

נושא:

תיקוני תמונה כתוצאה מתוארה לקויה והיתוך תמונות לתקן בעיות פוקוס חלק

מרצה:

אמיר הנדלמן

מגישיים:

רוסלאן אוסמנוב – 327480026

Ruslan31bar@gmail.com

Git : [LINK](#)

(ראה אופן שימוש עמ' 15) ([GUI DFE](#))

(ראה אופן שימוש עמ' 31) ([GUI DCT](#))

תוכן עניינים

Table of Contents

3	Double-function enhancement algorithm for low-illumination images based on Retinex theory
3	מבוא
3	Basic Theory
3	Retinex Theory
4	אלגוריתם
5	TV – denoise
6	Double-Function Image Enhancement
7	Three-Dimensional Gamma Correction
7	Saturation Regulation
8	תוצאות
14	הסבר על שימוש בGUID
15	Discrete Cosine Transform – based Image Fusion
15	מבוא
15	התמרת קוסינוס בדידה
16	התמרת קוסינוס בדידה רב חלוציה
20	היתוך
20	הערכת ביצועים
20	עם תמונה יhos
21	ללא תמונה יhos
22	תוצאות
30	הסבר על שימוש בGUID
32	Appendix A – Script
42	Appendix B – Script

Double-function enhancement algorithm for low-illumination images based on Retinex theory

מאמץ זה מציע אלגוריתם לשיפור תמונה באיכות גבוהה כדי לפחות את הבעיה של הגברת רעש ושיפור מוגזם כתוצאה מניגודיות נמוכה ותאורה לא אחידה בתחילת של שיפור תמונה בתאורה עם עצמה. מודל הוריאציה הכללת משתמש להשגת תונות ערוץ V ו-S מוחלקות, וטרנספורמציה גמא אדרטיבית משמשת לשיפור הפרטים בערך V של התמונה. לאחר מכן משתמשים באלגוריתם משופר של Retinex רב סולס לשגת תונות משופרות שונות של ערוצי V, ושתי התנות מתמזגות בהתאם למפלילות העצמה המוקנית של התנות. לבסוף, פונקציית הגמא הולמת מידית משמשת כדי לתקן את התמונה המוגזגת ולהתאים את רווחת התמונה.

מבוא

יש שתי הקטגוריות של אלגוריתמים לשיפור תמונה בתאורה נמוכה: אלגוריתמים מסורתיים ולמידה عمוקה. אלגוריתמים מסורתיים כוללים מיפוי, השוואת היסטוגרמָה, טרנספורמציה של Wavelets ותיאורית Retinex. שיטות המיפוי משתמשות בפונקציות לא ליניאריות לעיבוד גלובלי של תונות בתאורה נמוכה ומתאימות את משראת השיפור של אזורים שונים כדי למגוון עיוותים בתמונה. השוואת היסטוגרמָה יכולה להשיג את אפקט השיפור על ידי הרחבות הטווח הדינמי של התמונה. טרנספורמציה Wavelet יכולה לעבד גם את תחום התדר וגם את התחום המרחבי של התמונה בו-זמנית. תיאורית-h Retinex מחלוקת תמונה לרכיב אירוע ורכיב השתקפות ומסירה את רכיב האירוע מהתמונה המקורית כדי לקבל את רכיב החשתקפות. על ידי מודלים שונים של נתיבים, ניתן לחלק אותו ל Retinex מבוססת נתיב אקרטי, Retinex מבוססת ריבוי איטרציות ו Retinex מבוססת מרכז/הקיף.

מאמץ זה מציע אלגוריתם מסורתי לשיפור תמונה תאורה נמוכה, תוך שימוש במודל הוריאציה הכללת (TV – Total Variation) כדי לטשטש את תונות ערוץ ה-V ואת טרנספורמציה הגמא ההסתגלותית כדי לשפר את הפרטים של תונות שכבת המבנה המתקבל לאחר ביטול רעש. פונקציית הטנגנס החיפרבולית והפונקציה הלוגריתמית משולבות כדי לשנות באמצעותם המשופרת. לבסוף, הרווחת התמונה מודנית משמשת להתחמת התמונה המפורטת ולתיקון התמונה המשופרת. לבסוף, הרווחת התמונה מותאמת על ידי סוף אוטומטי כדי לקבל את האפקט הסופי.

Basic Theory

Retinex Theory

הרעון המרכזי של Retinex הוא שתמונה מורכבת מרכיב אירוע ורכיב משתקף. עצמת רכיב האירוע קובעת את הטווח הדינמי של פיקסל תמונה, בעוד הרכיב המשתקף הוא תכונה אינגרנטית של האובייקט ואינו משתנה עם רכיב האירוע. הדגש הוא כדלקמן:

$$I(x, y) = R(x, y) \cdot L(x, y)$$

כאשר (x, y) הוא תומנת התצפית, (y, x) הוא מרכיב החשתקפות ו- (y, x) L הוא מרכיב האירוע. בדרך כלל זה מומר לצורה לוגריתמית כדי להפחית את המורכבות החישובית:

$$\log R(x, y) = \log I(x, y) - \log L(x, y).$$

בהתבסס על תיאוריה זו, הוצע אלגוריתם SSR (single-scale retinex) קלאסי. פונקציית גרעין וגוש עובה קונבולוציה עם התמונה המקורית כדי להעניק את רכיב האירוע, ורכיב החשתקפות מתקבל כדי למש את השיפור של תונות בעלות תאורה נמוכה. הנוסחה היא כדלקמן:

$$\log R_i(x, y) = \log I_i(x, y) - \log(I_i(x, y) \otimes G(x, y))$$

כשאר גרעין גאוסיאני :

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

כאשר סטיטית תקן של גaus קטן, התמונה המשופרת שומרת יותר פרטים, אך היא עלולה לגרום בклות לעיוות צבע בתמונה. סטיטית תקן של גaus גדול, התמונה המשופרת שומרת על צבע טוב, אך התמונה הופכת למוטושתת, ואפקט ההילה מתרחש בклות באזור הבHIR יותר.

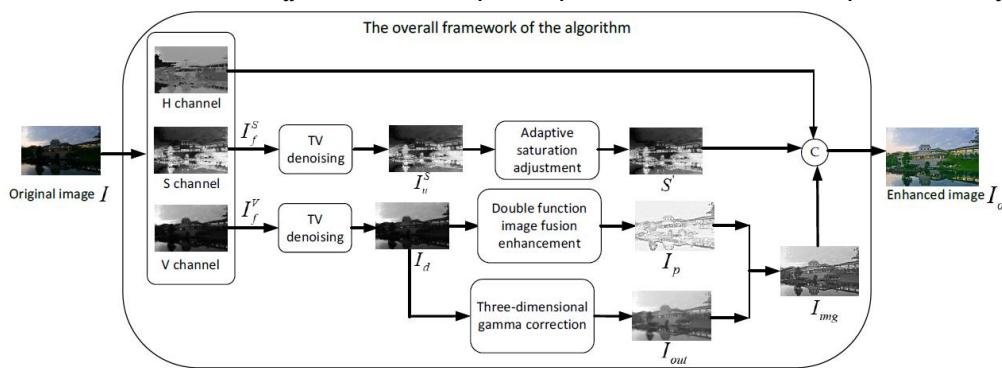
על מנת לפטור את הביעות הניל, הוצע אלגוריתם MSR(multi-scale retinex) לשיפור תמונות בהארה נמוכה בקנה מידה שונה, והביטוי שלו הוא כדלקמן :

$$\log R_i(x, y) = \sum_j^N \omega_j \{ \log I_i(x, y) - \log(I_i(x, y) \otimes G_j(x, y)) \}$$

כאשר $N=3$, המיצג את שלושת הדרגות, ω מיצג את המשקל של דרגת j , ובאופן כללי $\omega_1 = \omega_2 = \omega_3 = \frac{1}{3}$. G_j מיצג את פונקציית הגרעין הגaussית המתאימה עם סטיטית תקן שונה.

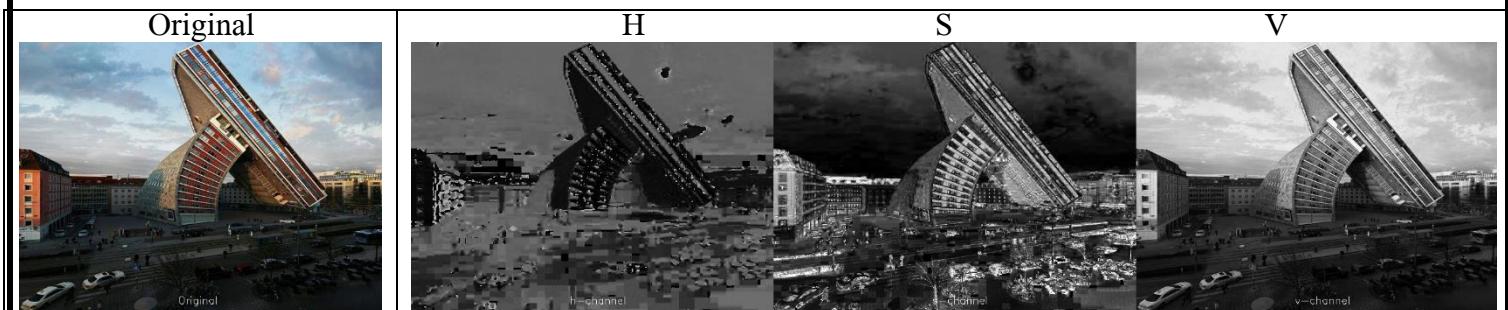
אלגוריתם

כדי להשיג ביצועי שיפור באיכות גבוהה ללא הגברת רעש במצב תאוריה נמוכה, אלגוריתם עיל המבוסס על תיאורית Retinex. תרשימים הכוללים לאלגוריתם שיפור התמונה מוצגים באירור 1. האלגוריתם המוצע מורכב מיחידת TV denoising, יחידת התאמת רויה אדפטיבית, יחידת שיפור היתוך תמונה כפולה ויחידה תלת ממדית לתיקון גמא. התמונה המקורי היא תמונה הקלט בפורמט RGB, אשר מונרת תחילת לפורמט HSV, ולאחר מכן מתקבלות תמונות ערוץ S וערוץ V. I_{img}^S ו I_{img}^V מתקבלות באמצעות שיטות עיבוד שונות. לבסוף, תמונות שלושת הערוצים שהתקבלו מיוצרות מחדש HSV וRGB. יש להחליק את תמונה ערוץ V על ידי denoising TV כדי לקבל את התמונה I_u^V , ואז הפרטים משופרים על ידי שיפור גמא אדפטיבי כדי לקבל את התמונה I_d . יש להחליק את תמונה S-channel רק כדי לקבל את התמונה I_u^S .



איור 1. תרשימים זרימה של אלגוריתם

נראה את תוצר אלגוריתם בכל אחד מהשלבים בעזרת תמונה הבאה:



TV – denoise

על פי תיאורית הרטינקס, לאחר הסרת רכיב האירוע בתמונה, הרכיב המשתקף המתקבל יכול להראות את הצבע האמיתי של האובייקט, כך שלאלגוריתמים הקשורים המבוססים על תיאורית הרטינקס יש יכולת שימור צבע טובה. בפועל, קשה להעיר במדוק את מרכיב האירוע בתמונה, מה שהוביל לליקויים ברכיב ההשתקפות המתקבל, להרווס את המתאם בין שניים ערוצי צבע, ולהוביל לעיוות צבע בתוצאה שיפור התמונה. כדי למנוע מצב זה, ניתן להעביר את התמונה מרחב צבע HSV. באמצעות ניטוח תחילה הרעש בתהילך שיפור התמונה, יש לסנן טרנספורמציה מרכז בערך S ו- V. לכן, כדי להפחית את הגברת הרעש בתהילך שיפור התמונה, יש לסנן תחילתה את ערוצי התמונה הנ"ל. מבין שיטות הגינון המבוססות על משוואות דיפרנציאליות חילקוות, מודל TV- denoise הוכח כאחד האלגוריתמים היעילים ביותר. באמצעות מודל זה, ניתן להפוך את בעיית הניפוי התמונה לבעית אופטימיציה כדי לפתור את פונקציית האנרגיה המינימלית, והבטיו מוצג כדלקמן:

$$\min_{I_u^c} E(I_u^c) = \min_{I_u^c} \iint_{\Omega} |\nabla I_u^c| dx dy + \frac{\lambda}{2} \iint_{\Omega} (I_u^c - I_f^c)^2 dx dy$$

ובצורה דיפרנציאלית :

$$|\nabla I_u^c| = \sqrt{I_{u_x}^c{}^2 + I_{u_y}^c{}^2}; \quad \nabla = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y} \right) \quad I_{u(i,j)}^{c(n+1)} = I_{u(i,j)}^{c(n)} - \Delta t \lambda (I_{u(i,j)}^{c(n)} - I_{f(i,j)}^c) + \Delta t \left(\nabla \cdot \left(\frac{\nabla I_{u(i,j)}^{c(n)}}{|\nabla I_{u(i,j)}^{c(n)}|} \right) \right)$$

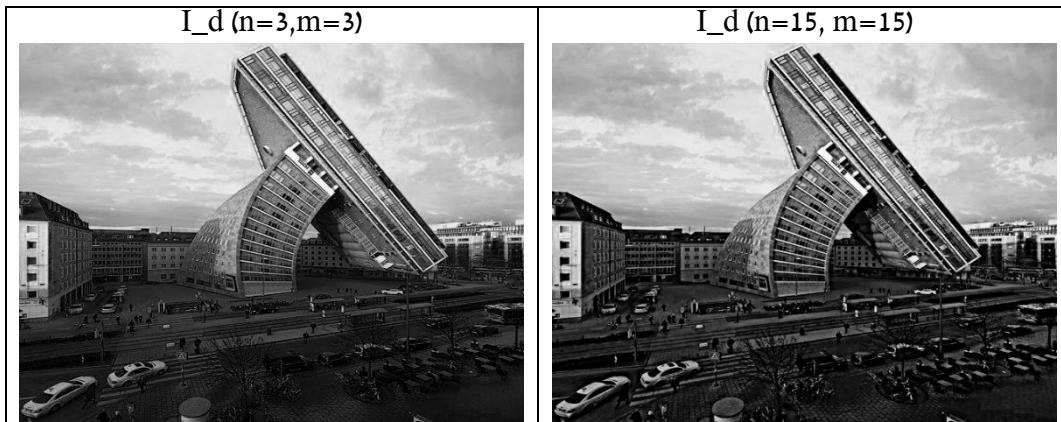
כאשר, $\lambda=100$ מייצג את מספר האיטרציות, ו- $\Delta t=0.25$ מייצג את קפיצה בזמן. λ – הינו פרמטר רגולרייזציה, ככל שערך קטן יותר כך התוצאות גדול יותר.

	V – channel after TV (I_u_V)	S – channel after TV (I_u_S)
$\lambda = 90$		
$\lambda = 0.05$		

למרות שרוב מידע התמונה נשמר בתהליך-TV-denoising, חלק מפרטי התמונה גם אובדים. לכן, יש לשפר את פרטי התמונה, המידע האפור והשונות של התמונה משמשים כדי להגדיר את פרמטרי טרנספורמציה של גמא כדי להציג את פרטי התמונה. הנוסחה היא כדלקמן:

$$b(x,y) = \frac{\sum_{m,n} (I_u^V(x,y) - N(x,y))^2}{m \cdot n}, \quad N(x,y) = \frac{\sum_{m,n} I_u^V(x,y)}{m \cdot n}, \quad I_d = I_u^V(x,y) \left(\frac{N(x,y) + b(x,y)}{I_u^V(x,y)} \right)$$

כאשר, כאשר N ו- b הם הרץ האפור הממוצע והשונות של האзор המקומי בגודל (m , n) של נקודת הפיקסל שבמרכזו (x,y) .



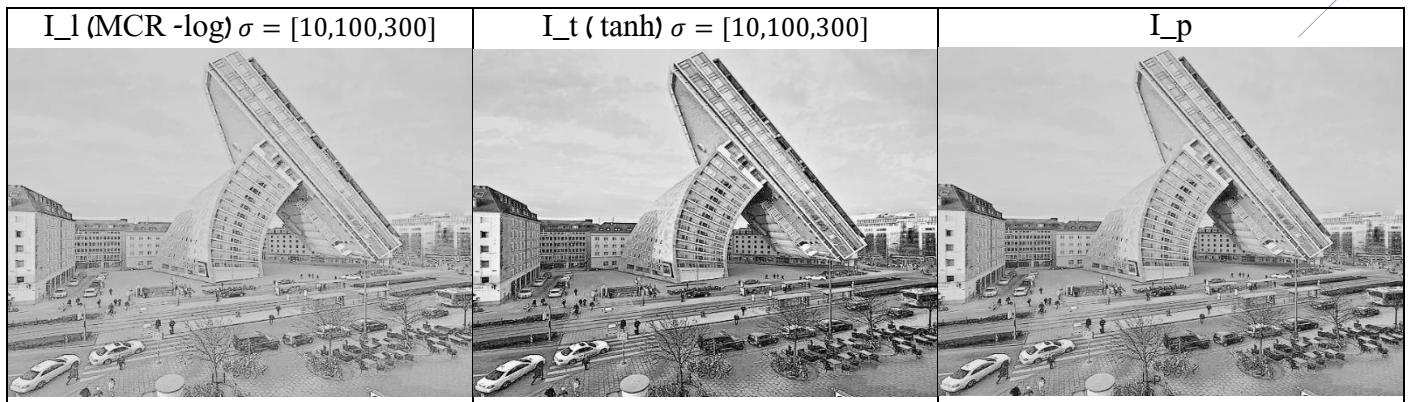
Double-Function Image Enhancement

כפי שהזכר לעיל, האלגוריתם המסורתי של MSR משתמש בפונקציית הגרעין של גאות המדגישה מידע מרחבי כדי להעיר את תומנת האירוע ומתעלם ממידע הקצוות של התמונה, מה שmobiel להתרחשות של הלילה בתמונה המשופרת במקור האור. ניתן לפרט בעיה זו ביעילות על ידי שימוש בסיכון מודרך שקובע את שניים כדי להעיר את תומנת האירוע. ניתן למשש את השיפור של האזור הכהה וכןן לעצב שיפור מוגדם של האזור הבבורי, אפקט השיפור בין שתי המועלות אינו ברור; אפקט השיפור של כל התמונה עזום, בעוד שהפונקציה הלוגריתמית משתנה מאוד לאחר טרנספורמציה הפוכה. לכן, מאמר זה מנצל את המאפיינים של שניים. פרמטרי ההיתוך מתוכננים בהתאם לבחרות האזור המקומי של התמונה כדי להשיג איזון בין שתי התמונות. הנוסחה מוצגת להלן:

$$I_p = \alpha I_l + \beta I_t \quad \alpha = \frac{\sum_{m,n} I_t}{\sum_{m,n} I_l}, \quad I_t = \sum_j^N \omega_j \left\{ \tanh \left(\frac{I_d(x,y)}{I_d(x,y) \otimes G_j(x,y)} \right) \right\}, \\ \alpha + \beta = 1$$

כאשר, I_l הינו שיפור בעזרת אלגוריתם MSR, I_t – הינו מושג שיפור של טנגנס היפרבולי.
 $G_j(x,y)$ – הינו גרעין טשטוש עם סטיית תקן σ שונה.

لتמונה המתקבלת על ידי הפונקציה הלוגריתמית יש בהירות כללית גבוהה יותר, ואובדן רב יותר של פרטיים מקומיים, בעוד שלתמונה המתקבלת על ידי פונקציית הטנגנס היפרבולית יש בהירות כללית נמוכה יותר, ניגודיות נמוכה ומוטשטשת. התמונה המתקבלת על ידי שילוב שתי הפונקציות יכולה לדכא שיפור מוגדם.

