בס"ד

במסמך זה אסביר את כל הקודים בפרויקט לפי הסדר. המסמך נועד לתת פרטים והסברים טכניים שלא מופיעים בספר הפרויקט המצורף.  
חשוב מאוד לקרוא את ספר הפרויקט בעיון לפני קריאת מסמך זה, מכיוון שמסמך זה מסתמך על הבנה מלאה של הרעיון הכללי של הפרויקט.

**ניקוי רעשים – מערכת מבוססת רשתות סקלריות לא מנורמלות:**

**Denoising\DATA\_SET\_GRAY\_SCALAR\Std25\createDataSet.m:**

בקוד זה ניצור את ה-Dataset עבור 81 הרשתות הסקלריות לניקוי רעשים בסטיית תקן של 25.  
עבור כל פס תדר:

נקרא תמונה אחרי תמונה מתוך תמונות ה-Dataset שהורדנו מהאינטרנט ואם התמונה צבעונית נדאג להמירה לגווני אפור (זו פעולה שנבצע למרות שכל התמונות בתיקייה בגווני אפור, על מנת שהתוכנה לא תקרוס במידה ותוכנס לתיקייה תמונה צבעונית).

נרעיש את התמונה ע"י הוספת רעש גאוסי עם תוחלת 0 וסטיית תקן של sigma אותה הגדרנו ל-25. נשים לב שלאחר תוספת הרעש התקבלו ערכים גדולים מ-255 ולכן נמיר אותם ל-255 ובנוסף התקבלו ערכים קטנים מ-0 ולכן אותם נמיר ל-0. בנוסף הערכים המתקבלים לא שלמים ומכיוון שאנו רוצים לדמות תמונה אמיתית עם רעש, נעגל את כל הערכים בעזרת uint8.

יצירת ה-data:  
נבצע קונבולוציה של התמונה הרועשת עם המסנן הנוכחי. את התמונה במוצא נשטח לווקטור עמודה – זה יהיה ה-data המתקבל מהתמונה הנוכחית. את הוקטור המתקבל נשרשר ל-data של המסנן הנוכחי.

יצירת ה-label:   
נבצע קונבולוציה של התמונה הנקייה מרעשים עם המסנן הנוכחי. את התמונה במוצא נשטח לווקטור עמודה – זה יהיה ה-label המתקבל מהתמונה הנוכחית. את הוקטור המתקבל נשרשר ל-label של המסנן הנוכחי.

כשנסיים לעבור על כל התמונות באותו מסנן, נקבל cell בגודל 1X2 שבעמודה הראשונה שלו מופיע וקטור עמודה גדול של data ובעמודה השנייה שלו מופיע וקטור עמודה גדול של label. כל cell כזה נשמור בשם מתאים בהתאם לכל מסנן.

**Denoising\Scalar-std25\adam,4layer,256,sigmoid\buildTheNetwork\ buildTheNetwork\ buildTheNetwork.py:**

זה הקוד הראשון שאנו כותבים בפייתון ולכן חשוב להדגיש כמה דברים:  
ראשית לאחר בחינת כמה סביבות עבודה, אני ממליץ להשתמש בסביבת visual studio.  
במהלך הפרויקט נדרש להוריד לא מעט מודולים (ספריות) בגרסאות שונות וללא ספק הכי קל לבצע זאת ב-visual studio.  
אמנם זו לא הסביבה הנפוצה בפייתון, אך זו ללא ספק הסביבה שענתה לי על הדרישות בצורה הטובה ביותר.

אנו שומרים את המודל בפורמט h5 (זו הדרך הנפוצה לשמירת מודלים של רשתות). המטרה היא שנוכל לקרוא אותו ב-Matlab ולהשתמש בו.  
כאן חובה להשתמש בגרסת tensorflow של 2.1.0 ובכך לקבל גרסת keras של 2.2.4.  
במידה ומשתמשים ב-visual studio ניתן להוריד את הספרייה בקלות ע"י הרצת השורה:  
pip install tensorflow==2.1.0 –user  
אם לא משמשים בגרסה זו, לא ניתן לקרוא ב-Matlab את המודל.

תחילה טוענים את ה-modules הדרושים לקוד.  
מכיוון שנדרש לאמן 81 רשתות אנו עוברים בלולאה על 81 ה-Datasets שיצרנו עבור 81 פסי התדר השונים.

טוענים את ה-data וה-label הרלוונטיים לאותו מסנן. נשים לב שאנו משתמשים ב-numpy.array ולא ב-list. הסיבה היא המיקום בזיכרון. בהמשך נשתמש ברשימה של רשימות אשר תופסת הרבה יותר מקום בזיכרון מאשר מערך דו ממדי של numpy (זאת מכיוון שכל list שאנו יוצרים תופסת מקום נוסף בזיכרון שנועד למימוש תכונות ה-list), לכן נקפיד להתחיל לעבוד כבר מעכשיו עם מערכי numpy.

* טוענים את קובץ ה-mat שיצרנו ב-Matlab לתוך משתנה בשם mat. הטעינה מתבצעת באמצעות שימוש ב-scipy.io.loadmat.
* אנחנו לא יודעים מה אורך וקטורי העמודה בכל תא, ועל מנת לגלות אנו בודקים מה האורך למשל של mat[0][0] – כלומר mat בשורה ה-0 ובעמודה ה-0.
* עבור ה-train נשתמש ב-80% מהדוגמאות ועבור ה-validation ב-20% הנותרים.
* את ה-data לוקחים מהעמודה הראשונה (אינדקס 0) ואת ה-label לוקחים מהעמודה השנייה (אינדקס 1).

ארכיטקטורת הרשת כתובה בקצרה בנתיב לקוד הנוכחי.

* בחרנו ברשת מכילה 4 שכבות Dense בעלות 256 פרספקטרונים כל אחת, כאשר שכבת האקטיבציה בכל שכבה היא sigmoid.
* כאמור, לעת עתה אנו עובדים ברשת סקלרית, כלומר הרשת מקבלת במבואה סקלר לכן input\_shape=(1,), בנוסף רשת סקלרית מוציאה במוצאה גם כן סקלר ולכן הוספנו בסוף שכבה עם פרספקטרון אחד.
* נשתמש ב-optimizer של adam כאשר ה-loss מחושב לפי mse.
* נאמן את הרשת על 5 Epochs (בדיעבד, אימנו תחילה על יותר Epochs אך ראינו שאין שינוי ברשת אחרי בערך 5 Epochs. לכן זו כמות האפוקים שנבחרה).

אנו מעוניינים כרגע להציג את ה-loss של ה-Train ושל ה-Validation באחוזים, ולכן נחשב אותם מתוך האנרגיה של ה-Train וה-Validation בהתאמה.

