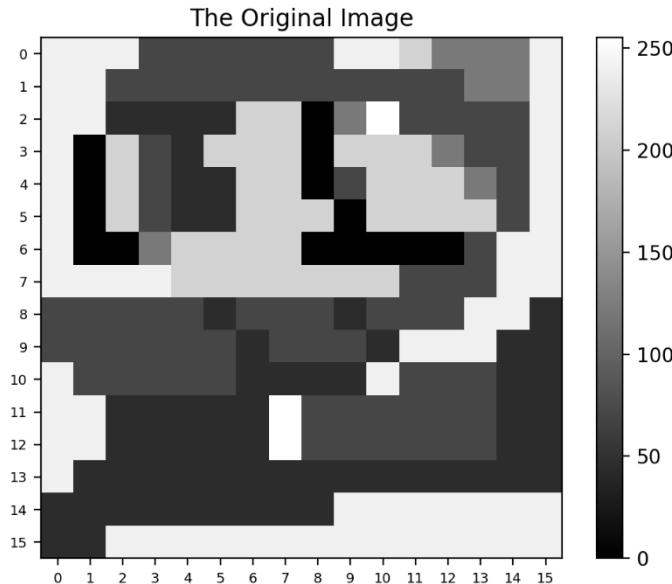


חלק ראשון

שאלה 1:

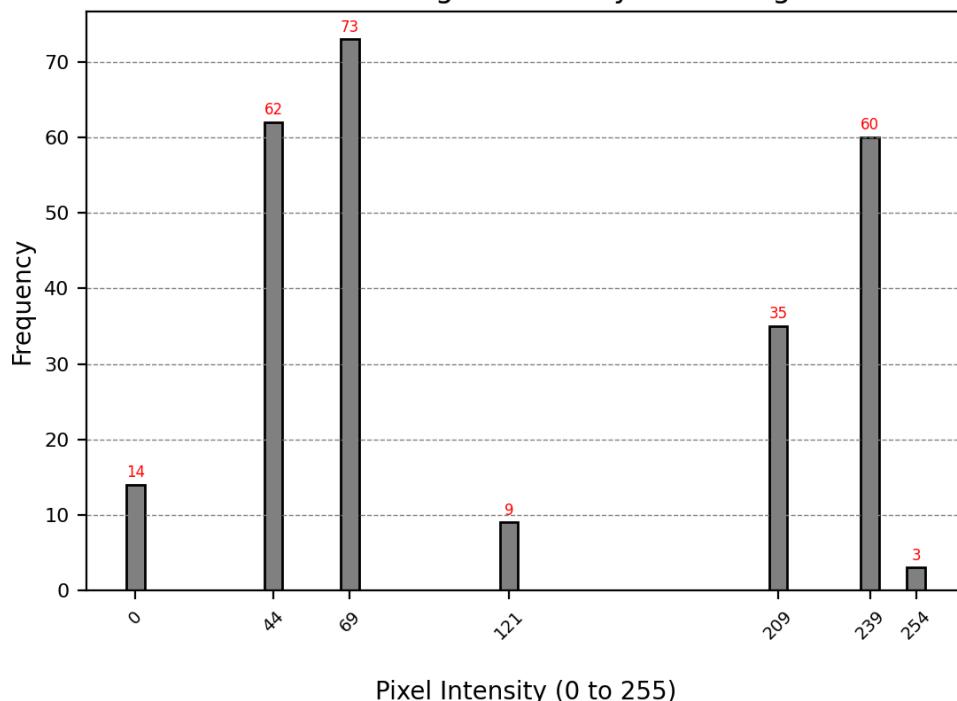
רוב הקוד בשאלת זו הוא ויזואלייזציה, לכן, לא צירפנו את הקוד ב-PDF אלא רק בתיקייה שהגשנו.
נתבונן בתמונה המקורי:



a. היסטוגרמת התמונה הנתונה:

Pixel Intensity (0-255)	0	44	69	121	209	239	254
Frequency	14	62	73	9	35	60	3

Part A - Histogram of Grayscale Image



d. הפעלת מסנן חציו בגודל 3x3 על הפיקסלים שבמסגרת הכתומה:

נפעיל מסנן חציו בגודל 3x3 באופן הבא:

(הנחה שלקחנו – כאשר הפעלנו את המסנן על הגבולות של האזור החתום, נעזרנו בפיקסלים של שאר התמונה ולא ריפדנו בערכים אחרים)

1 סעיף

240	240	240	70	70	70	70	70	240	240	210	122	122	122	240	
240	240	70	70	70	70	70	70	70	70	70	70	122	122	240	
240	240	45	45	45	45	210	210	0	122	255	70	70	70	240	
240	0	210	70	45	45	210	210	0	210	210	210	122	70	240	
240	0	210	70	45	45	210	210	0	70	210	210	210	122	70	240
240	0	210	70	45	45	210	210	210	0	210	210	210	210	70	240
240	0	0	122	210	210	210	210	0	0	0	0	0	70	240	240
240	240	240	210	210	210	210	210	210	210	70	70	70	70	240	240
70	70	70	70	70	45	70	70	45	70	70	70	240	240	45	
70	70	70	70	70	70	45	70	70	70	45	240	240	240	45	45
240	70	70	70	70	70	45	45	45	240	70	70	70	45	45	
240	240	45	45	45	45	45	255	70	70	70	70	70	45	45	
240	240	45	45	45	45	45	255	70	70	70	70	70	70	45	45
240	45	45	45	45	45	45	45	45	240	240	240	240	240	240	45
45	45	45	45	45	45	45	45	45	240	240	240	240	240	240	45
45	45	240	240	240	240	240	240	240	240	240	240	240	240	240	240

210	210	210	0	210	210	210
45	210	210	0	70	210	210
45	210	210	0	210	0	210
210	210	210	0	0	0	0
210	210	210	210	210	210	70
45	70	70	70	45	70	70
70	45	70	70	70	45	240

ה גורם כי כל מערך הריבועים קבוץ כתום נזקן על ידי אוסף 3x3.

210	210	210
45	210	210
45	210	210

ולפניהם 210 מערך אחד: 1x1

רוצח מערך 3x3 כל מערך שפכימן נזקן

בפניהם ונהר איזה:

45, 45, 210, 210, 210, 210, 210, 210, 210

יה קולקטיבי לה 210 זה ורשות גורם כתום נזקן מכך.

210	210	0
210	210	0
210	210	210

ולפניהם 210 מערך אחד: 1x2

רוצח מערך 3x3 כל מערך שפכימן נזקן

בפניהם ונהר איזה:

0, 0, 210, 210, 210, 210, 210, 210, 210

יה קולקטיבי לה 210 זה ורשות גורם כתום נזקן מכך.

210	0	210
210	0	70
210	210	0

: 1x3 פירנש 210 פירנש

|בנוי מ-3 משבצות 3x3 מלבני

: 3 משבצות ו-3 משבצות

$$\frac{0, 0}{}, 0, 70, \underline{210}, 210, 210, 210, 210$$

. 210 -> פירנש 0 ירע

0	210	210
0	70	210
210	0	210

: 1x4 פירנש 210 פירנש

|בנוי מ-4 משבצות 3x3 מלבני

: 3 משבצות ו-3 משבצות

$$\frac{0, 0}{}, 0, 70, \underline{210}, 210, 210, 210, 210$$

. 210 -> פירנש 70 ירע

210	210	210
70	210	210
0	210	210

: 1x5 פירנש 210 פירנש

|בנוי מ-5 משבצות 3x3 מלבני

: 3 משבצות ו-3 משבצות

$$\frac{0, 70}{}, 210, 210, \underline{210}, 210, 210, 210, 210$$

. 210 ירע משבצות 3x3 מלבני

45	210	210
45	210	210
210	210	210

: 2x1 פירנש 210 פירנש

|בנוי מ-2 משבצות 3x3 מלבני

: 3 משבצות ו-3 משבצות

$$\frac{45, 45}{}, 210, 210, \underline{210}, 210, 210, 210, 210$$

. 210 ירע משבצות 3x3 מלבני

210	210	0
210	210	210
210	210	0

: 2x2 μ g/ml 210 μ g/ml

↳ Final 12'000 160'000 3x3 110'000

: յ Յ Ր Ո Ր Ա Կ Կ Յ Ն Ո Ւ Ծ Ի Շ Ճ Ճ

$$\overbrace{0, 0, 210, 210, 210, 210, 210, 210, 210}^{\rightarrow}$$

בדיוק נזכר בזאת כי מטרת החקיקה היא לסייע לאם בוגריה.

..... תרגום היגייניסטי

לבסוף, לאחר הפעלת מסנן חיצון בגודל 3X3 על כל הפיקסלים באזורי הכתום, קיבל:

Part B - Median Filter

Original Cropped Image

210	210	210	0	210	210	210
45	210	210	0	70	210	210
45	210	210	210	0	210	210
210	210	210	0	0	0	0
210	210	210	210	210	210	70
45	70	70	70	45	70	70
70	45	70	70	70	45	240

After Median Filter Image

A 5x5 grid diagram with colored cells and numerical labels. The grid is defined by a thick orange border. The cells are colored as follows:

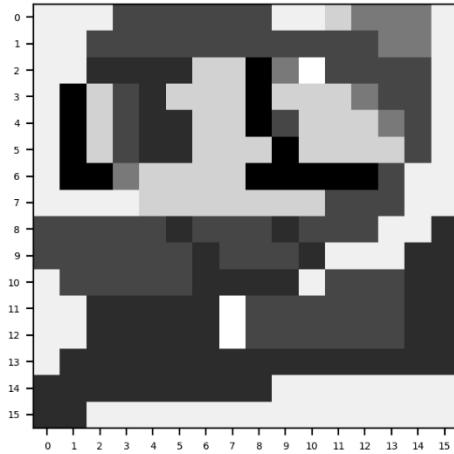
- Row 1: All cells are dark gray.
- Row 2: Cells 1, 2, 4, and 5 are light gray; Cell 3 is dark gray with a red "70".
- Row 3: Cells 1, 2, 4, and 5 are light gray; Cell 3 is dark gray with a red "0".
- Row 4: Cells 1, 2, 4, and 5 are light gray; Cell 3 is white with a red "210".
- Row 5: Cells 1, 2, and 3 are light gray; Cells 4 and 5 are dark gray with red "70"s.

Red numerical labels are placed in the following cells:

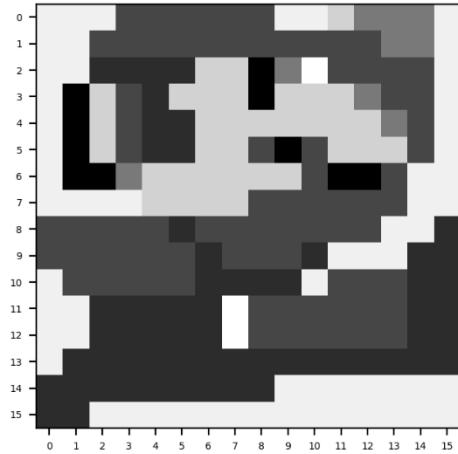
- Cell (1, 1) has a red "210".
- Cell (1, 2) has a red "210".
- Cell (1, 4) has a red "210".
- Cell (1, 5) has a red "210".
- Cell (2, 3) has a red "70".
- Cell (3, 3) has a red "0".
- Cell (4, 3) has a red "210".
- Cell (5, 3) has a red "210".
- Cell (5, 4) has a red "70".
- Cell (5, 5) has a red "70".
- Cell (5, 1) has a red "70".
- Cell (5, 2) has a red "70".
- Cell (5, 4) has a red "70".
- Cell (5, 5) has a red "70".

Part B - Median Filter

Original Image

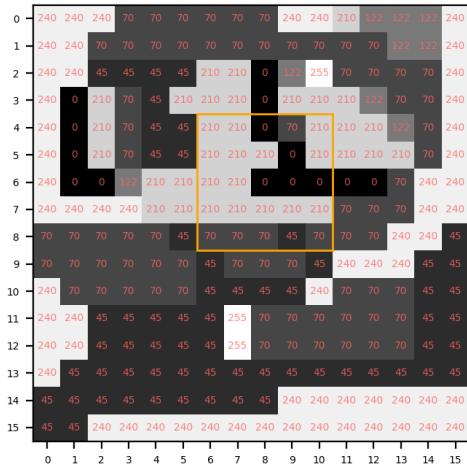


After Median Filter Image

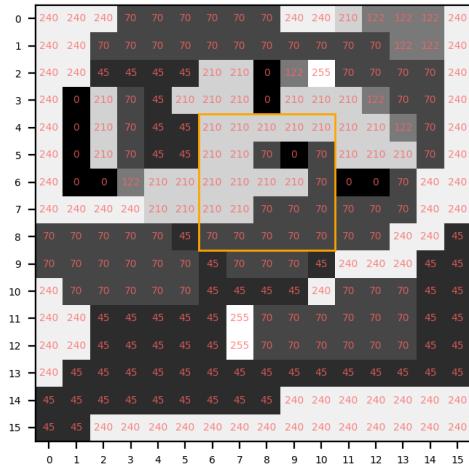


Part B - Median Filter

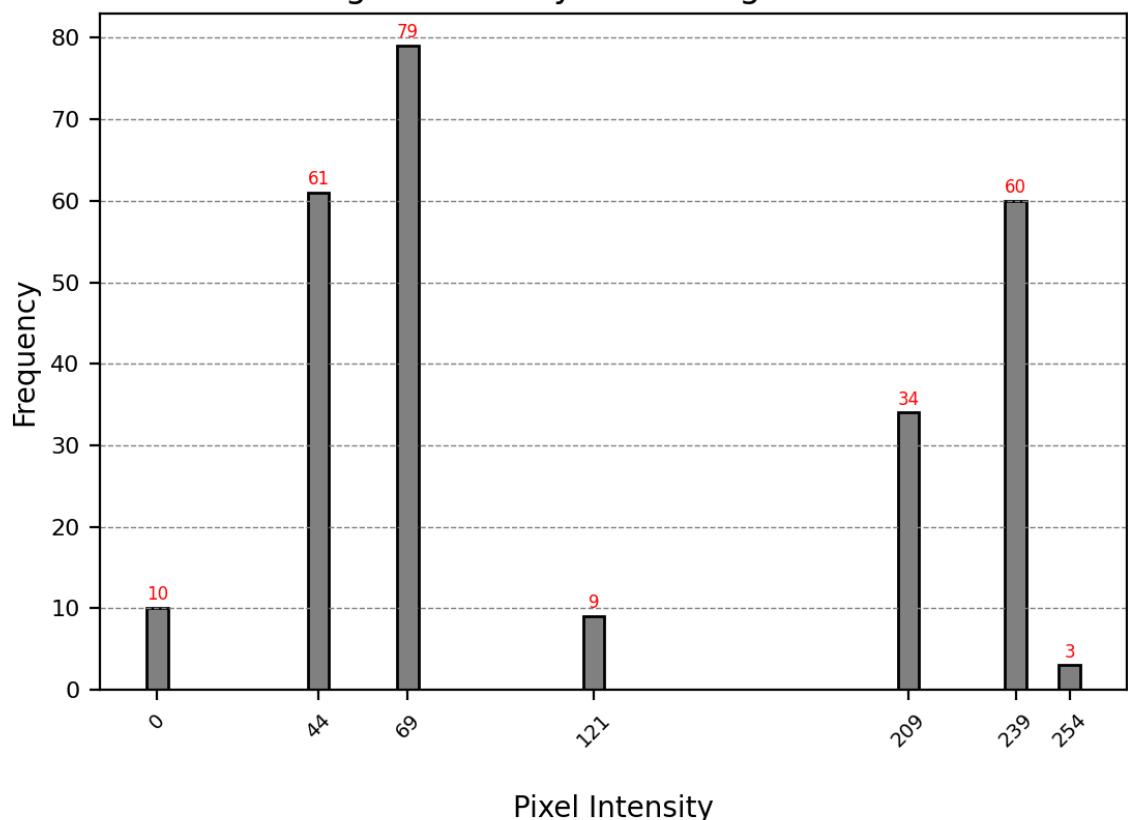
Original Image



After Median Filter Image



Part B - Histogram of Grayscale Image after Median Filter



Pixel Intensity (0-255)	0	44	69	121	209	239	254
Frequency	10	61	79	9	34	60	3

c. הפעלת מסנן החלקה בסיסי בגודל 3X3 על הפיקסלים שבמסגרת הכתומה:

נפעיל מסנן החלקה בסיסי בגודל 3X3 באופן הבא:
(הנחה שלקחנו – כאשר הפעילנו את המסנן על האזור החתוון, נעזרנו בפיקסלים של שאר התמונה ולא רידנו בערכים אחרים)
(הנחה נוספת – עיגול הערכים היה באופן הבא: 0.5 ומעלה – מעגלים למעלה, כל השאר – מעגלים למטה)

c. גורן סעיף א' מתרגם כך כתוב בספר נון תמלוגם אמצע 3X3.

210	210	210
45	210	210
45	210	210

גורן סעיף א' מתרגם כמפורט מטה:

רכס א' מילג'ת הנקודות

וילג'ת נ-ב:

$$\text{Sum} = 210 + 210 + 210 + 45 + 210 + 210 + 45 + 210 + 210 = 1560 \rightarrow \frac{1560}{9} = 173\frac{1}{3}$$

אלאיך שאלג'ת נ-ב רישום גנוי (173), אלאיך שאלג'ת נ-ב רישום גנוי (174).

מכן נאזרחה אלה וביניהם 210 והוקג'ות 173 גורר גוונת נון קוליה.

210	210	0
210	210	0
210	210	210

גורן סעיף א' מתרגם כמפורט מטה:

רכס א' מילג'ת הנקודות

וילג'ת נ-ב:

$$\text{Sum} = 210 + 210 + 0 + 210 + 210 + 0 + 210 + 210 + 210 = 1470 \rightarrow \frac{1470}{9} = 163\frac{1}{3}$$

אלאיך שאלג'ת נ-ב רישום גנוי (163), אלאיך שאלג'ת נ-ב רישום גנוי (164).

מכן נאזרחה אלה וביניהם 210 והוקג'ות 163 גורר גוונת נון קוליה.

210	0	210
210	0	70
210	210	0

$$: 1 \times 3 \text{ შევადა } 210 \quad 675 \text{ უდის}$$

ר' פירש ר' יוסר סח' מ' אס' ר' יוסר

$$: \frac{1}{g} - \lambda \uparrow m_1$$

$$\text{Sum} = 210 + 0 + 210 + 210 + 0 + 70 + 20 + 210 + 0 = 1120 \rightarrow \frac{1120}{9} = 124\frac{4}{9}$$

(124) $\sqrt{m} \sqrt{n} = \sqrt{mn}$, (125) $\sqrt{m} \sqrt{n} \neq \sqrt{m+n}$

בגד נארהה אלה הטעינהו ב-210 הילק'ות מהן 124 גיאור גיאור נוד ורעה.