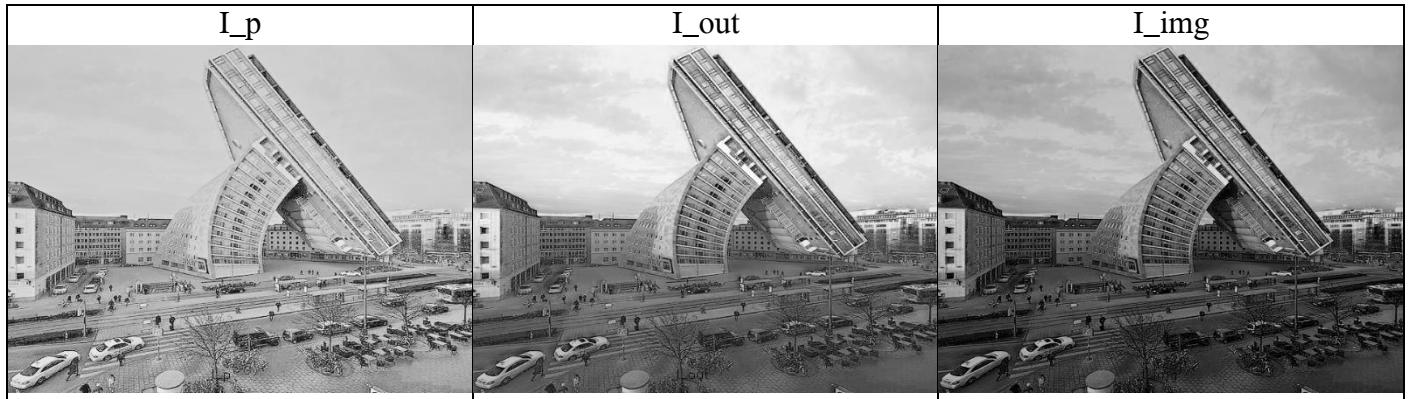


Three-Dimensional Gamma Correction

על מנת לפתור את בעיית "ערפל לבן" לעיל, סעיף זה משתמש בהמרת גמא כדי להתאים את התמונה המפוארת ולאחר מכן, על בסיס זה, לתקן את התמונה המשופרת. בהתחשב בבהירות ובמידע המפורט של התמונה, ערכי פרמטר הגמא נקבעים משלושה מימדים, כמוור קנה המיידה האפור, השונות והערך גרדיאנט של התמונה. הנוסחה היא כדלקמן:

$$\begin{aligned}
 C - \text{היינו הוויריאנס שמחושב} \\
 \text{אותו דבר כמו}: & \quad B(x, y) = \frac{\sum_{m,n} |\nabla I_d(x, y)|}{m \cdot n}, \quad I_{\text{out}} = I_d^{(\psi \cdot \exp(A) + \mu \cdot \exp(B) + \nu \cdot \exp(C))} \\
 b(x, y) = & \frac{\sum_{m,n} (I_u^V(x, y) - N(x, y))^2}{m \cdot n}, \quad A(x, y) = \frac{I_d(x, y)}{\max_{n \times m} I_d(x, y)}
 \end{aligned}$$

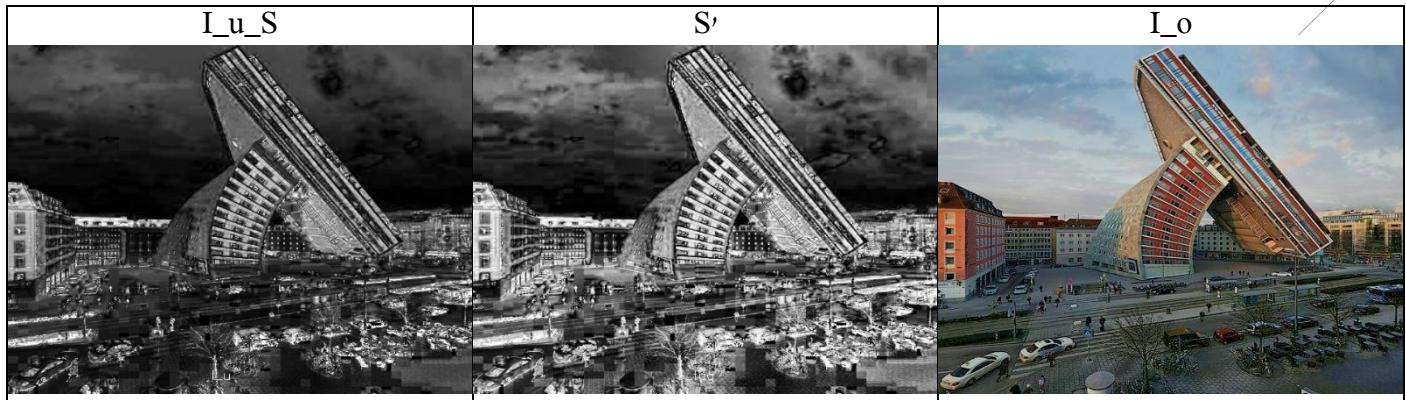
הניגודיות של התמונה המתוקנת משופרת משמעותית ובהירות הכוללת של התמונה מתואמת.



Saturation Regulation

בתמונה בעלת עצמת האירה נמוכה, רווית התמונה באזורי הכהה נמוכה. הפיכה ישירה של תמונות הערוץ המעובדות לתמונה RGB תוביל לTOPFADE של "נקודה כהה" באזורי הכהה של התמונה. ניתן להקל על TOPFADE זו על ידי התאמת ערז הרויה של התמונה, והנוסחה מוצגת כדלקמן:

$$\text{כאשר } S_m \text{ הוא הערך הממוצע I באזור (m,n) שברeczy נקודה (i,j), } S_g \text{ הוא גרדיאנט של I בנק' (i,j)} \quad S' = \begin{cases} 1 + 0.8 \cdot \log \left(\frac{S_m}{I_u^S(i,j) + 0.5 \cdot S_g(i,j)} \right), & I_u^S(i, j) \leq \vartheta \\ \exp \left(\frac{S_m - I_u^S(i,j)}{2} \right), & \text{else,} \end{cases}$$

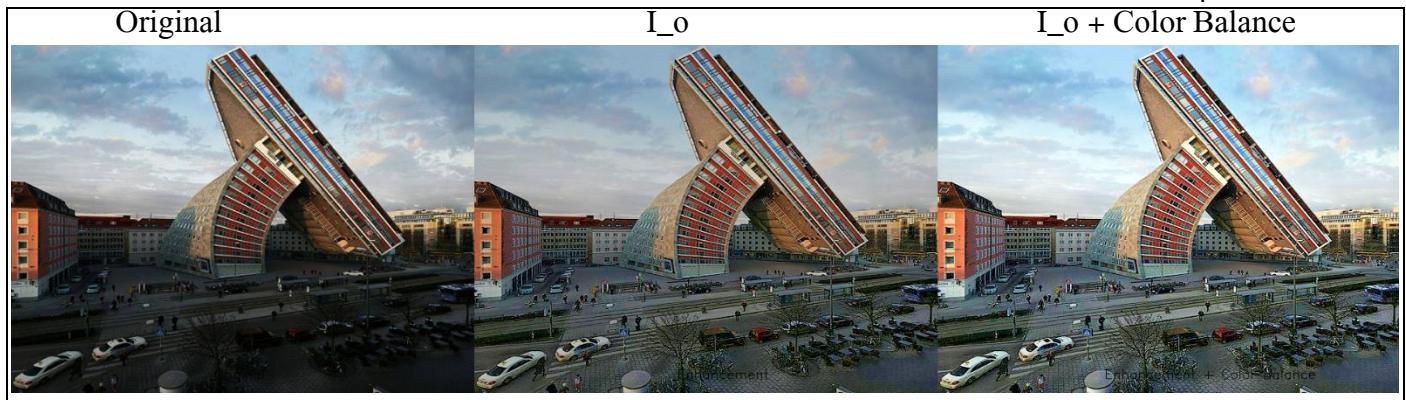


כדי לאמת את יעילות האלגוריתם, בחרנו כמה תמונות לניסויי השיפור; גודל התמונה הוא 480x640, אלגוריתמים להשוואה:

MSRCR, CLAHE, AHE (Adaptive Histogram Equalization), DFE (Proposed)
בנוספַף לモצא של אלגוריתם DFE בוצע מתייחס ניגודיות התמונה באמצעות איזון הצבע הפשטוט
ב尤טור עם אחוז חיתוך של 1% בשני הקצוטות. תיאור של אלגוריתם הינו במאמר הבא:

[Simplest Color Balance](#)

כתוצאה מתיקבלת:

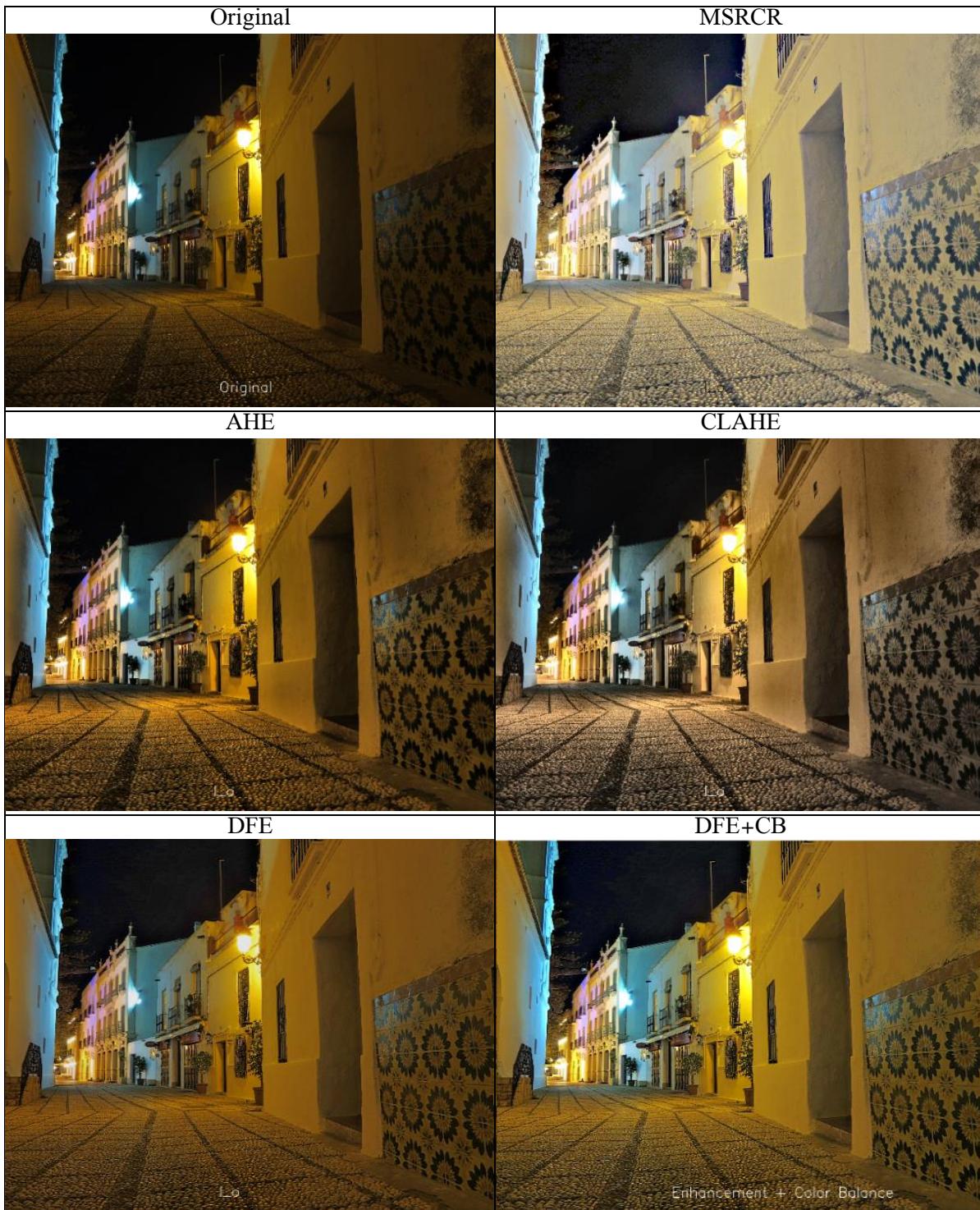


תוצאות

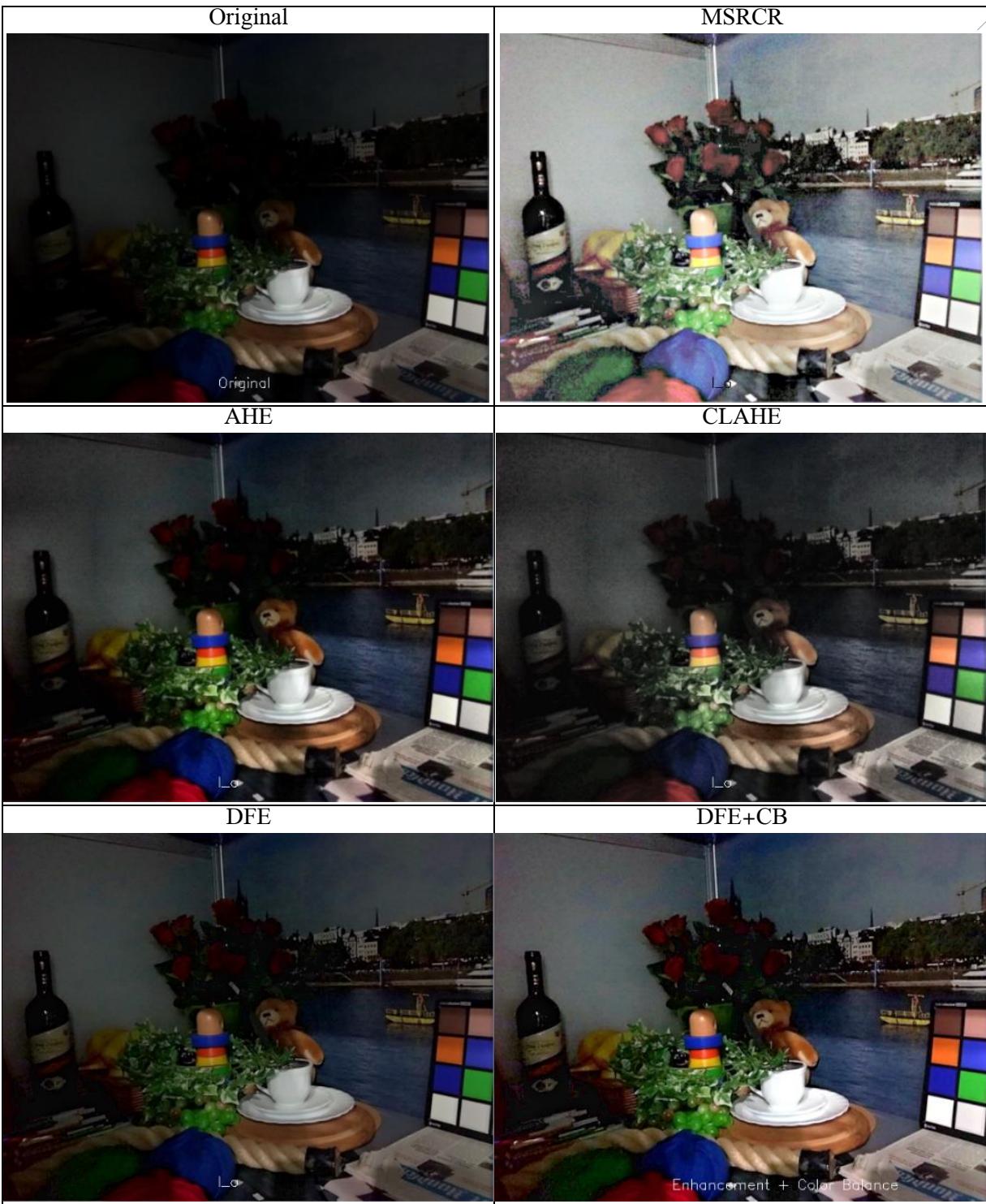
מטריות לבחינה הימנו:

יחס אות לרעש מקסימי (PSNR) משמש למדידת הרעש של התמונה המשופרת, דמיון מבני (SSIM) משמש למדידת הדמיון המבני בין התמונה המשופרת לתמונה המקורי כדי לשפט את העיינות של התמונה. סטיית תקן (SD) משמשת כדי לשקוף את מידת הפיזור של פיקסלים ביחס לממוצע לאפין ניגודיות התמונה. אנטרופיה המידע (E) משמשת כדי לשקוף את עוצמת המידע בתמונה ולאפין את מורכבות התמונה.

- כל שהערכים של SD, SSIM, PSNR ו-Eו גבוהים יותר, כך אפקט השיפור/תיקון טוב יותר.



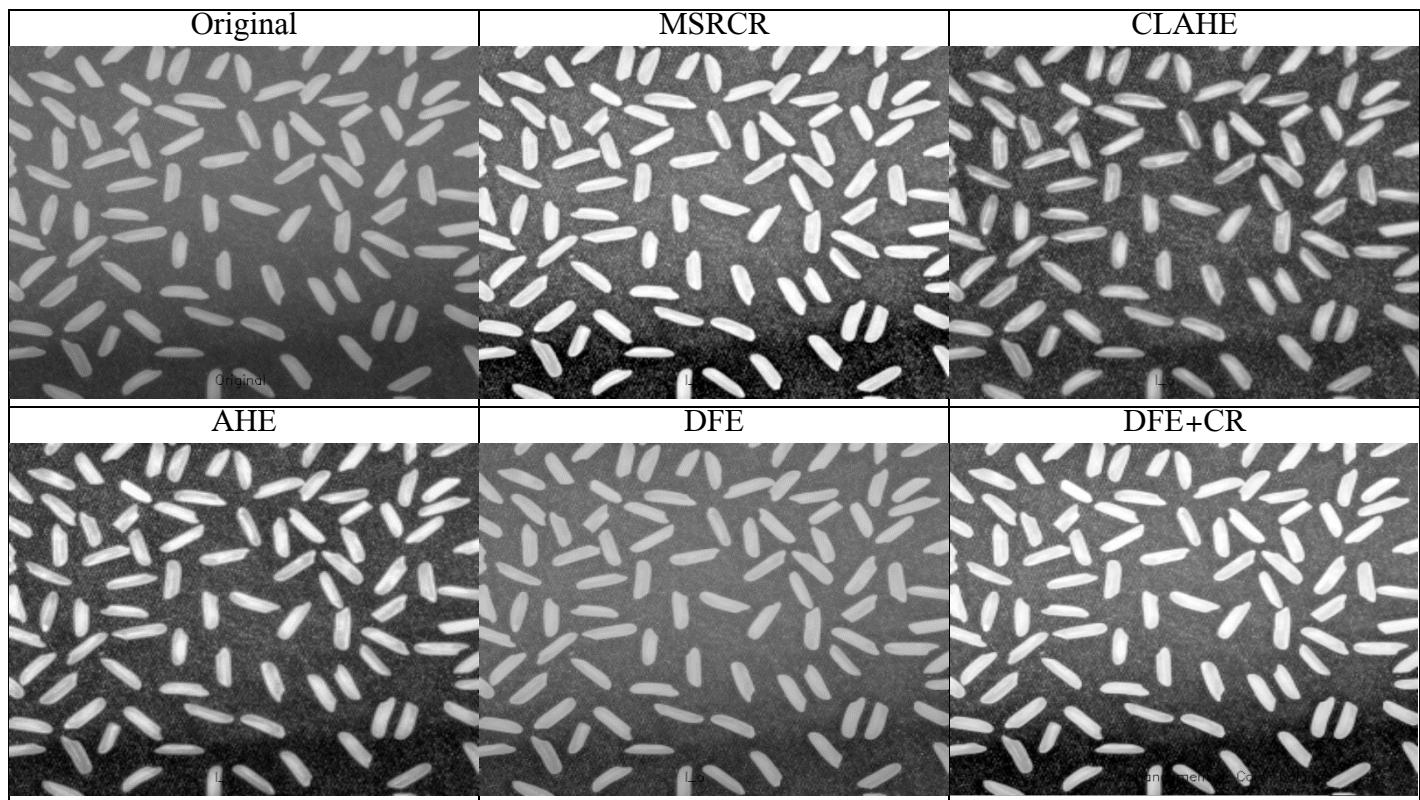
Model	PSNR	SSIM	SD	IE
MSRCR	8.8678	0.4688	55.113	7.5658
CLAHE	15.5905	0.6529	54.7259	7.5135
AHE	20.2313	0.8143	46.4206	7.3091
(Proposed) DFE	21.3653	0.7577	40.0822	7.1468



Model	PSNR	SSIM	SD	IE
MSRCR	6.7682	0.2338	58.4677	7.6574
CLAHE	18.3605	0.6047	45.4709	6.7732
AHE	19.8661	0.7196	44.1037	6.8306
(Proposed) DFE	21.6438	0.6561	33.7318	6.6549



Model	PSNR	SSIM	SD	IE
MSRCR	9.0612	0.4342	60.9012	7.7085
CLAHE	18.7062	0.6463	71.4885	7.4609
AHE	21.3625	0.7561	72.2199	7.4578
(Proposed) DFE	19.0163	0.6382	56.2955	7.3439

