

פרויקט מעשי במבוא לבינה מלאכותית

נושא: זיהוי זכר או נקבה

מגישים: אריאל ורביין, ניתאי דגלושן.

מרצה: ראובן כהן.

מתרגם: דניאל אביצדור.

תאריך הגשה: 15 לאוקטובר 2023.

תוכן עניינים

מבוא	עמוד 2
שלב ראשון – איסוף המידע והכנתו	עמוד 5
שלב שני – טרנספורמציה ואוגומנטציה	עמוד 7
שלב שלישי – בחירת מודל לאימון	עמוד 10
שלב רביעי – בניית המודל	עמוד 13
שלב חמישי – טיפול בהתאמת יתר	עמוד 18
שלב שישי – בחירת היפר פרמטרים	עמוד 22
שלב שביעי – ניתוח התוצאות	עמוד 31
שימוש במודל	עמוד 37
הערות	עמוד 39

מבוא

במסגרת הפרויקט המעניינו בחרנו לבנות אלגוריתם למידה לקביעה האם אדם הוא גבר או נקבה על סמך תמונה.

בחרנו לעסוק בנושא זה משום שאנו מאמינים כי הוא מהוות נקודת מוצא טובה לעולם של ראייה ממוחשבת, ואנו מאמינים כי הוא אפשר לנו להתנסות בתהליכי הבנייה של אלגוריתם למידה. התוצר הסופי של הפרויקט יוכל תוכנה אשר מכריעה האם האדם בתמונה הוא זכר או נקבה.

על מנת למשוך זאת, נפרט את השלבים העיקריים אותם נבעו ונסביר עליהם בקצרה.

1. איסוף המידע והכנתו

בשלב זה, נחפש דאטאסט הכלול במספר רב של תמונות עם אנשים, ותיוגן לפי "זכר" או "נקבה". עליינו לוודא כי הדאטאסט שמצאנו הוא מאוון. ככלומר מספר התמונות עם גברים, שווה למספר התמונות עם נשים. איזון הדאטאסט הוא חשוב כדי להימנע מהטיה (bias) של המודל – למשל אם מספר התמונות של גברים גדול ממספר התמונות של נשים, המודל עשוי להיות "אופטימי" מידי בניבו, מה שוביל לביצועים גרועים בעת זיהוי פרוצופים של נשים.

לאחר שאספנו את הדאטאסט, נחלק אותו לשתי קבוצות : training and testing . ואת ההשלה לשתי קבוצות validation training : עליינו לחלק את הדאטאסט ביחסים נכונים כפי שנפרט בהמשך .

2. טרנספורמציה ואוגמננטציה

נרצה קודם לזכור איחידות בתמונות (שיהיו בגודל קבוע, ובפורמט של tensor). נסביר את היתרונות בהמשך . ובנוסף נבעו אוגמננטציה על התמונות, ככלומר יצירת שונות אקראית ביניהם (של פרספקטיבה, תאורה, חזרות וכדומה) על מנת להגדיל את הגיון בין התמונות. האוגמננטציה כוללת סיבוב, היפוך, חיתוך, שינוי בהירות או ניגודיות, הוספת רעש ועוד. טרנספורמציות אלו יעזרו למודל ללמידה על זהות אותו אובייקט בתנאים שונים ולהיות אמין יותר בחיזויים שלו.

3. בחירת מודל לאימון

נבחר מודל אותו נאמן עם הדאטאסט שלנו. המודל שנבחר צריך לעמוד על דרישות שונות על מנת שייתן לנו תוצאות טובות ומדויקות ככל שניתן. נשאף לעבוד עם CNN או עם הרחבות שלו, כפי שלמדנו בתרגול .

4. **בנייה המודל**

לאחר בחירת המודל, נחקר על הארכיטקטורה שלו ונמשח את המחלקות הדירושות. נמשח בנוסף את הפונקצייה הראשית שאחראית על אתוחול המחלקות וקריאה לפונקציית האימון.

5. **טיפול בהתאמת יתר**

נתמודד עם מצב של התאמת יתר בעזרת השיטות השונות של מדנו. צעד זה הוא חשוב כדי להבטיח שהמודל יצליח למדוד בפורה טובה בעזרת *training set*, *test set* שוגם הביצועים שלו על ה-*test set* יהיו גבוהים. דוגמאות לדרכי התמודדות עם התאמת יתר הן נורמליזציה, שכבות *Batch Norm*, *Dropout* ו-*early stopping*.

6. **בחירה הייפר פרמטרים**

נבחר את ההיפר פרמטרים עבור המודל שבחרנו בעזרת *validation set*. לבחירה זו יש חשיבות מכרעת בתוצאות המודל, וכן בחירת ההיפר פרמטרים תיושה בקפדיות, תוקן מחקר באינטראנטי, ניסוי וטעיה, או אלגוריתמים ייעודיים כדוגמת *Grid Search*. דוגמאות להיפר פרמטרים הן פונקציית *loss*, *activation*, *learning rate* ו-*regularization*.

7. **ניתוח התוצאות**

נבחן את דיוק המודל הסופי בעזרת אוסף *test*. נזהה האם התקבל מצב של *overfitting* ונכתוב את המסקנות הסופיות.

שייה לנו בהצלחה :)



שלב ראשוני – איסוף המידע והבנתו

איסוף המידע הוא שלב קריטי בכל פרויקט של למידת מכונה. ההצלחה של המודל תלויה באיכות, בכמות ובשונות של המידע בו משתמשים בשלב האימון. באימון נדרש Alfpi תמונות כדי להגיע למודל בעל דיוק גבוה.

במקרה שלנו, נרצה דאטאסט המכיל תמונות בהם מופיעים גברים ותמונות בהם מופיעות נשים. בכל קטגוריה יש לפחות רוחב ככל הנitin במאפייני התמונה (מאפיינים שונים, בזווית שונה, תוארה שונה, וכדומה). זאת על מנת להבטיח שנות גובהה במדוע שמקבל המודל בשלב האימון כדי לקבל חיזוי מדויק יותר בעבר כל תמונה שהגיעו למודל.

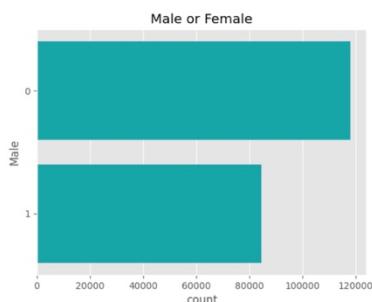
את מאגר המידע ל沉נו מאתר [kaggle](#), המציע מגוון רחב מאוד של מאגרי מידע לשימוש ציבורי.

קישור למסץ הנתונים שבחרנו [CelebFaces Attributes \(CelebA\) Dataset](#)

image_id	Arched_Eyebrows	Male	Bags_Under_Eyes	Bald
000001.jpg	1	1	-1	-1
000002.jpg	-1	-1	1	-1
000003.jpg	-1	-1	-1	-1
000004.jpg	-1	1	-1	-1
000005.jpg	1	1	-1	-1
000006.jpg	1	1	-1	-1
000007.jpg	-1	1	1	-1
000008.jpg	1	-1	1	-1
000009.jpg	1	1	-1	-1
000010.jpg	-1	1	-1	-1

הדאטאסט מורכב מתיקייה המכילה 202,599 תמונות של אנשים. וקובץ CSV המכיל מאפיינים על כל תמונה (בפורמט של "כן" או "לא", 1 או 0). כלומר, לכל תמונה מתאימה רשומה אחת בקובץ CSV, כאשר כל רשומה מתארת את התמונה לפי מאפיינים שונים. בין המאפיינים, ניתן למצוא: האם האדם קירח, האם האדם מחיך, האם האדם צער, האם האדם זכר וכדומה.

אנו בחרנו כאמור שהפרויקט שלנו יעסוק בזיהוי זכר או נקבה. וכך נשתמש במאפיין `Male` שנמצא בקובץ.



בשלב הראשון, נבדוק כמה תמונות יש לנו של גברים וכמה של נשים. צעד זה יעזור לנו לקבוע האם הדאטאסט שלנו מאוזן – קלומר מסטר התמונות של גברים ומספר התמונות של נשים זהה. נשתמש בסקריפט פיתון על מנת לספר כמותות אלו.

ניתן לראות על פי הגרף כי הדאטאסט אינו מאוזן לחלוטין. וכך, לאור הסיבות שהציגו במבוא לפרויקט, נרצה לאוזן אותו. בשל העובדה שגם כמות התמונות של גברים (הקטגוריה עם פחות תמונות, לפי הגרף) הוא כה רב – מעל 80,000 תמונות, נבחר לאוזן את הדאטאסט באופן נאיבי על ידי בחירת 80,000 תמונות אקראיות של

גברים וכמוות זהה של תמונות אקריאות של נשים. גם זאת נבע על ידי סקריפט בפייתו, שיצור לנו 2 תיקיות **Male** ו-**Female**, שיבילו כל אחת 80,000 תמונות.

כעת, נרצה לחלק את הדאטאסט ל-3 תיקיות: **training**, **validation** ו-**testing**. בחרנו לחלק את הדאטאסט ביחס של 80% עבור **training**, 10% עבור **validation** ו-20% עבור **testing**. בחרנו דוקא ביחס זה משום שלמדנו בתרגול כי הוא הפופולרי והאפקטיבי. באופן כללי חשוב שחלק גדול מהדאטאסט יהיה **training** כדי שהאימון יוכל להתבצע בצורה טובה, אך גם חשוב שחלק לא מבוטל יהיה מוקדש עבור **testing** על מנת שנוכל להעריך בצורה טובה את תוצאות האימון.



ביצעוו זאת בעזרת סקריפט בפייתו שמעריבב את הדאטאסט ברנדומליות ומחילק אותו בין שלושת התיקיות.

שלב שני – טרנספורמציה ואוגמנטציה

כעת, נעבד עם ספריית `pytorch`, שתאפשר לנו ליצור מימוש משלנו לdataset ע"י ירושא מהמחלקה `Dataset` מספרית `torch.utils.data.Dataset` ולמש פונקציות בסיסיות אשר הכרחיות לתפקיד של הדאטאסט והאינטגרציה שלו עם שאר המודולים מאוחר יותר בשלב האימון.

נקרא למחלקה `GenderDataset`, המאותחלת באופן הבא :

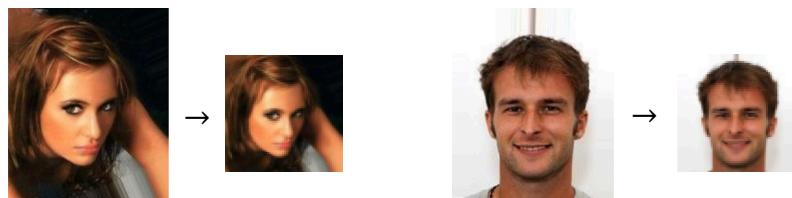
```
def __init__(self, root_dir, csv_path, transform=None, train=False, val=False, test=False):
```

כלומר, היא מקבלת את הנתיב של הדאטאסט במחשב, את הנתיב של קובץ CSV, את הטרנספורמציה שנרצה להפעיל על התמונות, ואת סוג הדאטאסט (validation `test`, `train` או `validation`). הבניי יוצר רשימה של נתיבי התמונות ששייכות לאותו דאטאסט, יחד עם התיאוג (1 עבור "גבר", 0 עבור "אישה") לכל תמונה.

