רשתות שיוריות

בפרק זה נסקור את מבנה הרשת ResNet, אשר היוותה את הבסיס לאלגוריתם המנצח בתחרות ImageNet בשנת 2015, עם אחוז דיוק Top 5 Accuracy של 96.43%. הסוד להצלחתה של ResNet הוא שימוש ברשתות עמוקות במיוחד: בעוד שהרשתות הזוכות בתחרות בשנים הקודמות היו רשתות עם 20-30 שכבות, מפתחי ResNet השתמשו ברשתות בעלות עד 150 שכבות.

דבר ידוע בקהילת זיהוי התמונה בשנים אלו היה שרשתות קונבולוציה עמוקות מצליחות להכליל טוב יותר משטוחות: המבנה ההיררכי של השכבות מאפשר לרשת לחלץ מאפיינים מורכבים מתמונת המקור, מאפיינים שימושיים למשימת הסיווג העוקבת. יחד עם זאת, רשתות עמוקות מהוות אתגר קשה יותר לאימון, עקב הצורך לפעפע את הגרדיאנט דרך מספר רב של שכבות. חישוב גרדיאנט זה מוטה לבעיות, שכן פעפוע הגרדיאנט לאחור דרך שכבה נתונה מבוצע על ידי **כפל** הגרדיאנט הקודם בגרדיאנט של שכבה זו. תוצאת הכפל של מספר רב של ערכים נוטה להתאפס (אם כופלים לרוב ערכים קטנים), או לשאוף לאינסוף (אם כופלים לרוב ערכים גדולים) – זוהי התופעה של הגרדיאנט המתאפס/המתפוצץ, אשר מובילה לאי התכנסות אלגוריתם האופטימיזציה.

הוספת שכבות נורמליזציה כגון Batch-Norm ואתחול נכון של ערכי הפרמטר הפכו את תהליך האימון של רשתות קונבולוציה עמוקות, עם עשרות שכבות, לאפשרי, אך רשתות עמוקות יותר עדיין הפגינו ביצועים פחותים: על סט הבדיקה כמובן אך גם על סט האימון עצמו, כך שלא היה מדובר בהתאמת יתר. דבר זה עומד בסתירה לאינטואיציה, שכן על פניו, רשת עמוקה יכולה ללמוד לחלץ מאפיינים זהים לאלו של רשת בעלת פחות שכבות ולהעביר אותם הלאה מבלי לבצע כל חישוב נוסף בשכבות הנוספות, וכך להגיע לביצועים זהים.

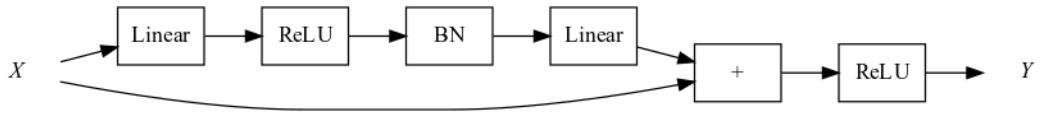
הבחנות אלו הובילו את צוות החוקרים האחראי לפיתוח ResNet להסיק כי פתרון אפשרי לבעיה יהיה לאפשר לשכבות ברשת ללמוד את פונקציית הזהות בצורה קלה יותר. פתרון זה בא לידי ביטוי ברכיב פשוט אשר נתאר לפרטים מיד, זהו חיבור הדילוג (skip connection). חיבור שכזה מהווה מעקף למספר שכבות, ונועד להפוך את פונקציית הזהות ל"ברירת המחדל" של שכבות אלו. תוך שימוש ברכיב זה, על השכבות המוקפות בחיבור דילוג ללמוד רק את **התוספת** לפונקציית הזהות שיש לחלץ, ואם אין כזו – איפוס הפרמטרים של השכבות יוביל למעבר קדימה של פונקציית הזהות.

בלוקים שיוריים

חיבורי דילוג באים לידי ביטוי בבלוקים שיוריים (residual blocks), אשר הרשת ResNet מורכבת משרשור שלהם ברצף. תחילה נגדיר בלוק שיורי עבור רשת בעלת קישוריות מלאה, בכדי לדון בתכונותיו, ואחרי כן נעבור להגדרת הבלוק המקורית, עבור רשתות קונבולוציה.

```
class ResBlockMLP(nn.Module):
    def __init__(self, in_features):
        super().__init__()
        self.relu = nn.ReLU()
        self.Z1   = nn.Linear(in_features,in_features)
        self.bn   = nn.BatchNorm1d(in_features)
        self.Z2   = nn.Linear(in_features,in_features)
    def forward(self, X):
        Y1   = self.Z1(X)
        Y1   = self.bn(Y1)
        Y1   = self.relu(Y1)
        Y1   = self.Z2(Y1)
        Y2    = Y1+X            #skip connection
        Y   = self.relu(Y2)
        return Y
```

