

**עבודה סופית בקורס "שיטות בעיבוד תמונה וראייה ממוחשבת"
0560.4463**

ד"ר יונתן אוסטרומצקי, גב' ליאורה מזנגיה

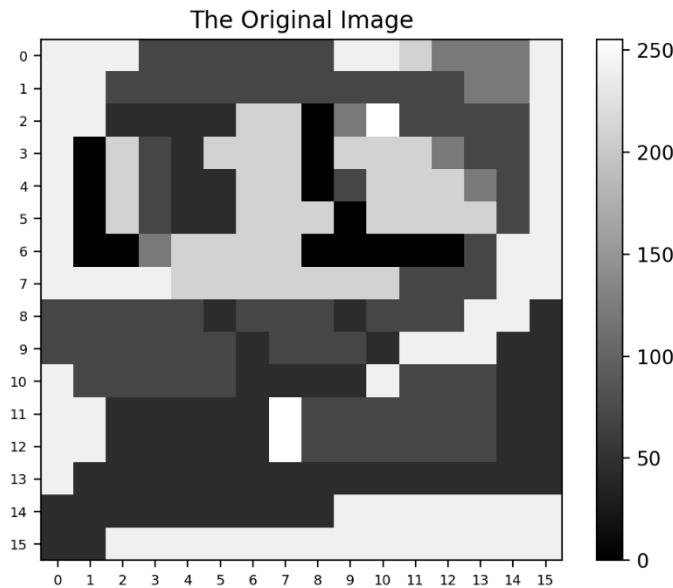
**שם הסטודנט: יובל בקרוב
תעודת זהות: 209424803**

מילואים

חלק ראשון

שאלה 1:

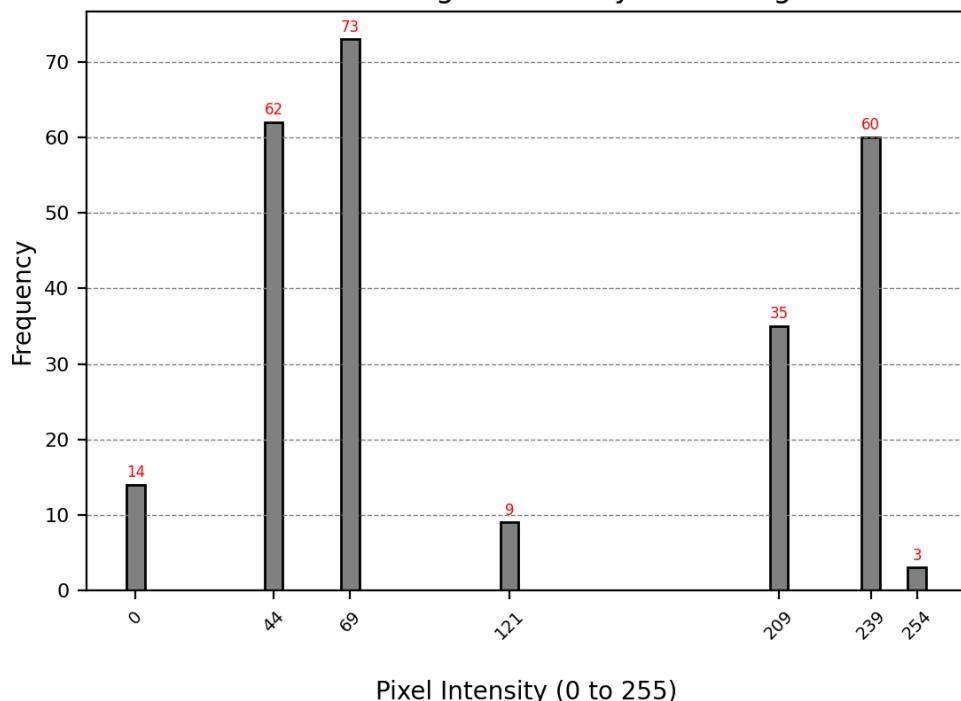
רוכב הקוד בשאלת זו הוא ויזואלייזטור, לכן, לא צירפנו את הקוד ב-PDF אלא רק בתיקייה שהגשנו.
נתבונן בתמונה המקורי:



a. היסטוגרמה 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:

הועל מסנן חציו בגודל 3x3 באופן הבא:

(הנחה שלקחנו – כאשר הפעילו את המסנן על הגבולות של האזור החתוור, 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	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	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	240	240	240	240	240	240	
45	45	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 ניתן להראות כך כתוב רג' נס 3x3.

210	210	210
45	210	210
45	210	210

ולפניהם 210 נס 1x1 :

רמזו נס 3x3 מוקדם סדרה נס 210

בפניהם נס 1x3 :

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

יה קדימה לה 210 ה-2 ו-3 ה-2 וה-4 ה-2 וה-5 ה-2.

210	210	0
210	210	0
210	210	210

ולפניהם 210 נס 1x2 :

רמזו נס 3x3 מוקדם סדרה נס 210

בפניהם נס 1x4 :

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

יה קדימה לה 210 ה-2 ו-3 ה-2 וה-4 ה-2 וה-5 ה-2.

210	0	210
210	0	70
210	210	0

: 1 x 3 figures 210 for 10 years

לפנינו ידועות מטרית 3×3 ו- 120°

பிரமன கீழானு ஸ்ரீத

לעומת זה, מטרת ה- \lim היא לשלוט ב- $f(x)$, כלומר:

0	210	210
0	70	210
210	0	210

$$: 1 \times 4 \text{ ပါတ်မှု } 210 \quad \text{စွဲ } 10 \text{ မှု}$$

לעומת מטרית אובייקט מודולרי

јејтинг и квизи

210	210	210
70	210	210
0	210	210

: 1 x 5 ရက်သာ 210 ရပါင် ၇၀၈

ל-בנין אס-טראנס 3x3 מיל' 120

• 13m ne 13n1 82d8

$$\overrightarrow{0, 70, 210, 210, 210, 210, 210, 210, 210}$$

ג) דנילר הול 210 כ- ערבית ג'ערת תניינן מות (טבאל).

45	210	210
45	210	210
210	210	210

: 2x1 Figure 210 ပါရမ်းကြပ်

לפניהם מושג ב-3x3 מילון

јирим ик ирнији бирд

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

ימד אוניברסיטה הפתוחה 210 כ- 1000 גייר טכני מודול תיכונן

210	210	0
210	210	210
210	210	0

: 2x2 μ g/ml 210 μ g/ml

↳ Final 1D case 16 Jan 3x3 rule

: յ Յ Ր Ո Ր Ա Կ Կ Յ Ն Ո Ւ Ծ Ի Շ Ճ Ճ

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

מג פולחן הלה 210 כ-10% גורר נזק ומות מוגע.

..... תרגום מילוי

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 through 4 are light gray, and Cell 5 is dark gray.
- Row 3: Cells 1 through 4 are light gray, and Cell 5 is dark gray.
- Row 4: Cells 1 through 4 are light gray, and Cell 5 is dark gray.
- Row 5: Cells 1 through 4 are light gray, and Cell 5 is dark gray.

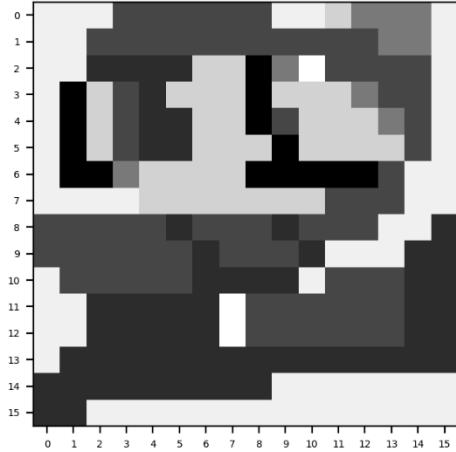
Numerical labels are placed in some of the cells:

- Row 1: All cells contain the value 210.
- Row 2: Cells 1, 2, and 4 contain 210; Cell 3 contains 70; Cell 5 contains 0.
- Row 3: Cells 1, 2, and 4 contain 210; Cell 5 contains 70.
- Row 4: Cells 1, 2, and 4 contain 210; Cell 5 contains 70.
- Row 5: Cells 1, 2, and 4 contain 210; Cell 5 contains 70.

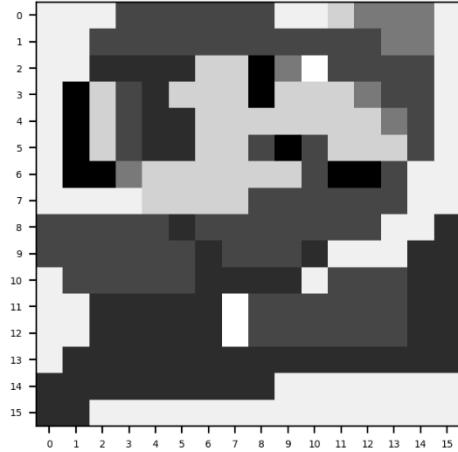
The values 210 and 70 are displayed in red text, while the value 0 is displayed in white text.

Part B - Median Filter

Original Image

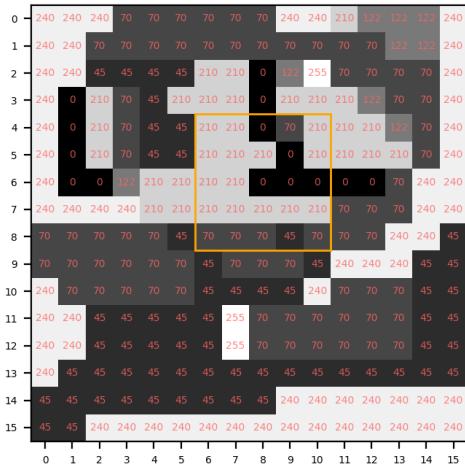


After Median Filter Image

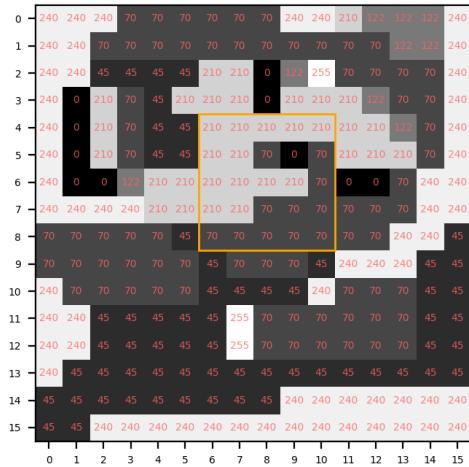


Part B - Median Filter

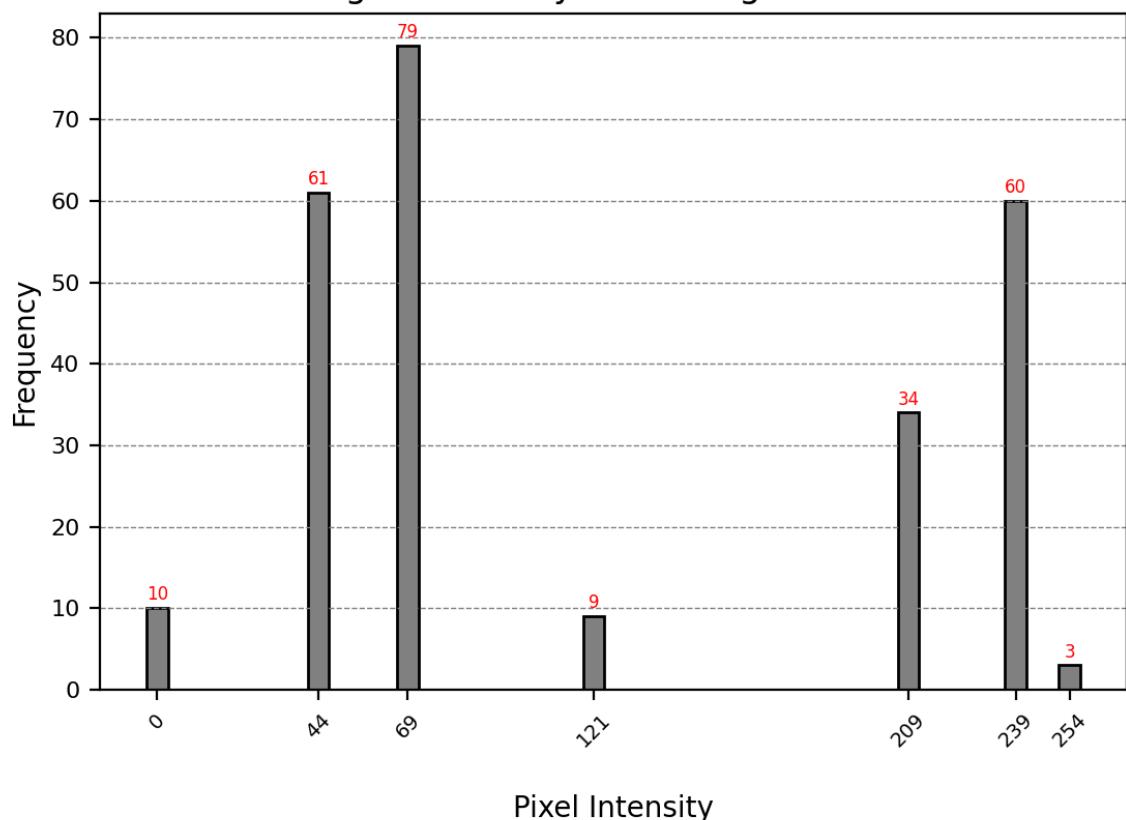
Original Image



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 הפעלנו את המשך על הגבולות של האזור החתור, נעצרנו בפיקולים של שאר

(התמונה ולא ריפדנו בערכים אחרים)

(הנחה נוספת – עיגול הערכים היה באופן הבא: 0.5 ומעלה – מעגנים למעלה, כל השאר – מעגנים למטה)

כ. גורן צי ארכיאולוג מגרינטוויל וקס הכתובת נסוי נזק ותלול

210	210	210
45	210	210
45	210	210

: 1x1 ρ_{air} 210 ρ_{air} 708

לכוד נסיך הרים ורשות

$$: \frac{1}{g} - \lambda \gamma \rho_0 |$$

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

(173) $\pi \ln \sqrt{f(x)}^{\frac{1}{2-n}}$, (n מילויים), (174) $\pi \sqrt{\ln \sqrt{f(x)}}^{\frac{1}{2-n}}$, (n מילויים)

בג' נארה אלה ה-210 וה-173 הנקודות הינהן מוקמו.

210	210	0
210	210	0
210	210	210

: 1x2 ρ_{PVNa} 210 ρ_{TiO_2} 708

የኢትዮጵያ ተስፋዥ ስርዱ አውራድ

$$\therefore \frac{1}{g} - 2 \geq m_1$$

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

(163) $\pi \text{Gnrf} \sqrt{2\pi} \frac{1}{2-N}$, (164) $\pi \text{Gnrf} \sqrt{2\pi} \frac{1}{N-1}$

בז' נאורה זה הופיע ב-210 הפקות גראן טריי 163 מילון נול מילון.

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

$$: \frac{1}{g} - \lambda \rightarrow \mathbb{R}$$

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

(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 πιπίνα 210 σοτσί 708

የኢትዮጵያ ከፍርድ ስነዎች ዘመን

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

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

(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 מושביה

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

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

(172) $\nabla \ln f(x) \frac{1}{2-N}$ (נ' מכוון ימינה, (172) $\nabla \ln f(x)$) $\neq -\nabla \ln f(x)$

בכל נאורה מה שפָּרְאֵס 210 הנקודות הראשית 171 ג'יג'ר ג'יג'ר נוֹעַ נוֹעַ ג'יג'ר.

45	210	210
45	210	210
210	210	210

פָּרֶסֶת כְּלַבֵּב 210 פָּרֶסֶת

רְכָב אֲלֵי גִּלְעָד הַגְּדוּלִים

: $\frac{1}{9} \cdot 8 = 8$

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

.(173) פָּרֶסֶת כְּלַבֵּב 1/2 - N גִּלְעָד, (174) פָּרֶסֶת כְּלַבֵּב 1/2 - N גִּלְעָד, פָּרֶסֶת כְּלַבֵּב 1/2 - N גִּלְעָד.

גִּלְעָד הַגְּדוּלִים 173 פָּרֶסֶת כְּלַבֵּב 1/2 - N גִּלְעָד.

210	210	0
210	210	210
210	210	0

פָּרֶסֶת כְּלַבֵּב 210 פָּרֶסֶת

רְכָב אֲלֵי גִּלְעָד הַגְּדוּלִים

: $\frac{1}{9} \cdot 8 = 8$

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

.(163) פָּרֶסֶת כְּלַבֵּב 1/2 - N גִּלְעָד, (164) פָּרֶסֶת כְּלַבֵּב 1/2 - N גִּלְעָד, פָּרֶסֶת כְּלַבֵּב 1/2 - N גִּלְעָד.

גִּלְעָד הַגְּדוּלִים 163 פָּרֶסֶת כְּלַבֵּב 1/2 - N גִּלְעָד.

