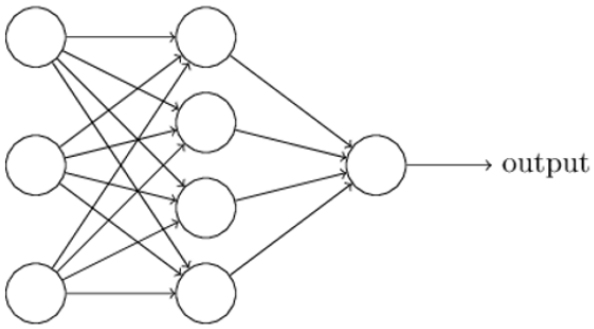
Multi-Object Optimization Course

Artificial Neural Network Optimization



<u>Authors:</u>

Stav Jacob

Yarden Turgeman

Alon Mizrahi

Alon Laron

תוכן עניינים

1 מבוא

רשת עצבית מלאכותית (ANN – Artificial Neural Network), רשת נוירונים או רשת קשרית הוא מודל מתמטי חישובי, שפותח בהשראת תהליכים מוחיים או קוגניטיביים המתרחשים ברשת עצבית טבעית ומשמש במסגרת למידת מכונה. רשת מסוג זה מכילה בדרך כלל מספר רב של יחידות מידע (קלט ופלט) המקושרות זו לזו, קשרים שלעיתים קרובות עוברים דרך יחידות מידע "חבויות" (Hidden Layer). צורת הקישור בין היחידות, המכילה מידע על חוזק הקשר, מדמה את אופן חיבור הנוירונים במוח. השימוש ברשתות עצביות מלאכותיות נפוץ בעיקר במדעים קוגניטיביים, ובמערכות תוכנה שונות - בהן: מערכות רבות של אינטליגנציה מלאכותית המבצעות משימות מגוונות - זיהוי תווים, זיהוי פנים, זיהוי כתב יד, חיזוי שוק ההון, מערכת זיהוי דיבור, זיהוי תמונה, ניתוח טקסט ועוד.

2 הגדרת הבעיה

משימת הקצה אותה אנו מבצעים על ידי שימוש ברשת עצבית (רשת נוירונים כפי שהוסברה קודם לכן, ועוד יורחב עליה בהמשך) היא משימת קלסיפיקציה - כלומר משימת סיווג. באופן ספציפי משימת הסיווג שתבוצע במסגרת פרויקט זה כוללת סיווג בין 5 מצבים שונים של כף יד אנושית (1-5 class) זאת כפי שמתואר באיור הבא.

איור 1: 5 מצבי היד השונים

חיזוי זה נעשה מבלי להתבסס על מידע וויזואלי, חשמלי, עצבי או כל סוג מידע אחר הנוגע לכף היד עצמה אלא מבוסס על סט רחב של חיישני כוחות הממוקמים באמה.

מערכת החיישנים אשר נעשה בה שימוש הינה מערכת מקורית, אשר תוכננה ונבנתה לשם ביצוע מחקר מקיף בנושא קלסיפיקציה של מצבי כף היד (רוב המערכת מודפסת במדספת תלת מימד).

איור 2: מערכת החיישנים עצמה והמערכת בתצורתה "המולבשת" על הנסיין.

כעת נגדיר באופן מפורט את פרמטרי הבעיה:

בעבור הרשת העצבית האותות המתקבלים ממערכת החישה שהוצגה קודם לכן יוגדרו מעתה כאותות INPUT. חמשת מצבי כף היד יוגדרו כ- OUTPUT של הרשת העצבית. את הרשת נעביר שלבי אימון על מנת שתוכל "ללמוד" את אופן החיזוי הנכון כאשר שלבים אלו יבוצעו בעזרת סט מדידות אחד. בנוסף לכך, הרשת תעבור מספר הרצות של בחינת אופן הדיוק שלה, כלומר בכמה אחוזים הרשת דייקה בחיזוי שלה, זאת יבוצע על ידי סט נתונים אחר.

<u>פרמטרי התכן-</u> שלושת פרמטרי התכן עליהם נשלוט בעת ביצוע האופטימיזציה הם מספר השכבות החבויות ברשת, גודלן של השכבות וכן גודל השכבה הראשונה .