הטרנספורמציה שבחרנו להשתמש בה היא

```
# Define transformations for image resizing
transform = transforms.Compose([
    transforms.Resize(64),           # Resize the image to a consistent size
    transforms.CenterCrop(64),        # Take only the 128x128 pixels in the center
    transforms.ToTensor(),           # Convert the image to a PyTorch tensor
])
```

כלומר, נשנה את הגודל של כל תמונה להיות קבוע ושווה ל 64×64 פיקסלים. חשוב שהטרנספורמציה תשמור על המאפיינים החשובים בתמונה (שהפנים לא יתהפכו, שיראו את השיער וכדומה), ולכן נדגום מספר תמונות ונראה האם הטרנספורמציה שבחרנו אכן טובה.



בנוסף, נעביר את הפורמט של כל תמונה מ`PILImage` לפורמט של `tensor`, מבנה הנתונים שאיתו עובד. באופן כללי, טנзорים הם מערכיים רב-ממדיים שמייצגים את המידע שМОזן לרשות אותה אנו בונים. היתרון המרכזי של הטנзорים הוא שניתן להעביר אותם לGPU, מה שМОוביל ל垦יצור משמעותי בזמן האימון.

מלבד הטרנספורמציה, ציינו במבוא את החשיבות של השונות בין התמונות, המתקבלת בעזרת אוגמנטציה – אותה בחרנו למש בתוך הרשת עצמה, בשלב ה-`pre-processing` של התמונות בתוך הרשת (לאחר בדיקה ראיינו כי גישה זו מניבה ביצועים מהירים יותר של תהליכי האימון).

פונקציית האוגמנטציה שבחרנו להשתמש בה היא

```
self.augment = torch.nn.Sequential(  
    transforms.RandomPerspective(distortion_scale=0.2, p=0.7),  
    transforms.RandomHorizontalFlip(p=0.3),  
    transforms.RandomRotation(degrees=(0, 180)),  
    transforms.GaussianBlur(kernel_size=3, sigma=(0.1, 0.5)),  
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])  
)
```

סביר על חלקו האוגמנטציה :

Random Perspective □ - משנה את הפרספקטיב של התמונה, שינוי המדמיה אפקט תלת ממד על

ידי עיות פינות התמונה. הparameter `distortion_scale` שולט בכמות העיות, ו-`k` הוא ההסתברות להפעיל טרנספורמציה זו על תמונה כלט.

Random Horizontal Flip □ - הופך את תמונה הקלט באופן אופקי וההסתברות לטרנספורמציה זו

הוא `k`. התוצאה של טרנספורמציה זו היא בעצם תמונה מראה של התמונה המקורית.

Random Rotation □ - מסובב את תמונה הקלט בזווית אקראית בטוחה המעלות שצוין במשתנה

. במקורה שלנו בין 0 ל-180 מעלות. `degrees`

Gaussian Blur □ - מסנן טשטוש גaus אקראי על תמונה הקלט, עם גודל `kernel` של 3 וסטיית תקן

אקראית בטוחה הסיגמא. מטרת טרנספורמציה זו היא לדמות מצב אמיתי בו המודל נדרש לספק חיזוי בעבור תמונה מטושטשת כתוצאה מצלמה באיכות נמוכה או חסר יציבות של הצלם.

Normalize □ - מNORMALIZ את ערכי הפיקסלים של תמונה קלט למומצע ולסטיית תקן שצוינו. בלמידה

מכונה, נורמליזציה משמשת לעתים קרובות לנורמל (standardize) של ערכי הפיקסלים של כל תמונות הקלט, כך שהמודל יוכל ללמידה מתקינות ודפוסים העקבאים בכל התמונות.

חשוב לציין כיצד בחרנו את ערכי הparamטרים של פונקציית הטרנספורמציה. לאחר שעברנו על מספר פרויקטים אחרים שנעודו ליזהו אובייקטים בתמונות וקריאת הקוד שלהם, ראיינו כי ערכים אלו חוזרים על עצמם. בנוסף, קראנו מספר מאמרם בנושא וגם הם המליצו על ערכים אלו כבסיס טוב לאוגמנטציה בפרויקטים שנעשה בהם עיבוד תמונות. הם ציינו גם שערכים אלו יכולים להשנות בין פרויקט לפרויקט ושמומלץ לשחק איתם, אך לאחר אימון המודל ראיינו כי ערכים אלו הניבו תוצאות טובות והחלטנו להשתמש בהם.

הנתונים שלנו כמעט מוכנים לנמרוי לאימון, מימשו את `dataset` שיכיל אותן וביצעו עלייהן את הטרנספורמציות הדורשות כדי ליצור פורמט אחד של התמונות לקריאת תהליכי האימון. בנוסף, מימשו פונקציית אוגמנטציה שתיצור לנו שונות בין התמונות, ותופעל בתהליך האימון עצמו. חסר לנו רק דבר אחד, וזה טעינת התמונות לזכרו בעת ריצת המודל – התבצע עזרה מודול ייעודי הנקרא `DataLoader`.
שנាទחן אותו כך :

```
self.train_loader = DataLoader(self.train_dataset, batch_size=batch_size,  
                               shuffle=True, num_workers=8, persistent_workers=True,  
                               pin_memory=False, drop_last=True)
```

סביר על הפרמטרים השונים :

– **train_dataset** □ הדאטאסט ממנו נרצה לטעון את התמונות.

– **Batch Size** □ – באופן כללי, batch הוא תת-קבוצה של נתונים האימון שימושים באיטרציה אחת בשלב האופטימיזציה של המודל (חלק מתהליכי האימון). הרעיון הוא לחלק את נתונים האימון לקבוצות קטנות יותר, כך שאלגוריתם האופטימיזציה יוכל לעבוד עם תת-קבוצה של הנתונים בכל פעם. אימון בצורה כזו יעיל יותר, מכיוון שבכל רגע טעונים מספר מסויים בלבד של נתונים לזכור, ומאפשר לבצע עדכונים לפרמטרים של המודל בתדריות גבוהה יותר.

– **Shuffle** □ – פרמטר שמשמעותו היא שנתוני האימון יוערבו באופן אקראי לפני כל epoch בשלב האימון. ערבוב זה נדרש למנוע מהמודל לשנן את סדר נתונים האימון ולגרום למצב של overfitting.

– **Num Workers** □ – מגדיר את מספר processes (תהליכי עבודה) בהם השתמש לטיענת הנתונים. הגדרת ערך זה לגודל מ0 מאפשרת לטעון את הנתונים במקביל, מה שיכל לשפר את מהירות תהליכי האימון.

– **Persistent Workers** □ – שומר על processes שנפתחים בתהליכי האימון "חיבם" בין איטרציות, ככלمر, כאשר מגיע חדש, DataLoader י יכול לשלוח אותו לאחד מתהליכי הקיימים, במקום לפתוח ולסגור תהליך בעבר כל batch.

– **Drop Last** □ – הפרמטר מוגדר להיות True, מה ש אומר שהאחרון batch יישטף אם הוא קטן מה-size batch. דבר זה נעשה כדי להבטיח שלכל batches יהיה אותו גודל, מה שיכל לפשט את תהליכי האימון.

– **Pin Memory** □ – עוזר ל-DataLoader להעתיק טזוריים לתוך CUDA Memory (בשימוש בעיקר בהקשר של GPU של חברת NVIDIA) לפני החזרתם. דבר זה יכול לזרז את תהליכי העברת המידע מהזיכרון של GPU.

לאחר שהסבירנו על הcnt הנתונים, והדרכם בה אנו מיבאים אותם לקוד, מבצעים עליהם את הטרנספורמציות הנדרשות לתהליכי אימון איקוטי, וטיענותם לקוד ולתהליכי האימון, נתחיל לדבר על המודל עצמו.

שלב שלישי – בחירת מודל לאימון

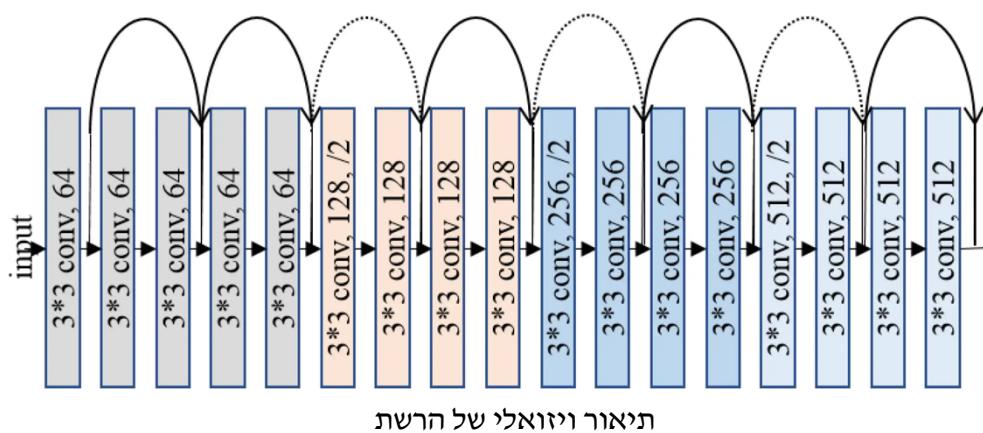
המודל אותו בחרנו למש בפרויקט הוא מודל בארכיטקטורת ResNet, או Residual Network. ארכיטקטורה זו הינה הרחבה של CNN הקלאסי שלמדנו בתרגול.

הבעיה של ResNet בא לפתור היא בעיה הנקרהת vanishing gradient, שמתרכשת ככל שהעומק של הרשות גדל, ומוסיפים עוד ווד שכבות לרשות - הגרדיינטם של פונקציית ההפסד שמכפלים בין השכבות בחולול לאחר עולמים להפוך לקטנים באופן אקספוננציאלי, ובאופן אפקטיבי "להיעלם". מצב זה הופך את עדכון המשקלים בשכבות האלה לבטתי אפשרי, ועלול להוביל את הרשות לקשיי בלימוד של תכונות חשובות, מה שיביא לbijouteries גורעים.

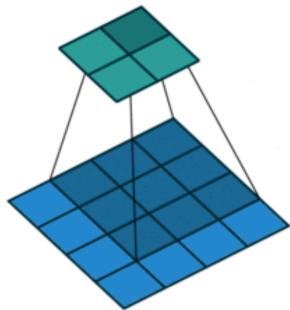
כדי להתמודד עם בעיה זו, הארכיטקטורה של ResNet משתמשת בskip connections כדי לאפשר לגרדיינטם לעבור יישורות מהקלט לפלט בכל בлок בראש, תוך שהם מدلגים על השכבות באמצעות הבלוק. הקפיצות הללו מאפשרות להתמודד עם בעיית הרשות vanishing gradient, בכך שהן דואגות שערבי הגרדיינטם לא "ייעלם" גם כאשר ישן שכבות רבות, ומאפשרות לפתח רשות עמוקה מאוד בלי לפגוע בbijouteries הרשות ואף להועיל להם.

אררכיטקטורה זו הפכה את ה-ResNets להיות פתרון פופולרי ויעיל מאוד במשימות של זיהוי אובייקטים בתמונות, סיווג של תמונות ועוד.

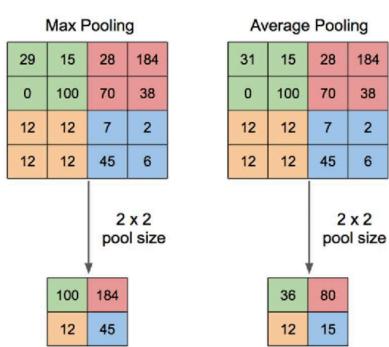
בפרויקט זה, נבחר להשתמש ב-ResNet18 (מיימוש של ResNet עם 18 שכבות).



על מנת למש את ארכיטקטורה זו, ראשית נסביר עליה ועל השכבות והבלוקים המרכזיים אותה.