יכן כליאת -

לבסוף, לאחר הפעלת מסנן היחסיקה הבסיסי בגודל 3X3 על כל הפיקסלים באזורי הכתום, נקבל:

Part C - Mean Filter

Original Cropped Image

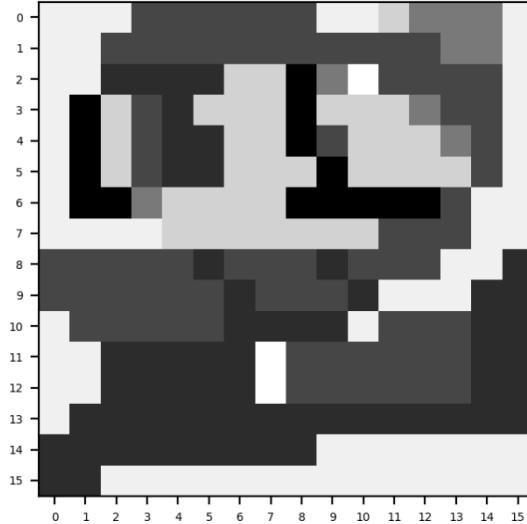
210	210	210	0	210	210	210
45	210	210	0	70	210	210
45	210	210	210	0	210	210
210	210	210	0	0	0	0
210	210	210	210	210	210	70
45	70	70	70	45	70	70
70	45	70	70	70	45	240

After Mean Filter Image

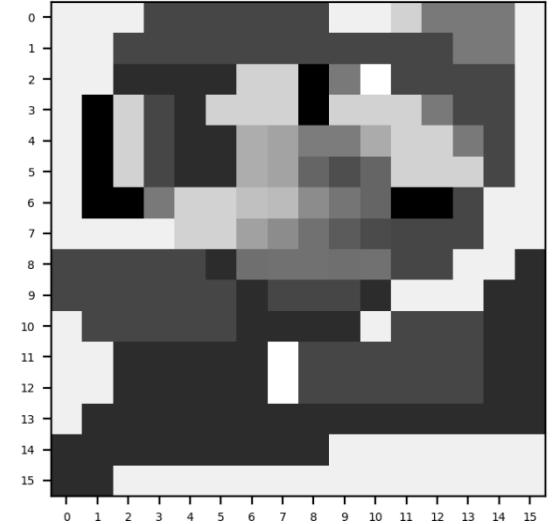
173	163	124	124	171
173	163	101	78	101
192	187	140	117	101
161	140	114	91	75
111	114	114	111	114

Part C - Mean Filter

Original Image



After Mean Filter Image

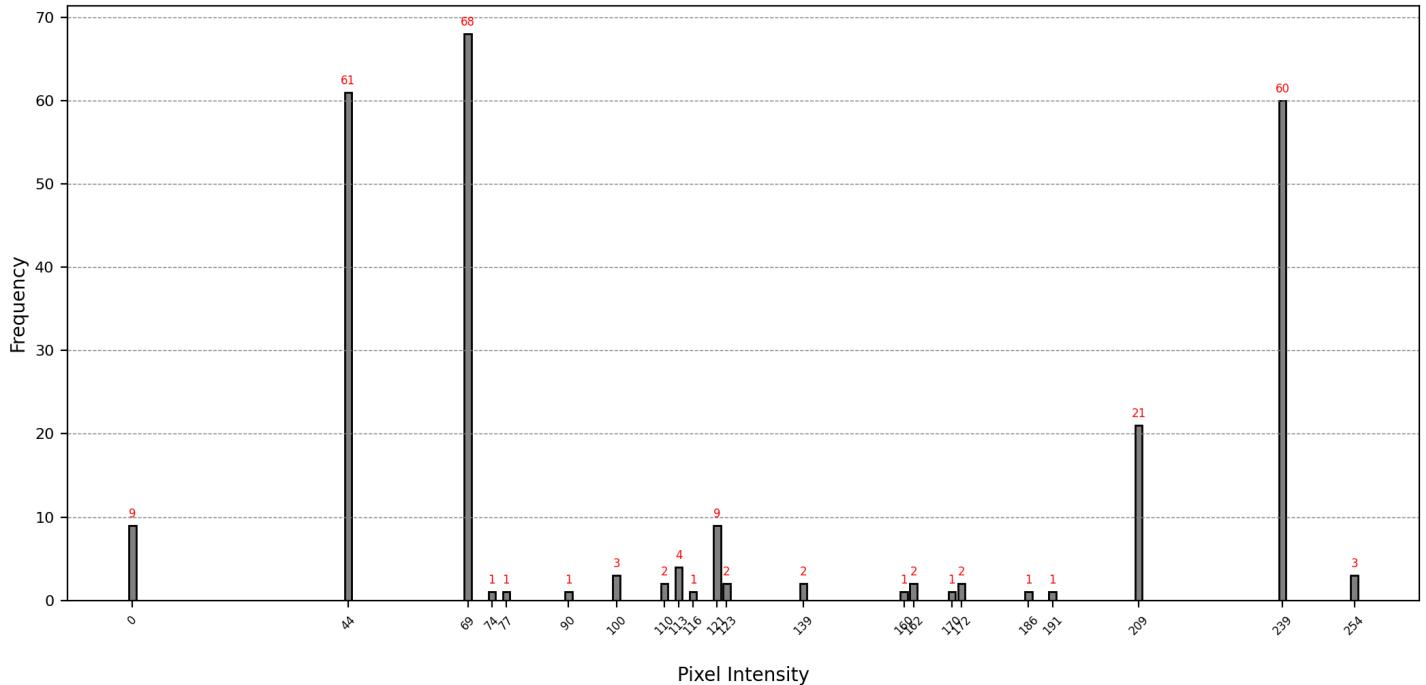


Part C - Mean Filter

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 חציון (Median Filter)

חזקנות:

mphicit רעש בעילות: מסנן החציון ייעיל במיוחד בהסרת רעש מסווג "מלח ופלפל", ככלומר רעש שבו יש נקודות שחומות ולבנות באופן אקראי.

שומר על קצוות (Edges): בנגוד למסנן החלקה בסיסי, מסנן החציון שומר טוב יותר על הקצוות בתמונה, מכיוון שהוא מחליף את הפיקסל בערך החצויוני ולא במוצעו. זה שימושי בתמונות שבהן חשוב לשומר על פרטיהם חדים.

חולשות:

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

פחות עיל ברעש מסווג גאוסיאני: מסנן החציון פחות מתאים להפחיתת רעש גאוסיאני (התפלגות נורמלית), שבו ערכי הפיקסלים קרובים למוצע אך לא קיצוניים. במקרה זה, התוצאה עשויה להיראות פחותות חלקה.

- מסנן החלקה בסיסי (Mean Filter)

חזקנות:

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

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

חולשות:

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

פחות עיל ברעש "מלח ופלפל": מכיוון שמסנן החלקה בסיסי מחשב ממוצע לכל הפיקסלים בסביבה, הוא עלול להפיץ את הרעש "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 חציון: מתאים יותר מכיוון שהוא מסיר ערכים קיצוניים מבלוי להשפיע הרבה על הפיקסלים שמסביב.

מסנן החלקה בסיסי: יטשטש את הרעש עם הפיקסלים 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 חציון: 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 מקובוקים)
מסנן חציון : שומר על הקצוות ומסיר רعش תוך שמירה על פרטיים חדים.
מסנן החלקה בסיסי : עלול לטשטש את הקווים והפרטים הקטנים, מה שייגרם לאיבוד פרטיים בתמונה.

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

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

e. גנטכל 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 המקורית, בהשוואה למסנן חציון. הנה ההסבר לכך ואיך ניתן היה לגשת לשחזור התמונה.

מסנן החלקה בסיסי (Basic Smoothing Filter)

מסנן החלקה בסיסי מבוסס על מיצוע ערכים בשכונת פיקסלים, כך שכל פיקסל בתמונה החדש הוא ממוצע של הערך המקורי ושל הפיקסלים השכנים. תהליך זה יוצר טשטוש, אך שומר מידע מהפיקסלים השכנים באופן שניינט לחזות אותו.

כדי לשחזור את התמונה המקורי, ניתן להשתמש בתהליך שנקרא דה-קונבולוציה (deconvolution), אשר מנסה "להפוך" את פעולה המיצוע ולשזר את הערכים המקוריים על סמך התמונה המתוושתת וגרעין המיצוע הידוע. תהליך זה דורש:

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

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

מסנן חציון (Median Filter)

מסנן חציון מחליף כל פיקסל בערך החציני של שכונתו, מה שיוכל לשנות את הערכים המקוריים משמעותית, במיוחד באזוריים עם ניגודיות גבוהה. פעולה החציון היא לא לינארית ולא הפיכה, מכיוון שהיא מחליף את הערכים המקוריים במקומות מצטאים, ולכן קשה מאוד לשזר את התמונה המקורי.

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.
- לחופין, אם התמונה מכילה דפוסים ידועים, ניתן לננות לשימוש במודלים של מידת מכונה שינסו לנבא את המידע החסר על בסיס סט נתוניים דומים.

אתגרים: מכיוון שמסנן חציון מחליף את הערכים המקוריים בערכים חציניים, אין דרך ישירה לשזר את המידע המקורי, מה שהופך את השזרה למורכב ולעתית אף לבליי אפשרי.

לסיום, **מסנן החלקה בסיסי** קל יותר לביטול באמצעות טכניקות דה-קונבולוציה, מכיוון שהשפעותיו צפויות ונינעות לשזר באופן יחסית. לעומת זאת, **מסנן חציון** נחسب 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-Patch בלבד:

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

1. נאותחל Confidence Matrix כך שכל פיקסל חסר הוא 0 ושאר הפיקסלים 1 –
2. נמצאת כל הפיקסלים החסרים בתמונה – missing_pixels של 3×3 .
3. עבור כל פיקסל חסר:
 - a. נאותחל מערך של פיקסלי המסגרת, אם לא קיימים סיים לולא – 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 מקסימלי, ניקח את הראשון מביניהם) – selected_pixel
 - d. נשלוף את ה-Patch (בגודל 3×3 (window_size)) סביבו אותו פיקסל נבחר – selected_patch
 - e. נמצאת החלון בגודל 3×3 שעריך RMSE בין ה-Patch והוא הנמוך 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)

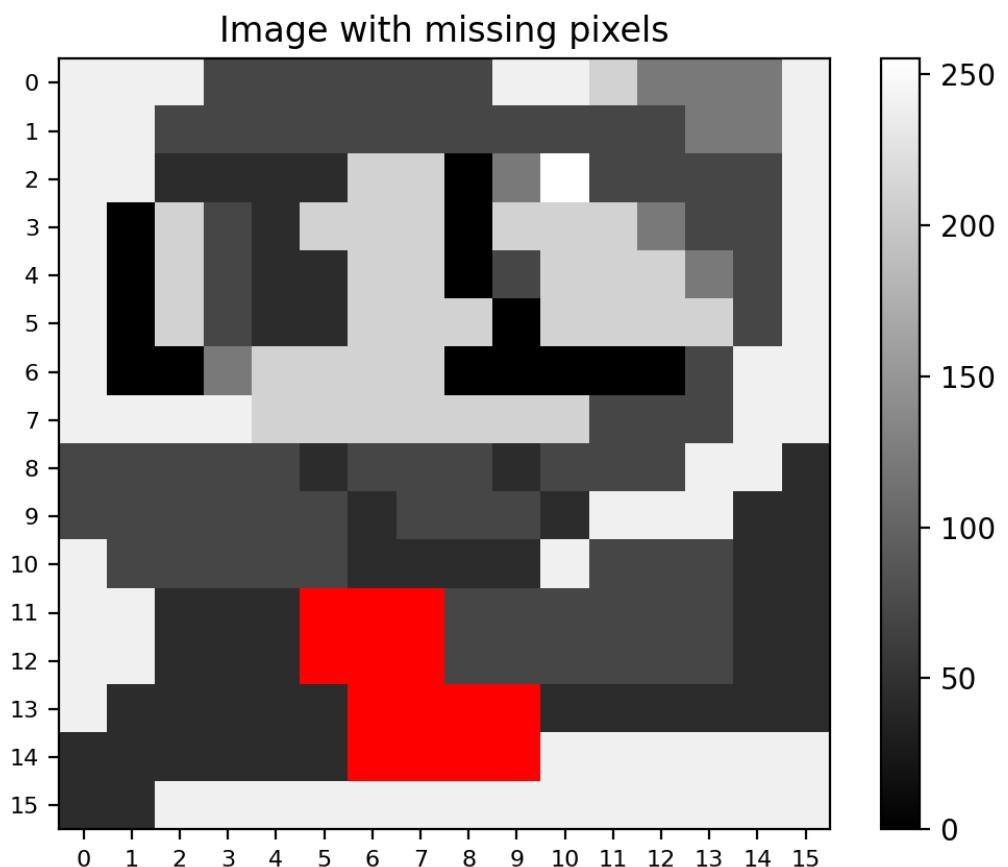
    # Display results
    display_image(image, "Image with missing pixels")