<u>מרחב המטרות-</u> מרחב המטרות שלנו מורכב משתי מטרות שונות. המטרה הראשונה היא דיוק מרבי של הרשת. לפיכך, זוהי בעיית מקסימום שכן אנו מעוניינים לקבל דיוק גבוהה ככל האפשר (בשאיפה אופטימלית ל- 100% דיוק) של חיזוי מצבי היד השונים. $accuracy = \frac{\sum correct\ classifications}{\sum total\ classifications} \times 100\%$

חשוב לציין כי דיוק הרשת נבדק על סט בדיקה שלא בוצע עליו אימון.

המטרה השנייה היא גודל הרשת, כאשר נבצע הערכה של גודל הרשת בעזרת מדידת מספר הפרמטרים המצויים ברשת שלנו. גודל הרשת משפיע על מגוון רחב של פרמטרים כדוגמת זמן החישוב, מאמץ מיחשובי ועוד. גודל הרשת, שכאמור ישוערך על ידי מספר הפרמטרים, הוא פונקציה של שכבות הנוירונים וכן של גודל כל אחת מהשכבות:

$$size = func(number\ layers\ , layer\ size)$$

לפיכך, זוהי בעיית מינימום שכן אנו שואפים לבצע חישוב מדויק ככל האפשר, אך בלי לבזבז כח מחשוב גדול מדי ולאורך זמן ארוך מדי. כך שלבסוף מתקבלת עבורנו בעיית מקסימום - מינימום.

3 שיטת הפתרון

בחלק זה נתאר את הפונקציה שלנו ואופן ביצוע החישובים. היות ובמקרה זה לא מדובר בפונקציה אלגברית שניתנת לתיאור מתמטי מידי, אלא ברשת נוירונים המבצעת אימון וחיזוי (לטובת ביצוע קלסיפיקציה, כפי שתואר קודם) של מצבי כף יד, נציג קוד Python אשר מבצע הלכה למעשה את המשימה. כלל הקוד יצורף כנספח בסוף הדוח.

בראשית הקוד נייבא את כלל החבילות הדרושות ל- Python לשם ביצוע המטלות הדרושות.
השלב הבא הינו ייבוא של המידע. מידע זה נאסף קודם לכן על ידי מערכת החיישנים אשר מונחת על האמה ותוארה גם כן בפרק קודם.
לאחר מכן מתבצע עיבוד וקידוד של המידע למשתנים המתאימים ביותר על מנת לאפשר עבודה נוחה בשלבים הבאים.
לאחר מכן נגדיר את הרשת העצבית עצמה, אשר משמשת כפונקציית המטרה, כאשר בתוך פונקציה זו מתבצע אימון ובחינה של הרשת. הפונקציה מקבלת את משתני התכן ומחזירה את המטרות:

```
def obj_func(n_first_layer ,n_hidden ,s_hidden):
  set_seed(5)
  # define model
  model = Sequential()
  model.add(Dense(n_first_layer, input_dim=28, activation='relu', input_shape=(28,)))
  for i in range(n_hidden):
    model.add(Dense(s_hidden, activation='relu'))
  model.add(Dense(5, activation='softmax'))
  # model.summary()
  trainableParams = np.sum([np.prod(v.get_shape()) for v in model.trainable_weights])
  nonTrainableParams = np.sum([np.prod(v.get_shape()) for v in model.non_trainable_weights])
  totalParams = trainableParams + nonTrainableParams
  # compile the model
  model.compile(optimizer='adam',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
  # fit the model
  model.fit(X_train, y_train, epochs=50, batch_size=20, verbose=0)
  # evaluate the model
  loss, acc = model.evaluate(X_test, y_test, verbose=0)
  return(totalParams,round(acc*100,2))
```

בקטע המוצג לעיל ניתן לראות את אופן הגדרת הרשת (בהתאם למספר השכבות, גודל השכבות, וגודלה של השכבה הראשונה), ניתן לראות שפונקציית האקטיבציה הנבחרת עבור מרבית השכבות הינה "relu" ואילו עבור השכבה האחרונה מוגדרת פונקציית האקטיבציה "softmax" אשר דרושה בשכבה האחרונה היות וישנו צורך לספק קלסיפיקציה בין 5 מצבים סופיים.
לאחר מכן ניתן לראות את הגדרת הפרמטרים המאומנים ואלו שאינם מאומנים, כאשר לאחר שלב זה מתבצע שלב האימון של הרשת בהתאם למידע המסופק, ניתן לראות שמספר ה epochs (כלומר מחזורי האימון) הוא קבוע בשלב זה על ערך של 50. בנוסף הפרמטר batch_size המתאר את כמות דוגמאות האימון שבהן יעשה שימוש בכל איטרציה גם הוא מקובע על ערך של 20.
לבסוף ניתן לראות שמבוצע חישוב של הדיוק המתקבל בסוף שלב האימון וזהו גם הפלט של פונקציה זו.

