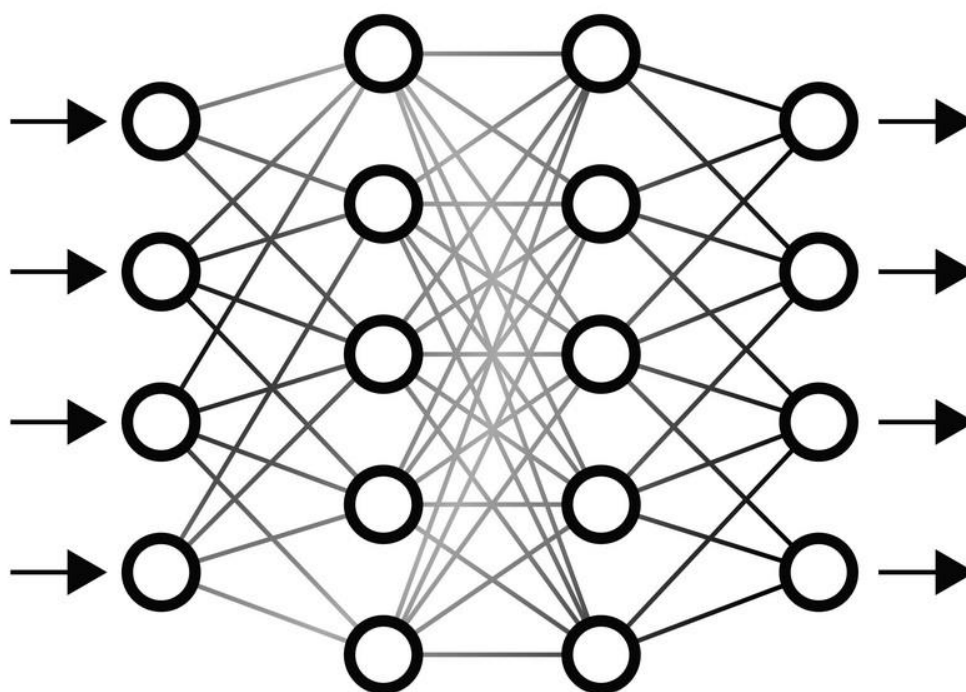


דוח פרויקט סוף – כשל חומרים ראייה ממוחשבת בפרקטוגרפיה

מגישים:

יונתן פצ'ו
חוליאן לוינטון
דניאל סופר





תוכן עניינים

3.....	הקדמה
3	מוטיבציה וצורך טכנולוגי
3.....	למידה עמוקה ומערכות לומדות
4.....	בסיס מתמטי
4	שכבת קונבולוציה
5	שכבת פולינג
6	פתרון סטטיסטי
6	השגיאה והקטנתה
7.....	הפרויקט
7	בסיס הנתונים
8	שיטות למידה
10	קוד
11	תוצאות וביצועים
12	השפעת איכות התמונה על ביצועי המערכת
13	כיווני פיתוח עתידיים
14.....	סיכום
15.....	נספחים

הקדמה

בפרויקט זה נרצה ליצור שיטה לראייה ממוחשבת בפרקטוגרפיה ובקרת איכות של חומרים. המטרה הסופית הינה קבלת מאגר מידע של תמונות משטחי שבירה או תמונות של פגמים נפוצים ונתבקש לבנות או להשתמש בכלי ראייה ממוחשבת לשם ניתוח התמונות. לשם כך, נעזרנו בכלים וידע מתחום המערכות הלומדות, רשת קיימת מאומנת על בסיס נתונים רחב ומאגר תמונות של משטחי שבירה ויצרנו שיטה המסווגת תמונות פרקטוגרפיה תחת שתי תגיות: שבר פריך ושבר משיך. על מנת להסביר את הנעשה בפרויקט זה, נרצה תחילה לספק את המוטיבציה ואת הבסיס לעולם הלמידה העמוקה.

מוטיבציה וצורך טכנולוגי

הגדרת הבעיה: נדרש להבדיל בין שבר משיך ולפריך, בצורה מהירה וללא התערבות האדם.

בימינו, חוקרי פרקטוגרפיה ובקרת איכות של חומרים רוצים לתעש תהליכים של פיתוח חומרים ותהליכי עיבוד תוצאות ניסויים על מנת ליצור תהליך מהיר יותר. הרצון הוא ליצור שיטות ממוחשבות או רובוטיות אשר יקבלו אליהם כקלט תוצאות ניסוי או תמונות מדגם מסוים ויספקו כפלט את מרחב הפרמטרים המייצג את הנתונים. התהליך המיקרוסקופי יכול להיות רובוטי/ממוחשב, מה שיביא לפיתוח תהליך מהיר יותר של חומרים ובקרת איכות בזמן אמת.

