נושא:

**תיקוני תמונה כתוצאה מתאורה לקויה**

**והיתוך תמונות לתיקון בעיות פוקוס חלקי**

מנחה:

אמיר הנדלמן

מגישים:   
רוסלן אוסמנוב - 327480026

**תוכן עניינים**

Table of Contents

[Double-function enhancement algorithm for low-illumination images based on Retinex theory 3](#_Toc132831399)

[מבוא 3](#_Toc132831400)

[BASIC THEORY 3](#_Toc132831401)

[Retinex Theory 3](#_Toc132831402)

[Discrete Cosine Transform – based Image Fusion 6](#_Toc132831403)

[מבוא 6](#_Toc132831404)

[התמרת קוסינוס בדידה 6](#_Toc132831405)

[התמרת קוסינוס בדידה רב רזולוציה 7](#_Toc132831406)

[היתוך 11](#_Toc132831407)

[הערכת ביצועים 11](#_Toc132831408)

[עם תמונת יחוס 11](#_Toc132831409)

[ללא תמונת יחוס 12](#_Toc132831410)

[תוצאות 13](#_Toc132831411)

[הסבר על שימוש בGUI 20](#_Toc132831412)

[Appendix A – Script 22](#_Toc132831413)

[Appendix B – Script 23](#_Toc132831414)

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

מאמר זה מציע אלגוריתם לשיפור תמונה באיכות גבוהה כדי לפתור את הבעיות של הגברת רעש ושיפור מוגזם הנגרים כתוצאה מניגודיות נמוכה ותאורה לא אחידה בתהליך של שיפור תמונה בתאורה עם עוצמה נמוכה. מודל הווריאציה הכוללת משמש להשגת תמונות ערוץ V ו-S מוחלקות, וטרנספורמציית גמא אדפטיבית משמשת לשיפור הפרטים בערוץ V המוחלק של התמונה. לאחר מכן משתמשים באלגוריתם משופר של Retinex רב סולם להשגת תמונות משופרות שונות של ערוצי V, ושתי התמונות מתמזגות בהתאם לאמפליטודת העוצמה המקומית של התמונות. לבסוף, פונקציית הגמא התלת ממדית משמשת כדי לתקן את התמונה הממוזגת ולהתאים את רווית התמונה. נעשה שימוש בגישת (LOE) למדידת הטבעיות (יעילות) של התמונה המשופרת.

## מבוא

יש שתי הקטגוריות של אלגוריתמים לשיפור תמונה בתאורה נמוכה: אלגוריתמים מסורתיים ולמידה עמוקה. אלגוריתמים מסורתיים כוללים מיפוי, השוואת היסטוגרמה, טרנספורמציה של Wavelets ותיאוריית Retinex. שיטות המיפוי משתמשות בפונקציות לא ליניאריות לעיבוד גלובלי של תמונות בתאורה נמוכה ומתאימות את משרעת השיפור של אזורים שונים כדי למנוע עיוות תמונה. השוואת היסטוגרמה יכולה להשיג את אפקט השיפור על ידי הרחבת הטווח הדינמי של התמונה. טרנספורמצית Wavelet יכולה לעבד גם את תחום התדר וגם את התחום המרחבי של התמונה בו-זמנית. תיאוריית ה-Retinex מחלקת תמונה לרכיב אירוע ורכיב השתקפות ומסירה את רכיב האירוע מהתמונה המקורית כדי לקבל את רכיב ההשתקפות. על פי מודלים שונים של נתיבים, ניתן לחלק אותו ל Retinex מבוססת נתיב אקראית, Retinex מבוססת ריבוי איטרציות ו Retinex מבוססת מרכז/הקיף.

מאמר זה מציע אלגוריתם מסורתי לשיפור תמונת תאורה נמוכה, תוך שימוש במודל הווריאציה הכוללת

(TV – Total Variation) כדי לטשטש את תמונת ערוץ ה-V ואת טרנספורמציה הגמא ההסתגלותית כדי לשפר את הפרטים של תמונת שכבת המבנה המתקבלת לאחר ביטול רעשים. פונקציית הטנגנס ההיפרבולית והפונקציה הלוגריתמית משולבות כדי לשלוט באמפליטודה של שיפור התמונה, בעוד שפונקציית הגמא התלת ממדית משמשת להתאמת התמונה המפורטת ולתיקון התמונה המשופרת. לבסוף, הרוויה של התמונה מותאמת על ידי סף אוטומטי כדי לקבל את האפקט הסופי.