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

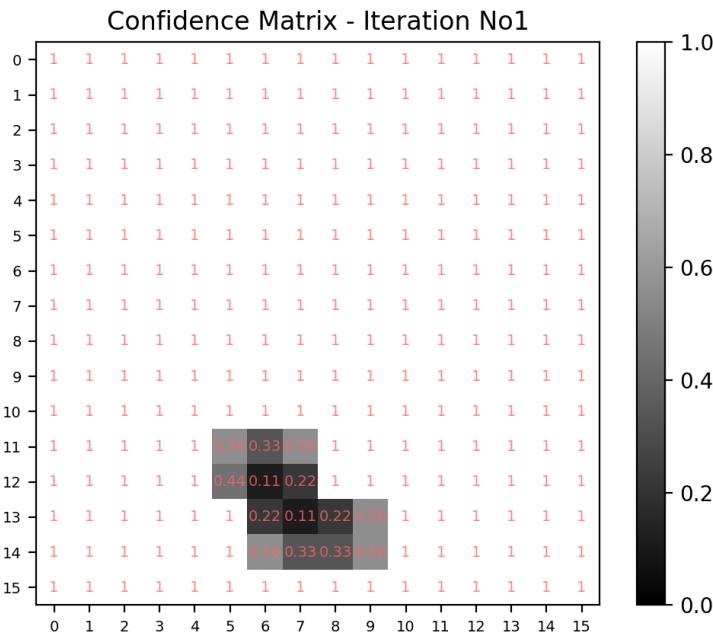
    display_images_side_by_side("Before and After Patch-Based Inpainting", image, 'Before', filled_image, 'After', with_ticks=True)
    display_image(image, "Initial Image", with_intensity=True)
    display_image(filled_image, "Final Image", with_intensity=True)

```

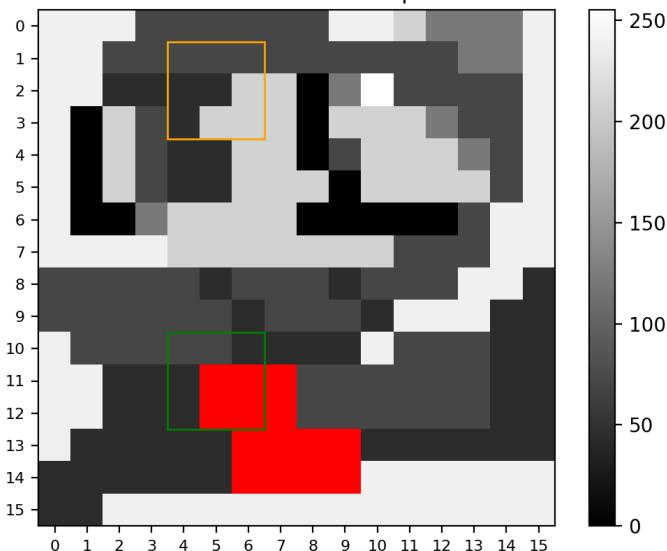
התמונה הנתונה:



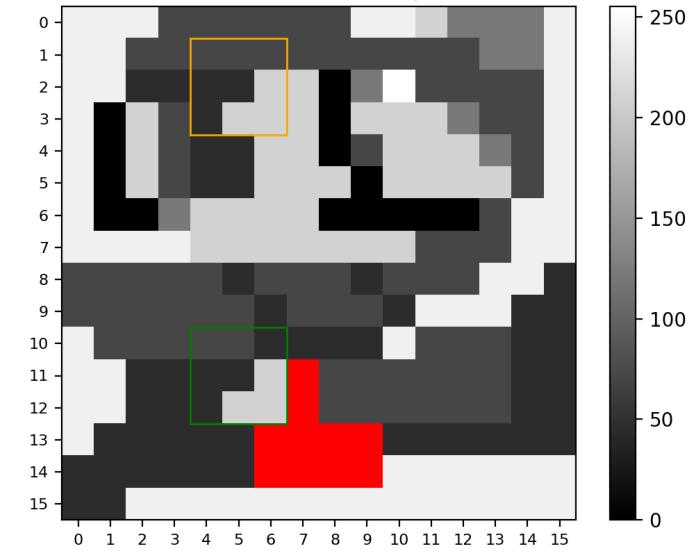
איטרציה 1 :



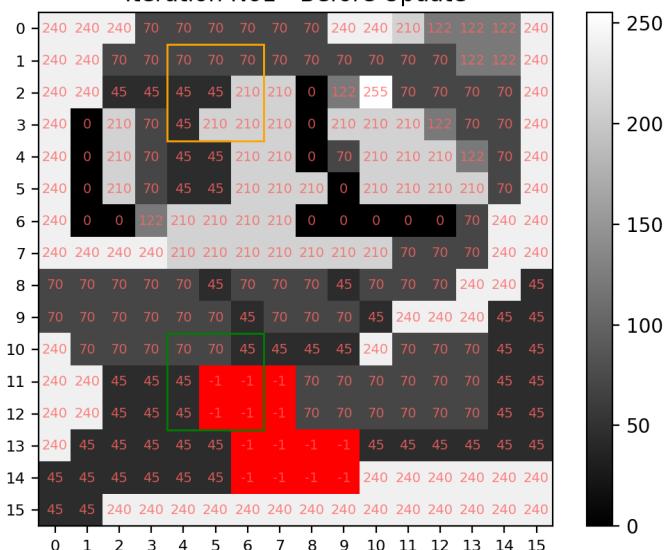
Iteration No1 - Before Update



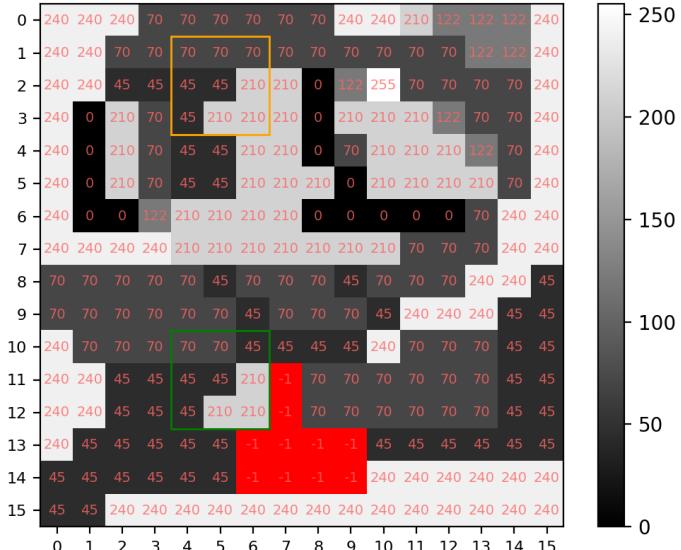
Iteration No1 - After Update



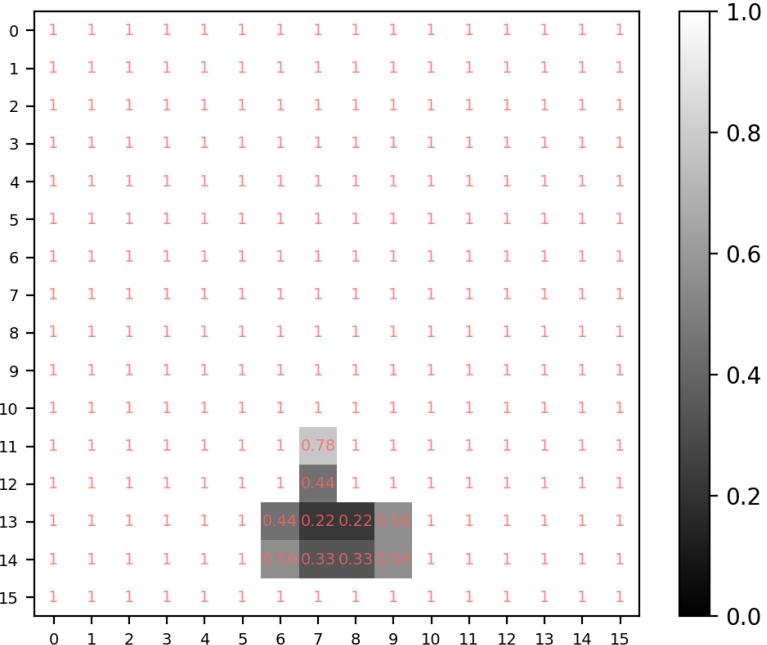
Iteration No1 - Before Update



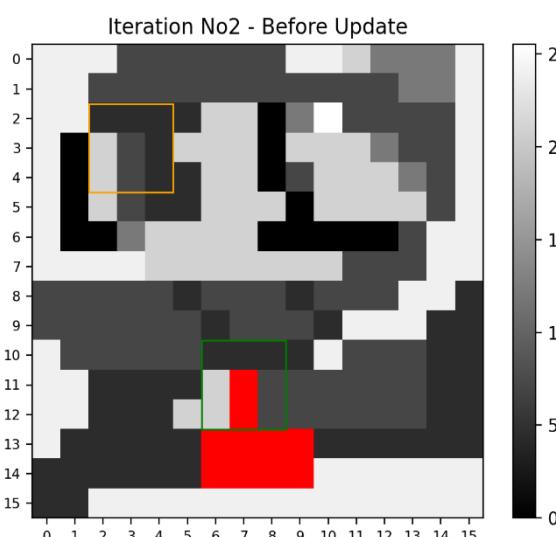
Iteration No1 - After Update



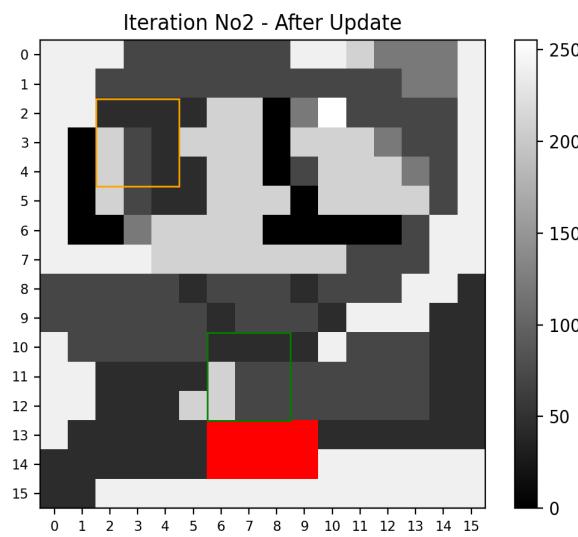
Confidence Matrix - Iteration No2



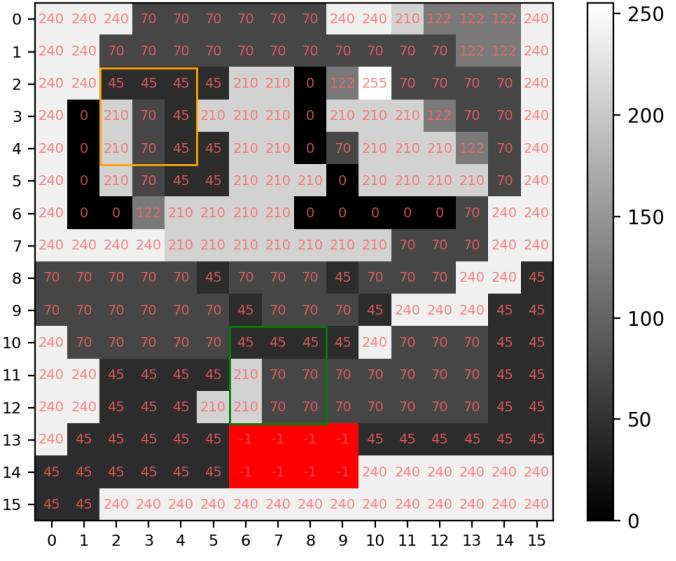
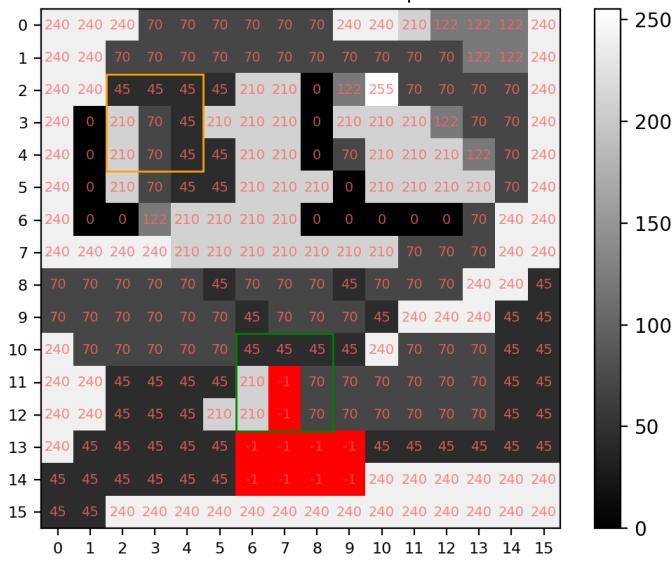
איטרציה 2:



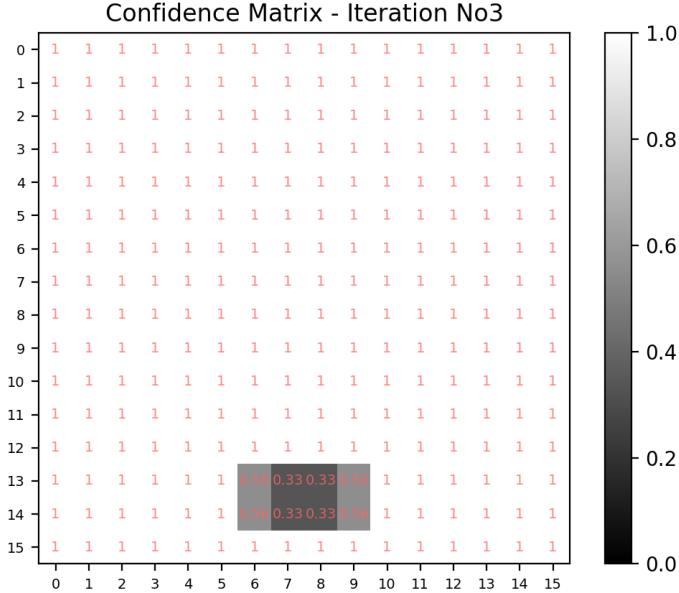
Iteration No2 - Before Update



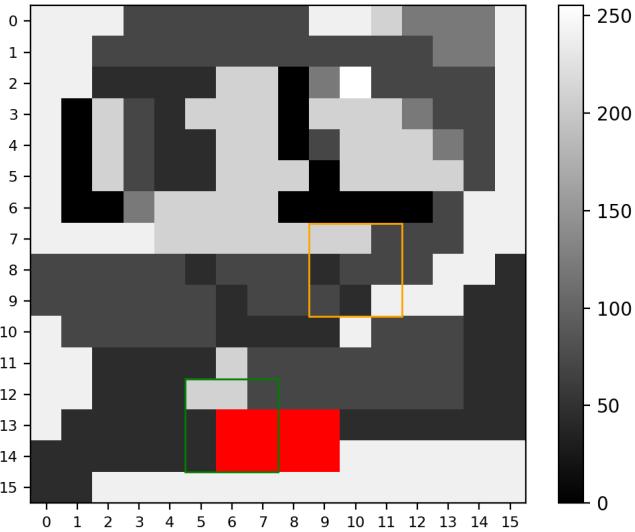
Iteration No2 - After Update



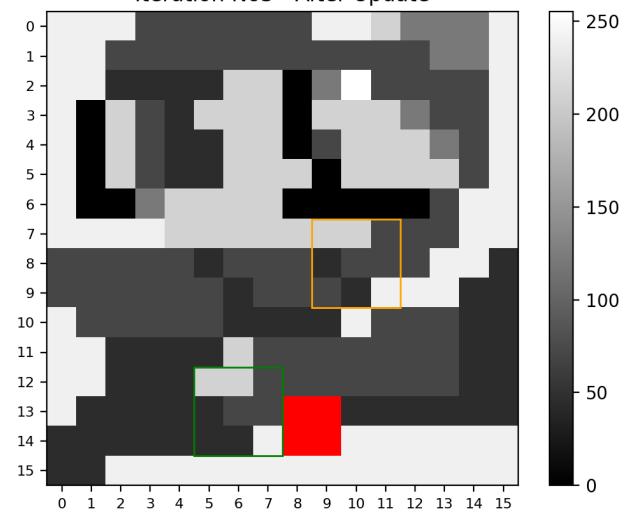
איטרציה 3:



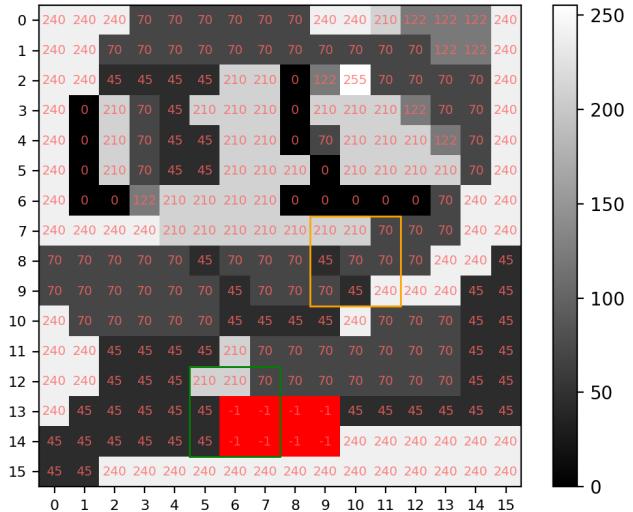
Iteration No3 - Before Update



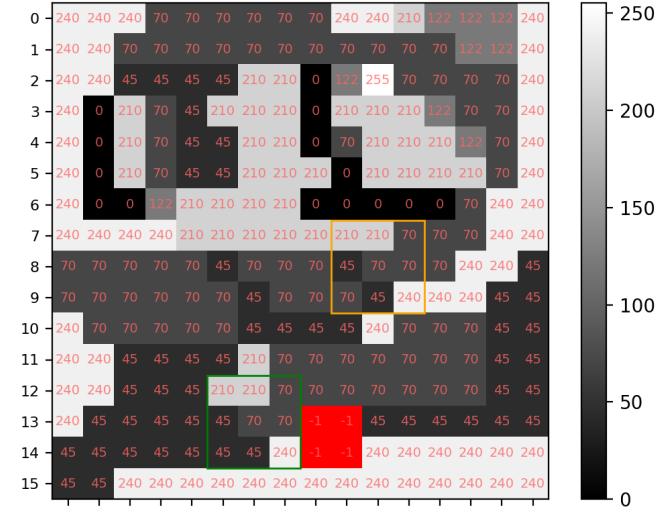
Iteration No3 - After Update



Iteration No3 - Before Update

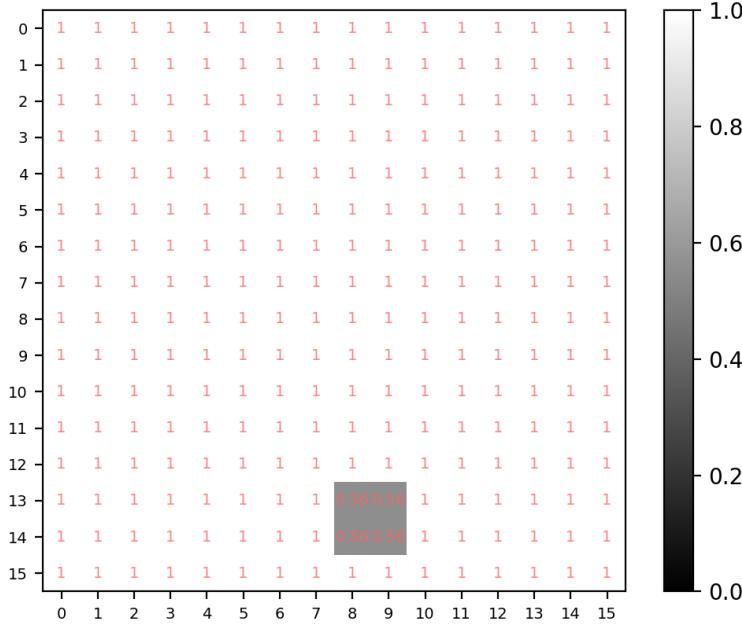


Iteration No3 - After Update

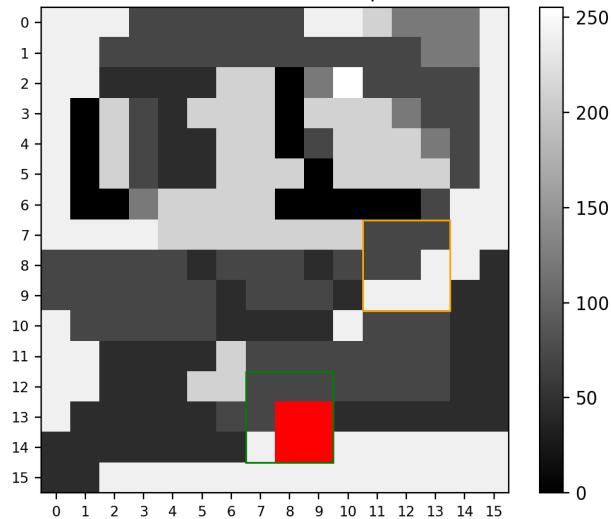


איטרציה 4:

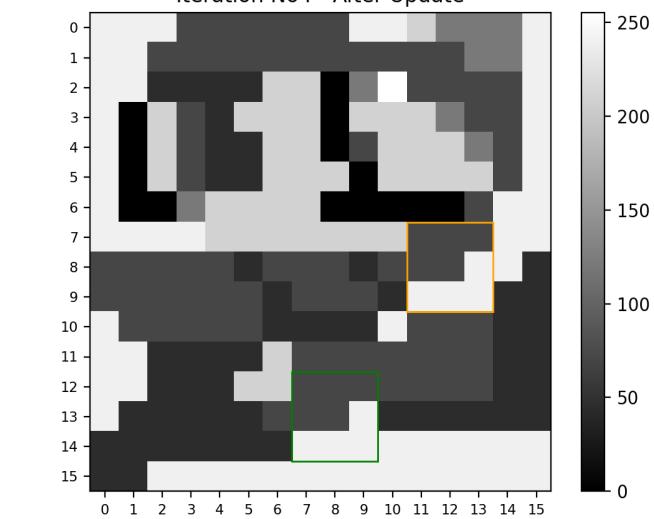
Confidence Matrix - Iteration No4



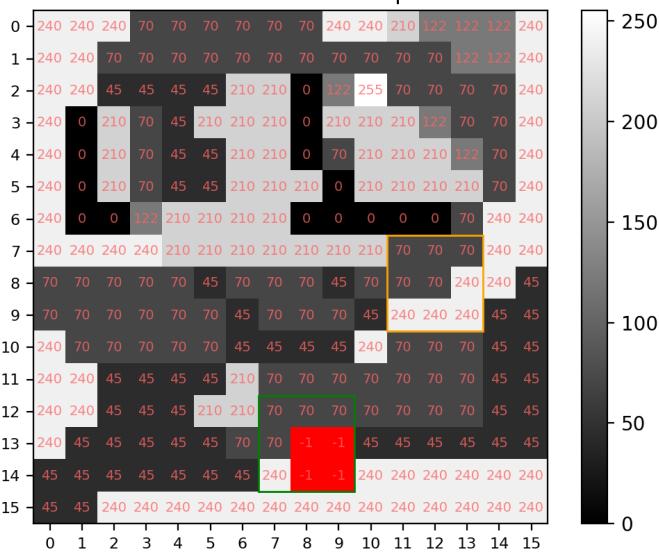
Iteration No4 - Before Update



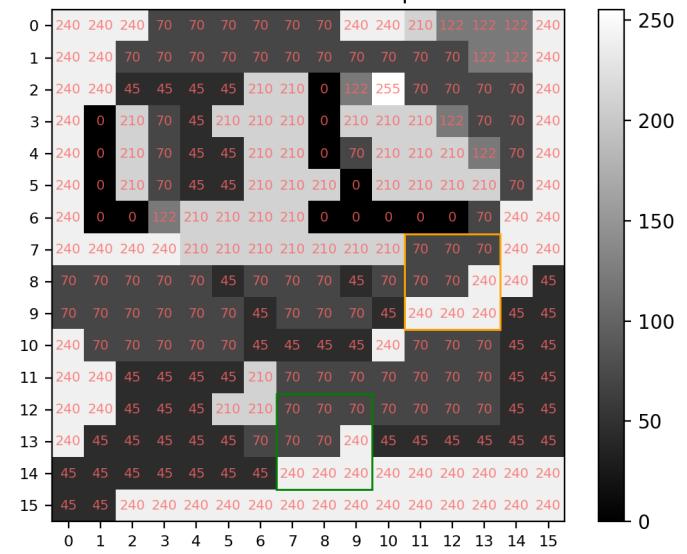
Iteration No4 - After Update



Iteration No4 - Before Update

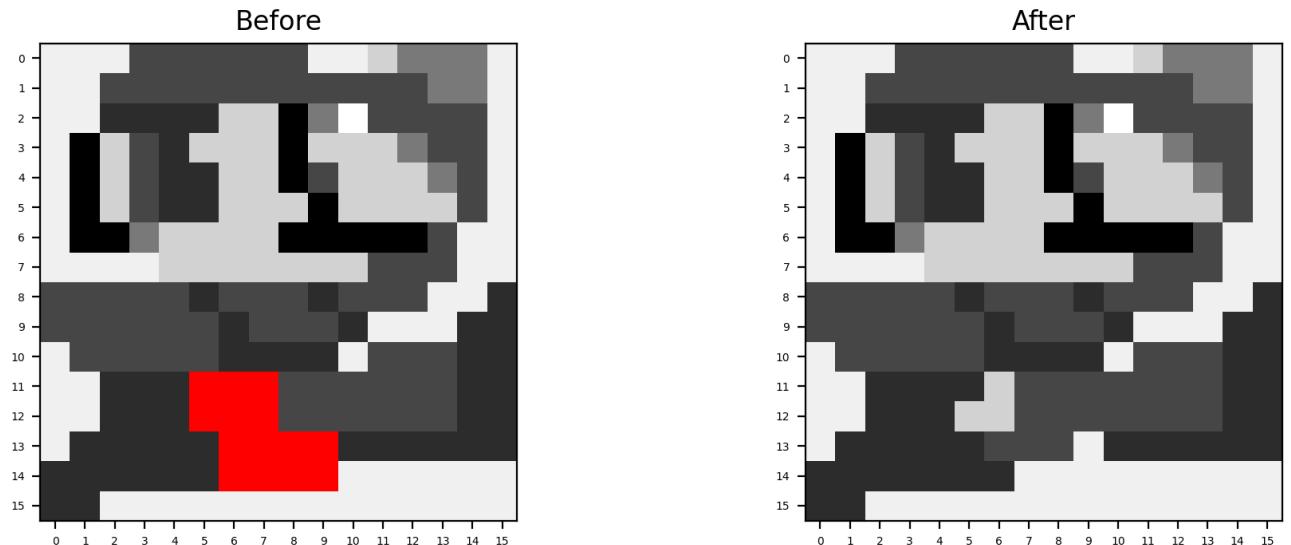


Iteration No4 - After Update

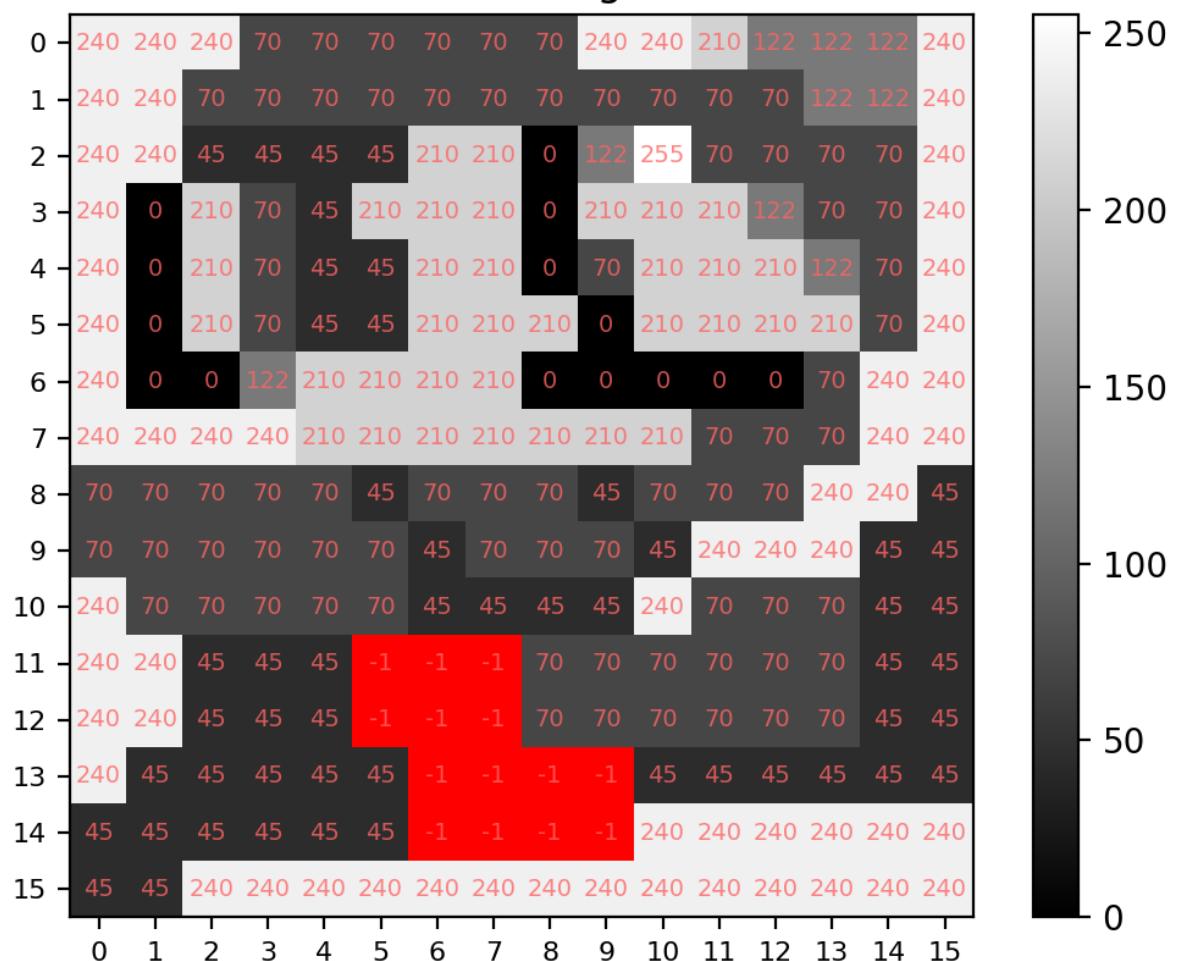


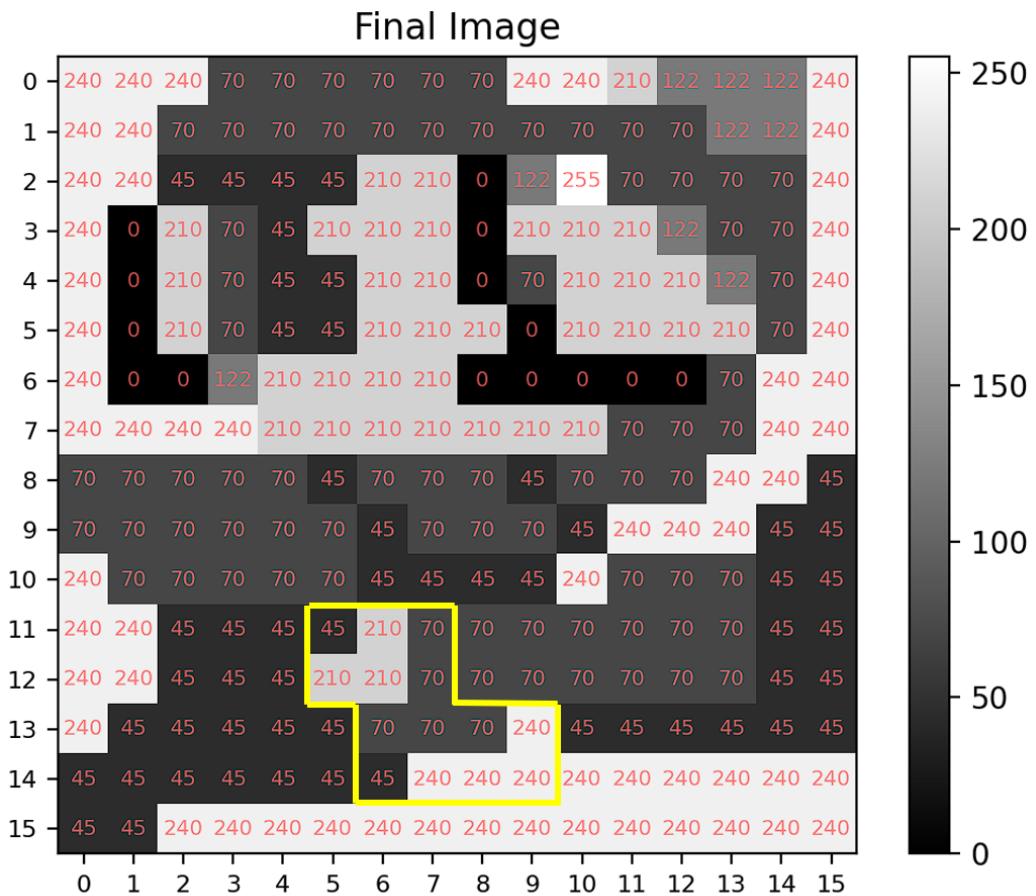
הפלט הסופי:

Before and After Patch-Based Inpainting



Initial Image





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



חלק שני:

התמונה שלי:

“DIP_803.png”

a. הצגה וניתוח של התמונה המקורי

הfonkzia נממש את הפונקציה (analyze_image(input_image_path, output_folder_path) מקבלת כקלט את התמונה שלי, ומיקום של תיקיה אליה ישמרו הפלטים 1-8 ותשמרו אותם בתיקיות הפלט.

התמונה שלי: “DIP_803.png”
תיקיות הפלט: “outputs-a”.

2a -----