4 האלגוריתמים שנבחרו

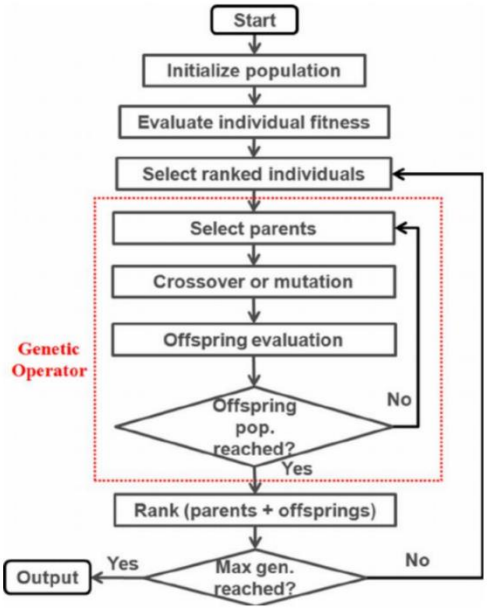
4.1 הסבר לבחירת האלגוריתמים

NSGA II אלגוריתם זה נבחר בשל הפופולריות שלו ויעילותו הידועה בפתרון בעיות אופטימיזציה בעלות שתי מטרות. בנוסף, על פי מאמר [1] אלגוריתם זה יודע להתמודד עם בעיות בדיקה מורכבות, מסוגל למצוא פיזור טוב יותר של פתרונות והתכנסות טובה יותר לחזית האמיתית של פארטו-אופטימלית.

MOEAD אלגוריתם אשר נפוץ גם כן בפתרון בעיות עם יותר ממשתנה מטרה אחד. בנוסף, על פי מאמר [2], נמצא כי אלגוריתם זה בעל ביצועים טובים יותר או דומים ל- NSGA II וברצוננו לאמת זאת עבור הבעיה שלנו. נציין כי האלגוריתם שנבחר לנו על ידי המרצה לא קיים במאגר pymo, לכן זה היה שיקול נוסף בבחירת האלגוריתם.

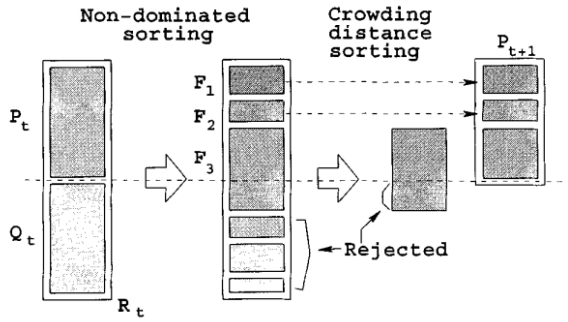
4.2 NSGA II

האלגוריתם NSGA II הינו אחד האלגוריתמים הנפוצים ביותר ממשפחת אלגוריתמים אבולוציוניים רב-אובייקטיביים. אלגוריתם זה מציג גישה של מיון מהיר עבור הפתרונות הבלתי נשלטים. כמו כן, קיים אופרטור גנטי היוצר מאגר זיווג על ידי שילוב אוכלוסיות ההורים והצאצאים ובחירת הפתרונות הטובים ביותר.



איור 3 : תרשים זרימה של אלגוריתם **NSGA II**

כפי שמופיע באיור 1, תחילה יש לבצע אתחול של אוכלוסיית ההורים והצאצאים. אוכלוסייה זו נבחרת בצורה אקראית ועבור הקבוצה המאוחדת מחשבים את משתני האופטימיזציה בכדי לבצע מיון של הבלתי נשלטים. בשלב הבא מבצעים דירוג לפי שיטת המחסנית (Tournament Selection-TS), כלומר, ראשית נכנס סט הפתרונות עם הדירוג הגבוה ביותר 1F. במידה וקבוצה זו קטנה מ-N (גודל האוכלוסייה) נכניס את הסט הבא בדירוג 2F, כפי שמופיע באיור 2, בדרך זו האליטיזם מובטח.