## BASIC THEORY

### Retinex Theory

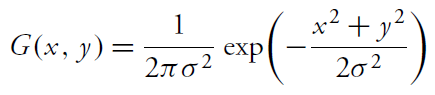
הרעיון המרכזי של Retinex הוא שתמונה מורכבת ממרכיב אירוע ומרכיב משתקף. עוצמת רכיב האירוע קובעת את הטווח הדינמי של פיקסל תמונה, בעוד הרכיב המשתקף הוא תכונה אינהרנטית של האובייקט ואינו משתנה עם רכיב האירוע. הדגם הוא כדלקמן:

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



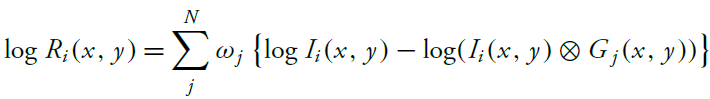
בהתבסס על תיאוריה זו, הוצע אלגוריתם SSR (single-scale retinex) קלאסי. פונקציית גרעין גאוס עוברת קונבולוציה עם התמונה המקורית כדי להעריך את רכיב האירוע, ורכיב ההשתקפות מתקבל כדי לממש את השיפור של תמונות בעלות תאורה נמוכה. הנוסחה היא כדלקמן:



כשאר גרעין גאוסיאני:

כאשר סטיית תקן של גאוס קטן, התמונה המשופרת שומרת יותר פרטים, אך היא עלולה לגרום בקלות לעיוות צבע בתמונה. סטיית תקן של גאוס גדול, התמונה המשופרת שומרת על צבע טוב, אך התמונה הופכת למטושטשת, ואפקט ההילה מתרחש בקלות באזור הבהיר יותר.

על מנת לפתור את הבעיות הנ"ל, הוצע אלגוריתם MSR(multi-scale retinex) לשיפור תמונות בהארה נמוכה בקנה מידה שונה, והביטוי שלו הוא כדלקמן:

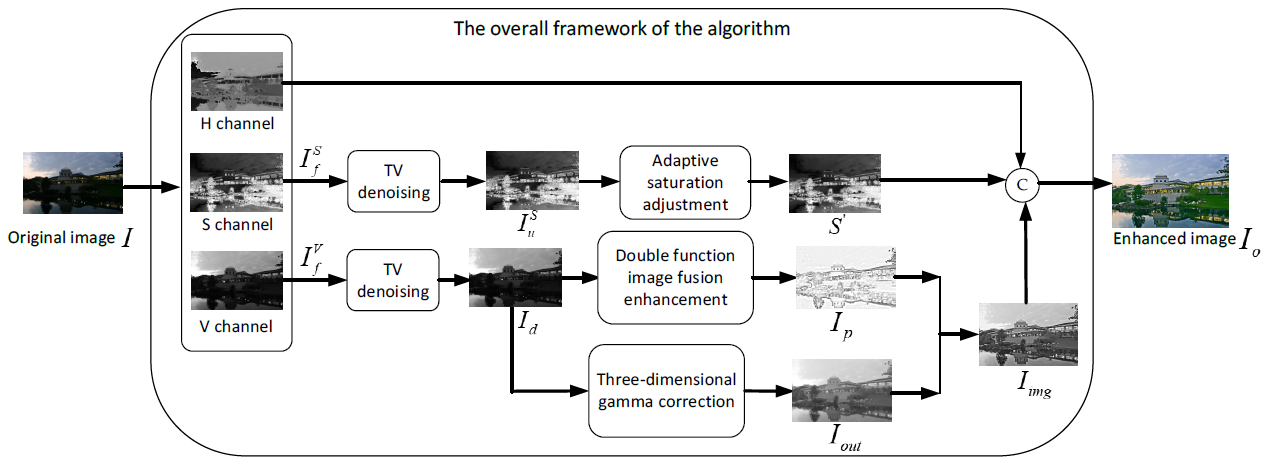


כאשר N=3 , המייצג את שלושת הסולמות,מייצג את הפרמטר הראשון של סולם j, ובאופן כללי

. מייצג את פונקציית הגרעין הגאוסית המתאימה עם סטיית תקן שונה.