```
def analyze_image(input_image_path, output_folder_path):
    """
    Analyze an image and save outputs including grayscale and color histograms,
    gradient maps, and gradient intensity.

    Parameters:
    input_image_path (str): Path to the input image.
    output_folder_path (str): Path to the folder where outputs will be saved.
    """

    # Load the image
    image = cv2.imread(input_image_path)
    if image is None:
        print(f"Error: Could not load image from {input_image_path}. Check the file path.")
        return

    # Determine if the image is grayscale or color
    if len(image.shape) == 3 and image.shape[2] == 3:
        # Check if the color channels are all identical (which means the image is grayscale)
        is_color = not np.array_equal(image[:, :, 0], image[:, :, 1]) or not np.array_equal(image[:, :, 1], image[:, :, 2])
    else:
        is_color = False # The image is grayscale if it has only one channel

    # Ensure the output directory exists
    if not os.path.exists(output_folder_path):
        os.makedirs(output_folder_path)

    # 1. Convert to grayscale and save
    gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) if is_color else image
    cv2.imwrite(os.path.join(output_folder_path, 'grayimg.png'), gray_image)
```

```

# 2. Plot and save grayscale histogram
plt.figure()
plt.hist(gray_image.ravel(), 256, [0, 256], color='black')
plt.title('Grayscale Histogram')
plt.xlabel('Pixel Intensity')
plt.ylabel('Frequency')
plt.savefig(os.path.join(output_folder_path, 'gray_histogram.png'))
plt.close()

# 3-5. Plot and save color histograms if the image is color
if is_color:
    color = ('b', 'g', 'r')
    for i, col in enumerate(color):
        plt.figure()
        hist = cv2.calcHist([image], [i], None, [256], [0, 256])
        plt.plot(hist, color=col)
        plt.title(f'{col.upper()} Histogram')
        plt.xlabel('Pixel Intensity')
        plt.ylabel('Pixel Frequency')
        plt.savefig(os.path.join(output_folder_path, f'{col}_histogram.png'))
        plt.close()

# 6. Compute and save x-gradient (Sobel filter)
gradient_x = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=3)
cv2.imwrite(os.path.join(output_folder_path, 'gradient_x.png'), np.abs(gradient_x).astype(np.uint8))

# 7. Compute and save y-gradient (Sobel filter)
gradient_y = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=3)
cv2.imwrite(os.path.join(output_folder_path, 'gradient_y.png'), np.abs(gradient_y).astype(np.uint8))

# 8. Compute and save gradient intensity
gradient_magnitude = cv2.magnitude(gradient_x, gradient_y)
cv2.imwrite(os.path.join(output_folder_path, 'gradient_intensity.png'), np.abs(gradient_magnitude).astype(np.uint8))

print(f"All outputs saved in {output_folder_path}")

```

מחוץ לפונקציה - הגדרת תמונה הקלט, תיקיית הפלט וזמן האופרציה:

```
# Get the directory of the current script
script_dir = os.path.dirname(os.path.abspath(__file__))

# Construct the full path to the image
input_image_path = os.path.join(script_dir, 'DIP_803.png')

# Create a generic "outputs-a" folder within the same directory as the script
output_folder_path = os.path.join(script_dir, "outputs-a")

# Check if the "outputs-a" folder exists; if not, create it
if not os.path.isdir(output_folder_path):
    os.makedirs(output_folder_path)