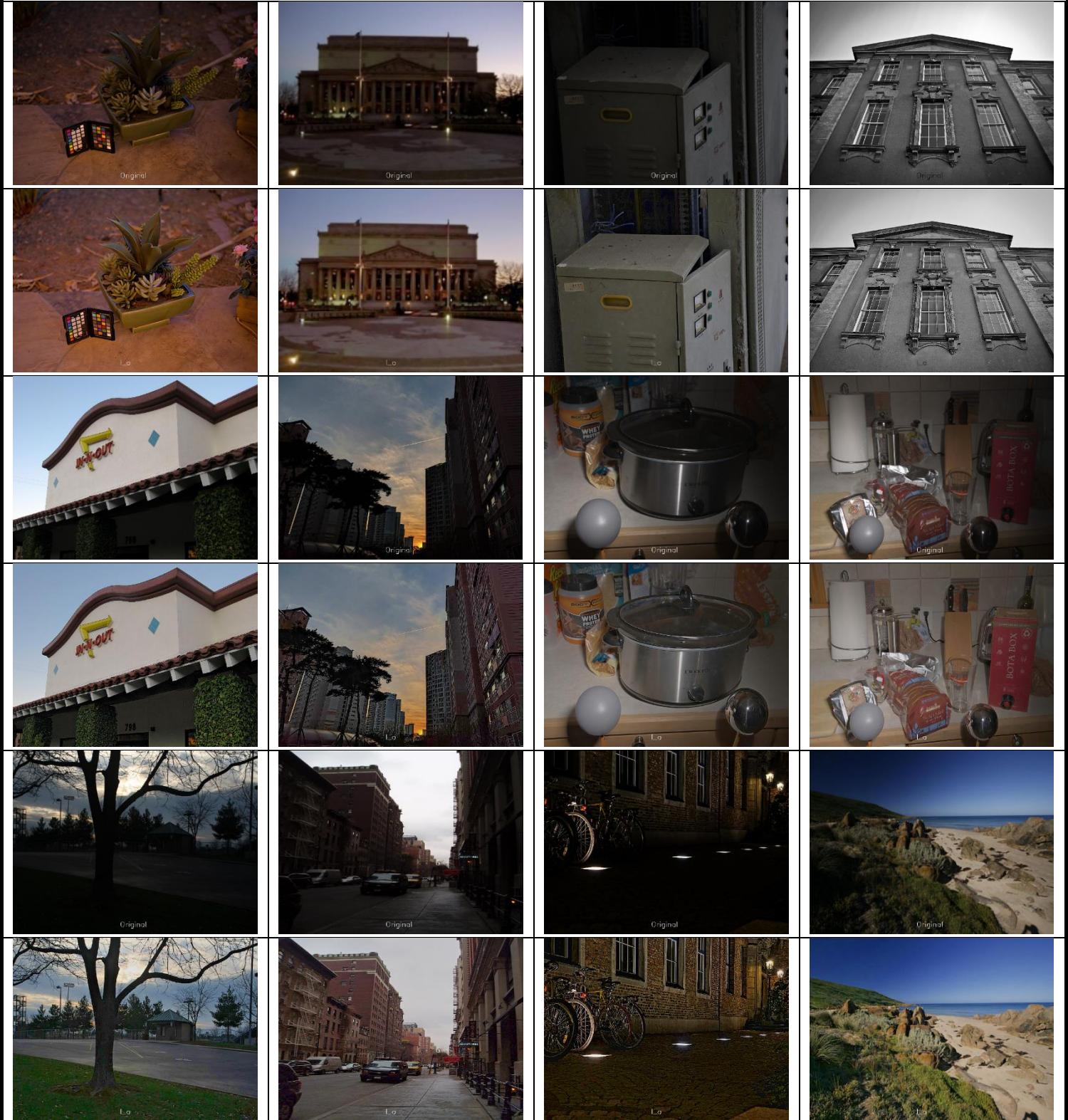


מסקנות:

מאמר זה הציע אלגוריתםיעיל לשיפור תמונה המשוגל להתגבר על הבעיות של הגברת רעש ושיפור מוגזם בឋאליך של שיפור תמונה בתאורה נמוכה. עבודה זו מאמצת שיטת TV-Denoise ליעיבוד מקדים של תמונה, והטרנספורמציה הגמא האדפטיבית משמשת לשיפור פרטיה בתמונה. פונקציית tanh ופונקציית הלוגריתם (MSR) מושולבות כדי להשיג את התמונה המשופרת המקדימה כדי למנוע שיפור תמונה מוגזם. טרנספורמציה גמא תלת ממדית משמשת כדי להתאים ולתקן את התמונה המשופרת. מהתווצאה הסובייקטיבית, נמצא שהתמונה המשופרת בוצרה יعلاה.

בנוסף לאלגוריתם עובד פחות טוב על תמונה עם ערך אפור בלבד מאשר תמונה בעלי שלושה ערוצים.

עוד תמונות •



הסבר על שימוש בGUI

GUI רשום בשפת פיתון

את GUI ניתן להפעיל בעזרת שני דרכי:

.1

בעזרת שימוש בקובץ Double-function enhancement algorithm.py יש לפתח את קובץ זה בסביבת עבודה של PyCharm או Visual Studio Code

יש לתקן ספריות הבאות (לעומץ פקודות):

```
pip install tk
pip install numpy
pip install matplotlib
pip install scipy
pip install opencv-python
pip install scikit-image
```

.2

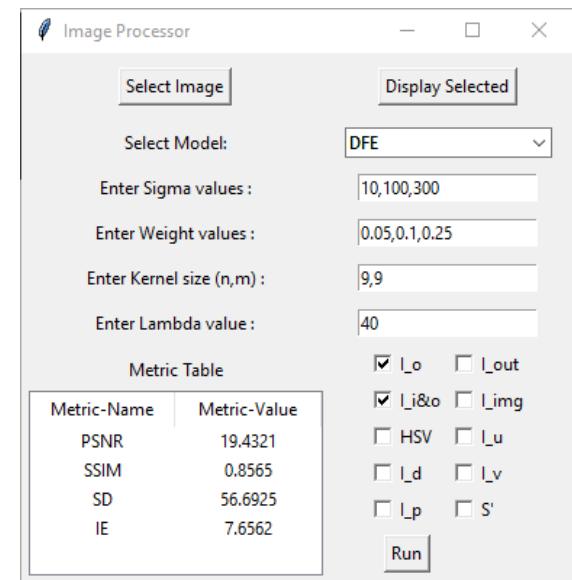
בעזרת הפעלת קובץ --Double Click Me To Run The App --.bat

* ראשית יש להוריד את תיקיה למחשב ולבצע Unzip

** לא נדרש להתקן ספריות

שימוש:

- – Select Images
- – Display Selected
- ** כדי להמשיך וללחוץ על כפתורים אחרים בחלון דרכים (1) לחיצה חלון הפתוח, סגירתו חלון אפשרית בשלושה דרכים (2) עזרת עכבר בלחיצה על X (3) בעזרת לחיצה על מקש ימין של עכבר כאשר סמן נמצא באזורי החלון פתוח.
- Select model – בחירת אלגוריתם שיפור תמונה, לכל אלגוריתם ניתן אופציונות עבור פרמטרים שונים.
- TV – Denoiser ל Lambda
- 3 dim gamma correction - פרמטרים ל Wights
- - פרמטרים ל גרעין טשטוש גאוסיאני Sigma
- – כפתור הרצה.



- ** כדי להמשיך וללחוץ על כפתורים אחרים קודם יש לסגור את חלון הפתוח, סגירתו חלון אפשרית בשלושה דרכים (1) לחיצת מקש (2) עזרת עכבר בלחיצה על X (3) בעזרת לחיצה על מקש ימין של עכבר כאשר סמן נמצא באזורי החלון פתוח.

• הקוד נמצא ב Appendix A - Script Git

Discrete Cosine Transform – based Image Fusion

היתוך תמונה באמצעות טכניקת המרת קוסינוס בדידה (MDCT) מרובת רזולוציות נבנה ונבדק. הביצועים של גישה זו מושווים לאלו של טכניקת היתוך תמונה ידועה המבוססת על wavelets. ביצוע היתוך תמונה MDCT זהה בערך ל-wavelets. זה מאד פשוט מבחינה חישובית ועשי להתאים ליישומים בזמן אמת.

מבוא

היתוך תמונה מרובה-חישנים (MIF) היא טכניקה להגדלת הרזולוציה המרחבית של תמונות מרובי-חישנים עם פרטים נמוכים תוך שמירה על המידע הספקטרלי שליהן על-ידי שילוב של שתי תמונות או יותר. MIF הופיע לאחרונה כנשא מחקר חדש וمبתייח לעיבוד תמונות. צבא, חישה מרוחק, ראיית מכונה, רובוטיקה, מעקב, מרכיבים וריאיה משופרות והדמיה רפואיים הם חלק מהענפים שנחנים מ-MIF.

MIF מנסה להתמודד עם האתגר של מיזוג תוכן המידע של תמונות רבות (או שנאספו מישיות חיישן הדמיה שונות) שצולמו מאותה סצנה על מנת להשיג תמונה משולבת המשלבת את המידע הטוב ביותר מהתמונות המקור השונות.

כתוצאה לכך, התמונה הממזוגת תהיה עדיפה על כל אחת מתמונות המקור. MIF יכול להתבצע בשלוש רמות שונות, בהתאם לשלב המיזוג: רמת פיקסל, רמת תכונה ורמות החלטה. במחקר זה, מຕאר MIF מבוסס רמות פיקסל, המתאר תהליך היוצר תמונה ממוגנת אחת עם תיאור מדויק יותר מתמונות המקור הבודדות.

ה-MIF הפשט ביותר הוא לऋת את הממוצע של פיקסלים של תמונות המקור ברמת האפור. שיטה זו תגרום לתוצאות לא רצויות שונות וניגודיות נמוכה יותר בתמונה הממזוגת. טרנספורמציות (התמרת) רבות כגון wavelet, פירמידות, תדר מרחבי, עיבוד אוטומטי סטטיסטי ותאורית הקבוצות המתווששות הוצעו כדי לטפל בבעיות אלו. התמרת wavelet מרובות רזולוציות עשויה להיות שימושית עבור לוקלייזציה הן בתחום המרחבי והן בתחום התדר. טרנספורמציה(התמרת) wavelet תציג מידע כיווני ברמות פירוק ומידע ייחודי ברזולוציות שונות.

טרנספורמציה(התמרת) של קוסינוס בדיד רב-רזולוציה (MDCT) משמשת במחקר זה כדי למוג את תמונות המקור בעלות פוקוס חלק. אחד הקריטריונים הבסיסיים ליישום טכניקות היתוך על תמונות הוא רישום תמונה, כמו גם המידע בתמונות המקור חייב להיות מיושר ורשום כראוי לפני היתוך.

התמרת קוסינוס בדידה

התמרת קוסינוס בדידה (DCT) היא טרנספורמציה חשובה בעיבוד תמונה. מקדמי DCT גדולים מרכזים באזור התדר נמוך; לפיכך, ידוע כבעל תכונות דחיסה אנרגטית מצוינות. התמרת הקוסינוס הבודד D₁ הופך את (n, k) ל- $X(k)$ באורך N.

$$X(0) = \sqrt{\frac{1}{N}} \sum_{n=0}^{N-1} x(n), \quad k=0$$

המשוואה (1) הופכת למקרה

הטרנספורמציה הראשון שהוא הממוצע של כל הדגימות ברכף והוא ידוע כמקדם DC, ומקדמי AC.

טרנספורמציה אחריהם ידועים כמקדמי AC.

טרנספורמציה הקוסינוס ההיפוכה מוגדרת כ:

$$x(n) = \sum_{k=0}^{N-1} \alpha(k) X(k) \cos\left(\frac{\pi(2n+1)k}{2N}\right), \quad 0 \leq n \leq N-1 \quad (3)$$

$$X(k) = \alpha(k) \sum_{n=0}^{N-1} x(n) \cos\left(\frac{\pi(2n+1)k}{2N}\right), \quad 0 \leq k \leq N-1 \quad (1)$$

$$\text{where } \alpha(k) = \begin{cases} \sqrt{\frac{1}{N}} & k = 0 \\ \sqrt{\frac{2}{N}} & k \neq 0 \end{cases} \quad (2)$$

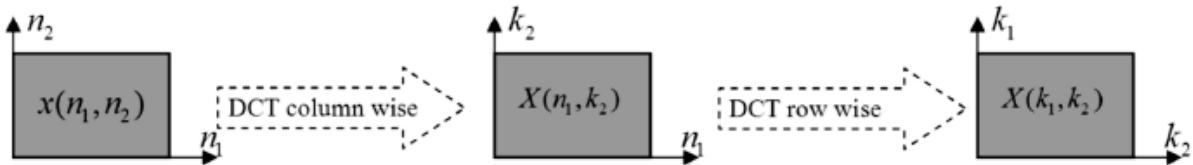
כאשר $\cos\left(\frac{\pi(2n+1)k}{2N}\right)$ הינה פונק' גרעין אמיתי וDISKRETIT.

ה-DCT 2 הוא הרחבה ישירה של 1D DCT. המרת הקוסינוס הבודד הדו-ממדי $X(k_1, k_2)$ של תמונה מוגדר כ:

$$x(n_1, n_2) = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \alpha(k_1) \alpha(k_2) X(k_1, k_2) \cos\left(\frac{\pi(2n_1+1)k_1}{2N_1}\right) \cos\left(\frac{\pi(2n_2+1)k_2}{2N_2}\right), \quad 0 \leq n_1 \leq N_1-1, \quad 0 \leq n_2 \leq N_2-1 \quad (5)$$

$$\text{והתמרה הפוכה: } X(k_1, k_2) = \alpha(k_1) \alpha(k_2) \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) \cos\left(\frac{\pi(2n_1+1)k_1}{2N_1}\right) \cos\left(\frac{\pi(2n_2+1)k_2}{2N_2}\right), \quad 0 \leq k_1 \leq N_1-1, \quad 0 \leq k_2 \leq N_2-1 \quad (4)$$

וגם IDCT זה טרנספורמציות אשר ניתנות להפרדה, היתרונו של תכונה זו היא חישוב 2D DCT או IDCT ניתן לבצע שני שלבים על ידי 1D IDCT או 1D DCT על עמודות ולאחר מכן על שורות של תמונה (n1,n2). כפי שמצוג באיור 1.



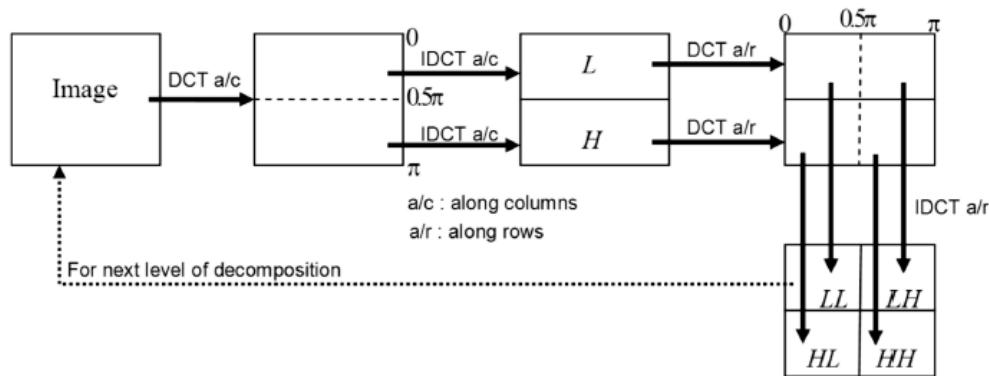
איור מס 1 ביצוע התמרת קוסינוסים בדידה דו ממדית בעזרת תכונת ההפרדה

התמרת קוסינוס בדידה רב רזולוציה

טרנספורמציה קוסינוס מרובת רזולוציה דיסקרטית (MDCT) היא טכניקת עיבוד אותן הדומה להתמורות wavelets אשר מסנן את האות בנפרד על ידי מסני תגובה תדר סופית (FIR) במעבר נמוך ובמעבר גבוה, ואז הפלט של כל מסנן מופחת (הורדת צב דגימה) בפקטור של שניים כדי להשיג את רמת הפירוק הראשונה. כדי להשיג את רמת הפירוק השנייה. ניתן לחזור על תהליך זה כדי להשיג רמות אחרות של פירוק.

ה-MDCT מחליף את מסני FIR ב-(DCT, התמונה. כדי לבצע DCT בעבורת פירוק בתחום התדר על ידי יישום DCT באופן עצמאי. ממחצית הנקודות הראשונות (0 עד $\pi/2$) נתונות ל-IDCT לקבالت את התמונה שעברה דרך מסנן נומוכים L, וממחצית השניה ($\pi/2$ עד π) נתונה ל-IDCT לקבالت את התמונה שעברה דרך מסנן גבוהים H. התמונה שעברה דרך מסנן נומוכים L עבורה טרנספורמציה לתחום התדר על ידי יישום DCT בזוירה של שורה. ממחצית הנקודות הראשונות (באופן השורה) נתונות ל-IDCT כדי לקבל את התמונה LL שהועברה דרך מסנן נומוכים, וממחצית נקודות הנותרות נתונות ל-IDCT לקבالت את התמונה המועברת דרך מסנן גבוהים LH. התמונה שעברה דרך מסנן גבוהים H עבורה טרנספורמציה לתחום התדר על ידי יישום DCT באופן שורה. ממחצית הנקודות הראשונות (באופן השורה) נתונות ל-IDCT ההפוך כדי לקבל את התמונה HL שהועברה דרך מסנן נומוכים, וממחצית נקודות הנותרות נתונות ל-IDCT לקבالت את התמונה המועברת דרך מסנן גבוהים HH.

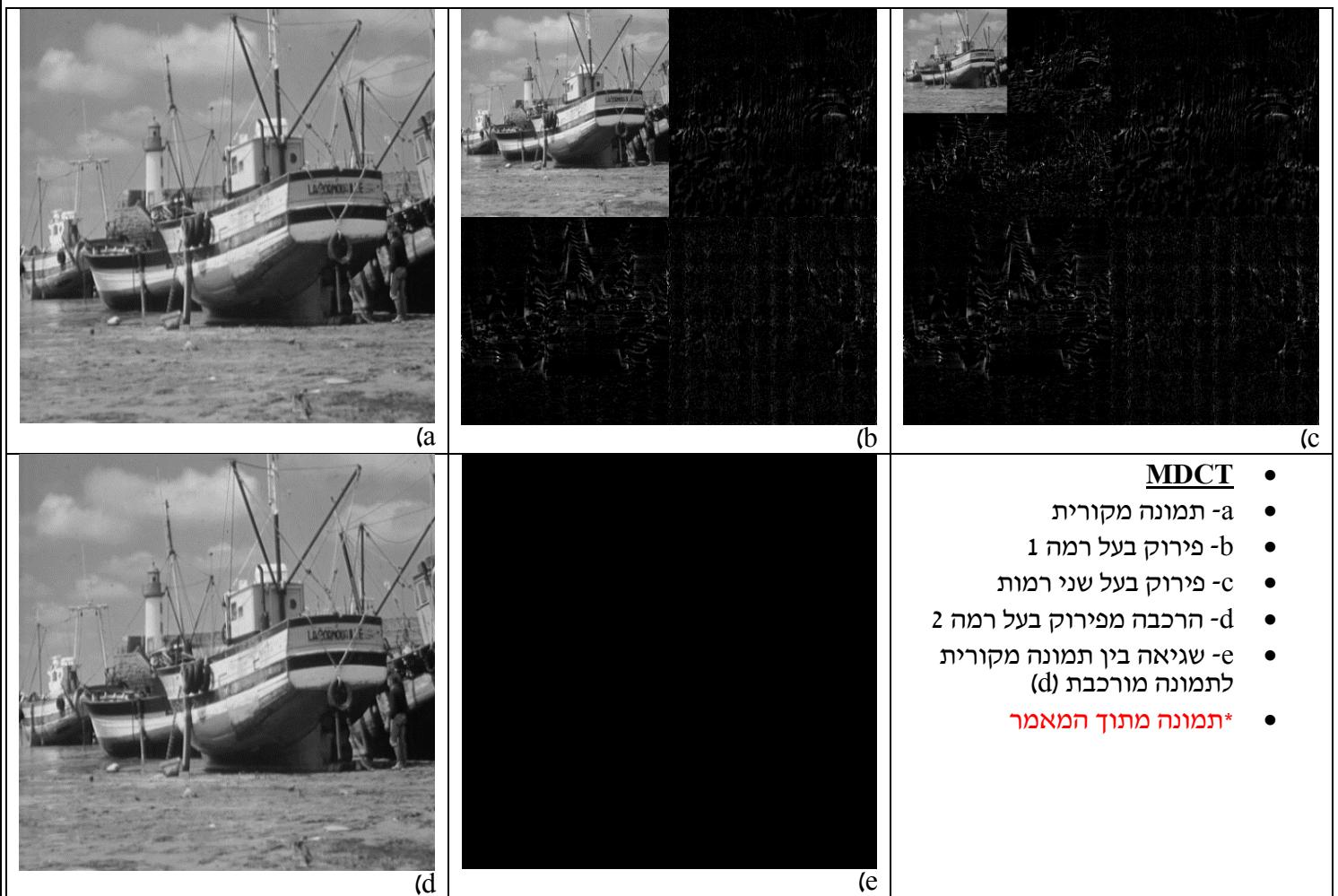
ה-LL מכיל את מידע התמונה הממוצע התואם לפס התדרים הנומוכים של הפירוק הרב-סקאלי. הוא מייצג את הגרסה המוחלקת ותת הדגימה של תמונה המקור ויכול להיחשך כקירוב של תמונה המקור(Approximations). התמונות LH, HL ו-HH הן תת-תמונות מפורחות המכילות מידע ציוני (אופקי, אנכי ואלכסוני) של תמונה המקור עקב כיוון מרחביים(Details). ניתן להשיג ריבוי רזולוציה על ידי יישום רקורסיבי של אותו אלגוריתם על מקדמי המעבר הנומוכים (LL) מהפירוק הקודם. את אלגוריתם ניתן לראות באיור 2.



