אופיר בן שוהם
ת.ז 208642496

דוח תרגיל 3 - למידה חישובית

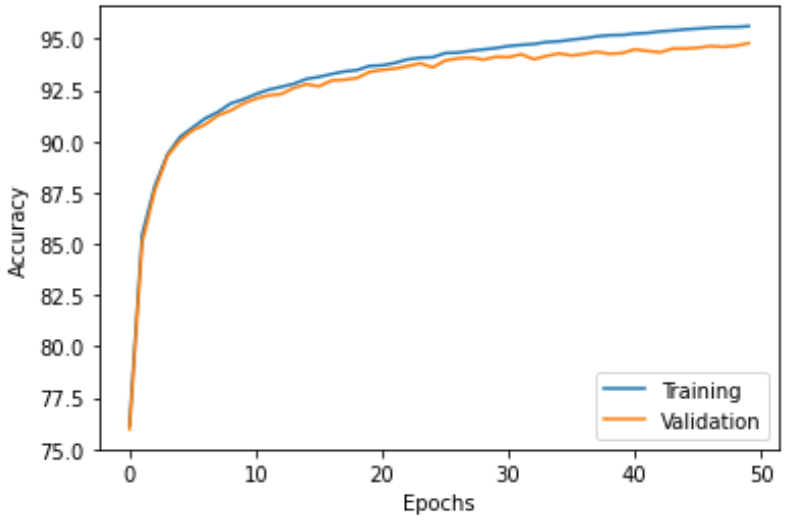
חלק 1:

בחלק הראשון, המטרה הייתה לבנות מודל עבור MNIST classification.
בחנתי שלוש רשתות נוירונים - אחת עם hidden layer בודד, מתוך המחברת המצורפת בתרגיל.
לאחר מכן הוספתי hidden layer נוסף, כך שיהיו שתי שכבות ברשת. עד כה המודלים נבנו מאפס, ללא שימוש בספריות כמו Pytorch/Keras/tf.
לבסוף, מימשתי רשת נוירונים עם 2 שכבות נסתרות, בעזרת Pytorch.

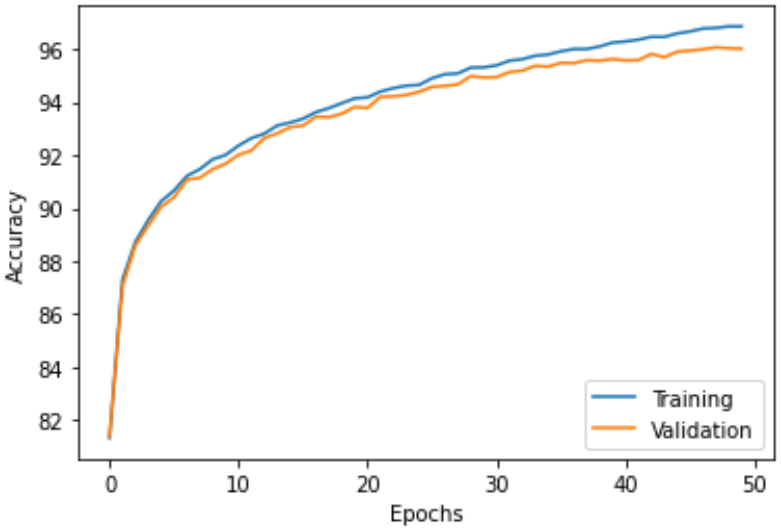
להלן ההשוואה בין התוצאות השונות שלהם:

מודל	מספר אפוקים	test accuracy(%)	validation accuracy(%)	train accuracy(%)	macro auc
רשת נוירונים בסיסית מאפס, עם hidden layer בודד, כפי שיש במחברת המצורפת.	50	94.54	94.78	95.61	0.996
רשת נוירונים מאפס עם שתי שכבות נסתרות	50	95.78	96.01	96.85	0.997
רשת נוירונים עם pytorch	20	91.66	91.89	92.25	0.991