שכבות קונבולוציה – שכבות אלו מהוות כלי החישוב המרכזי ברשתות CNN וכן גם ב-ResNets. השכבה מקבלת טנזור המייצג את התמונה, בגודל (גודל הפלט)(batch) × (מספר ערוצי הפלט) × (גובה הפלט) × (רוחב הפלט). לאחר שהפלט מסיים לעבור בשכבה מתקבל טנзор המייצג תמונה מופשטת יותר הנקראת feature map. התמונה המופשטת מתקבלת בעזרת חישוב dot product של הפלט (נקרא גם קרנל) על כל פוזיציה בתמונה המקורית.



שכבות איגום מקסימלי – שכבות אלו מקבלות מטריצה ומחולות אותה לאזורים בגודל קבוע (למשל 2×2) ומכל אזור נלקח הערך המקסימלי בו. טכנית זו מאפשרת לנו להקטין את הגודל של הפלט ובכך להקל על המודל מבחינה חישובית, תוך שמירה על הערכים והתכונות החשובות שבלט.

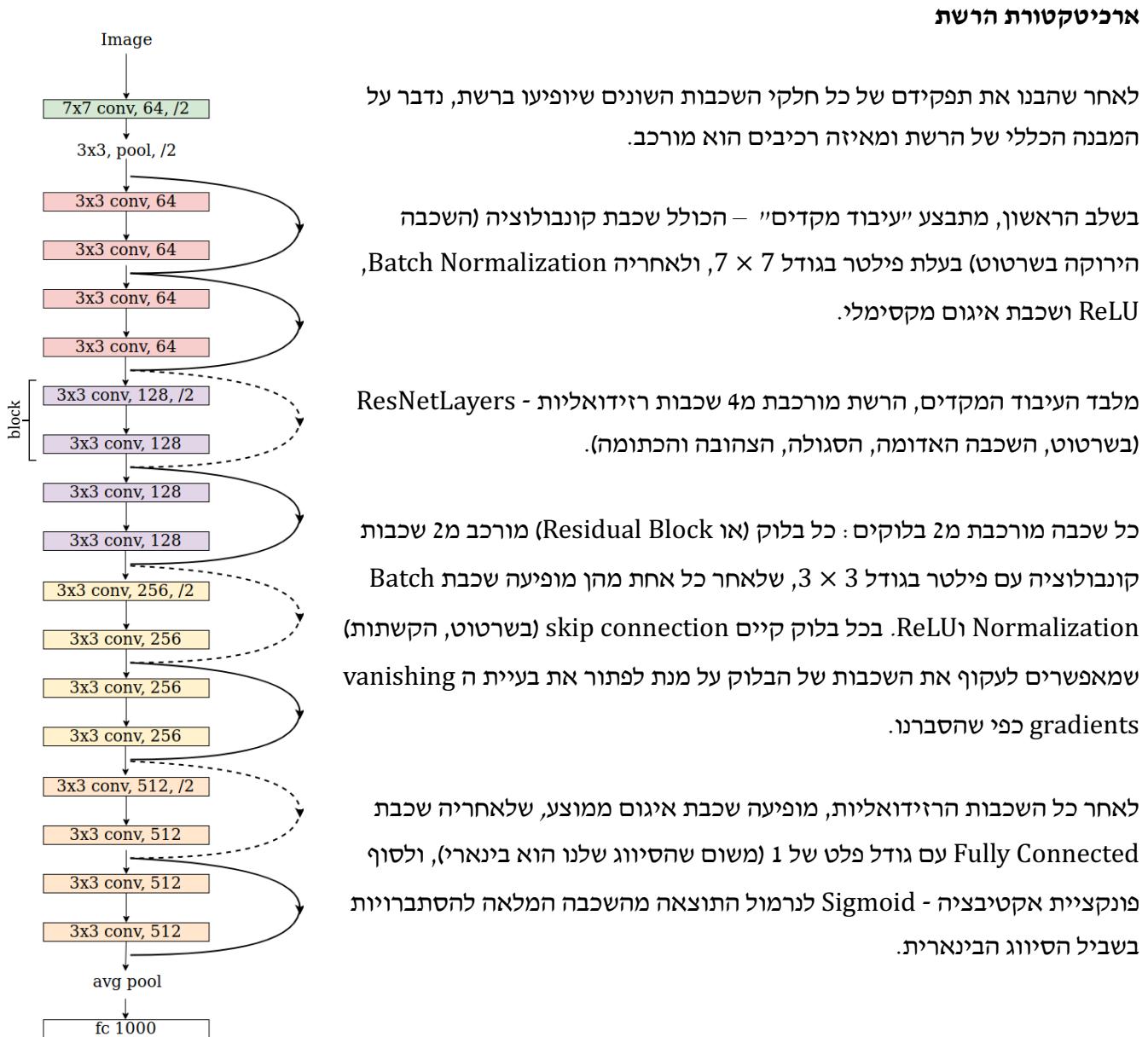
שכבות איגום ממוצע – מבחינת ריעוניות, היא עובדת כמו האיגום המקסימלי – מחלקת את מטריצת הפלט לאזורים ומכל אזור מחשבת ולוקחת את הממוצע (בניגוד ללקיחת הערך המרבי באיגום המקסימלי).

שכבות Bath Normalization – מטרתה של שכבה זו היא ליעיל ולשפר את האימון של המודל ובכך להפחית את מספר epochים הנדרשים בשלב האימון, על ידי נרמול הפלט של השכבה שקדמה לה. שכבות אלו מופיעות בכך כל אחדי שכבות קונבולוציה ולפניהם פונקציית האקטיבציה ומטרתן לנרמל את הפלט שיוצאה מהקונבולוציה ונכנס לפונקציית האקטיבציה. נפרט עוד על שכבות אלו בשלב החמיישי של הפרויקט.

שכבות Fully Connected – שכבות אלו ממוקמות בדרך כלל כשלב האخرונה לפני שכבה הפלט. כל ניירון בשכבה זו מחובר באופן מלא, כלומר מקבל קליטים מכל ניירון בשכבה הקודמת, ובמצע קומבינציה ליניארית שלהם על ידי משקלם ומקדם bias, ומפעיל את פונקציית האקטיבציה (עליה נפרט מיד) כדי לקבל את הפלט.

ReLU, Sigmoid – פונקציות אקטיבציה פופולריות ברשתות נוירונים. $f(x) := \max\{0, x\}$ מוגדרת להיות $f(x) = 0$ והיתרונות הבולטים שלה הם יעילות חישובית, קלות המימוש ועזרה במניעת vanishing gradient שעלול לקרות כאשר משתמשים בפונקציות אקטיבציה אחרות כגון sigmoid או tanh. השתמשנו בפונקציה זו כפונקציית אקטיבציה בין שכבות הרשת עצמן, על מנת לתת לרשת למדוד באופן לא ליניארי.

Sigmoid לעומת זאת מוגדרת להיות $\sigma(x) := \frac{1}{1+e^{-x}}$. היא בעצם ממפה כל קלט לערך בין 0 ל-1, שאליו ניתן להתייחס כהסתברות. פונקציה זו שימושית במיוחד לביעות סיוג ביןארית, כמו במקרה שלנו, מכיוון שהיא מספקת הסתברות שקלט מסוים שייך לאחת משתי אפשרויות ולכן השתמשנו בה כפונקציית אקטיבציה בשכבה الأخيرة (השתמשנו בה רק בשכבה الأخيرة ולא בתוך הרשת כיון שהיא עלולה לגרום לביעית (vanishing gradient).



שלב רביעי – בניית המודל

כעת, נציג את המימוש של הרשות ונסביר אותו.

מימוש הבלוק

```
class Block(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1,
                           dropout_prob=0.0):
        super(Block, self).__init__()

        # down-sample layer for the residual connection, if stride > 1
        if stride > 1:
            self.downsampling = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1,
                         stride=stride),
                nn.BatchNorm2d(out_channels)
            )
        else:
            self.downsampling = None

        # two convolution layers with batch normalization and ReLU activation
        self.conv1 = nn.Conv2d(in_channels=in_channels,
                             out_channels=out_channels,
                             kernel_size=3, stride=stride,
                             padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)

        self.conv2 = nn.Conv2d(out_channels, out_channels,
                             kernel_size=3, stride=1, padding=1)
        self.bn2 = nn.BatchNorm2d(out_channels)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=dropout_prob)
```

ראשית, נעיר שהערך `stride` זה גודל הקפיצות שהפיטר עושה בשכבות הקונволוציה (ללא קשר לגודל הפיטר). אם `stride=1` אז השכבה אינה משנה את גודל הטנזור שהוא מקבל. אחרת, הטנזור שנקלט יהיה בגודל שונה ולא יוכל להווסף אותו לפיטר (כפי שנראה מיד בפונקציית `forward`). לכן, במצב זה יש צורך בdownsampling שמטרמתה להתאים את הגודלים בין גודל הקלט של השכבה לגודל הטנзор הסופי.

מלבד זאת, נגדיר את מרכיבי הבלוק עליהם דיברנו - שכבות קונволוציה, שכבות Batch Norm ואת פונקציית ReLU בה משתמש אקטיבציה פונקציית Aktevtsia בתוך הבלוקים עצמם.

בשלב זה, הגדרנו את המשתנים שיהיו לכל בלוק, כתע נראה מה התהליך שעובר הקלט בתוך הבלוק בעת ביצוע החישוב, ככלمر בקריאה לפונקציה `forward`.

בכל בLOC, הקלט (x) עובר תחילה בשכבה קונבולוציה (conv1) של אחריה שכבה Batch Norm (bn1) ואז. ומשם פעם נוספת הוא עובר בשכבה קונבולוציה (conv2) של אחריה שכבה Batch Norm (bn2).ReLU. ולאחר מכן, נוסיף את הערך המקורי של הקלט למתוצאה שהתקבלה (תיק שימוש down sampling להתקמת הגדים במידת הצורך).

לבסוף, מפעילים את פונקציית האקטיבציה ReLU פעם נוספת.

```
# forward pass of the block
def forward(self, x):
    # Save the input as the identity tensor for the residual connection
    identity = x

    # Perform the first convolutional layer operation
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)

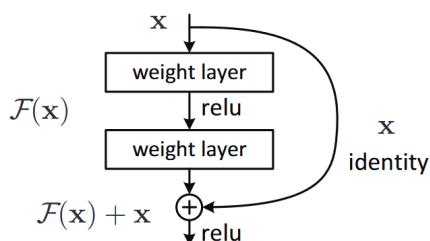
    # Perform the second convolutional layer operation
    x = self.conv2(x)
    x = self.bn2(x)

    # If downsample layer is defined, apply it to the identity
    # tensor for the residual connection layer
    if self.downsampling:
        identity = self.downsampling(identity)

    # Add the identity tensor to the output of the second convolutional
    x += identity

    # Apply the ReLU activation function to the sum of the
    # convolutional layers and the identity tensor
    x = self.relu(x)

    # Return the output of the ResidualBlock
    return x
```



נשים לב כיצד ה connection skip מומוש בפונקציה זו, בדומה להמחשה:

מימוש השכבה הרצידואלית

```
# define a ResNet layer
class ResNetLayer(nn.Module):
    def __init__(self, in_channels, out_channels, stride, dropout_rate):
        super(ResNetLayer, self).__init__()
        layers = [
            Block(in_channels, out_channels, stride, dropout_prob=dropout_rate),
            Block(out_channels, out_channels, stride, dropout_prob=dropout_rate)
        ]
        # create a sequential container of the two blocks
        self.layers = nn.Sequential(*layers)

    def forward(self, x):
        # pass input through the two blocks of the sequential container
        return self.layers(x)
```

מימוש זה הינו די פשוט, כאמור כל שכבה מורכבת משני בלוקים (ניתן לראות זאת בהגדרת המשתנה `layers`). אנו יוצרים אובייקט בשם `layers` שהוא `property` של המחלקה, שמטרתו המרכזית היא לבצע את הפעולות הכתובות בתוכו אחת אחרי השניה. פונקציית `forward` מעבירה את הקלט לאובייקט `layers`, שיגרום לביצוע הפעולות המוגדרות בבלוקים, בסדר המוגדר של הבלוקים.