0	210	210
0	70	210
210	0	210

: 1x4 πιπέρια 210 σοτσί 708

የኢትዮጵያ ከፋይ ስነዕና ማርያም

$$\therefore \frac{1}{g} - 2 \uparrow m$$

$$\text{Sum} = 0 + 210 + 210 + 0 + 70 + 210 + 210 + 0 + 210 = 1170 \rightarrow \frac{1170}{9} = 124 \frac{4}{9}$$

(124) $\sqrt{6N} \sqrt{\ln N} \frac{1}{\epsilon^2 - N}$, (125) $\sqrt{6N} \sqrt{\ln N} \frac{1}{\epsilon^2 - N}$

בגד פנורמה מה שבטו 210 הילק גויה 124 גויה גויה ווונט גויה.

210	210	210
70	210	210
0	210	210

: 1 x 5 μ m \approx 210 μ m

לכוד נסחף ורשותה מינהלית

$$: \frac{1}{g} - 2 \uparrow^{(n)} 1$$

$$\text{Sum} = 210 + 210 + 210 + 70 + 210 + 210 + 0 + 210 + 210 = 1540 \rightarrow \frac{1540}{9} = 171\frac{1}{9}$$

8. אוניברסיטאות ומוסדות מחקר נאכרים במקומות שונים, (172) מ- \sqrt{N} מילויים לא- \sqrt{N} מילויים.

בגד כנראה לא היה 210 הילק גיאת 171 גיאר נול ווילג'ה.

45	210	210
45	210	210
210	210	210

פָּרֶסֶת כְּלַבֵּב : 2×1 פָּרֶסֶת כְּלַבֵּב
פָּרֶסֶת כְּלַבֵּב : $\frac{1}{2} - \frac{1}{2}$ פָּרֶסֶת כְּלַבֵּב

$$\text{Sum} = 45 + 210 + 210 + 45 + 210 + 210 + 210 + 210 + 210 = 1560 \rightarrow \frac{1560}{9} = 173\frac{1}{3}$$

.(173) פָּרֶסֶת כְּלַבֵּב $\frac{1}{2} - \frac{1}{2}$ פָּרֶסֶת כְּלַבֵּב , (174) פָּרֶסֶת כְּלַבֵּב $\frac{1}{2} - \frac{1}{2}$ פָּרֶסֶת כְּלַבֵּב .

גִּזְבָּן כְּאֹרֶת הַלְּבָב 173 גִּזְבָּן הַלְּבָב וְעַדְעָה .

210	210	0
210	210	210
210	210	0

פָּרֶסֶת כְּלַבֵּב : 2×2 פָּרֶסֶת כְּלַבֵּב
פָּרֶסֶת כְּלַבֵּב : $\frac{1}{4} - \frac{1}{4}$ פָּרֶסֶת כְּלַבֵּב

: $\frac{1}{4} - \frac{1}{4}$ פָּרֶסֶת כְּלַבֵּב

$$\text{Sum} = 210 + 210 + 0 + 210 + 210 + 210 + 210 + 0 = 1470 \rightarrow \frac{1470}{9} = 163\frac{1}{3}$$

.(163) פָּרֶסֶת כְּלַבֵּב $\frac{1}{2} - \frac{1}{2}$ פָּרֶסֶת כְּלַבֵּב , (164) פָּרֶסֶת כְּלַבֵּב $\frac{1}{2} - \frac{1}{2}$ רַבְבָּן .

גִּזְבָּן כְּאֹרֶת הַלְּבָב 163 גִּזְבָּן הַלְּבָב וְעַדְעָה .

וְעַדְעָה .

לבסוף, לאחר הפעלת מסנן היחסיקה הבסיסי בגודל 3×3 על כל הפיקסלים באזור הכתום, נקבל:

Part C - Mean Filter

Original Cropped Image

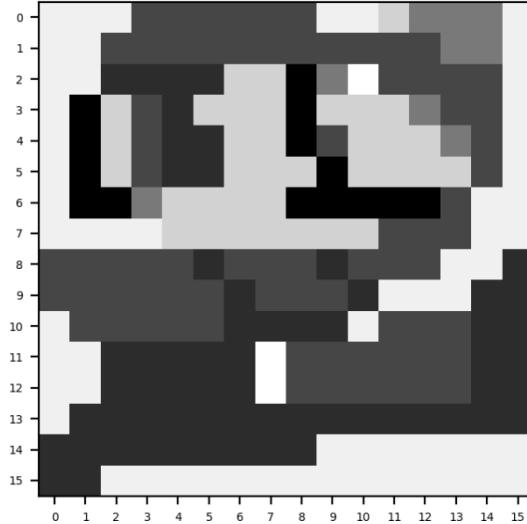
210	210	210	0	210	210	210
45	210	210	0	70	210	210
45	210	210	210	0	210	210
210	210	210	0	0	0	0
210	210	210	210	210	210	70
45	70	70	70	45	70	70
70	45	70	70	70	45	240

After Mean Filter Image

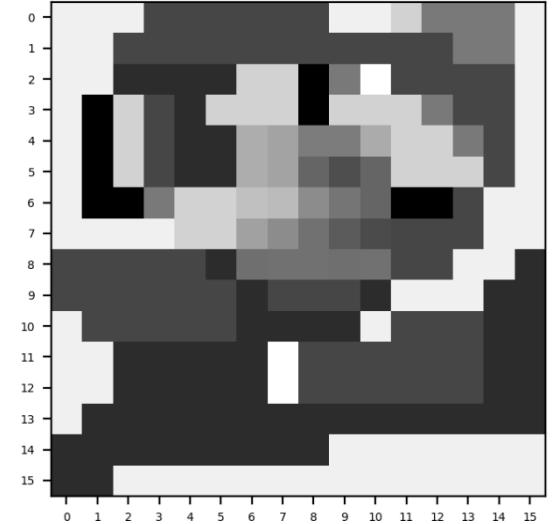
173	163	124	124	171
173	163	101	78	101
192	187	140	117	101
161	140	114	91	75
111	114	114	111	114

Part C - Mean Filter

Original Image



After Mean Filter Image

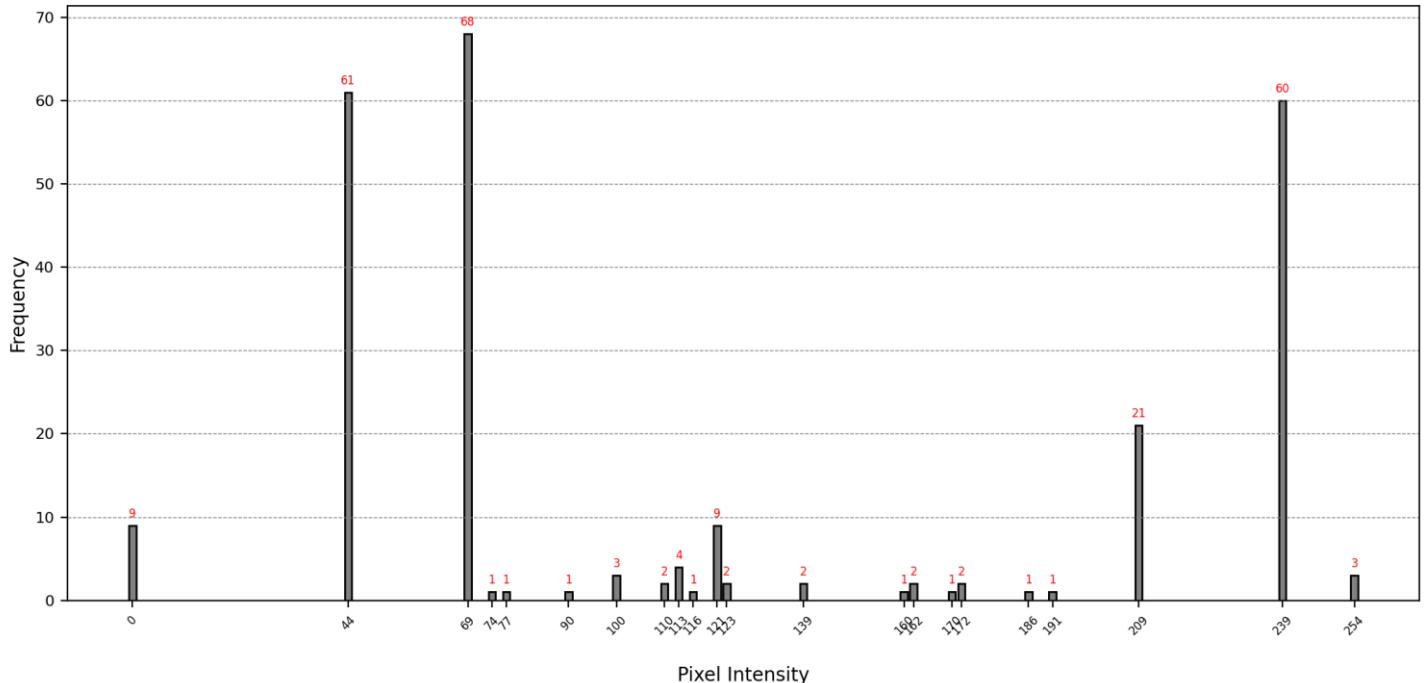


Part C - Mean Filter

Original Image

After Mean Filter Image

Part C - Histogram of Grayscale Image after Mean Filter applied



Pixel Intensity (0-255)	0	44	69	74	77	90	100	110	113	116	121	123
Frequency	9	61	68	1	1	1	3	2	4	1	9	2

Pixel Intensity (0-255)	139	160	162	170	172	186	191	209	239	254
Frequency	2	1	2	1	2	1	1	21	60	3

Pixel Intensity (0-255)	139	160	162	170	172	186	191	209	239	254
Frequency	2	1	2	1	2	1	1	21	60	3

פ. בהסתמך על התוצאות שקיבלנו:

א. חזקנות וחולשות של כל שיטה:

- מסנן חיצון (Median Filter)

חזקנות:

mphicit רעש בעילות: מסנן החיצון ייעיל במיוחד בהסרת רעש מסוג "מלח ופלפל", ככלمر רעש שבו יש נקודות שחומות ולבנות באופן אקראי.

שומר על קצוות (Edges): ביגוד למסנן החלקה בסיסי, מסנן החיצון שומר טוב יותר על הקצוות בתמונה, מכיוון שהוא מחליף את הפיקסל בערך החיצוני ולא במוצעו. זה שימושי בתמונות שבהן חשוב לשמר על פרטיהם המקוריים.

חולשות:

דורש חישוב רב יותר: סידור הערכים בכל אזור 3×3 כדי למצאו את החיצון הוא תהליך מורכב יותר מבוחינת חישוב, במיוחד עבור מסננים גדולים יותר.

פחות יעיל ברעש מסווג גאוסיאני: מסנן החיצון פחות מתאים להפחיתת רעש גאוסיאני (התפלגות נורמלית), שבו ערכי הפיקסלם קרובים למוצע אך לא קיצוניים. במקרה זה, התוצאה עשויה להיראות פחותה חלקה.

- מסנן החלקה בסיסי (Mean Filter)

חזקנות:

mphicit רעש גאוסיאני בעילות: מסנן החלקה הבסיסי ייעיל בהפחיתת רעש גאוסיאני על ידי מיצוע ערכי הפיקסלם, מה שנונע מראה חלק יותר.

דורש פחות חישוב: חישוב ממוצע הוא בדרך כלל מהיר יותר מחישוב חיצון, מה שהופך את מסנן החלקה הבסיסי למהיר ויעיל יותר מבוחנת חישוב.

חולשות:

מטשטש קצוות: מיצוע נוטה לטשטש קצוות חדים, מה שמחלייש את המעברים בין פיקסלם עם ניגדיות גבוהה. לכן, הוא פחות מתאים בתמונות שבהן חשוב לשמור על פרטיהם המקוריים.

פחות יעיל ברעש "מלח ופלפל": מכיוון שמסנן החלקה בסיסי מחשב ממוצע לכל הפיקסלם בסביבה, הוא עלול להפיץ את רעש "מלח ופלפל" למקום להסיר אותו.

॥. האם יש שיטה אחת שתמיד תהיה טובה יותר?

לא, אין שיטה אחת שתמיד עדיפה. הבחירה בין השיטות תלויה בסוג הרעש ובמטרה של עיבוד התמונה. הנה כמה דוגמאות המדגימות מתי כל שיטה עדיפה:

דוגמה 1: תמונה עם רעש "מלח ופלפל"

מסנן חיצון: מתאים יותר מכיוון שהוא מסיר ערכים קיצוניים מבלוי להרטיע הרבה על הפיקסלם שמסביב.

מסנן החלקה בסיסי: יטשטש את הרעש עם הפיקסלם הסמוכים, מה שיוצר טשטוש לא טבעי במקומות להעלים את הרעש.

דוגמה 2: תמונה עם רעש גאוסיאני (התפלגות נורמלית)

מסנן החלקה בסיסי: מתאים להפחיתת רעש גאוסיאני, מכיוון שהוא מחליך את התמונה על ידי מיצוע השינויים הקטנים.

מסנן חיצון: יכול להפחית את רעש הגאוסיאני, אך עשוי לא להניב תוצאה חלקה כמו מסנן החלקה בסיסי.

דוגמה 3: תמונה עם פרטיים חדים (כמו טקסט או צירום מקווקויים)
מסנן חציון : שומר על הקצוות ומסיר רוש תוך שמירה על פרטיים חדים.
מסנן החלקה בסיסי : עלול לטשטש את הקווים והפרטים הקטנים, מה שיגרום לאיבוד פרטיים בתמונה.

דוגמה 4: תמונה אחידה עם רוש נמוך (ללא פרטיים קרייטיים)
שני המנסננים : עשויים להניב תוצאות טובות, אך מסנן החלקה בסיסי יהיה מהיר ויעיל יותר מכיוון שהפרטים אינם קרייטיים ואין צורך לשמור על קצוות חדים.

לסיכום, אין מסנן אחד שתמיד יהיה טוב יותר; כל מסנן מתאים למקרים שונים לפי סוג הרוש והצורך
בשמירת פרטיים מסוימים בתמונה.

e. נסתכל על שתי הפעולות – מסנן חציון SV. מסנן החלקה בסיסי:

בין שתי הפעולות **מסנן החלקה בסיסי** (basic smoothing filter) נחשב בדרך כלל קל יותר לביטול או לפחות להערכתה מחדש של התוצאה המקורייה, בהשוואה ל**מסנן חציון**. הנה ההסבר לכך ואיך ניתן היה לגשת לשחזור התמונה.

מסנן החלקה בסיסי (Basic Smoothing Filter)

מסנן החלקה בסיסי מבוסס על מיצוע ערכים בשכונות פיקסלים, כך שכל פיקסל בתמונה החדש הוא ממוצעו של הערך המקורי ושל הפיקסלים השכנים. תהליך זה יוצר טשטוש, אך שומר מידע מהפיקסלים השכנים באופן שנייתן לחזותאותו.

כדי לשחרר את התמונה המקורייה, ניתן להשתמש בתהליך שנקרא דה-קונבולוציה (deconvolution), אשר מנסה "להפוך" את פעולה המיצוע ולשחרר את הערכים המקוריים על סמך התמונה המטושטשת וגרעין המיצוע המקורי. תהליך זה דורש:

- ידע לגבי גודל וסוג המסנן (למשל, מסנן מיצוע בגודל 3x3).
- אלגוריתם דה-קונבולוציה, כמו מסנן וינר (Wiener Filter), אשר מטרתו לשחרר הערכה של התמונה המקוריית על ידי מזעור השפעת הטשטוש.

אתגרים: דה-קונבולוציה עשויה להיות מושפעת מרעש, ולכן השחזור לא יהיה מושלם. עם זאת, האופי הצפוי של מסנן החלקה בסיסי הופך אותו לפחות יתר לשחזור בהשוואה למסנן חציון.

מסנן חציון (Median Filter)

מסנן חציון מחליף כל פיקסל בערך החצינוי של שכנותו, מה שיוכל לשנות את הערכים המקוריים משמעותית, במיוחד במקרים עם ניגודיות גבוההה. פעולה החציון היא לא לינארית ולא הפיכה, מכיוון שהיא מחליפה את הערכים המקוריים במקומות מסוימים, וכך קשה מאוד לשחרר את התמונה המקורייה.

שחזור יהיה בוגדר הערכה בלבד. גישה אפשרית (אך לא מושלמת) יכולה לכלול:

- שימוש בשיטות אופטימיזציה כדי לנחש את הערכים המקוריים על סמך התמונה לאחר סינון החציון, תוך ניסיון למזרע את ההבדלים.
- לחופשי, אם התמונה מכילה דפוסים ידועים, ניתן לנסוט לשימוש במודלים של מידת מכונה שינסו לנבע את המידע החסר על בסיס סט נתוניים דומים.

אתגרים: מכיוון שמסנן חציון מחליף את הערכים המקוריים בערכים חצינויים, אין דרך ישירה לשחרר את המידע המקורי, מה שהופך את השחזור למורכב ועתידי אף לבליי אפשרי.

לסיכום, **מסנן החלקה בסיסי** קל יותר לביטול באמצעות טכניקות דה-קונבולוציה, מכיוון שהשפעותיו צפויות ונינעות לשחזור באופן ייחודי. לעומת זאת, **מסנן חציון** נחשב בלתי הפיך בדרך כלל, כיון שהוא מאבד יותר מדי מידע על הערכים המקוריים של הפיקסלים.

שאלה 3:

אלגוריתם של שיטת Patch-Based Confidence Level תוך שימוש ב-Confidence Level בלבד:

- נניח חלון window_size עבור חישוב ה-Confidence Level של 3X3.

1. נאותחל Confidence Matrix כך שכל פיקסל חסר הוא 0 ושאר הפיקסלים 1 –
2. נמצאת כל הפיקסלים החסרים בתמונה – missing_pixels
3. עבור כל פיקסל חסר:
 - a. נאותחל מערך של פיקסל המסגרת, אם לא קיימים סיים לולאה – missing_pixels
 - b. נחשב Confidence Level לכל אחד מפיקסלי המסגרת
 - c. נמצאת הפיקסל עם ה-Confidence Level הגבוה ביותר (אם קיימים כמה בעלי אותו ערך מינימלי, ניקח את הראשון מביניהם) – selected_pixel
 - d. נשלוף את ה-Patch (בגודל 3X3 (window_size)) סביבו אותו פיקסל נבחר – selected_patch
 - e. נמצאת החלון בגודל 3X3 שעריך RMSE בין ה-Patch והוא הנמור ביותר בתמונה – (אם קיימים כמה בעלי אותו ערך מינימלי, ניקח את הראשון מביניהם) – best_patch_location
 - f. נשלוף את הפיקסלים של החלון הנבחר – best_patch
 - g. נמלא את הפיקסלים החסרים ב-Patch בעזרת הפיקסלים של החלון הנבחר (לפי הסדר).
 - h. נעדכן את ה-Confidence Matrix ב-1ים בכל מקום שמיילנו בפיקסלים חדשים.
 - i. נעדכן את התמונה המקורי בעזרת ה-Patch.
 - j. נעדכן את מערך הפיקסלים החסרים לאחר המילוי.