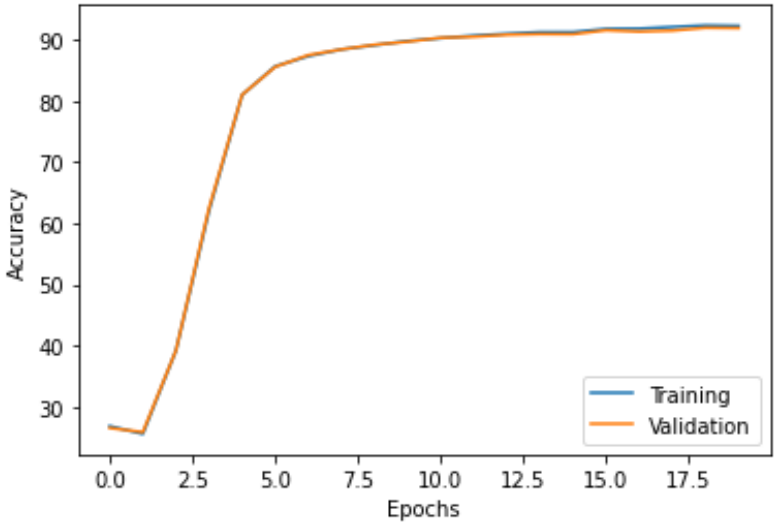
להלן גרף accuracy per epoch עבור המודל הראשון:



ועבור המודל השני:



ועבור המודל השלישי:



עבור המודל השלישי. נראה שה-train וה-validation די אחידים כאן באחוזי הדיוק. התבקשנו לבצע 20 אפוקים עבור השלישי, מניסיונות שבדקתי במידה ומספר האפוקים שלו גדל אז האחוזי דיוק שלו ממשיכים להשתפר בהתאם, גם על ה-train וגם על ה-validation וגם על ה-test.
עבור המודל השני נראה שלקראת הסוף יש overfitting קטן מאחר והדיוק על ה-train עולה והדיוק על ה-validation נשאר די דומה. כנ"ל עבור המודל הראשון.
עבור שלושת המודלים, ה-macro auc יצא 0.99, ההבדלים הקטנים ביניהם מתוארים בטבלה לעיל.

חלק 2:

קישור למחברת - https://github.com/Ofir408/cnn-flowers-classification

בחלק זה, המטרה הייתה לבנות מודל עבור בעיית ה-flower classification. השתמשתי ב-Transfer Learning משני מודלים:

1. מודל resnet152
2. מודל convnext_large

עשיתי fine tuning לכל אחד מהמודלים שלהם, ובניתי מודל fine-tuned על כל אחד מהם, עבור הבעיה הנוכחית שלנו. השתמשתי ב-Pytorch.

עבור resnet:
במקום השכבה האחרונה שלו, הוספתי:

1. פונקציה לינארית ממספר הפיצרים שלו בשכבה האחרונה אל 256.
2. פונקציית אקטיבציה Relu
3. פונקציה לינארית מ256 פרמטרים ל102.
4. פונקציית LogSofmax.

כנ"ל עבור convnext_large, רק כמובן עם מספר פיצרים שונה ב(1), עבורו הפונקציה הלינארית שהוגדרה במקום (1) היא מ1536 פיצרים ל-256, ומשם באופן דומה כפי שפורט לעיל.

פיצלתי את ה-dataset באופן הבא:

1. 50% עבור ה-train
2. 25% עבור validation set
3. 25% עבור test set

פירוט על ה-Preprocessing שבוצע:

<u>איסוף הדאטא:</u>
טענתי את ה- imagelabels.mat שמכיל את התיוגים. טענתי גם את 102flowers.tgz שמכיל את התמונות.
יצרתי 3 תיקיות: train, validation, test, שבכל אחת מהן יהיו התמונות שרלוונטיות אליה.
לאחר מכן, עבור כל אחת מהתיקיות האלו, יצרתי תתי תיקיות כך שכל תיוג יהיה תיקייה נפרדת.
העתקתי עם סקריפט את התמונות שרלוונטיות לכל דאטאסט, אל התיוג המתאים שלו.
לדוגמה עבור תמונה ששייכת ל- train והתיוג שלה 1, היא תופיע בתוך תיקייה בשם 1 שנמצאת בתוך תיקייה בשם train, וכן הלאה. הכל כמובן בוצע באופן אוטומטי בעזרת קוד שמופיע במחברת שלי.

