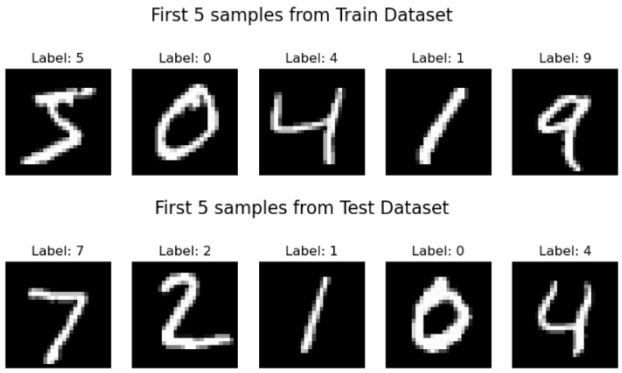
בס"ד

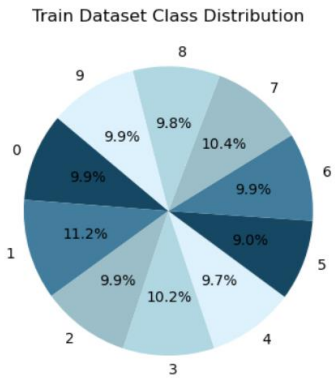
Project 1 - Neural network

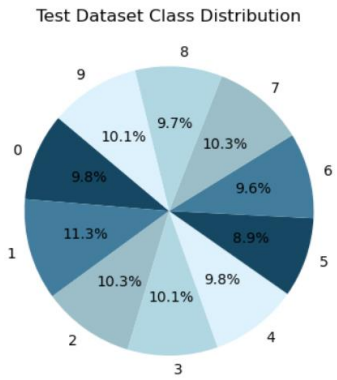
מגישה: גל חלילי

Data exploration:

```
Dataset labels - classes: ['0 - zero', '1 - one', '2 - two', '3 - three', '4 - four', '5 - five', '6 - six', '7 - seven', '8 - eight', '9 - nine']
```







ניתן לראות שיש יותר דוגמאות של הספרה 1 גם בסט האימון וגם בtest.

יש מחלקות שבהן האחוזים דיי דומים וכאלו שפחות – יכול להיות שישפיע בהמשך.

בס"ד

בניית הרשת:

גודל שכבת ה input וה-output תלוי בבעיה שנדרשים לפתור-

שכבת input – מספר הנוירונים בשכבת הקלט נקבע לפי מספר הפיצ'רים שיש לכל sample ב-Dataset.
שכבת הקלט היא שכבת Flatten, אשר מעצבת מחדש את תמונות הקלט לטנזור חד מימדי.
כל תמונה במערך הנתונים של MNIST היא תמונה בגווני אפור של 28X28 פיקסלים.
לאחר ההשטחה, לטנזור הקלט יש גודל של 28*28 = 784.

שכבת output – מספר הנוירונים בשכבת הפלט נקבע לפי מספר הקלאסים שיש בבעיה (ב-Dataset).
ב-MNIST יש תמונות של ספרות מ- 0 עד 9 כלומר 10 ספרות שונות וכל סיפרה היא בעצם class כלומר יש 10 קלאסים שונים בDataset זה - לכן בניתי את הרשת כך שבשכבת הפלט יש 10 נוירונים.

בין שכבת הפלט לקלט יש שכבות חבויות (hidden layers) – אנחנו יכולים לבחור כמה שכבות לשים ואת גודלן – אלו הם hyperparameters.

(בסוף הקובץ הוספתי הסבר מפורט על הארכיטקטורה של המודל הסופי).

חיפוש hyperparameters:

בשביל לקבוע את ערכי פרמטרים אלו השתמשתי ב-grid search – עוברים על כל הקומבינציות האפשריות של הערכים הנתונים ומחפשים את הפרמטרים שנותנים את הביצועים הכי טובים.

ניסוי 1-

num_layers: [1, 2, 3], layer_size: [64, 128, 256].

בניסוי זה קיבלתי את הaccuracy הגבוה ביותר 0.91935 עבור 2 שכבות בגודל 256.