האלגוריתם בקוד:

```
def calculate_patch_confidence(confidence, x, y, window_size):  
    """  
    Calculates the average confidence for a patch (window) centered at a given pixel in the confidence map.  
  
    Args:  
        - confidence (array-like): A 2D array representing the confidence map of the image.  
        - x (int): The x-coordinate (column) of the center of the patch.  
        - y (int): The y-coordinate (row) of the center of the patch.  
        - window_size (int): The size of the square patch (window) centered at (x, y).  
    """  
  
    half_w = window_size // 2  
    patch = confidence[max(0, x - half_w):x + half_w + 1, max(0, y - half_w):y + half_w + 1]  
    return np.mean(patch)
```

```
def calculate_rmse(patch1, patch2):  
    """  
    Calculates the Root Mean Square Error (RMSE) between two image patches, considering only valid pixels.  
  
    Args:  
        - patch1 (array-like): The first image patch (2D array).  
        - patch2 (array-like): The second image patch (2D array) to compare against the first patch.  
    """  
  
    valid_mask = (patch1 >= 0) # Only compare known pixels  
    if np.sum(valid_mask) == 0:  
        return np.inf # If there are no valid pixels, return a large error  
    return np.sqrt(np.mean((patch1[valid_mask] - patch2[valid_mask])**2))
```

```

def find_best_match(image, selected_patch, window_size):
    """
    Finds the location of the best matching patch in the image based on the lowest RMSE (Root Mean Square Error).

    Args:
    - image (array-like): The input image (2D array) in which to search for the best matching patch.
    - selected_patch (array-like): The patch (2D array) to compare with patches in the image.
    - window_size (int): The size of the square patch (window) used for the search.
    """

    best_rmse = np.inf
    best_match_location = (-1, -1)
    half_w = window_size // 2
    image_h, image_w = image.shape

    # Search for patches in regions with known pixels
    for i in range(half_w, image_h - half_w):
        for j in range(half_w, image_w - half_w):
            patch = image[i - half_w:i + half_w + 1, j - half_w:j + half_w + 1]
            if np.any(patch == -1):  # Skip patches with missing pixels
                continue
            rmse = calculate_rmse(selected_patch, patch)
            if rmse < best_rmse:
                best_rmse = rmse
                best_match_location = (i, j)

    return best_match_location

```

```

def patch_based_inpainting(original_image, window_size):
    """
    Performs patch-based inpainting on an image to fill missing pixels by finding and copying similar patches.

    Args:
    - original_image (array-like): The input image (2D array) with missing pixels marked as -1.
    - window_size (int): The size of the square patch (window) used for the inpainting process.
    """

    image = original_image.copy()

    # Initialize confidence matrix (1 for known pixels, 0 for missing pixels)
    confidence = np.where(image >= 0, 1.0, 0.0)

    half_w = window_size // 2
    missing_pixels = np.argwhere(image == -1)  # Find the missing pixels

    iteration_number = 1

    while len(missing_pixels) > 0:
        boundary_pixels = []

        # Find boundary pixels (missing pixels adjacent to known pixels)
        for (x, y) in missing_pixels:
            if np.any(confidence[max(0, x-half_w):x+half_w+1, max(0, y-half_w):y+half_w+1] > 0):
                boundary_pixels.append((x, y))

        if not boundary_pixels:
            break  # No boundary pixels to fill

        # Calculate the confidence for each of the boundary pixels
        confidence = np.where(image >= 0, 1.0, 0.0)
        previous_confidence = confidence.copy()
        for (i,j) in boundary_pixels:
            # Update confidence for newly filled pixel
            confidence[i, j] = calculate_patch_confidence(previous_confidence, i, j, window_size)

```

```

display_matrix(confidence, f"Confidence Matrix - Iteration No{iteration_number}", with_ticks=True, with_values=True, vmin=0, vmax=1)

# Find the boundary pixel with the highest confidence
max_confidence = -np.inf
selected_pixel = (-1,-1)
for (i,j) in boundary_pixels:
    if confidence[(i,j)] > max_confidence:
        selected_pixel = (i,j)
        max_confidence = confidence[(i,j)]

# Extract the patch around the selected pixel
selected_patch = image[selected_pixel[0] - half_w:selected_pixel[0] + half_w + 1,
                       selected_pixel[1] - half_w:selected_pixel[1] + half_w + 1]

# Find the best matching patch in the known region
best_match_location = find_best_match(image, selected_patch, window_size)

# Get the best matching patch
best_patch = image[best_match_location[0] - half_w:best_match_location[0] + half_w + 1,
                  best_match_location[1] - half_w:best_match_location[1] + half_w + 1]

# display_patch(selected_patch, "patch")
# display_patch(best_patch, "best patch")

display_image(image, f"Iteration No{iteration_number} - Before Update", window_size, best_match_location, selected_pixel)
display_image(image, f"Iteration No{iteration_number} - Before Update", window_size, best_match_location, selected_pixel, with_intensity=True)

# Fill in the missing pixels in the selected patch
missing_mask_in_patch = (selected_patch == -1)
selected_patch[missing_mask_in_patch] = best_patch[missing_mask_in_patch]

# Update the confidence matrix (1 in each filled cell)
confidence[selected_pixel[0] - half_w:selected_pixel[0] + half_w + 1,
           selected_pixel[1] - half_w:selected_pixel[1] + half_w + 1] = 1

# Update the image with the selected patch
image[selected_pixel[0] - half_w:selected_pixel[0] + half_w + 1,
       selected_pixel[1] - half_w:selected_pixel[1] + half_w + 1] = selected_patch

display_image(image, f"Iteration No{iteration_number} - After Update", window_size, best_match_location, selected_pixel)
display_image(image, f"Iteration No{iteration_number} - After Update", window_size, best_match_location, selected_pixel, with_intensity=True)
iteration_number+=1

# Recompute the missing pixels
missing_pixels = np.argwhere(image == -1)

return image

```

```

def main():

    # Define the 16x16 pixel intensity values from the image
    image = np.array([
        [240, 240, 240, 70, 70, 70, 70, 240, 240, 210, 122, 122, 122, 240],
        [240, 240, 70, 70, 70, 70, 70, 70, 70, 70, 122, 122, 122, 240],
        [240, 240, 45, 45, 45, 210, 210, 0, 122, 255, 70, 70, 70, 70, 240],
        [240, 0, 210, 70, 45, 210, 210, 210, 0, 210, 210, 210, 122, 70, 70, 240],
        [240, 0, 210, 70, 45, 45, 210, 210, 0, 70, 210, 210, 210, 122, 70, 240],
        [240, 0, 210, 70, 45, 45, 210, 210, 0, 210, 210, 210, 210, 122, 70, 240],
        [240, 0, 210, 70, 45, 45, 210, 210, 0, 210, 210, 210, 210, 210, 70, 240],
        [240, 0, 0, 122, 210, 210, 210, 0, 0, 0, 0, 0, 0, 70, 240, 240],
        [240, 240, 240, 210, 210, 210, 210, 210, 210, 70, 70, 70, 240, 240, 240],
        [70, 70, 70, 70, 45, 70, 70, 45, 70, 70, 70, 70, 240, 240, 45],
        [70, 70, 70, 70, 45, 70, 70, 45, 70, 70, 70, 70, 240, 240, 45, 45],
        [240, 70, 70, 70, 70, 45, 45, 45, 240, 240, 240, 240, 45, 45, 45],
        [240, 240, 45, 45, 45, 45, -1, -1, -1, 70, 70, 70, 70, 70, 45, 45],
        [240, 240, 45, 45, 45, -1, -1, -1, 70, 70, 70, 70, 70, 45, 45],
        [240, 45, 45, 45, 45, 45, -1, -1, -1, 45, 45, 45, 45, 45, 45],
        [45, 45, 45, 45, 45, 45, -1, -1, -1, 240, 240, 240, 240, 240, 240, 240],
        [45, 45, 240, 240, 240, 240, 240, 240, 240, 240, 240, 240, 240, 240, 240]
    ], dtype=np.int32)

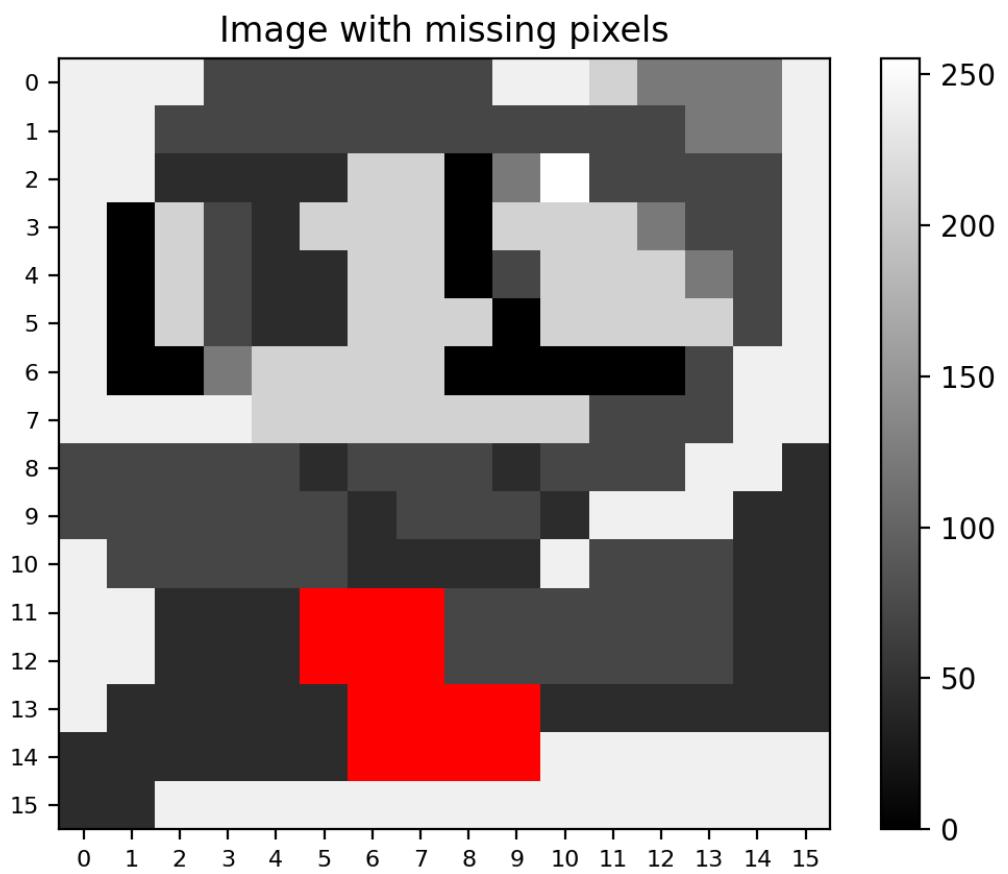
    # Display results
    display_image(image, "Image with missing pixels")

    # Perform patch-based inpainting
    window_size = 3
    filled_image = patch_based_inpainting(image, window_size)

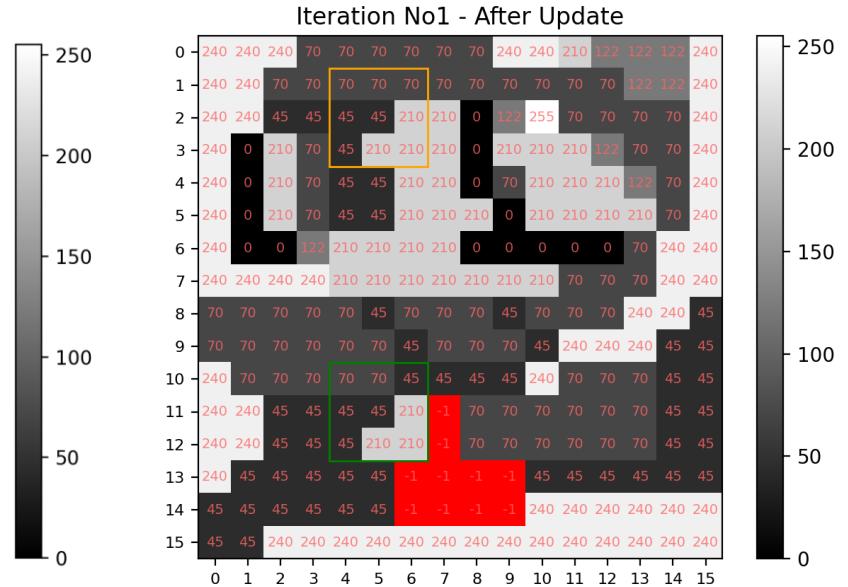
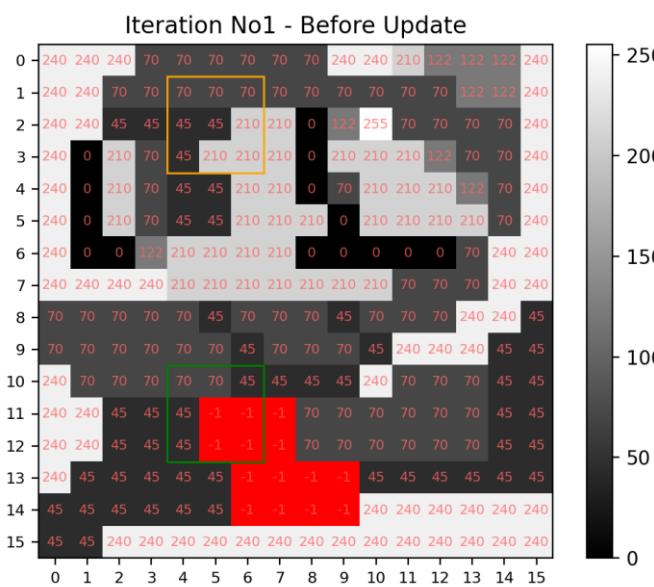
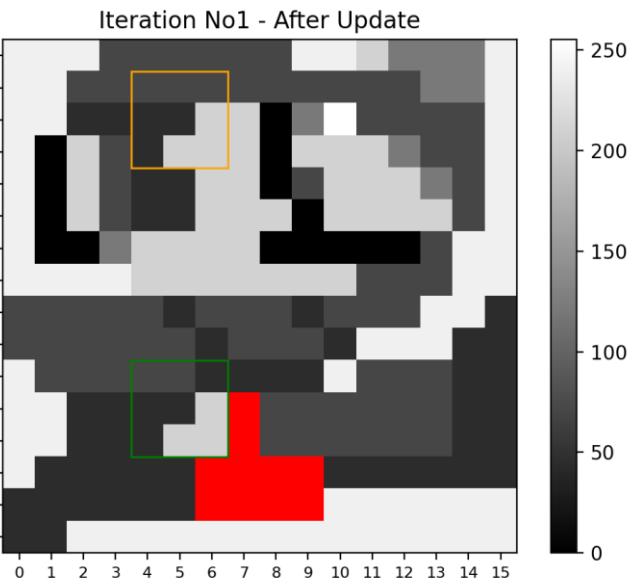
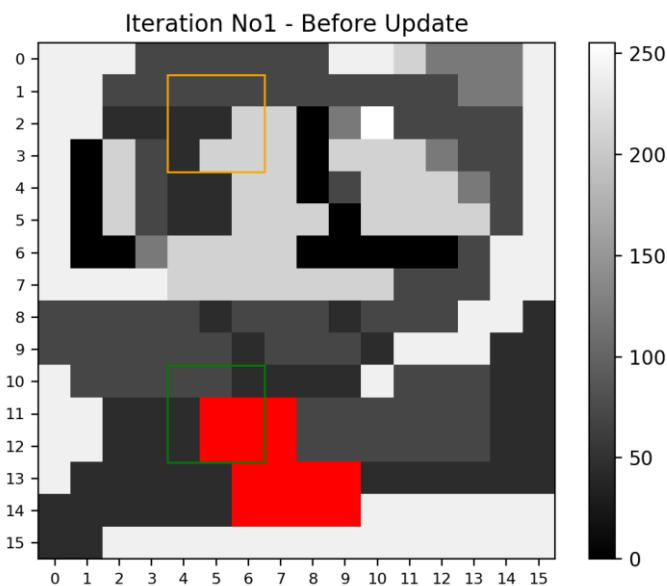
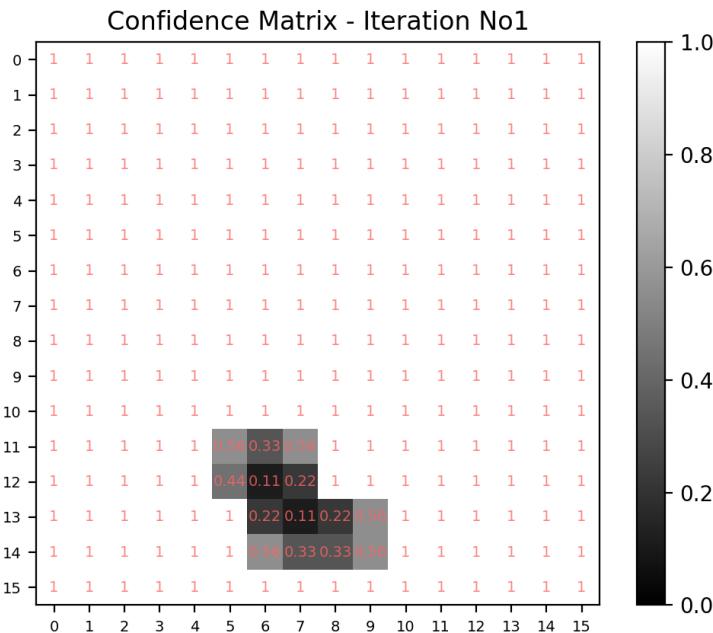
    display_images_side_by_side("Before and After Patch-Based Inpainting", image, 'Before', filled_image, 'After', with_ticks=True)
    display_image(image, "Initial Image", with_intensity=True)
    display_image(filled_image, "Final Image", with_intensity=True)