<u>טרנספורמציה שבוצעה על התמונות במסגרת ה-preprocessing:</u>

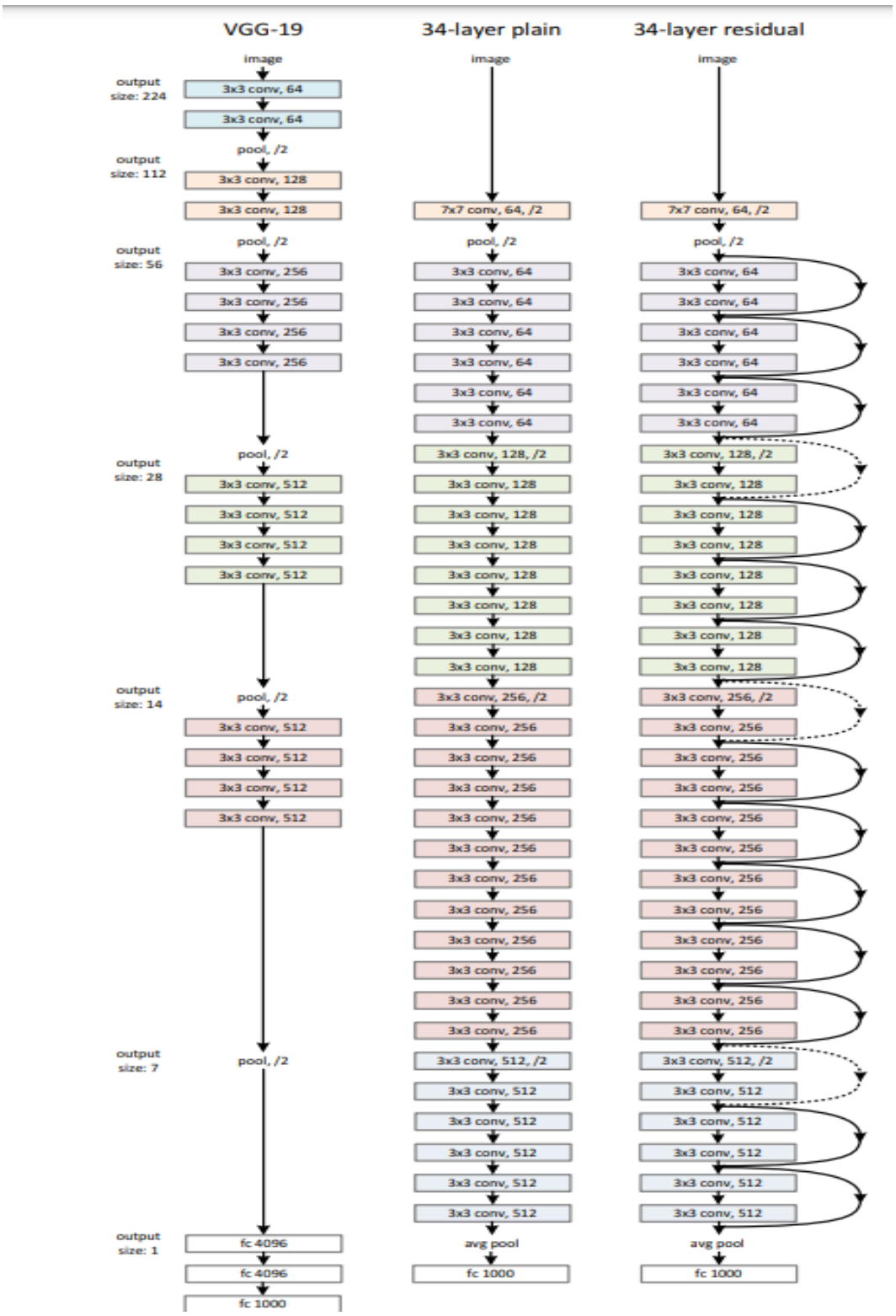
1. שיניתי את הגודל של התמונה ל256.
2. עשיתי Center Crop של 224.