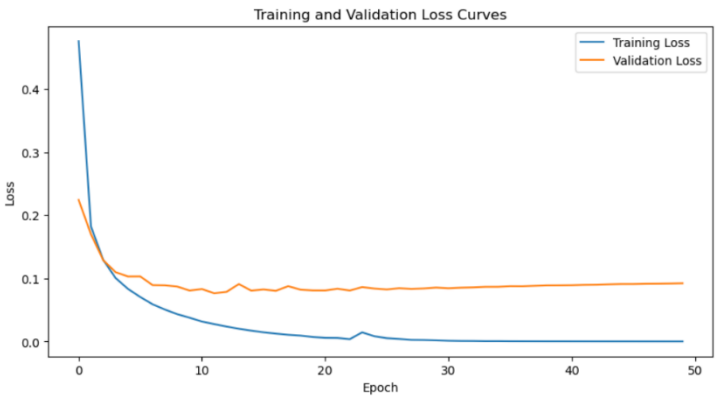
זו לא משימה קשה ולכן הגיוני שלא צריך רשת מאוד גדולה בשביל הסיווג הזה.

ניסוי 2-

הגדרתי רשת עם 2 שכבות חבויות כל אחת בגודל 256 ועכשיו בדקתי איך קבוע קצב הלמידה משפיע על הביצועים כלומר עברתי עם grid search על lr: [0.1, 0.01, 0.001].

התוצאה הטובה ביותר הייתה accuracy =0.977 כאשר הlr היה 0.1.

זה גרף שמציג את ה Loss של הtrain ושל הvalidation עבור כל epoch.



ניתן לראות שהLoss על הtrain יורד ואילו

על ה validation עולה. זה מקרה קלאסי של

Overfit. נמשיך בניסויים ונראה האם המצב משתנה.

בס"ד

הסבר אפשרי:

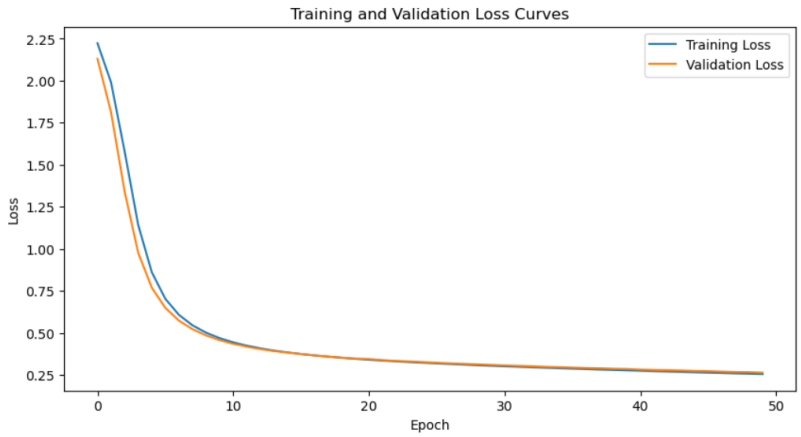
לפני כן ה lr היה 0.01 כלומר הגדלתי את קצב הלמידה והביצועים השתפרו. ניתן לומר שזה קרה מכיוון שהצלחתנו להגיע קרוב יותר למינימום הלוקלי או שהגענו למינימום לוקלי אחר נמוך יותר כי כל פעם עשינו צעד גדול יותר.

ניסוי 3-

כעת רציתי לבדוק האם שינוי שיטת האופטימיזציה יעזור. לכן שוב השתמשתי בgrid search.

optimizer: ['gd', 'sgd', 'mini_batch']

התוצאה הטובה ביותר הייתה accuracy = 0.98138 עבור GD.



ניתן לראות שהLoss של ה Train וגם של

הValidation במגמת ירידה מתמדת ותופעת

ה-Overfit נעלמה.

הסבר אפשרי:

GD אומנם פחות יעיל מבחינת זמן חישוב כי עוברים על כל הדוגמאות בסט האימון אבל בגלל שהוא משתמש בכל איטרציה בכל הדוגמאות הוא יותר יציב מאשר לקחת כל פעם דוגמה אחת (SGD) או קבוצה של דוגמאות (Mini batch) ולכן הביצועים השתפרו.

בכל הניסויים הללו השתמשתי בRelU בתור פונקציית אקטיבציה.

לא ניסיתי לשנות את פונקציית האקטיבציה אפילו שהיא hyperparameter מכיוון שהגעתי לביצועים טובים (אמרנו בהרצאות שזו פונקציית האקטיבציה שבדרך כלל עובדת הכי טוב ולכן החלטתי בהתחלה להשתמש בה).