* עבור האנרגיה של ה-Train:  
  נחשב את סכום הריבועים של האיברים ב-Label של ה-Train ונחלק בכמות האיברים.
* עבור האנרגיה של ה-Validation:  
  נחשב את סכום הריבועים של האיברים ב-Label של ה- Validationונחלק בכמות האיברים.

את הגרף נשמור לקובץ בשם plotResults<numFilter>.pdf יחד עם מספר המסנן וכן את המודל עם המקדמים נשמור לקובץ model<numFilter>.h5 יחד עם מספר המסנן.

* נשתמש ב-plt.figure על מנת לפתוח גרף.
* נשתמש ב-plt.title על מנת להוסיף כותרת לגרף.
* בעזרת plt.plot נשרטט את שני הגרפים עבור ה-loss של ה-Train ושל ה-Validation.
* נוסיף מקרא לגרפים בעזרת plt.legend.
* נוסיף רשת לגרף באמצעות plt.grid.
* ולבסוף נשמור את הגרף לקובץ pdf באמצעות יצירת קובץ pdf ע"י PdfPages ואז שמירת האיור לקובץ ע"י savefig.
* נשמור כמובן את ה-model לקובץ h5 בעזרת save.

**Denoising\Scalar-std25\adam,4layer,256,sigmoid\RepairImage.m**:

את התוצאות אנו בוחנים על 6 תמונות, אלו הן 6 התמונות שמופיעות במאמר שאליו נשווה את התוצאות שלנו ומתואר בספר.

נכתוב לולאה שטוענת לקוד את כל הרשתות שאימנו בקוד הקודם ב-Python. את הרשתות נטען לתוך cell בשם netCell.

על מנת לבחון את התוצאות נצטרך להרעיש את התמונה, להכניס אותה לרשת ולחשב את ה-PSNR וה-SSIM של התמונה המתוקנת ביחס לתמונה המקורית.

ראשית, נרעיש את התמונה בדיוק באותו אופן שבוצע בקוד ליצירת ה-Dataset.  
לאחר מכן נעבור על כל המסננים ועבור על מסנן נבצע את הפעולות הבאות:

1. נבצע קונבולוציה של התמונה בכניסה עם המסנן הנוכחי.
2. נשטח את התוצאה לקבלת וקטור.
3. את התוצאה נכניס לרשת אך במקום להשתמש בפעולת activation החלטנו לממש בעצמנו את פעולת הרשת ע"י קריאה ערכי ה- Weightsוה- Bias של כל שכבה. כאן נכנס היתרון המרכזי של המחקר (שהוסבר בתקציר) שנגזר מהעובדה שהרשת שמכילה רק שכבות Dense – קלות המימוש.
4. את הוקטור המתקבל במוצא הרשת נמיר חזרה לממדי התמונה.
5. נוסיף את התוצאה לתוך ImageRepair – התמונה המשוחזרת (נקיה מרעשים עד כמה שהצלחנו).

בחינת התוצאות תבוצע בהתאם השלבים הבאים:

1. הורדת קצוות שנוספו ל- ImageRepairבעקבות פעולת הקונבולוציה אשר מרחיבה את ממדי התמונה.
2. חישוב PSNR ו-SSIM על מנת להשוות בין התמונה שהתקבלה לתמונה המקורית.
3. נציג ב-figure אחד את שלושת התמונות: התמונה המקורית, התמונה הרועשת והתמונה ביציאה מהרשת. בנוסף נשמור את כל ה-figures.

את ערכי PSNR ו-SSIM נשמור בקבצי mat בשמות PSNR ו-SSIM בהתאמה.

כעת נישאר עם אותו ה-Dataset שיצרנו, נאמן ונבחן רשתות שונות. הפעולות המבוצעות הן בדיוק הפעולות שתוארו לעיל רק עם רשתות שונות. הקודים מתוארים בתיקיות הבאות:

**Denoising\Scalar-std25\adam,2layer,256,sigmoid**

**Denoising\Scalar-std25\SGD,4layer,256,sigmoid**

**ניקוי רעשים – מערכת מבוססת רשתות סקלריות מנורמלות:**

נעבור ליצירת Dataset מנורמל בתחום [-1,1]:

**Denoising\DATA\_SET\_GRAY\_SCALAR\_NORMALIZED\Std25\ createDataSet.m:**

לכל פס תדר אנו מוסיפים מקדם נרמול. זהו בעצם מקדם שנחלק בו לפני הכניסה לרשת ונכפיל בו ביציאה ממנה. כך בעצם בכניסה לרשת תהיה בטווח ערכים  
[-1,1] כפי שאנו רוצים.   
עבור על פס תדר (עבור כל רשת) קיים מקדם נרמול שונה.

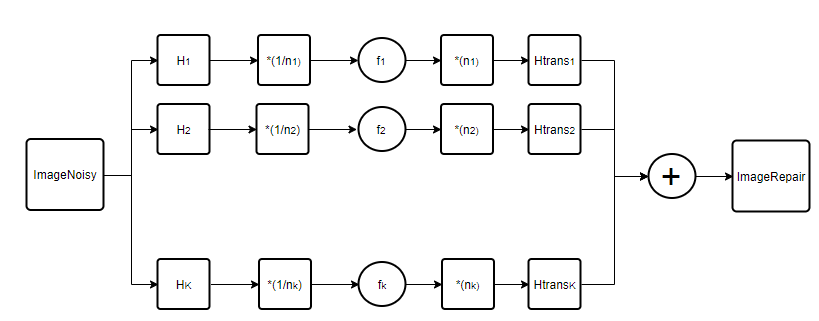
ב-createDataSet.m נמצא את הערך המקסימלי בערך מוחלט מתוך ה-data וה-label של כל פס תדר. נשמור את כל הערכים האלה בקובץ בשם normalized.mat.  
את כל ה-data וה-label של כל מסנן נחלק במקדם זה, כך הנתונים של הכניסה והיציאה מהרשת איתם הרשת מתאמנת יהיו בטווח ערכים [-1,1].

**Denoising\Scalar-std25\Normelized-adam,4layer,256,sigmoid\buildTheNetwork\ buildTheNetwork\buildTheNetwork.py:**

נותר ללא שינוי מכיוון שלא דרוש שינוי ברשת ובאימונה.