# Call the function
analyze_image(input_image_path, output_folder_path)
```

b.

-grayimg.png .1

מה רואים:

קיבלנו את התמונה שלנו בגווני אפור. תמונה זו היא גרסה מפשיטתה שבה כל הצבעים הוסרו, ונשארו רק גוונים של אפור. מייצגת עצמת האור בתמונה, כאשר לבן מייצג עצמה גבואה ושחור מייצג עצמה נמוכה.

כיצד מתקשר לתמונה המקורית:

התמונה המקורית הייתה צבעונית, והפיכתה לגווני אפור מפשיטת את המידע על ידי הסרת הצבע, ומאפשרת לנתח את הבתיות בלבד, מה שיכל לעזור בניתוח אינטנסיביות וכו'.



.2 - gray_histogram.png

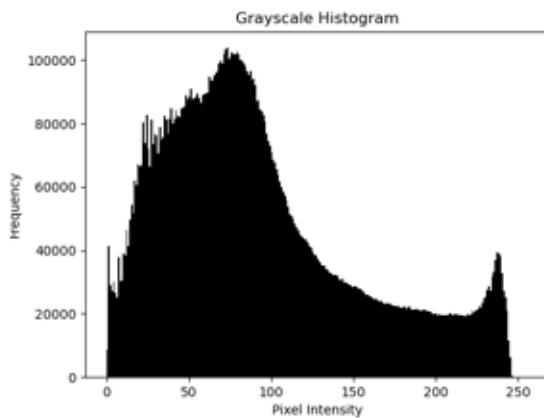
מה רואים:

ההיסטוגרמזה מציגה את התפלגות הערכים בגווני האפור בתמונה. על ציר ה-X יש את ערכי הבاهירות מ-0 עד 255, ועל ציר ה-Y את מספר הפיקסלים לכל ערך בהירות.

ניתן לראות שההתפלגות מכילה הרבה ערכים בתחום הנמר (90-0), מה שמעיד על הרבה אזוריים כהים בתמונה. ישנה פסגה נוספת בערך גובה מסובב ל-255 (לבן), מה שמעיד על אזוריים לבנים/בHIRIM/מווארים מאוד בתמונה.

כיצד מתקשר לתמונה המקורית:

ההיסטוגרמזה משקפת את כמות הפיקסלים בכל גוון אפור, מה שעזר להבין את רמת הבاهירות הכללת בתמונה ואת הניגודיות שלה. פסגות בהיסטוגרמזה מצביעות על רמות עצמה המופיעות בתדרות גובהה בתמונה. הפסגה המשמעותית ביותר היא בחלק הכהה של ההיסטוגרמזה, מה שמעיד על תמונה יחסית כהה (הגיוני כי יש המון כל, ועלי העצים כהים).



.3 - red_histogram.png

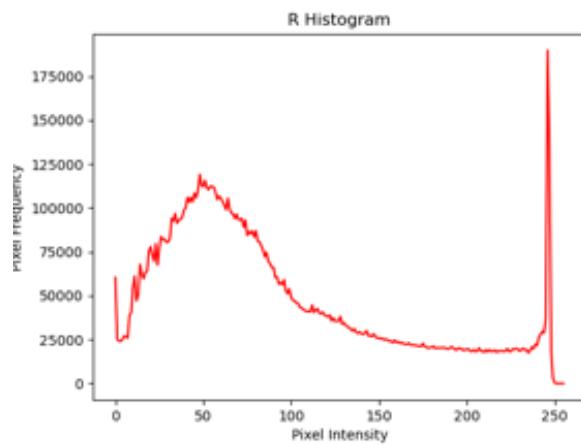
מה רואים:

ההיסטוגרמזה מראה את התפלגות ערכי הפיקסלים בעורץ הצבע האדום של התמונה. ניתן לראות פסגה בינונית בערכים הנמוכים (15-100), מה שמעיד על כך שהצבע האדום קיים מעט יחסית במרבית התמונה ואיןו דומיננטי.

ישנה קפיצה משמעותית סביבה הערך המקסימלי (240-250), מה שמעיד על לא מעט פיקסלים מווארים מאוד באדום (ניתן לראות שישנה שקייה ליד האיש מאחור, אוזדים עם כנפיים אדמדמות והמון עליהם קטנים בצבע אדום – כתום מפוזרים על הדשא).

כיצד מתקשר לתמונה המקורית:

היסטוגרמזה זו ממחישה כיצד עצמת העורץ האדום מפוזרת על פני התמונה. פסגות גבוהות מצביעות על כך שעוצמת אדום מסוימת היא דומיננטית – עוזר להבין כמה חזק ובולט הצבע האדום בתמונה. הרבה פיקסלים בעלי מעט יחסית צבע אדום ולא מעט פיקסלים בצבע אדום מואר יותר.

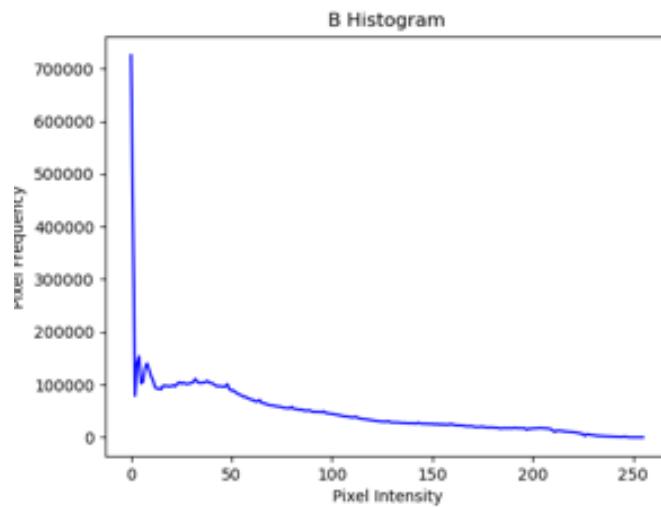


4. - blue_histogram.png מה רואים:

ההיסטוגרמזה זו מראה את התפלגות ערכי הצבע הכחול בתמונה. ניתן לראות שרוב מוחץ מהפיקסלים בתמונה נמצאים בצד שמאל של ההיסטוגרמזה הכחולה. ניתן להסיק שברוב מוחץ מהפיקסלים בתמונה אין כחול או יש מעט מאוד כחול.

כיצד מתקשר לתמונה המקורית:

ההיסטוגרמזה זו משקפת את אופן פיזור העוצמות הכחולות בתמונה, ונונתנת תובנות לגבי השפעתו של הרכיב הכחול. ניתן להבין שהשפעתו של הרכיב הכחול די' זניחה בתמונה. הגינס של האדם האמצעי ושקיות הצלב הכחולות לדוגמא יהיו עם פיקסלים מוארים יותר בכחול וימוקמו בצד ימני של ההיסטוגרמזה הכחולה – אך הם מעטים ביחס לשאר התמונה.



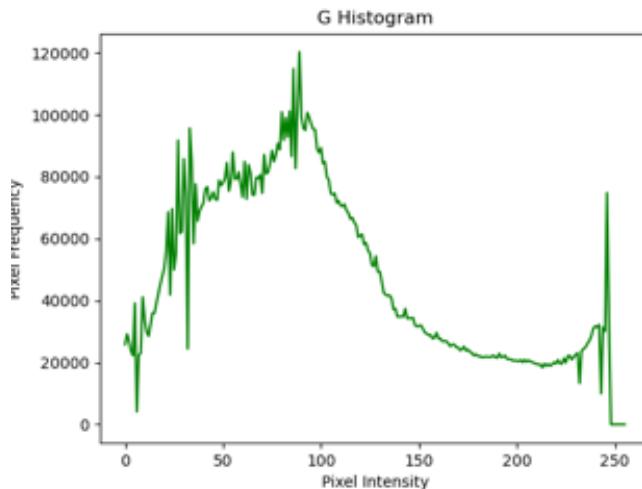
5. - green_histogram.png

מה רואים:

ההיסטוגרמה זו מציגה את התפלגות ערכי הצבע הירוק בתמונה. ניתן לראות פסגות רבות בחלקים שבין הערכים 150-20, ופסגה נוספת חדה בצד ימין של ההיסטוגרמה (230-255).

כיצד מתקשר לתמונה המקורית:

דומה להיסטוגרמה האדומה והכחולה, אבל לעירען הירוק. זה עוזר בהבנת התרומה של הצבע הירוק לתמונה. ניתן לראות פיזור די גדול של ההיסטוגרמה, מה שמעיד על המון פיקסלים עם המון גווני ירוק. ניתן להבין מכך שהתמונה מוצפת בהמוני פיקסלים ירוקים יחסית (המוני דשא ועלים ירוקים בתמונה). הדשא ברובו מוצל, דבר שיכול להסביר מדוע ההיסטוגרמה הירוקה מוקבצת בעיקר בצדיה השמאלי.



6. - gradient_x.png

מה רואים:

קיבלנו מפה המציגת את הגרדיאנטים (שינויים בעוצמה) בכיוון x (אופקי). התמונה מראה את השינויים בעוצמת הפיקסלים בכיוון האופקי, כך שהאזורים עם הבדל חד בערכים יראו כקוויים בהירים יותר.

כיצד מתקשר לתמונה המקורית:

תמונה זו מדגישה קצוות אופקיים בתמונה המקורית. אזורים בהירים תואמים למקומות שבהם יש שינוי אופקי משמעותי. כאשר מזהים קצוות, ניתן להבין טוב יותר את המבנה הכללי של האובייקטים בתמונה, מה שעוזר בעיבוד תמונה ובזיהוי צורות.



.7 - gradient_y.png

מה רואים:

קיבلونו מפה המציגה את הגרדיאנטים בכיוון y (אנכי). התמונה מדגישה את השינויים בעוצמת הפיקסלים בכיוון זה, כך שאזורי עם שינוי חד בערכיהם נראים כקווים בהירים יותר.

כיצד מתקשר לתמונה המקורית:

תמונה זו מדגישה קצוות אנכיים בתמונה המקורית. אזורים בהירים מציננים היכן יש שינוי אנכי ממשמעותי בעוצמה.



.8 - gradient_intensity.png

מה רואים:

קיבلونו מפה המציגה את עוצמת הגרדיאנטים הכלולת על פני התמונה. זהו שילוב של הגרדיאנטים בכיוונים האופקי (x) והאנכי (y). התמונה מציגה את העוצמות הכלולות של השינויים בבהירות, ומדגישה את הקצוות והמעברים החדשניים בתמונה.

כיצד מתקשר לתמונה המקורית:

תמונה זו מדגישה את הקצוות והגבלוות בתוך התמונה המקורית, שבהם יש שינויים חזקים בעוצמה. משולבים הגרדיאנטים של ציר $-x$ וה- $-y$, ומתאפשרה תצוגה מקיפה של כל השינויים המשמעותיים בתמונה.



ב. תיקון ושיפור התמונה המקורי

- a. נමיש את הפונקציה `first_op(input_image_path, output_folder_path)` (grayimg.png) הנקוצה מקבלת כקלט את התמונה המקורי בגווני אפור מהסעיף הקודם, ומקום של תיוקה אליה ישמר הפלט של הפונקציה.
תמונה הקלט: "DIP_803.png"
תיקייה הפלט: "outputs-b"

2b -----

```
def first_op(input_image_path, output_folder_path):
    """
    Perform a geometric transformation (rotation) on the grayscale image using OpenCV.

    Parameters:
    input_image_path (str): Path to the input grayscale image.
    output_folder_path (str): Path to the folder where the output image will be saved.
    """

    # Load the image and check if it's color or grayscale
    img = cv2.imread(input_image_path)
    if img is None:
        print(f"Error: Could not load image from {input_image_path}")
        return

    # Define a fixed angle of rotation. The default angle chosen is 15 degrees counter-clockwise.
    rotation_angle = -15 # Rotating by 15 degrees counter-clockwise

    # Get image dimensions
    (h, w) = img.shape[:2]

    # Define the rotation center and get the rotation matrix
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, rotation_angle, 1.0)

    # Perform the rotation with bilinear interpolation
    img_rotated = cv2.warpAffine(img, M, (w, h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REPLICATE)

    # Save the rotated image
    rotated_image_path = f"{output_folder_path}/{img_firstop.png}"
    cv2.imwrite(rotated_image_path, img_rotated)

    return rotated_image_path
```

מחוץ לפונקציה - הגדרת תמונה הקלט, תיקיית הפלט וזמן אופרציה:

```
# Create a generic "outputs-b" folder within the same directory as the script
output_folder_path = os.path.join(script_dir, "outputs-b")

# Check if the "outputs-b" folder exists; if not, create it
if not os.path.isdir(output_folder_path):
    os.makedirs(output_folder_path)

# Construct the path to "img_grayimg.png" inside the "outputs-a" folder
input_image_path = os.path.join(script_dir, "outputs-a", "grayimg.png")
# # First operation: Rotation
first_op(input_image_path, output_folder_path)
```

a. הפעולה הראשונה שבחרת夷 לשוטה היא **טרנספורמציה גיאומטרית (סיבוב)**. באמצעות טרנספורמציה גיאומטרית ביצעת סיבוב של התמונה ב 15 מעלות נגד ציון השעון. כפי שWOODAI ניתן לראות, התמונה כמעט מסובבת, וסיבוב שלה חזרה כדי שתהיה ישרה יעוז להבחן טוב יותר בפרטים בתמונה מלבד את הראש. סיבוב התמונה מקל על הערכת אופרציות עוקבות שעשוות להיות תלויות ביישור נכון. תחילה עם סיבוב מבטיחה שהתמונה מכוננת נכון לשליבי עיבוד נספחים, שכן תמונה לא מיושרת עלולה להשפיע על ניתוחים או פעולות התלויות במיקום האובייקטים בתמונה.

ישום רוטציה:

הפעלו סיבוב של 15 מעלות נגד ציון השעון כדי ליישר את התמונה, תוך שימוש ב cv2.getRotationMatrix2D(center, rotation_angle, scale) על מנת ליצור מטריצה סיבוב שבמראכה נקודת המרכז של התמונה. מטריצה זו מתארת כיצד יש להזיז כל פיקסל בהתאם על מרכז התמונה (center), זווית הסיבוב (rotation_angle) וכן המידה שאומר מה יהיה גודל התמונה המסובבת בהשוואה לתמונה המקורי (scale).

הסיבוב הוחל עם cv2.warpAffine() תוך שימוש באינטרפולציה בילינארית להחלקה, ו-borderMode=cv2.BORDER_REPLICATE כדי למלא קצוות בערכי הפיקסלים הקרובים ביותר, תוך הימנענות מגבולות שחורים. ישור זה הופך את התמונה למתחימה יותר לשליבי העיבוד הבאים על ידי ביטול הטיה ראשונית וממן תוצאה ממורכצת.

הבדל בין התמונה המקורית לתמונה שהתקבלה:

כיוון: התמונה המסובבת מיושרת כתעב בצורה שונה בהשוואה למקורו. על ידי סיבוב התמונה ב-15 מעלות נגד ציון השעון, כל הטיה ראשונית מתוקנת, מה שגורם לתמונה להיראות מפולשת יותר. העצים עומדים ישר האנשים מאוכנים לkrkע, וכך גם האוזניים.

מראה: נעשה באמצעות אינטרפולציה בילינארית. התמונה המסובבת מלאה את כל האזוריים סביב הקצוות שנוצרו על ידי הסיבוב בפיקסלים משוכפלים כדי למנוע גבולות שחורים. זה שומר על עקביות ויזואלית.

פרמטרים לסיבוב:

זווית סיבוב: זווית הסיבוב מוגדרת ל-15 מעלות (נגד כיוון השעון) כדי לתקן כל הטיה ראשונית בתמונה, וליחס אותה לפני עיבוד נוסף.

מרכז התמונה: מרכז התמונה, מחושב (רוחב // 2, גובה // 2), משמש כцентр לסיבוב. זה שומר על איזון התמונה סביר הנקודה המרכזית שלה, ומבטיח שאך חלק מהתמונה לא יוזז רוחק מדי מהמסגרת.

קדם קנה מידה: נשתמש במקדם קנה מידה של 1.0, כלומר גודל התמונה נשאר זהה לאחר הסיבוב.

הסברים מעמיקים למושגים שבוצעו בהם שימוש באופרציה זו:

```
# Define a fixed angle of rotation. The default angle chosen is 15 degrees counter-clockwise.
rotation_angle = -15 # Rotating by 15 degrees counter-clockwise

# Get image dimensions
(h, w) = img.shape[:2]

# Define the rotation center and get the rotation matrix
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, rotation_angle, 1.0)