איור מס 2 פירוק רב רזולוציה בעזרת התמרת קוסינוסים בדידה DCT

כעת אנו נציג את פירוק רב שלבי של התמונות (מתוך המאמר ומה שצולם בנוסך) בעזרת DCT וWavelet.

MDCT (Multilevel Discrete Cosine Transform)

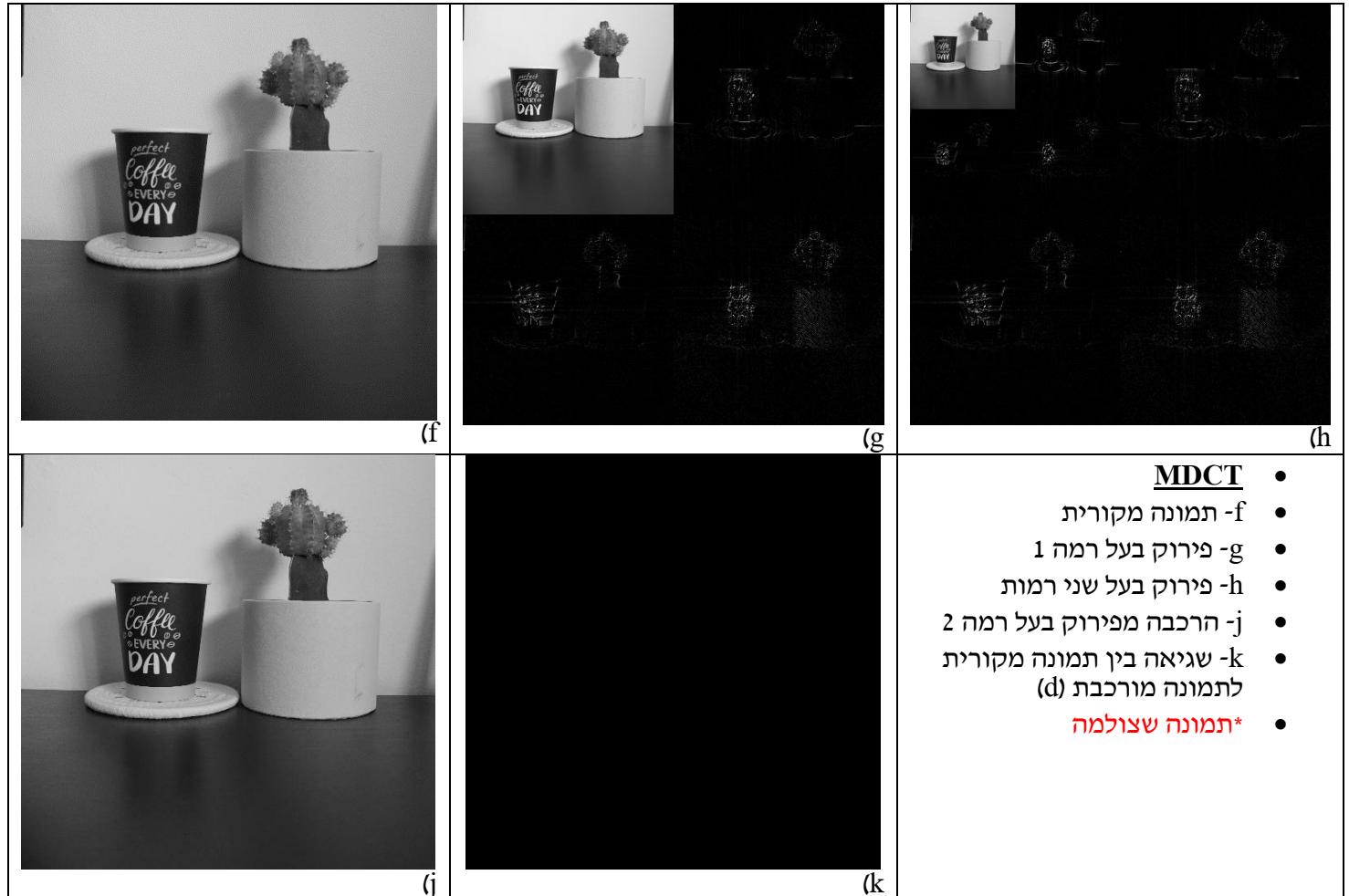


MDCT

- תמונה מקורית
- פירוק בעל רמה 1
- פירוק בעל שני רמות
- הרכבה מפירוק בעל רמה 2
- שגיאה בין תמונה מקורית לתמונה מורכבת (d)

*תמונה מתוך המאמר

-
-
-
-
-
-
-
-
-

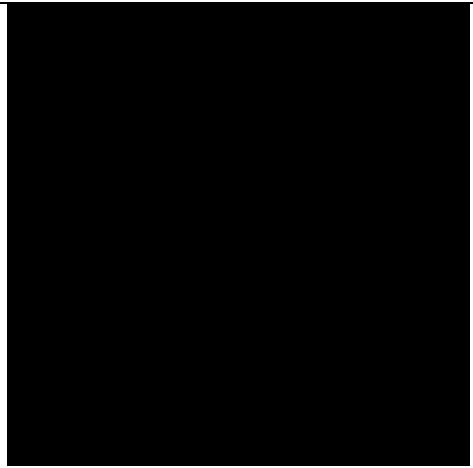


MDWT (Multilevel Discrete Wavelet Transform)



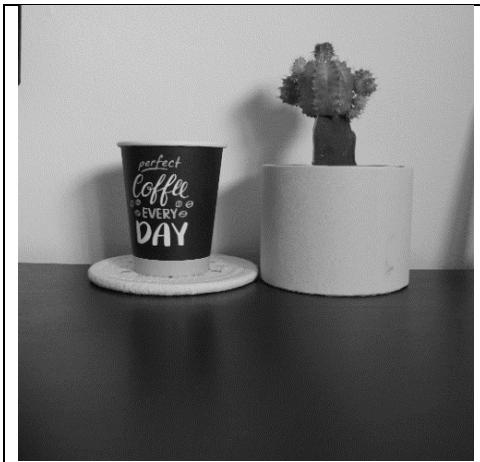


(d)



(e)

- MDWT**
- a- תמונה מקורית
 - b- פירוק בעל רמה 1
 - c- פירוק בעל שני רמות
 - d- הרכבה מפירוק בעל רמה 2
 - e- שגיאה בין תמונה מקורית לתמונה מורכבת (d)



(f)



(g)

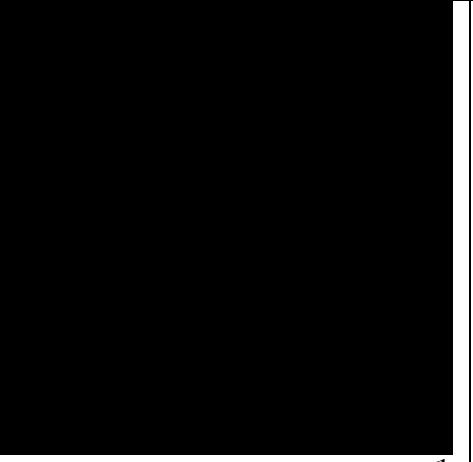


(h)

- MDWT**
- f- תמונה מקורית
 - g- פירוק בעל רמה 1
 - h- פירוק בעל שני רמות
 - i- הרכבה מפירוק בעל רמה 2
 - j- שגיאה בין תמונה מקורית לתמונה מורכבת (d)



(j)

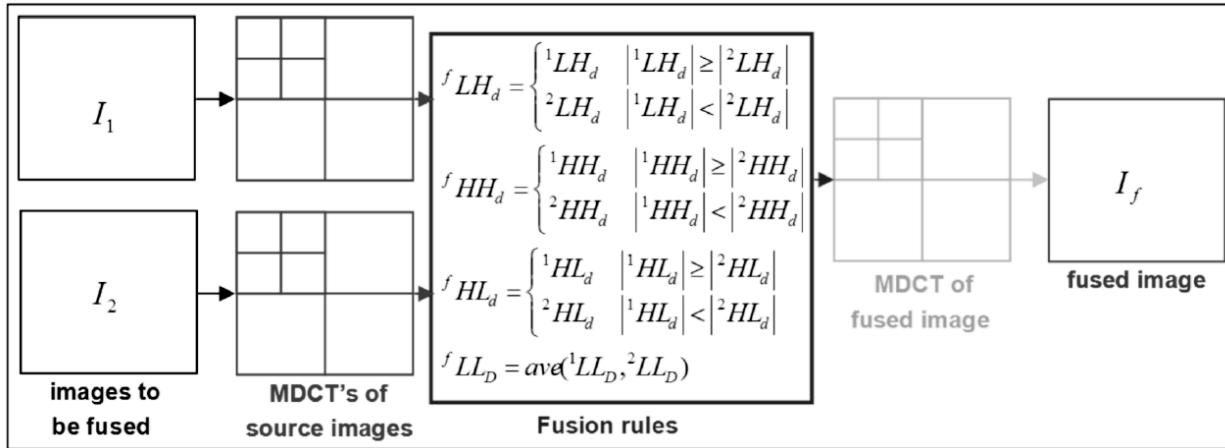


(k)

בשני התמונות שגיאה מרבית בין תמונה מקורית לתמונה שעברה פירוק והרכבה , בסדר גודל של $.10^{-13}$

היתוך

הדיagramה הסכמתית של היתוך תמונות על בסיס הפיקסל מבודס MDCT מוצגת באירור 3. נשים לב שהשינוי של האלגוריתם הנוכחי הוא השימוש ב-MDCT במקום Wavelets או פירמידות. התמונות להיתוך I₁ ו-I₂ מפורקות ל-D רמות באמצעות MDCT.



אייר מס 3 הדיאגרמה הסכמתית עבור היתוך תמונות ברמת פיקסלים מבוססת MDCT

I1 ו-I2 מפורקות הינן :
בכל רמת פירוק (D=1, 2,..., D), כל היתוך יהיה

בחירת הערך המוחלט הגדול ביותר של מקדמי הפרטים (Details), מכון שהמקדים ה"פרטים" מתאימים לשינויי בהירות חדים יותר בתמונות כמו קצוטות וגבולות אובייקט וכו'. מקדים אלה נעים סבב האפס. בrama הגסה ביותר (D=d), כל היתוך לוקח את המוצע של מקדי ה"קירוב" (Approximations) של MDCT שכן מקדי הקירוב הם גסים יותר הרמה היא הגרסה המוחלטת ותת הדגימה של התמונה המקורית. לאחר היתוך מותבצע תהליך הרכבה לייצור תמונה נוספת אשר נמצאת בפוקוס מלא.

הערכת ביצועים

עם תמונה יhos

כאשר תמונה יhos זמינה, הביצועים של אלגוריתמי היתוך תמונה ניתן להעריך באמצעות המדדים הבאים :

1. Percentage fit error (PFE)

למעשה זאת שגיאה ייחסת כאשר, הנורמה היא האופרטור לחישוב הערך הסינגולרי הגדול ביותר. מחושב כנורמה של ההבדל בין הפיקסלים של תמונה יhos ותמונה לאחר היתוך, חלקי נורמה של תמונה יhos.

יהיה אפס מתי גם תמונות היחסו וגם תמונות מתמצגות דומות לחלוטין, יחס שגיאה תגדל כאשר התמונה המmozגת שונה מתמונה היחסו.

2. Peak signal to noise ratio (PSNR)

כאשר L הוא מספר רמות האפור בתמונה. הערך של המטריקה תהיה בעלת ערך גבוה כאשר היתוך ותמונה יhos דומות. ערך גבוה יותר מרמז על היתוך טוב יותר.

$$PSNR = 20 \log_{10} \left(\frac{L^2}{\frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N (I_r(i,j) - I_f(i,j))^2} \right)$$

יש טעות אמרת להיות log₁₀, אם משתמשים ב log נדרש להוציא את חזקה 2 מתוך log. •

$$SSIM = \frac{(2\mu_{I_r}\mu_{I_f} + C_1)(2\sigma_{I_rI_f} + C_2)}{(\mu_{I_r}^2 + \mu_{I_f}^2 + C_1)(\sigma_{I_r}^2 + \sigma_{I_f}^2 + C_2)}$$

3. Measure of structural similarity (SSIM). אוטות של תמונות טבעיות יהיו מובנים מאד והפיקסלים שלהם חושפים תלות חזקה. תלות זו ישאו מידע חיוני על מבנה האובייקט. מטריקה זו משווה תכניות מקומיות של עצמות פיקסלים שעברו נורמליזציה עבור בהירות וניגודיות.

אני בחרתי לחתך חלון בגודל של 11×11 לחישוב ולאחר מכן מכנו כל התוצאות עוברו מיצעו לקבלת ערך בודד.

כאשר :

$$\sigma_{I_r}^2 = \frac{1}{MN-1} \sum_{i=1}^M \sum_{j=1}^N (I_r(i,j) - \mu_{I_r})^2 \quad \mu_{I_f} = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N I_f(i,j) \quad \mu_{I_r} = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N I_r(i,j)$$

$$\sigma_{I_f}^2 = \frac{1}{MN-1} \sum_{i=1}^M \sum_{j=1}^N (I_f(i,j) - \mu_{I_f})^2 \quad \sigma_{I_rI_f} = \frac{1}{MN-1} \sum_{i=1}^M \sum_{j=1}^N (I_r(i,j) - \mu_{I_r})(I_f(i,j) - \mu_{I_f})$$

לא תמונה יהוס

כאשר תמונה יהוס אינה זמינה, ניתן להשתמש במדדים הבאים כדי לבדוק את הביצועים של האלגוריתמים.

1. Standard deviation (SD)

h_{I_f} היא ההיסטוגרמה המנורמלת של תמונה ממוגנת. ידוע שטיטיתת תקן מרכיבת חלקיקי אותן ורעש. מגד זה יהיה קטן יותר בהיעדר רעש. הוא מודד את הניגודיות בתמונה הממוגנת. לתמונה עם ניגודיות גבוהה תהיה סטיית תקן גבוהה.

2. Cross entropy (CE)

אנטרופיה צולבת כוללת של תמונות המקור I_1, I_2 , והתמונה ממוגנת היא :

$$CE(I_1; I_f) = \sum_{i=0}^L h_{I_1}(i) \log \left(\frac{h_{I_1}(i)}{h_{I_f}(i)} \right) \quad CE(I_2; I_f) = \sum_{i=0}^L h_{I_2}(i) \log \left(\frac{h_{I_2}(i)}{h_{I_f}(i)} \right) \quad CE(I_1, I_2; I_f) = \frac{CE(I_1; I_f) + CE(I_2; I_f)}{2}$$

3. Spatial frequency (SF)

קריטריון תדר מרחבוי

$$SF = \sqrt{RF^2 + CF^2}$$

תדרי השורה

$$RF = \sqrt{\frac{1}{MN} \sum_{x=1}^M \sum_{y=2}^N [I_f(x,y) - I_f(x,y-1)]^2}$$

תדרי عمودה

$$CF = \sqrt{\frac{1}{MN} \sum_{y=1}^N \sum_{x=2}^M [I_f(x,y) - I_f(x-1,y)]^2}$$

תדרות זו בתחום המרחבי מצינית את רמת הפעולות הכוללות בתמונה הממוגנת. (y,x) הוא אינדקס הפיקסלים.

פתרונות

התוצאות הבאות הן עבור תמונה ממאמר עם תמונה ייחוס (תהייה שונה במטריקות ממה שモצאים במאמר מכoon שלא מצאתי תמונות אלו באינטרנט וביצועי צילום מסך) ובונוסף תמונות נוספות שצולמו עם מוקד f=2.8.

חשוב לציין שיש חשיבות רבה לתמונות אשר נלקחו בגודל ומיקום זהה, ורוק פוקוס שונה בתלות אזור.

	Image_1	Image_2	Reference_Image
MDCT Level 1			
	Fused_Image	Error_Image	
MDWT Level 1			

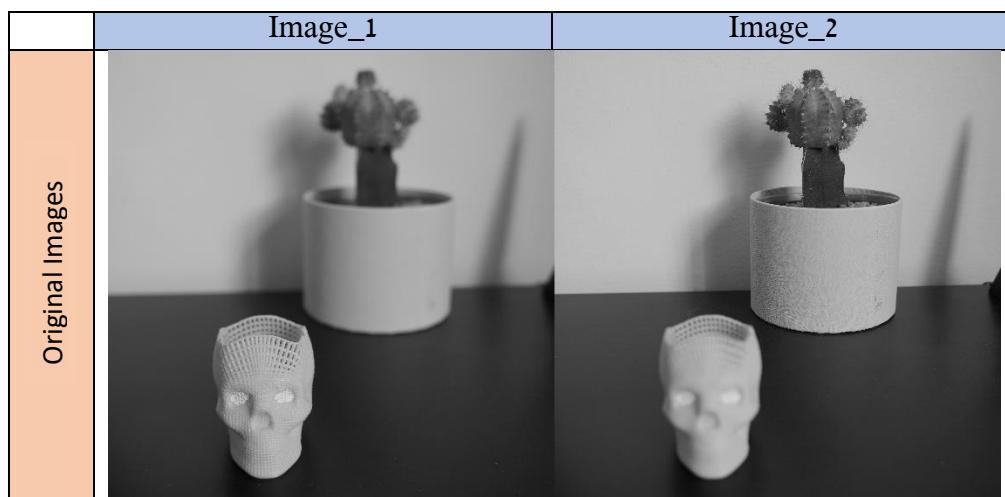
MDCT Level 2	 <small>Approx: mean Details: max</small>		
MDWT Level 2	 <small>Approx: mean Details: max</small>		
MDCT Level 8	 <small>Approx: mean Details: max</small>		