```

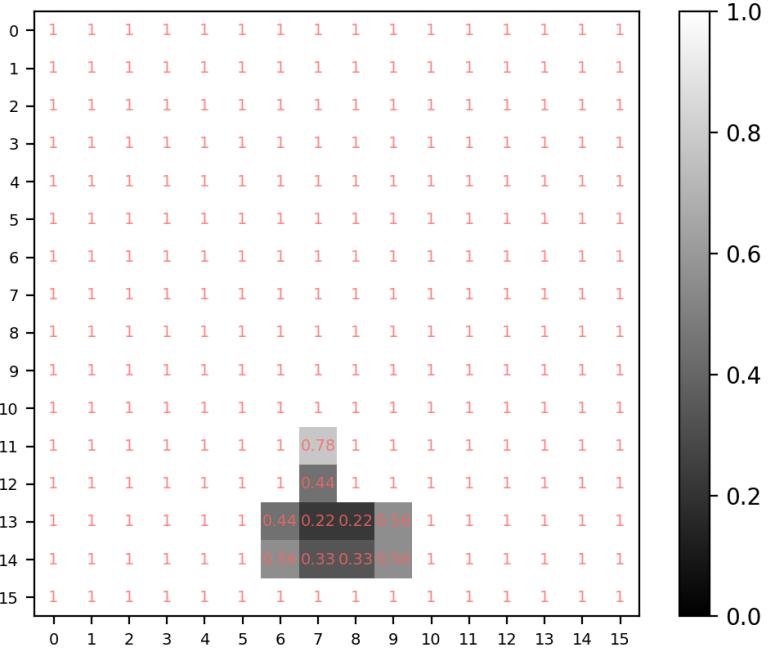
התמונה הנגטונה:



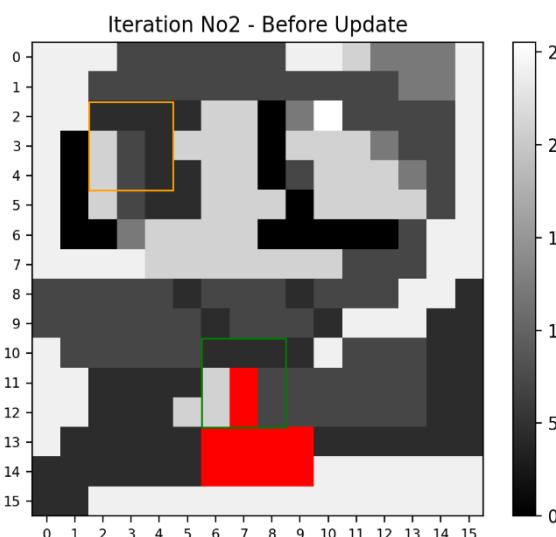
איטרציה 1 :



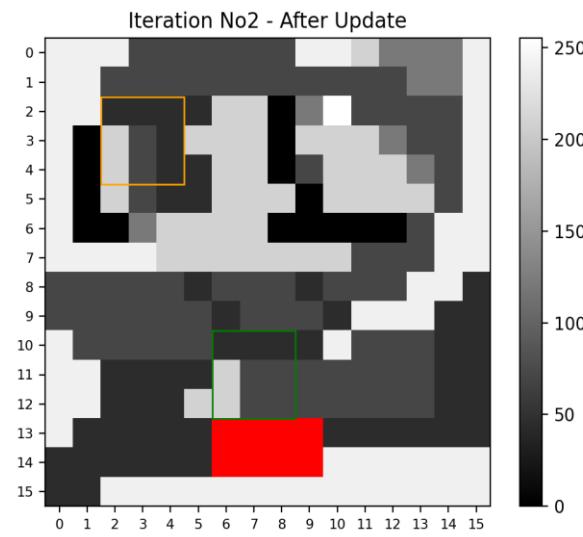
Confidence Matrix - Iteration No2



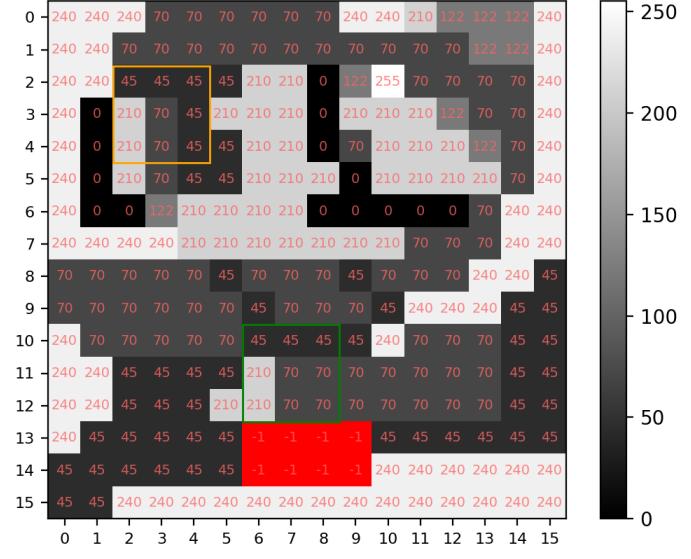
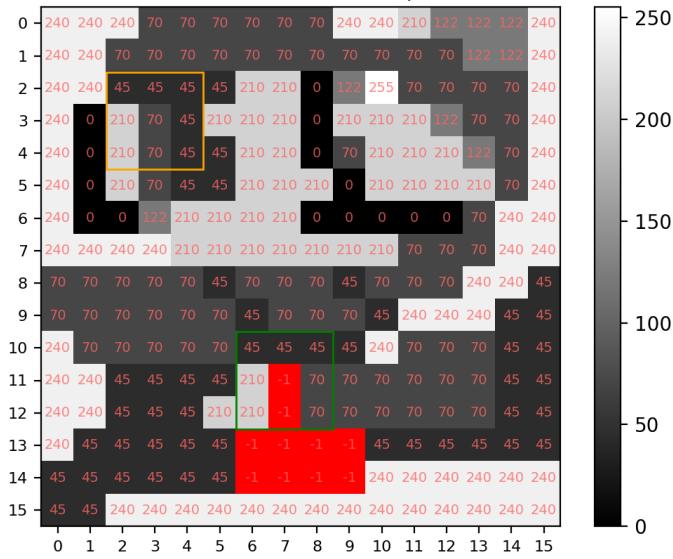
איטרציה 2:



Iteration No2 - Before Update

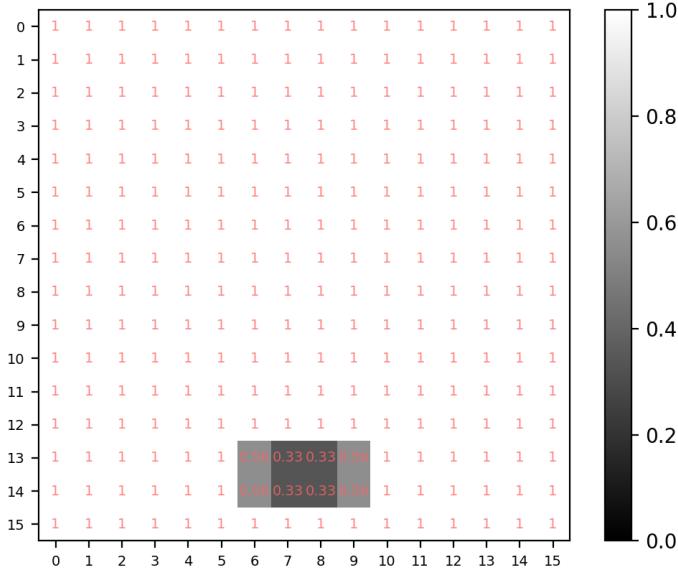


Iteration No2 - After Update

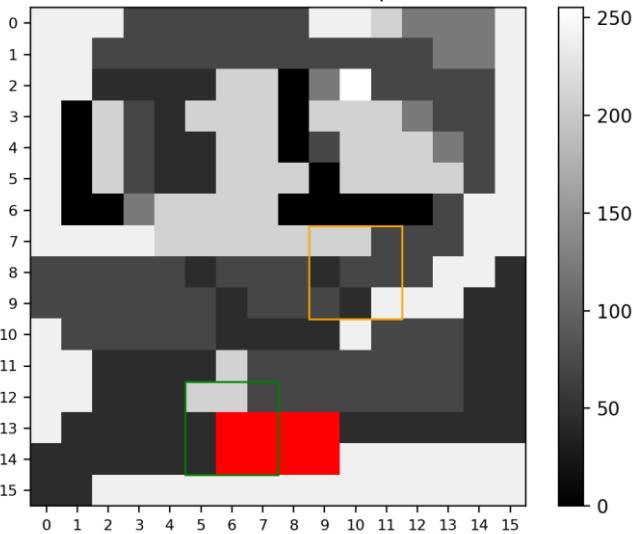


איטרציה 3:

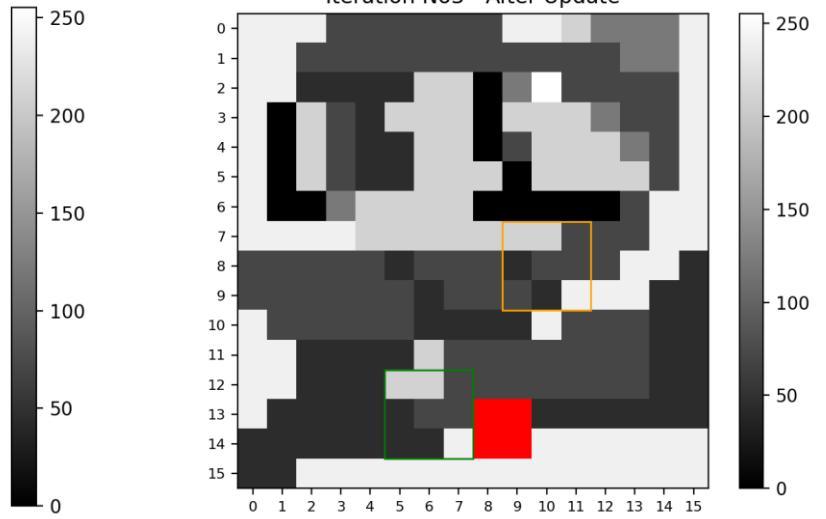
Confidence Matrix - Iteration No3



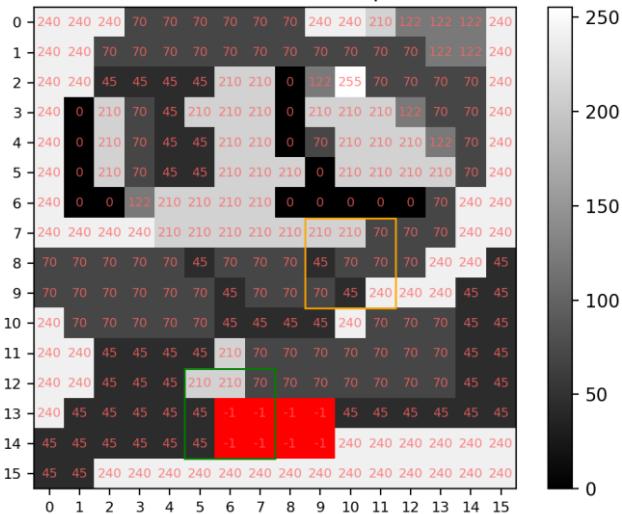
Iteration No3 - Before Update



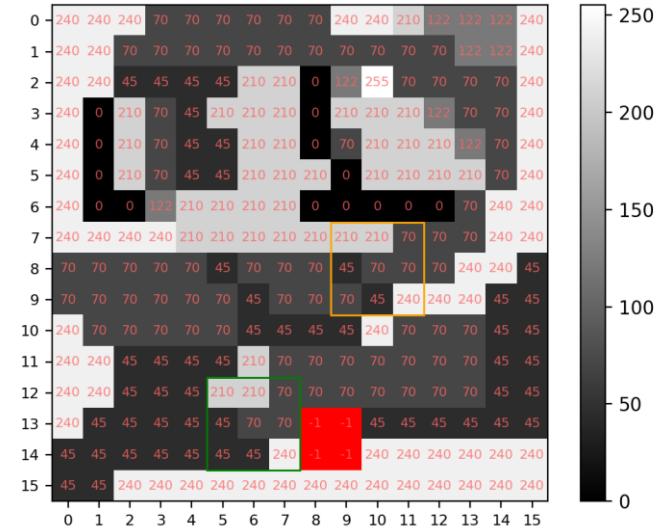
Iteration No3 - After Update



Iteration No3 - Before Update

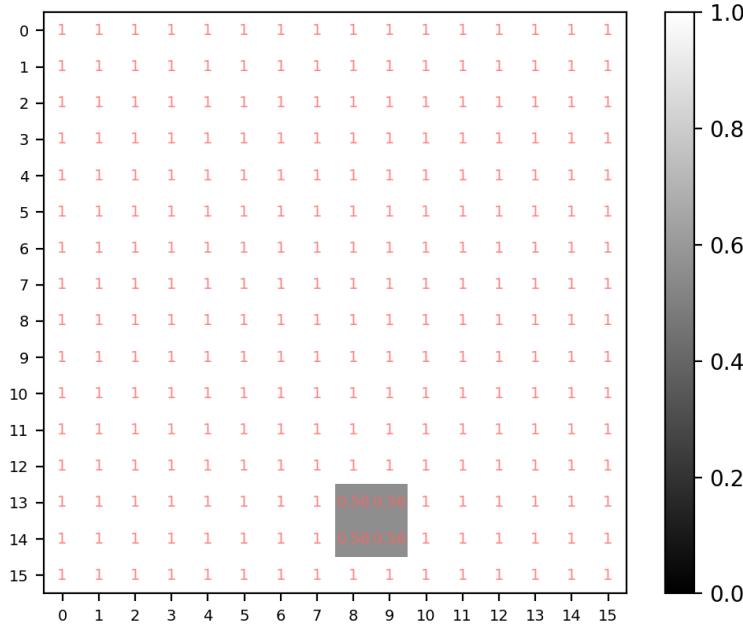


Iteration No3 - After Update

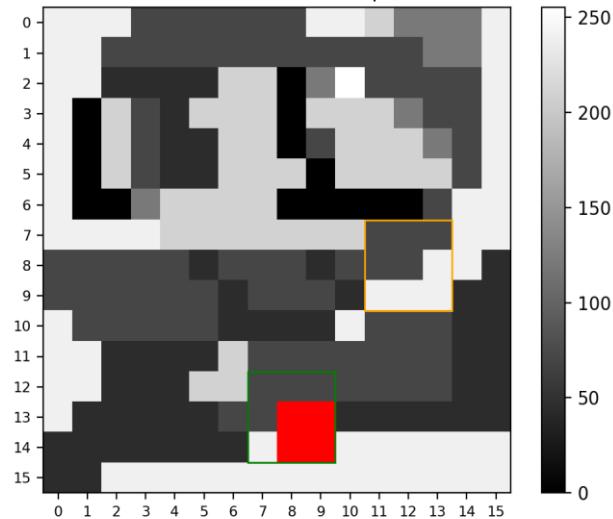


איטרציה 4:

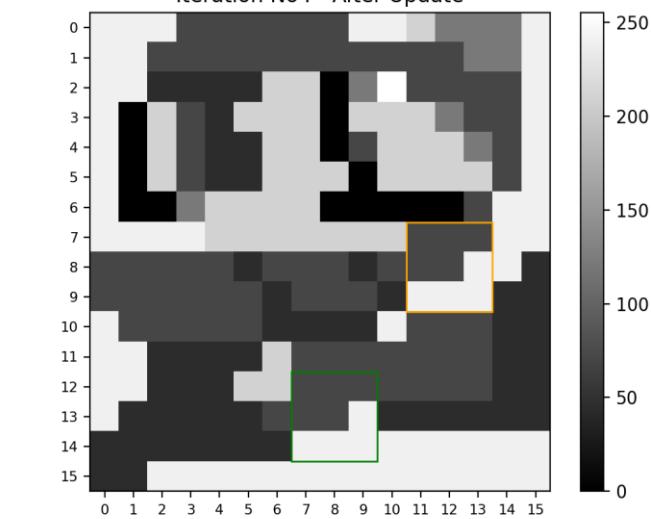
Confidence Matrix - Iteration No4



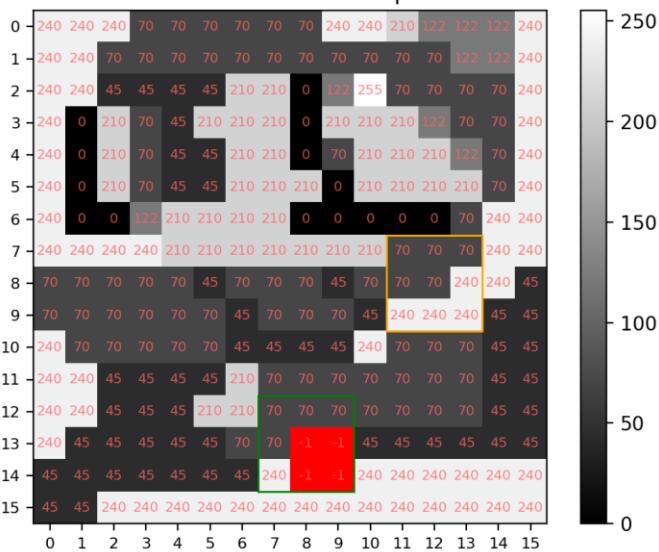
Iteration No4 - Before Update



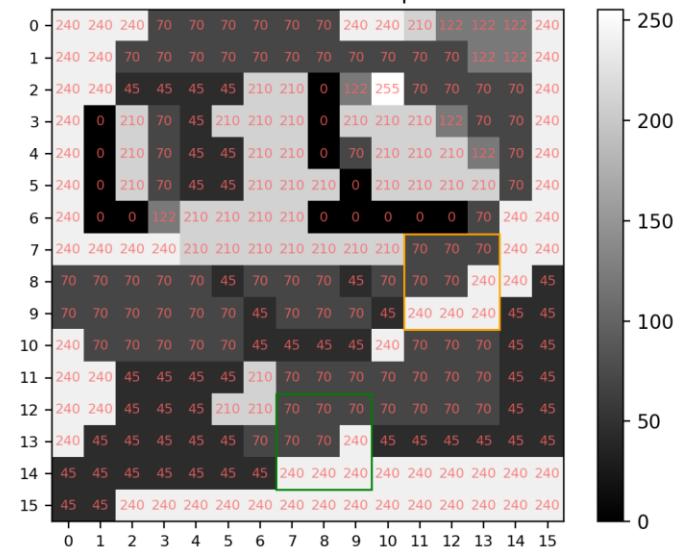
Iteration No4 - After Update



Iteration No4 - Before Update

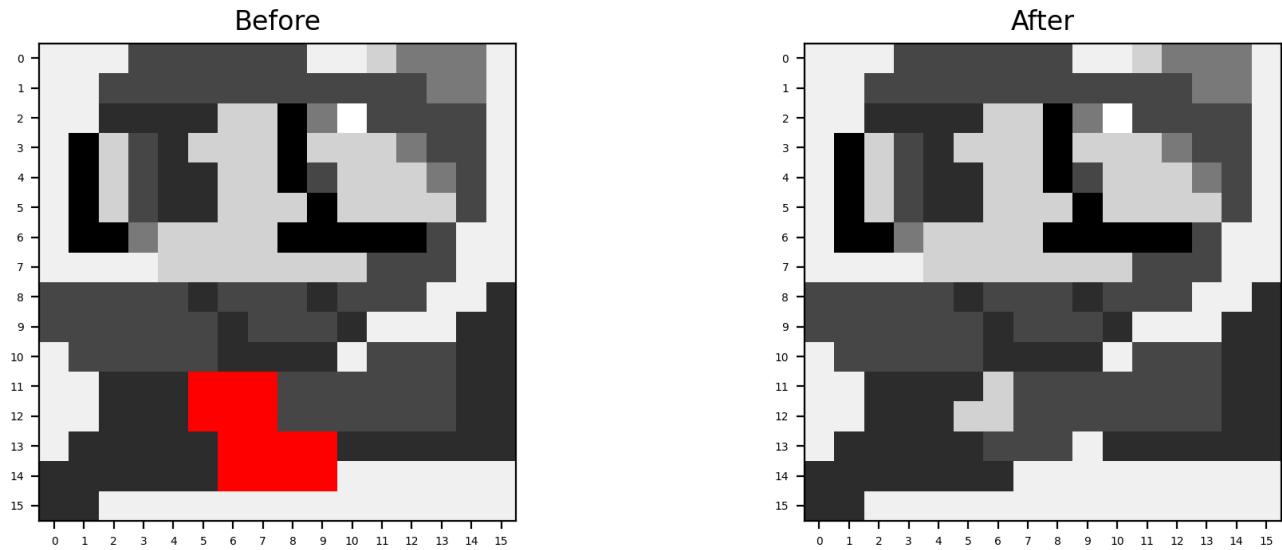


Iteration No4 - After Update

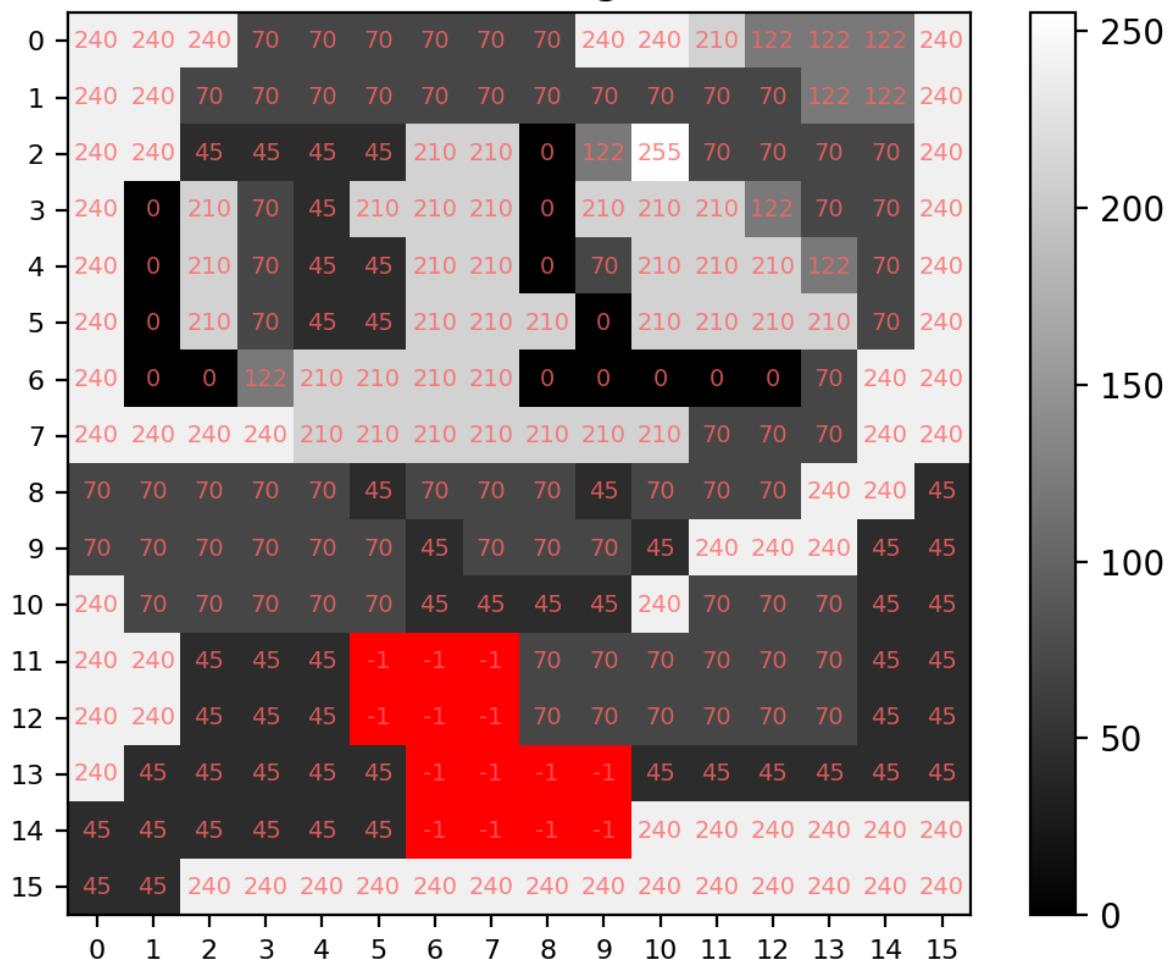


הפלט הסופי:

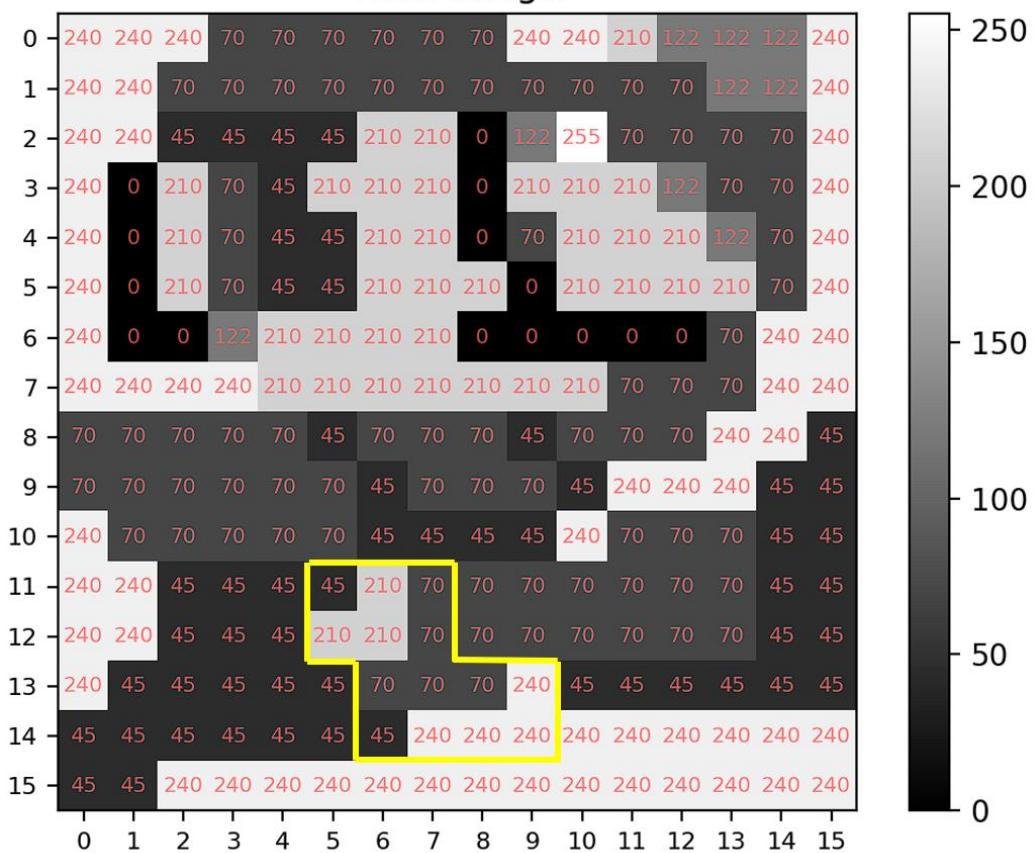
Before and After Patch-Based Inpainting



Initial Image



Final Image



לסיום, ניתן לראות שלפי הגדרת האלגוריתם, העדיפות לבחירת הפיקסל לתקן וה-patch שאיתו נשלים את אזור הפיקסל, יהיו החלקים הראשונים משמאלי (לפי לוגיקת הריצעה על מערך). לעומת זאת, אם כל המשתנים זהים, העריכים החסרים יתמלאו משמאלו למעלה לימיינ למטה.

חלק שני:

במהלך החלק האישי בפרויקט השתמשתי בספרית CV2 והפונקציות שתחתיה על מנת לבצע חלק מהפעולות שלי בקוד. لكن אציג קודם את השימוש של כל הפונקציות החיצונית עם פירוט על כל אחת מהן, וביקוד המוצג קיימת הערה לשימוש בפונקציה צזו (בפעם הראשונה שפונקציה צזו מופיעה בקוד). בסיום, קטועי הקוד מצורפים לאחר כל סעיף בעמוד נפרד.

cv2.cvtColor(src, code)

הפונקציה **cv2.cvtColor** – משמשת להמרת צבעים בתמונה מפורמט צבע אחד לפורמט צבע אחר, למשל מ-RGB ל-BGR או מ-BGR לגוני אפור (grayscale).

פרמטרים:

1. src – התמונה אותה נמייר.
2. code - קוד המיצג את סוג ההמרה שרצים לבצע.

אפשרויות:

- cv2.COLOR_BGR2GRAY – דגל המציין פעולה לבצע המרה מRGB ל-grayscale
- cv2.COLOR_BGR2RGB – דגל המציין פעולה לבצע המרה מ-BGR ל-RGB
- cv2.COLOR_RGB2BGR – דגל המציין פעולה לבצע המרה מ-RGB ל-BGR

איך היא עובדת בפועל: (המרה מRGB ל-grayscale)

בעת ההמרה מRGB לגוני אפור, OpenCV מבצע ש כולול של הערכים בערוצים הכחול, הירוק והאדום כדי לחשב את הערך של כל פיקסל בגוני אפור. הפונקציה מבצעת חישוב זהה:

$$\text{Gray} = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

המשקלים לכל אחד מהערוצים (אדום, ירוק וכחול) נקבעו בהתאם לרגישות של העין לצבעים שונים, כך שהבהירות של התמונה המתקבלת בגוני אפור תיצג נכון את תפיסת הבاهירות של העין שלנו.

cv2.Sobel(src, ddepth, dx, dy, ksize)

הפונקציה **cv2.Sobel** – משמשת לחישוב הנגזרת של התמונה בכיוון מסוים (X או Y) על מנת לזהות קצוות בתמונה.

פרמטרים:

1. src – תמונה הקלט (בדרך כלל בשחור-לבן, אך ניתן גם להשתמשocab בעובוניות).
2. ddepth - עומק התמונה המתקבלת (רוחב הערכים בפיקסלים),

למשל:

- cv2.CV_8U – ערכי פיקסלים בין 0 ל-255.
- cv2.CV_64F – ערכים מסווג מקודה צפה ברוחב 64 סיביות – דיקי מירבי וככל ערכים שליליים.
- dx – סדר הנגזרת בכיוון האופקי (ציר ה-X)
- dy – סדר הנגזרת בכיוון האנכי (ציר ה-Y)
- .5. ksize - גודל ה-kernel שבו משתמשים לחישוב הנגזרת

איך היא עובדת בפועל:

Sobel מחשב את הנגזרת של התמונה על ידי ביצוע קונבולוציה בין גרעין (kernel) בגודל 3X3 עם ערכי הפיקסלים בתמונה (כל פעם על פיקסל יחיד), כאשר הגרעין מזהה את שינוי השינוי בעוצמת הפיקסלים בכיוון האופקי (X) או האנכי (Y). חישוב מתבצע על ידי הכפלת הפיקסלים בערכים החזוביים והשליליים של הגרעין וככימה של התוצאה, זהה מייצג את הנגזרת או השינוי בעוצמה.

הקרנליים של Sobel בגודל 3x3:

1. קרNEL לחישוב הנגזרת בכיוון אופקי (X):

מיועד לזהות שינויים בכיוון האופקי. הוא מזזה שינויים בעוצמת הפיקסלים מימין לשמאל או משמאל לيمין (כמו קצנות אנכיות).

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

2. קרNEL לחישוב הנגזרת בכיוון אנכי (Y):

מיועד לזהות שינויים בכיוון האנכי. הוא מזזה שינויים בעוצמת הפיקסלים מלמעלה למטה או מלמטה למעלה (כמו קצנות אופקיים).

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

cv2.covertScaleAbs(gradient)

הfonקציה **cv2.covertScaleAbs** – משמשת להמרת ערכי המטריצה לערכים מוחלטים (אבסולוטיים) ומיפויים לערכים שלמים בין 0 ל-255 על מנת שנוכל להציג את התוצאה של הגרדיינט כתמונה.

איך היא עובדת בפועל:

הfonקציה ממירה כל ערך שהתקבל בנגזרת לחיבוי (חישוב ערך מוחלט) ואז ממירה את כל הערכים ל-0-255 על ידי נרמול (אם יש ערכים עשרוניים היא גם מעגלת).

הקבועים EVENT_LBUTTONDOWN ו- cv2.EVENT_LBUTTONUP

הקבוע **cv2.EVENT_LBUTTONDOWN** והקבוע **cv2.EVENT_LBUTTONUP** – מייצגים אירועי של העכבר - Mouse Events – לחיצות של העכבר על המסך או שחרור, בעצםאפשרים לדעת מה המשתמש לחץ.

cv2.setMouseCallback(window_name, onMouse)

fonקציה **cv2.setMouseCallback** – משמשת להרצת הfonקציה onMouse כאשר יש אירועי עכבר בחלון .window_name

cv2.waitKey(1)

הfonקציה **cv2.waitKey** – מחכה 1 אלף שניות כדי לקבל קלט מהמשתמש לפני שימושה לעדכן את התמונה ולרענן את המסך.

cv2.GaussianBlur(src, ksize, sigmaX, sigmaY, borderType)

הfonקציית **cv2.GaussianBlur** – משמשת להפעלת טשטוש גאוסי על תמונה. היא מבצעת את זה על ידי פילטר המבוסס על פונקציית גאוס, שמטשטש את התמונה על ידי החלקה של הפיקסלים.

פרמטרים:

1. src - תמונה הקלט (source) שעליה רוצים להפעיל את הפילטר.
2. ksize - גודל הגרען – (kernel size) זהו הגודל של המטריצה שהחלקה את הפיקסלים. זהו פרמטר בצורת זוג ערכים (רוחב, גובה), למשל (5, 5).
3. Xsigma - סטיית תקן (standard deviation) בציר ה-X, שקובעת את כמות הטשטוש בכיוון זה.
4. YSigma (אופציונלי) - סטיית תקן בציר ה-Y, אם לא מסופקת, היא תהיה שווה ל-Xsigma.
5. borderType (אופציונלי) - סוג הטיפול בגבולות התמונה. ברירת המחדל היא cv2.BORDER_REFLECT_101, cv2.BORDER_DEFAULT – אשר יוצר השתקפות של הפיקסלים של התמונה בגבולות לצורך הפעלת הקונבולוציה.

איך היא עובדת בפועל:

מתבצעת פעולה קונבולוציה על התמונה לפי גודל הפילטר הנבחר (כל שיורר גדול התמונה תהיה חלקה יותר), כאשר הפילטר מבטא את הנוסחה של ההתפלגות הגaussית:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

פילטר של 3x3:

	1	2	1
1/16	2	4	2
	1	2	1

cv2.addWeighted(src1, alpha, src2, beta, gamma)

הfonקציית **cv2.addWeighted** – משמשת למיזוג שתי תמונות או ערכאים בצורה משוקלת, כך שכל תמונה מקבלת משקל מסוים.

פרמטרים:

1. src1 - התמונה או העrazץ הראשון. זהו מערך של פיקסלים (יכול להיות תמונה צבעונית או בגוני אפור).
2. alpha - המשקל שמכפל בערכי הפיקסלים של התמונה הראשונה **src1**. הוא קובע את מידת ההשפעה של התמונה הראשונה בתוצאה הסופית.
3. src2 - התמונה או העrazץ השני. כמו **src1**, זהו מערך של פיקסלים.
4. beta - המשקל שמכפל בערכי הפיקסלים של התמונה השנייה **src2**. הוא קובע את מידת ההשפעה של התמונה השנייה בתוצאה הסופית.
5. gamma - קבוע שנוסף לכל הפיקסלים אחרי חישוב השקלול של שתי התמונות. זהו ערך מסוים קבוע שמתוויף לתוצאה הסופית. לדוגמה, אם **gamma = 0**, אז אין שינוי בתוצאה הסופית אחריו השקלול.

הfonקציה מבצעת את החישוב הבא עבור כל פיקסל בתוצאה:

$$\text{output}(x, y) = \alpha \cdot \text{src1}(x, y) + \beta \cdot \text{src2}(x, y) + \gamma$$

שילוב שתי הפונקציות האחרונות מאפשר ליצור את התהיל'ר Unsharp Mask. התהיל'ר מאפשר חידוד תמונה באמצעות שילוב בין תמונה מטושטשת לתמונה המקורית. הרעיון הוא להציג את הקצוות בתמונה על ידי הפחיתת התמונה המטושטשת מהתמונה המקורית, מה שמוביל להדגשת האזוריים שבהם יש שינויים חדים בעוצמת הפיקסלים (קצוזות).

GaussianBlur מטושטש את התמונה המקורית באמצעות טשטוש גאוס. `cv2.addWeighted` משלב את התמונה המקורית עם המטושטשת בצורה משקללית באופן הבא: נתן משקל גבוה לתמונה המקורית, נתן משקל שלילי לתמונה המטושטשת, וכך זה מוסיף את הקצוות בתמונה על ידי החסרת החלקים המטושטשים.

cv2.split(img)

הפונקציה `cv2.split` – משמשת לפיצול תמונה הקלט לעורצים מהם היא מורכבת. לכן, אם היא בפורמט RGB נקבל את העורצים R G B לפי הסדר. אם BGR אז נקבל את העורצים B G R לפי הסדר. אם grayscale אז יתקבל העורץ הבודד ממנו היא מורכבת.

cv2.merge(img_channels)

הפונקציה `cv2.merge` – על אותו עיקנון של פונקציית `cv2.split`, `cv2.merge` עשויה את אותו התהיל'ר רק הפור – מאחד את העורצים הניטנים לתמונה יחידה.

cv2.medianBlur(src, ksize)

הפונקציה `cv2.medianBlur` – משמשת ליישום פילטר חיצוני (Median Filter) על תמונה. מטרת הפילטר החיצוני היא להפחית רעש בתמונה על ידי החלקת הפיקסלים, כאשר הרעך של כל פיקסל בתמונה מחושב על ידי מציאת הרעך החיצוני של הפיקסלים בסביבה הקרובה.