# Perform the rotation with bilinear interpolation
img_rotated = cv2.warpAffine(img, M, (w, h), flags=cv2.INTER_LINEAR, borderMode=cv2.BORDER_REPLICATE)
```

הfonקציה `cv2.getRotationMatrix2D` משמשת ליצור מטריצת סיבוב דו-ממדית על תמונה, המאפשרת לסובב את התמונה סביב נקודת מרכז מוגדרת בזווית רציה.

פרמטרים

1. **Center:** נקודת המרכז של הסיבוב בפורמט (y, x) סביר נקודת זו התמונה תסובב. בדרך כלל בוחרים במרכז התמונה כדי שהסיבוב יהיה "במקומם".
2. **rotation_angle:** הזווית (במעלות) שבה רוצים לסובב את התמונה.
 - זווית חיובית תגרום לסיבוב נגד כיוון השעון.
 - זווית שלילית תגרום לסיבוב עם כיוון השעון.
3. **Scale:** קנה המידה של התמונה. כאשר `scale` הוא 1.0, התמונה שומרת על גודלה המקורי.
 - ערך גדול מ 1.0 יגדיל את התמונה.
 - ערך קטן מ 1.0 יקטין אותה.

הfonקציה `cv2.warpAffine` משמשת לעיות או שינוי של תמונה באמצעות מטריצת טרנספורמציה לינארית. זהו כל עוצמתי בעיבוד תמונה שמאפשר לבצע פעולות כמו תרגום, סיבוב, שינוי קנה מידת והטיות.

בfonקציה זו ישנו מספר משתנים:

- `img_rotated` - זהו משתנה שמקבל את התמונה החדשה אחרי השינוי (הסיבוב במקרה זה).
`cv2.warpAffine(img, M, (w, h), flags=cv2.INTER_LINEAR
-borderMode=cv2.BORDER_REPLICATE)`

הפונקציה `cv2.warpAffine` לוקחת את התמונה המקורית (`img`) , ומבצעת עליה עיות או טרנספורמציה בהתאם למטריצת הטרנספורמציה (`M`) , שהיא מטריצה סיבוב במקרה זה.

img - התמונה המקורית שאותה אנחנו רצים לסובב.

M - מטריצת הטרנספורמציה, שמכילה את פרט הסיבוב. את מטריצת הסיבוב זה אפשר לחשב עם `cv2.getRotationMatrix2D` מטריצה זו מגדרה את זווית הסיבוב (במעלה), נקודת המרכז (סיבוב) מהבצע הסיבוב) ושינוי קנה מידה (במקרה הצורך).

(h, w) - הגודל של התמונה החדשה לאחר הסיבוב. במקרה זה (`h, w`), מייצג את הרוחב והגובה המקוריים של התמונה, כך שהתמונה תישאר בגודל שלא לאחר הסיבוב.

flags=cv2.INTER_LINEAR - מצין את שיטת האינטראפולציה שבה משתמש בעת שינוי הפיקסלים. כאן משתמשים ב, `cv2.INTER_LINEAR`-שזה אומר "אינטראפולציה בילינארית". השיטה הזאת מסייעת להחליק את הפיקסלים ולעשות את המעבר ביניהם יותר טבעי.

borderMode=cv2.BORDER_REPLICATE - מצין את שיטת מילוי הגבולות, כלומר, מה עושים עם הפיקסלים הריקים שיוכלים להופיע בקצוות אחרי הסיבוב.

BORDER_REPLICATE, הפיקסלים בקצוות משוכפלים החוצה, כך שהשטח הריק מתמלא בפיקסלים שימושתיים מהשורה והעמודות הקרובות.

אינטראפולציה בילינארית היא שיטה לחישוב ערכים חדשים בנקודות שאינן ידועות ברשות של ערכים, כאשר נעזרים בערכים הידועים בנקודות הקרובות ביותר. במקרה של עבודה תמונה, אינטראפולציה בילינארית משמשת בדרך כלל להגדלת התמונה, סיבוב או מתייה שלה.

באינטראפולציה בילינארית, הערך של כל פיקסל חדש מחושב על בסיס הערכים של ארבעת הפיקסלים הקרובים ביותר בתמונה המקורית. אם ונניח שהפיקסלים הסמוכים נמצאים במרחק 2×2 מסיבב לפיקסל שנחננו רצים לחשב את ערכו, אז אינטראפולציה בילינארית מבצעת את השלבים הבאים:

1. **בחירה ארבעת הפיקסלים הקרובים**: נניח שהפיקסל החדש שאנו רצים לחשב את ערכו נמצא בין ארבעה פיקסלים בתמונה המקורית (נגדי A, B, C, D).

2. **חישוב ביניים אחד**: תחילתה, מבצעים אינטראפולציה בקו אחד (לדוגמא, בכיוון ה-X). מחשבים שערći ביניים על פי ערכי הפיקסלים A ו B בצד אחד, ו C ו D בצד השני. כל ערך ביניים מחושב באמצעות אינטראפולציה לינארית פשוטה בין שני פיקסלים לאורך אותו קו.

3. **חישוב ביניים שני**: לאחר שמת�בלים שני הערכים הביניוניים בכיוון ה-X, מבצעים אינטראפולציה נוספת בכיוון ה-Y בין הערכים הביניוניים שכבר קודם לנו, כדי לקבל את הערך הסופי של הפיקסל החדש.

במקרה שלנו נרצה לסובב את התמונה. כאשר כל פיקסל בתמונה החדש מתרגם חזקה למקומות בתמונה המקורי, לרוב הוא "יפול" על נקודה שבין פיקסלים קיימים, ולכן צריך לחשב את ערכו של פיקסל חדש זה על סמך הערכים של הפיקסלים הסמוכים לו בתמונה המקורי – כאן נכנסת האינטראפולציה הבילינארית.

הפרמטר `borderMode=cv2.BORDER_REPLICATE` משמש לטיפול בגבולות של תמונה במהלך פעולה עיבוד, כמו סיבוב, מתייה או יישום של מסנן.

כאשר מבצעים עיבוד תמונה כמו סיבוב או פילטרים על תמונה, נדרשים ערכים של פיקסלים מעבר לגבולות התמונה, במיוחד באזורי שבקצהה. לאחר מכן פיקסלים מחוץ לגבולות, יש לקבוע כיצד למלא את האזור זה.

מה עושה BORDER_REPLICATE

האפשרות cv2.BORDER_REPLICATE אומרת למלא את הפיקסלים החסרים בגבולות התמונה באמצעות השכל של הערך של הפיקסל הקרוב ביותר בקצת התמונה. כלומר, הפיקסלים בקצת התמונה "מעתקים" החוצה וכן נוצרת המשכיות, שמאפשרת לעבוד על כל התמונה מבלי להשאיר רווחים ריקים.

דוגמה

נניח שאנו מסובבים תמונה ב-15 מעלות. במהלך הסיבוב ייווצרו פינות ריקות כי התמונה נוטה בזוויות, ובמקומות שהזוויות הריקות ישארו "ש קופות" או עם ערכים חסרים. BORDER_REPLICATE משכפל את הפיקסלים של הגבולות וממלא את הפינות הריקות באותו צבעים או ערכי פיקסלים של הפיקסלים הקרובים ביותר.

יתרונות וחסרונות

- **יתרון :** יוצר מראה אחיד ומונע רווחים ריקים בקצוות התמונה.
- **חיסרון :** במקרים מסוימים עלול להוביל לאזורי לא טבעיות בקצת התמונה, מכיוון שהකצאות משוכפלים באופן שלא תמיד מתאים לתוך הפנימי של התמונה – מה שקרה במקרה שלנו.

פתרון:



- ב. נමיש את הפונקציה `second_op(input_image_path, output_folder_path)` הפעnkציה מקבלת כקלט את התמונה `img` ומקום של תיקיה אליה ישמר הפלט של הפונקציה.
תמונה הקלט: `img_firststop.png`
תיקיית הפלט: `outputs-b`

```
def second_op(input_image_path, output_folder_path):
    """
    Crop the image to remove black rectangles (central crop).

    Parameters:
    input_image_path (str): Path to the input image.
    output_image_path (str): Path to the cropped output image.
    """

    # Load the image and check if it's color or grayscale
    img = cv2.imread(input_image_path)
    if img is None:
        print(f"Error: Could not load image from {input_image_path}")
        return

    # Get the image dimensions
    height, width = img.shape[:2]

    # Define the crop box (cropping 15% from right and left, and 20% from above and below)
    left = int(width * 0.15)
    upper = int(height * 0.20)
    right = int(width * 0.85)
    lower = int(height * 0.80)

    # Crop the image
    img_cropped = img[upper:lower, left:right]

    # Save the cropped image
    cropped_image_path = f"{output_folder_path}/img_secondop.png"
    cv2.imwrite(cropped_image_path, img_cropped)

    return cropped_image_path
```

מחוץ לפונקציה - הגדרת תמונה הקלט, תיקיית הפלט (נותרה זהה אך לא נעדכן) ודימון האופרציה:

```
# Construct the path to "img_firstop.png" inside the "outputs-b" folder
input_image_path = os.path.join(script_dir, "outputs-b", "img_firstop.png")
# # Second operation: Cropping
second_op(input_image_path, output_folder_path)
```

הפעולה השנייה שבחרתי לעשות על התמונה היא **חיתוך התמונה להסרת קצוות/גבולות לא רצויים** שהופיעו לאחר הסיבוב (פעולה ראשונה). לאחר סיבוב התמונה בפעולה הראשונה, הופיעו אזורים מוטשטשים בפינות של התמונה. חיתוך מאפשר לי להסיר את הקצוות הללו, וכיצדאה מכך לקבל תמונה נקייה וOMEMOKDAT YOTER. חיתוך גם עוזר לשמר על החלק המרכזי והרלוונטי ביותר של התמונה, ומכך אותה לשלב ניתוח נוספת לנוסף להסחות דעת מהקצוות המעוותים.

```
# Crop the image
img_cropped = img[upper:lower, left:right]
```

החיתוך נעשה בעזרת חיתוך מערך, וההתוצאה החתוכה נשמרה כ-`img_secondop.png`.

ההבדל בין התמונה המסובבת לתמונה החתוכה שהתקבלה:

הסרת גבולות מעוותים: הקצוות המעוותים שהוצעו בשלב הקודם הוסרו, ונשאר רק החלק המרכזי של התמונה. מכיוון שהחיתוך הוא מלבי, נחתכו גם חלקים מהתמונה שאינם הקצוות. וידיאתי שככל מה שנחתך אינם מהווים חלק חיוני בתמונה, זהה בסדר לחיתוך אותו יחד עם הקצוות.

מיוקד בתוכן החשוב: על ידי חיתוך קצוות מיוחדים התמונה עדין מציגה את הפרטים החיוניים (עצים, אנשים, אוזדים, דשא), ומשפרת את נראות של התמונה לעיבוד הבא.

פרמטרים לחיתוך:

גבולות חיתוך: על מנת להפטר מהקצוות הבעניתיים של התמונה, בחרתי להסיר 15% מהצד השמאלי, ומהצד ימני של התמונה ו- 20% מלמלה ומלמטה של התמונה. ערכיהם אלו נבחרו על סמך הכמות האופיינית של קצוות בעייתיים שהופיעו לאחר הסיבוב של התמונה, מה שדרש חיתוך מתון כדי להסיר את הקצוות הבעייתיים מבלי לאיים תוכן חיוני.

פלט:



c. נமמש את הפונקציה `third_op`(`input_image_path`, `output_folder_path`)
הfonkzia makbelat kklut at hthmuna `img_secondop.png` omikom shel tikia alihya yshmer hplut shel
fonkzia.

thmuna haklut: `.img_secondop.png`"

Tikiyit hplut: `outputs-b`"

```
def third_op(input_image_path, output_folder_path):
    """
    Apply a basic sharpening filter to the image to enhance the edges and save it as 'img_thirdop.png'.

    Parameters:
    input_image_path (str): Path to the input image.
    output_folder_path (str): Path to the folder where the output image will be saved.
    """

    # Load the image and check if it's color or grayscale
    img = cv2.imread(input_image_path)
    if img is None:
        print(f"Error: Could not load image from {input_image_path}")
        return

    # Define the sharpening kernel with 5 at the center
    sharpening_kernel = np.array([[0, -1, 0],
                                 [-1, 5, -1],
                                 [0, -1, 0]])

    # Apply the sharpening filter
    img_sharpened = cv2.filter2D(img, -1, sharpening_kernel)

    # Save the sharpened image
    sharpened_image_path = f"{output_folder_path}/img_thirdop.png"
    cv2.imwrite(sharpened_image_path, img_sharpened)

    return sharpened_image_path
```

מחוץ לפונקציה - הגדרת תמונה הקלט, Tikiyit hplut (נותרה זהה אז לא נעדכן) ודימון האופרציה:

```
# Construct the path to "img_secondop.png" inside the "outputs-b" folder
input_image_path = os.path.join(script_dir, "outputs-b", "img_secondop.png")
# # Third operation: Sharpening
third_op(input_image_path, output_folder_path)
```

הפעולה השלישית והאחרונה שבחרתי לעשות על התמונה היא **פעולות סביבה – פילטר חידוד בסיסי**. הפעלת מסנן חידוד בסיסי כדי לשפר קצוט ופרטים בתמונה. המסנן המחידד הוא מטריצה 3×3 עם ערך מרכזי של 5 וארבעה ערכי (-1). הוא מדגיש בעדינות קצוט על ידי הגדלת עוצמת הפיקסל המרכזי ביחס לשכנים. מסנן זה נבחר מכיוון שהוא מספק רמת חידוד מואצת מבלית להכניס ניגודיות או רעש מוגדים, מה שהופך אותו לאיידיאלי להדגשת פרטים עדינים. נראה כך:

```
[ 0, -1,  0],  
[-1,  5, -1],  
[ 0, -1,  0]
```

המסנן הופעל באמצעות (`cv2.filter2D()`), המחליל את אפקט החידוד על פני כל התמונה.

הבדל בין התמונה החתוכה לתמונה המוחודדת שהתקבלה:

בשילובת התמונה הקודמת החתוכה, התוצאה המוחודדת הנוכחית מבוסנת יותר, מדגישה טוב יותר את הפרטים הרלוונטיים בתמונה (עצים, דשא, אוזדים, אנשים), עם בהירות משופרת וקצוט מוגדרים יותר.

פרמטרים לחיתוך:

הערך המרכזי (גרעין) 5 שבמסנן מספק אפקט חידוד מותן מבלית לייצור ניגודיות מוגצתת, בעוד שהערכים (-1) שמסביבם מדגישים ביעילות קצוט בתמונה. הגרעין זהה משייך חידוד מואץ ומשפר פרטים בתמונה.

הסברים מעמיקים למושגים שבוצע בהם שימוש באופרציה זו:

Define the sharpening kernel with 5 at the center

```
sharpening_kernel = np.array([[ 0, -1,  0],  
                             [-1,  5, -1],  
                             [ 0, -1,  0]])
```

Apply the sharpening filter

```
img_sharpened = cv2.filter2D(img, -1, sharpening_kernel)
```

בקטע הקוד הזה אנחנו מגדירים מסנן (Kernel) לחידוד התמונה ומימושים אותו על התמונה המקורית כדי להבליט את הקצוט והפרטים בתמונה. המסנן מחזק את הניגודיות המקומית על ידי חיזוק פיקסלים במקומות עם שינויים חדים בעוצמת התאורה.

1. יצרת מסנן חידוד (Kernel):
- o אנו מגדירים מסנן בגודל 3×3 .

- הערך המركזי במסנן הוא 5, והוא הערך שמודגש, כלומר, הפיקסל המركזי יחזק ממשמעותית.
- הערכים של 1- מסביר לערך המركזי מורידים את הערכים של הפיקסלים השכנים, מה שיוצר חיזוק בניגדיות בין הפיקסל המركזי לסבירתו.
- כך נוצר אפקט שבו פיקסלים במקומות בהם יש שניי חד (למשל בקצוות) הופכיםבולטים יותר.

2. יישום מסנן החידוד:

- הפענציה cv2.filter2D לוקחת את התמונה המקורי (img) ומבצעת עליה קונבולוציה עם מסנן החידוד שיצרנו (sharpening_kernel).
- הפרמטר 1- מצין שהפלט ישמר על אותו עומק כמו התמונה המקורי.
- בעת חישוב הקונבולוציה, כל פיקסל בתמונה המקורי מוחלף בערך חדש שנחושב על סמך הפיקסלים הסובבים אותו, מה שיוצר את אפקט החידוד.

איך זה עובד?

מסנן החידוד מדגיש את ההבדלים בין הפיקסל המركזי לפיקסלים סוביביו. אם הפיקסל המركזי בהיר והשכנים שלו כהים יותר (או להיפך), הערך המתתקבל יגבר את הבבירות או הכהות שלו, מה שמדגיש את המעברים החדים בתמונה ומחדד אותה.

פלט סופי:



- האוזים שבקובוי שבקו שיראו בתמונה המקורי לפני השינויים, נראים בבירור בתמונה החדשה. התמונה הזו, לאחר ביצוע החידוד, היא הגרסה המתואמת והמשופרת ביותר מכיוון שהיא משלבת ישור, מיקוד ובהירות יחד עם הדגשה משופרת של קצוות בעדרת מסנן החידוד.

בתמונה זו:

ישור: הסיבוב הראשון תיקן כל הטיה, ישר את התמונה והציב בסיס מוצק לעיבוד נוספת. מיקוד: חיתוך הקצוות הסיר גבולות מיוחדים, ואפשר לשים את תשומת הלב בתוכן הראשי ללא הסחות דעת. בהירות: פעולה החידוד הוסיפה שיפור עדין ומואזן לקצוות, מה שגרם לפרטיים עדינים לבנות בצורה ברורה יותר מבליל יצור רעש לא רצוי. מסנן חידוד סופי זה מביא לידי ביטוי טקסטורות ופרטים בתמונה, ומיצר תמונה נקייה, חדה ומושכת חזותית. ההשפעה המשולבת של פעולות אלו מביאה לתמונה מעודנת ביותר, שלפי דעתינו, מדגישה את המאפיינים החיוניים ביעילות.

הסברים על שימוש בפונקציות מוכנות מראש:

ספרית CV2