**Denoising\Scalar-std25\Normelized-adam,4layer,256,sigmoid\ RepairImageWithH1.m:**

את RepairImage.m נפצל לשתי אופציות המתוארות להלן:  
אופציה א - :RepairImageWithH1.m   
פשוט נחלק במקדם הנרמול של כל פס תדר לפני הכניסה לרשת ונכפול בו ביציאה מהרשת.  
תרשים בלוקים של המערכת עם הנרמול (עם המסנן הראשון):

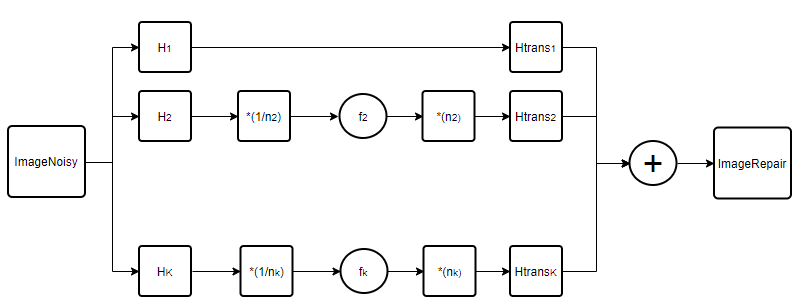


לכן השינויים בקוד הם:

1. לאחר שיטוח "התמונה" בתדר לאחר הקונבולוציה, נחלק את הוקטור המתקבל במקדם הנרמול המתאים לאותו מסנן.
2. בלאחר מכן, ביציאה מהרשת נכפיל שוב באותו מקדם נרמול.

**Denoising\Scalar-std25\Normelized-adam,4layer,256,sigmoid\ RepairImageWithoutH1.m:**

המסנן הראשון הוא פס תדר ה-DC, זהו מסנן שמבצע ממוצע. הממוצע של הרעש הוא בעצם התוחלת, כלומר 0. לכן תיאורטית פס התדר הראשון כלל לא צריך תיקון של הרשת.  
אנו רוצים לבדוק טענה זו ולראות האם לדבר זה יש השפעה ממשית על התוצאות.  
תרשים בלוקים של המערכת עם הנרמול (ללא תיקון למסנן הראשון):



ההבדל מהקוד הקודם הוא שכעת את פס התדר הראשון מבצעים מחוץ ללולאה.  
מעבירים את התמונה בקונבולוציה עם המסנן הראשון, לאחר מכן מבצעים קונבולוציה עם המסנן המשוחלף ומוסיפים את התוצאה לתמונה המתוקנת.

**ניקוי רעשים – מערכת מבוססת רשת וקטורית לא מנורמלת:**

**Denoising\DATA\_SET\_GRAY\_VECTOR\Std25\createDataSet.m:**

למשתנה שישמור את ה-data וה-label שאנו יוצרים למערכת אנו קוראים newDataAndLabel.   
נתבונן על "הטבלה" שיצרנו ברשת הסקלרית כאשר יצרנו את 81 קבצי ה-mat:

|  |  |  |
| --- | --- | --- |
| DataAndLabel | | |
| label | data |  |
| mX1 | mX1 | פס תדר 1 |
| mX1 | mX1 | פס תדר 2 |
| ... | ... | ... |
| mX1 | mX1 | פס תדר 81 |

הערה: כל שורה בטבלה מייצגת קובץ mat שיצרנו, כאשר m זהו מספר הדוגמאות שקיבלנו מכל התמונות ב-Dataset.

ה-newDataAndLabel שאנו רוצים כעת שונה במבנהו מה-DataAndLabel שיצרנו ברשת הסקלרית. הסיבה היא שכעת יש רשת אחת והיא מקבלת וקטור בגודל 81 שבו כל איבר מגיע ממסנן אחר.  
לכן, ברשת הוקטורית מבנה זה לא מתאים מכיוון שבמבנה זה אין וקטורים של 81X1 כפי שאנו צריכים. לכן נבצע המרה של המבנה לעיל למבנה הבא:

|  |  |
| --- | --- |
| newDataAndLabel | |
| label | data |
| 81X1 | 81X1 |
| 81X1 | 81X1 |
| ... | ... |
| 81X1 | 81X1 |

הערה: כמות השורות במבנה זה היא כמות הדוגמאות.

נסביר ע"י שימוש במספרים אמיתיים מה-Dataset שלנו:  
ברשת הסקלרית יצרנו טבלה עם 81 שורות שבכל שורה יש 2 עמודות שבכל אחת מהן ישנו וקטור עמודה בגודל 12,438,400. כעת אנו מבצעים המרה שתשמור על כל הנתונים אך בסדר אחר:  
אנו רוצים טבלה עם 12,438,400 שורות שבכל שורה 2 עמודות שבכל שורה מהן ישנו וקטור עמודה בגודל 81.  
כלומר כל שורה במבנה החדש תהווה דוגמה לרשת הוקטורית שאנו מאמנים.

עד כאן זה היה בתאוריה, אך במציאות הטבלאות לעיל גדולות מאוד ודורשות המון זיכרון, לכן גם המעבר עליהן דורש המון זיכרון (ולעיתים אף גורם לקריסת התוכנית).  
נפתור בעיה זו ע"י אלגוריתם שלוקח בכל פעם 5 תמונות מה-Dataset ומבצע עליו את הפעולות שהוסברו.  
בסופו של דבר נקבל כמה קבצים קטנים יותר (בערך 1.5GB כל אחד) במקום קובץ אחד גדול שגם אם היינו מחכים הרבה זמן ויוצרים אותו הייתה נוצרת בעיה בקריאתו ב-Python עקב גודלו.

ועכשיו לקוד:

* מתחילים בלולאת while אין סופית שנעצרת כאשר נגמרות התמונות בתיקיית התמונות (נבדק באמצעות הפונקציה hasdata).
* בכל איטרציה של הלולאה החיצונית עוברים על חמש תמונות מתיקיית התמונות.
* קוראים תמונה מתוך התיקייה ושוב אם היא צבעונית משנים אותה לגווני אפור.
* מרעישים את התמונה באותו אופן שהוסבר לעיל.
* בונים את הטבלה לעיל שיצרנו ב-Dataset לרשת הסקלרית (הטבלה עם ה-81 שורות).
* כאשר סיימנו לקבל את הטבלה עבור 5 תמונות נבצע את ההמרה מטבלת DataAndLabel ל-newDataAndLabel.  
  כל ההעברה מתבצעת בלולאה דו ממדית שבה לוקחים איבר מכל מסנן ויוצרים דוגמאות של וקטורי עמודה בגודל 81.
* לבסוף נשמור את קבצי newDataAndLabel שנוצרו.

**C:\Users\user\Desktop\Studies\Semester 8\myProject\Denoising\vector-std25\adam,4layers,256-256-256-256,sigmoid\buildTheNetwork\ buildTheNetwork\buildTheNetwork.py**

נטען את הנתונים בקבצי ה-mat שיצרנו לתוך ארבעה מערכי numpy מסוג float32. באותו אופן כמו ברשת הסקלרית 80% מהנתונים משמשים עבור ה-Train ו-20% נוספים משמשים עבור ה-Validation. רק שכאן הטעינה מתבצעת בצורה שונה.

