכן מימשנו את הטשטוש הגאוסייני ובדקנו אלו פרמטרים הכי מתאימים.

עבור תמונה 3:



טשטוש חציוני
`medianBlur(image0, 11)`

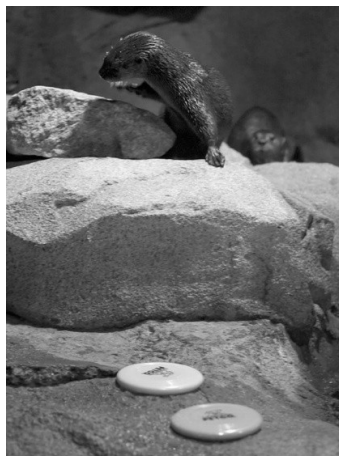


$$MSE = 0.368$$



בתמונה הזו זיהינו טשטוש אבל לא חלק כמו בתמונה 2 והטשטוש "מחולק לאזורים" לכן בדקנו את המימוש של טשטוש חציוני ומצאנו את הפרמטרים שדייקו את הטשטוש.

עבור תמונה 4:



טשטוש עם גרעין אנכי
`filter2D(image0`
`, -1, kernel)`



$$MSE = 9.394$$



בתמונה זו זיהינו שיש טשטוש שונה שנראה שנמשך יותר כלפי מטה לכן מימשנו טשטוש עם גרעין אנכי כך שהגרעין הוא מטריצה 15×15 כשכל פיקסל בטור האמצעי בעל ערך $\frac{1}{15}$ והשאר 0.

עבור תמונה 5:

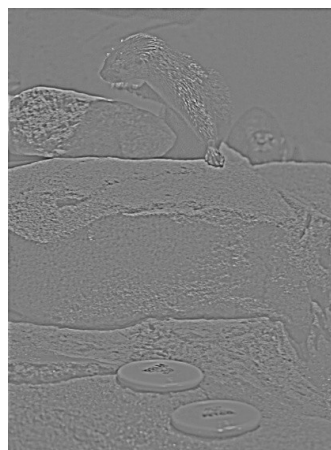


החלק החד של
התמונה

$$\text{Image5} = \text{image0} - \text{image2}$$



$$MSE = 10.144$$



זיהינו שבדומה לתמונה במצגת השיעור זה נראה כמו החלק החדד של התמונה והוא מתקבל מהחסרה של תמונה 2 (המטושטשת) מהתמונה המקורית.

עבור תמונה 6:



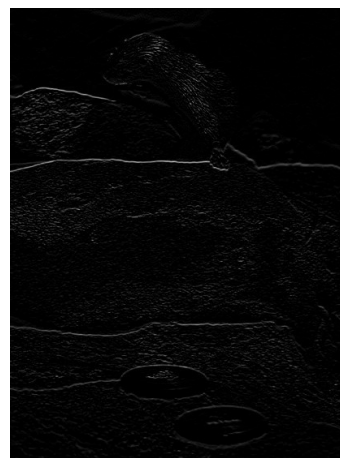
זיהוי קצוות אופקי

= גרעין

-0.26	-0.52	-0.26
0	0	0
0.26	0.52	0.26

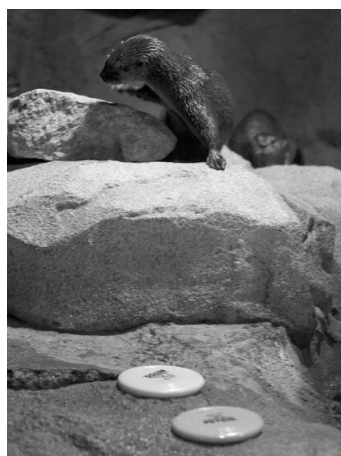


$$MSE = 3.934$$



זיהינו שבדומה לתמונה במצגת התרגול התמונה נראית כמו זיהוי קצוות אופקי (הקצוות האנכיים פחות מודגשים) במימוש בדקנו מספר ערכים למטריצה וקיבלנו שזו המתאימה ביותר.

עבור תמונה 7:



הזזה מעגלית של
התמונה בחצי מגובה



$$MSE = 1.182$$



בתמונה הזו נראה שהייתה הזזה מעגלית של התמונה בציר האנכי כחצי מגובה התמונה. בדקנו באיזה גובה מתחלפת התמונה בין למעלה למטה וראינו שזה קורה בדיוק באמצע (שורה 399).