## אלגוריתם

כדי להשיג ביצועי שיפור באיכות גבוהה ללא הגברה של רעש במצב תאורה נמוכה, אלגוריתם יעיל המבוסס על תיאוריית Retinex. תרשים הכולל של אלגוריתם שיפור התמונה מוצג באיור 1. האלגוריתם המוצע מורכב מיחידת TV denoising, יחידת התאמת רוויה אדפטיבית, יחידת שיפור היתוך תמונה כפולה ויחידה תלת ממדית לתיקון גמא. התמונה המקורית היא תמונת הקלט בפורמט RGB, אשר מומרת תחילה לפורמט HSV, ולאחר מכן מתקבלות תמונות ערוץ S וערוץ V. ו-מתקבלות באמצעות שיטות עיבוד שונות. לבסוף, תמונות שלושת הערוצים שהתקבלו מיוצרות מחדש לפורמט HSV ולאחר מכן מומרות לפורמט RGB. יש להחליק את תמונת ערוץ ה-V על ידי TV denoising כדי לקבל את התמונה , ואז הפרטים משופרים על ידי שיפור גמא אדפטיבי כדי לקבל את התמונה. יש להחליק את תמונת S-channel רק כדי לקבל את התמונה .



איור 1. תרשים זרימה של אלגוריתם

# 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 מאוד פשוט מבחינה חישובית ועשוי להתאים ליישומים בזמן אמת.

## מבוא

היתוך תמונה מרובה-חיישנים (MIF) היא טכניקה להגדלת הרזולוציה המרחבית של תמונות מרובי-חיישנים עם פרטים נמוכים תוך שמירה על המידע הספקטרלי שלהן על-ידי שילוב של שתי תמונות או יותר. MIF הופיע לאחרונה כנושא מחקר חדש ומבטיח לעיבוד תמונות. צבא, חישה מרחוק, ראיית מכונה, רובוטיקה, מעקב, מערכות ראייה משופרות והדמיה רפואית הם חלק מהענפים שנהנים מ-MIF.

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

כתוצאה מכך, התמונה הממוזגת תהיה עדיפה על כל אחת מתמונות המקור. MIF יכול להתבצע בשלוש רמות שונות, בהתאם לשלב המיזוג: רמת פיקסל, רמת תכונה ורמת החלטה. במחקר זה, מתואר MIF מבוסס רמת פיקסל, המתאר תהליך היתוך שיוצר תמונה ממוזגת אחת עם תיאור מדויק יותר מתמונת המקור הבודדות.

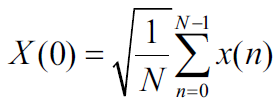
ה-MIF הפשוט ביותר הוא לקחת את הממוצע של פיקסלים של תמונות המקור ברמת האפור. שיטה זו תגרום לתוצאות לא רצויות שונות וניגודיות נמוכה יותר בתמונה הממוזגת. טרנספורמציות (התמרות) רבות כגון wavelet, פירמידות, תדר מרחבי, עיבוד אותות סטטיסטי ותאוריית הקבוצות המטושטשות הוצעו כדי לטפל בבעיות אלו. התמרת wavelet מרובות רזולוציות עשויות להיות שימושיות עבור לוקליזציה הן בתחום המרחבי והן בתחום התדר. טרנספורמציה(התמרה) waveletתציע מידע כיווני ברמות פירוק ומידע ייחודי ברזולוציות שונות.