מניעת overfitting:

השתמשתי בשתי שיטות למניעת Overfit:

1. Dropout-

זו שכבה שמטרתה לאפס חלק מהנוירונים משמע לא להעביר דרכה את הערכים. מטרת השכבה ליצור הכללה ולמנוע overfitting. בכל איטרציה אנחנו משמיטים חלק מהנוירונים, נאפס כל נוירון בסיכוי p (p גדול יותר גורר יותר נוירונים מאופסים) כך שבעצם חסר לו מידע אחר כל פעם והוא עדיין מנסה להצליח (למזער את הלוס), מה שגורם לו "לא להיות תלוי" בקומבינציה הספציפית של הפיצ'רים שראה בסט האימון – כלומר לומד להכליל ולא לשנן את הקשרים הספציפיים שראה.

בס"ד

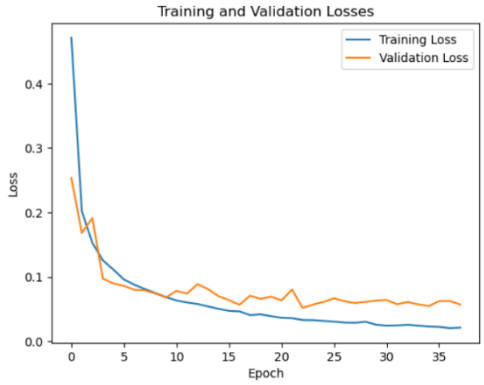
2. Early stopping-

גם מספר הepochים יכול להשפיע על overfit. נסתכל על ה-validation set loss (בקיצור vsl) ונחליט לעצור את האימון כך שהוא לא יגדל. בעצם השיטה אומרת שכל פעם שמזהים עלייה בvsl נשמור את הepoch בו זה קרה ונתחיל לספור n (לבחירתנו) epochs שבהם נבדוק אם זו אכן מגמת עלייה אמיתית או שזו הייתה עלייה לצורך ירידה. אם זו אכן מכמת עלייה ולא זיהינו מאז ירידה בvls אז נחליט לעצור את הריצה ב epoch ששמרנו. אבל אם זיהינו מאז ירידה אז נמשיך בתהליך.

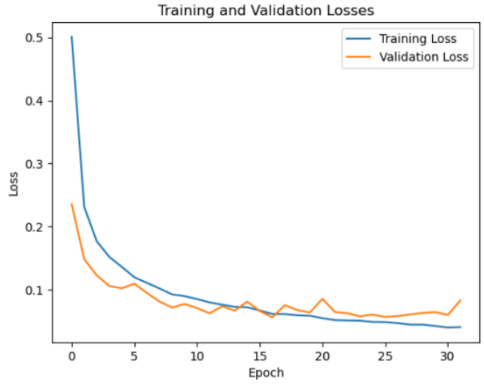
שיטה זו עוזרת במניעת overfitting בגלל שעליה ב loss של ה-validation מסמנת על overfit לכן לפי עלייה זו נחליט מתי לעצור בשביל לא להיכנס לoverfit. בגלל ששיטה זו עוזרת למנוע overfitting היא עוזרת למודל להכליל טוב יותר במקום לשנן כי עוצרת אותו לפני שהוא מתחיל ללמוד קשרים ותלויות שספציפיים לדוגמאות שהוא רואה בסט האימון ואם ילמד וישנן אותן הוא יהיה טוב מאוד על הtrain ופחות טוב על הvalidation ולכן הvls יעלה.

יצרתי שכבת dropout אחרי כל שכבה. ניסיתי כמה ערכים של p ושל n אציג את הערכים והגרפים:

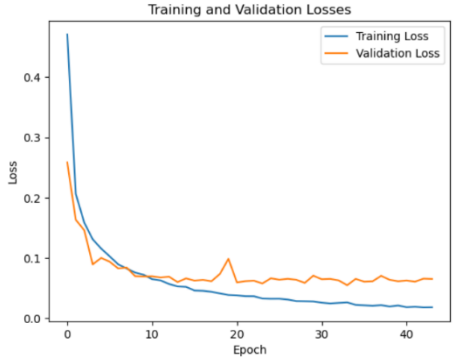
acc = 0.9849 <- P = 0.2, n= 15



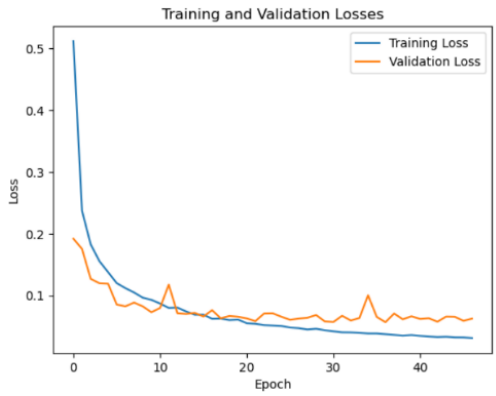
acc = 0.9768 <- P = 0.3 , n= 15



acc = 0.9844 <- P = 0.2 , n= 10



acc = 0.9842 <- P = 0.3, n= 10



acc = 0.9863 <- P = 0.3 , n= 20

בס"ד

התחלתי מ- p=0.2 ו-n=15 וראיתי שיש overfit לכן החלטתי לנסות להגדיל את p כלומר להשתיק עוד נוירונים ובעצם להקשות עוד על הלמידה כדי שלא ישנן את מה שראה ב-train.

אז הגדלתי את p להיות 0.3 וn עדיין 15 – זה שיפר את ה-overfitting אבל עדיין רציתי לנסות לשפר עוד יותר.

ניסיתי p = 0.3, n=10 כי חשבתי שאולי אני נותנת יותר מידי אפוקים בניסיון להשתפר ואז המודל משתפר קצת על הvalidation אבל הרבה על הtrain . בגרף ניתן לראות שזה לא עזר ואפילו ה-overfitting נהיה חמור יותר.

משם ניסיתי את ההפך – להגדיל את מספר האפורים בטענה שאולי אני לא נותנת מספיק זמן להשתפרות ומפספסת. אז ניסיתי p=0.3, n=20 וזה גם לא שיפר ורק החמיר.

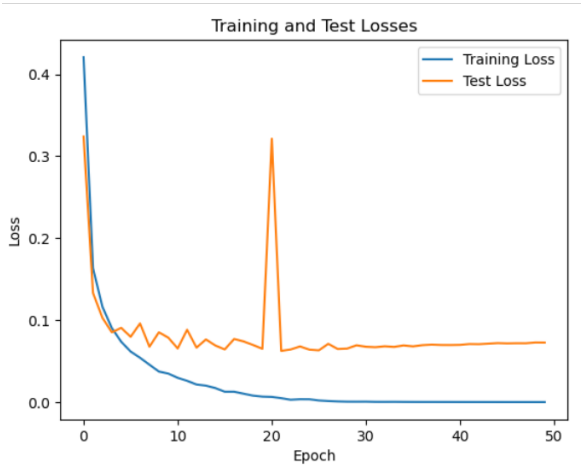
מפה ניסיתי p=0.2, n=10 רציתי לחזור לdropout ההתחלתי ולראות מה קורה כאשר אני מקטינה את n. זה החמיר את הoverfit מהניסויי הקודם כי הקשתי פחות על המודל.

אנסה להסביר למה פתאום יש overfit:

- n נמוך מידיי יכול להפסיק את האימון לפני שהמודל מתכנס לפיתרון אופטימלי, מה שיכול להוביל לביצועים לא טובים על ה- validation ויותר טובים על סט האימון עצמו.

- המודלים עם ה dropout יותר מורכבים מאשר המודל הקודם שניסיתי ויכול להיות שהם מורכבים מידיי עבור המשימה. אם המודל מורכב מדי עבור מערך הנתונים, הוא עשוי ללכוד רעש בנתוני האימון במקום בדפוסים הבסיסיים – מה שעלול להוביל ל-overfit.

כעת נאמן את המודל שקיבלתי מניסוי 3 על כל סט האימון ונשערך על הtest.



יש overfitting מובהק

אולי המודל שלי למד ושינן יותר מידיי ב train ומה שהוא שינן

כן עבד בvalidation אבל על הtest המודל הגיע ללוס גבוה

יותר – overfit.