למידה עמוקה ומערכות לומדות

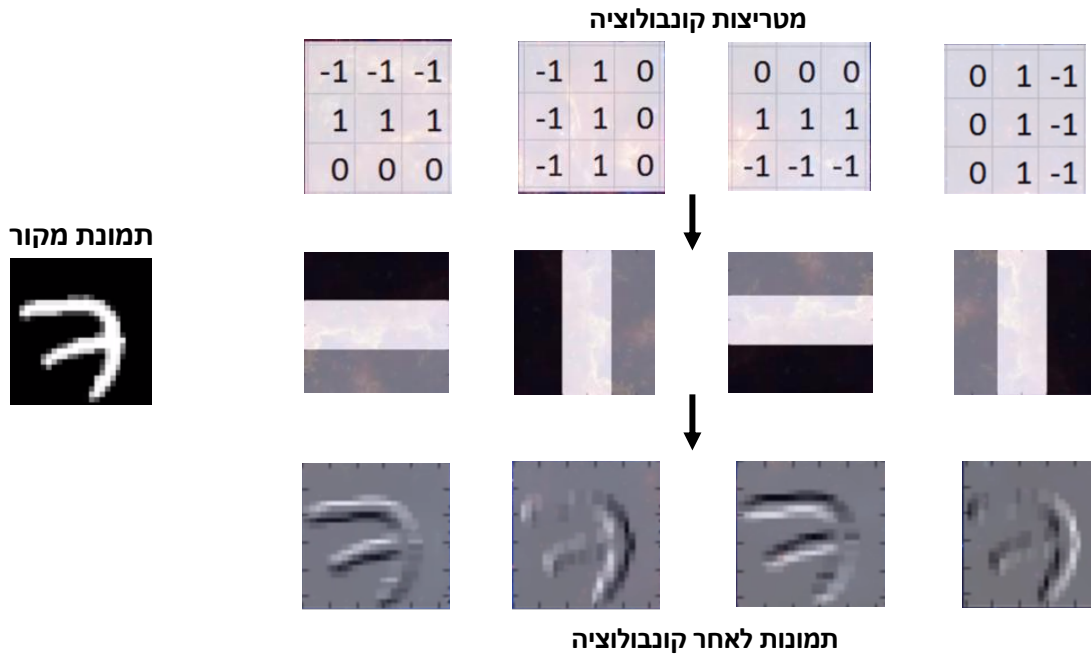
למידה עמוקה (Deep learning) שייך לעולם הבינה המלאכותית. מטרת הבינה המלאכותית היא ליצור טכניקות שיאפשרו למחשבים לחקות התנהגויות ופעולות של האדם ואף ליצור ביצועים משופרים לפעולות אלו.

תחת הבינה המלאכותית נמצא תחום למידת המכונה (Machine learning). תחום זה הוא תת תחום במדעי המחשב ובינה מלאכותית המשיך לתחומי הסטטיסטיקה והאופטימיזציה. התחום עוסק בפיתוח אלגוריתמים המיועדים לאפשר למחשב ללמוד מתוך דוגמאות, ופועל במגוון משימות חישוביות בהם התכנות הקלאסי אינו אפשרי. המטרה המרכזית של Machine learning היא טיפול ממוחשב בנתונים מן העולם האמיתי עבור בעיה מסוימת, כאשר לא ניתן לכתוב תוכנת מחשב עבורה. למשל, בעיית זיהוי שמומחה אנושי מסוגל לפתור אך לא מסוגל לכתוב את הכללים לזיהוי בצורה מפורשת, או שהם משתנים עם הזמן ולא ניתנים לכתיבה מראש. המטרה היא מידול, חיזוי או גילוי של עובדות לגבי העולם האמיתי.

בתוך תחום למידת המכונה, נמצאת הלמידה העמוקה (Deep learning). למידה עמוקה מתבצע על ידי רשת נוירונים (Artificial Neural Network) אשר הוא מודל מתמטי חישובי שפותח בהשראת תהליכים מוחיים או קוגניטיביים המתרחשים ברשת עצבית טבעית. רשת מסוג זה מכילה בדרך כלל מספר של יחידות מידע (קלט ופלט) במקושרות זו לזו. צורת הקישור בין היחידות, המכילות מידע על חוזר הקשר, מדמה את אופן החיבור בין הנוירונים במוח. השימושים של שיטה זו כוללים מגוון משימות כגון: זיהוי תווים, זיהוי פנים, זיהוי כתב יד, חיזוי שוק ההון, מערכת זיהוי דיבור, ניתוח טקסטים ועוד.



שימוש של מטריצות קובבולוציה בסיסיות לזיהוי מאפיינים בסיסים – קווי מתאר אנכיים ואופקיים:

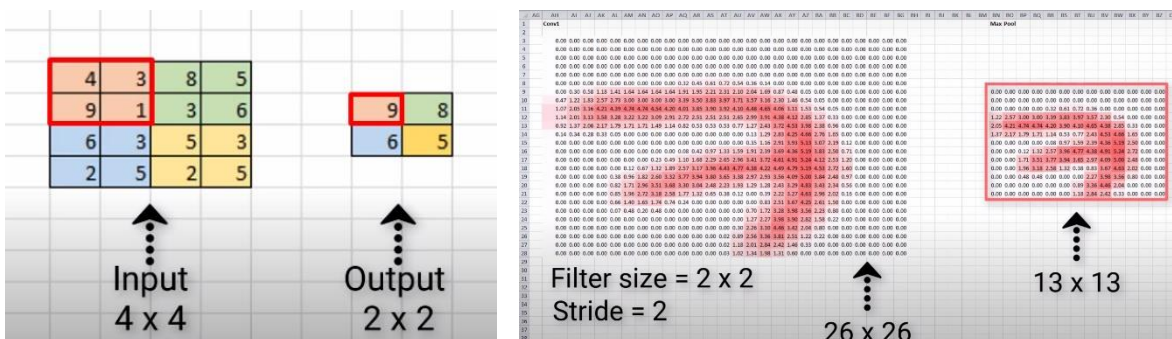


ניתן לראות בתמונות לעיל שהתמונה לאחר שימוש במטריצת הקובבולוציה מבליטה דברים מסוימים (נצבעים בלבן), למשל התמונה השמאלית, ניתן לזהות שקווי המתאר האופקיים העליונים נצבעו בלבן.

ככל שנשתמש במטריצות קובבולוציה מסובכות יותר, נוכל לזהות מאפיינים מורכבים יותר (חלקים עגולים, קווים חדים, כנפיים, מקור וכו'). זיהוי מאפיינים רבים ומסובכים עוזר למערכת לקבל החלטה טובה יותר באשר למה שהיא רואה. כלומר נקבל פתרון סטטיסטי טוב יותר.

שכבת פולינג:

שכבה נוספת שעניינה הקטנת זמן החישוב, היא שכבת פולינג. הבסיס מאחורי כל שכבת פולינג, הוא הקטנת נפח המידע אותו יש לעבד תוך שמירה על החלקים החשובים שבמידע. למשל ב- max pooling, מחלקים את המטריצה (התמונה) לתתי מטריצות ומכל תת מטריצה מחלצים את הערך המקסימלי. בדוגמה להלן ניתן לראות כיצד הצורה של המספר 7 נשמרה אבל הנפח קטן (מטריצה 13X13 במקום 26X26).





leopard



פתרון סטטיסטי

המערכת כאמור משקללת את המאפיינים שהיא זיהתה ובחרת את המשקל שיש לתת לכל מאפיין בגיבוש ההחלטה. התוצאה הסופית המתקבלת היא פתרון סטטיסטי (באיור להלן), שמייצג את התפלגות הודאות של המערכת בפתרון.