```
import cv2
```

ספריית cv היא הממשק של OpenCV ל Python (ספרייה ראייה ממוחשבת ולמידת מכונה בקוד פתוח). זו ספרייה פופולרית לעיבוד תמונה המציעה כלים למגוון רחב של שימושות עיבוד תמונה.

סקירה קצרה של הפונקציות המרכזיות ש-`cv` מציעה ואיך הן יושמו בקוד שלך:

- טיענת ושמירת תמונה:

באמצעות `cv`, ניתן לטען תמונות מקובציים ולשמור את התמונות המעובדות חזרה לקובציים. לדוגמה:

```
img = cv2.imread('image.png') # טיענת תמונה
cv2.imwrite('processed_image.png', img) # שמירת תמונה
```

- המרות צבע:

OpenCV מאפשרת המרות בין מרחבי צבע שונים (למשל, RGB, גווני אפור HSV). בקוד שלך השתמשתי ב `cv2.cvtColor` כדי להמיר תמונות לגווני אפור:

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

פעולה זו מסייעת בניתוח התמונה על ידי הסרת מידע הצבע והתקדמות רק בעוצמות.

- שינוי תמונה:

OpenCV כוללת פונקציות לשינויים בתמונה, כגון שינוי גודל, סיבוב, חיתוך ועיוות.

לדוגמה, בפועלות הסיבוב שלו:

```
M = cv2.getRotationMatrix2D(center, angle, scale)
rotated_img = cv2.warpAffine(img, M, (width, height))
```

קוד זה יוצר מטריצת טרנספורמציה עבור סיבוב ומישם אותה על התמונה.

- סינון וקונבולוציה:

OpenCV מציעה פונקציות ליישום מסננים שונים על תמונות. השתמשתי בהן בפועלות החידוד שלי באמצעות גרעין קונבולוציה:

```
sharpened_image = cv2.filter2D(img, -1, sharpening_kernel)
```

כאן `cv2.filter2D`, מבצע קונבולוציה על התמונה באמצעות גרעין (`kernel`) מסוים, שיכולה לחדר קצוט ופרטים בתמונה בהתאם לערכי הגרעין.

- חישוב גרדיאנט:

OpenCV כוללת פונקציות לזיהוי קצוטות וחישוב גרדיאנטים, כמו אופרטור Sobel שבו השתמשתי:

```
gradient_x = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=3)
```

```
gradient_y = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=3)
```

קוד זה מחשב גרדיאנטים בכיווני X ו-Y, מה שעזר בזיהוי קצוטות ושינויים בעוצמת האור.

- חישוב היסטוגרמה:

בספריית OpenCV הפונקציה `cv2.calcHist` מחשבת את ההיסטוגרמה של ערכי הפיקסלים בתמונה, ומשמשת לניתוח התפלגות ערכי הפיקסלים. ההיסטוגרמה מציגה את כמות הפיקסלים לכל ערך אינטנסיביות (עוצמת צבע), ובכך מספקת תמונה בורורה על רמת התאורה והניגודיות של התמונה. בקורס שילוי:

```
hist = cv2.calcHist([image], [i], None, [256], [0, 256])
```

לחישוב גרדיאנטים השתמשתי בפילטר סובל, אציג הסבר קצר:

מסנן סובל(Sobel)

מסנן סובל טכניתה נפוצה לזייהו קצוטות בעיבוד תמונה, המיעדת להציג אזורים בעלי תדריות מרוחبات גבוהות - כמו קצוטות. המסנן פועל על ידי שימוש קובולציה עם גרעינים ספציפיים שמחשבים את קצב השינוי בעוצמת הפיקסלים (או "גרדיאנט") בתמונה. מסנן זה מועיל במיוחד למציאת הקצוטות לציר ה-A-ובציר ע, המיצגים את הגרדיאנטים האופקיים והאנכיים בהתאם.

יסודות מסנן סובל:

מסנן סובל מיישם שני גרעינים נפרדים על תמונה כדי לחשב את הגרדיאנטים בכיוון X ו- Y, כל גרעין הוא מטריצה קטנה המיועדת להציג שינויינו בעוצמה בכיוון מסוים.

המסנן משתמש בשני גרעיני 3X3:

- Ax: לזייהו שינויי אופקיים.
- Gy: לזייהו שינויי אנכיים.

$$\text{גרuin אופקי}(x) : \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\text{גרuin אנכי}(y) : \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

גרעינים אלה פועלים על ידי החלקה על פני התמונה וביצוע כפל רכיבי עם הפיקסלים שמתחthem. התוצאה היא ערך פיקסל חדש המשקף את השינוי בעוצמה בכיוון נתון. הבדל גדול בעוצמה מביא לתגובה חזקה של הקצה.

חישוב הגרדיאנטים ב-X ו- Y :

הפונקציה `cv2.Sobel` מיישמת את הגרעינים הללו כדי לחשב את הגרדיאנט בכיוון X ו- Y:

`gradient_x = cv2.Sobel(gray_image, cv2.CV_64F, 1, 0, ksize=3)`: מחשב את הגרדיאנט בכיוון X על ידי שימוש בגרuin ה-X (1 בפרמטר x).

`gradient_y = cv2.Sobel(gray_image, cv2.CV_64F, 0, 1, ksize=3)`: מחשב את הגרדיאנט בכיוון Y על ידי שימוש בגרuin ה-Y (1 בפרמטר y).

cv2.CV_64F מציין שאנו רוצים תוצאה בפורמט 64-ビט, המבטיח דיוק בחישוב הגרדיינט, במיוחד אם ישנו ערכים שליליים.

חישוב גודל הגרדיינט:

לאחר קבלת `x_gradient`, `y_gradient`-השלב הבא הוא לחשב את גודל הגרדיינט. גודל הגרדיינט נתן ממד לשני הcoli בעצמה, ומשלב את השפעות הגרדיינטיים של `x` ו-`y`.

גודלו הגרדיינט מחושב כך:

$$\sqrt{(gradient_y)^2 + (gradient_x)^2} = gradient_magnitude$$

ב-OpenCV הדבר נעשה באמצעות (`y_magnitude`, `gradient_x`, `gradient_y`) אשר מחשבת את הנורמה האוקלידית של הגרדיינטיים. פועלה זו מציגת את עוצמת הקצוות ללא תלות בכיוונם, ומדגישה את כל הקצוות בתמונה.

שמירת תמונות הגרדיינט:

התוצאות של מסנן סובול `x_gradient` ו- `y_gradient` נשמרות כתמונות. תמונות אלה מראות היכן מתרחשים שינויים בעצמה בכיוונים אופקי ואנכי.

תמונה `gradient_magnitude` מייצגת את עוצמת הקצוות הכוללת, ומספקת מבט ברור יותר על כל הקצוות בתמונה.

AIR מסנן סובול מסיע בדיהו' קצוות:

דיהו' קצוות: מסנן סובול מדגיש אזורים שבהם עוצמת הפיקסלים משתנה במהירות, מה שמתאים לרוחב לקצוות או גבולות בין עצמים.

מידע על כיוון: על ידי חישוב גרדיינטיים הן בכיווני `X` והן בכיווני `Y`, מסנן סובול מספק מידע על כיוון הקצוות.

ניתוח תמונה וחילוץ מאפיינים: קצוות הם מאפיינים קריטיים בתמונות, וסובול הוא בחירה נפוצה לחילוץ קצוות, ומשמש במקרים עיבוד נוספים כמו דיהו' אובייקטים וקטועי תמונה.

קבלת ספירה הסקריפט שלי על מנת לשמר בה את כל הקבצים

```
((script_dir = os.path.dirname(os.path.abspath(__file__)))
```

שורה זו מוצאת את הספרייה בה נמצא הסקריפט הנוכחי.

```
__file__ מספק את הנתיב של הסקריפט,
```

ו- os.hopf אותו לנטייב מלא מהשורש של המערכת, לא משנה מאייפה הסקריפט הופעל. os.path.dirname(script_dir) מוציא את החלק של הספרייה ושומר אותו ב-var dir_name.

בנייה נתיב לתמונה

:input_image_path = os.path.join(script_dir, 'image.png')

השורה זו יוצרת את הנתיב המלא לתמונה בשם 'image.png' על ידי שילוב script_dir עם שם הקובץ. כך, הסקריפט יוכל לחפש את התמונה באותה תיקיה בה הוא נמצא.

הגדרת תיקייה פלט

:output_folder_path = os.path.join(script_dir, 'output')

השורה זו מגדירה את path.output על ידי שילוב script_dir output עם ".output". תיקייה זו תשמש לאחסון קבצי הפלט.

ג. הציגו וניתוח של התמונה המתוקנת

.img_thirdop.png', אך כעת הפונקציה מקבלת כקלט את התמונה המתוקנת כולם, נפעיל את הפונקציה (analyze_image(input_image_path, output_folder_path) על התמונה המתוקנת.

הfonקציה היא אותה הפונקציה שמיימשנו בסעיף א'.

תמונה הקלט: ".img_thirdop.png"

תיקית הפלט: ."outputs-c"

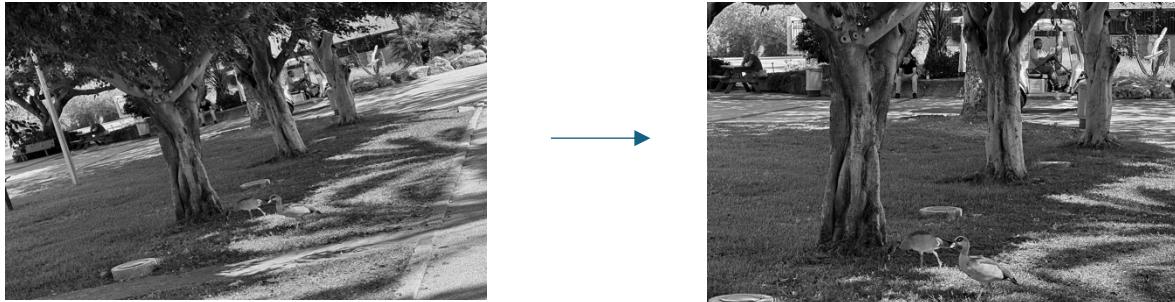
2c -----

```
# Create a generic "outputs-c" folder within the same directory as the script
output_folder_path = os.path.join(script_dir, "outputs-c")

# Check if the "outputs-c" folder exists; if not, create it
if not os.path.isdir(output_folder_path):
    os.makedirs(output_folder_path)