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

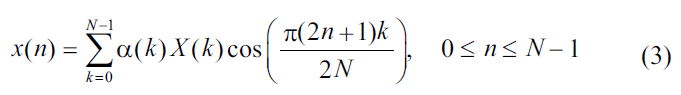
## התמרת קוסינוס בדידה

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

Text

Description automatically generated with medium confidence

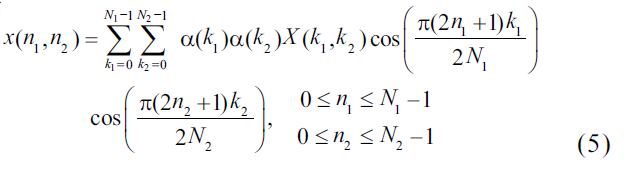
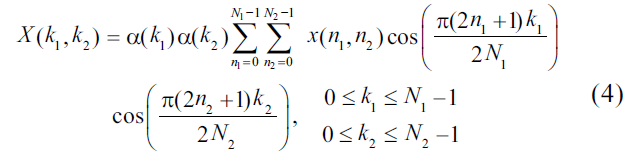
אפשר לראות שעבור k=0, המשוואה (1) הופכת למקדם הטרנספורמציה הראשון שהוא הממוצע של כל הדגימות ברצף והוא ידוע כמקדם DC, ומקדמי טרנספורמציה אחרים ידועים כמקדמי AC.

טרנספורמציה הקוסינוס ההפוכה מוגדרת כ:



כאשר הינה פונק' גרעין אמיתית ודיסקרטית.

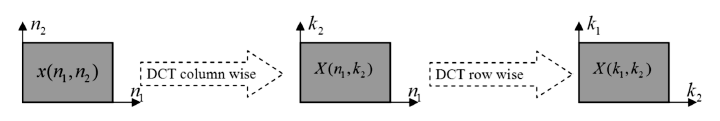
ה-2D DCT הוא הרחבה ישירה של 1D DCT. המרת הקוסינוס הבדיד הדו-ממדי X(k1,k2) של תמונה מוגדר כ:



והתמרה הפוכה:



DCT וגם IDCT הם טרנספורמציות אשר ניתנות להפרדה, היתרון של תכונה זו היא חישוב 2D DCT או IDCT ניתן לבצע בשני שלבים על ידי 1D DCT או IDCT 1D על עמודות ולאחר מכן על שורות של תמונה x(n1,n2) כפי שמוצג באיור 1.



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

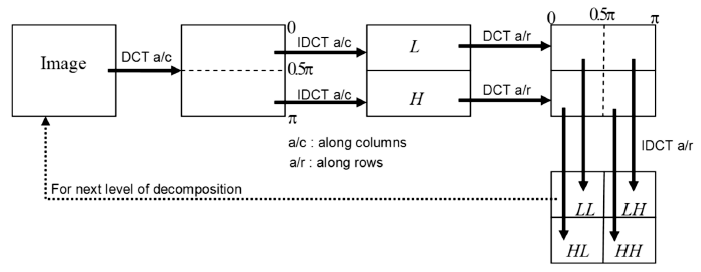
## התמרת קוסינוס בדידה רב רזולוציה

טרנספורמציה קוסינוס מרובת רזולוציה דיסקרטית (MDCT) היא טכניקת עיבוד אותות הדומה להתמרות wavelets. אשר מסנן את האות בנפרד על ידי מסנני תגובת תדר סופית (FIR) במעבר נמוך ובמעבר גבוה, ואז הפלט של כל מסנן מופחת (הורדת קצב 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 מסנני FIR ב-Discrete Cosine Transform (DCT). כדי לבצע MDCT, התמונה עוברת פירוק בתחום התדר על ידי יישום DCT באופן עמודה. מחצית הנקודות הראשונות (0 עד) נתונות ל-IDCT לקבלת את התמונה שעברה דרך מסנן נמוכים L, ומחצית השנייה ( עד ) נתונה ל-IDCT לקבלת את התמונה שעברה דרך מסנן גבוהיםH . התמונה שעברה דרך מסנן נמוכים L עוברת טרנספורמציה לתחום התדר על ידי יישום DCT בצורה של שורה. מחצית הנקודות הראשונות (באופן השורה) נתונות ל-IDCT כדי לקבל את התמונה LL שהועברה דרך מסנן נמוכים, ומחצית נקודות הנותרות נתונות ל-IDCT לקבלת את התמונה המועברת דרך מסנן גבוהים LH. התמונה שעברה דרך מסנן גבוהים H עוברת טרנספורמציה לתחום התדר על ידי יישום DCT באופן שורה. מחצית הנקודות הראשונות (באופן השורה) נתונות ל-IDCT הפוך כדי לקבל את התמונה HL שהועברה דרך מסנן נמוכים, ומחצית נקודות הנותרות נתונות ל-IDCT לקבלת את התמונה המועברת דרך מסנן גבוהים HH.