למשל בתמונה להלן ניתן לראות שהמערכת קיבלה תמונה של נמר, והיא בטוחה ב-70% שמה שהיא רואה זה נמר, ובטוחה ב-20% שזהו יגואר ו-7 אחוז צ'יטה, ו-3 אחוז נמר שלג.

השגיאה

ישנן מספר דרכים להגדיר את השגיאה, הדרך הבסיסית היא המרחק הריבועי בין הפתרון הסטטיסטי לפתרון האנליטי. (במקרה של הדוגמה לעיל – 100% נמר).

$$err = (1 - 0.7)^2 + (0 - 0.2)^2 + (0 - 0.07)^2 + (0 - 0.03)^2 + (0 - 0)^2$$

השגיאה תלויה הן בפתרון הסטטיסטי והן בפתרון האנליטי. כאשר הפתרון הסטטיסטי עצמו תלוי בכל אותם הפרמטרים שהוגדרו לפני כן בשכבות.

הקטנת השגיאה – Gradient descent

מאחר והפתרון הסטטיסטי הוא פונקציה של הפרמטרים המתמטיים שבאמצעותם הוא חושב, פונקציית השגיאה עצמה היא פונקציה של הפרמטרים המתמטיים השונים (למשל הפרמטרים של פונקציית הקובולוציה, או פונקציית המשקלים של המאפיינים בהסקת הפתרון). עבור פונקציית שגיאה חיובית, הקטנת השגיאה שקולה למציאת הגרדיאנט המינימלי (המספר השלילי עם הגודל הכי גדול) של פונקציית השגיאה. הקטנת השגיאה בעצם מביאה לידי שינוי השכבות (שבתורן יניבו פתרון טוב יותר).

איטרציה

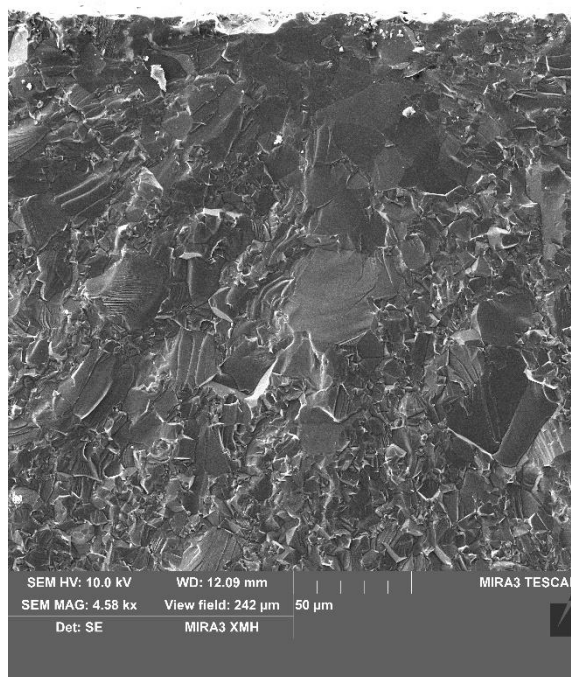
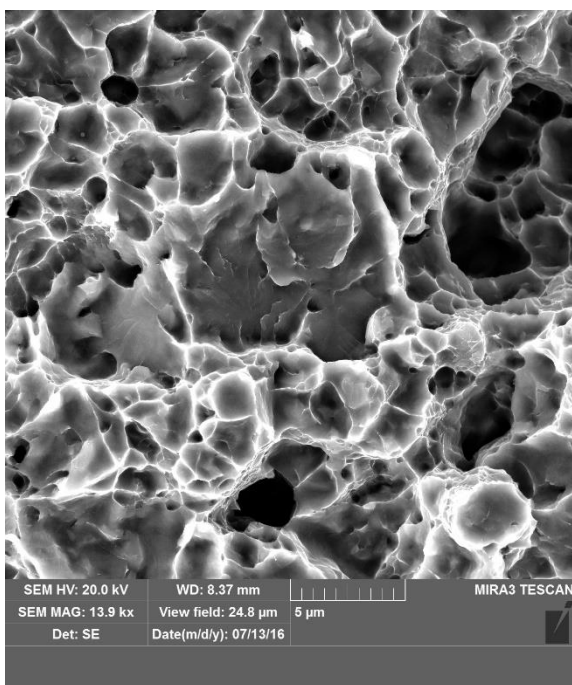
כעת המערכת חוזרת על התהליך עם שכבות טובות יותר, ומקבלת פתרון סטטיסטי טוב יותר (בתקוה). התהליך חוזר על עצמו תוך שיפור השכבות מפעם לפעם ובכך משפר את יכולת הזיהוי של המערכת, עד לתנאי עצירה.

הפרויקט

בסיס הנתונים:

לצורך הפרויקט השתמשנו בבסיס נתונים מתויג שסופק לנו ע"י פרופ' אוסובסקי המכיל:

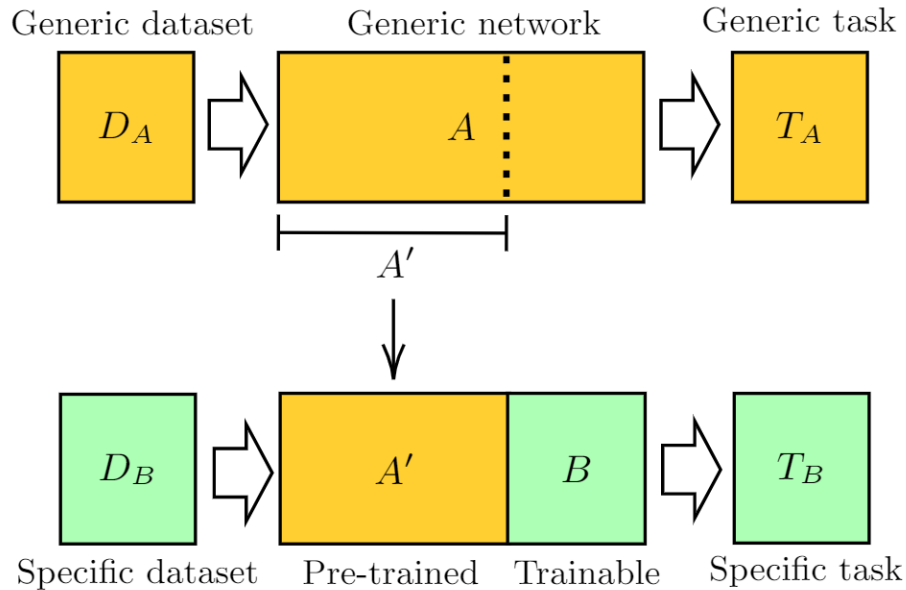
- 20 תמונות SEM של שברים פריכים בסקאלות שונות
- 42 תמונות SEM של שברים משיכים בסקאלות שונות



את בסיס הנתונים חילקנו לשני חלקים, אימון ובדיקה ביחס חצי-חצי, זאת על מנת שנקבל מספיק מידע לאמן את הרשת שלנו באמצעותו וגם נקבל מספיק מידע שנוכל לבדוק באמצעותו את הדיוק שנערוך בסוף האימון בצורה מייצגת.

שיטות למידה:

מכיוון שברשותנו בסיס נתונים קטן יחסית לתחום הלמידה העמוקה, בו בסיסי הנתונים מגיעים למאות אלפים ומיליוני פריטים, בחרנו לעבוד בשיטת Transfer Learning. עקרון השיטה הינו בחירת רשת קיימת ומאומנת על מידע רב ומגוון, מחיקת השכבה האחרונה (שכבה לינארית בסיסית) וחיבור שכבה לינארית חדשה בעלת מספר נייטרונים כמספר התגיות בבסיס המידע שלנו.

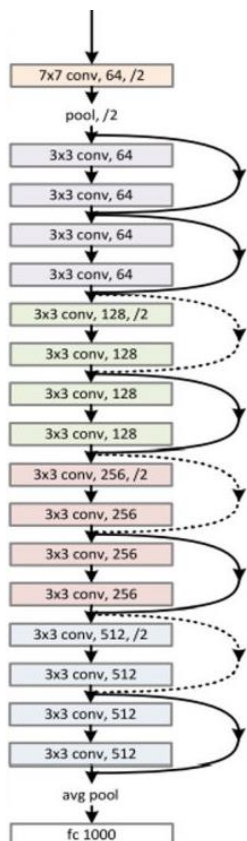


בדרך זו אנו משתמשים ברשת המאומנת בתור Feature Extractor המזהה דפוסים בולטים בתמונה אשר היא מקבלת כפלט, לאחר זיהוי הפיצ'רים הבולטים, השכבה האחרונה החדשה מתאמנת על בסיס המידע ומבצעת שקלול עבור חשיבות כל פיצ'ר לתגית המתאימה (תהליך זה הינו מציאת המשקלים לקשתות המחוברות בין השכבה האחרונה לקודמתה).

בריוקט זה בחרנו להשתמש ברשת ResNet18, נציג את ארכיטקטורת הרשת:

Layer Name	Output Size	ResNet-18
conv1	$112 \times 112 \times 64$	$7 \times 7, 64, \text{stride } 2$
conv2_x	$56 \times 56 \times 64$	$3 \times 3 \text{ max pool, stride } 2$
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$
conv3_x	$28 \times 28 \times 128$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$
		$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
conv4_x	$14 \times 14 \times 256$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$
		$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
conv5_x	$7 \times 7 \times 512$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$
average pool	$1 \times 1 \times 512$	$7 \times 7 \text{ average pool}$
fully connected	1000	$512 \times 1000 \text{ fully connections}$
softmax	1000	

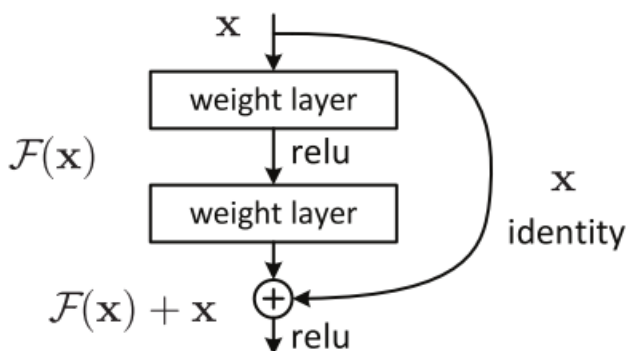
רשת זו בעלת 17 שכבות Convolution וזוג שכבות Pooling כמתואר בסכמה:



תכונה המייחדת את משפחת Residual Neural Networks או בקיצור, ResNet היא תכונת Skip Connections, כפי שניתן לראות בסכמה קיים מעבר מידע בין השכבות ללא מעבר בשכבה כלשהי או בפונקציית נרמול.

תכונה זו מאפשרת לרשת להיות גם פשוטה וגם לזהות מאפיינים מורכבים, שהולכים לאיבוד ברשתות עמוקות לאחר מעבר במספר רב של שכבות.