3. המרתי ל-tensor
4. ביצעתי נורמליזציה נפוצה ב-pytorch עם mean של $[0.485,\ 0.456,\ 0.406]$ ו- std של $[0.229,\ 0.224,\ 0.225]$

<u>טעינת הדאטא</u>: בעזרת ImageFolder של torchvision.

<u>אופן האימון</u>: אימנתי את המודלים ב-colab עם gpu חינמי.

<u>פירוט נרחב יותר על המודל הראשון - fine tuning ל-resnet152:</u>
מודל ResNet עם 152 שכבות.



מימין - ארכיטקטורה בסיסית של resnet. ל-resnet-152 יש 152 שכבות. לפי המאמר, resnet-152 השיג טעות של 3.57% מעל ImageNet. למרות שהמודל מאוד עמוק (עם הרבה מאוד שכבות) הוא מצליח להגיע לביצועים מאוד טובים ונמנע מבעיית ה-Vanishing Gradients.
ב-resnet-152 יש הרבה שכבות שמורכבות משכבות קונבולוציה, שכבות max polling, שכבות fully connected.

במאמר מוצגת ארכיטוקטורה של resnet-152 עבור ה-ImageNet:

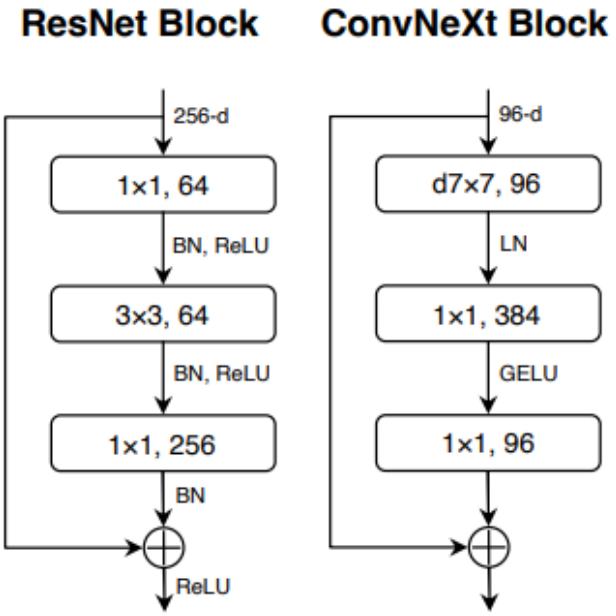
layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

כאמור ניתן לראות בו שימוש בקונבולוציות, ב-max polling, ב-average pooling, ב-fully connected וב-softmax.

על שכבת ה-fully connected האחרונה של resnet, עשיתי fine tuning עבור בעיית ה-flower classification שלנו ושיניתי אותה, כך שבמקום הנוכחית היא תכיל את הבא:

1. פונקציה לינארית ממספר הפיצרים שלו בשכבה האחרונה אל 256.
2. פונקציית אקטיבציה Relu.
3. פונקציה לינארית מ256 פרמטרים ל102.
4. פונקציית LogSofmax.

פירוט נרחב יותר על המודל השני - fine tuning ל-convnext_large:
קישור למאמר הרלוונטי: https://arxiv.org/pdf/2201.03545.pdf