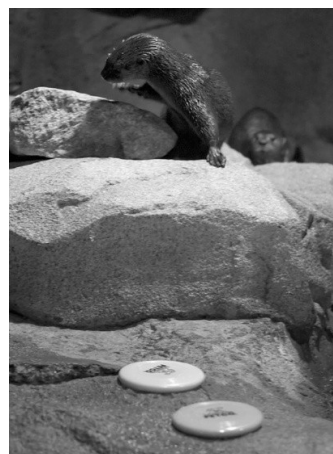
עבור תמונה 8:



ללא שינוי רק מעבר
לשחור לבן

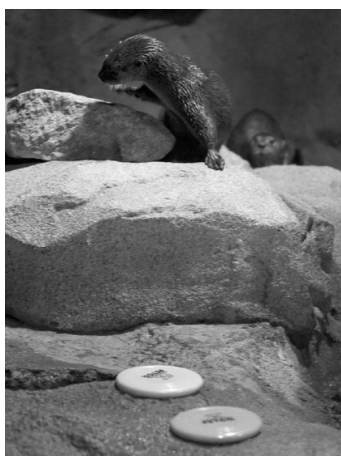


$$MSE = 0.347$$



בתמונה זו לא הבחנו בשינוי התמונה נראית כמו התמונה המקורית רק בשחור לבן. המרנו את התמונה לשחור לבן באמצעות "cv2.COLOR_BGR2GRAY" בזמן טעינת התמונה.

עבור תמונה 9:



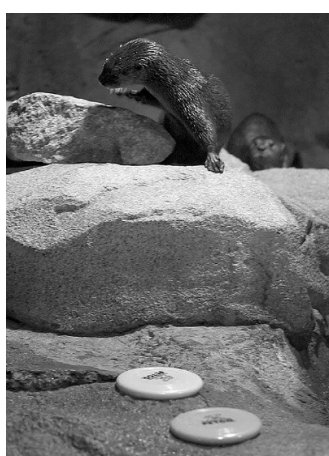
חידוד התמונה

= גרעין

0	-0.585	0
-0.585	3.34	-0.585
0	-0.585	0

→

$MSE = 16.594$



התמונה נראתה לנו מחודדת יותר, בעיקר ניתן להבחין באזור הפרווה הבהקת. ניסינו גרעינים שונים לחידוד התמונה וקיבלנו את התוצאה הטובה ביותר מהגרעין הזה.

שאלה 2

סעיף א': מימוש פילטר ביליטרלי.

בחלק הראשון של הפונקציה "אתחלנו משתנים", ווידאנו שתמונת הקלט היא מסוג np.float64, שמרנו את מספר השורות והעמודות, אתחלנו את תמונת הפלט להיות בגודל של התמונה שקיבלנו רק שערכי כל הפיקסלים הם 0, יצרנו מסכה גאומטרית לפי הרדיוס הנתון ו-stdSpatial וריפדנו את התמונה בשיקוף של הקצוות על מנת שנוכל להפעיל את המסכה על כל התמונה המקורית.

```
def clean_Gaussian_noise_bilateral(im, radius, stdSpatial, stdIntensity): 3 usages
    # Your code goes here
    # Ensure the input image is a float64 for precise calculations
    im = im.astype(np.float64)

    # Get the dimensions of the input image
    rows, cols = im.shape

    # Initialize the output image
    cleanIm = np.zeros_like(im)

    # Create spatial Gaussian mask (gs)
    x, y = np.meshgrid(*[np.arange(-radius, radius + 1), np.arange(-radius, radius + 1)])
    gs = np.exp(-(x ** 2 + y ** 2) / (2 * stdSpatial ** 2))

    # Pad the image to handle border pixels
    padded_im = np.pad(im, pad_width=radius, mode='reflect')
```