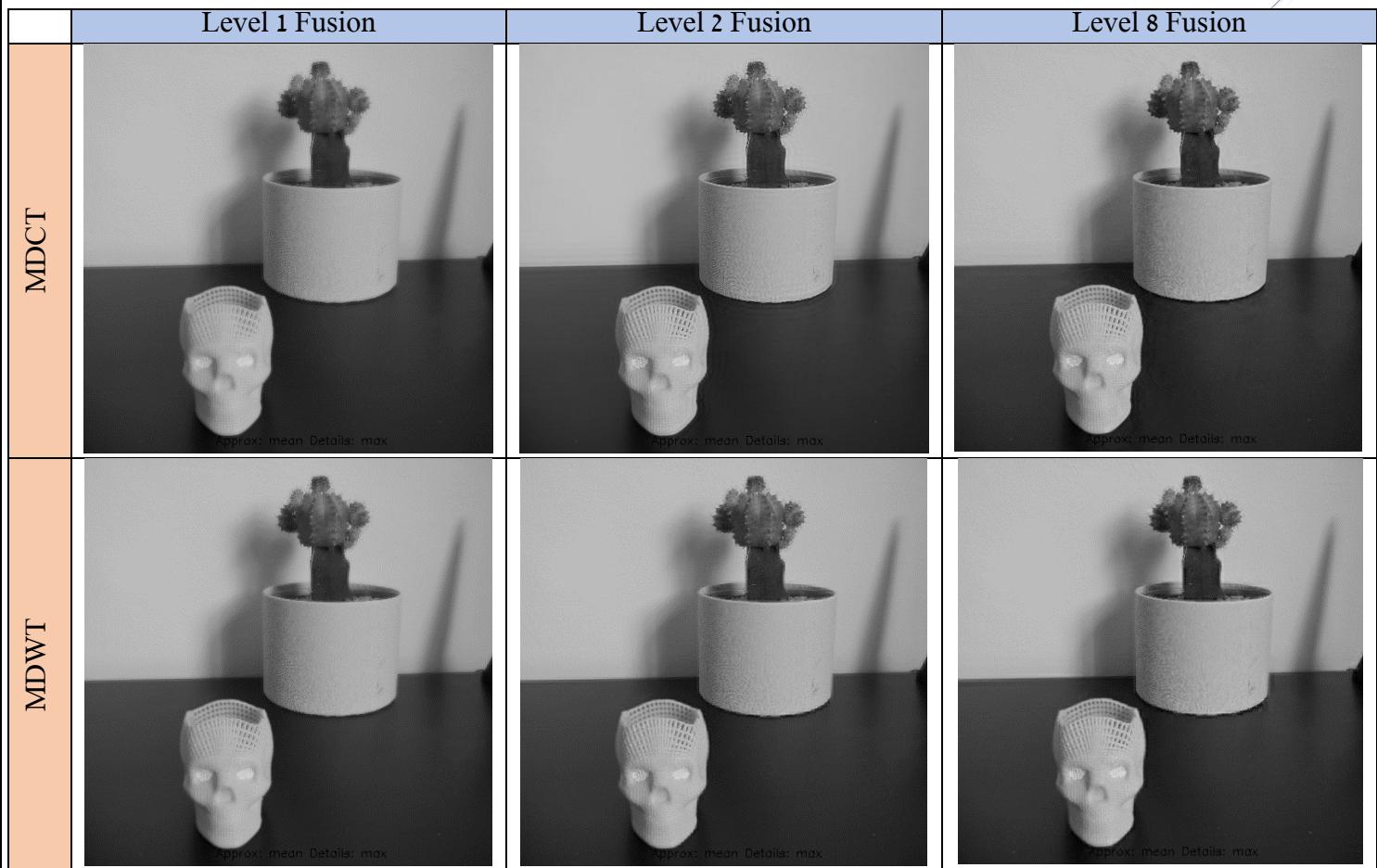


Level of Decomposition	Algorithm	With reference image			Without reference image		
		PFE	PSNR	SSIM	CE	SD	SF
1	MDCT	3.7012	29.4315	0.9701	1.2062	45.9533	8.7602
1	MDWT	3.5579	29.7743	0.9737	7.2569	46.2713	10.4643
2	MDCT	3.3231	30.3675	0.9646	7.0789	46.4826	11.467
2	MDWT	3.0984	30.9754	0.9742	7.5978	47.0271	12.6106
4	MDCT	1.3315	38.3112	0.9906	1.6225	48.6852	13.1857
4	MDWT	2.2186	33.8768	0.9675	7.4041	49.0441	13.4315
8	MDCT	0.9726	41.0397	0.9936	1.0642	49.642	13.2462
8	MDWT	2.2214	33.8657	0.9665	5.5177	50.1258	13.4005

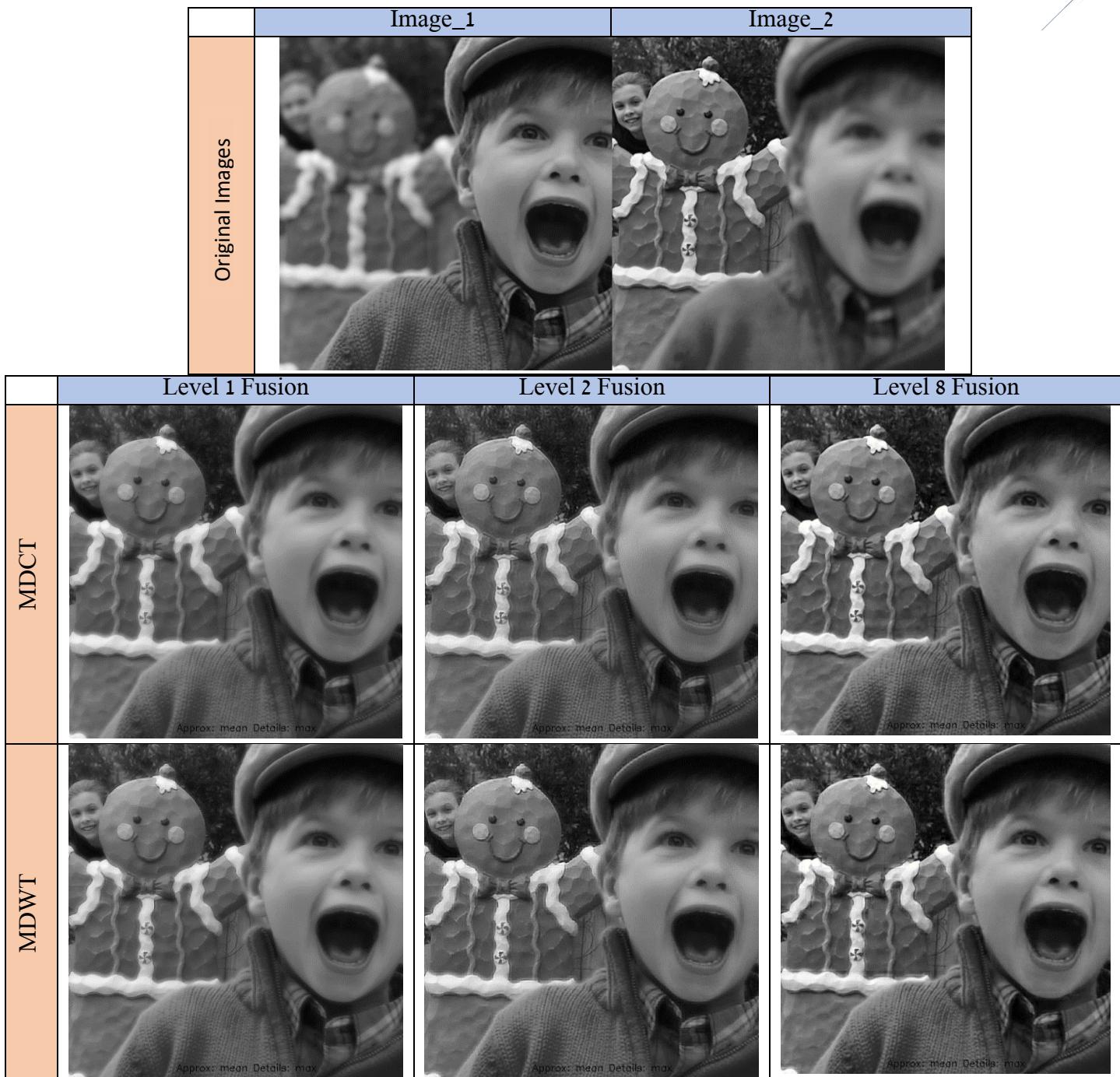
מסקנות :

ניתן לראות כי התמונות המאוחדות בעזרת MDCT ו-wavelet הן כמעט דומות עברו תמונות אלה. מdziי הביצועים להערכת אלגוריתמי היתוך התמונה מוצגים בטבלה. המדידים המוצגים בטבלה עם צבע ירוק טובים יותר מאחרים. הביצועים של MDCT כמעט דומים לאלו של wavelets. רמה גבוהה יותר של פירוק גוררת היפוך ביצועים, לפיה טבלה עד לרמה 2 ביצועי של wavelets הם טובים יותר מזה של DCT (צבע צהוב), אך בפירוק יותר גבוה ביצועים של DCT טובים יותר (צבע ירוק).





Level of Decomposition	Algorithm	Without reference image		
		CE	SD	SF
1	MDCT	0.6753	60.7842	10.8101
1	MDWT	0.8117	60.8348	10.9403
2	MDCT	0.5923	60.9136	11.7342
2	MDWT	0.7668	60.9843	11.6317
4	MDCT	0.5831	61.102	11.9692
4	MDWT	0.6348	61.2286	11.8002
8	MDCT	0.6275	61.5079	11.9812
8	MDWT	0.6644	61.6061	11.7941

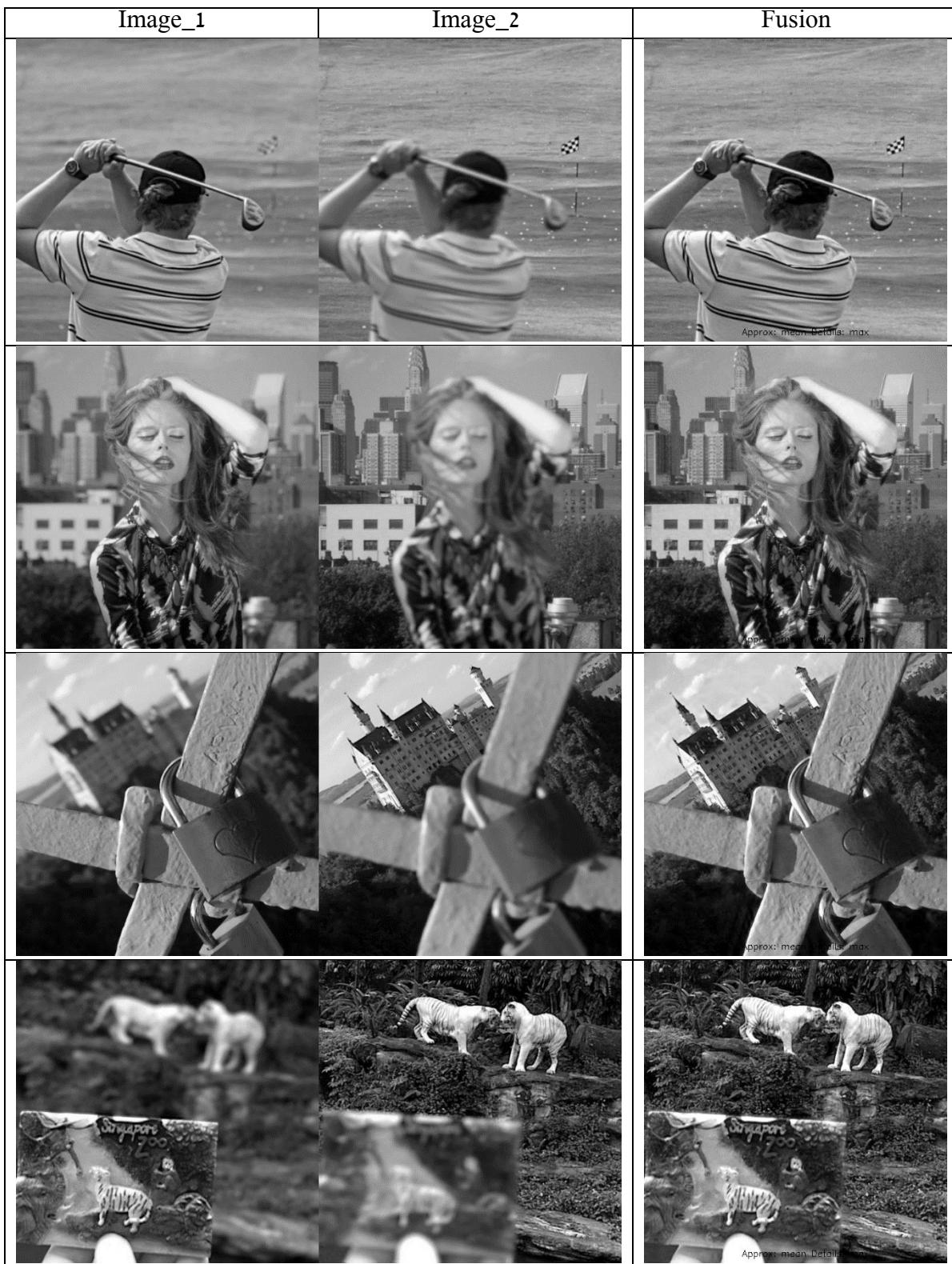


Level of Decomposition	Algorithm	Without reference image		
		CE	SD	SF
1	MDCT	0.0396	46.2209	8.537
1	MDWT	0.046	46.462	10.2323
2	MDCT	0.0593	46.6581	11.1503
2	MDWT	0.2761	47.1389	12.2903
4	MDCT	0.0832	48.4128	12.7545
4	MDWT	0.6317	48.9569	13.1278
8	MDCT	0.1045	49.8778	12.8213
8	MDWT	0.5437	50.2511	13.1017

עוד תמונות עברו DCT בלבד:

	Image_1	Image_2		
Original Images				
MDCT	Level 1 Fusion	Level 2 Fusion	Level 8 Fusion	
	Image_1	Image_2		
Original Images				
MDCT	Level 1 Fusion	Level 2 Fusion	Level 8 Fusion	

עוד תמונות עברו DCT עם עומק 8



מסקנות:

היתוך תמונה ברמת פיקסלים על ידי אלגוריתם MDCT יושם והוערך. הביצועים של אלגוריתם זה מושווים לטכניקת היתוך תמונה ידועה על ידי wavelets. המסקנה היא שהיתוך תמונה על ידי MDCT דומה כמעט לוזה של wavelets. זה מאד פשוט מבחינה חישובית והוא יכול להתאים היבר לישומים בזמן אמת. היתוך תמונה על ידי רמה גבוהה יותר של פירוק מספק תוצאות היתוך טובות יותר.

ניסויו לשלב תМОנות רפואיות :

	MRT	CT	
Original Images			
	Level 1 Fusion Approx: mean, Details: max	Level 1 Fusion Approx: min, Details: max	Level 7 Fusion Approx: mean, Details: max
MDCT			
MDWT			

הסבר על שימוש בGUI

GUI רשום בשפת פיתון

את GUI ניתן להפעיל בעזרת שני דרכי:

.3

בעזרת שימוש בקובץ DCT - based Image Fusion.py או בסביבת העבודה Visual Studio Code

יש לתקן ספריות הבאות (לעומץ פקודות):

```
pip install tk
pip install numpy
pip install matplotlib
pip install scipy
pip install opencv-python
pip install scikit-image
pip install PyWavelets
pip install tabulate
```

.4

בעזרת הפעלת קובץ --Double Click Me To Run The App --.bat

*ראשית יש להוריד את תיקיה למחשב ולבצע unzip

**לא נדרש להתקין ספריות

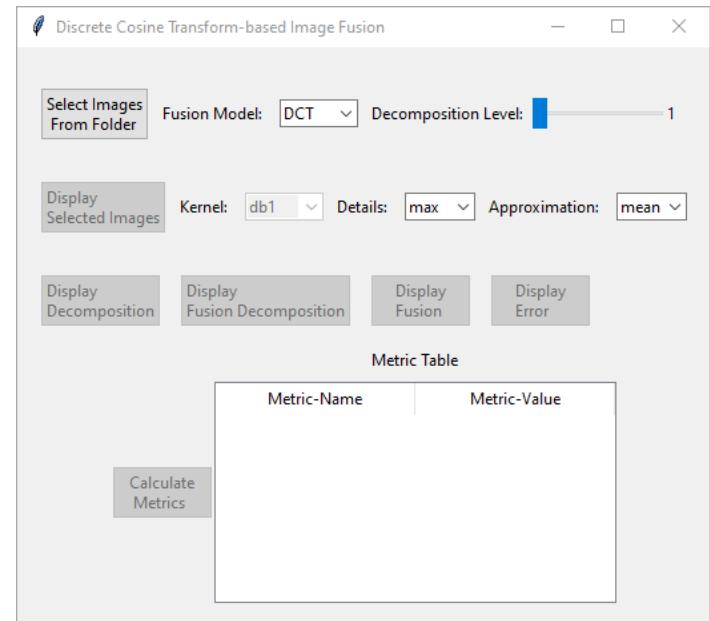
שימוש:

• Select Images from Folder – בחירה מרובה של תמונות בעזרת החזקת מקש CTRL ובבחירה בעזרת עכבר או סימון בעזרת עכבר בחזקת מקש שמאלי. ניתן לבחור מ2 עד 3 תמונות כאשר תמונה שלישיית היא תמונה ייחוס.

בעט בחירה של שני תמונות בלבד תאפשר לבחירה על כל הcptוראים פרט לכפטורן,Display Error,מכוון שנדרשת תמונה ייחוס.

• Display Selected Images – כפטור שיציג את תמונות שנבחרו,* כדי להמשיך ולהזוז על cptוראים אחרים קודם יש לסגור את חלון הפתוח, סגירת חלון אפשרית בשלושה דרכיים: 1) לחיצה מקש ESC (2) בעזרת עכבר בלחיצה על X (3) בעזרת לחיצה על מקש ימין של עכבר כאשר סמן נמצא באזורי החלון פתוח.

• Display Decomposition - הצגת פירוק של תמונות שנבחרו להיתוך (תמונה ראשונה ושניה)



- – בחירת אלגוריתם היתוך, כאשר נבחר DWT תחתאר אוופציה לשינוי Kernel Model [haar, db1] (Mother Wavelet) ** לא נראה שיש הבדל!!
- – רמת הפירוק של התמונות לצורך היתוך [1-8], הפעלה וקורסיבית של פירוק על תמונה LL.
- – Approximation ו – Details – בחירת אופן מיזוג של התמונות [min, max, mean]
- – Display Fusion Decomposition – הציגת פירוק לאחר היתוך, היתוך מתבצע לפי בחירה של Approximation ו Details
- – Display Fusion – הציגת תמונה ממוגנת לאחר היתוך
- – Display Error – אופציה הזאת קיימת רק בבחירה של שלוש תמונות כאשר תמונה שלישית היא תמונה ייחוס, מציגה שגיאה בין תמונה ממוגנת לתמונה ייחוס.
- – Calculate Metrics – הציגת מטריות של יעילות של אלגוריתם, כאשר נבחרו רק שני תמונות (תמונות למיזוג) תוצג רק 3 מטריות, וכאשר תבחר 3 תמונות תוצג 6 מטריות.
- **** כדי להמשיך וללחוץ על כפתורים אחרים קודם יש לסגור את חלון הפתוח, סגירת חלון אפשרית בשלושה דרכים:** 1) לחיצת מקש ESC 2) בעזרת עכבר בלחיצה על X (3) בעזרת לחיצה על מקש ימיון של עכבר כאשר סמן נמצא באזור החלון פתוח.