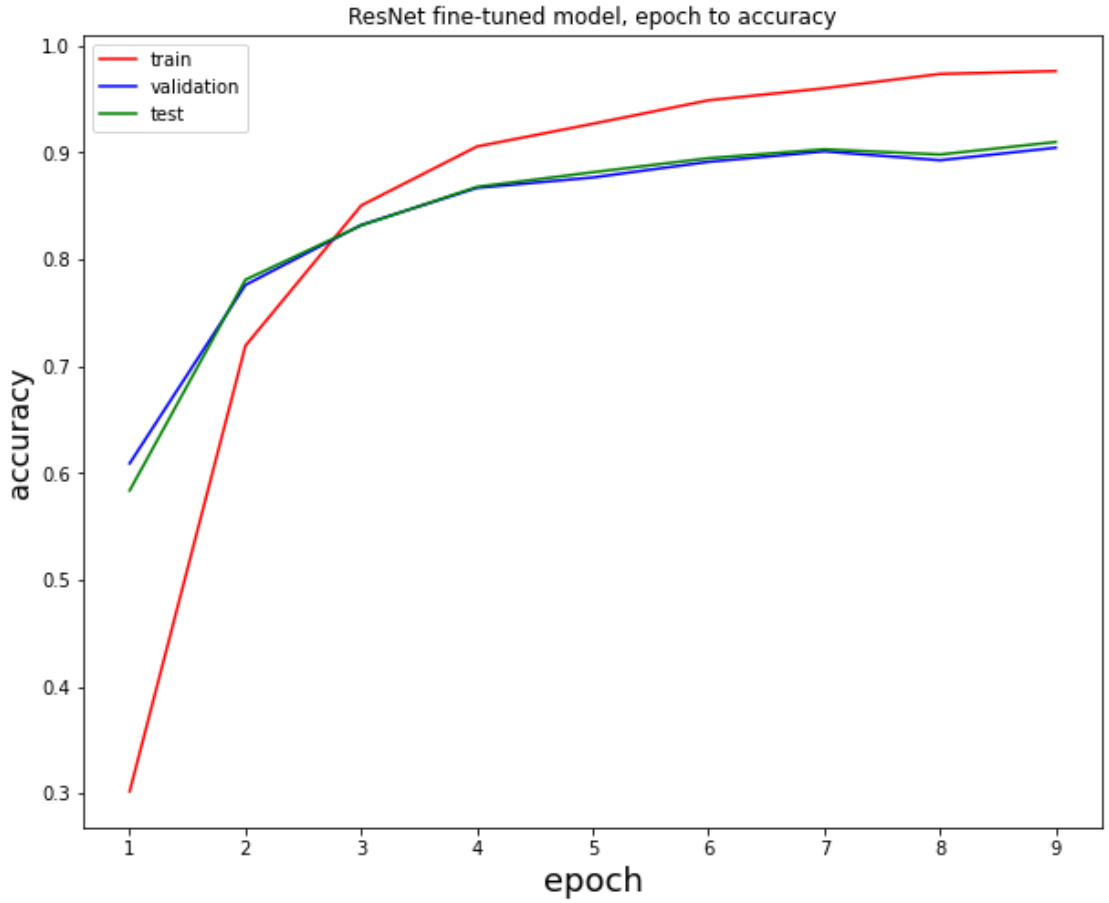
ב-convnext לעומת resnet, הבלוק מורכבת ממימדים שונים, ופונקציות אקטיבציה שונות. לדוגמה אפשר לראות שב-Resnet משתמשים ב-ReLu בתור פונקציית אקטיבציה בעוד שב-ConvNext משתמשים בפונקציית אקטיבציה אחרת בשם GELU.

באופן כללי, מודל convnext נבנה ממודל ResNet50 בסיסי (פירוט לגבי ארכיטקטורת ResNet ניתן למצוא בעמודים הקודמים) ועשו לו מודרניזציה בחלקיו השונים והוסיפו לו Transformer. בנוסף הגדילו לו את ה- Kernel size לעומת Resnet.
בנוסף הוסיפו נורמליזציה לבלוקים של ה-Transformer.