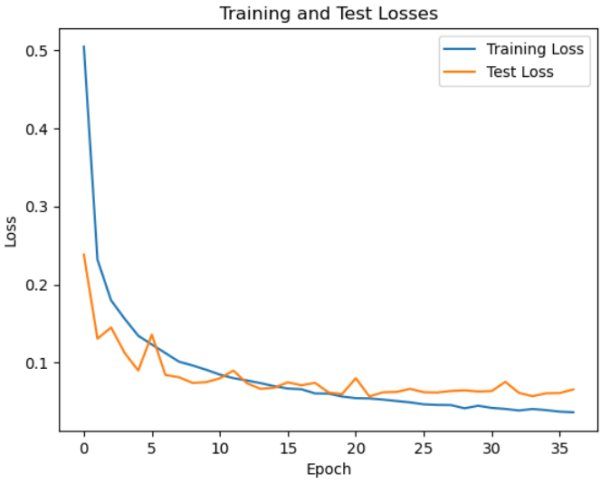
*יכול להיות שהשוני בהתפלגות בין הtrain ל-test שהראתי

בהתחלה גרם לזה.

בס"ד

אני אנסה כעת להוסיף את השיטות למניעת Overfitting שניסיתי קודם לכן.

בחרתי תחילה לנסות n= 15 , P = 0.3 שזה היה נראה לי הניסוי הכי מוצלח. קיבלתי את הגרף הבא:

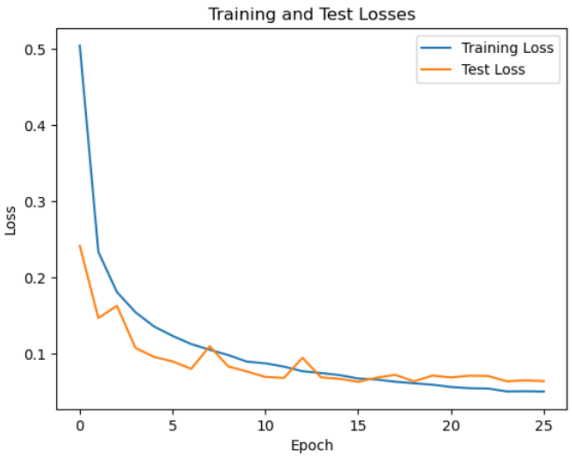


הdropout וה-early stopping עזרו מאוד במניעת

הoverfitting אבל עדיין ארצה לבדוק האם אני מצליחה

להוריד עוד יותר.

לכן הקטנתי את n ל-10 כי חשבתי שאולי אני נותנת יותר מידיי זמן להשתפר (הקטנת מספר האפוקים זו גם שיטה שלמדנו שעוזרת למנוע Overfitting). קיבלתי את הגרף הבא:



אכן יש שיפור והתוצאה אכן מספקת.

הצלחתי להגיע ל Test Accuracy = **0.9811**

הארכיטקטורה הסופית של הרשת שנבחרה:

כפי שהסברתי בהתחלה יש שכבת קלט שמשטחת את התמונה לטנזור חד מימדי.
לאחר מכן יש שתי שכבות חבויות עם 256 נוירונים כל אחת:

-השכבה הראשונה לוקחת את הטנזור משכבת הקלט שגודלו 784 ומטילה אותו לטנזור בגודל 256.
מפעילים עליו פונקציית אקטיבציה מסוג RelU.
כעת מופעלת שכבת dropout עם p = 0.3 כלומר 30% מהנוירונים מושתקים.
-השכבה השנייה לוקחת את הטנזור שיצא כפלט מהשכבה הקודמת (אחרי פונקציית האקטיבציה ושכבת הdropout) שהוא בגודל 256 ומטילה אותו לטנזור גם כן בגודל 256. ושוב מפעילים גם כאן את RelU ולאחר מכן יש עוד שכבת dropout שגם בה p = 0.3.

בס"ד

כעט הגענו לשכבה האחרונה – שכבת הפלט שלוקחת את הטנזור שיצא משכבת הdropout האחרונה ומטילה אותו לגודל 10 שזה באיזה כמות הcalss שיש לנו בבעיה – בעצם מקבלים נוירון עבור כל class.

הפלט של המודל הוא raw scores (logits) המיוצרים על ידי שכבת הפלט. ציונים אלה מועברים דרך פונקציית softmax במהלך ה inference כדי לקבל הסתברויות המייצגות את הסבירות של כל מחלקה.