* אנו לא יודעים באיזה גודל עתיד להיות המערך מכיוון שהוא מורכב מכמה קבצי mat ולא מקובץ אחד. לכן נגדיר מערך ריק (numpy.empty) של איברים בגודל 81 (וקטורי הדוגמאות).

נעבור על כל קבצי ה-mat ועבור כל קובץ :

* נכניס לתוך מערך ה-currentTrainData את 80% השורות הראשונות של קובץ ה-mat הנוכחי.
* לאחר מכן את המערך הנוכחי currentTrainData נשרשר למערך כללי trainData באמצעות הפונקציה vstack.
* כך נעשה גם עבור: validationData, trainLabel, validationLabel.

בבניית ארכיטקטורת הרשת נשים לב שכעת משום שהרשת וקטורית אנו משנים את ה-input\_shape ל-81 וכן את שכבת ה-dense האחרונה נשנה גם כן ל-81 פרספקטרונים.

השינוי האחרון הוא שכאן כבר לא נתעסק עם השגיאה באחוזים, ולכן אין צורך בחישוב האנרגיה.

**Denoising\vector-std25\adam,4layers,256-256-256-256,sigmoid\imageRepair.m:**

נקודות חשובות על השינויים במעבר לרשת הוקטורית ב-RepairImage.m:כאמור הכניסה לרשת היא וקטור שכל איבר בו מגיע מקונבולוציה עם מסנן אחר. על מנת לממש זאת, יצרנו inputMatrix – מטריצה שבה 81 שורות (שורה עבור כל מסנן) ובכל שורה מופיעה "התמונה בתדר" לאחר קונבולוציה עם המסנן המתאים ושיטוח לוקטור שורה.

* את inputMatrix אנו יוצרים כמובן לפני הכניסה לרשת.
* ביציאה מהרשת על מנת לקבל את כל את התמונות בפס תדר מסויים לוקחים את השורה הרלוונטית ממטריצת inputMatrix.

כאמור את ה-PSNR וה-SSIM אנחנו מחשבים בעזרת ממוצע של 10 ריצות, ולכן נוספה לולאה חיצונית עם המשתנה iteration שרצה מ-1 עד Num\_iteration\_for\_average.  
ה-PSNR וה-SSIM נשמרים עם תוצאות כל הריצות ובעמודה האחרונה (ה-11) מופיעות התוצאות הממוצעות של כל הריצות יחד.

* מסיבה זו, את כמות העמודות ב-PSNR וב-SSIM נגדיר ל- Num\_iteration\_for\_average+1.
* לשם כך, נוספה לולאה בסוף הקוד שרק מחשבת ממוצעים של כל שורה ומכניסה את התוצאות לעמודה ה-11 בטבלה בשורה המתאימה לכל תמונה.

כעת נישאר עם אותו ה-Dataset שיצרנו, נאמן ונבחן רשתות שונות. הפעולות המבוצעות הן בדיוק הפעולות שתוארו לעיל רק עם רשתות שונות. הקודים מתוארים בתיקיות הבאות:

**Denoising\vector-std25\adam,2layers,1024-512,sigmoid**

**Denoising\vector-std25\adam,6layers,256-256-256-256-256-256,sigmoid**

**ניקוי רעשים – מערכת מבוססת רשת וקטורית מנורמלת:**

**Denoising\DATA\_SET\_GRAY\_VECTOR\_NORMALIZED\Std25\ createDataSet.m:**

המעבר מרשת וקטורית לרשת וקטורית מנורמלת קצת יותר בעייתי מהמעבר מרשת סקלרית לרשת סקלרית מנורמלת, נסביר:  
ברשת הסקלרית כל קובץ mat היה מיועד עבור מסנן מסויים. פשוט מצאנו את המקסימום בקובץ זה וחילקנו בו. כעת, ברשת הוקטורית, לא ניתן למצוא את מקדם הנרמול לפני יצירת כל קבצי ה-Dataset החדש, מכיוון שבכל פעם נוספות תמונות חדשות.

הפתרון לכך מעט מסובך. נשמור בוקטור normalized (שהוא וקטור בגודל 81) את מקדמי הנרמול המקסימליים שאנו מוצאים. נייצר את ה-newDataAndLabel כרגיל ללא חילוק במקדמי הנרמול.   
הסבר לביצוע: במקום לשמור את מקדם הנרמול המתקבל מריצה על כל 5 תמונות נוספות בכל פעם, נבדוק אם הוא גדול ממקדם הנרמול הקודם עבור אותו מסנן ואם כן נשמור אותו.  
בדרך זו באמת נשמור רק את מקדמי הנרמול המקסימליים עבור כל מסנן שיבטיחו להכניס את **כל הכניסות** לרשת לטווח [-1,1] בצורה נכונה.

לאחר מכן נעבור על כל קבצי ה-mat שיצרנו, נזכיר שקבצים אלו מכילים טבלה עם המון שורות שבכל שורה 2 עמודות שבכל עמודה וקטור עמודה בגודל 81. נחלק כל וקטור כזה בוקטור מקדמי הנרמול. איבר חלקי איבר. כך נבצע חילוק במקדם נרמול תלוי מסנן.  
חילוק וקטורים איבר-איבר ב-Matlab מתבצע ע"י הוספת אופרטור הנקודה (.) לפני אופרטור החילוק (/).

השינויים הדרושים בקוד:

* נוסיף שורה לשמירת מקדמי הנרמול המקסימלים עבור כל מסנן.
* לאחר סיום לולאת ה-while החיצונית ומעבר על כל התמונות, נעבור על כל קבצי ה-mat ששמרנו ונחלק את וקטורי הדוגמאות בוקטור מקדמי הנרמול כפי שמוסבר לעיל.

**Denoising\vector-std25\Normalized,adam,4layers,256-256-256-256,sigmoid\buildTheNetwork\buildTheNetwork\buildTheNetwork.py**

אימון הרשת נשאר ללא שינוי מהקוד הקודם לאימון הרשת (עבור הרשת הוקטורית הלא מנורמלת).

**Denoising\vector-std25\Normalized,adam,4layers,256-256-256-256,sigmoid\imageRepair.m:**

נזכור שברשת הוקטורית יש לנו את המטריצה input\_matrix שבנינו שבה 81 שורות, שורה עבור כל מסנן. אנו צריכים לפני הכניסה לרשת לחלק כל שורה במטריצה במקדם הנרמול המתאים לאותו מסנן, ומיד ביציאה מהרשת להכפיל במקדמים אלה.  
ב-Matlab ניתן לבצע את החלוקה הבאה בפשטות בצורה הבאה למשל:  
[1,2;3,4]./[2;6]=[0.5,1;0.5,0.6667]

* בפועל ממש לפני הכנסת inputMatrix לרשת נחלק אותה בוקטור normalized.
* וממש אחרי יציאת inputMatrix מהרשת נכפיל אותה בוקטור normalized.