ראו בהגדרת השכבות הליניאריות בעת אתחול הבלוק שמספר נוירוני הפלט (הפרמטר השני) שווה למספר נוירוני הקלט. זו אינה בחירה אקראית, שכן במעבר קדימה דרך הבלוק יש לסכום את פלט שכבות אלו איבר-איבר עם הקלט המקורי, ועל כן המימדים חייבים להסכים. חישוב זה, הנשמר במשתנה `Y2` הופך בלוק זה לשיורי: לפני האקטיבציה האחרונה מוסיפים לפלט הבלוק את הקלט המקורי ללא עיבוד. פעולת החישוב המבוצעת בבלוק מאויירת להלן.



מכיוון שפעולת הסכום היא גזירה, ניתן לפעפע אחורה את הגרדיאנט דרך הבלוק. לא רק שדבר זה אפשרי, הוא אף יכול להועיל לתהליך הלמידה: חיבור הדילוג מאפשר לגרדיאנט לעבור אחורה ללא הפרעה במקרים בהם הגרדיאנט של שאר הבלוק מתאפס. נראה זאת דרך חישוב מפורש, ולצורך זה נסמן את טנזור הפלט על רכיביו ב- $Y=\left(y_0,\dots,y_n\right)$, את טנזור הקלט ב- $X=\left(x_0,\dots,x_n\right)$ ובדומה את חישובי הביניים בבלוק בסימונים

$$Y1=\left(y_{1,0},\dots,y_{1,n}\right),$$

$$.Y2=\left(y_{2,0},\dots,y_{2,n}\right)$$

כעת, נניח שגרדיאנט פונקציית המחיר לפי פלט הבלוק חושב זה מכבר ויש לפעפע אותו לאחור. כלומר, הערכים $\frac{\partial C}{\partial y_k}$ ידועים לכל $k=0,\dots,n$ ויש לחשב את $\frac{\partial C}{\partial x_m}$ לכל $m=0,\dots,n$. נתחיל בשימוש ראשון בכלל השרשרת,

$$.\frac{\partial C}{\partial x_m}=\sum_{k=0}^{n}\frac{\partial C}{\partial y_k}\frac{\partial y_k}{\partial x_m}$$

את $\frac{\partial y_k}{\partial x_m}$ נחשב בתורו כך:

$$\frac{\partial y_k}{\partial x_m}=\frac{\partial \mathrm{ReLU}\left(y_{2,k}\right)}{\partial x_m}=\frac{\partial \mathrm{ReLU}\left(y_{2,k}\right)}{\partial y_{2,k}}\frac{\partial y_{2,k}}{\partial x_m}=1\left\{y_{2,k}\ge 0\right\}\frac{\partial y_{2,k}}{\partial x_m}$$

ונזכיר ש

$$1\left\{x\ge 0\right\}=\begin{cases}1 & x\ge 0\\ 0 & x<0\end{cases}$$

היא הנגזרת של $\mathrm{ReLU}\left(x\right)$ אשר חישבנו בעבר. בשלב זה יש לשים לב ש

$$y_{2,k}=y_{1,k}+x_k$$

לפי הגדרת הבלוק ולכן

$$.\frac{\partial y_{2,k}}{\partial x_m}=\frac{\partial\left(y_{1,k}+x_k\right)}{\partial x_m}=\frac{\partial y_{1,k}}{\partial x_m}+\frac{\partial x_k}{\partial x_m}=\begin{cases}\frac{\partial y_{1,k}}{\partial x_m} & k\ne m\\ \frac{\partial y_{1,k}}{\partial x_m}+1 & k=m\end{cases}$$

מכאן שגם אם הנגזרות דרך מסלול החישוב העליון בבלוק (ראו האיור לעיל) מתאפסות, קרי

$$\frac{\partial y_{1,k}}{\partial x_m}=0 \text{ לכל } k \text{ ו-} m,$$

עדיין מתקיים

$$,\frac{\partial y_{2,k}}{\partial x_m}=\begin{cases}0 & k\neq m\\ 1 & k=m\end{cases}$$

ובהצבה בחזרה בנוסחאות הקודמות מתקבל

$$.\frac{\partial C}{\partial x_m}=\sum_{k=0}^{n}\frac{\partial C}{\partial y_k}\frac{\partial y_k}{\partial x_m}=\sum_{k=0}^{n}\frac{\partial C}{\partial y_k}\cdot\frac{\partial y_{2,k}}{\partial x_m}\cdot 1\{y_{2,k}\geq 0\}=\frac{\partial C}{\partial y_m}\cdot 1\{y_{2,m}\geq 0\}$$