ולבסוף, מימוש הרשות כולה ResNet18

```
# define the ResNet18 architecture
class ResNet18(nn.Module):
    def __init__(self, dropout_rate):
        super(ResNet18, self).__init__()
        # data augmentation layer
        self.augment = torch.nn.Sequential(
            T.RandomPerspective(distortion_scale=0.2, p=0.7),
            T.RandomHorizontalFlip(p=0.3),
            T.RandomRotation(degrees=(0, 180)),
            T.GaussianBlur(kernel_size=3, sigma=(0.1, 0.5)),
            T.Normalize(mean=[0.485, 0.456, 0.406],
                        std=[0.229, 0.224, 0.225])
        )
        # initial convolution layer, batch normalization,
        #ReLU activation and max pooling
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=64,
                             kernel_size=7, stride=2, padding=3)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU()
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.in_channels = 64
        # four ResNet layers with different number of blocks
        self.layer1 = ResNetLayer(64, 64, stride=1, dropout_rate=dropout_rate)
        self.layer2 = ResNetLayer(64, 128, stride=2, dropout_rate=dropout_rate)
        self.layer3 = ResNetLayer(128, 256, stride=2, dropout_rate=dropout_rate)
        self.layer4 = ResNetLayer(256, 512, stride=2, dropout_rate=dropout_rate)
        # global average pooling and fully connected layer
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc1 = nn.Linear(512, 1)
        # Sigmoid function for normalizing output to probabilities
        self.sigmoid = nn.Sigmoid()
```

ניתן לראות במימוש את פונקציית האוגמנטציה עליה דיברנו בשלב השני, את העיבוד המקדים, ארבע השכבות הרזידואליות, לבסוף את שכבת איגום ממוצע שלאחוריה Fully Connected יחד עם פונקציית אקטיבציה .Sigmoid

במימוש הרשות יש בנוסף פונקציית forward הממומשת באופן הבא

```
def forward(self, x):
    # Apply data augmentation to the input
    x = self.augment(x)
    # Apply the first convolution layer
    x = self.conv1(x)
    # Apply batch normalization to the output of the first convolution layer
    x = self.bn1(x)
    # Apply the ReLU activation function
    x = self.relu(x)
    # Apply max pooling to the output of the ReLU activation function
    x = self.maxpool(x)

    # Apply the four ResNet layers in sequence
    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)
    # Apply adaptive average pooling to the output of the final ResNet layer
    x = self.avgpool(x)
    # Flatten the output of the adaptive average pooling layer
    x = x.reshape(x.shape[0], -1)
    # Apply the fully connected layer with a single output unit
    x = self.fc1(x).squeeze(1)
    # Apply the sigmoid activation
    return self.sigmoid(x)
```

המימוש שלה אכן היה צפוי, מעבירים את הקלט לאורך הרשות (ומשטחים את הפלט לווקטור לפני שכבת (Fully Connected) ולבסוף מחזירים אותו.

סיימנו את מימוש המודל. נעבור למימוש האימון עצמו.

בחרנו למש את המימון בעזרת מחלקה Trainer שמאוחלת באופן הבא

```
def __init__(self, train_dataset, val_dataset):  
    self.train_dataset = train_dataset  
    self.val_dataset = val_dataset
```

למחלקה יש מетодה define_model שחתימתה:

```
def define_model(self, dropout_rate=0.3, learning_rate=0.01,  
                 weight_decay=1e-5, batch_size=64, patience=6):
```

המטרה של מетодה זו היא לאותחל את המודל לפי היפר-פרמטרים שנבחר. נפרט על בחירת היפר-פרמטרים המתאימים בשלב השישי.

METHOD נספהה של המחלקה Trainer היא כמפורט, train, שחתימתה:

```
def train(self, num_epochs=40):  
    היא מקבלת את מספר epochים אותו נרצה להריץ, ותאמן את המודל – תזק שימרת העתק של המודל  
    לאחר כל איפוק על הדיסק במחשב.
```

לאחר כל כתיבת המחלקות השונות. נאמן את המודל באופן הבא:

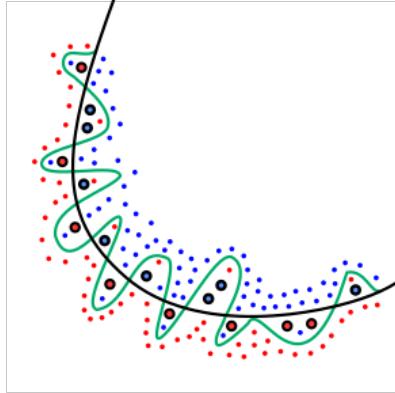
```
transform = t.Compose([  
    t.Resize(64), # Resize the image to a consistent size  
    t.CenterCrop(64), # Take only the 64x64 pixels in the center  
    t.ToTensor(), # Convert the image to a PyTorch tensor  
)  
  
train_dataset = GenderDataset(root_dir='./dataset/train',  
                               csv_path='./dataset/list_attr.csv',  
                               transform=transform,  
                               train=True)  
  
validation_dataset = GenderDataset(root_dir='./dataset/validation',  
                                    csv_path='./dataset/list_attr.csv',  
                                    transform=transform,  
                                    train=True)  
  
trainer = Trainer(train_dataset, validation_dataset)  
trainer.define_model(dropout_rate=0.3, learning_rate=0.01, weight_decay=1e-5,  
                     batch_size=64)  
trainer.train()
```

כלומר, נגידר את ה transform ונאתחל את הדאטאסט של האימון ושל הווילידציה. נאותחל את ה Trainer ונגדיר את המודל לפי היפר-פרמטרים שנבחר. ולבסוף נאמן את המודל.

נזכיר כי לפניו שנאמן את המודל באופן הזה, עליו לחפש היפר פרמטרים טובים, נפרט על כך בשלב השישי.

שלב חמישי – טיפול בהתאמת יתר

התאמת יתר היא בעיה שבה המודל מותאם יתר על המידה לאוסף הנתונים עליו אומן (training set) ועל כן מצליח פחות בבייצוע תחזיות על דוגמאות חדשות.



הקו הירוק מייצג מודל עם התאמת יתר, והקו השחור מייצג מודל מוסדר. הקו הירוק מתאים יותר לנתחוני האימון, אך הוא תלוי בהם יותר מדי ולכן הוא צפוי להיות בעל שגיאה גדולה יותר בסוג נתונים חדשים מאשר המודל השחור.

התאמת יתר (או Overfitting), מתרחשת לרוב כאשר אנו מאפשרים למודל להקטין את loss ללא הגבלה. במצב כזה, המודל יתאים את עצמו גם לשגיאות ולרעש שבנתוני האימון, ובכך אינו יצליח למדוד בצורה טובה ולהקליל עבור תמצאות חדשות.

בפרויקט, נרצה להקדים תרופה למכה, ונמשח את הפתרונות שלמדנו בקורס למניעת התאמת יתר. נעבור על כל שיטה ונסביר כיצד מימושו אותה בפרויקט.

1. נרמול (Normalization)

נורמליזציה של הנתונים מתייחסת לטכנית המשמשת למניעת להפחת התאמת יתר, על ידי הבטחה שכל נתוני האימון הם עקביים – הפעלת טרנספורמציה על הנתונים אשר תזודא שכל המאפיינים בעלי סולמות דומים, כך שלא יהיה מאפיין כלשהו ש-”משתלט” או ”מסתיר” את המאפיינים האחרים. זה לא מונע באופן ישיר התאמת יתר, אבל זה יכול לפחות את תהליכי האימון.

אנו מימושו נורמליזציה של הנתונים בתוך שלב האוגמנטציה, בעזרת

```
t.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
```

בשורה זו, אנו מבצעים נורמליזציה לכל הפיקסלים בתמונה (שלושת המספרים של mean ושל std מתייחסים ל-`RGB` = red, green, blue). המטרה היא בעצם להביא את המוצע וסטיית התקן של ערכי הצבעים לו, בהתאמת. ערך mean מוחסר מכל פיקסל, וערך std מוחסר מכל פיקסל. בחרנו דוגמא ערכים אלו כיון שראינו שהם חוזרים על עצם במקומות רבים, וומומלץ להשתמש בהם כאשר מאמנים על דאטאסט עם תמצאות ”רגילותות” – אנשים, מבנים, חיות, וכדומה (ערכים אלו ככל הנראה קרובים לתוחלת וסטיית התקן האמיטיים של התמצאות).

Batch Normalization Layers .2

עם הסבר דומה לחסיבות של הנורמליזציה לפני העברת התמונה ברשות הנוירונים – علينا להבין כי זה לא מספיק. התמונה עוברת דרך רשות הנוירונים, שמורכבת משכבות רבות. כל שכבה מקבלת קלט את הפלט של השכבה הקודמת. אם נפעיל את הנורמליזציה רק בהתחלה, שאר השכבות של הרשות עלולות לקבל קלט לא מוגדר. המטרה של שכבות Batch Norm היא לטפל במקרה זה בדיקות: שכבות אלו הן שכבות רגילים שמצוות להגדרת הרשות בין השכבות הקיימות, מטרתן לחתות את הקלט של השכבה הקודמת, ולנורמל אותו לפני העברתו לשכבה הבאה.

אנו מימושו את שכבות ה-Batch Norm בתוך הגדרת ארכיטקטורת ResNet18

```
self.bn1 = nn.BatchNorm2d(64)
: ReLU
x = self.bn1(x)
```

ובפונקציית ה-forward, לפני פונקציית האקטיבציה :

ניתן להוסיף שכבות Batch Norm נוספות. אך איננו חווינו התאמת יתר משמעותית בכך שלא היה לנו צורך בכך.

Early Stopping .3

שיטה זו, כפי שהיא מעיד, מתייחסת לעצירת האימון (ambil להמשיך להEpochים נוספים) כאשר מבחנים בכך שהמודל אינו משתפר יותר. אימון המודל למשך מספר רב מדי של epochים עלול להוביל למצב של התאמת יתר. ולכן השיטה מציעה לבדוק לאחר כל epoch האם המודל השתפר. בדיקה זאת מתבצעת על ידי חישוב loss על סט נתונים נפרד, שלא היה חלק באימון המודל (הרי ברור שלא נבחן בהתאם יתר אם נסתכל על training set בלבד) – ככלומר כאן אנו משתמשים ב-set validation. אנו מגדירים משתנה הנקרא "סבלנות", שיגיד לנו אחרי כמה epochים לא מוצלחים רצופים (בهم loss שהוחשב על ה-set validation) אינו השתפר), נזכיר כי מתחילה להיווצר התאמת יתר ונפסיק את האימון.

מיימשו את שיטה זו בפונקציית `train` של המחלקה `Trainer`:

```
best_val_loss = float('inf') # best validation loss = infinity
current_patience = 0 # Initialize the patience counter

for epoch in range(num_epochs):
    # the training logic for a single epoch. Irrelevant for this
    # explanation.

    # Evaluate on the validation dataset
    val_loss, val_accuracy, _, _, _ = self.evaluate()

    # Check if validation loss has improved
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        current_patience = 0 # Reset patience counter
    else:
        current_patience += 1 # Increment patience counter

    # Check if early stopping criteria met
    if current_patience >= self.patience:
        print(f'Early stopping after {epoch + 1} epochs with no improvement in
              validation loss.')
        break # Stop the training.
```

Regularization .4

בשלב האימון, אנו נרצה להימנע ממצב שבו משקלים קשוחות בין הנוירונים מגיעים לערבים גבוהים מדי. מצב זה הוא רע מאוד והוא גורם להתאמת יתר – המודל נהיה מסובך ו-”רגיש” לרעשים קטנים מה שגורם לביצועו להיות גרועים עבור תמונות חדשות שלא ראה. ערבי משקלים גבוה מדי יכול לגרום למצב של ”גרדייאנטים מטופצצים” בו הגרדייאנטים נהים מאוד גדולים, באופן שמקשה להתכנס למינימום בתהליך `Gradient Decsend`.