כעת נישאר עם אותו ה-Dataset שיצרנו, נאמן ונבחן רשת נוספת. הפעולות המבוצעות הן בדיוק הפעולות שתוארו לעיל רק עם רשת שונה. הקוד מתואר בתיקיה הבאה:

**Denoising\vector-std25\Normalized,adam,2layers,1024-512,sigmoid**

כפי שמוסבר בספר כעת נעבור לבחינת התוצאות של הרשת הטובה ביותר שקיבלנו עבור סטיות תקן שונות.

התיקיות הבאות מכילות את הקודים שהוסברו לעיל רק עם סטיות תקן שונות בהתאם לשם התיקייה:

**Denoising\vector-std1**

**Denoising\vector-std2**

**Denoising\vector-std5**

**Denoising\vector-std10**

**Denoising\vector-std15**

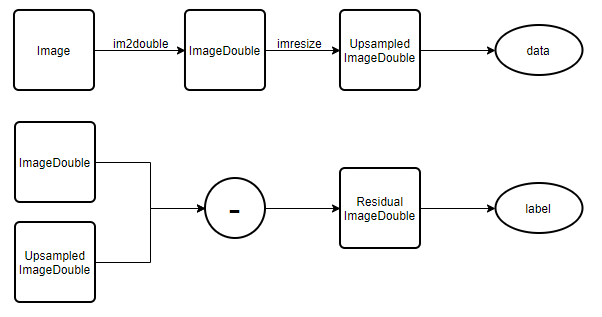
**Denoising\vector-std20**

**סופר רזולוציה - תמונות בגווני אפור בגודל 100X100 עם יחס הגדלה של 2:**

**SuperResolution\result\_100X100\_scale2\createDataSet.m:**

הרעיון כללי:   
קבלת תיקייה עם תמונות והמרתה ל-4 תיקיות נפרדות השמות:  
trainUpsampledImages, trainResidualImages, validationUpsampledImages, validationResidualImages

1. מכיוון שכרגע אנו עובדים עם תמונות בגווני אפור, נשנה כל תמונה שאנו קוראים מה-Dataset שלנו לתמונות בגווני אפור.
2. תמונת ה-residual כוללת בתוכה גם ערכים שליליים ובנוסף לכך אנו גם מנרמלים את ערכי התמונה ולכן לא ניתן לשמור אותה כתמונה (עם סיומת סטנדרטית של תמונה).   
   לכן נשמור את המידע בקבצי.



* תחילה שומרים את שמות ארבעת התיקיות במשתנים בתוכנה
* נוסיף 4 בדיקות (בדיקה עבור כל תיקיה:   
  אם תיקייה מסוימת לא קיימת ניצור אותה
* יוצרים imageDatastore לצורך גישה לכל התמונות (כפי שבוצע כבר בקודים קודמים).
* עוברים על כל ה-imageDatastore:  
  - קוראים תמונה  
  - ממירים אותה לגווני אפור  
  - חותכים ממנה את החלק השמאלי העליון בגודל 100X100 בעזרת imcrop  
  - מכניסים אותה לתחום [0,1] בעזרת im2double ונקבל את ImageDouble  
  - מקטינים אותה ומגדילים אותה חזרה לצורך קבלת תמונת רזולוציה נמוכה שהוגדלה – UpsampledImageDouble  
  - מחסרים בין התמונה ברזולוציה הגבוהה לתמונה המוגדלת לצורך קבלת ResidualImageDouble  
  - בסוף הלולאה מופיע תנאי שמפצל 80% ל-Train ו-20% ל-Validation

**SuperResolution\result\_100X100\_scale2\buildTheNetwork-adam,4layers,256-256-256-256,relu,batch32\buildTheNetwork\buildTheNetwork.py:**

חלק א – טעינת קבצי ה-mat של ה-Dataset מארבעת התיקיות שיצרנו ב-Matlab:

נסביר על טעינת קבצי ה-mat מתיקייה מסוימת:  
הרעיון הכללי הוא להכניס את המידע מכל תיקייה לרשימה של תמונות (כלומר, רשימה של רשימות דו ממדיות).  
אמנם יש לשים לב לדבר חשוב מאוד:  
רשימות רב ממדיות (list) תופסות הרבה יותר מקום בזיכרון לעומת מערכי numpy. זה דבר מאוד משמעותי מכיוון שזיכרון ה-RAM מוגבל כמובן. אנו חייבים לצמצם כמה שיותר בצריכת זיכרון זה. לשם המחשה טעינת תיקייה של תמונות ששוקלת בערך 2GB, צורכת בערך 14GB כאשר שומרים אותה ברשימה תלת ממדית כפי שתיארנו. לעומת זאת אם נשמור אותה ב-numpy array בעל שלושה ממדים היא תצרוך בדיוק 2GB.

נגדיר מערך תלת ממדי כאשר הגודל החיצוני הוא מספר קבצי ה-mat (כמות התמונות), הממד השני הוא מספר השורות בכל תמונה והממד השלישי הוא מספר העמודות בכל תמונה.  
נעבור על כל הקבצים בתיקייה ונמלא את המערך התלת ממדי. בסיום עבודה על כל קובץ נדפיס את שמו למסך לשם נוחות ודיבוג הקוד בזמן הריצה.

הערה:  
נשים לב שלאחר מילוי המערך התלת ממדי נוסיף ממד של 1 בסוף לשם התאמה לשכבת הכניסה של הרשת. זאת ניתן להבין מאחר וללא הוספת הממד מתקבלת הודעת השגיאה הבאה:

ValueError: Input 0 of layer conv2d is incompatible with the layer:  
expected ndim=4, found ndim=3. Full shape received: [None, 100, 100]

נשים לב שזה לא קורה בגלל ה-1 שכתוב בשכבה הראשונה של הרשת - גם אם מורידים אותו נשארים עם אותה שגיאה בדיוק. בנוסף גם אם מורידים אותו וגם את הממד הנוסף שהוספנו לכל מערך תלת ממדי נשארים עם אותה השגיאה. לאחר קריאה אודות שגיאה זו הבנתי שהוספת ממד של 1 במקום המתאים זהו הפתרון הנדרש.  
הוספת הממד מתבצעת ע"י הפונקציה expand\_dims.