איור 4 : תהליך ה- **NSGA II**

לאחר מכן, האוכלוסייה עוברת תהליך אבולוציוני של Crossover&Mutation ליצירת אוכלוסייה חדשה בגודל N. הפתרונות שיבחרו עבור האוכלוסייה החדשה יקבעו לפי שיטת Crowding Distance (CD). הרעיון בשיטה זו הוא לפלג באופן אחיד כמה שיותר פתרונות ולתת עדיפות לפתרונות רחוקים יותר שנותנים יותר מידע. ככל שערך מדד זה גדול יותר כך הוא טוב יותר. פתרון i בשיטת CD יוביל על פתרון j בשיטת TS אם אחד מהתנאים הבאים מתקיים :

1. אם לפתרון i יש דירוג גבוה יותר.
2. אם לשני הפתרונות יש את אותו הדירוג, אך לפתרון i יש CD גבוה יותר מאשר לפתרון j.

במידה והתהליך אינו נפגש בתנאי עצירה, תהליך זה חוזר על עצמו עד שיתקבלו פתרונות מספקים שיפיקו את החזית האופטימלית.

4.3 MOEA/D

האלגוריתם MOEA/D הינו אלגוריתם אבולוציוני רב-אובייקטי המבוסס על דקומפוזיציה (פירוק). האלגוריתם הוצע לראשונה על ידי Zhang and Li בשנת 2007 ומאז צבר תאוצה ופותחו תתי אלגוריתמים נוספים על בסיסו. האלגוריתם מפרק בעיית אופטימיזציה רב-אובייקטית למספר תת-בעיות אופטימיזציה סקלריות ובו-זמנית מייעל אותן. כל תת-בעיית עוברת אופטימיזציה על ידי שימוש במידע ממספר בעיות המשנה השכנות, מה שגורם ל-MOEA/D להיות בעל מורכבות חישובית נמוכה יותר בכל דור מאשר MOGLS ו- NSGA II. יתרון נוסף הוא שניתן לשלוט על כמות תת הבעיות שהאלגוריתם פותר, כלומר ניתן לשלוט ברזולוציה של החזית או לחלופין להתמקד באזור מסוים בחזית.

The algorithm works as follows:

Input:
- MOP (1);
- a stopping criterion;
- N: the number of the subproblems considered in MOEA/D;
- a uniform spread of N weight vectors: $\lambda^1, \ldots, \lambda^N$;
- T: the number of the weight vectors in the neighborhood of each weight vector.

Output: EP.

Step 1) Initialization:

Step 1.1) Set EP $= \emptyset$.

Step 1.2) Compute the Euclidean distances between any two weight vectors and then work out the T closest weight vectors to each weight vector. For each $i = 1, \ldots, N$, set $B(i) = \{i_1, \ldots, i_T\}$, where $\lambda^{i_1}, \ldots, \lambda^{i_T}$ are the T closest weight vectors to λ^i.

Step 1.3) Generate an initial population $x^1, \ldots, x^N$ randomly or by a problem-specific method. Set $FV^i = F(x^i)$.

Step 1.4) Initialize $z = (z_1, \ldots, z_m)^T$ by a problem-specific method.

Step 2) Update:

For $i = 1, \ldots, N$, do

Step 2.1) Reproduction: Randomly select two indexes k, l from $B(i)$, and then generate a new solution y from x^k and x^l by using genetic operators.

Step 2.2) Improvement: Apply a problem-specific repair/improvement heuristic on y to produce y'.

Step 2.3) Update of z: For each $j = 1, \ldots, m$, if $z_j < f_j(y')$, then set $z_j = f_j(y')$.

Step 2.4) Update of Neighboring Solutions: For each index $j \in B(i)$, if $g^{te}(y'|\lambda^j, z) \leq g^{te}(x^j|\lambda^j, z)$, then set $x^j = y'$ and $FV^j = F(y')$.

Step 2.5) Update of EP:

Remove from EP all the vectors dominated by $F(y')$.