ה-LL מכיל את מידע התמונה הממוצע התואם לפס התדרים הנמוכים של הפירוק הרב-סקאלי. הוא מייצג את הגרסה המוחלקת ותת הדגימה של תמונת המקור ויכול להיחשב כקירוב של תמונת המקור(Approximations). התמונות LH, HL ו-HH הן תת-תמונות מפורטות המכילות מידע כיווני (אופקי, אנכי ואלכסוני) של תמונת המקור עקב כיוון מרחבי(Details). ניתן להשיג ריבוי רזולוציה על ידי יישום רקורסיבי של אותו אלגוריתם על מקדמי המעבר הנמוכים (LL) מהפירוק הקודם.

את אלגוריתם ניתן לראות באיור 2.



איור מס 2 פירוק רב רזולוציה בעזרת התמרה קוסינוסים בדידה DCT

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

**MDCT**

|  |  |  |
| --- | --- | --- |
| (c | (b | a) |
| * **MDCT** * 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) * \*תמונה מתוך המאמר | (e | (d |

|  |  |  |
| --- | --- | --- |
| (h | (g | f) |
| * **MDCT** * f- תמונה מקורית * g- פירוק בעל רמה 1 * h- פירוק בעל שני רמות * j- הרכבה מפירוק בעל רמה 2 * k- שגיאה בין תמונה מקורית לתמונה מורכבת (d) * \*תמונה שצולמה | Shape  Description automatically generated with low confidence  (k | (j |

**MDWT**

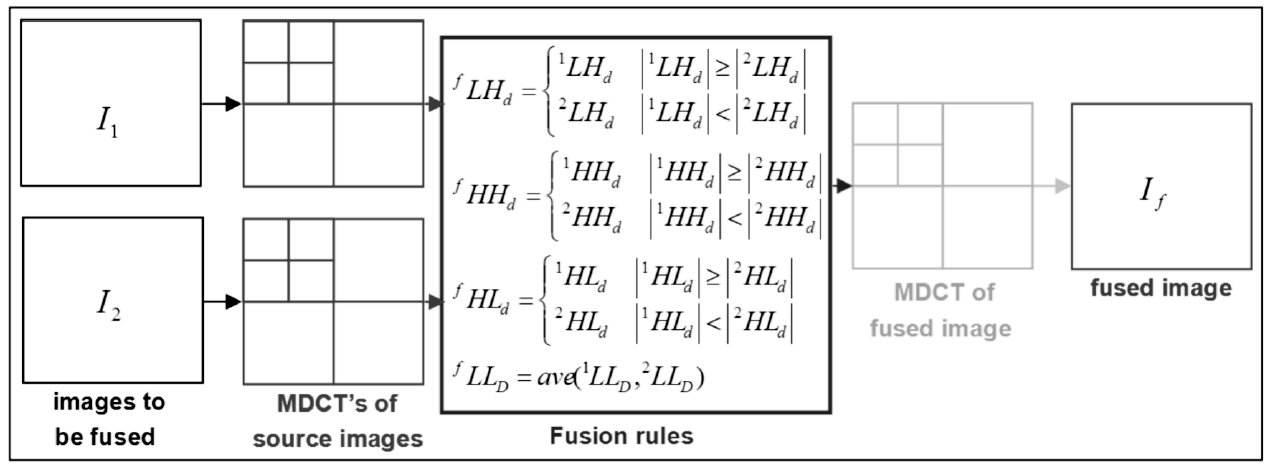
|  |  |  |
| --- | --- | --- |
| (c | (b | a) |
| * **MDWT** * a- תמונה מקורית * b- פירוק בעל רמה 1 * c- פירוק בעל שני רמות * d- הרכבה מפירוק בעל רמה 2 * e- שגיאה בין תמונה מקורית לתמונה מורכבת (d) | (e | (d |

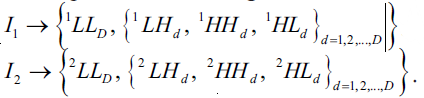
|  |  |  |
| --- | --- | --- |
| (h | (g | f) |
| * **MDWT** * f- תמונה מקורית * g- פירוק בעל רמה 1 * h- פירוק בעל שני רמות * j- הרכבה מפירוק בעל רמה 2 * k- שגיאה בין תמונה מקורית לתמונה מורכבת (d) | Shape  Description automatically generated with low confidence  (k | (j |

* בשני ההתמרות שגיאה מרבית בין תמונה מקורית לתמונה שעברה פירוק והרכבה , בסדר גודל של .