חלק ב – טעינת המשקלים (Weight) שיצרנו בקוד שמייצר את המסננים:

הסבר על יצירת המסננים:  
בכניסה לרשת ישנה שכבת קונבולוציה שמטרתה לפרק את התמונה ל-81 תמונות בפסי תדר שונים. ההמרה מתבצעת ע"י קונבולוציה של התמונה בכניסה עם 81 מסננים שונים עבור 81 פסי התדר. גודל כל מסנן הוא 9X9, והמסננים הם בעצם פונקציות בסיס של התמרת DCT.   
זוהי ההתמרה שב-"ניקוי רעשים" ביצענו מחוץ לרשת – עכשיו היא מתבצעת בתוך הרשת.

הקוד המחשב את 81 המסננים ושומר אותם לקובץ mat הינו:

**create\_DCT\_H\_and\_H\_trans\_From\_Amir\_code**

בסוף הקוד הוספתי לולאה אחת שאחראית על התאמת הממדים של המשקלים בצורה נכונה:

פעולה מקדימה למציאת הממדים הנדרשים:  
ראשית ב-Python בניתי את השכבה הראשונה והאחרונה ללא אתחול וקיבוע של המשקלים, כך ע"י שימוש בפקודה model.layer[i].get\_weight()[0] קיבלתי את המבנה בו שמורים ה-Weight, וע"י הפקודה model.layer[i].get\_weight()[1] קיבלתי את המבנה בו שמורים ה-Biases.

מסקנות:  
בשכבה הראשונה:   
המשקלים: shape(9,9,1,81) – כלומר הממד השלישי קבוע 1 והממד הרביעי הוא מספר המסנן.  
הביאסים: shape(81,)  
בשכבה האחרונה:   
המשקלים: shape(9,9,81,1) – כלומר בממד השלישי הוא מספר המסנן וממד הרביעי קבוע 1.  
הביאסים: shape(1,)

מכאן ידעתי איך לבצע את שינוי הממדים ב-Matlab לממדים הנכונים וזה מה שביצעתי בלולאה בקוד create\_DCT\_H\_and\_H\_trans\_From\_Amir\_code.

הערה:  
נשים לב שב-Matlab אין משמעות לממד של 1 בממד האחרון, אלא רק במעבר ל-Python.   
לכן במשקלים של השכבה האחרונה הוספנו ממד של 1 רק ב-Python בעזרת שורת הקוד:

WeightsForLast = numpy.expand\_dims(WeightsForLast, axis=3)

הבדיקה שהאתחול והקיבוע של הפרמטרים הצליח היא ע"י הכנסת תמונה לרשת, וחיסור בין התמונה במוצא לתמונה בכניסה - באמת הכל מתאפס חוץ מ-4 פיקסלים בכל דופן (בגלל הקונבולוציה עם מסנן בגודל 9X9).

חלק ג – יצירת ארכיטקטורת הרשת ואימונה:

כאמור, בכניסה לרשת ישנה שכבת קונבולוציה שמטרתה לפרק את התמונה ל-81 תמונות בפסי תדר שונים. ההמרה מתבצעת ע"י קונבולוציה של התמונה בכניסה עם 81 מסננים שונים עבור 81 פסי התדר. גודל כל מסנן הוא 9X9, והמסננים הם בעצם פונקציות בסיס של התמרת DCT.

לבסוף מופיעה שכבת קונבולוציה אשר מטרתה לבצע את התמרת DCT ההפוכה, ע"י ביצוע קונבולוציה של כל תמונה בפס תדר כלשהו עם המסנן המשוחלף באותו פס תדר, ולבסוף לחבר את כל פסי התדר לקבלת תמונה אחת.

הערה חשובה:  
השכבה בכניסה – שאחראית על פירוק התמונה לפסי תדר שונים והשכבה ביציאה – שאחראית על הרכבת התמונה חזרה מתמונות פסי התדר לא מתאמנות כלל. נבצע זאת ע"י קביעת פרמטר trainable בשכבות אלה לערך false.

בין שתי שכבות אלה יופיעו "שכבות Dense" שכרגע גם הם ממומשות עם שכבות "Conv2D", ונסביר מדוע:

אנו רוצים בעצם להשתמש בדיוק באותו רעיון אשר ביצענו ברשת הוקטורית של ניקוי רעשים (De-Noise) – אנו רוצים לעבור על וקטורים בגודל 81 המורכבים מפיקסל אחד מכל פס תדר.

**על מנת לבצע זאת נשתמש בקונבולוציה דו ממדית עם גודל מסנן 1X1.  
הפעלת מסנן כזה שקולה לנוירון אחד שפועל על כל 81 השכבות וסורק את כל הפיקסלים.  
אם נקצה N מסננים כאלה הדבר שקול ל-N פרספקטרונים.  
בצורה זו אפשר ליצור מבנה רב שכבתי שזהה למבנה שבנינו ברשת הוקטורית של הפחתת רעש איתו אנו רוצים לעבוד.**

נשים לב שאחריי שכבות אלו חובה לשים שכבה עם 81 פרספקטרונים מכיוון שאם נכנסים וקטורים בגודל 81 למבנה הרב שכבתי אנו רוצים שגם יצאו ממבנה זה וקטורים בגודל 81.

בקוד המוצג להלן שמנו באמצע 4 שכבות בעלות 256 פרספקטרונים כל אחת (בהתאם לרשת הטובה ביותר שקיבלנו מניקוי הרעשים בחלק הראשון של הפרויקט) ופונקציית האקטיבציה של כל שכבה היא relu.

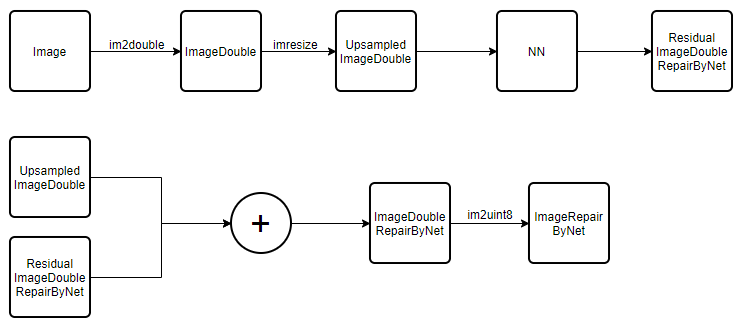
הערה:  
חשוב לציין שתחילה ניסיתי עם פונקציית אקטיבציה של sigmoid (כמו שביצענו בניקוי הרעשים) והתוצאות היו פחות טובות בצורה מאוד משמעותית – התוצאות שהתקבלו ע"י שימוש באינטרפולציית Bicubic היו טובות יותר מאשר התוצאות עם הרשת. מכן ניתן ללמוד על חשיבותה הרבה של פונקציית האקטיבציה.

חלק ד – שמירת המודל וגרף האימון:

שכבה זו נשארה ללא שינויים כלל.