ניתן לראות סכמה מפורטת של בלוק בעל Skip Connection:





קוד:

בפריקט השתמשנו בשפת Python ובספריית Pytorch – Deep Learning. נציג בחלק זה את עיקר הקוד שכתבנו לביצוע משימה זו, בנוסף הקוד המלא מצורף בנספחים. עבור כל מקטע קוד מסומן, קיים הסבר כללי לתכליתו בקוד.

```
if __name__ == "__main__":
    Train_New = True
    plt.ion()
    data_transforms = {
        'train': transforms.Compose([
            transforms.Resize(1280),
            transforms.CenterCrop(1000),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]),
        'val': transforms.Compose([
            transforms.Resize(1280),
            transforms.CenterCrop(1000),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]),
    }

    data_dir = 'E:\Semester VI\Failure of Materials\dataset\mk2_jpeg'
    image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                                    data_transforms[x])
                      for x in ['train', 'val']}
    dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                                    shuffle=True, num_workers=4)
                  for x in ['train', 'val']}
    dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
    class_names = image_datasets['train'].classes
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    ## Display images
    inputs, classes = next(iter(dataloaders['train']))

    # Make a grid from batch
    out = torchvision.utils.make_grid(inputs)

    if Train_New:
        model_conv = torchvision.models.resnet18(pretrained=True)
        for param in model_conv.parameters():
            param.requires_grad = False

    # Parameters of newly constructed modules have requires_grad=True by default
    num_ftrs = model_conv.fc.in_features
    model_conv.fc = nn.Linear(num_ftrs, 2)

    model_conv = model_conv.to(device)

    criterion = nn.CrossEntropyLoss()

    # Observe that only parameters of final layer are being optimized as
    # opposed to before.
    optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

    # Decay LR by a factor of 0.1 every 7 epochs
    exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)

    model_conv = train_model(model_conv, criterion, optimizer_conv,
                              exp_lr_scheduler, num_epochs=3)

    PATH = './PetchNet.pth'
    torch.save(model_conv, PATH)
else:
    PATH = './PetchNet.pth'
    model_conv = torch.load(PATH)
```

1

2

3

4

5

6

7

8

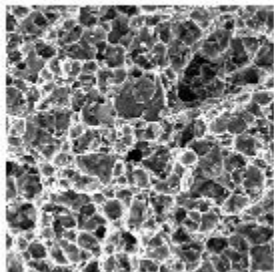
הקוד לפי חלקים:

1. הגדרת טרנספורמציות לתמונות
 - חיתוך התמונות והתאמת רזולוציה על מנת ליצור input אחיד בבניסה לרשת
 - התמרת פורמט התמונה לטיפוס "Tensor" (מטריצה בספריית Pytorch)
2. טעינת בסיס הנתונים לתוכנה וביצוע הטרנספורמציות שתוארו בסעיף 1
3. חלוקת המידע לזוג וקטורים: וקטור טכורים של תמונות ווקטור מספרים המתאר את סוג התמונה (משיך/פריך)
4. טעינת הרשת המאומנת ResNet
5. הוספת שכבה לרשת
6. הגדרת פרמטרי אימון
7. אימון הרשת
8. שמירת הרשת המאומנת

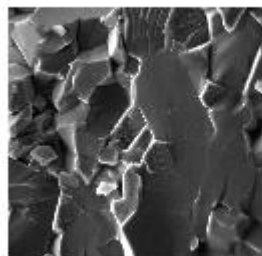
תוצאות וביצועים:

לאחר אימון הרשת שלנו, בדקנו אותה על סט הבדיקה שכלל כ-30 תמונות של שברים משיכים ופריכים
 קיבלנו דיוק של 96%
 נציג מדגם מתמונות הבדיקה יחד עם תיוג החיזוי של הרשת שלנו:

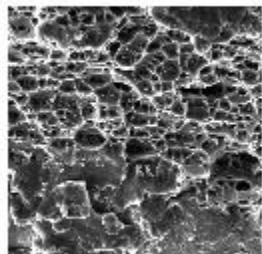
predicted: ductile



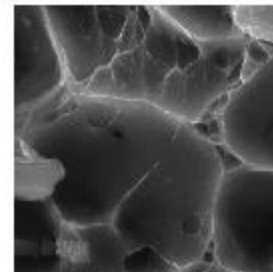
predicted: brittle



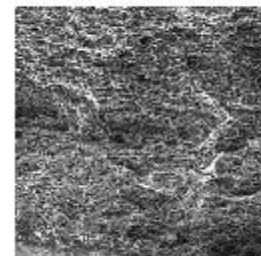
predicted: ductile



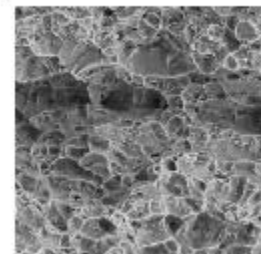
predicted: brittle



predicted: ductile



predicted: ductile



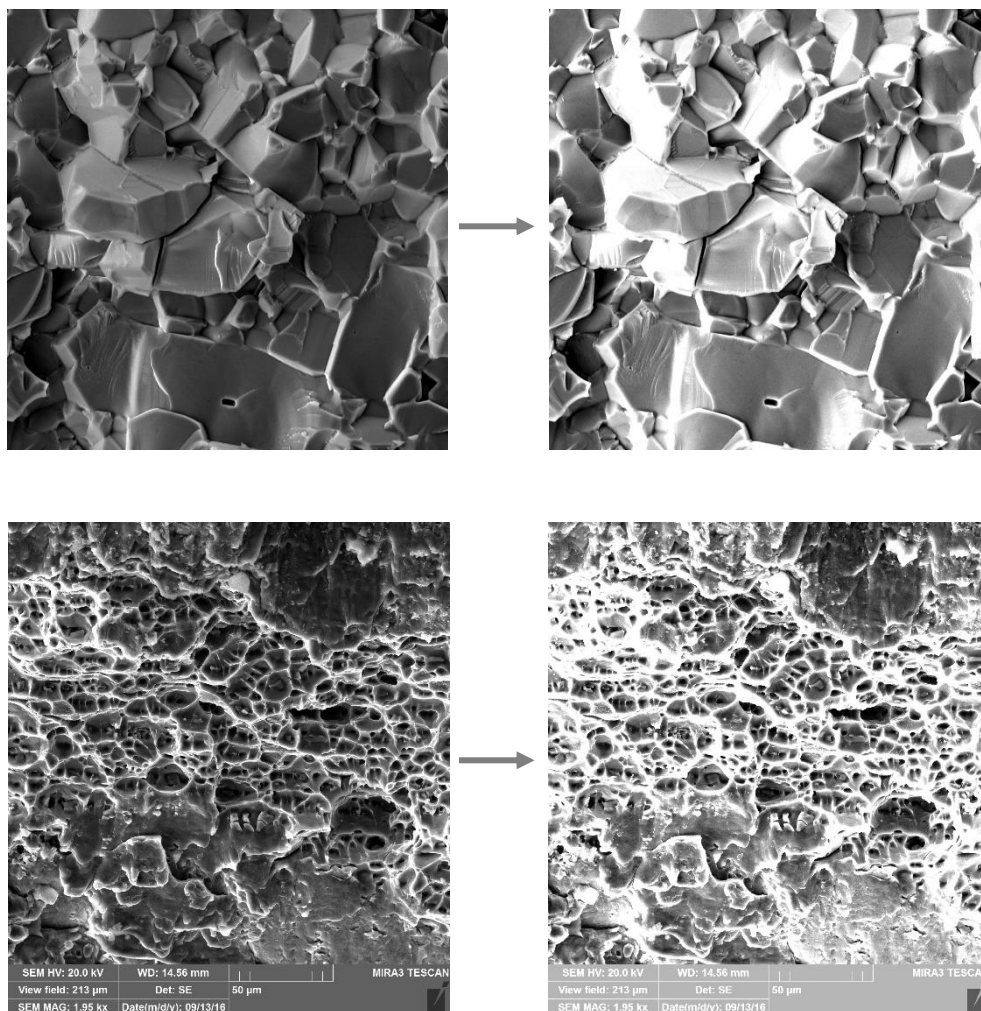
השפעת איכות התמונה על ביצועי המערכת:

פרמטר אחד בו נבחר על מנת לבדוק את השפעתו על דיוק וביצועי המערכת הוא איכות התמונה, איכות הנתונים המשמשים כקלט למודל. נרצה לבדוק האם חשיפת יתר של אור (over saturation) עלולה להשפיע על הביצועים ואחוזי הדיוק.

ההשערה ראשונית שלנו היא שאחוזי הדיוק לא ירדו בצורה דרסטית (1% - 3%) זאת מכיוון שחשיפת יתר בתמונה גורמת לאיבוד מידע בפרטים הקטנים ובפועל לא פוגמת "בקווי המתאר" של השבר.

הרשת שלנו מבדילה בין סוגי השברים ע"י זיהוי קווי המתאר המתאימים לכל סוג שבר (קווי ישרים – שבר פריך, עיגולים ואליפסות – שבר משיך) ולכן חשיפת היתר לא תגרום לאיבוד הרבה מידע הרלוונטי לניתוח של הרשת שלנו.

על מנת לאמת או לסתור השערה זו, נשנה את סט הבדיקה על מנת שיחווה תופעה מלאכותית של over saturation:



נבדוק את ביצועי הרשת מחדש על מנת לבדוק את אחוזי הדיוק החדשים:

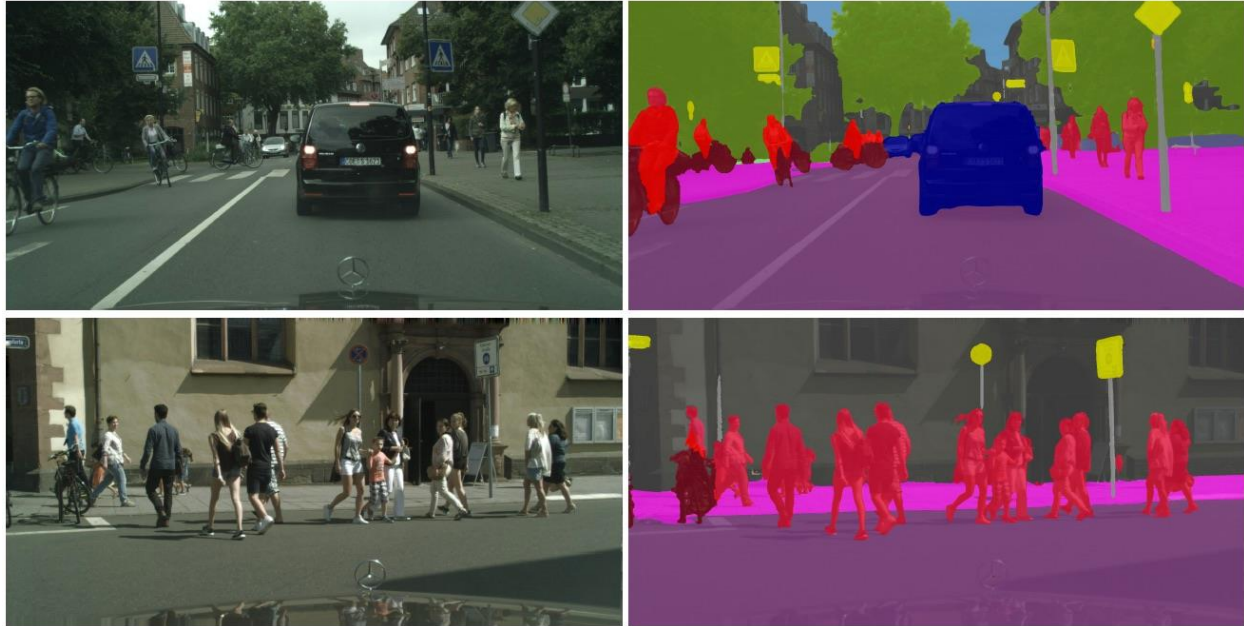
קיבלנו דיוק של 94%

כלומר, ההשערה שלנו לגבי הדיוק נכונה, עם זאת ייתכן וקיימת סיבות אחרות להצלחת הרשת בזיהוי לאחר חשיפת היתר, אך נכון לעכשיו אין באפשרותנו לומר בצורה חד משמעית את הסיבה.