- **הקוד נמצא ב Appendix B - Script ו Git**

Appendix A – Script

```

import tkinter as tk
from tkinter import filedialog
import numpy as np
import matplotlib.pyplot as plt
import cv2
from scipy import ndimage
from skimage import exposure ,metrics
from tkinter import ttk

def disp_img(img , title = 'img' ,text = {'text' : [None],'loc':[(165,500)]}):
    def mouse_callback(event, x, y, flags, param):
        if event == cv2.EVENT_RBUTTONDOWN:
            cv2.destroyAllWindows()

        I = img.copy()
        avg = np.mean(I)
        for i , val  in enumerate(text['text']):
            if avg> 100:
                cv2.putText(I, val, text['loc'][i], cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 1)
            else:
                cv2.putText(I, val, text['loc'][i], cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 1)

        cv2.imshow(title ,I)
        cv2.setWindowProperty(title, cv2.WND_PROP_TOPMOST, 1)

    # Associate the callback function with the named window
    cv2.setMouseCallback(title, mouse_callback)

    ##### Convert Color Spaces #####
def BGRtoHSV(BGR):
    hsv = cv2.cvtColor(BGR, cv2.COLOR_BGR2HSV)
    return cv2.split(hsv)

def HSVtoBGR(H,S,V):
    hsv = np.stack([H,S,V],axis=2)
    bgr = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
    return bgr

##### Denoise TV #####
def denoise_tv(image, weight=1/90, eps=1.e-6, max_num_iter=200):
    """Perform total-variation denoising on n-dimensional images.

    Parameters
    -----
    image : ndarray
        n-D input data to be denoised.
    weight : float, optional
        Denoising weight It is equal to 1/lambda . The greater `weight` , the more denoising .
    eps : float, optional
        Relative difference of the value of the cost function that determines
        the stop criterion. The algorithm stops when:
        ( $E_{n-1} - E_n$ ) <  $\epsilon * E_0$ 
        where  $E_0$  is the initial value of the cost function.
    max_num_iter : int, optional
        Maximal number of iterations used for the optimization.

    Returns
    -----
    out : ndarray
        Denoised array uint8 [0 - 255].
    """
    image = image.astype(np.float64) # convert image to float
    ndim = image.ndim
    p = np.zeros((image.ndim, ) + image.shape, dtype=image.dtype)
    g = np.zeros_like(p)
    d = np.zeros_like(image)
    i = 0
    while i < max_num_iter:
        if i > 0:
            # d will be the (negative) divergence of p
            d = -p.sum(0)
            slices_d = [slice(None), ] * ndim
            slices_p = [slice(None), ] * (ndim + 1)

```

```

for ax in range(ndim):
    slices_d[ax] = slice(1, None)
    slices_p[ax+1] = slice(0, -1)
    slices_p[0] = ax
    d[tuple(slices_d)] += p[tuple(slices_p)]
    slices_d[ax] = slice(None)
    slices_p[ax+1] = slice(None)
    out = image + d
else:
    out = image
E = (d ** 2).sum()

# g stores the gradients of out along each axis
# e.g. g[0] is the first order finite difference along axis 0
slices_g = [slice(None), ] * (ndim + 1)
for ax in range(ndim):
    slices_g[ax+1] = slice(0, -1)
    slices_g[0] = ax
    g[tuple(slices_g)] = np.diff(out, axis=ax)
    slices_g[ax+1] = slice(None)

norm = np.sqrt((g ** 2).sum(axis=0))[np.newaxis, ...] # calculate magnitude
E += weight * norm.sum() # Update cost function
tau = 1. / (2.*ndim) # calc step
norm *= tau / weight
norm += 1.
norm -= tau * g
p /= norm
E /= float(image.size)
if i == 0:
    E_init = E
    E_previous = E
else:
    if np.abs(E_previous - E) < eps * E_init:
        break
    else:
        E_previous = E
i += 1
print(i, tau)
return out.astype(np.uint8)

#####
# Adaptive Gamma Correction #####
def adaptive_gamma_transform(img, n,m):
    """
    Applies adaptive gamma transform on a given image.

    Args:
        img: A grayscale image to be processed.
        m: Size of the local area (height).
        n: Size of the local area (width).

    Returns:
        A gamma corrected image.
    """
    rows, cols = img.shape
    gamma_corrected = np.zeros((rows, cols))

    # Add small value to ignore zero division error and convert to float
    img = (img+1.)/255.

    for i in range(rows):
        for j in range(cols):
            rmin = max(0, i - m//2)
            rmax = min(rows, i + m//2 + 1)
            cmin = max(0, j - n//2)
            cmax = min(cols, j + n//2 + 1)
            local_area = img[rmin:rmax, cmin:cmax]

            N = np.mean(local_area) # calculate mean on local area n x m
            b = np.var(local_area) # calculate var on local area n x m

            # Calculate the gamma value.
            gamma = N/img[i,j] + b

            # Gamma correct the pixel value.
            gamma_corrected[i,j] = np.power(img[i,j], gamma)

    return (gamma_corrected*255).astype(np.uint8) # transform back to uint8 [0 - 255]

#####
# MSR #####

```

```

def get_ksize(sigma):
    # Opencv calculates ksize from sigma as
    # sigma = 0.3*((ksize-1)*0.5 - 1) + 0.8
    # then ksize from sigma is
    # ksize = ((sigma - 0.8)/0.15) + 2.0

    return int(((sigma - 0.8)/0.15) + 2.0)

def get_gaussian_blur(img, ksize=0, sigma=5):
    # Perform convolution I(i,j)*G(i,j)
    # if ksize == 0, then compute ksize from sigma
    if ksize == 0:
        ksize = get_ksize(sigma)

    # Gaussian 2D-kernel can be seperable into 2-orthogonal vectors
    # then compute full kernel by taking outer product or simply mul(V, V.T)
    sep_k = cv2.getGaussianKernel(ksize, sigma)

    return cv2.filter2D(img, -1, np.outer(sep_k, sep_k))

def ssr(img, sigma):
    # Single-scale retinex of an image
    # SSR(x, y) = log(I(x, y)) - log(I(x, y)*G(x, y))
    # G = surrounding function,( Gaussian )

    return np.log10(img) - np.log10(get_gaussian_blur(img, ksize=0, sigma=sigma)) + 1.0

def msr(img, sigma_scales=[15, 80, 250], apply_normalization=True):
    # Multi-scale retinex of an image
    # MSR(x,y) = sum(weight[i]*SSR(x,y, scale[i])), i = {1..n} scales
    img = img + 1.0 # add small value to ignore log(0)
    msr = np.zeros(img.shape)
    # for each sigma scale compute SSR
    for sigma in sigma_scales:
        msr += ssr(img, sigma)

    # divide MSR by weights of each scale
    # here we use equal weights
    msr = msr / len(sigma_scales)

    # computed MSR could be in range [-k, +l], k and l could be any real value
    # so normalize the MSR image values in range [0, 255]
    if apply_normalization:
        return cv2.normalize(msr, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8UC3)
    else:
        return msr

#####
##### Multi-scale Hyperbolic Tangent Enhancement #####
def tanh(img, sigma):
    # Single-scale Hyperbolic Tangent Enhancement
    # tanh(I(x,y) / (I(x, y)*G(x, y)))
    # G = surrounding function,( Gaussian )
    return np.tanh(img/get_gaussian_blur(img, ksize=0, sigma=sigma))

def mtanh(img, sigma_scales=[15, 80, 250]):
    # Multi-scale Hyperbolic Tangent Enhancement
    img = img + 1.0 # add small value to ignore zero division
    i_t = np.zeros(img.shape)
    # for each sigma scale compute tanh
    for sigma in sigma_scales:
        i_t += tanh(img, sigma)

    # divide tanh by weights of each scale
    # here we use equal weights 1/3
    i_t = i_t / len(sigma_scales)

    return (i_t*255).astype(np.uint8) # transform back to uint8 [0 - 255]

#####
##### Double-Function Image Enhancement #####
def DFIE(img , sigma=[10,40,300],n = 3,m= 3):
    i_l = msr(img,sigma).astype(np.float64) # calculate weighted MSR
    i_t = mtanh(img,sigma).astype(np.float64) # calculate weighted tanh

    # Using gausian to estimate mean , much faster than do in in loop
    i_l_mean = cv2.blur(i_l, (n, m))
    i_t_mean = cv2.blur(i_t, (n, m))
    a= i_t_mean/i_l_mean

```

```

alpha = cv2.normalize(a, None, 0, 1.0, cv2.NORM_MINMAX, dtype=cv2.CV_64F)

balanced = alpha*i_l + (1-alpha)*i_t

# # Same as above but take much longer to calculate
# rows, cols = img.shape
# balanced = np.zeros((rows, cols))

# for i in range(rows):
#     for j in range(cols):
#         rmin = max(0, i - m//2)
#         rmax = min(rows, i + m//2 + 1)
#         cmin = max(0, j - n//2)
#         cmax = min(cols, j + n//2 + 1)
#         # Calculate the indices for the local area.
#         local_i_l = i_l[rmin:rmax, cmin:cmax]
#         local_i_t = i_t[rmin:rmax, cmin:cmax]
#         alpha = np.mean(local_i_t)/np.mean(local_i_l)
#         balanced[i,j] = alpha*i_l[i,j]+(1-alpha)*i_t[i,j]

return balanced.astype(np.uint8) # transform back to uint8 [0 - 255]

#####
##### Three-Dimensional Gamma Correction #####
def three_dim_gamma_correction(image, weights=[0.05,0.05,0.2], n=3, m=3):
    # add some small value to ignore zero division and convert to float [0.0 - 1.0]
    image = (image+1.)/255.
    # Initialize output image
    output_image = np.zeros_like(image)
    rows, cols = image.shape

    gx = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
    gy = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
    Gr =np.hypot(gx,gy) # calculate magnitude, same as sqrt(gx**2 +gy**2)
    # Iterate over each pixel in the input image
    for i in range(rows):
        for j in range(cols):
            rmin = max(0, i - m//2)
            rmax = min(rows, i + m//2 + 1)
            cmin = max(0, j - n//2)
            cmax = min(cols, j + n//2 + 1)
            # Extract local region of size n x m around the pixel
            local_region = image[rmin:rmax, cmin:cmax]
            # Compute local maximum, mean gradient, and variance
            local_max = np.max(local_region)
            local_mean_gradient = np.mean(Gr[i:i+n, j:j+m])
            local_variance = np.var(local_region)

            # Compute gamma correction factor based on local statistics and weights
            gamma = weights[0]*np.exp(image[i,j]/local_max) + weights[1]*np.exp(local_mean_gradient) + weights[2]*np.exp(local_variance)

            # Apply gamma correction to pixel value
            output_image[i,j] = np.power(image[i,j],gamma)*255

    return output_image.astype(np.uint8) # transform back to uint8 [0 - 255]

#####
##### Adaptive Saturation Correction #####
def adaptive_saturation_adjustment(s_channel,n,m):
    # add some small value to ignore zero division and convert to float [0.0 - 1.0]
    s_channel= (s_channel+1.)/255
    rows, cols = s_channel.shape
    saturation_corrected = np.zeros((rows, cols))

    # Get x-gradient in "sx"
    sx = cv2.Sobel(s_channel, cv2.CV_64F, 1, 0, ksize=3)
    # Get y-gradient in "sy"
    sy = cv2.Sobel(s_channel, cv2.CV_64F, 0, 1, ksize=3)
    # Get square root of sum of squares
    Sg=np.hypot(sx,sy)

    # # Compute the global mean value of the S channel
    S_mean = np.mean(s_channel)

    for i in range(rows):
        for j in range(cols):
            rmin = max(0, i - m//2)
            rmax = min(rows, i + m//2 + 1)

```

```

cmin = max(0, j - n//2)
cmax = min(cols, j + n//2 + 1)
# Extract local region of size n x m around the pixel
local_region = s_channel[rmin:rmax, cmin:cmax]
# Calculate the average of the local area.
Sm = np.mean(local_region)

# Apply regulation
if s_channel[i,j] <= S_mean+Sg[i,j]:
    saturation_corrected[i,j] = 1+0.8*np.log10(Sm/(s_channel[i,j]+0.5*Sg[i,j]))
else:
    saturation_corrected[i,j] = np.exp((Sm-s_channel[i,j])/2)

return cv2.normalize(saturation_corrected, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8UC3)

#####
# Multi Scale Retinex with Color Restoration #####
def color_balance(img, low_per, high_per):
    '''Contrast stretch img by histogram equalization with black and white cap'''

    tot_pix = img.shape[1] * img.shape[0]
    # no.of pixels to black-out and white-out
    low_count = tot_pix * low_per / 100
    high_count = tot_pix * (100 - high_per) / 100

    # channels of image
    ch_list = []
    if len(img.shape) == 2:
        ch_list = [img]
    else:
        ch_list = cv2.split(img)

    cs_img = []
    # for each channel, apply contrast-stretch
    for i in range(len(ch_list)):
        ch = ch_list[i]
        # cummulative histogram sum of channel
        cum_hist_sum = np.cumsum(cv2.calcHist([ch], [0], None, [256], (0, 256)))

        # find indices for blacking and whiting out pixels
        li, hi = np.searchsorted(cum_hist_sum, (low_count, high_count))
        if (li == hi):
            cs_img.append(ch)
            continue
        # lut with min-max normalization for [0-255] bins
        lut = np.array([0 if i < li
                       else (255 if i > hi else round((i - li) / (hi - li) * 255))
                       for i in np.arange(0, 256)], dtype = 'uint8')
        # contrast-stretch channel
        cs_ch = cv2.LUT(ch, lut)
        cs_img.append(cs_ch)

    if len(cs_img) == 1:
        return np.squeeze(cs_img)
    elif len(cs_img) > 1:
        return cv2.merge(cs_img)
    return None

def msr_cr(img, sigma_scales=[15, 80, 250], alpha=125, beta=46, G=192, b=-30, low_per=1, high_per=1):
    # Multi-scale retinex with Color Restoration
    # MSR(x,y) = G * [MSR(x,y)*CRF(x,y) - b], G=gain and b=offset
    # CRF(x,y) = beta*[log(alpha*I(x,y) - log(I'(x,y)))]
    # I'(x,y) = sum(Ic(x,y)), c={0...k-1}, k=no.of channels

    img = img + 1.0
    # Multi-scale retinex and don't normalize the output
    msr_img = msr(img, sigma_scales, apply_normalization=False)
    # Color-restoration function
    crf = beta * (np.log10(alpha * img) - np.log10(np.sum(img, axis=2, keepdims=True)))
    # MSR
    msr_cr = G * (msr_img*crf - b)
    # normalize MSR
    msr_cr = cv2.normalize(msr_cr, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8UC3)
    # color balance the final MSR to flat the histogram distribution with tails on both sides
    msr_cr = color_balance(msr_cr, low_per, high_per)

    return msr_cr

#####
# CLAHE #####

```

```

def CLAHE(Img):
    # Convert image to LAB color space
    lab = cv2.cvtColor(Img, cv2.COLOR_BGR2LAB)

    # Split LAB image into separate channels
    l, a, b = cv2.split(lab)

    # Apply CLAHE to L channel
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
    cl = clahe.apply(l)

    # Merge the CLAHE enhanced L channel with the other LAB channels
    lab_cl = cv2.merge((cl,a,b))

    # Convert back to RGB color space
    final = cv2.cvtColor(lab_cl, cv2.COLOR_LAB2BGR)
    return final

#####
# Adaptive histogram equalization
#####
def AHE(Img):
    eq = exposure.equalize_adapthist(Img)
    return cv2.normalize(eq, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8UC3)

#####
# Metrics
#####
def PSNR(I_r, I_f):
    mse = np.mean((I_r - I_f) ** 2)
    max_pixel = 255
    psnr = 10 * np.log10(max_pixel ** 2 / mse)
    # psnr = 20 * np.log10(max_pixel/np.sqrt(mse))
    return round(psnr, 4)

def SD(I_f):
    # Compute the histogram
    hist, bins = np.histogram(I_f.flatten(), bins=256)
    # Compute the mean of the histogram
    mean = np.sum(hist * bins[:-1]) / np.sum(hist)

    # Compute the variance of the histogram
    variance = np.sum((bins[:-1] - mean) ** 2 * hist) / np.sum(hist)

    # Compute the standard deviation of the histogram
    return round(np.sqrt(variance), 4)

def SSIM(I_r, I_f, L=255):
    K1 = 0.01
    K2 = 0.03
    C1 = (K1 * L) ** 2
    C2 = (K2 * L) ** 2
    # INITs
    I2_2 = I_f ** 2 # I2^2
    I1_2 = I_r ** 2 # I1^2
    I1_I2 = I_r * I_f # I1 * I2
    # END INITs
    # PRELIMINARY COMPUTING
    mu1 = cv2.GaussianBlur(I_r, (11, 11), 1.5)
    mu2 = cv2.GaussianBlur(I_f, (11, 11), 1.5)
    mu1_2 = mu1 ** 2
    mu2_2 = mu2 ** 2
    mu1_mu2 = mu1 * mu2
    sigma1_2 = cv2.GaussianBlur(I1_2, (11, 11), 1.5)
    sigma1_2 -= mu1_2
    sigma2_2 = cv2.GaussianBlur(I2_2, (11, 11), 1.5)
    sigma2_2 -= mu2_2
    sigma12 = cv2.GaussianBlur(I1_I2, (11, 11), 1.5)
    sigma12 -= mu1_mu2
    t1 = 2 * mu1_mu2 + C1
    t2 = 2 * sigma12 + C2
    t3 = t1 * t2 # t3 = ((2*mu1_mu2 + C1).*(2*sigma12 + C2))
    t1 = mu1_2 + mu2_2 + C1
    t2 = sigma1_2 + sigma2_2 + C2
    t1 = t1 * t2 # t1 =((mu1_2 + mu2_2 + C1).*(sigma1_2 + sigma2_2 + C2))
    ssim_map = t3 / t1
    mssim = np.mean(ssim_map) # mssim = average of ssim map
    return round(mssim, 4)

```

```

def IE(I_f):
    # Add epsilon to avoid division by zero errors
    epsilon = 2**(-32)
    # Compute the histogram of the image
    hist, _ = np.histogram(I_f.flatten(), bins=256)

    # Calculate the total number of pixels in the image
    num_pixels = np.sum(hist) # same as N*M

    # Calculate the PMF by dividing each bin in the histogram by the total number of pixels
    hist_p = hist / num_pixels

    hist_p = np.clip(hist_p, epsilon, 1)
    E = -np.sum(hist_p * np.log2(hist_p))
    return round(E, 4)

def metric(I_ref, I_enc):
    I_ref_gray = cv2.cvtColor(I_ref, cv2.COLOR_BGR2GRAY).astype(np.float64)
    I_enc_gray = cv2.cvtColor(I_enc, cv2.COLOR_BGR2GRAY).astype(np.float64)
    info_ref = {'PSNR': PSNR(I_ref_gray, I_enc_gray), 'SSIM': SSIM(I_ref_gray, I_enc_gray, L=255), 'SD': SD(I_enc_gray), 'IE': IE(I_enc_gray)}
    return info_ref

def Model(Img, model = 'DFE', disp_selector = [False, False, False, False, False, False, False, False, False],
          sigma = [10, 40, 400], weights=[0.05, 0.05, 0.2], kernel = [9, 9], lam = 40):

    if model == 'DFE':
        # disp_selector = [Original, Original & I_o, HSV, I_d, I_p, I_out, I_img, I_u, S_tag]
        h, s, v = BGRtoHSV(Img)
        n, m = kernel
        ##### V - Channel #####
        I_v = denoise_tv(v, weight=1/lam, eps=1e-6, max_num_iter=100)
        I_d = adaptive_gamma_transform(I_v, n=3, m=3)
        I_p = DFIE(I_d, sigma, n, m)
        I_out = three_dim_gamma_correction(I_d, weights, n, m)
        I_img = ((I_out/255.*I_p/255.)*255).astype(np.uint8)

        ##### S - Channel #####
        I_u = denoise_tv(s, weight=1/lam, eps=1e-6, max_num_iter=100)
        S_tag = exposure.equalize_adapthist(I_u/255.)
        S_tag = cv2.normalize(S_tag, None, 0, 255, cv2.NORM_MINMAX, dtype=cv2.CV_8UC3)

        ##### I_o #####
        I_o = HSVtoBGR(h, S_tag, I_img)
        I_o = HSVtoBGR(h, I_u, I_img)
        I_o_cb = color_balance(I_o, 1, 1)
        performance = metric(Img, I_o)
        if disp_selector[0]:
            disp_img(I_o, title = 'I_o', text = {'text': ['I_o'], 'loc': [(280, 460)]})
        if disp_selector[1]:
            disp_img(np.block([[Img], [I_o], [I_o_cb]]), title = 'Enhancement', text = {'text': ['Original', 'Enhancement', 'Enhancement + Color Balance'], 'loc': [(280, 460), (640+280, 460), (640*2+230, 460)]})
        if disp_selector[2]:
            disp_img(np.block([[h, s, v]]), title = 'HSV', text = {'text': ['h-channel', 's-channel', 'v-channel'], 'loc': [(280, 460), (640+280, 460), (640*2+280, 460)]})
        if disp_selector[3]:
            disp_img(I_d, title = 'I_d', text = {'text': ['I_d'], 'loc': [(280, 460)]})
        if disp_selector[4]:
            disp_img(I_p, title = 'I_p', text = {'text': ['I_p'], 'loc': [(280, 460)]})
        if disp_selector[5]:
            disp_img(I_out, title = 'I_out', text = {'text': ['I_out'], 'loc': [(280, 460)]})
        if disp_selector[6]:
            disp_img(I_img, title = 'I_img', text = {'text': ['I_img'], 'loc': [(280, 460)]})
        if disp_selector[7]:
            disp_img(I_u, title = 'I_u', text = {'text': ['I_u'], 'loc': [(280, 460)]})
        if disp_selector[8]:
            disp_img(I_v, title = 'I_v', text = {'text': ['I_v'], 'loc': [(280, 460)]})
        if disp_selector[9]:
            disp_img(S_tag, title = 'S_tag', text = {'text': ['S_tag'], 'loc': [(280, 460)]})
        return performance

    if model == 'MSRCR':
        I_o = msrccr(Img, sigma_scales=sigma)
        performance = metric(Img, I_o)
        if disp_selector[0]:
            disp_img(I_o, title = 'I_o', text = {'text': ['I_o'], 'loc': [(280, 460)]})
        if disp_selector[1]:
            disp_img(np.block([[Img], [I_o]]), title = 'Enhancement', text = {'text': ['Original', 'Enhancement'], 'loc': [(280, 460), (640+280, 460)]})