**SuperResolution\result\_100X100\_scale2\RepairImage.m:**

נממש את תרשים המלבנים הבא:



* אנו רוצים לבחון את התוצאות על Datasets שונים ומוכרים אשר מכילים תמונות רבות והם: Set5, Set14, BSDS100, Urban100, Manga109.  
  אז נוסיף בתחילת הקוד קטע שבונה שתי טבלאות שישמשו עבור התוצאות של PSNR ושל SSIM.
* נעבור בלולאה על כל ה-Datasets.
* עבור כל Dataset נעבור על כל התמונות.
* נקרא תמונה מתוך ה-Dataset ונשמור אותה במשתנה Image.
* נמיר אותה ל-double ע"י im2double.
* נקטין אותה ואז נגדיל לקבלת UpsampledImageDouble.
* נכניס את התמונה לרשת לקבלת ResidualImageDouble.
* נחבר את ResidualImageDouble ואת UpsampledImageDouble לקבלת התמונה המתוקנת ע"י הרשת ImageDoubleRepairByNet.
* נחשב PSNR ו-SSIM של התמונה המתקבלת ביחס לתמונה ברזולוציה הגבוהה.
* לאחר שסיימנו לעבור על כל התמונות ב-Dataset נחשב ממוצע של התוצאות ונכניס את התוצאה למקום המתאים בטבלאות שיצרנו.
* לבסוף נשמור את טבלאות ה-PSNR וה-SSIM.

התיקיות הבאות בעלות אותם הקודים עם שינויים מינוריים (השינויים המהותיים הם ב-Dataset ולא בקודים והם מוסברים בספר פרויקט):

**סופר רזולוציה - תמונות בגווני אפור בגודל 512X512 עם יחס הגדלה של 2:**

**SuperResolution\result\_510X510\_scale2**

**סופר רזולוציה - תמונות בגווני אפור בגודל 512X512 עם יחס הגדלה של 3:**

**SuperResolution\result\_510X510\_scale3**

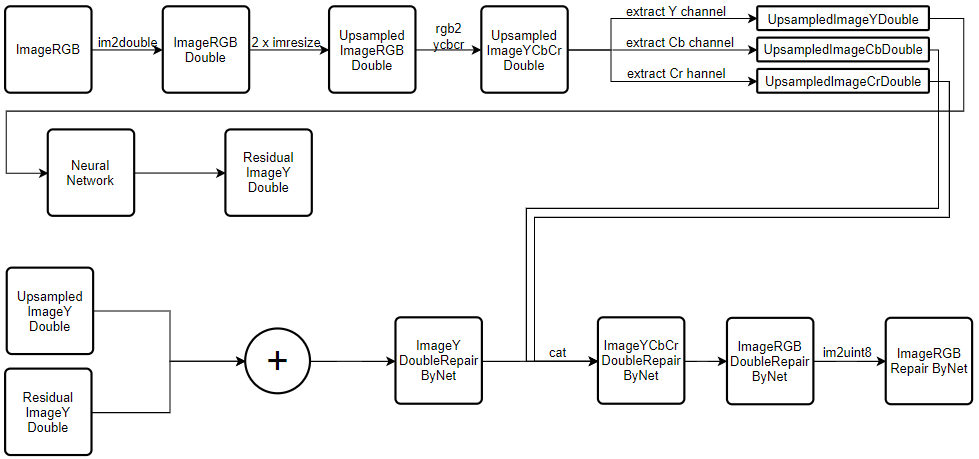
**סופר רזולוציה - תמונות צבעוניות בגודל 512X512 עם יחס הגדלה של 2:**

כאן לא יצרנו Dataset חדש מכיוון שעדיין נשתמש ברשת שמתקנת תמונות בגווני אפור ונסביר:

על מנת לבצע סופר רזולוציה על תמונה צבעונית עלינו לעבור ממרחב RGB למרחב YCbCr. על פי המוצג במאמר בספר, כאשר עוברים למרחב YCbCr מספיק לבצע את תיקון הרשת רק על ערוץ ה-Y ואילו את ערוצים Cb ו-Cr להעביר פשוט באינטרפולציית Bicubic.  
המעבר למרחב YCbCr חוסך לנו את השימוש ב-3 רשתות שונות עבור R,G,B בנפרד.

**SuperResolution\result\_510X510\_Color\_scale2\RepairImage.m:**

בקוד הבא נבצע בחינה של ה-Datasets השונים עבור תמונות צבעוניות. לשם כך נזכיר את תרשים מלבנים המציג את פעולת ה-Super Resolution על תמונה צבעונית:



נסביר את עיקרי השינויים בקוד:

* הפעם בודקים אם התמונה היא לא צבעונית ואם היא אכן לא צבעונית הופכים אותה לצבעונית ע"י שכפול הערכים לממדי RGB.
* נחתוך חלק של 510X510 מהתמונה.
* נמיר אותה ל-double ע"י im2double.
* את הפעולות האלה נבצע רק לשם חישוב PSNR ו-SSIM:  
  - נמיר את התמונה למרחב YCbCr ע"י שימוש בפונקציה rgb2ycbcr.  
  - נבודד את Y מתוך התמונה לקבלת ImageYDouble
* נקטין את התמונה ונגדיל אותה לקבלת התמונה המוגדלת במרחב RGB והיא UpsampledImageRGBDouble
* נמיר את התמונה למרחב YCbCr ע"י שימוש בפונקציה rgb2ycbcr לקבלת UpsampledImageYCbCrDouble
* נבודד את ה-Y מהתמונה ונקבל את UpsampledImageYDouble
* נכניס את UpsampledImageYDouble לרשת ונקבל במוצאה את ResidualImageYDouble
* נחבר את ResidualImageYDouble ואת UpsampledImageYDouble לקבלת ImageYDoubleRepairByNet
* נחשב PSNR ו-SSIM של ImageYDoubleRepairByNet ביחס ל-ImageYDouble.  
  ונחשב גם PSNR ו-SSIM של UpsampledImageYDouble
* ביחס ל-ImageYDouble.

**סופר רזולוציה - תמונות צבעוניות בגודל 512X512 עם יחס הגדלה של 3:**

בדיוק אותה התיקייה לעיל רק שהפעם עם יחס הגדלה של 3.

**סופר רזולוציה - תמונות צבעוניות בגודל 512X512 עם יחס הגדלה של 2 ורעש בסטיית תקן 5:**

**SuperResolution\result\_510X510\_scale2\_with\_Noise\_5\createDataSet.m:**

בחלק זה ניצור Dataset באתו אלגוריתם מפעמים קודמות רק שהפעם נרעיש את התמונה המוקטנת.  
לצורך פעולה זו פשוט נוסיף את השורות לביצוע הרעשת תמונה (הוספת הרעש גאוסי בסטיית תקן רצויה, המרת מספרים קטנים מ-0 ל-0, המרת מספרים גדולים מ-255 ל-255, עיגול הערכים) לאחר קבלת התמונה המוקטנת. מעבר לכך הכל אותו הדבר פשוט עם שמות משתנים שונים מעט.