Add $F(y')$ to EP if no vectors in EP dominate $F(y')$.

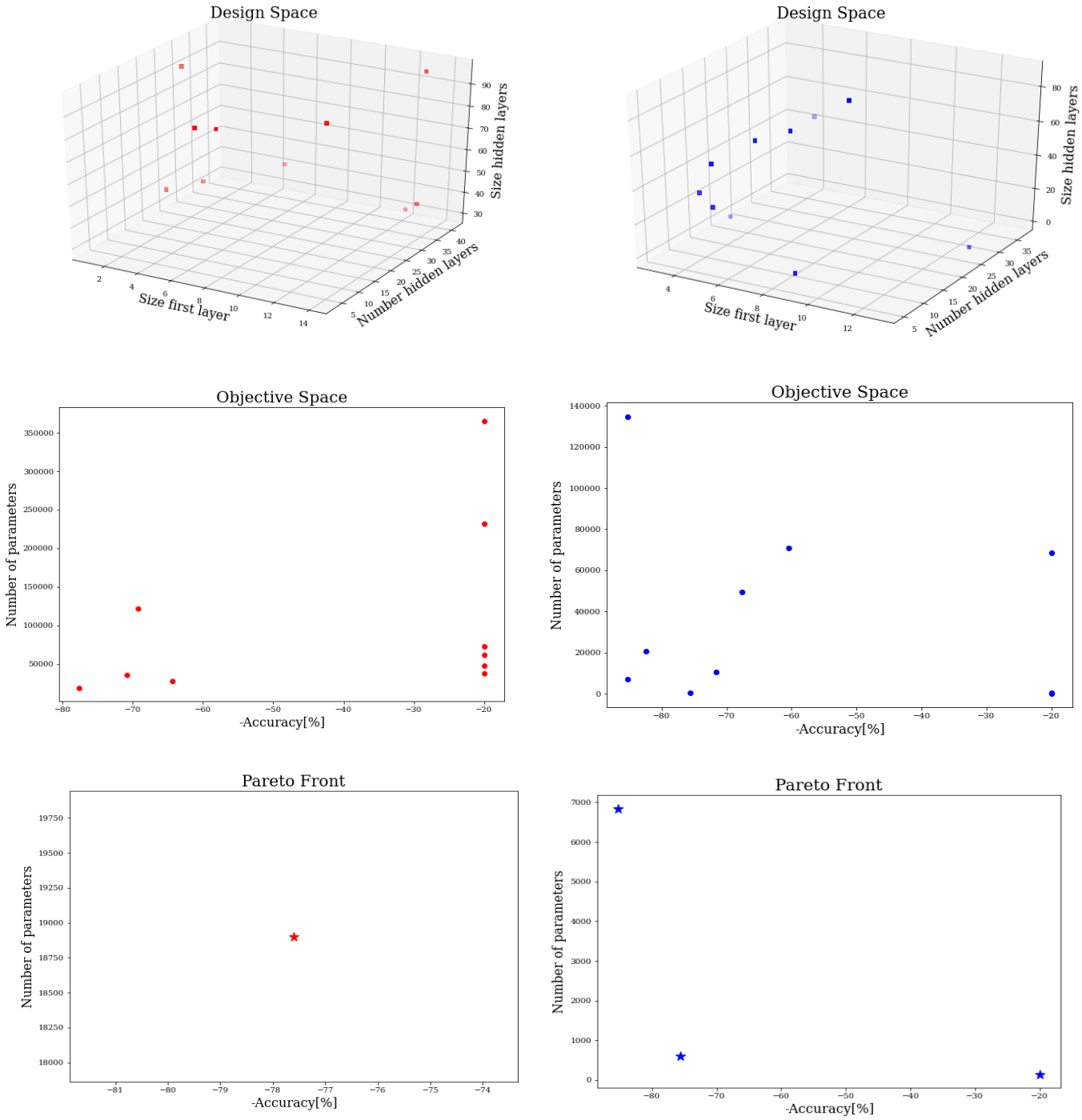
Step 3) Stopping Criteria: If stopping criteria is satisfied, then stop and output EP. Otherwise, go to **Step 2**.