אנו בחרנו להשתמש ברגולרייזציה מסוג L2, הנקראת גם Weight Decay. הרעיון שלה היא ”להעניש” משקלים גבוהים, על ידי הוספת הביטוי $\lambda \sum_{i,j} (W_{i,j})^2$ loss שנבקש למזער. כך, המודל לא יסתפק בסולו מינימלי, אלא עליו לנקות בחשבונו גם משקלים קשוחות בראשת אינס גבוהים מדי.

אנו הוספנו את הרגולרייזציה בהגדרת המודל:

```
optimizer = torch.optim.AdamW(self.model.parameters(), self.lr,
                             weight_decay=self.weight_decay)
```

כאשר הערך `weight_decay` קבוע באיזו עוצמה נענייש ערבים גדולים (הוא היל בביטוי).

Dropout .5

שיטת זו משנה את רשת הנוירונים עצמה, לצורך טיפול בהתאם יתר. הרעיון של שיטה זו היא לבחור באקראיות נוירונים ולמחוק אותם מהרשות, יחד עם הקשרים שלהם לנוירונים בשכבות הצמודות.

שיטת זו יוצרת רשת פשוטה יותר, ומונעת את הנוירונים מלהיות תלויים אחד בשני. אנו מגדירים Dropout Rate שיקבע את הסיכוי של נוירון להימחק. Dropout מתרחש לפני כל epoch בשלב האימון של המודל.

אנו הוספנו dropout בהגדרת ארכיטקטורת ResNet18 (בבלוקים) :

```
self.dropout = nn.Dropout(p=dropout_prob)
```

כפי שprzedנו בתרגול, השיטה הטובה ביותר להתרמודד ולהימנע מההתאמת יתר היא על ידי שימוש בכלל השיטות לעיל, וכך עשינו. בדיעבד, אכן כמעט ולא חווינו התאמת יתר ולכן בחרנו שלא לנסות שילוב אחר של שיטות רגולרייזציה.



**THE BEST WAY TO
EXPLAIN OVERFITTING**

שלב שלישי – חיפוש היפר פרמטרים

ההיפר-פרמטרים הם מאפיינים ומערכות המגדירים את המודל ואת אופן הלמידה, הם אינם נלמדים בתהיליך האימון אלא נקבעים מראש. הם קובעים את ההתנהגות של המודל ויש להם חשיבות מכרעת בביצועים שלו.

דוגמאות להיפר פרמטרים העיקריים הם :

פונקציית ההפסד (loss) זהה פונקציה מתמטית שתפקידה לחשב את ההבדל בין פלט החיזוי של המודל לבין הסיווג האמתי. מטרת האלגוריתם הלמידה של המודל הוא למזער את ערכי פונקציית ההפסד, מה שוביל לחיזויים טובים ומדויקים יותר, משום שצמצום השגיאה מוביל לחיזוי קרוב יותר לסיוג האמתי. אנו בחרנו להשתמש בפונקציית השגיאה BCELoss – Binary Cross-Entropy, שהיא פונקציית הפסד שכיחה בבעיות של סיווג בinaire. בחרנו בה מכיוון שכבר מחשילה ניתן להבין שהיא מתאימה לבניית הסיווג שלנו. נוסחת הפונקציה :

$$L_{BCE} = -\frac{1}{n} \sum_{i=1}^n (Y_i \log \hat{Y}_i + (1 - Y_i) \log(1 - \hat{Y}_i))$$

כאשר Y זה הסיווג האמתי של התמונה ו \hat{Y} זה הסיווג שהמודל חזה.

פונקציית האקטיבציה זו הפונקציה שתופעל על הפלט מהשכבות. אנו בחרנו להשתמש בפונקציית האקטיבציה ReLU לאורך עמק הרשת, ובפונקציית האקטיבציה Sigmoid בשלב האחרון של הרשת. אנו בחרנו לפעול כך לאחר שראינו כי גישה זו חוזרת על עצמה במאמרים שונים. בנוסף, בחרנו על היתרונות של גישה זו (אותם ציינו בשלב השלישי של הפרויקט – בהסבר על ReLU וsigmoid).

Optimizer הוא אלגוריתם האופטימיזציה בו נעשה שימוש בעדכון המשקלים ומשתני הרשת בתהיליך האימון. אנו בחרנו להשתמש בAdam עליו המליצו במקומות רבים (וגם באחד התרגולים של הקורס).

Kernel Size הוא גודל הernal (הפילטר) בשכבות הקונволוציה. אנו בחרנו בגודל של 3, שהוא מודול פופולרי בימוש הארכיטקטורה של ResNet18. ערך זה לרוב מניב תוצאות טובות וומלץ להשתמש בו "כנקודות פתיחה". אנו קיבלו תוצאות טובות עם גודל זה ולכן בחרנו להשייר אותו.

טרנספורמציה ואוגמנטציה אלו, כפי שהסבירנו בתחילת הפרויקט, שלבים בהם אנו צריכים אחדות בין התמונות (שיהיו בגודל קבוע עבור הרשת) וגם שונות אקרואית ביניהם על מנת ליצור גיוון. אנו בחרנו את הטרנספורמציה לכוז כל תמונה כך שהממד הקטן שלה (אורך או רוחב) יהיה 64 פיקסלים, ולאחר מכן לחת את הריבוע 64×64 במרכזה התמונה. בחרנו בטרנספורמציה זו משום שראינו כי היא אינה חותכת את המאפיינים החשובים של כל תמונה (על ידי בחינה אקרואית של דוגמאות). בנוסף, בחרנו שגודל כל תמונה יהיה 64 על 64 כיוון שראינו כי המאפיינים החשובים בכל תמונה נשארים בורורים, וגם נמנעו מגודל גדול מדי של תמונה כדי להקל על המודול ועל תהליכי האימון. את האוגמנטציה (שכוללת בתוכה גם את הנורמליזציה) בחרנו למש עם ערכיהם שהמליצו עליהם כבסיס טוב.

את ההיפר פרמטרים הללו השארנו קבועים ובחרנו תוך מחקר ובדיקה במקומות אינטראקטיבים. אמם, קיימים היפר פרמטרים אשר יותר רגילים מבוחינת הביצועים שלהם בין פרויקט לפרוייקט, ולפיכך נסתמך פחות על המלצות, ונחפש עבורם ערכים אופטימליים בעצמו.

Learning Rate – אחת מההיפר פרמטרים הקritisטים ביותר באימון המודול. קובע את גודל הצעדים של **Gradient Descend** בכל איטרציה.

Batch Size – פרמטר זה קובע את כמות הדוגמאות מהדאטאסט שיישמשו בכל איטרציה של **Gradient Descend**. כמות קטנה יכולה לגרום לעדכונים "רוועשים" של משקל הרשת, אמם כמות גדולה מגדילה את הזמן שלוקח למודל להתאמן.

Regularization Strength – גודל העוצמה שבה נעניש ביחס לloss, כאשר המודול מסתבך יתר על המידה. במקרה של רגולרייזציה מסוג L2, פרמטר זה הוא `weight_decay`, שהסבירנו עליו בשלב הקודם.

Dropout Rate – הסיכוי של ניירון יחיד להימחק, על פי שהשיטה שהסבירנו בשלב הקודם.

כאמור, לא ניקח ערכים מומליצים עבור היפר פרמטרים אלו. אמם, אנו יכולים לצפות באיזה טווח של ערכים עבור כל פרמטר נצהה לתוצאות טובות. למשל, Learning Rate שהוא בטוחה בין 0.01 ל-0.001 נראתה ניתנת תוצאות לא רעות. כך גם Batch Size שהוא בין 32 ל-128.

שימוש ב-Grid Search

אנו נממש Grid Search על מנת למצוא את הערכים האופטימליים עבור כל אחד מההיפר-פרמטרים הללו. Grid Search הוא אלגוריתם די נאיבי למציאת היפר פרמטרים אופטימליים. הרעיון הוא להגדיר אילו ערכים נרצה לנסות עבור כל היפר פרמטר. לאחר מכן, נבור על כל קומבינציה אפשרית של ערכים אלו ונאמן את המודל לפיהם.

לאחר כל אימון, נעריך את ביצועי המודל בעזרת סט נתונים נפרד (שלאלקח חלק בתהליכי האימון – Validation Set), ונשмар את הקומבינציה שהניבה את הביצועים הטובים ביותר.

אנו בחרנו להעריך את ביצועי המודל עבור כל קומבינציה לפי $F1$ Score, שמודגר להיות

$$F1\ Score := \frac{2 \times Precision \times Recall}{Precision + Recall}$$

$F1$ Score גבוהה (כלומר קרובה ל1) מעיד גם על $Precision$ טוב וגם על $Recall$ טוב. ולכן הוא מהוות מודד מעולה בעזרתו נשווה בין ביצועי המודל עבור כל קומבינציה של היפר-פרמטרים.

המשך בעמוד הבא

מימוש של Grid Search

```
import os
from train import Trainer, init_datasets

# Create a function to train and evaluate a model for a given set of hyperparameters
def train_evaluate_model(learning_rate, batch_size, weight_decay, dropout_rate):
    # Train the model with the current hyperparameters
    train_dataset, validation_dataset, _ = init_datasets()
    trainer = Trainer(train_dataset, validation_dataset)
    trainer.define_model(dropout_rate=dropout_rate, learning_rate=learning_rate,
                          weight_decay=weight_decay, batch_size=batch_size)

    trainer.train(num_epochs=30)

    _, _, validation_f1, _, _ = trainer.evaluate()

    print(f"==> Current training resulted with F1={validation_f1}")
    # Return the F1 score
    return validation_f1

# Define hyperparameter ranges
learning_rates = [0.001, 0.005, 0.01]
batch_sizes = [32, 64]
weight_decays = [1e-5, 1e-4]
dropout_rates = [0.2, 0.3]

best_f1_score = 0.0
best_hyperparameters = {}

# Create the directory for saving model checkpoints if it doesn't exist
checkpoint_dir = 'trained_models'
if not os.path.exists(checkpoint_dir):
    os.makedirs(checkpoint_dir)

# Iterate over hyperparameters
for learning_rate in learning_rates:
    for batch_size in batch_sizes:
        for weight_decay in weight_decays:
            for dropout_rate in dropout_rates:
                current_f1 = train_evaluate_model(learning_rate, batch_size,
                                                weight_decay, dropout_rate)

                # Check if the current combination of hyperparameters resulted in a
                # better F1 score
                if current_f1 > best_f1_score:
                    best_f1_score = current_f1
                    best_hyperparameters = {
                        'learning_rate': learning_rate,
                        'batch_size': batch_size,
                        'weight_decay': weight_decay,
                        'dropout_rate': dropout_rate
                    }

# Print the best hyperparameters and corresponding F1 score
print("Best Hyperparameters:", best_hyperparameters)
print("Best F1 Score:", best_f1_score)
```

כפי שניתן לראות, בחרנו לנסוט 3 ערכים שונים עבור ה *Learning Rate*, 21 ערכים שונים עבור ה *Batch Size*, ו-24 קומבינציות שונות עבור ה *Dropout Rate* ו- *Weight Decay*.