```

```

        return performance
    if model == 'CLAHE':
        I_o = CLAHE(Img)
        performance = metric(Img,I_o)
    if disp_selector[0]:
        disp_img(I_o, title = 'I_o' ,text = {'text' : ['I_o'],'loc':[(280,460)]})
    if disp_selector[1]:
        disp_img(np.block([[Img],[I_o]]), title = 'Enhancement' ,text = {'text' :
        ['Original','Enhancement'],'loc':[(280,460),(640+280,460)]})

    return performance
if model == 'AHE':
    I_o = AHE(Img)
    performance = metric(Img,I_o)
    if disp_selector[0]:
        disp_img(I_o, title = 'I_o' ,text = {'text' : ['I_o'],'loc':[(280,460)]})
    if disp_selector[1]:
        disp_img(np.block([[Img],[I_o]]), title = 'Enhancement' ,text = {'text' :
        ['Original','Enhancement'],'loc':[(280,460),(640+280,460)]})

return performance

class ImageProcessorGUI:

    def __init__(self, master):
        self.master = master
        master.title("Image Processor")
        w = 380
        h = 370
        # open window in the center of screen
        screen_width = master.winfo_screenwidth() # get the screen width
        screen_height = master.winfo_screenheight() # get the screen height
        x = int((screen_width / 2) - (w / 2))
        y = int((screen_height / 2) - (h / 2))
        master.geometry('{x}x{y}+{x}+{y}'.format(w, h, x, y)) # window.geometry('wxh+x+y')

        # Select image button
        self.select_image_button = tk.Button(master, text="Select Image", command=self.select_image)
        self.select_image_button.grid(row=0, column=0, padx=10, pady=10)

        # Display Image button
        self.select_image_button = tk.Button(master, text="Display Selected",state='disabled', command= self.display_selected)
        self.select_image_button.grid(row=0, column=1, padx=10, pady=10)

        # Model combobox
        self.model_label = tk.Label(master, text="Select Model:")
        self.model_label.grid(row=1, column=0, padx=10, pady=5)

        self.model_combobox = ttk.Combobox(master, values=['MSRCR', 'CLAHE', 'AHE', 'DFE'],textvariable= 'Select',state='disabled')
        self.model_combobox.bind("<>ComboboxSelected>", self.en)
        self.model_combobox.grid(row=1, column=1, padx=10, pady=5)

        # Value entries
        self.values_label_sigma = tk.Label(master, text="Enter Sigma values :")
        self.values_label_sigma.grid(row=2, column=0, padx=10, pady=5)

        self.values_entry_sigma = tk.Entry(master,state='disabled')
        self.values_entry_sigma.grid(row=2, column=1, padx=10, pady=5 )

        self.values_label_weight = tk.Label(master, text="Enter Weight values :")
        self.values_label_weight.grid(row=3, column=0, padx=10, pady=5)

        self.values_entry_weight = tk.Entry(master,state='disabled')
        self.values_entry_weight.grid(row=3, column=1, padx=10, pady=5)

        self.values_label_kernel = tk.Label(master, text="Enter Kernel size (n,m) :")
        self.values_label_kernel.grid(row=4, column=0, padx=10, pady=5)

        self.values_entry_kernel= tk.Entry(master,state='disabled')
        self.values_entry_kernel.grid(row=4, column=1, padx=10, pady=5)

        self.values_label_lambda = tk.Label(master, text="Enter Lambda value :")
        self.values_label_lambda.grid(row=5, column=0, padx=10, pady=5)

        self.values_entry_lambda = tk.Entry(master,state='disabled')
        self.values_entry_lambda.grid(row=5, column=1, padx=10, pady=5)

```

```

# Checkboxes
self.checkbox_frame = tk.Frame(master)
self.checkbox_frame.grid(row=6, column=1, padx=10)

self.checkbox_labels = ['I_o', 'I_i&o', 'HSV', 'I_d', 'I_p', 'I_out', 'I_img', 'I_u', 'I_v', "S"]
self.checkbox_vars = [tk.BooleanVar() for i in range(len(self.checkbox_labels))]
self.checkbox_buttons = []

for i in range(len(self.checkbox_labels)):
    self.checkbox_buttons.append(tk.Checkbutton(self.checkbox_frame, text=self.checkbox_labels[i],
variable=self.checkbox_vars[i],state='disabled'))
    if i < len(self.checkbox_labels) // 2:
        self.checkbox_buttons[i].grid(row=i, column=0, sticky='w')
    else:
        self.checkbox_buttons[i].grid(row=i-len(self.checkbox_labels) // 2, column=1, sticky='w')

self.metric_table = ttk.Frame(master)
self.metric_table.grid(row=6, column=0, padx=5)
self.metric_table_label = ttk.Label(self.metric_table, text="Metric Table")
self.metric_table_label.grid(row=7, column=0, pady=(5, 5))

self.metric_table_treeview = ttk.Treeview(self.metric_table, height=5)
self.metric_table_treeview.grid(row=8, column=0)
self.metric_table_treeview['columns'] = ("Metric-Name", "Metric-Value")

# format columns
self.metric_table_treeview.column("#0", width=0, stretch=False)
self.metric_table_treeview.column("Metric-Name", width=100, minwidth=100, anchor="center")
self.metric_table_treeview.column("Metric-Value", width=100, minwidth=100, anchor="center")

# create headings
self.metric_table_treeview.heading("#0", text="", anchor="w")
self.metric_table_treeview.heading("Metric-Name", text="Metric-Name", anchor="center")
self.metric_table_treeview.heading("Metric-Value", text="Metric-Value", anchor="center")

# Run button
self.run_button = tk.Button(self.checkbox_frame, text="Run", state='disabled', command=self.run)
self.run_button.grid(row=7, column=0, padx=10, pady=5 )



def en(self, event):
    self.run_button.config(state='normal')
    if self.model_combobox.get() == 'MSRCR':
        self.values_entry_sigma.configure(state='normal')
        self.values_entry_sigma.delete(0,'end')
        self.values_entry_sigma.insert(1,'10,100,300')
        self.values_entry_weight.configure(state='disabled')
        self.values_entry_kernel.configure(state='disabled')
        self.values_entry_lambda.configure(state='disabled')
        for i in range(len(self.checkbox_buttons)):
            self.checkbox_buttons[i].configure(state='disabled')
            self.checkbox_buttons[i].deselect()
        self.checkbox_buttons[0].configure(state='normal')
        self.checkbox_buttons[1].configure(state='normal')

    elif self.model_combobox.get() == 'DFE':
        self.values_entry_sigma.configure(state='normal')
        self.values_entry_sigma.delete(0,'end')
        self.values_entry_sigma.insert(1,'10,100,300')
        self.values_entry_weight.configure(state='normal')
        self.values_entry_weight.delete(0,'end')
        self.values_entry_weight.insert(1,'0.05,0.1,0.25')
        self.values_entry_kernel.configure(state='normal')
        self.values_entry_kernel.delete(0,'end')
        self.values_entry_kernel.insert(1,'9,9')
        self.values_entry_lambda.configure(state='normal')
        self.values_entry_lambda.delete(0,'end')
        self.values_entry_lambda.insert(1,'40')
        for i in range(len(self.checkbox_buttons)):
            self.checkbox_buttons[i].configure(state='normal')

    else:
        self.values_entry_sigma.configure(state='disabled')
        self.values_entry_weight.configure(state='disabled')
        self.values_entry_kernel.configure(state='disabled')
        self.values_entry_lambda.configure(state='disabled')
        for i in range(len(self.checkbox_buttons)):
```

```

        self.checkbox_buttons[i].configure(state='disabled')
        self.checkbox_buttons[i].deselect()
        self.checkbox_buttons[0].configure(state='normal')
        self.checkbox_buttons[1].configure(state='normal')

    for i in self.metric_table_treeview.get_children():
        self.metric_table_treeview.delete(i)

    def select_image(self):
        file_paths = filedialog.askopenfilenames(title="Select Image Files", filetypes=[("Image files",
        "*.jpg;*.jpeg;*.png;*.gif;*.tif;*.bmp;")])
        if file_paths:
            image = cv2.imread(file_paths[0])
            self.image = cv2.resize(image, (640, 480))
            self.select_image_button.config(state='normal')
            self.model_combobox.config(state='readonly')
        else:
            self.image = None

    def display_selected(self):
        disp_img(self.image , title = 'Reference' ,text = {'text' : ['Original'],'loc':[(280,460)]})
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    def insert_data_to_metric_table(self):
        for i in self.metric_table_treeview.get_children():
            self.metric_table_treeview.delete(i)

        for idx, (key, value) in enumerate(self.info.items()):
            self.metric_table_treeview.insert(parent='', index='end', iid=str(idx), values=(key, value))

    def run(self):
        # Get values from GUI elements
        sigma = None
        weights = None
        kernel = None
        lam = None
        Img = self.image
        model = self.model_combobox.get()

        if model == 'DFE':
            sigma = [int(num)for num in self.values_entry_sigma.get().split(',')]
            weights = [float(num)for num in self.values_entry_weight.get().split(',')]
            kernel = [int(num)for num in self.values_entry_kernel.get().split(',')]
            lam = float(self.values_entry_lambda.get())
        elif model == 'MSRCR':
            sigma = [int(num)for num in self.values_entry_sigma.get().split(',')]

        checkboxes = [var.get() for var in self.checkbox_vars]

        # Display processed image
        self.info = Model(Img ,model =model ,disp_selector = checkboxes,sigma = sigma,weights=weights ,kernel =kernel ,lam = lam)
        self.insert_data_to_metric_table()
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    if __name__ == '__main__':
        root = tk.Tk()

        gui = ImageProcessorGUI(root)
        root.mainloop()

```

Appendix B – Script

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import dct, idct
import cv2
import pandas as pd
from skimage.metrics import structural_similarity as ssim
import pywt
from tabulate import tabulate
import tkinter.ttk
from tkinter import filedialog

images = None

#####
##### Functions #####
#####

def create_mosaic(images):
    # Decomposition plot function
    def img_norm(img):
        return img / img.max() # Normalize values

    if len(images) == 1: # if only one image in a list, return original image
        return images[0]

    elif len(images) <= 4: # if 4 images in a list, create mosaic
        block1 = np.block([[img_norm(images[0]), img_norm(images[2])], [img_norm(images[1]), img_norm(images[3])]])
        return block1

    else: # if more than 4 images, recursively apply this function on LL
        return np.block([[img_norm(create_mosaic(images[4:])), img_norm(images[2])], [img_norm(images[1]), img_norm(images[3])]])

#####

def MDCT(image, level=1):
    """
    Perform Multi Resolution DCT on an image.

    Parameters:
    image: ndarray
        The input image as a numpy array
    level: int, optional (default=1)
        The number of decomposition levels to perform.

    Returns:
    coeffs: list
        A list of numpy arrays containing the IDCT coefficients [LL_1,LH_1,HL_1,HH_1,LL_2,LH_2,HL_2,HH_2,etc...]
    """
    coeffs = []

    for i in range(level):
        # Perform DCT on columns
        dct_cols = dct(image, axis=0, type=2, norm='ortho')

        # Split into upper and lower halves along rows
        split_row = dct_cols.shape[0] // 2
        upper_half = dct_cols[:split_row, :]
        lower_half = dct_cols[split_row:, :]

        # Perform IDCT on each half along columns
        idct_upper = idct(upper_half, axis=0, type=2, norm='ortho')
        idct_lower = idct(lower_half, axis=0, type=2, norm='ortho')

        # Combine the halves and perform DCT on rows
        dct_rows_upper = dct(idct_upper, axis=1, type=2, norm='ortho')
        dct_rows_lower = dct(idct_lower, axis=1, type=2, norm='ortho')

        # Split into LL, LH, HL, HH coefficients
        split_col_u = dct_rows_upper.shape[1] // 2
        split_col_l = dct_rows_lower.shape[1] // 2
        LL_f = dct_rows_upper[:, :split_col_u]
        LH_f = dct_rows_upper[:, split_col_u:]
        HL_f = dct_rows_lower[:, :split_col_l]
        HH_f = dct_rows_lower[:, split_col_l:]
        LL = idct(LL_f, axis=1, type=2, norm='ortho')
        LH = idct(LH_f, axis=1, type=2, norm='ortho')
        HL = idct(HL_f, axis=1, type=2, norm='ortho')
        HH = idct(HH_f, axis=1, type=2, norm='ortho')

        coeffs.append(LL)
        coeffs.append(LH)
        coeffs.append(HL)
        coeffs.append(HH)

    return coeffs

```

```

HH = idct(HH_f, axis=1, type=2, norm='ortho')

# Append the coefficients to the list
coeffs.append(LL)
coeffs.append(LH)
coeffs.append(HL)
coeffs.append(HH)
# Update the image for the next level
image = LL.copy()
return coeffs


def IMDCT(coeff):
    """
    Perform Multi Resolution IDCT on a coefficient.

    Parameters:
    image: list of ndarray
        A list of numpy arrays containing the IDCT coefficients [LL_1,LH_1,HL_1,HH_1,LL_2,LH_2,HL_2,HH_2,etc...]

    Returns:
    LL: image as a numpy array

    """
    batches = []
    for i in range(0, len(coeff), 4): # split images list to batches for each level
        batches.append(coeff[i:i + 4])
    batches = batches[::-1] # revert list of batches

    def block_LL(LL_I, LH_I, HL_I, HH_I): # assembly block for one level
        LL = dct(LL_I, axis=1, type=2, norm='ortho')
        LH = dct(LH_I, axis=1, type=2, norm='ortho')
        HL = dct(HL_I, axis=1, type=2, norm='ortho')
        HH = dct(HH_I, axis=1, type=2, norm='ortho')
        block_up_col = idct(np.block([[LL, LH]]), axis=1, type=2, norm='ortho')
        block_low_col = idct(np.block([[HL, HH]]), axis=1, type=2, norm='ortho')
        block_up_row = dct(block_up_col, axis=0, type=2, norm='ortho')
        block_low_col = dct(block_low_col, axis=0, type=2, norm='ortho')
        block1 = idct(np.block([[block_up_row], [block_low_col]]), axis=0, type=2, norm='ortho')
        return block1

    LL = []
    if len(batches) == 1: # if only one level
        return block_LL(batches[0][0], batches[0][1], batches[0][2], batches[0][3])
    else: # if multiple levels, apply assembly recursively
        for i, val in enumerate(batches):
            if i == 0:
                LL.append(block_LL((val[0]), val[1], val[2], val[3]))
            else:
                LL.append(block_LL(LL[i - 1], val[1], val[2], val[3]))
    return LL[-1]


#####
# Wavelets #####
def MDWT(image, level, wavelet):
    """
    Perform Multi Resolution DWT on an image.

    Parameters:
    image: ndarray
        The input image as a numpy array
    level: int, optional (default=1)
        The number of decomposition levels to perform.
    wavelet: Mother wavelet
    Returns:
    coeffs: list
        A list of numpy arrays containing the IWCT coefficients [LL_1,LH_1,HL_1,HH_1,LL_2,LH_2,HL_2,HH_2,etc...]
    """
    coeffs = []
    for i in range(level):
        coeffs2 = pywt.dwt2(image, wavelet)
        LL, (LH, HL, HH) = coeffs2
        # Append the coefficients to the list
        coeffs.append(LL)
        coeffs.append(LH)
        coeffs.append(HL)
        coeffs.append(HH)
        # Update the image for the next level

```

```

image = LL.copy()

return coeffs


def IMDWT(coeff, wavelet):
    """
    Perform Multi Resolution IDWT on a coefficient.

    Parameters:
    image: list of ndarray
        A list of numpy arrays containing the IDCT coefficients [LL_1,LH_1,HL_1,HH_1,LL_2,LH_2,HL_2,HH_2,etc...]
    wavelet: Mother wavelet
    Returns:
        LL: image as a numpy array
    """

    batches = []
    for i in range(0, len(coeff), 4):
        batches.append(coeff[i:i + 4])
    batches = batches[::-1]

    def block_LL(LL_I, LH_I, HL_I, HH_I, wavelet):
        return pywt.idwt2((LL_I, (LH_I, HL_I, HH_I)), wavelet)

    LL = []
    if len(batches) == 1:
        return block_LL(batches[0][0], batches[0][1], batches[0][2], batches[0][3], wavelet)
    else:
        for i, val in enumerate(batches):
            if i == 0:
                LL.append(block_LL((val[0]), val[1], val[2], val[3], wavelet))
            else:
                LL.append(block_LL(LL[i - 1], val[1], val[2], val[3], wavelet))
    return LL[-1]


def fusion_dct(img_1, img_2, level, details, approx, display_decomposition_mosaic=False, display_fusion_mosaic=False, display_fusion=False, err=False):
    def get_max_magnitude_pixel(A, B):
        # Ensure images are of the same shape
        assert A.shape == B.shape
        A = A.astype(np.float64)
        B = B.astype(np.float64)
        # Calculate the magnitude of each pixel in the images
        mag1 = np.abs(A)
        mag2 = np.abs(B)

        # Determine which image has the higher magnitude pixel for each location
        higher_mag_mask = mag1 > mag2

        # Combine the masks with the original images to obtain the maximum magnitude pixels
        max_pixels = np.zeros_like(A)
        max_pixels[higher_mag_mask] = A[higher_mag_mask]
        max_pixels[~higher_mag_mask] = B[~higher_mag_mask]

        # Return the maximum magnitude pixels
        return max_pixels

    def get_min_magnitude_pixel(A, B):
        # Ensure images are of the same shape
        assert A.shape == B.shape
        A = A.astype(np.float64)
        B = B.astype(np.float64)
        # Calculate the magnitude of each pixel in the images
        mag1 = np.abs(A)
        mag2 = np.abs(B)

        # Determine which image has the smaller magnitude pixel for each location
        lower_mag_mask = mag1 < mag2

        # Combine the masks with the original images to obtain the minimum magnitude pixels
        min_pixels = np.zeros_like(A)
        min_pixels[lower_mag_mask] = A[lower_mag_mask]
        min_pixels[~lower_mag_mask] = B[~lower_mag_mask]

        # Return the minimum magnitude pixels
        return min_pixels

```

```

def mouse_callback(event, x, y, flags, param):
    # Function that response for closing window of figure by clicking on right button and on X
    if event == cv2.EVENT_RBUTTONDOWN:
        cv2.destroyAllWindows()

coeff = []
coeffs_1 = MDCT(img_1, level)
coeffs_2 = MDCT(img_2, level)
for i, val in enumerate(coeffs_1): # Fusion
    if i % 4 == 0: # Approximation coefficients in the list
        if approx == 'min':
            coeff.append(get_min_magnitude_pixel(coeffs_2[i], coeffs_1[i]))
        elif approx == 'max':
            coeff.append(get_max_magnitude_pixel(coeffs_2[i], coeffs_1[i]))
        else:
            mean_coeff = (coeffs_1[i] + coeffs_2[i]) / 2
            coeff.append(mean_coeff)
    else: # Details coefficients in the list
        if details == 'min':
            coeff.append(get_min_magnitude_pixel(coeffs_2[i], coeffs_1[i]))
        elif details == 'max':
            coeff.append(get_max_magnitude_pixel(coeffs_2[i], coeffs_1[i]))
        elif details == 'mean':
            # coeff.append(np.mean(np.array([coeffs_1[i],coeffs_2[i]]),axis=0))
            mean_coeff = (coeffs_1[i] + coeffs_2[i]) / 2
            coeff.append(mean_coeff)
        else:
            print("Invalid metric for details")
MDCT1 = create_mosaic(coeffs_1)
MDCT2 = create_mosaic(coeffs_2)
MDCT_FUSION = create_mosaic(coeff)
IMDCT_FUSION = IMDCT(coeff)

if display_decomposition_mosaic:
    combined_i = cv2.hconcat([MDCT1, MDCT2])
    cv2.putText(combined_i, 'IDCT_IMG1', (220, 480), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    cv2.putText(combined_i, 'IDCT_IMG2', (MDCT1.shape[1] + 220, 480), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    cv2.imshow(f'Decomposition level: {level}', combined_i)
    cv2.setWindowProperty(f'Decomposition level: {level}', cv2.WND_PROP_TOPMOST, 1)
    # Associate the callback function with the named window
    cv2.setMouseCallback(f'Decomposition level: {level}', mouse_callback)
if display_fusion_mosaic:
    cv2.putText(MDCT_FUSION, f'Approx: {approx} Details: {details}', (165, 500), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    cv2.imshow(f'Decomposition of Fusion ,level: {level}', MDCT_FUSION)
    cv2.setWindowProperty(f'Decomposition of Fusion ,level: {level}', cv2.WND_PROP_TOPMOST, 1)
    # Associate the callback function with the named window
    cv2.setMouseCallback(f'Decomposition of Fusion ,level: {level}', mouse_callback)
if display_fusion:
    cv2.putText(IMDCT_FUSION, f'Approx: {approx} Details: {details}', (165, 500), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.imshow('Fusion', IMDCT_FUSION.copy() / 255)
    cv2.setWindowProperty('Fusion', cv2.WND_PROP_TOPMOST, 1)
    # Associate the callback function with the named window
    cv2.setMouseCallback('Fusion', mouse_callback)
if not err:
    cv2.waitKey(0)
    cv2.destroyAllWindows()
return IMDCT(coeff), coeff, MDCT1, MDCT2, create_mosaic(coeff)

def fusion_dwt(img_1, img_2, level, wavelet, details, approx, display_decomposition_mosaic=False, display_fusion_mosaic=False, display_fusion=False, err=False):
    def get_max_magnitude_pixel(A, B):
        # Ensure images are of the same shape
        assert A.shape == B.shape
        A = A.astype(np.float64)
        B = B.astype(np.float64)
        # Calculate the magnitude of each pixel in the images
        mag1 = np.abs(A)
        mag2 = np.abs(B)