בחלק השני של הפונקציה עברנו על כל פיקסל בתמונה ויצרנו לו חלון בהתאם לרדיוס, חישבנו את g_i כמו שמתואר בתרגיל, שילבנו את המסכות, נרמלנו את המסכה המשולבת ולבסוף חישבנו את הערך שצריך להיות בפיקסל בתמונה הנקייה. בסוף הפונקציה נמיר את תמונת הפלט ל-`np.uint8` ונחזיר אותה.

```
# Loop over each pixel in the image
for i in range(rows):
    for j in range(cols):
        # Extract the local window
        window = padded_im[i:i + 2 * radius + 1, j:j + 2 * radius + 1]

        # Compute intensity Gaussian mask (gi)
        gi = np.exp(-((window - im[i, j]) ** 2) / (2 * stdIntensity ** 2))

        # Compute combined mask
        combined_mask = gs * gi

        # Normalize the combined mask
        combined_mask /= np.sum(combined_mask)

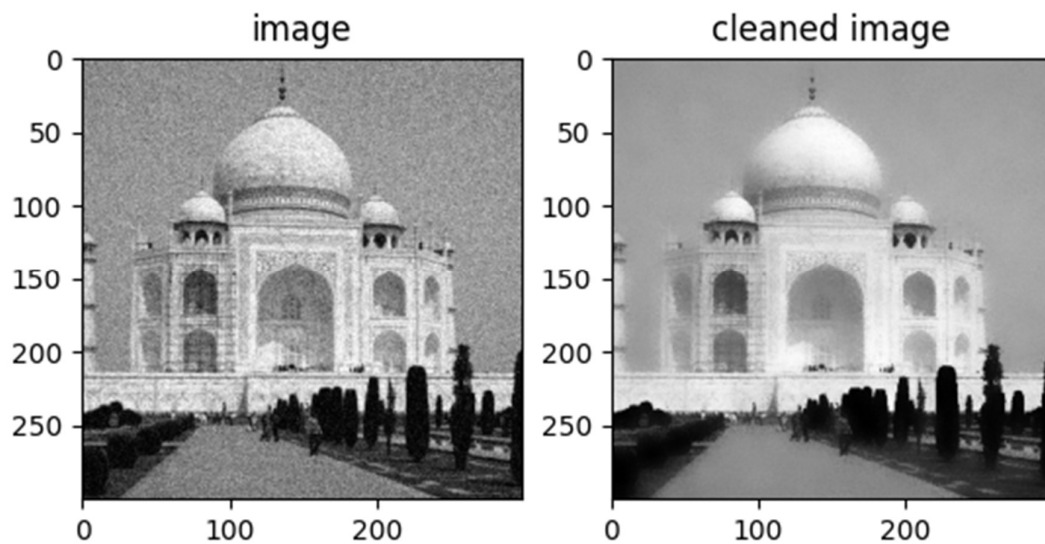
        # Compute the filtered value for the current pixel
        cleanIm[i, j] = np.sum(combined_mask * window)

# Convert the result back to uint8
cleanIm = np.clip(cleanIm, a_min: 0, a_max: 255).astype(np.uint8)

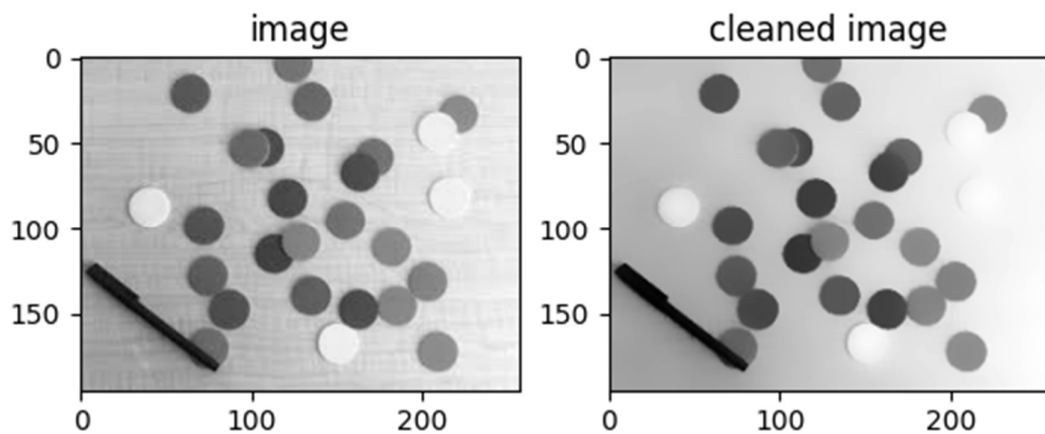
return cleanIm
```