ນצין כי לצערנו ניסינו של יותר ערכים היה לוקח זמן רב מדי, ולכן נאלצנו לנסוט רק את הערכים שאנו מאמינים כי סביר שיניבו את התוצאות הטובות ביותר.

במידת הצורך, נפעיל *Grid Search* נוספת "ברזולוציה גבוהה יותר" סבב הערכים שהניבו את התוצאות הטובות יותר (כלומר, אם למשל *Learning Rate* של 0.005 נותן את התוצאות הטובות ביותר, ננסה לעשות *Grid Search* חדש עם 0.004, 0.005 ו-0.006).

לסיקום, חלק מההיפר פרמטרים בחרנו תוך בדיקה ומחקר אינטנסיבי. עברנו על מאמרם וראינו ערכים מומלצים ומתי כדאי להשתמש בהם. קראנו על היתרונות של שימוש בערכים שונים ולאחר בדיקת המקרה הפרטי שלנו – לקחנו את הערכים המתאימים והרלוונטיים.

היפר פרמטרים אחרים, מצאנו תוך השוואה שביצעו בעזרת *Grid Search*. את ההשוואה ביצעו תוך בדיקה של ערכים "מבטייחים" שונים, ושילובים שלהם.

ນצין כי את הריצה של אלגוריתם *Grid Search* ביצעו באמצעות *Google Colab* שנtran לנו את האפשרות להריץ על GPU, שם החישובים יכולים להתבצע במהירות רבה יותר.

Import the dataset for training the model.

```
✓ 18s [2] from google.colab import drive
      drive.mount('/content/drive')
      Mounted at /content/drive
```

```
✓ [3] !unzip '/content/drive/MyDrive/introToAI/dataset.zip'
```

Check whether a GPU is available and can be used for computation.

```
✓ 5s [4] import torch
      torch.cuda.is_available()
      True
```

Run the Grid Search algorithm.

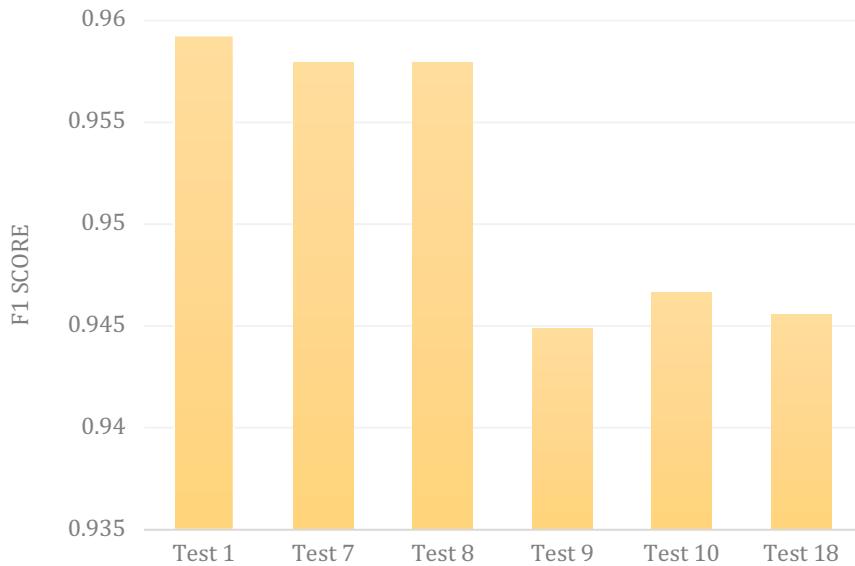
```
!python3 grid_search.py
```

לשימוש *Google Colab* יצרנו מחברת חדשה, ייבאנו את הדאטאסט מהדרייב, בדקנו כי אכן ניתן להשתמש בGPU לצורך החישובים, ולבסוף, הרצנו את האלגוריתם ועקבנו אחר התוצאות.

כמו כן, *Google Colab* מציע לנו לבחור את החומרה אליה נרץ את האלגוריתם. בחרנו בGPU T4 שכן זה הזרם חזק ביותר.

ריצת אלגוריתם *Grid Search* נמשכה בקירוב 20 שעות. כלומר קצת פחות משעה עבור כל קומבינציה (כזכור, בדקנו 24 קומבינציות).

לאחר ריצת Grid Search, קיבלנו את התוצאות הבאות.



בגרף, ניתן לראות את שלושת תוצאות Grid Search הטובות ביותר (שהתקבלו בניסיונות 1, 7-8) לצד שלושת תוצאות Grid Search הגרועות ביותר (שהתקבלו בניסיונות 9, 10 ו-18).

מה שמעניין אותנו כמובן היא התוצאה הטובה ביותר, שהתקבלה בניסוי הראשון והיא $F1\ Score = 0.9593$ מה שמעניין לנו, התקבלה עבור היפר פרמטרים הבאים:

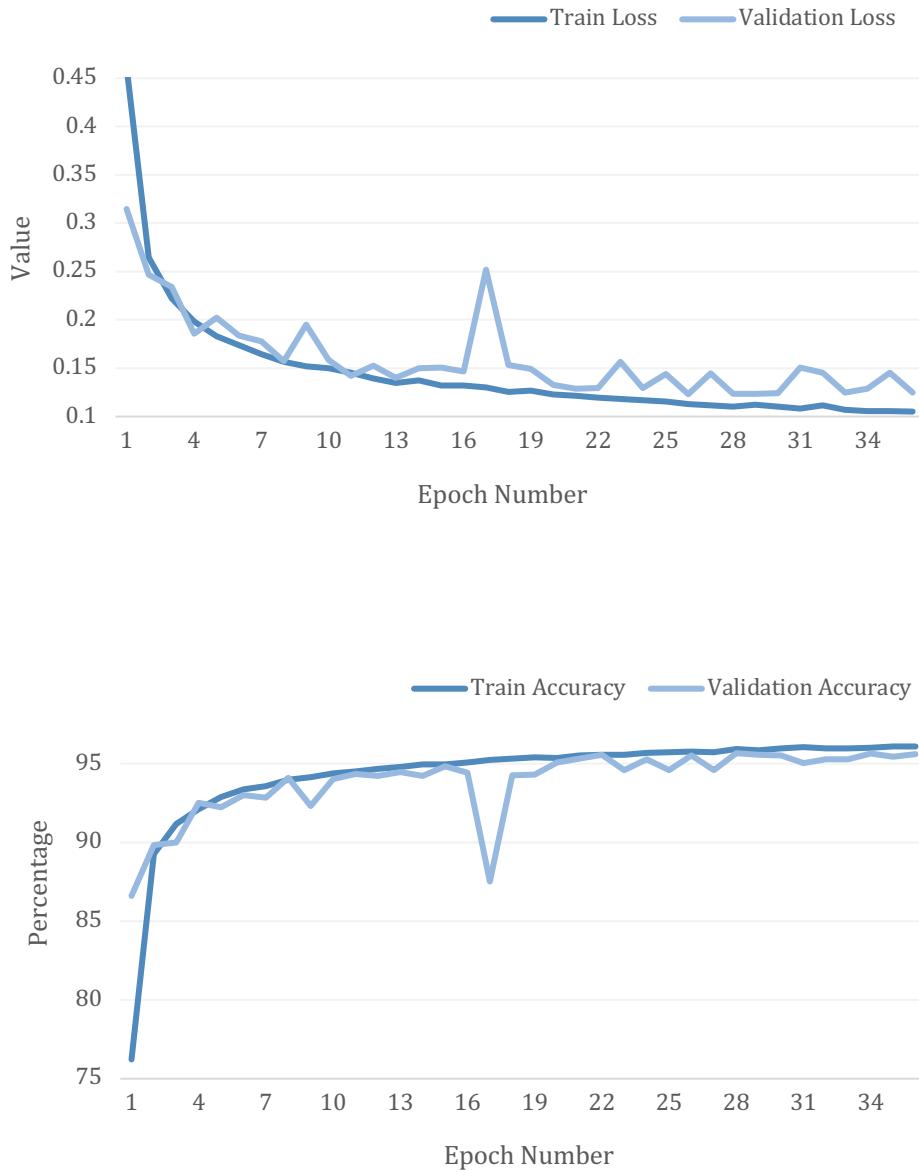
$$\begin{array}{ll} \text{Learning Rate} = 0.001 & \text{Batch Size} = 32 \\ \text{Weight Decay} = 0.00001 & \text{Dropout Rate} = 0.2 \end{array}$$

ולכן נבחר להשתמש בהם.

נאמן שנית את המודל עם היפר פרמטרים אלו, כאשר ניתן לו לróż למשך מספר רב יותר של epochים (עבור Grid Search הסתפקנו ב-30 epochים לכל יותר עבור כל ניסוי, כדי לkür את זמן הריצת האלגוריתם). על מנת להימנע ממצב של התאמת יתר, נקבע את patience להיות 10 עבור early stopping.

ריצת האימון הפסיקה לאחר 36 epochים כתוצאה מחוסר שיפור בValidation Loss.

תוצאות האימון הסופי עבור ה-Training Set וה-Validation Set



וקיבלו בנוסך F1 Score של **0.9563** עבור ה-Validation.

אמנם תוצאות אלו טובות, נסעה להמשיך לחפש היפר פרמטרים אופטימליים יותר לטובת שיפור התוצאות.

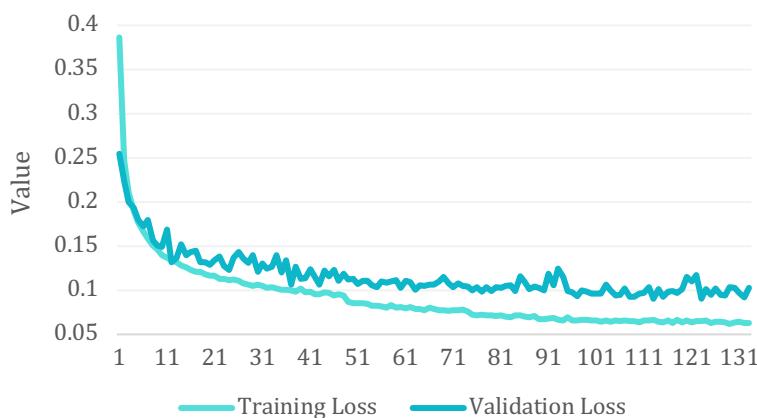
שיפור באמצעות Learning Rate Scheduling

בRICT אלגוריתם Grid Search בדקו ערכים קבועים עבור Learning Rate ומצאו כי 0.001 הוא הערך האופטימלי (מבין אלו שבדקו). נסה בעת לשנות גישה ובעור היפר פרמטר זה, לאפשר לו לשנתות באופן דינامي בזמן תהליך האימון.

קובע את גודל הצעד שニיך בכל פעם לכיוון המינימום, בעת עדכון משקولات הרשות (לפי שיטת Learning Rate Scheduling). הרעיון מאחורי Learning Rate Scheduling הוא שכאשר אנו מתקרבים יותר ויתר למינימום, נרצה להקטין את גודל הצעד שלנו על מנת להתקרב באופן מדויק ככל הניתן למינימום.