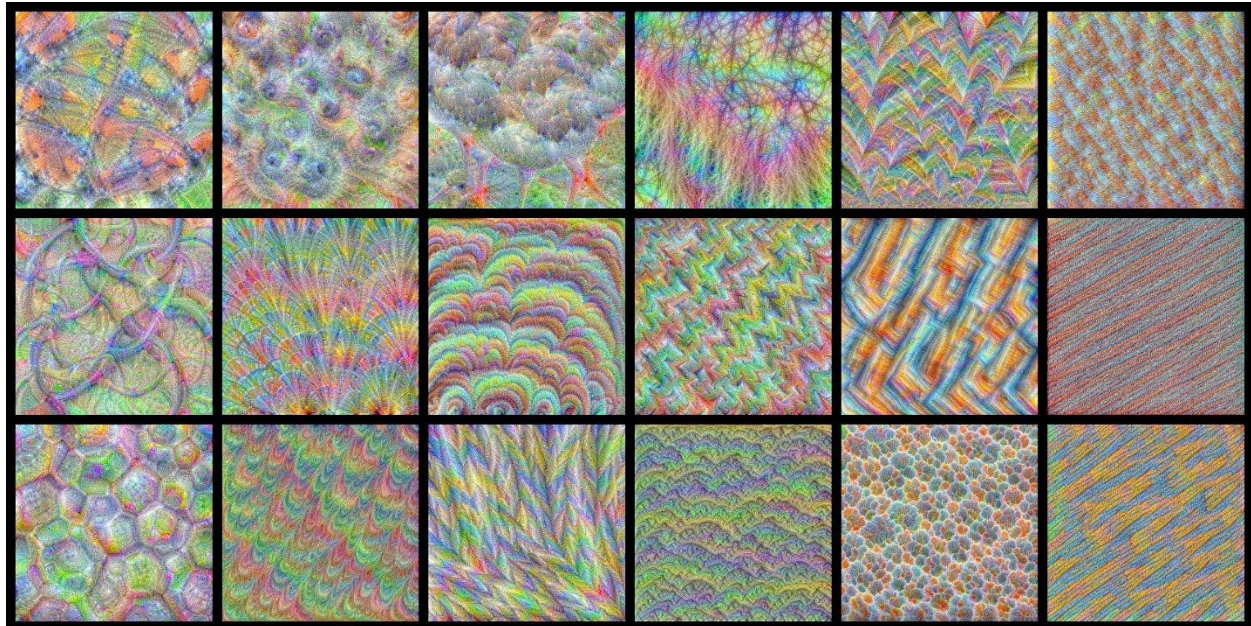


כיווני פיתוח אפשריים:

- הרחבת בסיס הנתונים למספר סוגים שברים נוספים, לדוגמה שברי התעייפות.
 - ביצוע סגמנטציה לאזורים פריכים ואזורים משיכים בפני שבר מעורבים באמצעות הרשת המאומנת שלנו.
- דוגמה לסגמנטציה בראייה ממוחשבת:



- זיהוי הפיצ'רים (Convolution Filters) המובילים ברשת זיהוי השברים שלנו, ובדיקת ההתאמה לתיאוריה.
- דוגמה לפיצ'רים אשר התקבלו ברשת ResNet50:



סיכום

בפרויקט זה פיתחנו שיטה לראייה ממוחשבת של פרקטוגרפיה ומשטחי שבירה בעזרתה ניתן לסווג בין שני סוגי שבר: משיך ופריך. המערכת מקבלת בקלט תמונת SEM של משטח שבירה ומוציאה כפלט את סוג השבר שהתרחש. המערכת מבצעת זאת על ידי שימוש ברשת נוירונים אשר מאומנת לזיהוי תבניות על בסיס מידע הכולל מיליוני תמונות. השכבה האחרונה של הרשת המאומנת מספקת תבניות אשר קיימות בתמונה הנתונה והשכבה אשר מסווגת את התמונה, משכללת את המשקלים של התבניות הנתונות ומספקת תוצאה לסוג השבר.

לאחר אימון הרשת החדשה על מאגר של כ-30 תמונות פרקטוגרפיה שקיבלנו עבור הפרויקט, ביצענו בדיקה עבור סט תמונות אחר שכלל כעוד 30 תמונות וקיבלנו דיוק בתוצאות של 96%. קל לראות כי קיבלנו דיוק גבוהה בתוצאות הניתוח של הרשת. זאת בזכות מספר מאפיינים: רשת ה-ResNet בה השתמשנו בבסיס מאומנת על מספר רב מאוד של תמונות כך שמסוגלת להבדיל בין תבניות ודפוסים רבים בתמונה נתונה. בנוסף, הסיווג הסופי שלנו מסווג בין שני סוגים של שברים כך שהדיוק המינימלי נתון על ידי 50%.

ככיווני פיתוח עתידיים, ניתן לשפר את המערכת שפיתחנו (להגדיל את אחוזי הדיוק שלה) על ידי הגדלת בסיס הנתונים ואימון הרשת על מספר גדול יותר של תמונות. בנוסף, ניתן לבצע סיווג של חלקים בתוך משטח השבירה כך שיזהה בין אזורים של שבר משיך ואזורים של שבר פריך. פיתוח נוסף יכול להיות בדיקת התאמה של אופן הסיווג לתיאוריה על ידי חילוף המאפיינים לפיהם הרשת מסווגת את התמונות.