תוצאות:

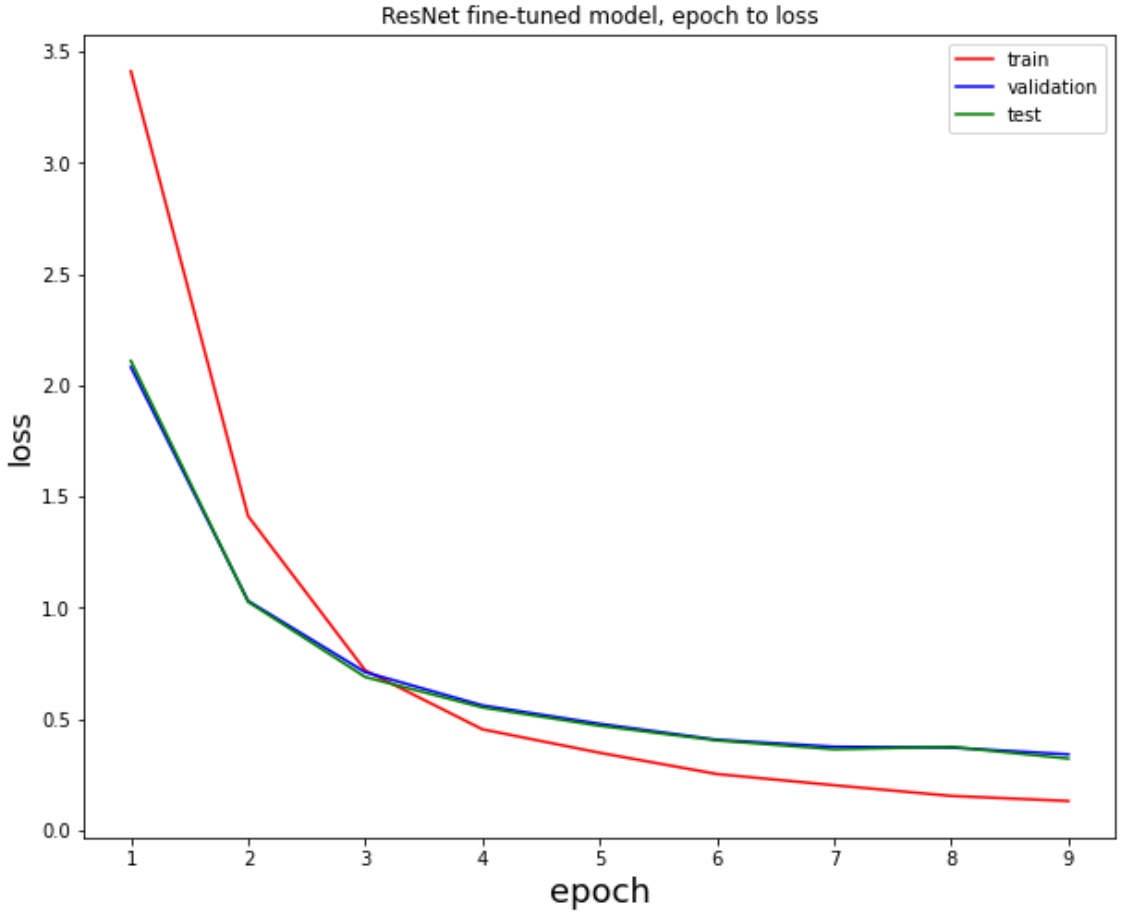
עבור המודל הראשון (fine tuning על resnet-152):

גרף של accuracy כתלות ב-epoch:
הרצתי 9 אפוקים



הגעתי ל- 90% דיוק על ה-validation, וקצת יותר על ה- test כפי שאפשר לראות בגרף למעלה.

בנוסף נראה גם גרף של loss כתלות ב-epoch:

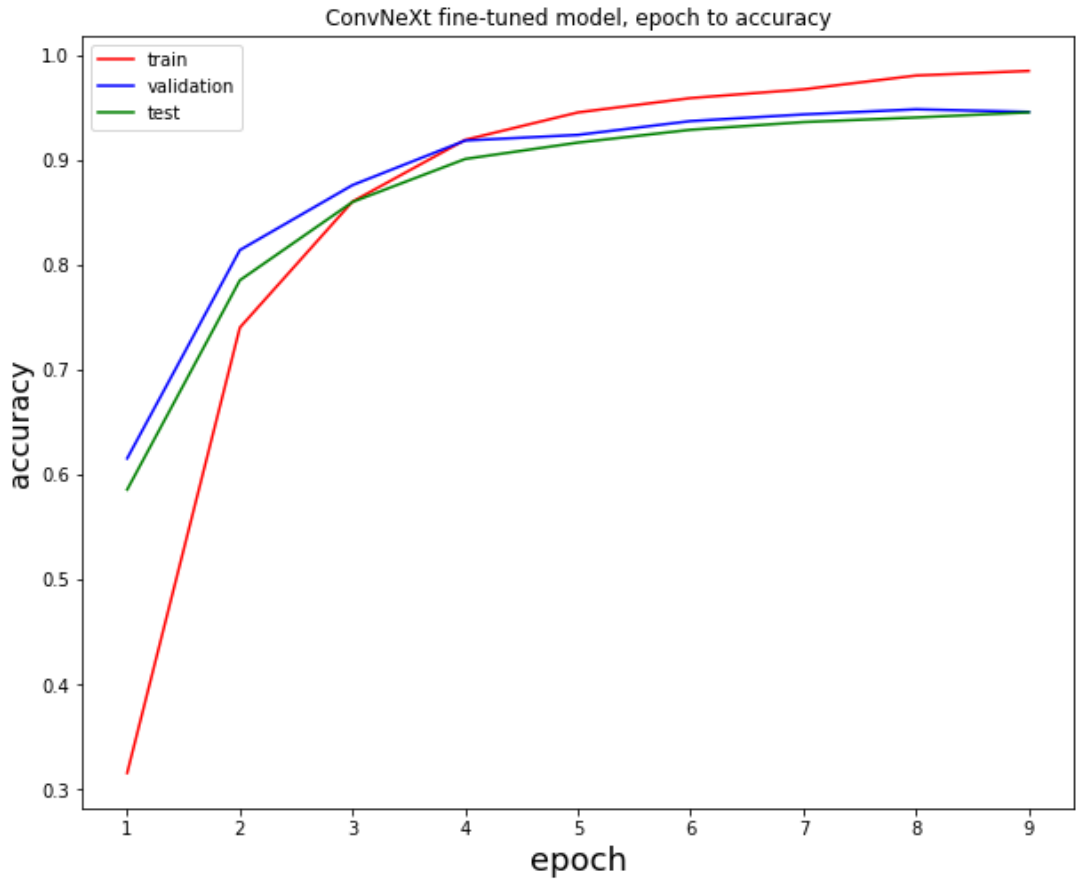


ניתן לראות שה-loss די ממשיך לרדת ככל שמספר האפוקים ממשיך, גם על ה-train, גם על ה-validation וגם על ה-test. כן יש עלייה קטנה ב-validation, test בין 7 ל8 אבל אפשר לראות שב-9 זה די מתאזן וחוזר למצב הרגיל. אם נסתכל גם על אחוזי הדיוק ב7 לעומת 9, נראה שב9 יש אחוזי דיוק טובים יותר על כל אחד מהם (train, test, validation) ולכן השארתי את האימון עם התשעה אפוקים.

עבור המודל השני - בו ביצעתי fine tuning על convnext_large.

כאשר הרצתי מחדש (עם פיצול רנדומלי חדש על הדאטאסטים), קיבלתי תוצאות די דומות עד כדי סטייה קטנה של 0.2+- באחוזי דיוק, שנבעו מרנדומיזציה.