        # Determine which image has the higher magnitude pixel for each location
        higher_mag_mask = mag1 > mag2

        # Combine the masks with the original images to obtain the maximum magnitude pixels
        max_pixels = np.zeros_like(A)
        max_pixels[higher_mag_mask] = A[higher_mag_mask]
        max_pixels[~higher_mag_mask] = B[~higher_mag_mask]

        # Return the maximum magnitude pixels
    return max_pixels

```

```

return max_pixels

def get_min_magnitude_pixel(A, B):
    # Ensure images are of the same shape
    assert A.shape == B.shape
    A = A.astype(np.float64)
    B = B.astype(np.float64)
    # Calculate the magnitude of each pixel in the images
    mag1 = np.abs(A)
    mag2 = np.abs(B)

    # Determine which image has the smaller magnitude pixel for each location
    lower_mag_mask = mag1 < mag2

    # Combine the masks with the original images to obtain the minimum magnitude pixels
    min_pixels = np.zeros_like(A)
    min_pixels[lower_mag_mask] = A[lower_mag_mask]
    min_pixels[~lower_mag_mask] = B[~lower_mag_mask]

    # Return the minimum magnitude pixels
    return min_pixels

def mouse_callback(event, x, y, flags, param):
    if event == cv2.EVENT_RBUTTONDOWN:
        cv2.destroyAllWindows()

coeff = []
coeffs_1 = MDWT(img_1, level, wavelet)
coeffs_2 = MDWT(img_2, level, wavelet)
for i, val in enumerate(coeffs_1):
    if i % 4 == 0:
        if approx == 'min':
            coeff.append(get_min_magnitude_pixel(coeffs_1[i], coeffs_2[i]))
        elif approx == 'max':
            coeff.append(get_max_magnitude_pixel(coeffs_1[i], coeffs_2[i]))
        else:
            mean_coeff = (coeffs_1[i] + coeffs_2[i]) / 2
            coeff.append(mean_coeff)
    else:
        # Compare the magnitudes of the matrices using the specified metric
        if details == 'min':
            coeff.append(get_min_magnitude_pixel(coeffs_1[i], coeffs_2[i]))
        elif details == 'max':
            coeff.append(get_max_magnitude_pixel(coeffs_1[i], coeffs_2[i]))
        elif details == 'mean':
            # coeff.append(np.mean(np.array([coeffs_1[i], coeffs_2[i]]), axis=0))
            mean_coeff = (coeffs_1[i] + coeffs_2[i]) / 2
            coeff.append(mean_coeff)
        else:
            print("Invalid metric for details")
MDWT1 = create_mosaic(coeffs_1)
MDWT2 = create_mosaic(coeffs_2)
MDWT_FUSION = create_mosaic(coeff)
IMDWT_FUSION = IMDWT(coeff, wavelet)
if display_decomposition_mosaic:
    combined_i = cv2.hconcat([MDWT1, MDWT2])
    cv2.putText(combined_i, 'IDWT_IMG1', (220, 480), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    cv2.putText(combined_i, 'IDWT_IMG2', (MDWT1.shape[1] + 220, 480), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    cv2.imshow('Decomposition level: {level}', combined_i)
    cv2.setWindowProperty('Decomposition level: {level}', cv2.WND_PROP_TOPMOST, 1)
    cv2.setMouseCallback('Decomposition level: {level}', mouse_callback)
if display_fusion_mosaic:
    cv2.putText(MDWT_FUSION, 'Mother Wavelet: {wavelet} Approx: {approx} Details: {details}', (70, 500), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
    cv2.imshow('Decomposition of Fusion ,level: {level}', MDWT_FUSION)
    cv2.setWindowProperty('Decomposition of Fusion ,level: {level}', cv2.WND_PROP_TOPMOST, 1)
    # Associate the callback function with the named window
    cv2.setMouseCallback('Decomposition of Fusion ,level: {level}', mouse_callback)
if display_fusion:
    cv2.putText(IMDWT_FUSION, 'Approx: {approx} Details: {details}', (165, 500), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 1)
    cv2.imshow('Fusion', IMDWT_FUSION / 255.)
    cv2.setWindowProperty('Fusion', cv2.WND_PROP_TOPMOST, 1)
    # Associate the callback function with the named window
    cv2.setMouseCallback('Fusion', mouse_callback)
if not err:
    cv2.waitKey(0)
    cv2.destroyAllWindows()
return IMDWT(coeff, wavelet), coeff, MDWT1, MDWT2, create_mosaic(coeff)

```

```

# metrics
def PFE(I_r, I_f):
    norm_diff = np.linalg.norm(I_r - I_f)
    norm_I_r = np.linalg.norm(I_r)
    pfe = (norm_diff / norm_I_r) * 100
    return round(pfe, 4)

def PSNR(I_r, I_f):
    mse = np.mean((I_r - I_f) ** 2)
    max_pixel = 255
    psnr = 10 * np.log10(max_pixel ** 2 / mse)
    # psnr = 20 * np.log10(max_pixel/np.sqrt(mse))
    return round(psnr, 4)

def SD(I_f):
    # Compute the histogram
    hist, bins = np.histogram(I_f.flatten(), bins=256)
    # Compute the mean of the histogram
    mean = np.sum(hist * bins[:-1]) / np.sum(hist)

    # Compute the variance of the histogram
    variance = np.sum((bins[:-1] - mean) ** 2 * hist) / np.sum(hist)

    # Compute the standard deviation of the histogram
    return round(np.sqrt(variance), 4)

def MSSISM(I_r, I_f, L=255):
    K1 = 0.01
    K2 = 0.03
    C1 = (K1 * L) ** 2
    C2 = (K2 * L) ** 2
    # INITIS
    I2_2 = I_f ** 2 # I2^2
    I1_2 = I_r ** 2 # I1^2
    I1_I2 = I_r * I_f # I1 * I2
    # END INITIS
    # PRELIMINARY COMPUTING
    mu1 = cv2.GaussianBlur(I_r, (11, 11), 1.5)
    mu2 = cv2.GaussianBlur(I_f, (11, 11), 1.5)
    mu1_2 = mu1 ** 2
    mu2_2 = mu2 ** 2
    mu1_mu2 = mu1 * mu2
    sigma1_2 = cv2.GaussianBlur(I1_2, (11, 11), 1.5)
    sigma1_2 -= mu1_2
    sigma2_2 = cv2.GaussianBlur(I2_2, (11, 11), 1.5)
    sigma2_2 -= mu2_2
    sigma12 = cv2.GaussianBlur(I1_I2, (11, 11), 1.5)
    sigma12 -= mu1_mu2
    t1 = 2 * mu1_mu2 + C1
    t2 = 2 * sigma12 + C2
    t3 = t1 * t2 # t3 = ((2*mu1_mu2 + C1).*(2*sigma12 + C2))
    t1 = mu1_2 + mu2_2 + C1
    t2 = sigma1_2 + sigma2_2 + C2
    t1 = t1 * t2 # t1 = ((mu1_2 + mu2_2 + C1).*(sigma1_2 + sigma2_2 + C2))
    ssim_map = t3 / t1
    mssim = np.mean(ssim_map) # mssim = average of ssim map windows 11x11
    return round(mssim, 4)

def CE(I1, I2, I_f):
    def calculate_pmf(image):
        # Compute the histogram of the image
        hist, _ = np.histogram(image.flatten(), bins=256)

        # Calculate the total number of pixels in the image
        num_pixels = np.sum(hist) # same as N*M

        # Calculate the PMF by dividing each bin in the histogram by the total number of pixels
        pmf = hist / num_pixels

        return pmf

    def crossEntropy(Y, P):
        epsilon = 2**(-32)
        hist_y = calculate_pmf(Y)

```

```

hist_p = calculate_pmf(P)
# Add epsilon to avoid division by zero errors
hist_y = np.clip(hist_y, epsilon, 1)
hist_p = np.clip(hist_p, epsilon, 1)
CE = np.sum(hist_y * np.log(hist_y/hist_p))
return CE

return round((crossEntropy(I1, I_f) + crossEntropy(I2, I_f)) * 0.5, 4)

def SF(I_f):
    def RF(X):
        result = []
        for i in range(len(X)):
            row = []
            for j in range(1, len(X[i])):
                row.append((X[i][j] - X[i][j - 1]) ** 2)
            result.append(row)
        return np.sum(result) / (X.shape[0] * X.shape[1])

    def CF(X):
        result = []
        for i in range(1, len(X)):
            row = []
            for j in range(len(X[i])):
                row.append((X[i][j] - X[i - 1][j]) ** 2)
            result.append(row)
        return np.sum(result) / (X.shape[0] * X.shape[1])

    sf = np.sqrt(RF(I_f) + CF(I_f))
    return round(sf, 4)

def dict_to_table(data, fusion, level):
    # Transform Dictionary to Table form
    df = pd.DataFrame(data.items(), columns=['Fusion: ' + fusion + ' Level: ' + str(level), 'Metric Value'])
    table = tabulate(df, headers='keys', tablefmt='psql', showindex=False)
    print(table)

def fusion(images, level, fusion, wavelet, details, approx, display_decomposition_mosaic=False, display_fusion_mosaic=False,
          display_fusion=False, display_error=False, calculate_metrics=False):
    # Main Fusion function that combine all previous functions and activate them according to inputs
    def mouse_callback(event, x, y, flags, param):
        if event == cv2.EVENT_RBUTTONDOWN:
            cv2.destroyAllWindows()

    img_1 = images[0]
    img_2 = images[1]
    if len(images) > 2:
        reference_img = images[2] # Reference image

    if fusion == 'DCT':
        out, _, _, _ = fusion_dct(img_1, img_2, level, details, approx, display_decomposition_mosaic, display_fusion_mosaic,
                                   display_fusion, err=display_error)
    elif fusion == 'DWT':
        out, _, _, _ = fusion_dwt(img_1, img_2, level, wavelet, details, approx, display_decomposition_mosaic, display_fusion_mosaic,
                                   display_fusion, err=display_error)
    else:
        print('Error')

    if display_error:
        # Plot error Image
        cv2.imshow('Error', (out - reference_img) / 255.)
        cv2.setMouseCallback('Error', mouse_callback)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    # Perform Metrics
    if calculate_metrics:
        info_no_ref = {'CE': CE(img_1, img_2, out), 'SD': SD(out), 'SF': SF(out)}
        if len(images) > 2:
            info_ref = {'PFE': PFE(reference_img, out), 'PSNR': PSNR(reference_img, out),
                        'SSIM': MSSISM(reference_img, out, L=255)}
            info = {**info_ref, **info_no_ref}
        else:
            info = info_no_ref
        dict_to_table(info, fusion, level)

```

```

        return info

##### GUI #####
def mouse_callback(event, x, y, flags, param):
    if event == cv2.EVENT_RBUTTONDOWN:
        cv2.destroyAllWindows()

def select_images_from_folder():
    global images
    file_paths = filedialog.askopenfilenames(title="Select Image Files", filetypes=[("Image files", "*.jpg;*.jpeg;*.png;*.gif")])
    print(file_paths)
    images = []
    if file_paths:
        for file_path in file_paths:
            image = cv2.imread(file_path)
            image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            image = cv2.resize(image, (512, 512))
            image = image.astype(np.float64)
            images.append(image)
    enable_btn()
    return images

def start_fusion(display_fusion=False, display_fusion_mosaic=False, display_decomposition_mosaic=False, display_error=False,
calculate_metrics=False):
    global images

    if len(images) >= 2:
        details_param = details_combo.get()
        approx_param = approximation_combo.get()
        dwt_param = dwt_selected_combo.get()
        fusion_param = fusion_model_combo.get()
        level_param = int(decomposition_level_scale.get())

        info = fusion(images, level_param, fusion_param, dwt_param, details_param, approx_param, display_fusion, display_fusion_mosaic,
display_decomposition_mosaic, display_error, calculate_metrics)
        insert_data_to_metric_table(info)

def display_selected_images():
    global images
    if len(images) > 2:
        image_1 = images[0]
        image_2 = images[1]
        reference_img = images[2]
        cv2.imshow('Unfocudes and Reference Images', cv2.hconcat([image_1, image_2, reference_img]) / 255.)
        cv2.setMouseCallback('Unfocudes and Reference Images', mouse_callback)

    elif len(images) == 2:
        image_1 = images[0]
        image_2 = images[1]
        cv2.imshow('Unfocudes Images', cv2.hconcat([image_1, image_2]) / 255.)
        cv2.setMouseCallback('Unfocudes Images', mouse_callback)
    else:
        print('Select at least two image')
        cv2.waitKey(0)
        cv2.destroyAllWindows()

def enable_btn():
    global images
    if len(images) < 2 or len(images) > 3:
        display_error_btn.config(state='disabled')
        display_selected_images_btn.config(state='disabled')
        display_calculate_metrics_btn.config(state='disabled')
        display_decomposition_btn.config(state='disabled')
        display_fusion_btn.config(state='disabled')
        display_fusion_decomposition_btn.config(state='disabled')

    elif len(images) == 2:
        display_error_btn.config(state='disabled')
        display_selected_images_btn.config(state='normal')
        display_calculate_metrics_btn.config(state='normal')
        display_decomposition_btn.config(state='normal')
        display_fusion_btn.config(state='normal')
        display_fusion_decomposition_btn.config(state='normal')
    else:
        display_error_btn.config(state='normal')

```

```

display_selected_images_btn.config(state='normal')
display_calculate_metrics_btn.config(state='normal')
display_decomposition_btn.config(state='normal')
display_fusion_btn.config(state='normal')
display_fusion_decomposition_btn.config(state='normal')

def display_decomposition():
    print("display decomposition")

def update_decomposition_label(event):
    val = int(decomposition_level_scale.get())
    decomposition_level_label.config(text=str(val))

def enable_kernel_selection(event):
    current_fusion_model = fusion_model_combo.get()

    if current_fusion_model == "DWT":
        dwt_selected_combo.config(state="readonly")
    else:
        dwt_selected_combo.config(state="disabled")

def display_fusion():
    print("display fusion")

def insert_data_to_metric_table(info):

    if info is not None:
        for i in metric_table_treeview.get_children():
            metric_table_treeview.delete(i)

        for idx, (key, value) in enumerate(info.items()):
            metric_table_treeview.insert(parent='', index='end', iid=str(idx), values=(key, value))

    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == '__main__':
    # Main loop
    window = tkinter.Tk()
    window.title("Discrete Cosine Transform-based Image Fusion")
    w = 520
    h = 440
    # limit the window (to be un-resizable)
    window.minsize(w, h)
    window.maxsize(w, h)
    # open window in the center of screen
    screen_width = window.winfo_screenwidth() # get the screen width
    screen_height = window.winfo_screenheight() # get the screen height
    x = int((screen_width / 2) - (w / 2))
    y = int((screen_height / 2) - (h / 2))
    window.geometry('{0}x{1}+{2}+{3}'.format(w, h, x, y)) # window.geometry('wxh+x+y')

    frame0 = tkinter.ttk.Frame(window)
    frame0.grid(row=0, column=0, padx=10, pady=(30, 15), sticky="w")

    frame1 = tkinter.ttk.Frame(window)
    frame1.grid(row=1, column=0, padx=10, pady=15, sticky="w")

    frame2 = tkinter.ttk.Frame(window)
    frame2.grid(row=2, column=0, padx=10, pady=15, sticky="w")

    frame3 = tkinter.ttk.Frame(window)
    frame3.grid(row=3, column=0, padx=10)

    select_images_btn = tkinter.ttk.Button(frame0, text="Select Images\n From Folder", command=select_images_from_folder)
    select_images_btn.grid(row=0, column=0, padx=7)

    fusion_label = tkinter.ttk.Label(frame0, text="Fusion Model: ")
    fusion_label.grid(row=0, column=1, padx=1)

    fusion_values = ["DCT", "DWT"]
    fusion_model_combo = tkinter.ttk.Combobox(frame0, values=fusion_values, width=6, justify='left', state="readonly")

```

```

fusion_model_combo.grid(row=0, column=2, padx=7)
fusion_model_combo.set(fusion_values[0])
fusion_model_combo.bind('<>ComboboxSelected>', enable_kernel_selection)

decomposition_label = tkinter.ttk.Label(frame0, text="Decomposition Level: ")
decomposition_label.grid(row=0, column=3, padx=1)

# decomposition levels (scale widget)
decomposition_level_scale = tkinter.ttk.Scale(frame0, from_=1, to=8, orient="horizontal", command=update_decomposition_label)
decomposition_level_scale.grid(row=0, column=4)
decomposition_level_label = tkinter.ttk.Label(frame0, text='0')
decomposition_level_label.grid(row=0, column=5)
decomposition_level_scale.set(1)

display_selected_images_btn = tkinter.ttk.Button(frame1, text="Display\nSelected Images", state='disabled',
command=display_selected_images)
display_selected_images_btn.grid(row=0, column=0, padx=7)

kernel_label = tkinter.ttk.Label(frame1, text="Kernel: ")
kernel_label.grid(row=0, column=1, padx=1)

dwt_selected_values = ["db1", "haar"]
dwt_selected_combo = tkinter.ttk.Combobox(frame1, values=dwt_selected_values, width=6, justify='left', state="disabled")
dwt_selected_combo.grid(row=0, column=2, padx=7)
dwt_selected_combo.set(dwt_selected_values[0])

details_label = tkinter.ttk.Label(frame1, text="Details: ")
details_label.grid(row=0, column=3, padx=1)

details_values = ["max", "min", "mean"]
details_combo = tkinter.ttk.Combobox(frame1, values=details_values, width=5, state="readonly")
details_combo.grid(row=0, column=4, padx=7)
details_combo.set(details_values[0])

approximation_label = tkinter.ttk.Label(frame1, text="Approximation: ")
approximation_label.grid(row=0, column=5, padx=1)

approximation_values = ["mean", "min", "max"]
approximation_combo = tkinter.ttk.Combobox(frame1, values=approximation_values, width=5, state="readonly")
approximation_combo.grid(row=0, column=6, padx=7)
approximation_combo.set(approximation_values[0])

display_decomposition_btn = tkinter.ttk.Button(frame2, text="Display\nDecomposition", state='disabled', command=lambda:
start_fusion(True, False, False, False, False))
display_decomposition_btn.grid(row=0, column=0, padx=7)

display_fusion_decomposition_btn = tkinter.ttk.Button(frame2, text="Display\nFusion Decomposition", state='disabled', command=lambda:
start_fusion(False, True, False, False, False))
display_fusion_decomposition_btn.grid(row=0, column=1, padx=7)

display_fusion_btn = tkinter.ttk.Button(frame2, text="Display\nFusion", state='disabled', command=lambda: start_fusion(False, False,
True, False, False))
display_fusion_btn.grid(row=0, column=2, padx=7)

display_error_btn = tkinter.ttk.Button(frame2, text="Display\nError", state='disabled', command=lambda: start_fusion(False, False, False,
True, False))
display_error_btn.grid(row=0, column=3, padx=7)

metric_table_label = tkinter.ttk.Label(frame3, text="Metric Table")
metric_table_label.grid(row=0, column=1, pady=(0, 7))

display_calculate_metrics_btn = tkinter.ttk.Button(frame3, text="Calculate\nMetrics", state='disabled', command=lambda:
start_fusion(False, False, False, False, True))
display_calculate_metrics_btn.grid(row=1, column=0, padx=1)

metric_table_treeview = tkinter.ttk.Treeview(frame3, height=7)
metric_table_treeview.grid(row=1, column=1)
metric_table_treeview['columns'] = ("Metric-Name", "Metric-Value")

# format columns
metric_table_treeview.column("#0", width=0, stretch=False)
metric_table_treeview.column("Metric-Name", width=150, minwidth=100, anchor="center")
metric_table_treeview.column("Metric-Value", width=150, minwidth=100, anchor="center")

# create headings
metric_table_treeview.heading("#0", text="", anchor="w")
metric_table_treeview.heading("Metric-Name", text="Metric-Name", anchor="center")
metric_table_treeview.heading("Metric-Value", text="Metric-Value", anchor="center")

window.mainloop()

```