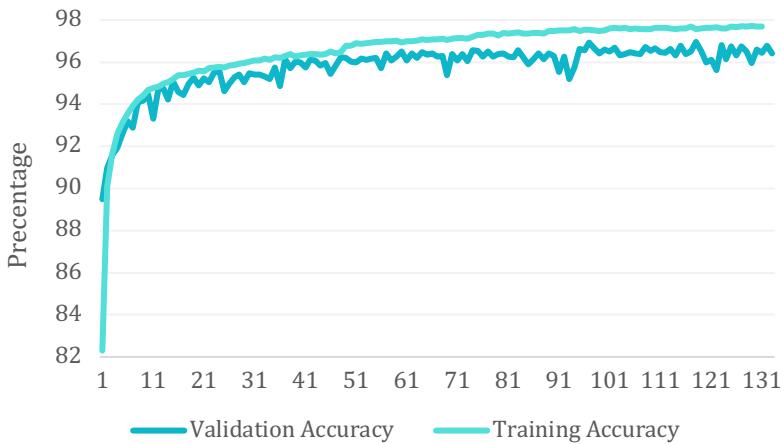
מצאו כי Learning Rate של 0.001 מביא ערבינו תוצאות טובות. גם הפעם, נסה לשימוש בערך זה, אולם כאשר אנו מתקרבים יותר ויתר למינימום, נרצה "לעוזן" את צעדינו ולהקטין את ה Learning Rate. וכך, נפאל באופן הבא: במקום לעצור את האימון לאחר 10 epoch לא מוצלחים (כחלק מ시יטת ה early stopping), נקטין את ה Learning Rate פה. ובמקרים, נקבע את epoch לעצור את האימון לאחר 20 epoch לא מוצלחים (כלומר אם גם 10 epoch לא מוצלחים, נשארו לא מוצלחים, נקבע כי עדכון ה Learning Rate לא עוזר ונסיים את תהליך האימון).

במהלך האימון, המודל הגיע לערך Validation Loss של 0.1062 epoch 37. למשך 10 epochs הבאים הוא לא הצליח לשפר ערך זה, ולכן epoch 47 הוא עדכן את ה Learning Rate מ 0.001 ל 0.0005. כמו שקייםנו, בepochs הבאים המשיך לשיפור מתחילה ל 0.1062. בסיום תהליך האימון, ה Validation Loss הגיע לערך מינימלי של 0.0915 epoch 133. והפסיק לאחר Validation Loss שלא היה שיפור ב Validation Loss. ניתן להתרשם משאר הנתונים על פי הגרף.



זהו שיפור מהניסיון הקודם, בו ה Loss Validation הצליח לרדת לערך מינימלי של 0.1231 בלבד.

כמו כן, ניתן לראות גם שיפור בדיקת (Accuracy) של המודל, על פי הגרף.



כאן, הגיעו לValidation Accuracy מקסימלי של 96.96%, בניגוד לניסיון הקודם שם הגיעו לTraining Accuracy מקסימלי של 95.632%. כמו כן, כאן הגיעו לValidation Accuracy של 97.71% ביחס לניסיון הקודם בו הגיעו ל-96.09%.

ולסיום, עם המודל החדש הגיעו לValidation F1 Score של 0.9698. זהו שיפור מאשר 0.9563 של המודל הקודם. בסך הכל, אכן ניתן להבחין בשיפור ולכון נבחר להשתמש במודל החדש עם Learning Rate Scheduling.

נציין בנוסף כי השימוש ב-Scheduling דרש בחירת היפר פרמטרים נוספים:

- הסבלנות בעדכון ה-Learning Rate, ככלומר אחרי כמה epochים לא מוצלחים נבחר להקטין את קצב הלמידה. אנחנו בחרנו ב-0.1.
 - פי כמה נקטין את קצב הלמידה בכל פעם. אנחנו בחרנו ב-0.5 (כלומר פי 2).
- ניתן לחפש בעורת Grid Search ערכיהם טובים עבור היפר פרמטרים אלו. אולם, הערכים שבחרנו להשתמש בהם (לאחר המלצות שקרנו באינטרנט) נתנו לנו תוצאות טובות ולכון בחרנו להשאיר אותם.

לסיכום, בשלב זה בחרנו היפר פרמטרים עבור המודל תוך שימוש במספרים (בדיקה באינטרנט, הרצת Grid Search, ניסוי וטעיה).

נבחר להשתמש במודל שנタン את התוצאות הכי טובות עבור Validation. זהו המודל שהתקבל לאחר epoch 118. מודל זה נתן את התוצאות

$$\text{Val Loss} = 0.0968 \quad \text{Val Accuracy} = 96.96\% \quad \text{Val F1 Score} = 0.9698$$

שלב שבייעי – ניתוח התוצאות

לאחר בחרית המודל הסופי שבו נשתמש. נוריד אותו למחשב שלנו (כזכור הרצינו את האימון בעזרת Google Colab). לאחר כל epoch, שמרנו העתק של המודל לדיסק, כפי שניתן לראות בקטע הקוד

```
# Save model version {epoch}
with open(f'trained_models/model_epoch{epoch + 1}.pkl', 'w'):
    torch.save(self.model.state_dict(), f'trained_models//model_epoch{epoch + 1}.pkl')
```

לכן, נוריד למחשב את הקובץ trained_models/model_epoch118.pkl. ניתן לעשות את זה על ידי הרצת

```
[ ] from google.colab import files
files.download('trained_models/model_epoch118.pkl')
```

במחברת.

כעת, לראשונה ניעזר בdataset Test לצורך ניתוח תוצאות האימון של המודל. במהלך בניית המודל, חשוב שלא נקבל החלטות לפי ביצועי המודל על ה-test. המטריה של נתוני המבחן היא לקבל מודד מדויק של עד כמה המודל יכול לסוג נתונים שלא ראה מעולם. אם נכלול את נתונים המבחן בשלב אימון המודל – לא נוכל לסמן באותה מידת על ניתוח התוצאות הסופי.

הקוד לניתוח התוצאות ישתמש בקובץ model_epoch118.pkl (שנשנה את שמו לmodel.pkl) שמכיל את המודל אותו אנו רוצים למדוד.

מדדיה הערצת המודל

בחרנו להעריך את תוצאות המודל לפי F1 Score, Precision, Recall, Accuracy. Accuracy – מחשב עבורנו איזה אחוז מההתמונות המודל הצליח לסוג נכון. ככל שערך זה קרוב יותר ל-100 אחוזים כך המודל הוא יותר מדויק.

Precision – מוגדרים להיות Recall-precision –

$$\text{Precision} := \frac{\text{TP}}{\text{TP} + \text{FP}} \quad \text{Recall} := \frac{\text{TP}}{\text{TP} + \text{FN}}$$

כלומר, מודד איזה חלק מההתמונות שהמודל סיווג כ- "גבר" הם אכן "גבר". וה-Recall – מודד איזה חלק מההתמונות של גברים, המודל אכן סיווג ל- "גבר".

במקרה שלנו, לא נעדי False Positive נמוך על פני False Negative. שכן אין סיבת שהמודל יעדיף לטעוג זכר נקבה או נקבה זכר (לעומת מכוניות אוטונומית למשל, שעדיף שתהיה "מחמירה" ותטעוג דברים כסכנתם מרוחת שאין באמת סכנה). ולכן, לא נעדי Precision גבוהה על פני Recall גבוהה, או הפוך.

מהו שילוב של Recall ו-Precision F1 Score? -
טוביים, עם טרייד טוב ביניהם.

הקוד להערכת המודל:

```
model = ResNet18(dropout_rate=0.2).to(device)

# Load the trained model
model.load_state_dict(torch.load('model.pkl', map_location=device))
model.to(device)

# Initialize the test dataset
_, test_dataset = init_datasets()

test_loader = DataLoader(test_dataset, batch_size=32, shuffle=True,
                        num_workers=8, persistent_workers=True,
                        pin_memory=False, drop_last=True)

model.eval() # Set the model to evaluation mode

total_outputs = torch.Tensor().to(device) # To store all outputs
total_labels = torch.Tensor().to(device) # To store all true labels

with torch.no_grad():
    for images, labels in test_loader:
        if torch.cuda.is_available():
            images = images.to(device)
            labels = labels.to(device)

        outputs = model(images)
        labels = labels.type(torch.FloatTensor).to(device)
        labels = labels.unsqueeze(1)

        # Append outputs and labels for calculating stats
        total_outputs = torch.cat((total_outputs, torch.round(outputs)), dim=0)
        total_labels = torch.cat((total_labels, labels), dim=0)

# Calculate evaluation metrics using the calculate_stats function
f1, precision, accuracy, recall = calculate_stats(total_outputs, total_labels)

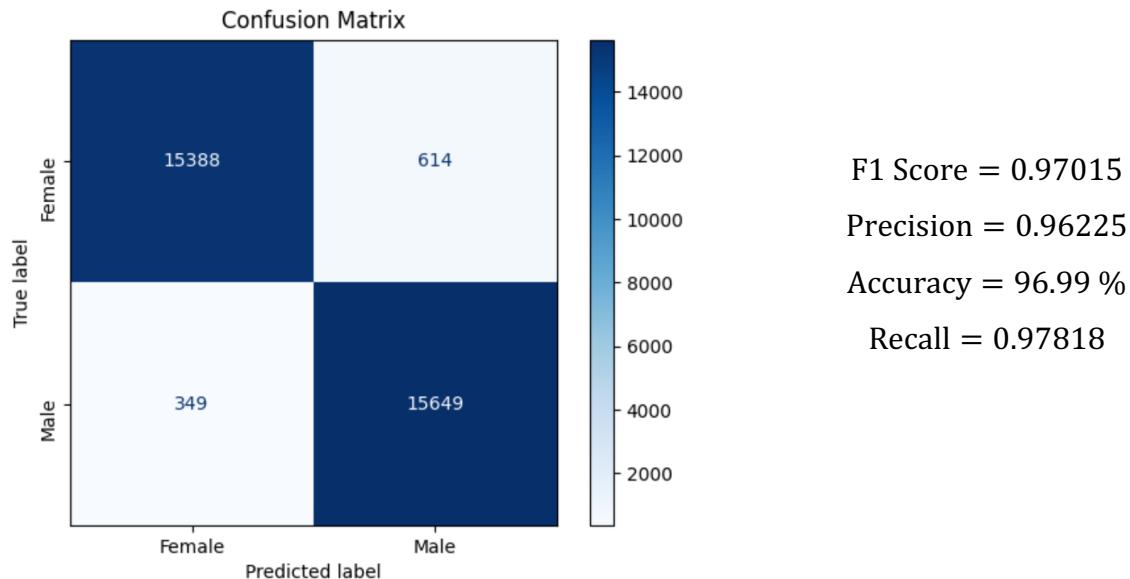
print(f"Test Results:")
print(f'\tF1 Score:\t{f1:.4f}')
print(f'\tAccuracy:\t{accuracy:.4f}')
print(f'\tPrecision:\t{precision:.4f}')
print(f'\tRecall:\t{recall:.4f}')
```

בנוסח, נציג את מטריצת הבלבול (Confusion Matrix) :

```
# Display Confusion Matrix:  
  
# Convert predictions and labels to NumPy arrays  
predictions = total_outputs.numpy()  
labels = total_labels.numpy()  
  
# Calculate the confusion matrix  
cm = confusion_matrix(labels, predictions)  
  
# Display the confusion matrix  
disp = ConfusionMatrixDisplay(confusion_matrix=cm)  
disp.plot(cmap=plt.cm.Blues, values_format='0f')  
  
# Add labels and title  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.title('Confusion Matrix')  
  
# Show the plot  
plt.show()
```

זכור כי "1" = גבר, ו"0" = נקבה.

תוצאות המודל



בsek הכל נציג כי אנו מרצוים עם תוצאות המודל 😊
כנו נציג שקיבלנו התאמת יתר קטנה, שכן הדיק של המודל עבור נתוני האימון היה 97.71%, בעוד שעל נתוני המבחן קיבלנו דיק של 96.99%. אולם, התאמת יתר זו היא זניחה (הודות לשיטות השונות בהם נמנענו מההתאמת יתר).