## היתוך

הדיאגרמה הסכמטית של היתוך תמונות על בסיס הפיקסל מבוסס MDCT מוצגת באיור 3. נשים לב שהשינוי של האלגוריתם הנוכחי הוא השימוש ב-MDCT במקום Wavelets או פירמידות. התמונות להיתוך I1 ו-I2 מפורקות לD רמות באמצעות MDCT.



איור מס 3 הדיאגרמה הסכמתית עבור היתוך תמונות ברמת פיקסלים מבוססת MDCT.

I1 ו-I2 מפורקות הינם:

בכל רמת פירוק (d = 1, 2,...,D), כלל ההיתוך יהיה

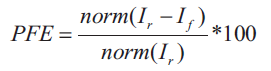
בחירת הערך המוחלט הגדול ביותר של מקדמי הפרטים (Details) ,מכוון שהמקדמים ה"פרטים" מתאימים לשינויי בהירות חדים יותר בתמונות כמו קצוות וגבולות אובייקט וכו'. מקדמים אלה נעים סביב האפס. ברמה הגסה ביותר (d=D), כלל ההיתוך לוקח את הממוצע של מקדמי "הקירוב"(Approximations) של MDCT שכן מקדמי הקירוב הם גסים יותר הרמה היא הגרסה המוחלקת ותת הדגימה של התמונה המקורית. לאחר היתוך מתבצע תהליך הרכבה ליצירת תמונה סופית אשר נמצאת בפוקוס מלא.

## הערכת ביצועים

### עם תמונת יחוס

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

1. Percentage fit error (PFE)

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

יהיה אפס מתי גם תמונות היחוס וגם תמונות מתמזגות דומות לחלוטין, יחס שגיאה תגדל כאשר התמונה הממוזגת שונה מתמונת היחוס.

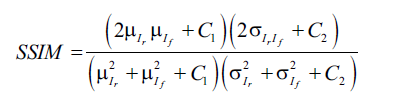
1. Peak signal to noise ratio (PSNR)

Text, schematic

Description automatically generatedכאשר L הוא מספר רמות האפור בתמונה. הערך של המטריקה תהיה בעלת ערך גבוה כאשר היתוך ותמונת יחוס דומות. ערך גבוה יותר מרמז על היתוך טוב יותר.

* יש טעות אמור להיות 10log ,אם משתמשים ב 20log נדרש להוציא את חזקת 2 מתוך ה.log

1. Measure of structural similarity (SSIM)

אותות של תמונות טבעיות יהיו מובנים מאוד והפיקסלים שלהם חושפים תלות חזקה. תלות זו יישאו מידע חיוני על מבנה האובייקט. מטריקה זו משווה תבניות מקומיות של עוצמות פיקסלים שעברו נורמליזציה עבור בהירות וניגודיות.

אני בחרתי לקחת חלון בגודל של 11x11 לחישוב ולאחר מכן כל התוצאות עוברו מיצעו לקבלת ערך בודד.

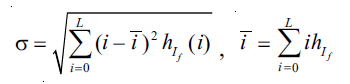
כאשר :

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

### ללא תמונת יחוס

כאשר תמונת יחוס אינה זמינה, ניתן להשתמש במדדים הבאים כדי לבדוק את הביצועים של האלגוריתמים.

1. Standard deviation (SD)

 היא היסטוגרמה המנורמלת של תמונה ממוזגת.

ידוע שסטיית תקן מורכבת מחלקי אות ורעש. מדד זה יהיה קטן יותר בהיעדר רעש. הוא מודד את הניגודיות בתמונה הממוזגת. לתמונה עם ניגודיות גבוהה תהיה סטיית תקן גבוהה.