החשוב בתוצאה זו אינו הערך עצמו, אלא שהוא אינו מתאפס: ללא חיבור הדילוג, התאפסות הגרדיאנט דרך שאר הבלוק היתה גוררת $\frac{\partial C}{\partial x_m}=0$, ובהתאם לכל פרמטר השייך לשכבה **הקודמת** לבלוק זה ברשת היינו מקבלים, שוב לפי כלל השרשרת,

$$.\frac{\partial C}{\partial w}=\sum_{m=0}^{n}\frac{\partial C}{\partial x_m}\frac{\partial x_m}{\partial w}=0$$

משמעות הדבר היא שהלמידה בעזרת אלגוריתם מבוסס גרדיאנט עבור כל הפרמטרים הקודמים לכל הבלוק היתה עוצרת.

בלוקים שיוריים ברשתות קונבולוציה

בעוד שהדיון לעיל מצביע על הרלוונטיות הכללית של חיבורי דילוג, ResNet עצמה היא עודנה רשת קונבולוציה, ובהתאם בבלוק השיורי המתאים לה יש שימוש בפעולות הקונבולוציה במקום האגרגציה הליניארית. ראו זאת בקטע הקוד הבא.

```
class ResBlockConv(nn.Module):
    def __init__(self, in_channels):
        super().__init__()
        self.relu=nn.ReLU()
        self.conv1=nn.Conv2d(in_channels,in_channels,3,
                             padding="same",bias=False)
        self.bn1=nn.BatchNorm2d(in_channels)
        self.conv2=nn.Conv2d(in_channels,in_channels,3,
                             padding="same",bias=False)
        self.bn2=nn.BatchNorm2d(in_channels)
    def forward(self, X):
        Y1  = self.conv1(X)
        Y1  = self.bn1(Y1)
        Y1  = self.relu(Y1)
        Y1  = self.conv2(Y1)
        Y1  = self.bn2(Y1)
        Y2  = Y1+X
        Y = self.relu(Y2)
        return Y
```

מלבד החלפת השכבות הליניאריות בפעולות קונבולוציה עם גרעין 3X3 וריפוד המשאיר את מימד תמונת הפלט כמימד תמונת הקלט (עמודה אחת ושורה אחת של אפסים בכל צד במקרה זה), ישנם שני שינויים קטנים נוספים הנוגעים לשכבות ה-Batch-Normalization:

1. בבלוק זה יש שימוש בשכבות נורמליזציה מיוחדות עבור קונבולוציות, המנרמלות **כל ערוץ** בנפרד (במקום כל פיקסל בשכבת BN קלאסית).
2. מכיוון ששכבות ה-`BatchNorm2d` לומדות פרמטר bias משלהן עבור כל ערוץ, ואותו מוסיפות לכל הפיקסלים בערוץ לאחר הנרמול, אין צורך בפרמטר המבצע אותו חישוב בשכבת הקונבולוציה. לכן מועבר הפרמטר `bias=False` לבנאי השכבה.

על ידי שימוש בריפוד ושמירת מספר ערוצי הקלט כמספר ערוצי הפלט, גודל הטנזור Y1 במעבר קדימה בבלוק זה זהה לגודל הטנזור המקורי X, וכך ניתן לסכום אותם בחיבור הדילוג. עם זאת, יש לזכור שמטרתן של שכבות הקונבולוציה היא ללמוד מספר הולך וגדל של מחלצי מאפיינים (גרעיני הקונבולוציה), תוך כדי הקטנת רזולוציית תמונת הקלט. בהתאם, בתכנון הרשת המלאה יש צורך בבלוק שיורי נוסף המאפשר גמישות זו. ראו את מימושו בקטע הקוד הבא.

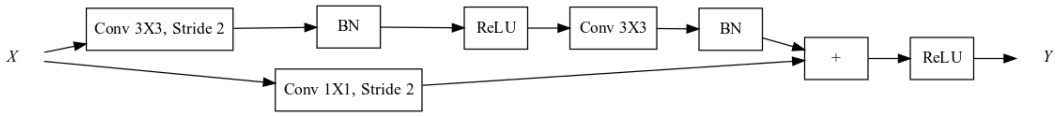
```
class ResBlockDownSamp(nn.Module):
    def __init__(self, in_channels):
        super().__init__()
        out_channels=in_channels*2                          #

        self.relu=nn.ReLU()
        self.conv1=nn.Conv2d(in_channels,out_channels,3,
                             padding=1,stride=2,bias=False)  #

        self.bn1=nn.BatchNorm2d(out_channels)
        self.conv2=nn.Conv2d(out_channels,out_channels,3,
                             padding="same",bias=False)
        self.bn2=nn.BatchNorm2d(out_channels)

        self.downsampX=nn.Conv2d(in_channels,out_channels,1  #
                                 ,stride=2,bias=False)
    def forward(self, X):
        Y1  = self.conv1(X)
        Y1  = self.bn1(Y1)
        Y1  = self.relu(Y1)
        Y1  = self.conv2(Y1)
        Y1  = self.bn2(Y1)
        Y2  = Y1 + self.downsampX(X)                        #
        Y   = self.relu(Y2)
        return Y
```