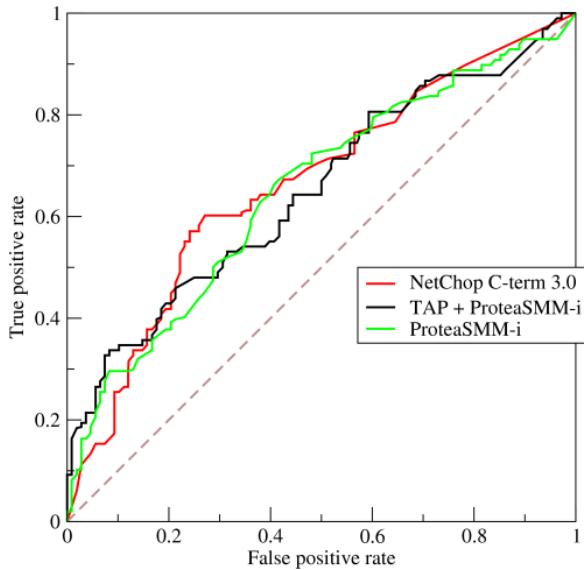
AUC, ROC Curve

נציין, כי בחישוב התוצאות בעמוד הקודם, בחרנו את סף החלטה להיות 0.5. כלומר, אם הערך שהמודול הוציא גדול מ-0.5, נקבע כי המודול סיוג את תמונה זו לזכר. אחרת, אם הערך שהמודול הוציא קטן מ-0.5, נקבע כי המודול סיוג את תמונה זו לנקבה. שכן הערך שהמודול מוציא הוא הסתברות שהתמונה היא של גבר, בחריה זו היא לגיטימית.

בחירת סף החלטה אחרים, יכולה לשנות את תוצאות המודול ולתת מטריצה לבול שונה. למשל, אם נבחר את סף ההחלטה להיות 1, נקבל כי כל התמונות יסוגו לנקבות (שהרי המודול מוציא ערך בין 0 ל-1). כਮובן בחירה זו היא גרוועה. המודול יהיה מצוין בסיווג נקבות (לא יטעה באף נקבה), אך יהיה גרווע מאד בסיווג זכרים (יטעה בכולם).

אמנם, יש מקרים בהם נרצה להעלוות או להוריד את סף ההחלטה מ-0.5. העלה של סף ההחלטה לערך גדול מ-0.5 מחייבת תקין את שיעור FP (שכן כעת המודול יעדיף לסווג ערכים כ"0") אמן תפגע בשיעור TP. לעיתים זה טרייד-אוּר שאנו מוכנים לקבל: למשל מודל שנבנה את הסיכויים שעכבר חוליה בניגף האבולה. מאוד חשוב לסווג את כל העכברים החולים באופן נכון לצורכי זהירות, גם אם זה אומר שנסוווג חלק מהעכברים הבריאים חולים. במצב זה, הורדת סף הבחירה היא בחירה לגיטימית.

ה (או ROC Curve) הוא גרפּה המהווה אמצעי לניתוח התוצאות של מסווג ביןאי. הגרף מתקבל על ידי התיוית שיעור TP מול שיעור FP, תחת ספי החלטה שונים.



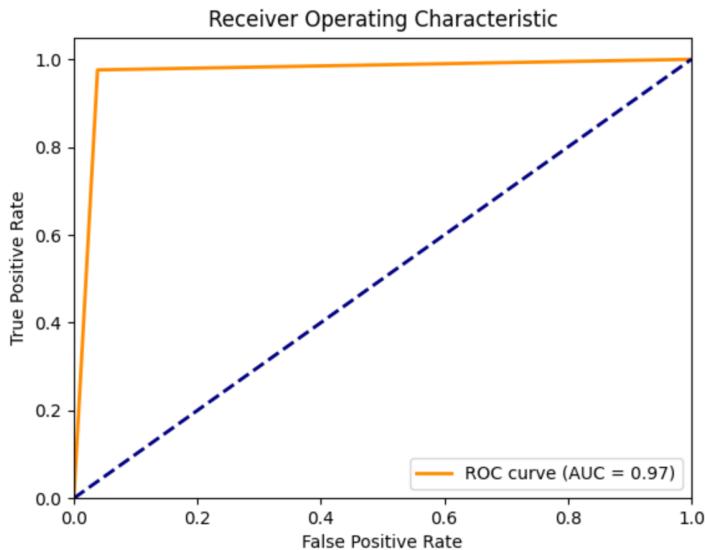
כל שהעקומה יותר רחוקה מהאלכסון התוצאות טובות יותר, שכן האלכסון הוא התוצאה היכולה להתקבל מתשובה אקראית לחלווטין (ניחוש אקרייאי ייתן $TP \text{ rate} \approx FP \text{ rate}$).

ה (Area Under Curve) הוא ממד שאומר לנו כמה העקומה רחוקה מהאלכסון, והוא השטח שמתකבל מתחת לעקומה. תוצאות גרוועות (העקומה קרובה לאלכסון) יהיו בעלי AUC קרוב ל-0.5. תוצאות טובות (העקומה רחוקה מעל האלכסון), יהיו בעלי AUC קרוב ל-1.

ניתן לחשב את ה-AUC ואת ROC Curve בעזרת הקוד הבא:

```
# Display ROC Curve, AUC:  
  
# Initialize arrays to store predictions and true labels  
y_true = total_labels.cpu().numpy()  
y_scores = total_outputs.cpu().numpy()  
  
# Calculate the ROC curve  
fpr, tpr, _ = roc_curve(y_true, y_scores)  
  
# Calculate the AUC  
roc_auc = auc(fpr, tpr)  
  
# Display the ROC curve  
plt.figure()  
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')  
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('Receiver Operating Characteristic')  
plt.legend(loc="lower right")  
plt.show()
```

ונקבל,



קיבלונו תוצאות טובות. משום שה-AUC קרוב מאוד ל-1 (ושווה 0.97), נסיק כי האלגוריתם כמעט ולא טועה לשני הçıוונים.

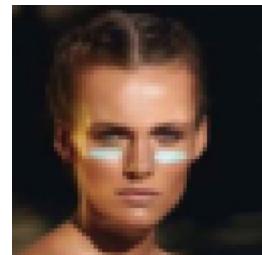
בחינת כישלונות של המודל

המודל אינו מושלם, והוא טואה עבור חלק מהדוגמאות בכל אחת משלושת הestyms של הנתונים (Train, Validation, Test). נציג מספר דוגמאות של תמונות (מסת נתונים המבחן) עבורם המודל טואה.

תמונה 5 – זכר.
המודל סיוג כנקבה.



תמונה 1 – נקבה.
המודל סיוג כזכר.



תמונה 6 – נקבה.
המודל סיוג כזכר.



תמונה 2 – נקבה.
המודל סיוג כזכר.



תמונה 7 – נקבה.
המודל סיוג כזכר.



תמונה 3 – זכר.
המודל סיוג כנקבה.



תמונה 8 – זכר.
המודל סיוג כנקבה.



תמונה 4 – זכר.
המודל סיוג כנקבה.



אכן אנו חושבים ש מרבית התמונות כאן באמת יכולות להעלות ספק (למשל תמונה 3, 6, 8 ואולי גם 7). כמו כן, ניתן לנחש ולהגיד שאולי המודל התבבלב בתמונות 4, 5 בשל כך שמופייעים שתי אנשים בתמונה.

שימוש במודל

ולסיום, ניתן לשחק עם המודל בעזרת קטע הקוד

```
import torch
from torchvision import transforms
from PIL import Image
from ResNet18 import ResNet18

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Load the model and set it to evaluation mode
model = ResNet18(dropout_rate=0.2).to(device)
model.load_state_dict(torch.load('model.pkl', map_location=device))
model.eval()

# Define the image transformations
transform = transforms.Compose([ transforms.Resize(64), transforms.CenterCrop(64),
                                 transforms.ToTensor()])
image_path = ""

while True:
    # Input path to the picture from the user
    image_path = input("Enter the path to the image: ")

    if image_path == "stop":
        break

    try:
        # Load and preprocess the image
        image = Image.open(image_path).convert('RGB')
        image = transform(image).unsqueeze(0).to(device)

        # Make a prediction using the model
        with torch.no_grad():
            output = model(image)

        predicted_value = output.item()

        if predicted_value > 0.5:
            probability = (predicted_value) * 100
            print(f"Male ({probability:.4f} %)")
        else:
            probability = (1 - predicted_value) * 100
            print(f"Female ({probability:.4f} %)")

    except FileNotFoundError:
        print("File not found. Please provide a valid image path.")
    except Exception as e:
        print(f"An error occurred: {e}")


```

הקוד פותח את התמונה בנתיב שהוזן, ומשתמש במודל כדי לסווג אותה כזכר או נקבה. בנוסף, הוא מדפיס את אחוז ה הודאות שלו (כasher 100% זה ודאי לגברי, ו 50% זה לא ודאי כלל). הקוד בכל פעם מבקש את הנתיב של התמונה הבאה לסיוג. אם רוצים לסיים את הרציה, נזין .stop

שיכחנו עם המודל בעצמנו, והיו לו תוצאות טובות :

```
>> python3 predict.py
Enter the path to the image: try/man1.jpg
Male (100.0000 %)
Enter the path to the image: try/man2.jpg
Male (96.3902 %)
Enter the path to the image: try/woman2.jpeg
Female (75.6504 %)
Enter the path to the image: try/man5.jpeg
Male (79.5763 %)
Enter the path to the image: try/woman1.jpeg
Female (55.9997 %)
Enter the path to the image: try/woman4.jpg
Female (99.3378 %)
Enter the path to the image: try/pic_of_jennifer_aniston.png
Female (88.2832 %)
Enter the path to the image: try/pic_of_brad_pitt.png
Male (100.0000 %)
Enter the path to the image: stop
```

התמונות שניתנו היו תמונות שלנו ושל חברים, או תמונות שלקחנו מהאינטרנט. לשמהותנו המודל צדק בכל התמונות. אמנם למשל עבור התמונה woman1.jpeg הוא לא היה וודאי כלל.



הערות

- סביר על תוכן כל קובץ בפרויקט ושמו.
- male_female_divide.py** - אחראי על יצרת 2 תיקיות Male ו-Female המכילות את כל התמונות בדאטאסט, מחולקות לפי הקטגוריה של זכר או נקבה (כזכור, קיבלנו את הדאטאסט כתיקייה אחת עם כל התמונות וקובץ CSV המכיל מאפיינים על כל תמונה).
- split_dataset.py** - אחראי על חלוקת התמונות שנמצאות ב2 התקיות Male ו-Female ל-3 התקיות (נוספות) בשם train, validation ו-test. החלוקת מתבצעת על פי היחס שקבענו.
- dataset.py** - מכיל את המחלקה GenderDataset ואת המתוודה לאתחול שלושת מאגרי הנתונים .(test, validation, train)
- ResNet18.py** - מכיל את מימוש הארכיטקטורה של המודל שבחרנו.
- train.py** - מכיל את המחלקה Trainer לצורך אתחול ואמון המודל. בנוסף מכיל פונקציית main לאימון הסופי של המודל לאחר בחירת ההיפר פרמטרים.
- grid_search.py** - מכיל את מימוש אלגוריתם Grid Search. משתמש במחלקה Trainer. לצורך אימון המודל עם הקומבינציות השונות של ערכי היפר פרמטרים.
- model.pkl** - מכיל את המודל הסופי שאימנו. קובץ זה הוא בעצם מה שעבדנו עליו לאורך הפרויקט.
- eval.py** - אחראי על ניתוח התוצאות בעזרת נתוני המבחן (תוך שימוש בmodel.pkl). מדפיס את המבדדים השונים ומציג מטريقת בלבול.
- predict.py** - מאפשר לשחק עם המודל ולנסות אותו עם תמונות לבחירתנו.