פרמטרים:

1. `src` - תמונה הקלט (source) שעלייה רצים להפעיל את הפילטר.
2. `ksize` - גודל הגרעין – (kernel size) זהו הגודל של המטריצה שמחילה את הפיקסלים. גודל הגרעין קובע את גודל השכונה שבה נבחרים הפיקסלים לחישוב החיצוני. ככל שהמספר גדול יותר, כך ההחלקה תהיה חזקה יותר.

AIR היא עובדת בפועל:

בדומה ל-`GaussianBlur`, מתבצעת פעולה קונבולוציה על התמונה לפי גודל הפילטר הנבחר (כל שייתר גדול התמונה תהייה חלקה יותר). היא מחליפה כל פיקסל בתמונה בערך החיצוני של הפיקסלים שבסביבתו, לפי גודל הגרעין שנבחר. כאן, הטיפול בגבולות הוא רק אפשרויות ברירת המחדל - `cv2.BORDER_REFLECT_101` – אשר יוצר השתקפות של הפיקסלים של התמונה בגבולות לצורכי הפעלת הקונבולוציה.

התמונה המשויכת אליו (DIP_811):



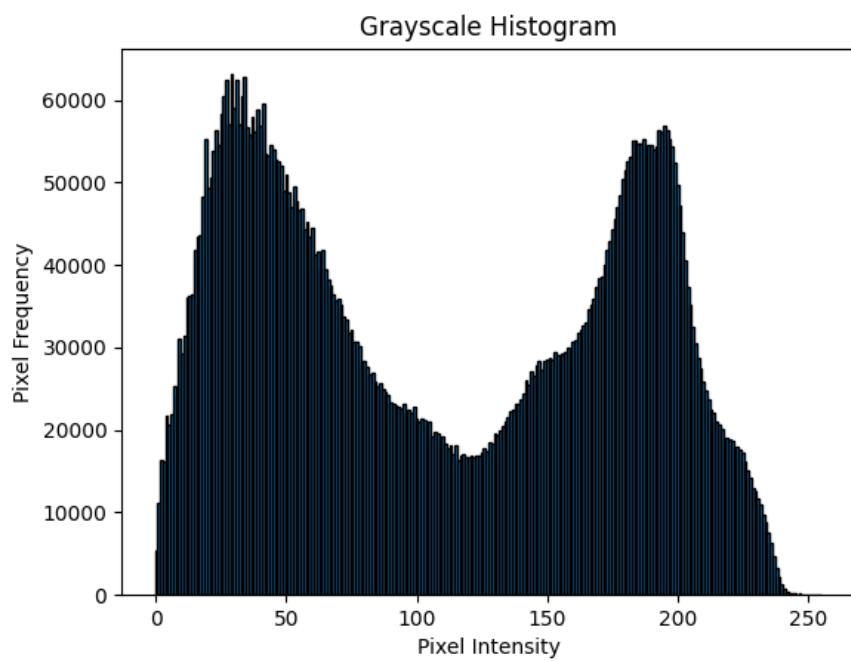
א. הצגה וניתוח של התמונה המקורית:

א. פלט פונקציה:

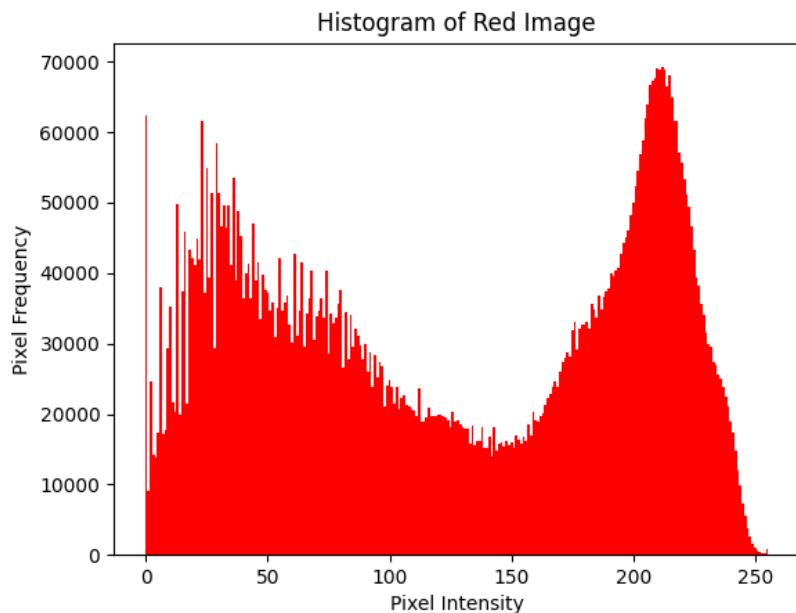
:grayimg.png .1



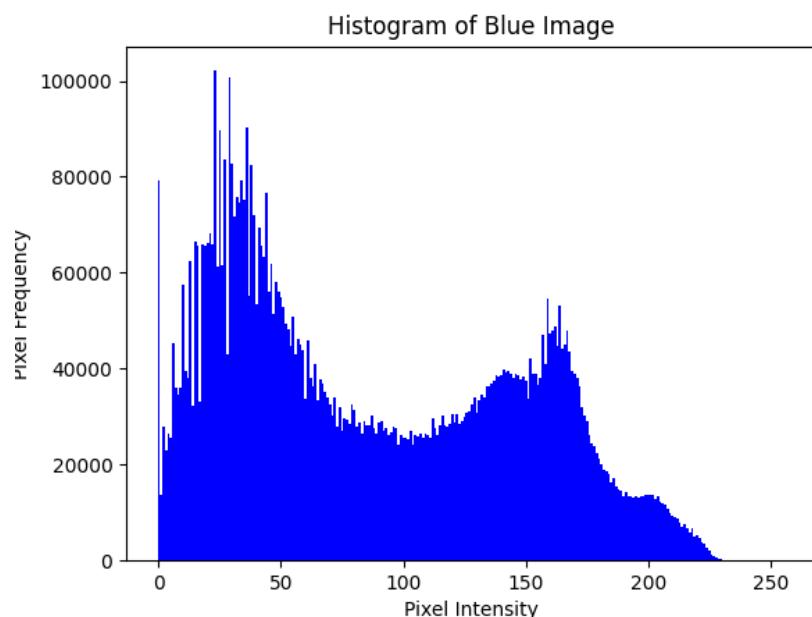
:gray_histogram.png .2



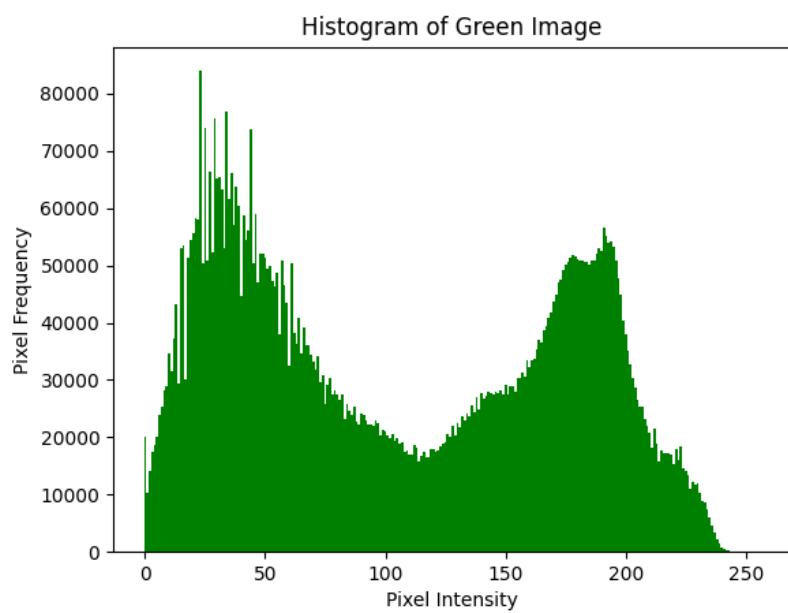
:red_histogram.png .3



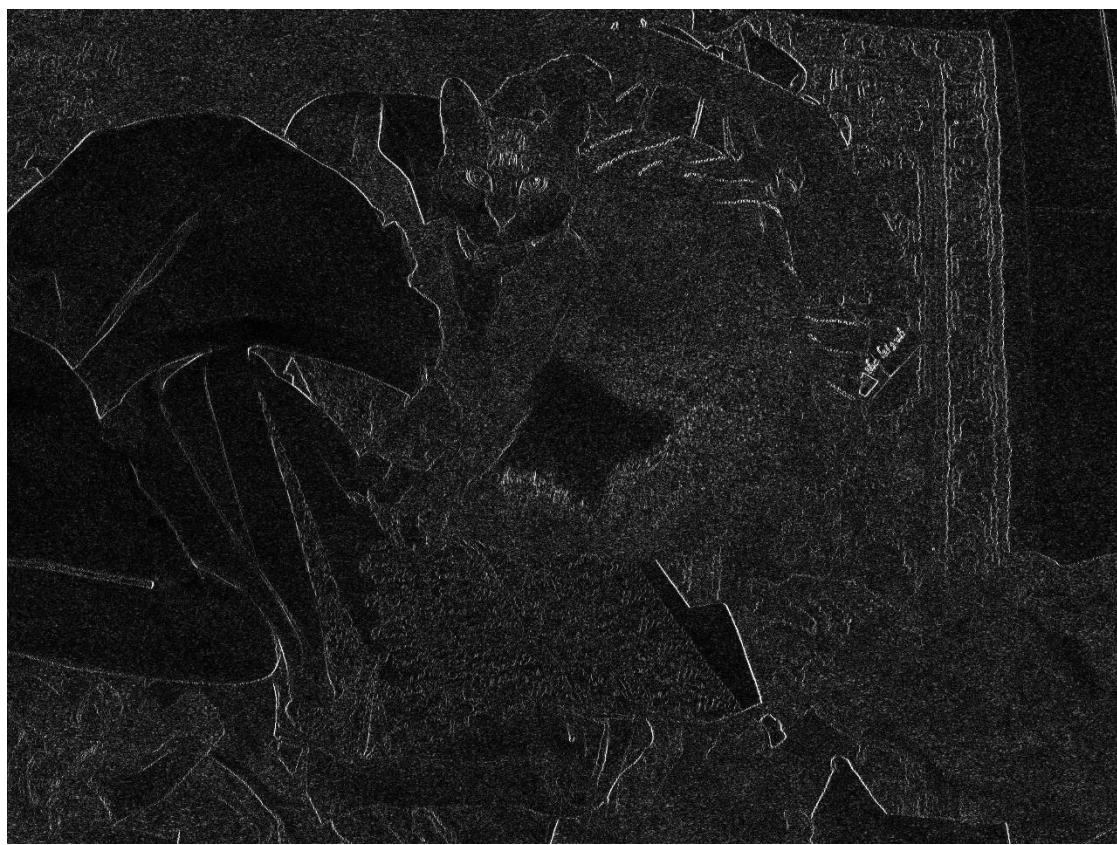
:blue_histogram.png .4



:green_histogram.png .5



:gradient_x.png .6



:gradient_y.png .7



:gradient_intensity.png .8



```

def analyze_image(input_image_path: str, output_folder_path: str) -> None:
    """
    Analyzes an input image and generates several outputs including grayscale conversion,
    histograms, and Sobel gradient calculations.

    Args:
        input_image_path (str): The path to the input image file.
        output_folder_path (str): The path to the folder where the analysis results will be saved.

    The function performs the following operations:
    1. Create a grayscale format (if not inputted already as one)
    2. Create a grayscale histogram
    3. Create a RGB histograms (if the inputted image is RGB)
    4. Calculate gradients X, Y and intensity of the image
    """

    # Load the image without any changes
    img = cv2.imread(input_image_path, cv2.IMREAD_UNCHANGED)
    if img is None:
        print(f"Error: Could not open or find the image.")
        return

    # Check if the image is grayscale or colored
    if len(img.shape) == 2:
        # print("The image is grayscale.")
        is_colored = False
    elif len(img.shape) == 3 and img.shape[2] == 3:
        # print("The image is colored.")
        is_colored = True
    else:
        print("The image format is not recognized.")
        return

```

```

if is_colored:
    # Convert the image to grayscale using cv2 if it's colored
    gray_image = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Explanation about that function is at the beginning of the Part 2 in the PDF file
    img_bgr = img
else:
    # If the image is already grayscale, use the loaded image
    gray_image = img

# Save the grayscale image
cv2.imwrite(os.path.join(output_folder_path, 'grayimg.png'), gray_image)

plt.clf()

# Calculate the histogram of the image in grayscale format
hist, bin_edges = np.histogram(gray_image, bins=np.arange(257))
plt.figure()
plt.bar(bin_edges[:-1], hist, width=1, edgecolor='black')
plt.xlabel('Pixel Intensity')
plt.ylabel('Pixel Frequency')
plt.title('Grayscale Histogram')

# Save the histogram plot
plt.savefig(os.path.join(output_folder_path, 'gray_histogram.png'))
plt.close()

if is_colored:
    # Split the BGR image into its red, green, and blue channels
    blue_channel, green_channel, red_channel = cv2.split(img_bgr)

    # Calculate and plot the histograms for each channel
    for channel, color, name in zip([blue_channel, green_channel, red_channel], ['blue', 'green', 'red'], ['Blue', 'Green', 'Red']):
        plt.clf()
        hist_channel, bin_edges_channel = np.histogram(channel, bins=np.arange(257))
        plt.figure()
        plt.bar(bin_edges_channel[:-1], hist_channel, width=1, color=color)
        plt.xlabel('Pixel Intensity')
        plt.ylabel('Pixel Frequency')
        plt.title(f'Histogram of {name} Image')

        # Save the histogram plot
        plt.savefig(os.path.join(output_folder_path, f'{color}_histogram.png'))
        plt.close()

```

```

# Calculate the gradient in the x direction using the Sobel operator
gradient_x = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=3) # Explanation about that function is at the beginning of the Part 2 in the PDF file
abs_gradient_x = cv2.convertScaleAbs(gradient_x) # Explanation about that function is at the beginning of the Part 2 in the PDF file
cv2.imwrite(os.path.join(output_folder_path, 'gradient_x.png'), abs_gradient_x)

# Calculate the gradient in the y direction using the Sobel operator
gradient_y = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=3)
abs_gradient_y = cv2.convertScaleAbs(gradient_y)
cv2.imwrite(os.path.join(output_folder_path, 'gradient_y.png'), abs_gradient_y)

# Calculate the gradient magnitude (intensity)
gradient_magnitude = np.sqrt(np.square(gradient_x) + np.square(gradient_y))
abs_gradient_magnitude = cv2.convertScaleAbs(gradient_magnitude)
cv2.imwrite(os.path.join(output_folder_path, 'gradient_intensity.png'), abs_gradient_magnitude)

```

```

def main():

    # Get current file location
    script_dir = os.path.dirname(__file__)

    # Create global variable for coordinates
    global coordinates

    # Part A -----
    # Initialize paths and namings
    image_path = os.path.join(script_dir, 'DIP_811.png')
    part_a_folder_name = "part-a"

    # Create the dedicated folder if missing
    output_folder_path = os.path.join(script_dir, 'output-images', part_a_folder_name)
    if not os.path.exists(output_folder_path):
        os.makedirs(output_folder_path)

    analyze_image(image_path, output_folder_path)

```

ב. הסברים:

- gray.png

מה רואים בתמונה: רואים את הגרסה המומרת של התמונה המקורית לגוני אפור. כל פיקסל מייצג בהירות בין שחור לבן, כך שההתמונה נראית כשילוב של גוונים שונים של אפור.

כיצד הפלט מתקשר לתמונה המקורית: הפלט שומר על המבנה והפרטים של התמונה המקורית, אבל מסיר את הצבעים. הבהירות של הפיקסלים מייצגת את הערכים המקוריים של הצבעים, מה שמאפשר להבין את העוצמה והטקסטורה של האובייקטים בתמונה בלי' הצבעים.

- gray_histogram.png

מה רואים בתמונה: רואים היסטוגרמָה של עוצמת הפיקסלים בתמונה בגוני אפור. הציר האופקי מייצג את רמות הבהירונות (מ-0, שחור, עד 255, לבן), והציר האנכי מייצג את מספר הפיקסלים בכל רמת בהירות.

כיצד הפלט מתקשר לתמונה המקורית: היסטוגרמָה מספקת נתונים על פיזור הבהירונות בתמונה המקורית. הפלט עוזר להבין כמה מהתמונה כהה או בהירה, ובאיזה עצומות בהירות קיימים הפיקסלים.

- red_histogram.png

מה רואים בתמונה: רואים היסטוגרמָה של עוצמת הפיקסלים בעורץ האדום של התמונה המקורית. הציר האופקי מייצג את רמות העוצמה של הצבע האדום (מ-0, ללא אדום, עד 255, אדום מלא), והציר האנכי מייצג את מספר הפיקסלים בכל רמת עצמה.

כיצד הפלט מתקשר לתמונה המקורית: הפלט מראה את ההתפלגות של הצבע האדום בתמונה המקורית. היסטוגרמָה זו מסייעת להבין את כמותו הצבע האדום בתמונה ואת הפיזור שלו, וכיצד הצבע האדום תורם להארה ולגוונים הכלליים של התמונה.

- blue_histogram.png

מה רואים בתמונה: רואים היסטוגרמָה של עוצמת הפיקסלים בעורץ הכחול של התמונה המקורית. הציר האופקי מייצג את רמות העוצמה של הצבע הכחול (מ-0, ללא כחול, עד 255, כחול מלא), והציר האנכי מייצג את מספר הפיקסלים בכל רמת עצמה.

כיצד הפלט מתקשר לתמונה המקורית: הפלט מראה את ההתפלגות של הצבע הכחול בתמונה המקורית. היסטוגרמָה נותנת נתונים לגבי כמה הצבע הכחול נוכח בתמונה, ומה הפיזור שלו. זה עוזר להבין את התרומה של הכחול לגוונים הכלליים של התמונה ולניגודיות שלה.

- green_histogram.png

מה רואים בתמונה: רואים היסטוגרמָה של עוצמת הפיקסלים בעורץ הירוק של התמונה המקורית. הציר האופקי מייצג את רמות העוצמה של הצבע הירוק (מ-0, ללא י록, עד 255, י록 מלא), והציר האנכי מייצג את מספר הפיקסלים בכל רמת עצמה.

כיצד הפלט מתקשר לתמונה המקורית: הפלט מציג את ההתפלגות של הצבע הירוק בתמונה המקורית. היסטוגרמָה מספקת מידע על כמות הצבע הירוק והפיזור שלו בתמונה, ומראה כיצד הצבע הירוק תורם לאיכות הגוונים והניגודיות בתמונה.

- gradient_x.png

מה רואים בתמונה: רואים תמונה המציגה את הגרדיאנט של הפיקסלים בכיוון האופקי (בכיוון ה-X). התמונה מדגישה את השינויים בעוצמת הפיקסלים לאורך כיוון זה, כשהאזורים עם הבדל חד בערכים נראים כקוויים בהירים יותר.