איור 5 : תיאור השלבים ב- **MOEA/D**

באלגוריתם זה תחילה נדרש לאתחל קבוצה חיצונית אליה יוכנסו אוכלוסיית הפתרונות הבלתי נשלטים בכל דור. בסיום התהליך נקבל פלט של קבוצה זו ממנה נדרש לבחור את הפתרון האידיאלי עבורנו. לאחר מכן, האלגוריתם מחשב את המרחק בין כל שני וקטורי משקל ואז מחשב את וקטורי המשקל ה-T (כמות השכנים עבור תת קבוצה i) הקרובים ביותר לכל וקטור משקל. בשלב הבא מיוצרת אוכלוסייה ראשונית בגודל N $\{x^1, \ldots, x^N\}$ אשר מייצגת את תת בעיות האופטימיזציה הסקלריות. תהליך זה מבוצע באופן רנדומלי. לאחר מכן מתבצעת הערכה לפונקציות האופטימיזציה $\{FV^1, \ldots, FV^N\}$ כאשר $FV^i = F(x^i)$. בשלב הבא מבצעים אתחול ל- $z = \{z_1, \ldots, z_m\}$ כאשר z_i מייצג את הערך הטוב ביותר (הנקודה האידיאלית) שנמצא עד כה עבור אובייקט f_i. בשלב הבא נכנס למשחק החלק האבולוציוני. עבור תת קבוצה i בוחרים באופן רנדומלי שני אינדקסים k ו-l מהקבוצה B(i) ואז מייצרים פתרון חדש y מתתי הקבוצות x^k ו- x^l על ידי שימוש באופרטור הגנטי. לאחר מכן משפרים את היוריסטיקה על y ליצירת y' ואז בודקים האם נקודת הרפרנס שלנו z_j קטנה מערך $f_j(y')$. אם כן, מאתחלים את ערך נקודת הרפרנס לערך החדש. העדכון מתבצע גם על פתרונות השכנים, והפתרונות הבלתי נשלטים נכנסים לקבוצת ה-EP. בשלב זה במידה וקיימים בקבוצת ה-EP פתרונות שנשלטים על ידי $F(y')$ הם מוסרים מהקבוצה. התהליך חוזר חלילה עד שפוגש בקריטריון העצירה ומוציא פלט של קבוצת ה-EP.

5 הדגמת הרצה עבור דור בודד

להלן דוגמת הרצה עבור דור בודד המכיל 10 פתרונות עבור האלגוריתמים הנבחרים :



איור 6 : דוגמת הרצה עבור דור בודד- **NSGAII** מימין בכחול, **MOEAD** משמאל באדום

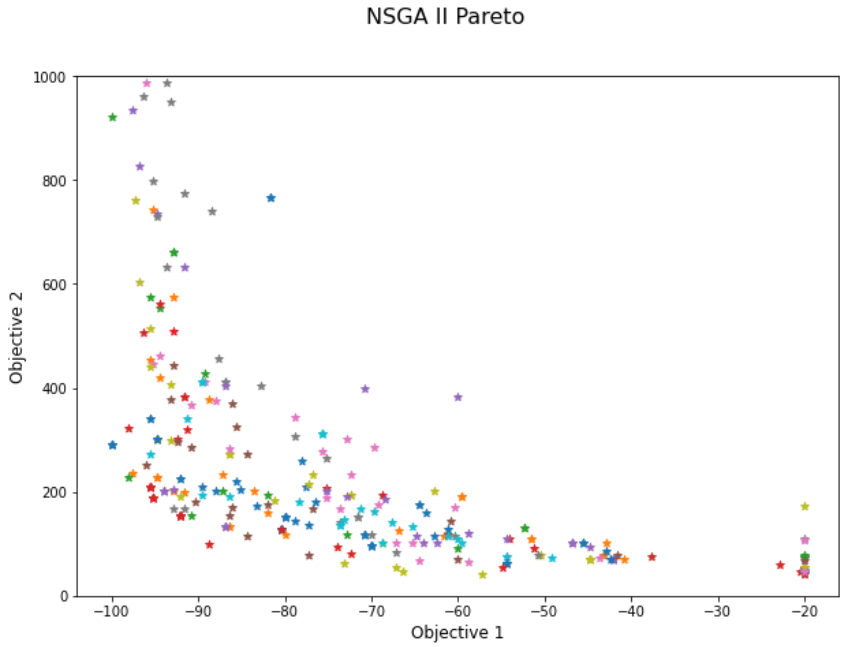
מההשוואה עבור דור בודד ניתן לראות כי אלגוריתם NSGA II הפיק מספר גדול יותר של פתרונות בלתי נשלטים לעומת אלגוריתם MOEAD. בנוסף לכך, הפתרונות בעלי פיזור טוב כך שקיימים שני פתרונות המקיימים מינימום בקצוות בכל אחד ממשתני האופטימיזציה ופתרון נוסף המקיים מינימום בשני המשתנים יחד ומציג תוצאה טובה מאוד מהצופה, בהתחשב בכך שמדובר בדור בודד.