סעיף ב': שיפור התמונות המורעשות

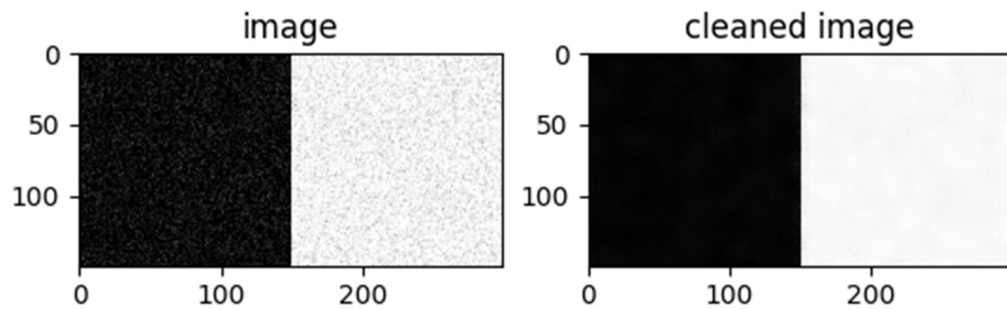
את תמונת ה-taj יהיה קשה לשפר בצורה שלא תפגע בשאר הפרטים בתמונה מכיוון שיש הרבה כאלה. את תמונת הכדורים יהיה ניתן לשפר אך לא בצורה מושלמת מכיוון שיש פרטים בתמונה שעלולים להעלם לאחר מימוש הפילטר. את תמונת הריבוע השחור והלבן ניתן יהיה לשפר בצורה טובה יחסית מכיוון שאין בה הרבה פרטים, נוכל להעביר לפונקציה ערכים גבוהים יותר שיגרמו להיעלמות כמעט מוחלטת של הרעש.



עבור תמונת ה-*taj* בחרנו ב- $radius=9$, $stdSpatial=8.5$, $stdIntensity=45$ לפי ניסוי וטעיה של ערכים שונים הגענו למסקנה שאלו הערכים שנותנים תמונה כמעט ולא מטושטשת ושחלקים מרכזיים מהתמונה לא נעלמו.



בתמונת הכדורים בחרנו ב- $radius=12$, $stdSpatial=9$, $stdIntensity=30$ לפי ניסוי וטעיה של ערכים שונים. הגענו למסקנה שאלו הערכים שנותנים תמונה חלקה כמעט ללא רעשים. ואכן התמונה יצאה חלקה ללא המרקם של הרקע וללא רעשים כמעט.



עבור תמונת הריבוע השחור והלבן בחרנו ב- $radius=7$, $stdSpatial=30$, $stdIntensity=90$ ערכים גבוהים יחסית במיוחד ב- $stdSpatial$, $stdIntensity$ על מנת להעלים את הרעשים בלי הרבה פחד לאבד את המידע בתמונה. התמונה הנקייה יצאה כמעט ללא רעשים ושוחזרה בצורה הטובה ביותר.

לסעיף הזה כתבנו פונקציה שתציג את התמונה הנקייה לעומת המקורית.

```
def plot_images(original_image, clean_image): 3 usages
    # Plots the 2 images the cleaned one next to the original one
    plt.subplot(121)
    plt.title("image")
    plt.imshow(original_image, cmap='gray')

    plt.subplot(122)
    plt.title("cleaned image")
    plt.imshow(clean_image, cmap='gray')

    plt.show()
```

וכתבנו קוד שיטען את התמונות, ינקה אותן בעזרת הפונקציה שמימשנו בסעיף א' והערכים השונים לכל תמונה, יציג אותן ולבסוף ישמור את התמונות הנקיות בתיקייה.

```
# Load the noisy images
taj_image = cv2.imread("taj.jpg", cv2.IMREAD_GRAYSCALE)
balls_image = cv2.imread("balls.jpg", cv2.IMREAD_GRAYSCALE)
noisy_gray_image = cv2.imread("Noisy6rayImage.png", cv2.IMREAD_GRAYSCALE)

# Apply the bilateral filter
clean_taj = clean_Gaussian_noise_bilateral(taj_image, radius=9, stdSpatial=8.5, stdIntensity=45)
clean_gray_image = clean_Gaussian_noise_bilateral(noisy_gray_image, radius=7, stdSpatial=30, stdIntensity=90)
clean_balls = clean_Gaussian_noise_bilateral(balls_image, radius=12, stdSpatial=9, stdIntensity=30)

# Show the images
plot_images(taj_image, clean_taj)
plot_images(balls_image, clean_balls)
plot_images(noisy_gray_image, clean_gray_image)

# Write the images to the folder
cv2.imwrite(filename="clean_taj.jpg", clean_taj)
cv2.imwrite(filename="clean_gray_image.jpg", clean_gray_image)
cv2.imwrite(filename="clean_balls.jpg", clean_balls)
```