כיצד הפלט מתקשר לתמונה המקורית: הפלט מדגיש את הקצוות והמעברים החדים בכיוון האופקי של התמונה המקורית.

- gradient_y.png

מה רואים בתמונה: רואים תמונה המציגה את הגרדיאנט של הפיקסלים בכיוון האנכי (בכיוון ה-Y). התמונה מדגישה את השינויים בעוצמת הפיקסלים לאורך כיוון זה, כשהאזורים עם שינוי חד בערכים נראים כקוויים בהירים יותר.

כיצד הפלט מתקשר לתמונה המקורית: הפלט מדגיש את הקצוות והמעברים החדים בכיוון האנכי של התמונה המקורית.

- gradient_intensity.png .8

מה רואים בתמונה: רואים את התמונה לאחר חישוב הגרדיינט הכללי, שהוא השילוב של הגרדיינטים בכיוונים האופקי (X) והאנכי (Y). התמונה מציגה את העוצמות הכוללות של השינויים בבהירות, ומדגישה את הקצוות והמעברים החדים בתמונה.

כיצד הפלט מתקשר לתמונה המקורית: הפלט מדגיש את כל הקצוות החדים והמעברים בתמונה המקורית, ללא קשר לכיוון השינוי. זה עוזר לזהות את הצורות והמבנה העיקריים בתמונה בצורה מדויקת יותר.

ב. תיקון ושיפור התמונה המקורי:

a. - first_op(input_image_path, output_folder_path)

בחرتني לבצע חיטור של התמונה (1), מכיוון שיש הרבה אלמנטים מסביב לחתול שאינם רלוונטיים. החתול נראה כאובייקט המרכזי והחשוב בתמונה, ולכן רציתי להתמקד בו בלבד. בפעם הראשונה נפתח חלון שמאפשר לי לבחור את האזור הרצוי לחיטור. לאחר שבחרתי את האזור, הדפסתי את הקואורדינטות ושמרתי אותן בקובץ כך שבחרצאות הבאות האזור יחתוך אוטומטית מבלתי שייה צורך לבחור אותו שוב.

התמונה המתקבלת:



השני לעומת התמונה המקורי הוא שכעת רק חלק מן התמונה שמור, ושאר האזורים נחתכו. הפרמטרים שהשתמשתי בפועל זה את הם הקואורדינטות הסופיות של הריבוע שאוינו חתכל' מהתמונה - (2460, 1650), (900, 160)).

```

def crop_image(image : np.ndarray, top_left = None, bottom_right = None):
    """
    Crops an image based on user-selected top-left and bottom-right coordinates or uses provided coordinates.

    Args:
        image (np.ndarray): The input image to be cropped.
        top_left (tuple, optional): Coordinates of the top-left corner for cropping (x, y). Defaults to None.
        bottom_right (tuple, optional): Coordinates of the bottom-right corner for cropping (x, y). Defaults to None.

    The function performs the following steps:
    1. If no coordinates are provided, displays the original image and allows the user to select
       the top-left and bottom-right corners by clicking on the image.
    2. If coordinates are provided, uses them directly to crop the image.
    3. Crops the image according to the selected or provided coordinates.
    4. Returns the cropped image along with the coordinates of the selected top-left and bottom-right corners.
    """

    if top_left is None and bottom_right is None:

        window_name = display_image(image, "Original Image")

        # Mouse callback function for selecting points
        def select_points(event, x, y, flags, param):
            nonlocal top_left, bottom_right

            if event == cv2.EVENT_LBUTTONDOWN: # Explanation about that function is at the beginning of the Part 2 in the PDF file
                top_left = (x, y)

            elif event == cv2.EVENT_LBUTTONUP:
                bottom_right = (x, y)
                cv2.destroyAllWindows()

        # Register the mouse callback
        cv2.setMouseCallback(window_name, select_points) # Explanation about that function is at the beginning of the Part 2 in the PDF file

```

```

# Wait until the user selects the points
while True:
    if top_left is not None and bottom_right is not None:
        break
    cv2.waitKey(1) # Explanation about that function is at the beginning of the Part 2 in the PDF file

# else:
#     print(f"Selected Top-left corner: {top_left}")
#     print(f"Selected Bottom-right corner: {bottom_right}")

# Crop the image
cropped_image = image[top_left[1]:bottom_right[1], top_left[0]:bottom_right[0]]

# display_image(cropped_image, "Cropped Image")
# cv2.waitKey(0)
# cv2.destroyAllWindows()

# Return the cropped image and the coordinates of the vertices
return cropped_image, (top_left, bottom_right)

```

```
def first_op(input_image_path: str, output_folder_path: str) -> np.ndarray:
    """
    Loads an image from the specified path, performs a cropping operation using predefined or user-selected
    coordinates, and saves the cropped image to the specified output folder.

    Args:
        input_image_path (str): The file path to the input image.
        output_folder_path (str): The directory where the cropped image will be saved.
    """

    # Load the image
    img_bgr = cv2.imread(input_image_path, cv2.IMREAD_UNCHANGED)

    if img_bgr is None:
        print(f"Error: Could not open or find the image.")
        return

    cropped_img, _ = crop_image(img_bgr, coordinates[0], coordinates[1])

    # Save the cropped image
    cv2.imwrite(os.path.join(output_folder_path, 'img_firstop.png'), cropped_img)

    return cropped_img
```

- second_op(input_image_path, output_folder_path) .b בחרתי לבצע פעולה סבiba – Unsharp Mask (8), מכיוון שרציתי להבליט את הפרטים החשובים של האובייקט המרכזי – החתול. חידוד בעזרת Unsharp Mask אפשר להdagish קצוץ וקווים, ותרם להבלת הפרטים בתמונה והפרק אותה לברורה יותר. באופן זה, ניתן לבדוק התמונה או לזיהוי האובייקט בצורה טובה יותר.

השתמשתי בטכנית להבלת פרטים בתמונה על ידי שילוב בין התמונה המקורי וגרסתה מטושטשת שלה. קודם, יצרתי גרסה מטושטשת של התמונה באמצעות Gaussian Blur. לאחר מכן, שילבתי את התמונה המקורי עם התמונה המטושטשת בעזרת פונקציית cv2.addWeighted(), כדי להdagish את הקצוץ ולהבליט פרטים חשובים. בנוסף, על מנת לקבל חידוד אמיתי יותר, חילקתי את התמונה לעורכי הצעדים שלו והפעלת עלי כל צבע בנפרד את החידוד. השימוש בפרמטרים של טשטוש (blur_strength) ואלפא (alpha) מאפשר שילטה ברמת החידוד ובאייזון בין הפרטים המודגשים לתמונה המטושטשת.

התמונה המתקבלת:



התמונה השנייה חדה יותר מהראשונה. החידוד מדגיש את הקצוץ ומבליט פרטים בתמונה, במיוחד סביב החתול, מה שMBOLIL לניגודיות גבוהה יותר ולהבלת הפרטים הקטנים.

בפעולת unsharp mask קיימים שני פרמטרים – blur_strength ו-alpha :
blur_strength = 5 - רמת הטשטוש הזו מספקה כדי להdagish את הקצוץ בתמונה, תוך שמירה על פרטים חשובים מבליל檠ם לטשטוש יתר.
alpha = 1.5 - הערך זה נותן משקל מסווגות יותר לתמונה המקורי, מה שMBOLIL לחידוד מודגש, אך עדין שומר על מראה טבעי. השילוב בין הטשטוש והחידוד עוזר להבליט את החתול בצורה מיטבית.

```
def unsharp_mask_single_channel(input_channel: np.ndarray, blur_strength=15, alpha=1.5):
    """
    Applies an unsharp mask to a single channel of an image to enhance its sharpness.

    Args:
        input_channel (np.ndarray): The input channel (grayscale or a single color channel) to be sharpened.
        blur_strength (int, optional): The strength of the Gaussian blur applied to create the mask.
            Must be an odd number. Defaults to 15.
        alpha (float, optional): The scaling factor to control the sharpness. A higher value increases sharpness.
            Defaults to 1.5.

    """
    blurred_channel = cv2.GaussianBlur(input_channel, (blur_strength, blur_strength), 0) # Explanation about that function is at the beginning of the Part 2 in the PDF file
    sharpened_channel = cv2.addWeighted(input_channel, 1 + alpha, blurred_channel, -alpha, 0) # Explanation about that function is at the beginning of the Part 2 in the PDF file
    return sharpened_channel
```

```
def second_op(input_image_path: str, output_folder_path: str) -> np.ndarray:
    """
    Applies an unsharp mask to an image to enhance its sharpness, whether it is grayscale or RGB.

    Args:
        input_image_path (str): The file path to the input image.
        output_folder_path (str): The directory where the sharpened image will be saved.
    """

    img = cv2.imread(input_image_path, cv2.IMREAD_UNCHANGED)
    if img is None:
        print(f"Error: Could not open or find the image.")
        return

    # Check if the image is grayscale or RGB
    if len(img.shape) == 2: # Grayscale image
        sharpened_image = unsharp_mask_single_channel(img, blur_strength=5, alpha=1.5)
    else: # RGB image
        # Apply unsharp mask to each channel (R, G, B) separately
        # Explanation about that function is at the beginning of the Part 2 in the PDF file
        channels = cv2.split(img) # Split the image into R, G, B channels
        sharpened_channels = []
        for channel in channels:
            sharpened_channels.append(unsharp_mask_single_channel(channel, blur_strength=5, alpha=1.5))
        # Explanation about that function is at the beginning of the Part 2 in the PDF file
        sharpened_image = cv2.merge(sharpened_channels) # Merge the sharpened channels back together

    # Save the sharpened image
    output_image_path = os.path.join(output_folder_path, 'img_secondop.png')
    cv2.imwrite(output_image_path, sharpened_image)

    return sharpened_image
```

.c - third_op(input_image_path, output_folder_path) בחרתי לבצע פעולה סבביה – Median Filter (9), מכיוון שבתמונה המעודכנת יש קצת רעשים והיא מצינית להסרת רעשים תוך שמירה על הקצוות של האובייקטים בתמונה. על ידי החלפת כל פיקסל בערך הממוצע שבסביבו זה עוזר ליצור תמונה חלקה יותר תוך שמירה על קווי המתאר והפרטים החשובים של האובייקט המרכזי – החתול, מבלי לפגוע באיכות הקצוות. תחילה, בדקתי אם התמונה היא בגוני אפור או צבעונית, וביצעת את הסינון בהתאם. במקרה של תמונה בגוני אפור, הפעלתית את המסנן ישרוט על כל הפיקסלים בתמונה, ואילו במקרה של תמונה צבעונית, חילקתי את התמונה לעוצמי הצבע (אדום, ירוק, כחול), החלטתי את המסנן על כל עורך בנפרד, ולאחר מכן מיזגתי אותו חזקה לתמונה אחת. כך שמרתי על כל עורך צבע בנפרד כדי להבטיח את איכות הסינון והפרטים בתמונה הסופית.

התמונה המתקבלת:



הפילטר הביא להפחחת רעשים בתמונה בהשוואה לתמונה הקודמת. התוצאה היא תמונה חלקה יותר, שבה הרעש מופחת, במיוחד באזוריים עם שינוי עצמה קלים (כגון הפרווה של החתול והרכע). ביחס לתמונה הקודמת, הרעש קטן משמעותית, אך הקצוות והפרטים החשובים עדין נשמרו.

בפועל הפקטור היחיד שהוא זה גודל חלון הסינון – **kernel_size**:
בחרתי אותו להיות בגודל המינימלי (3) כדי לא להחליק את התמונה יתר על המידה.

```
def apply_median_filter(input_image: np.ndarray, kernel_size=3):
    """
    Applies a median filter to an input image to reduce noise, handling both grayscale and RGB images.

    Args:
        input_image (np.ndarray): The input image, either in grayscale or RGB format.
        kernel_size (int, optional): The size of the kernel to be used for the median filter.
            | | | | | | | | It must be an odd number. Defaults to 3.
    """

    if len(input_image.shape) == 2: # Grayscale image
        denoised_image = cv2.medianBlur(input_image, kernel_size)
    else: # RGB image
        channels = cv2.split(input_image) # Split into R, G, B channels
        denoised_channels = []
        for channel in channels:
            denoised_channels.append(cv2.medianBlur(channel, kernel_size)) # Apply median filter to each channel
        denoised_image = cv2.merge(denoised_channels) # Merge back the channels

    return denoised_image
```

```
def third_op(input_image_path: str, output_folder_path: str) -> np.ndarray:
    """
    Loads an image from the specified path, applies a median filter to reduce noise, and saves the result.

    Args:
        input_image_path (str): The file path to the input image.
        output_folder_path (str): The directory where the denoised image will be saved.
    """

    img = cv2.imread(input_image_path, cv2.IMREAD_UNCHANGED)
    if img is None:
        print(f"Error: Could not open or find the image.")
        return

    denoised_image = apply_median_filter(img, kernel_size=3)

    # Save the denoised image
    output_image_path = os.path.join(output_folder_path, 'img_thirdop.png')
    cv2.imwrite(output_image_path, denoised_image)

    return denoised_image
```

התמונה הסופית טוביה יותר מהתמונה המקורי מכיוון שעברנו עליה שלוש פעולות שהביאו לשיפור ניכר:

- **חיתוך התמונה** - התמונה נחתכה כך שהיא מתמקדת באובייקט המרכזי (החתול), ובכך הוסרו פרטים לא רלוונטיים מהרקע.
- **Unsharp Mask** - שיפר את חדות התמונה על ידי הדגשת הקצוות והפרטים החשובים של החתול, מה שהפך את התמונה לברורה יותר.
- **Median Filter** - הסיר רעשים לא רצויים מהתמונה, מה שתרם למראה חלק ונקי יותר, במיוחד באזוריים אחידים.

השיפור הכלל מביא לתמונה שמקדשת יותר, חדה יותר, וברורה יותר עם פחות רעשים.

נשים לב שבפונקציה זו בהרצה הראשונה מצאתי את הקואורדינטות הרלוונטיות ובהרצאות הבאות (הקוד שנותן כאן) שלחותי את הפרמטרים הרלוונטיים – coordinates והפונקציה חתכה ישראת החלק הרצוי בתמונה (crop_image).

```
def fix_image(image_to_fix_path: str, output_folder_path: str):  
    """  
    Performs a sequence of image operations to enhance and clean an image, saving the results of each step.  
  
    Args:  
    image_to_fix_path (str): The file path to the input image that needs fixing.  
    output_folder_path (str): The directory where the results of each operation will be saved.  
    """  
  
    # Operation 1: Cropping the image  
    img_firstop = first_op(image_to_fix_path, output_folder_path)  
  
    # Operation 2: Applying Unsharp Mask filter to sharpen the image  
    second_input_image_path = os.path.join(output_folder_path, 'img_firstop.png')  
    img_secondop = second_op(second_input_image_path, output_folder_path) # Unsharp Mask  
  
    # Operation 3: Applying Median Filter to remove noise from the image  
    third_input_image_path = os.path.join(output_folder_path, 'img_secondop.png')  
    img_thirddop = third_op(third_input_image_path, output_folder_path) # Median Filter  
  
    return img_firstop, img_secondop, img_thirddop
```

```
# Part B -----  
# Initialize paths and namings  
image_to_fix_path = os.path.join(script_dir, 'output-images', part_a_folder_name, 'grayimg.png')  
part_b_folder_name = "part-b"  
  
# Create the dedicated folder if missing  
output_folder_path = os.path.join(script_dir, 'output-images', part_b_folder_name)  
if not os.path.exists(output_folder_path):  
    os.makedirs(output_folder_path)  
  
# Set cropping coordinates  
coordinates = ((900, 160), (2460, 1650))  
  
# Fix the image  
img_firstop, img_secondop, img_thirddop = fix_image(image_to_fix_path, output_folder_path)
```

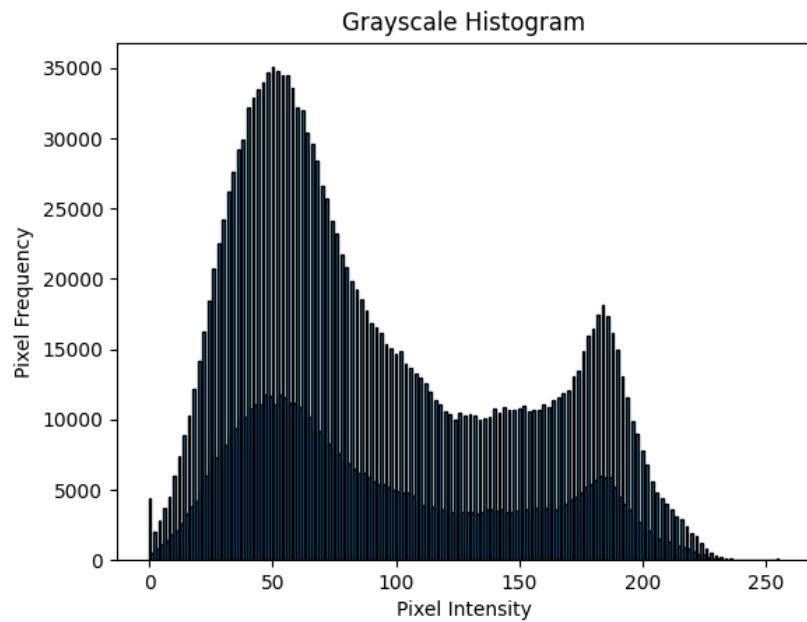
ג. הציגו וניתוח של התמונה המתוקנת:

א. פלטי פונקציה:

:grayimg.png .1



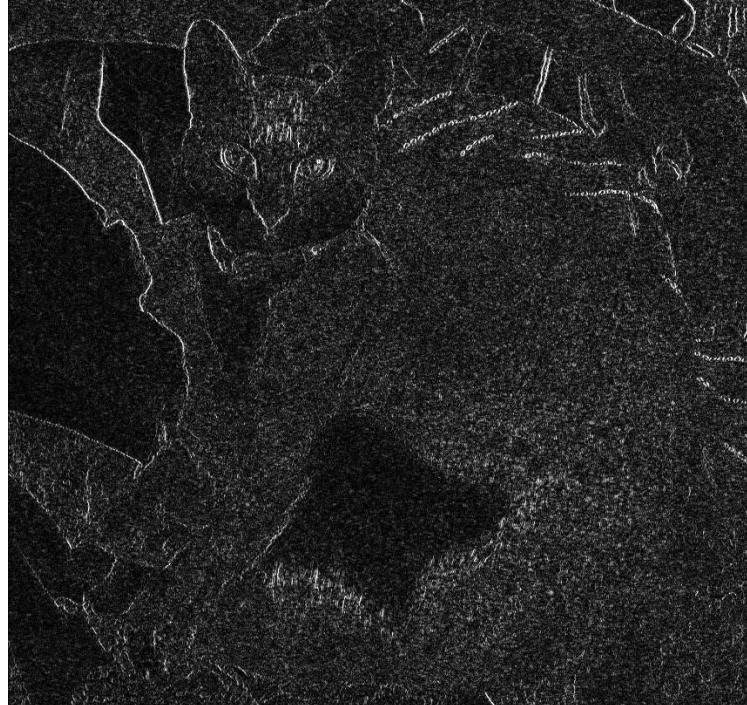
:gray_histogram.png .2



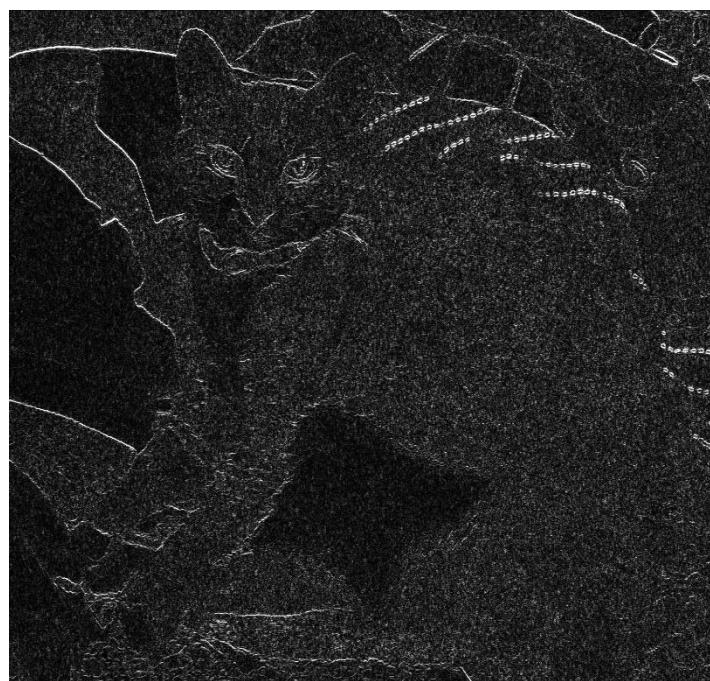
:red_histogram.png .3
:blue_histogram.png .4
:green_histogram.png .5

לא קיימים (הסביר סעיף הבא)

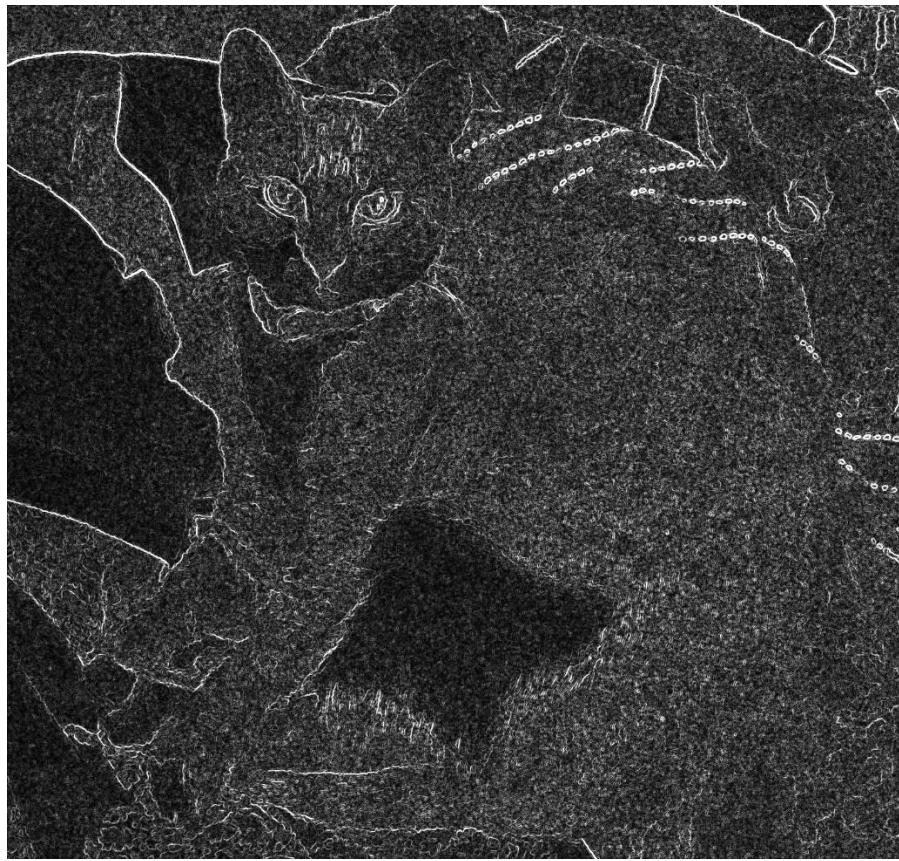
:gradient_x.png .6



:gradient_y.png .7



:gradient_intensity.png .8



כיוון שכתבתי את הקוד בצדקה עיליה ומודולרית, כל מה שהוספה כאן זה רק קריאה חוזרת לפונקציה עם פרמטרים חדשים:

```
# Part C -----
# Initialize paths and namings
image_path = os.path.join(script_dir, 'output-images', part_b_folder_name, 'img_thirdop.png')
part_c_folder_name = "part-c"

# Create the dedicated folder if missing
output_folder_path = os.path.join(script_dir, 'output-images', part_c_folder_name)
if not os.path.exists(output_folder_path):
    os.makedirs(output_folder_path)

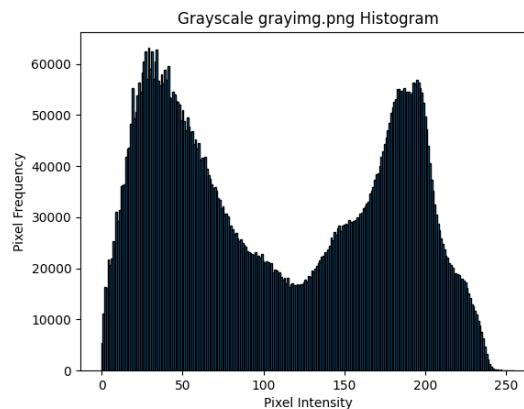
analyze_image(image_path, output_folder_path)
```

ב. הסברים:

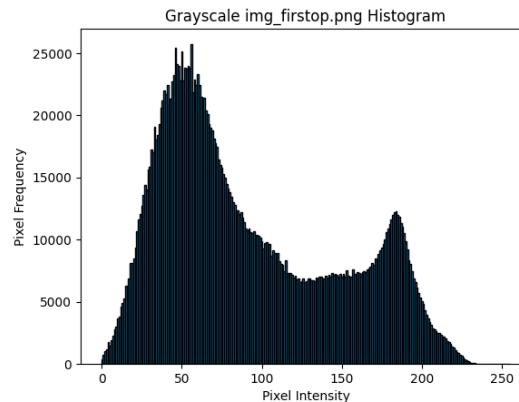
- **grayimg.png** .1

בתמונה החדש החתול נמצא במרכז התמונה ונראה ברור וחד יותר בעזרת חיתוך התמונה, חידוד והסרת הרעשיות ממנה.

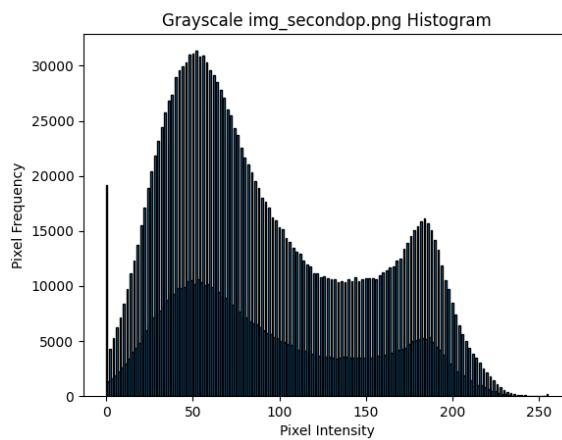
- **gray_histogram.png** .2



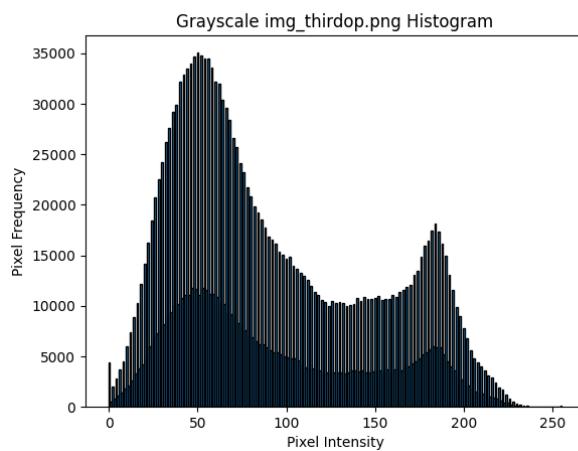
בהתחלתה, התמונה המקורי מציגה שתי גבעות עיקריות בהיסטוגרמה שלה, אחת מייצגת אזוריים כהים (כמו הצללים והבדים הכהים) והשנייה מייצגת אזוריים בהירים (כמו הבדים הלבנים והפראו של החתול). יש ירידה בין שתי הגבעות, מה שמראה על ניגודיות גבוהה בין האזוריים הכהים לבהירים בתמונה.



לאחר החיתוך סביר החתול, ההיסטוגרמה עדין שומרת על אותן גבעות כהות ובהירות, אבל בגלל שהוסר חלק מהתמונה, הגבעות הופכות ליותר מוקדמות. החיתוך צמצם את כמות האזוריים המגווונים, ובעיקר אזוריים בהירים יותר, כמו הבדים, ולכן הגבעה הימנית (פיקסלים בהירים) קטנה.



המעבר לפילטר Unsharp Mask מוגביר את הניגודיות בתמונה ומחדד את הקצוות, מה שמוסיף עוד הבדל בין האזוריים הכהים והבהירים. תוצאה של זה היא הפעת "שכבות" חדשות תחת הגבעות הראשיות בהיסטוגרמה, כאשר כל שכבה מייצגת שינוי עディין יותר של הגוונים שהופכים ליותר מובחנים וחדים.



לאחר הפעלת Median Filter, ההיסטוגרמה נעשית קצר חלקה יותר, שכן המסנן מוחק רעשים קטנים ומחליק את ההבדלים הדקים. עם זאת, המבנה הכללי נשמר – הגבעות העיקריות של כהה ובair נשארות בולטות, והשכבות שהתווסף בעקבות מס'icit ה- Unsharp Mask עדין נוכחות, אבל בצורה פחותה.

- red_histogram.png .3
- blue_histogram.png .4
- green_histogram.png .5

ההיסטוגrames RGB מיועדות לתמונות צבעוניות שבהן לכל פיקסל יש ערכי עצמה נפרדים עבור כל אחד מערוצי הצבע (Red, Green, Blue). במצב זה, ההיסטוגrama נפרדת עבור כל ערוץ עוזרת להבין את ההתפלגות של הצבעים בתמונה. בתמונה בגוני אפור, כל הפיקסלים מוגדרים באמצעות ערך בהירות אחד, ולכן ההיסטוגrama אחת מספקת לייצג את כל ההתפלגות של הבاهירות בתמונה. אין ערכי צבע נפרדים שדרושים ניתוח באמצעות ההיסטוגrames RGB.

- gradient_x.png .6
- gradient_y.png .7
- gradient_intensity.png .8

נראה שבשלושת סעיפים אלו אין שינוי ממשמעותי נראית לעין. נראה כיון שהשניים שבחרתי
היו יחסית עדינים.

ד. סעיף בונוס - תיקון ושיפור התמונה המקורית:
בסעיף זה הרצתי את אותן הפעולות שיצרתי בסעיף ב. – כמובן, חיתוך התמונה, median filter ו-unsharp filter – חיתוך התמונה first_op(input_image_path, output_folder_path) .a

התמונה המתקבלת:



השוני לעומת התמונה המקורי הוא שכעת רק חלק מן התמונה שמור, ושאר האזוריים נחתכו.
הפרמטרים שהשתמשתי בפועלה זאת הם הקואורדינטות הסופיות של הריבוע שאוינו חתכי
מהתמונה - ((1000, 175), (2530, 1660))

Unsharp Mask – second_op(input_image_path, output_folder_path) .b

התמונה המתקבלת:



התמונה השנייה יותר מהראשונה. החידוד מדגיש את הקצוות וմבליט פרטיהם בתמונה, במיוחד סביב החתול, מה שmobail לניגודיות גבואה יותר ולהבלט הפרטים הקטנים.

בפועל unsharp mask קיימים שני פרמטרים – blur_strength ו-alpha:
blur_strength = 5 – רמת הטשטוש הזו מספיקה כדי להציג את הקצוות בתמונה, יותר
שמירה על פרטים חשובים מבלתי Lagerם לטשטוש יתר.
alpha = 1.5 – הערך זה נותן משקל משווה יותר לתמונה המקורית, מה mobail לחידוד
מודגש, אך עדין שומר על מראה טבעי. השילוב בין הטשטוש והחידוד עוזר להבליט את
חתול בצורה מיטבית.

Median Filter – third_op(input_image_path, output_folder_path) .c

התמונה המתקבלת:



הפילטר הביא להפחית רעשים בתמונה בהשוואה לתמונה הקודמת. התוצאה היא תמונה חלקה יותר, שבה הרעש מופחת, במיוחד באזוריים עם שינוי עוצמה קלים (כגון הפרווה של החתול והרקע). ביחס לתמונה הקודמת, הרעש קטן משמעותית, אך הקצאות והפרטימ החשובים עדין נשמרו.

בפועל הפעלת היחידי שהוא זה גודל חלון הסינון – **kernel_size**:
בחרתי אותו להיות בגודל המינימלי (3) כדי לא להחליק את התמונה יתר על המידה.

- התמונה הסופית טוביה יותר מהתמונה המקורי מכיוון שעברנו עליה שלוש פעולות שהביאו לשיפור ניכר:
- **חיתוך התמונה** - התמונה נחתכה כך שהיא מתמקדת באובייקט המרכזי (החתול), ובכך הוסרו פרטים לא רלוונטיים מהרקע.
 - **Unsharp Mask** - שיפר את חדות התמונה על ידי הדגשת הקצאות והפרטימ החשובים של החתול, מה שהפך את התמונה לברורה יותר.
 - **Median Filter** - הסיר רעשים לא רצויים מהתמונה, מה שתרם למראה חלק ונקי יותר, במיוחד באזוריים אחידים.

השיפור הכלול מביא לתמונה שמצוקדת יותר, חדה יותר, וברורה יותר עם פחות רעשים.

אם כן, השתמשתי בפונקציה שיצרתني בסעיפים הקודמים ולכн, השינוי היחידי הוא זימן הפונקציה:

```
# Part D -----
# Initialize paths and namings
image_to_fix_path = os.path.join(script_dir, 'DIP_811.png')
part_d_folder_name = "part-d"

# Create the dedicated folder if missing
output_folder_path = os.path.join(script_dir, 'output-images' , part_d_folder_name)
if not os.path.exists(output_folder_path):
    os.makedirs(output_folder_path)

# Set cropping coordinates
coordinates = ((1000, 175), (2530, 1660))

# Fix the image
img_firstop, img_secondop, img_thirddop = fix_image(image_to_fix_path, output_folder_path)
```

ה. סעיף בונוס - תיקון ושיפור תמונה חדשה:

- בסעיף זה הרצתי את אותן הפעולות שיצרתי בסעיף ב. – כמובן, חיתוך התמונה, unsharp filter ו-.median filter

התמונה המקורית:



– חיתוך התמונה first_op(input_image_path, output_folder_path) .a

התמונה המתקבלת:



השוני לעומת התמונה המקורי הוא שכעת רק חלק מן התמונה שמרו, ושאר האזוריים נחתכו. הפרמטרים שהשתמשתי בפעולה זאת הם הקואורדינטות הסופיות של הריבוע שאוות חתכי מהתמונה - ((870, 1130), (2300, 3090))

Unsharp Mask – second_op(input_image_path, output_folder_path) .b

התמונה המתקבלת:



התמונה השנייה יותר מהראשונה. החידוד מדגיש את הקצוות ומליט פרטים בתמונה, במיוחד סביב האלפקה, מה שmobil לניגדיות גבואה יותר ולהבלת הפרטים הקטנים.

בפועלת unsharp mask קיימים שני פרמטרים – `blur_strenght` ו- `alpha` :
`blur_strenght = 5` – רמת הטשטוש הזו מספיקה כדי להציג את הקצוות בתמונה, תוך שמירה על פרטים חשובים מבלי לגרום לטשטוש יתר.
`alpha = 1.5` – הערך זה נותן משקל משווה יותר לתמונה המקורית, מה שmobil לחידוד מוגש, אך עדין שומר על מראה טבעי. השילוב בין הטשטוש והחידוד עוזר להבלת את האלפקה بصورة מיטבית.

Median Filter – third_op(input_image_path, output_folder_path) .c

התמונה המתקבלת:



הפילטר הביא להפחיתת רעשים בתמונה בהשוואה לתמונה הקודמת. התוצאה היא תמונה חלקה יותר, שבה הרעש מופחת, במיוחד באזוריים עם שינויים עצמאים. ביחס לתמונה הקודמת, הרעש קטן ממשמעותית, אך הקצוות והפרטים החשובים עדין נשמרו.

בפועלה הפעמטר היחיד שהוא זה גודל חלון הסינון – **kernel_size**:
בחרתי אותו להיות בגודל המינימלי (3) כדי לא להחליק את התמונה יתר על המידה.

- התמונה הסופית טוביה יותר מההתמונה המקורי מכיוון שעברנו עליה שלוש פעולות שה宾ו לשיפור ניכר:
- **חיתוך התמונה** - התמונה נחתכה כך שהיא מתמקדת באובייקט המרכזי (האלפקה), ובכך הוסר פרטים לא רלוונטיים מהרקע.
 - **Unsharp Mask** - שיפר את חזרות התמונה על ידי הדגשת הקצוות והפרטים החשובים של האלפקה, מה שהפך את התמונה לבורורה יותר.
 - **Median Filter** - הסיר רעשים לא רצויים מההתמונה, מה שתרם למראה חלק ונקי יותר, במיעוד באזוריים אחידים.

גם כאן, התוספת היחידה לקוד היא זימון הפונקציה עם פרמטרים חדשים:

```
# Part E -----
# Initialize paths and namings
image_to_fix_path = os.path.join(script_dir, 'IMG_811.png')
part_e_folder_name = "part-e"

# Create the dedicated folder if missing
output_folder_path = os.path.join(script_dir, 'output-images' , part_e_folder_name)
if not os.path.exists(output_folder_path):
    os.makedirs(output_folder_path)

# Set cropping coordinates
coordinates = ((870, 1130), (2300, 3090))

# Fix the image
img_firstop, img_secondop, img_thirddop = fix_image(image_to_fix_path, output_folder_path)
```