**SuperResolution\result\_510X510\_scale2\_with\_Noise\_5\RepairImage.m:**

כעת בהשוואת התוצאות ישנה בעיה, לא נכון להשוות את תוצאות הרשת לאינטרפולציית Bicubic מכיוון שאינטרפולציית Bicubic לא מפחיתה רעש אלא רק מגדילה את התמונה.  
לכן, על מנת לקבל תמונה רחבה על אופציות שונות לביצוע פעולה של הפחתת רעש וסופר רזולוציה נשווה בין שלושה אלגוריתמים שונים:  
1. ניקוי רעשים מהתמונה הרועשת באמצעות הרשת שנלמדה מהחלק הראשון בפרויקט ולאחר מכן ביצוע של אינטרפולציית Bicubic להגדלת התמונה.  
2. ניקוי רעשים מהתמונה הרועשת באמצעות הרשת שנלמדה מהחלק הראשון בפרויקט ולאחר מכן הגדלה ע"י Super Resolution עם הרשת מהחלק השני.  
3. שימוש ברשת שאנו מאמנים עכשיו שמבצעת גם ניקוי רעשים וגם Super Resolution.

על מנת שנוכל לבצע השוואה זו כתבנו שתי פונקציות ב-:Matlab

* **DeNoiseOneImageFunction** - מקבלת תמונה עם רעש, מתקנת אותה לפי החלק הראשון של הפרויקט ומחזירה אותה.
* **SuperResolutionOneImageFunction** - מקבלת תמונה ומבצעת עליה תהליך של Super Resolution כפי שביצענו בחלק השני של הפרויקט.

בקוד עצמו:

* בחלק המקדים הראשון - יצרנו טבלאות PSNR ו-SSIM מתאימות עבור ה-Datasets שנבחנים ועבור האלגוריתמים השונים.
* בחלק המקדים השני – טענו לתוכנה את: המסננים, הרשת לביצוע ניקוי רעשים, מקדמים הנרמול לניקוי רעשים, הרשת שמבצעת סופר רזולוציה והרשת שמבצעת את שתי הפעולות יחד (ניקוי רעשים וסופר רזולוציה).
* כמובן ששוב הרעשנו את התמונה המוקטנת שאנו בוחנים.
* ביצענו בקוד את שלושת האלגוריתמים בנפרד.
* הכנסנו את התוצאות הממוצעות על כל Dataset לטבלה.

**סופר רזולוציה - תמונות צבעוניות בגודל 512X512 עם יחס הגדלה של 2 ורעש בסטיית תקן 10:**

בדיוק אותה התיקייה לעיל רק שהפעם עם רעש בסטיית תקן של 10.

**ה-GUI:**

ה-GUI הוא בעצם איחוד של כל התוצאות הטובות ביותר שקיבלנו בפרויקט.

בתיקיה מופיעים כל הנתונים שהם נשתמש:

* 7 רשתות ניקוי רעשים בסטיות תקן של: 1,2,5,10,15,20,25 מתמונות בגווני אפור.
* מקדמי הנרמול של ניקוי הרעשים בכל סטיות תקן אלה.
* 2 רשתות לביצוע סופר רזולוציה על תמונה צבעונית ביחסי הגדלה של 2 ו-3, כאשר הרזולוציה המלאה היא 510X510.
* 2 רשתות לביצוע ניקוי רעשים וסופר רזולוציה על תמונה בגווני אפור ביחסי הגדלה של 2 כאשר הרזולוציה המלאה היא 510X510.
* הקובץ SLT\_P לקבלת המסננים.
* פונקציה אשר מקבלת תמונה בגווני אפור רועשת, רשת לניקוי רעשים, נתונים מ-SLT\_P ומחזירה תמונה נקיה מרעשים.
* פונקציה אשר מקבלת תמונה צבעונית, רשת לביצוע סופר רזולוציה, יחס הגדלה ומבצעת את תהליך הסופר רזולוציה.
* קובץ אפליקציה – MyProjectApp.mlapp

**MyProjectApplication\MyProjectApp.mlapp:**

טעינת הרשתות והקבצים לעיל לתוך משתני תוכנה:



הפונקציה BrowseImageButtonPushed נקראת כאשר לוחצים על לחצן חיפוש התמונה במחשב.

* נפתח חלון שבו מחפשים את התמונה הרצויה מתוך כל הפורמטים בעזרת הפונקציה uigetfile.
* מרכיבים מהנתיב ונשם הקובץ את הנתיב לקובץ ובעזרתו בודקים אם הקובץ אכן קיים ונטען כראוי.  
  אם לא מוציאים חלון שגיאה באמצעות שילוב של uiwait ו-msgbox.
* אם התמונה קיימת אנו מוודאים שהיא צבעונית ואם לא אז הופכים אותה לצבעונית. אנו עושים זאת מכיוון שבמסך ה-GUI ניתן להציג רק תמונות צבעוניות.

הפונקציה ActionButtonGroupSelectionChanged נקראת כאשר נבחרת פעולה רצויה לביצוע ע"י המשתמש.

* מוחקים את התמונה מה-Output Image ועל מנת שפעולה זו תבוצע מיד אנו כותבים את הפקודה drawnow.
* מוודאים שקיימת תמונה ב-InputImage ולא סתם לחצו על פעולה רצויה. אם לא קיימת תמונה תוצג הודעת שגיאה למסך.
* בודקים אם נדרש לבצע ניקוי רעשים, אם כן:
* לוקחים את התמונה מה-Input Image.
* ממירים אותה לגווני אפור (כי ניקוי רעשים עובד רק על גווני אפור).
* בחירת הרשת המתאימה ומקדמי הנרמול המתאימים בעזרת switch-case.
* אחרת, אם נדרש לבצע פעולה עם יחס הגדלה של 2:
* חותכים את התמונה לגודל של 255X255.
* אם התמונה קטנה מגודל זה תוצג הודעת שגיאה שהתמונה חייבת להיות לפחות בגודל 255X255.
* אם התמונה בגודל המתאים בוחרים את הרשת המתאימה לביצוע בעזרת switch-case.
* אחרת, אם נדרש לבצע סופר רזולוציה עם יחס הגדלה של 3:
* חותכים את התמונה לגודל של 170X170.
* אם התמונה קטנה מגודל זה תוצג הודעת שגיאה שהתמונה חייבת להיות לפחות בגודל 170X170.
* אם התמונה בגודל המתאים מבצעים את הפעולה.