6 הצגת התוצאות וניתוח סטטיסטי

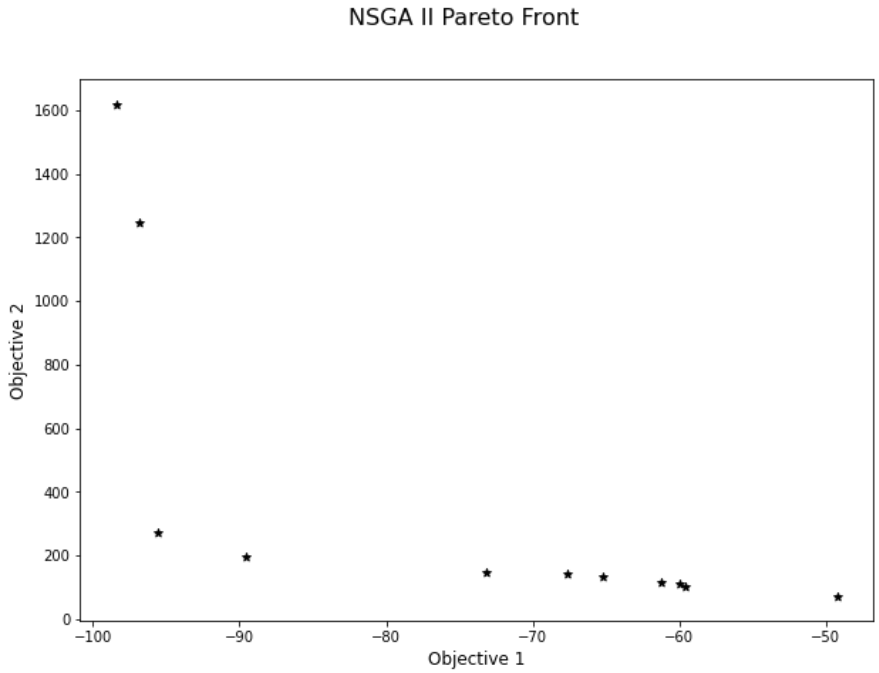
6.1 הצגת התוצאות

בחלק זה מוצגות תוצאות עבור 30 הרצות לכל אלגוריתם, כל הרצה היא בעלת 10 דורות עם אוכלוסייה התחלתית של 30 פריטים.

תוצאות עבור 30 הרצות באלגוריתם NSGA II :