נספחים

לינק להורדת הקוד:

https://drive.google.com/drive/folders/1kxntmr2fhuZFWprnm0-pe_k2-Mib_Z?usp=sharing

לינק להורדת הdataset:

<https://drive.google.com/drive/folders/1TIKfu-FdPDHFAAbL5rYypZus78WTSWdi?usp=sharing>

לכל בעיה ועזרה בנוגע לקוד ניתן לפנות למייל : ypetcho@gmail.com

```
from __future__ import print_function, division
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim import lr_scheduler
import numpy as np
import torchvision
from torchvision import datasets, models, transforms
import matplotlib.pyplot as plt
import seaborn as sns
import time
import os
import copy

def imshow(inp, title=None):
    """Imshow for Tensor."""
    inp = inp.numpy().transpose((1, 2, 0))
    mean = np.array([0.485, 0.456, 0.406])
    std = np.array([0.229, 0.224, 0.225])
    inp = std * inp + mean
    inp = np.clip(inp, 0, 1)
    plt.imshow(inp)
    if title is not None:
        plt.title(title)
    plt.pause(0.001) # pause a bit so that plots are updated

def train_model(model, criterion, optimizer, scheduler, num_epochs=25):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                model.train() # Set model to training mode
            else:
                model.eval() # Set model to evaluate mode

            running_loss = 0.0
            running_corrects = 0

            # Iterate over data.
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)

                # zero the parameter gradients
                optimizer.zero_grad()

                # forward
                # track history if only in train
```



```
with torch.set_grad_enabled(phase == 'train'):  
    outputs = model(inputs)  
    _, preds = torch.max(outputs, 1)  
    loss = criterion(outputs, labels)  
  
    # backward + optimize only if in training phase  
    if phase == 'train':  
        loss.backward()  
        optimizer.step()  
  
    # statistics  
    running_loss += loss.item() * inputs.size(0)  
    running_corrects += torch.sum(preds == labels.data)  
    if phase == 'train':  
        scheduler.step()  
  
epoch_loss = running_loss / dataset_sizes[phase]  
epoch_acc = running_corrects.double() / dataset_sizes[phase]  
  
print('{} Loss: {:.4f} Acc: {:.4f}'.format(  
    phase, epoch_loss, epoch_acc))  
  
# deep copy the model  
if phase == 'val' and epoch_acc > best_acc:  
    best_acc = epoch_acc  
    best_model_wts = copy.deepcopy(model.state_dict())  
  
print()  
  
time_elapsed = time.time() - since  
print('Training complete in {:.0f}m {:.0f}s'.format(  
    time_elapsed // 60, time_elapsed % 60))  
print('Best val Acc: {:.4f}'.format(best_acc))  
  
# load best model weights  
model.load_state_dict(best_model_wts)  
return model  
  
def visualize_model(model, num_images=6):  
    was_training = model.training  
    model.eval()  
    images_so_far = 0  
    fig = plt.figure()  
    with torch.no_grad():  
        for i, (inputs, labels) in enumerate(dataloaders['val']):  
            inputs = inputs.to(device)  
            labels = labels.to(device)  
  
            outputs = model(inputs)  
            _, preds = torch.max(outputs, 1)  
  
            for j in range(inputs.size()[0]):  
                images_so_far += 1  
                ax = plt.subplot(num_images//2, 2, images_so_far)  
                ax.axis('off')  
                ax.set_title('predicted: {}'.format(class_names[preds[j]]))  
                imshow(inputs.cpu().data[j])  
  
            if images_so_far == num_images:  
                model.train(mode=was_training)  
                return  
    model.train(mode=was_training)  
  
def show_architecture(model):  
    model_weights = [] # we will save the conv layer weights in this list  
    conv_layers = [] # we will save the 49 conv layers in this list  
  
    # get all the model children as list  
    model_children = list(model.children())  
    # counter to keep count of the conv layers  
    counter = 0  
  
    # append all the conv layers and their respective weights to the list  
    for i in range(len(model_children)):  
        if type(model_children[i]) == nn.Conv2d:  
            counter += 1  
            model_weights.append(model_children[i].weight)  
            conv_layers.append(model_children[i])  
        elif type(model_children[i]) == nn.Sequential:
```




```
        for j in range(len(model_children[i])):
            for child in model_children[i][j].children():
                if type(child) == nn.Conv2d:
                    counter += 1
                    model_weights.append(child.weight)
                    conv_layers.append(child)
# print(f"Total convolutional layers: {counter}")

# take a look at the conv layers and the respective weights
for weight, conv in zip(model_weights, conv_layers):
    # print(f"WEIGHT: {weight} \nSHAPE: {weight.shape}")
    print(f"CONV: {conv} ==> SHAPE: {weight.shape}")
# print(model_children)
# print(model)

if __name__ == "__main__":
    Train_New = True
    plt.ion() # interactive mode
    data_transforms = {
        'train': transforms.Compose([
            transforms.Resize(1280),
            transforms.CenterCrop(1000),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]),
        'val': transforms.Compose([
            transforms.Resize(1280),
            transforms.CenterCrop(1000),
            transforms.ToTensor(),
            transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
        ]),
    }
    data_dir = 'E:\Semester VI\Failure of Materials\dataset\mk2_jpeg'
    image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
                                                    data_transforms[x])
                      for x in ['train', 'val']}
    dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x], batch_size=4,
                                                    shuffle=True, num_workers=4)
                   for x in ['train', 'val']}
    dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
    class_names = image_datasets['train'].classes
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

    ## Display images
    inputs, classes = next(iter(dataloaders['train']))

    # Make a grid from batch
    out = torchvision.utils.make_grid(inputs)

    #imshow(out, title=[class_names[x] for x in classes])

    if Train_New:
        model_conv = torchvision.models.resnet18(pretrained=True)
        for param in model_conv.parameters():
            param.requires_grad = False

        # Parameters of newly constructed modules have requires_grad=True by default
        num_ftrs = model_conv.fc.in_features
        model_conv.fc = nn.Linear(num_ftrs, 2)

        model_conv = model_conv.to(device)

        criterion = nn.CrossEntropyLoss()

        # Observe that only parameters of final layer are being optimized as
        # opposed to before.
        optimizer_conv = optim.SGD(model_conv.fc.parameters(), lr=0.001, momentum=0.9)

        # Decay LR by a factor of 0.1 every 7 epochs
        exp_lr_scheduler = lr_scheduler.StepLR(optimizer_conv, step_size=7, gamma=0.1)
        model_conv = train_model(model_conv, criterion, optimizer_conv,
                                exp_lr_scheduler, num_epochs=3)

        PATH = './PetchNet.pth'
        # torch.save(model_conv.state_dict(), PATH)
        torch.save(model_conv, PATH)
    else:
        PATH = './PetchNet.pth'
        model_conv = torch.load(PATH)
        visualize_model(model_conv)
        show_architecture(model_conv)
```