השינויים בין בלוק זה לבלוק הקודם מסומנים ב-#, ונועדו להקטין את מימדי האורך והרוחב פי חצי בעוד שמספר הערוצים גדל פי שתיים, וזאת על ידי שימוש בגודל פסיעה של 2. באיור,



נתבונן במימדי הטנזורים של חישובי הביניים בבלוק זה כדי להבין את השימוש בשכבת הקונבולוציה השלישית, downsampX: עבור טנזור קלט X בגודל $N\times C\times H\times W$, לאחר הקונבולוציה הראשונה גודל המשתנה Y1 יהיה $N\times 2C\times\frac{H}{2}\times\frac{W}{2}$ (בהנחה שמימדי האורך והרוחב היו זוגיים), וכך גם לאחר הקונבולוציה השניה, אשר משמרת את המימדים. כעת ברצוננו לסכום את תוצאת הביניים עם הקלט כדי לייצר את חיבור הדילוג, אך מימדיהם שונים. לכן טנזור הקלט מוזן לתוך קונבולוציה עם גרעין 1X1, לצורך הכפלת מספר הערוצים וחציית מימדי האורך והרוחב. שאר החישוב דומה לזה המבוצע בבלוק הקודם.

רשתות שיוריות (ResNets)

בעוד שלעיל דנו ב-ResNet כאילו מדובר ברשת אחת, למעשה זהו מתכון לתכנון מספר רשתות בעלות עומק משתנה, אותן ניתן לטעון ישירות דרך הספרייה PyTorch:

```
import torchvision.models as models
resnet18 = models.resnet18()
print(resnet18)
```

בהרצת קטע קוד זה יוגדר אובייקט רשת ResNet בעלת 18 שכבות מבנה השכבות שלה יודפס. מהפלט ניתן לראות שהיא מורכבת מסדרה של בלוקים שיוריים רגילים המחוברים לבלוקים שיוריים המגדילים את מספר הערוצים. ההבדל בין רשת זו לבין רשתות ResNet אחרות, כגון ResNet50 הוא במספר הבלוקים בסדרה בלבד.

מעניין להתבונן בשכבות האחרונות ברשת, אליהן מוזן פלט מחלץ המאפיינים המבוסס על שכבות הקונבולוציה:

```
last_layers=list(resnet18.children())[-2:]
print(*last_layers,sep="\n")
```

פלט:

```
AdaptiveAvgPool2d(output_size=(1, 1))
Linear(in_features=512, out_features=1000, bias=True)
```

השכבה האחרונה בעלת אלף נוירונים, כצפוי מרשת אשר מיועדת לסווג את תמונות ImageNet לאלף מחלקות שונות .פלט זה יוזן לפונקציית ה-Softmax ליצירת הסתברויות השייכות למחלקות. השכבה הקודמת לה מחשבת ממוצע של כל אחד מ-512 הערוצים בפלט סדרת הקונבולוציות על פני מימדי האורך והגובה. בעזרת שכבה זו אפשר להזין לרשת קלט אשר אינו בדיוק הגודל הצפוי (תמונות 224X224): בהתאם לגודל התמונה המוזנת לרשת, אורך ורוחב הפלט של מחלץ המאפיינים ישתנו, אך מספר הערוצים יישאר 512 והשכבה האדפטיבית תתאים את המימדים לאלו הדרושים לשכבת הסיווג. כך יהיה ניתן להשתמש ברשת המאומנת גם עבור תמונות בעלות רזולוציה שונה מהמקורית.

שאלות לתרגול

1. הסבירו למה ברירת המחדל של בלוק שיורי היא לחשב את פונקציית הזהות אם הוא לא תורם להורדת פונקציית המחיר. התייחסו לתהליך אימון בו משתמשים ברגולריזציה של הפרמטרים.
2. ספרו את מספר הפרמטרים ב"ראש הסיווג" המופיע בסוף ResNet18. השוו מספר זה למספר הפרמטרים במסווג של AlexNet.