# Construct the path to "img_thirdop.png" inside the "outputs-b" folder
input_image_path = os.path.join(script_dir, "outputs-b", "img_thirdop.png")
analyze_image(input_image_path, output_folder_path)
```

1. קיבלנו את התמונה img_thirdop.png שלמו כמו שהוא, שהרי התמונה המתוקנת היא כבר בגוני אפור. התמונה בגוני אפור הפעם היא תמונה מתוקנת בהשוואה לתמונה בגוני אפור שהייתה בסעיף א' על התמונה המקורי. קל להבחן בשינויים (סיבוב, חיתוך, חידוד) שעברה התמונה המתוקנת בהשוואה לזה המקורי שהציגי בסעיף א'.



2. ישם הבדלים ניכרים בין ההיסטוגרמות האפורות לאחר ביצוע כל אחת מהאופרציות, בעיקר בגובה הפסגות ובחלקות התתפלגות. ההיסטוגרמה של התמונה חוק.ckpt נמוכה יותר, ככלור התדריות (או הספירה) של הפיקסלים בכל רמת עוצמה נמוכה יותר.

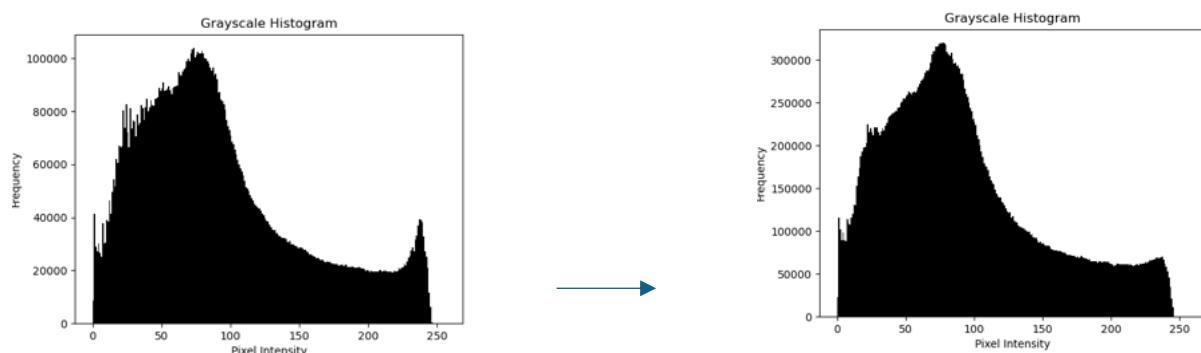
אנסה להסביר בקצרה כיצד הושפעה ההיסטוגרמת האפור כתוצאה מכל אחת מהאופרציות:

אופרציה ראשונה – סיבוב התמונה

כאשר מסובבים תמונה, במיוחד בזוויות כמו 15 מעלות, חלקים מסוימים של התמונה נעים מהמרכז אל הקצוות. כדי לשמר את הצורה המלבנית, אינטראפולציה מלאת פיקסלים לאורך הקצוות. זה מציג לעיתים קרובות מספר גדול יותר של פיקסל "ירקע" עם ערכיהם קרובים לאפור בפועל, בהתאם לשיטת האינטראפולציה.

האינטראפולציה מזיגה ערכי פיקסל מעורבבים בין פיקסלים שכנים כדי להחליק את הקצוות המסובבים. מיזוג זה מוביל לוחב ליותר עצמות בגווני אפור בטוחה הבנינים, מה שגורם להיסטוגרמה של `img_firstop` להיראות חלקה יותר עם פחת פסגות אינטנסיביות.

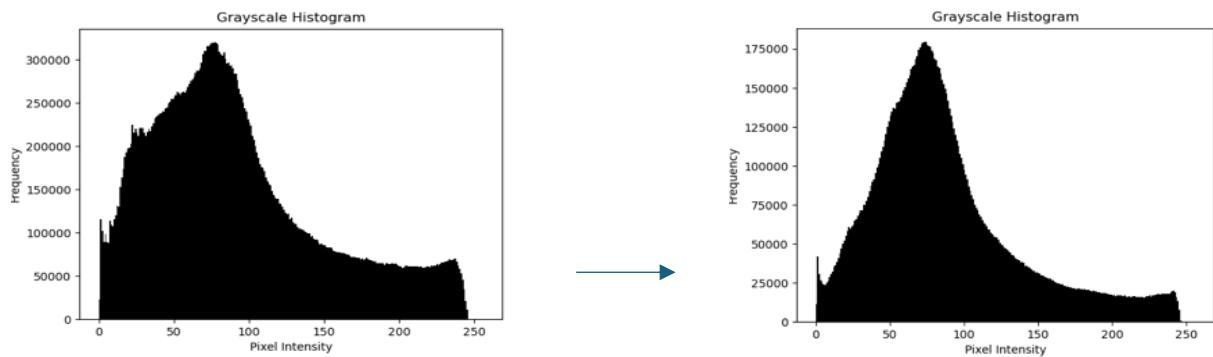
عقب הסיבוב והאינטראפולציה, הניגודיות והחדות של התמונה באזוריים מסוימים עשויים לרדת מעט. כתוצאה לכך, התפלגות ההיסטוגרמה עשויה להיות פחות מרכזית בטוחה עצמה מסוימים, בהשוואה למקור, שלו פסגות חדות וגבשות יותר.



אופרציה שנייה – חיתוך התמונה המסובבת

הבדלים בין ההיסטוגרמות האפורות של (`img_firstop.png` לפני חיתוך) ו- (`img_secondop.png` לאחר חיתוך) משקפים את ההשפעה של פעולה החיתוך. הריסתווגרמאות `img_firstop` ו- `img_secondop` מודדים דדה ודווחה יותר, עם פחות וריאציות בולטות בעוצמה. זה בעיקר בגלל שהחיתוך הסיר אזוריים עם גבולות שחורים או אזוריים כהים אחרים, שהיו חלק מהתמונה המסובבת ב-`img_firstop`. למשל החלק העליון של העצים הוא חלק כהה מאוד שהוסר לאחר החיתוך. כפי שציינו קודם, הקצוות שנוצרו עקב הסיבוב הם ברובם פיקסלים אפורים מרכזיים שגם מוסרים בפעולה החיתוך.

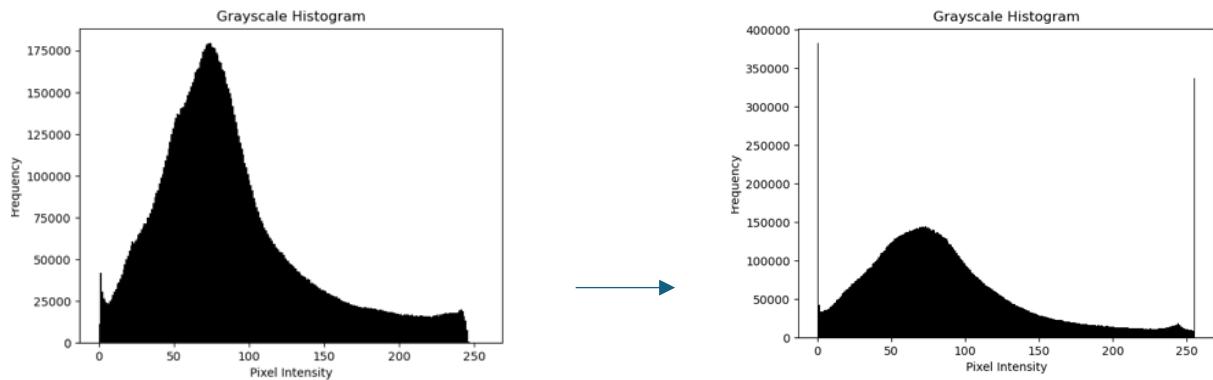
אותם אזוריים שחורים ואפורים מוסרים, מה שגורם להיסטוגרמה להצטמצם ולהתאחד. שינוי זה מפחית את הנוכחות של פיקסלים, מה שוביל להיסטוגרמה שמתמקדת יותר בתוכן ממשי של התמונה, כך שעוצמות רמה ביןוניות בין אפור לשחור בולטות יותר.



אופרציה שלישית – חידוד התמונה לאחר חיתוך וסיבוב

לאחר ביצוע פעולה החידוד, ניתן להבחין בהבדלים בהיסטוגרמאות בין `qck.secondop.png` (לפני החידוד) לבין `qck.thirdop.png` (אחרי החידוד).
ההיסטוגרמאות של התמונה המחדודה (`qck.png`) נמצאה יותר בהשוואה לזה של `qck.secondop.png`. ירידת הגובה נובעת מכך שפעולות החידוד מפזרת את עוצמות הפיקסלים, במיוחד באזורי ביןדים, ומגבירת את הניגודיות (קונטרסט).

בנוסף, השיאים החדשניים בהיסטוגרמה (0 ו-255), מצביעים על כך שפעולות החידוד הגבירו את הקונטרסט והביאה לפיזור רחב יותר בעוצמות הפיקסלים, כאשר קצנות התמונה מדורגים בקיצון של כהות ובהירות. זהו סימן לשיפור בקונטרסט ובדיק, ומראה על עיבוד אינטנסיבי שכול להבליט פרטים בצורה יתרה – יתרון שימושוניים בהדגשת פרטים בתמונה.



עבור סעיפים 3,4,5 (היסטוגרמות של הצבעים אדום, כחול וירוק) לא נקלט כלום, שכן, התמונה המעובדת לאחר שלוש האופרציות היא תמונה בגוני אפור. אין כל ערך להציג היסטוגרמות צבעוניות עבור תמונה בגוני אפור.

התשובה הבאה מתייחסת ל-3 הסעיפים הבאים בנושא גרדיאנטים:

ניתן לראות כי הפליט של התמונות המעובדות בהיר יותר. הגרדיאנט הבהיר יותר ב `ok3dop_im` - מעיד על כך שהקצוות והניגודיות בתמונות מודגשים יותר.

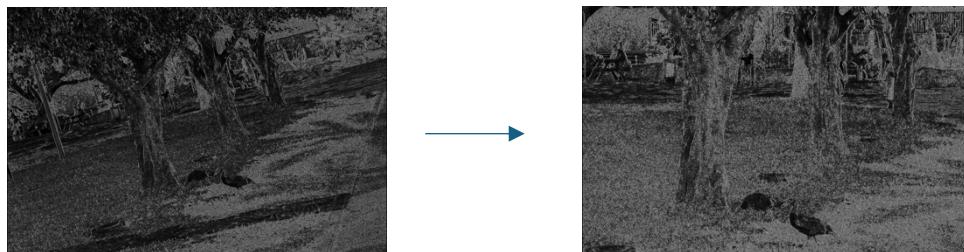
תחילה, פועלת הסיבוב שבוצעה כאופרציה ראשונה משנה את הגרדיאנט בתמונה. הסיבה לכך היא שגרדיאנטים מייצגים את כיוון ועוצמת השינויים בעוצמת הפיקסלים (בהירות), כלומר את המעברים בתמונה. כאשר מוסבבים תמונה, הכוונים של קווים המתאר משתנים בהתאם לזווית הסיבוב. זה משפיע על הגרדיאנט, כי הכיוון והערכים של הגרדיאנט מבוססים על האוריינטציה של קווים המתאר בתמונה המקורי. זווית של 15 מעלות היא יחסית קטנה, כך שהשינוי בכיווני הגרדיאנטים יהיה קל, אך עדין יורגש כטשטוש קל בכיוון האופקי והאנכי של קווים המתאר המקוריים.

שנית, החיתוך שבוצע כאופרציה שנייה משפיע על הגרדיאנט בתמונה בכך שהוא משנה את עוצמתו ואת התפזרותו. כאשר חלקיים נחתכים, קווים המתאר חסובים עשויים להימחק, מה שמקטין את עוצמת הגרדיאנט באזוריים שנפגעו. לבסוף, הקצוות החדשניים שנוצרים בעקבות החיתוך יכולם להוביל להופעת גרדיאנטים חדים לאורק קווי החיתוך. כך, בעוד שהחיתוך לא משנה את כיווני הגרדיאנטים המקוריים, הוא בהחלט משפיע על תפוצת ועוצמת הגרדיאנטים בתמונה.

לבסוף, באופרציה השלישיית חידדתי את התמונה. החידוד מגדיל את ההבדלים בעוצמות לאורק הקצוות, מה שmobiel לgradianant חזק יותר (ערכי פיקסל גבוהים יותר) ולתמונה כללית בהירה יותר של הגרדיאנט.

סה"כ, ניתן לראות כאן את הגרדיאנטים בציריהם ואת מפת הגרדיאנטים הכללית:

6. מפת גרדיאנטים בכיוון אופקי X -



7. מפת גרדיאנטים בכיוון אנכי י'



8. מפת עוצמת הגרדיאנטים –



ד. סעיף בונוס – שיפור התמונה המקורי

הפעלתה על התמונה הצבעונית המשויכת אליו את אותן הפעולות שהפעלתה על התמונה המקורי אפוא. הפעולות הן סיבוב (טרנספורמציה גיאומטרית), חיתוך וחידוד (פעולות סביבה – פילטר חדוד בסיסי).

תיקיות הפלט: “outputs-d”.

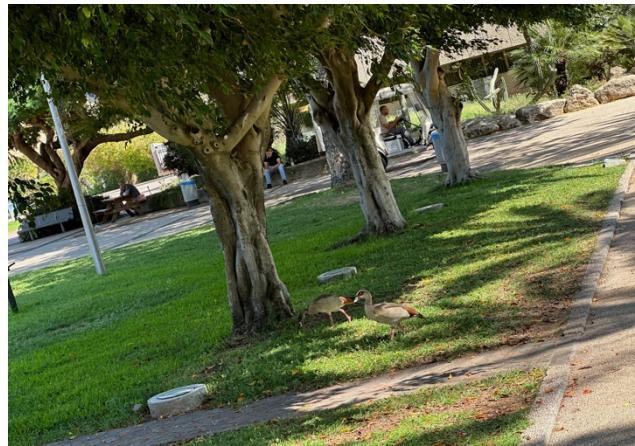
תמונה הקלט הראשונית: ”DIP_803.png”.

תמונה הקלט לאופרציה השנייה: ”img_firstop.png”.

תמונה הקלט לאופרציה השלישית: ”img_secondop.png”.

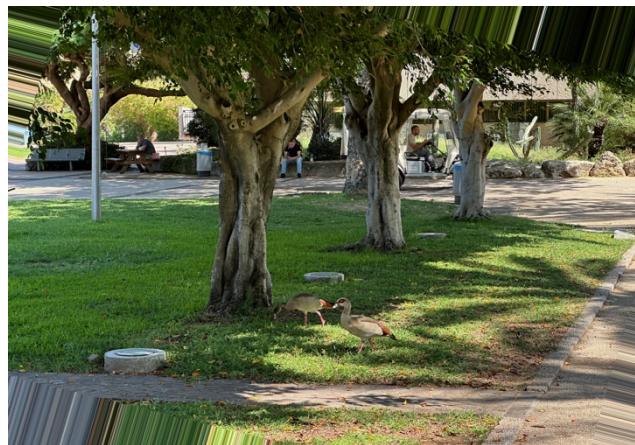
```
# 2d -----  
  
# Construct the full path to the image  
input_image_path = os.path.join(script_dir, 'DIP_803.png')  
  
# Create a generic "outputs-d" folder within the same directory as the script  
output_folder_path = os.path.join(script_dir, "outputs-d")  
  
# Check if the "outputs" folder exists; if not, create it  
if not os.path.isdir(output_folder_path):  
    os.makedirs(output_folder_path)  
  
# # First operation: Rotation  
first_op(input_image_path, output_folder_path)  
  
# Construct the path to "img_firstop.png" inside the "outputs-d" folder  
input_image_path = os.path.join(script_dir, "outputs-d", "img_firstop.png")  
# # Second operation: Cropping  
second_op(input_image_path, output_folder_path)  
  
# Construct the path to "img_secondop.png" inside the "outputs-d" folder  
input_image_path = os.path.join(script_dir, "outputs-d", "img_secondop.png")  
# # Third operation: Sharpening  
third_op(input_image_path, output_folder_path)
```

התמונה המקורית לפני שינויים:



ג. מכיוון שזו הפעם שעלייה עשינו את כל האופציות בסעיף ב' (רק הפעם התמונה צבעונית), נסובב גם אותה ב 15 מעלות נגד כיוון השעון כדי לסייע את התמונה. כפי שWOODAI מיטן לראות, התמונה מעת מסובבת, וסיבוב שלה חזרה כדי שתיה ישירה יעזור להבחן טוב יותר בפרטים בתמונה מלבד לסובב את הראש. התחילה עם סיבוב מבטיחה שהתמונה מכוונת נכון לשלי עיבוד נוספים.

התמונה המקורית לאחר טרנספורמציה גיאומטרית (סיבוב):



התמונה אומנם מיושרת, אך נוצרו מושלשים מוטשטשים בקצוות התמונה.

- d. ניעזר בפועלות החיתוך לחיתוך התמונה כדי להיפטר מקצוות מוטשטשים וכך להתמקד בפרטים החשובים באמת בתמונה (עצים, אנשים, אוודים, דשא).
מכיוון שזו אותה התמונה כמו בתמונה בסעיף ב' (רק הפעם צבעונית), החיתוך יהיה אותו חיתוך בדיק;
נחתוך 15% מהצד השמאלי ומהצד ימני של התמונה ו- 20% מלמעלה ומלמטה של התמונה.

התמונה המקורית לאחר סיבוב וחיתוך:



החיתוך חשף תמונה מעט מוטשטשת ולא מספיק جدا.

c. **התמונה המקורית לאחר סיבוב חיתוך וחידוד:**

לאחר החיתוך הפעיל פילטר חידוד בסיסי.

המסנן המحدد הוא מטריצה 3×3 עם ערך מרכזי של 5 וארבעה ערכי (-1). בדומה לתמונה המקורי אפור שבסעיף ב', מסנן זה נבחר מכיוון שהוא מספק רמת חידוד מאוזנת מבליל להכנס ניגודיות או רעש מוגזמים, מה שהופך אותו לאיידיאלי להדגשת פרטים עדינים. הוא מדגיש בעדינות קצotta על ידי הגדלת עצמת הפיקסל המרכזי ביחס לשכנים.



פעולה זאת השלים את שיפור התמונה והפכה אותה לברורה, חדה, מוקדת ומליטה את הפרטים החשובים בה.

ה. **סעיף בונוס – תמונה מהגלארייה שלי (חלון המטוס בדרך למלואים ☺)**

התמונה היא תמונה מהמטוס בדרך לבסיס בדרום הארץ.
הצלחות להוות את השכונה שלי מחלון המטוס ☺, היה מעניין לראות אותה מבט מלמעלה.

הפעלתה על התמונה החדש את אותן האופרציות שהפעלתה בסעיף ב'.
הפעולות הן סיבוב (טרנספורמציה גיאומטרית), חיתוך וחידוד (פיעולת סביבה – פילטר חידוד בסיסי).
אנסה בעזרת האופרציות הבאות לשפר את התמונה כדי להצליח לראות טוב יותר את השכונה.

תיקיית הפלט: "outputs-e".

תמונה הקלט הראשונית: "my_image.png".

תמונה הקלט לאופרציה השנייה: "img_firstop.png".

תמונה הקלט לאופרציה השלישית: "img_secondop.png"

2e -----

```
# Create a generic "outputs-e" folder within the same directory as the script
output_folder_path = os.path.join(script_dir, "outputs-e")

# Check if the "outputs-e" folder exists; if not, create it
if not os.path.isdir(output_folder_path):
    os.makedirs(output_folder_path)

# Construct the full path to the image-
input_image_path = os.path.join(script_dir, 'my_image.png')

# # First operation: Rotation
first_op(input_image_path, output_folder_path)

# Construct the path to "img_firstop.png" inside the "outputs-e" folder
input_image_path = os.path.join(script_dir, "outputs-e", "img_firstop.png")
# # Second operation: Cropping
second_op(input_image_path, output_folder_path)

# Construct the path to "img_secondop.png" inside the "outputs-e" folder
input_image_path = os.path.join(script_dir, "outputs-e", "img_secondop.png")
# # Third operation: Sharpening
third_op(input_image_path, output_folder_path)
```

התמונה המקורית לפני שינוי:



ג. כפי שודאי ניתן לראות, התמונה מעט מסובבת, וסובב גם אותה ב 15 מעלות נגד כיוון השעון כדי לישר את התמונה. סיבוב של התמונה חזקה כדי שתיהי ישירה יעזר להבחין טוב יותר בנוף מהחלון, מבלי לסובב את הראש. התחילה עם סיבוב מבטיחה שהתמונה מכונה היטב בשלב עיבוד נוספים.

התמונה שלי לאחר טרנספורמציה גיאומטרית (סיבוב):



התמונה אומנם מישרת, אך נוצרו מושלמים מטושטשים בקצוות התמונה.

ד. ניעזר בפעולות החיתוך לחיתוך התמונה כדי להיפטר מקצוות מטושטשים וכדי להתמקד בקהלות יותר בפרטם החשובים באמת בתמונה (השכונות והבניינים).

החלון של המטוס מופיע בהתקומות בנוף, והשמות הכהולים מסיטים את תשומת הלב מהשכונה. נחיתור 15% מהצד השמאלי ומהצד הימני של התמונה ו- 20% מלמעלה ומלמטה של התמונה.

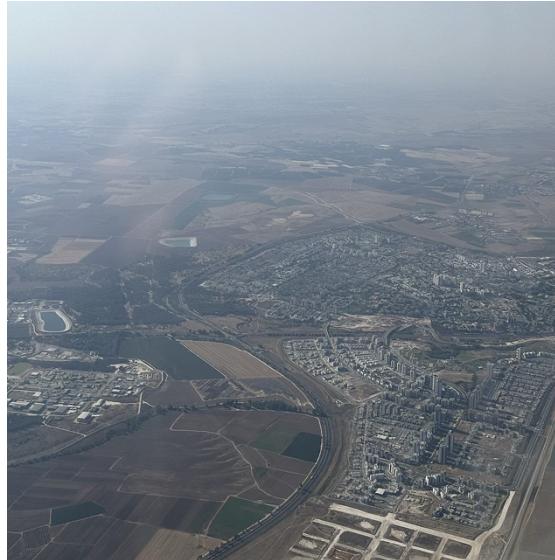
התמונה שלי לאחר סיבוב וחיתוך:



החתוך חשף תמונה מעט מטושטשת ולא מספיק ברורה.

כ. לאחר החיתוך הפעلت פילטר חידוד בסיסי.
המסנן המحدد הוא מטריצה 3×3 עם ערך מרכזי של 5 וארבעה ערכי (-1). בדומה לסעיף ב', מסנן זה נבחר מכיוון שהוא מספק רמת חידוד מסוימת מבלי להכניס ניגוזיות או רעש מוגזמים, מה שהופך אותו לאידיאלי להדגשת פרטים. הוא מדגיש בעדינות קצotta על ידי הגדלת עצמת הפיקסל המרכזי ביחס לשכנים.

התמונה המקורית לאחר סיבוב חיטור וחידוד:



ההידוד הצליך להבליט את השכונה והמבנים בה ולהפוך אותם למעט ברורים יותר.
פעולה זאת השלימה את שיפור התמונה והפכה אותה לברורה, מוקדת ומעניינת Ⓢ.