1. Cross entropy (CE)

אנטרופיה צולבת כוללת של תמונות המקור I1, I2 , והתמונה ממוזגת היא:

|  |  |  |
| --- | --- | --- |
| Text  Description automatically generated | A picture containing text  Description automatically generated | A picture containing logo  Description automatically generated |

1. Spatial frequency (SF)

|  |  |  |
| --- | --- | --- |
| תדירי עמודה | תדירי השורה | קריטריון תדר מרחבי |

תדירות זו בתחום המרחבי מציינת את רמת הפעילות הכוללת בתמונה הממוזגת. (x, y) הוא אינדקס הפיקסלים.

## תוצאות

התוצאות הבאות הן עבור תמונה ממאמר עם תמונת יחוס (תהיה שוני במטריקות ממה שמוצאים במאמר מכוון שלא מצאתי תמונות אלו באינטרנט וביצעתי צילום מסך) ובנוסף תמונות נוספות שצולמו עם מוקד 2.8f

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reference\_Image | Image\_2 | | Image\_1 | |
|  | | | | |
| Error\_Image | | Fused\_Image | | MDCT Level 1 |
|  | |  | |
|  | |  | | MDWT Level 1 |
|  | |  | | MDCT Level 2 |
|  | |  | | MDWT Level 2 |
|  | |  | | MDCT Level 8 |
|  | |  | | MDWT Level 8 |

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

מסקנות:

ניתן לראות כי התמונות המאוחדות בעזרתMDCT ו wavelet הן כמעט דומות עבור תמונות אלה. מדדי הביצועים להערכת אלגוריתמי היתוך התמונה מוצגים בטבלה. המדדים המוצגים בטבלה עם צבע ירוק טובים יותר מאחרים. הביצועים של MDCT כמעט דומים לאלו של wavelets. רמה גבוהה יותר של פירוק גוררת היפוך ביצועים ,לפי 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).

|  |  |  |
| --- | --- | --- |
| Image\_2 | Image\_1 |  |
|  | | Original Images |

|  |  |  |  |
| --- | --- | --- | --- |
| Level 8 Fusion | Level 2 Fusion | Level 1 Fusion |  |
|  |  |  | MDCT |
|  |  |  | MDWT |

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

|  |  |  |
| --- | --- | --- |
| Image\_2 | Image\_1 |  |
|  | | Original Images |

|  |  |  |  |
| --- | --- | --- | --- |
| Level 8 Fusion | Level 2 Fusion | Level 1 Fusion |  |
|  |  |  | MDCT |
|  |  |  | MDWT |

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

עוד תמונות עבור DCT בלבד:

|  |  |  |
| --- | --- | --- |
| Image\_2 | Image\_1 |  |
|  | | Original Images |

|  |  |  |  |
| --- | --- | --- | --- |
| Level 8 Fusion | Level 2 Fusion | Level 1 Fusion |  |
|  |  |  | MDCT |

|  |  |  |
| --- | --- | --- |
| Image\_2 | Image\_1 |  |
|  | | Original Images |

|  |  |  |  |
| --- | --- | --- | --- |
| Level 8 Fusion | Level 2 Fusion | Level 1 Fusion |  |
|  |  |  | MDCT |

**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 לזה של wavelets. זה מאוד פשוט מבחינה חישובית והוא יכול להתאים היטב ליישומים בזמן אמת. היתוך תמונה על ידי רמה גבוהה יותר של פירוק מספק תוצאות היתוך טובות יותר.

ניסיון לשלב תמונות רפואיות :

|  |  |  |
| --- | --- | --- |
| CT | MRT |  |
|  | | Original Images |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Level 7 Fusion  Approx: min, Details: max | Level 7 Fusion  Approx: mean, Details: max | Level 1 Fusion  Approx: min, Details: max | Level 1 Fusion  Approx: mean, Details: max |  |
|  |  |  |  | MDCT |
|  |  |  |  | MDWT |

## הסבר על שימוש בGUI

GUI רשום בשפת פיתון

**את GUI ניתן להפעיל בעזרת שני דרכים:**

בעזרת שימוש בקובץ DCT - based Image Fusion.py יש לפתוח את קובץ זה בסביבת עבודה של PyCharm או בVisual Studio Code