גרף של accuracy כתלות ב-epoch

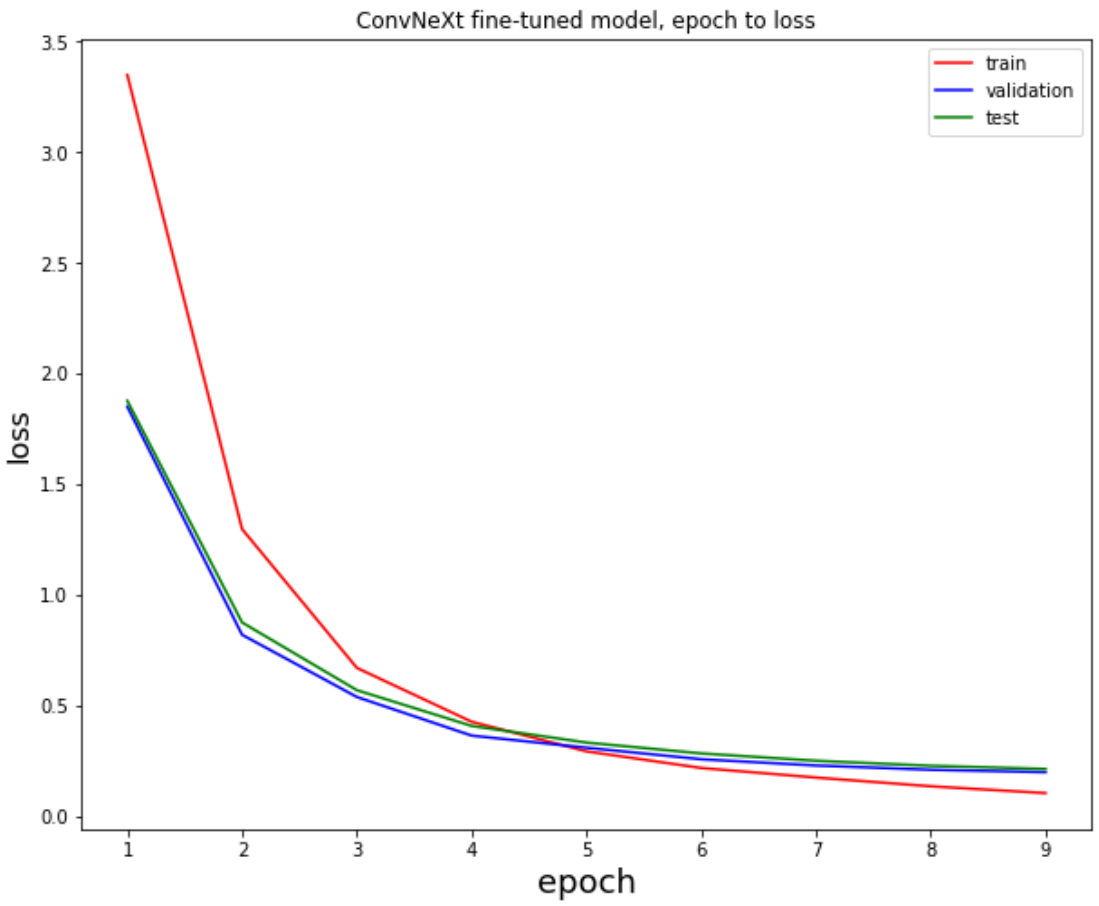


ניתן לראות שבגדול ככל שמספר האפוקים עולה אז הדיוק משתפר.
לבסוף קיבלנו אחוזי דיוק די גבוהים במודל הזה:

- קיבלנו validation accuracy של 0.9463
- ו- test accuracy דומה. ניתן לראות שבגרף לעיל ה-test, validation נמצאים באותה נקודה אחרי האפוק ה9 והאחרון.

כלומר הגענו לביצועים די גבוהים בזכות המודל השני.

בנוסף, מצורף גם גרף של loss כתלות ב-epoch עבור המודל השני:



גם כאן נראה שה-loss יורד ככל שמספר האפוקים עובר, והוא נמוך יותר עבור ה-train set לעומת ה-validation, test, שזה די הגיוני כל עוד הפער ביניהם לא גדול מדי. כאן הפער ביניהם נראה סביר ולא נראה שאנחנו בבעיית overfitting.

בנוסף לפי אחוזי הדיוק על ה-validation ועל ה-test נראה שהמודל כן יודע לעשות הכללה טובה גם לנתונים שלא ראה ב-train.

לסיכום,
עבור המודל ראשון קיבלנו 90% דיוק על הטסט ועבור המודל שני קיבלנו 94.63% דיוק על הטסט.