שאלה 3

סעיף א': שיפור התמונה המורעשת על ידי פילטרים.

בסעיף הזה מימשנו את שיפור התמונה על ידי שימוש בטשטוש חציוני עם גרעין בגודל 5 ולאחר מכן שימוש בפילטר ביליטרי עם הערכים $d=4$, $\sigma_{\text{Color}}=30$, $\sigma_{\text{Space}}=30$ "להחלקת התמונה". בסוף שמרנו את התמונה המשופרת והצגנו אותה בעזרת פונקציה דומה לפונקציית ההצגה משאלה קודמת. את הערכים של הפרמטרים קבענו לאחר ניסוי וטעיה על התמונה המקורית.

```
# Part A

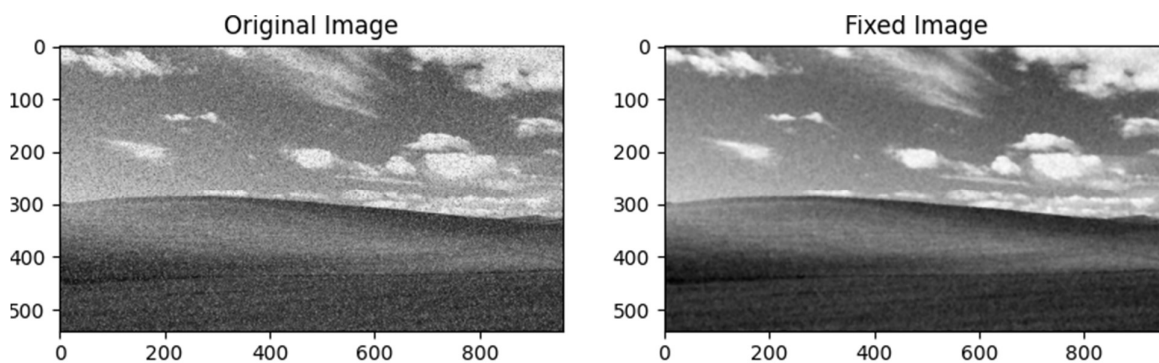
# Load the image
image = cv2.imread('broken.jpg', cv2.IMREAD_GRAYSCALE)

# Apply median filtering
median_filtered = cv2.medianBlur(image, ksize: 5) # Kernel size = 5x5

# Apply bilateral filter
bilateral_filtered = cv2.bilateralFilter(median_filtered, d: 4, sigmaColor: 30, sigmaSpace: 30)

# Save and display results
cv2.imwrite( filename: "fixed_broken_part_a.jpg", bilateral_filtered)

# Show the images
plot_images(image, bilateral_filtered)
```



סעיף ב': שיפור התמונה על ידי מיצוע של 100 תמונות.

בסעיף הזה טענו את 100 התמונות, חישבנו את הממוצע שלהן, "החלקנו" מעט את התמונה שהתקבלה על ידי פילטר בילטרלי עם הערכים $d=3$, $\sigma_{\text{Color}}=20$, $\sigma_{\text{Space}}=20$ שהצבנו לאחר ניסוי וטעיה, שמרנו אותה והצגנו אותה בדומה לסעיף א'.

```
# Part B

# Load the noised images
noised_images = np.load("noised_images.npy")

# Perform pixel-wise averaging
averaged_image = np.mean(noised_images, axis=0).astype(np.uint8)

# Post-processing with bilateral filter
final_image = cv2.bilateralFilter(averaged_image, d=3, sigmaColor=20, sigmaSpace=20)

# Save and display results
cv2.imwrite(filename="fixed_broken_part_b.jpg", final_image)

# Show the images
plot_images(image, final_image)
```