יש לתקין ספריות הבאות (לעריץ פקודות):

pip install tk

pip install numpy

pip install matplotlib

pip install scipy

pip install opencv-python

pip install scikit-image

pip install PyWavelets

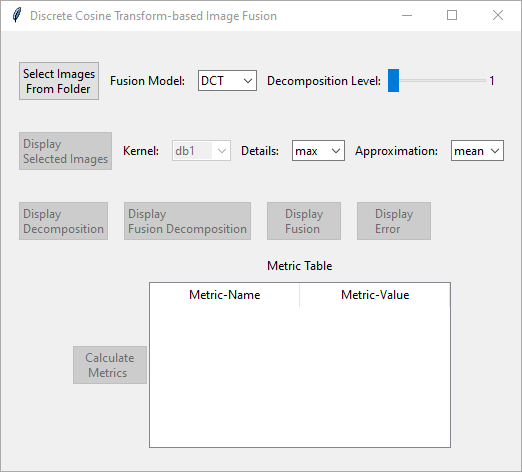
pip install tabulate

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

\*ראשית יש להוריד את תיקיה למחשב ולבצע Unzip

\*\*לא נדרש להתקין ספריות

**שימוש:**

* Select Images from Folder – בחירה מרובע של תמונות בעזרת החזקת מקש CTRL ובחירה בעזרת עכבר או סימון בעזרת עכבר בהחזקת מקש שמאלי.

ניתן לבחור מ2 עד 3 תמונות כאשר תמונה שלישית היא תמונת יחוס.

בעט בחירה של שני תמונות בלבד תתאפשר לחיצה על כל הכפתורים פרט לכפתור Display Error,מכוון שנדרשת תמונת יחוס.

* Display Selected Images – כפתור שיציג את תמונות שנבחרו, \*\*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 ימין של עכבר כאשר סמן נמצא באזור החלון פתוח.
* Display Decomposition - הצגת פירוק של תמונות שנבחרו להיתוך (תמונה ראשונה ושניה)
* Fusion Model – בחירת אלגוריתם היתוך, כאשר נבחר DWT תתאשר אופציה לשינוי Kernel (Mother Wavelet) [haar, db1] \*\* לא נירא שיש הבדל!!
* Decomposition Level – רמת הפירוק של התמונות לצורך היתוך [1-8], הפעלה רקורסיבית של פירוק על תמונת LL.
* Details ו – Approximation – בחירת אופן מיזוג של התמונות [min, max, mean]
* Display Fusion Decomposition – הצגת פירוק לאחר היתוך, היתוך מתבצע לפי בחירה של Details ו – Approximation.
* Display Fusion – הצגת תמונה ממוזגת לאחר היתוך
* Display Error – אופציה הזאת קיימת רק בבחירה של שלוש תמונות כאשר תמונה שלישית היא תמונת יחוס, מציגה שגיאה בין תמונה ממוזגת ל תמונת יחוס.
* Calculate Metrics – הצגת מטריקות של יעילות של אלגוריתם, כאשר נבחרו רק שני תמונות (תמונות למיזוג) תוצג רק 3 מטריקות, וכאשר תבחר 3 תמונות תוצג 6 מטריקות.
* **\*\*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 ימין של עכבר כאשר סמן נמצא באזור החלון פתוח.**
* **הקוד נמצא ב Appendix B - Script ו**[**Git**](https://github.com/deamon312/image-processing/tree/main/Project%201/First%20Article)

# Appendix A – Script

# 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])]])

########################################## DCT #########################################################

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')

        # 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\_RBUTTONUP:

            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

    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\_RBUTTONUP:

            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(f'Decomposition level: {level}', combined\_i)

        cv2.setWindowProperty(f'Decomposition level: {level}', cv2.WND\_PROP\_TOPMOST, 1)

        cv2.setMouseCallback(f'Decomposition level: {level}', mouse\_callback)

    if display\_fusion\_mosaic:

        cv2.putText(MDWT\_FUSION, f'Mother Wavelet: {wavelet} Approx: {approx} Details: {details}', (70, 500), cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (255, 255, 255), 1)

        cv2.imshow(f'Decomposition of Fusion ,level: {level}', MDWT\_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(IMDWT\_FUSION, f'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

    # 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 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\_RBUTTONUP:

            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\_RBUTTONUP:

        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('{}x{}+{}+{}'.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\n Metrics", 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()