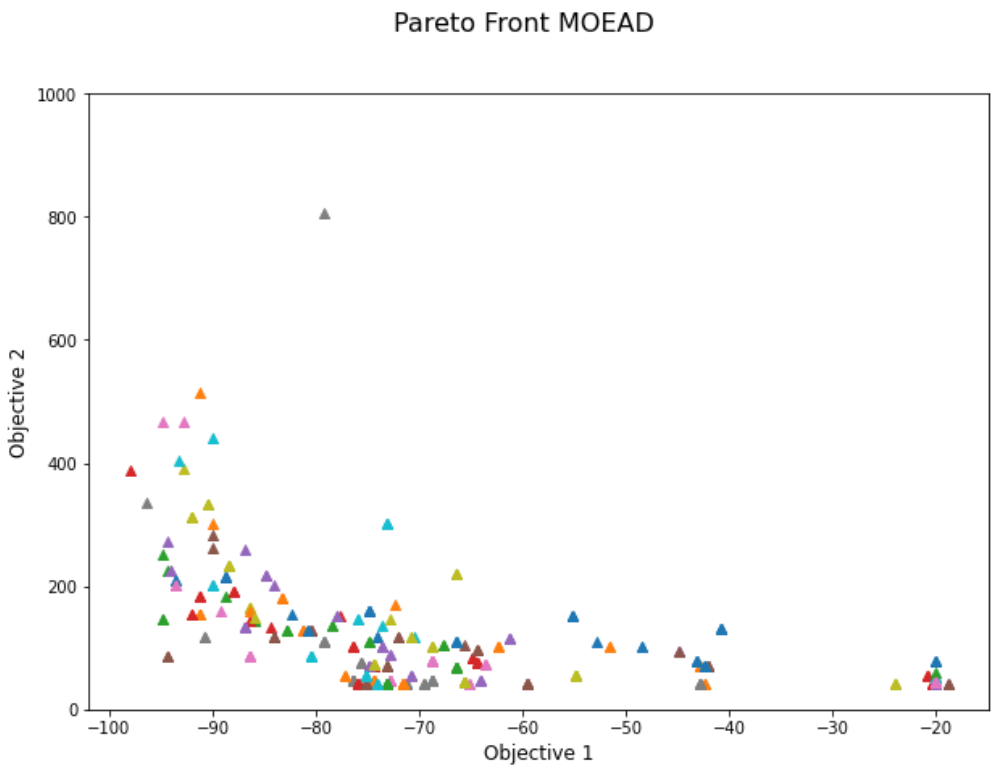


איור 7 : חזית האיחוד עבור 30 הרצות NSGA II

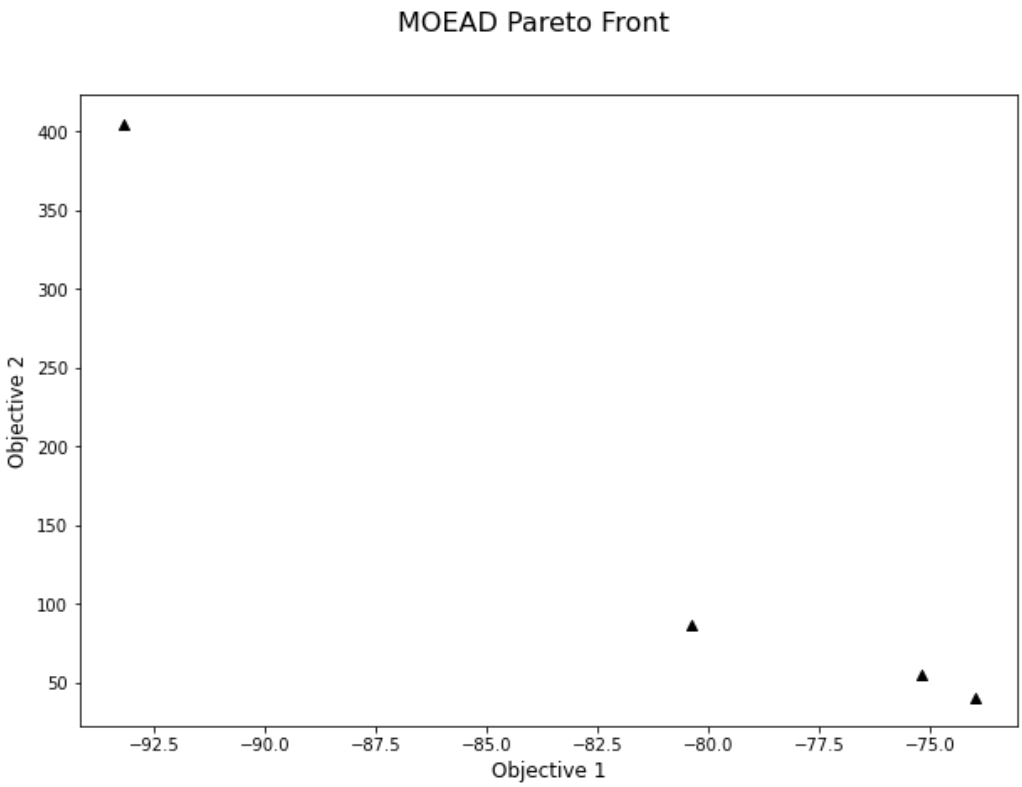


איור 8 : חזית פרטו עבור 30 הרצות NSGA II

תוצאות עבור 30 הרצות באלגוריתם MOEAD:



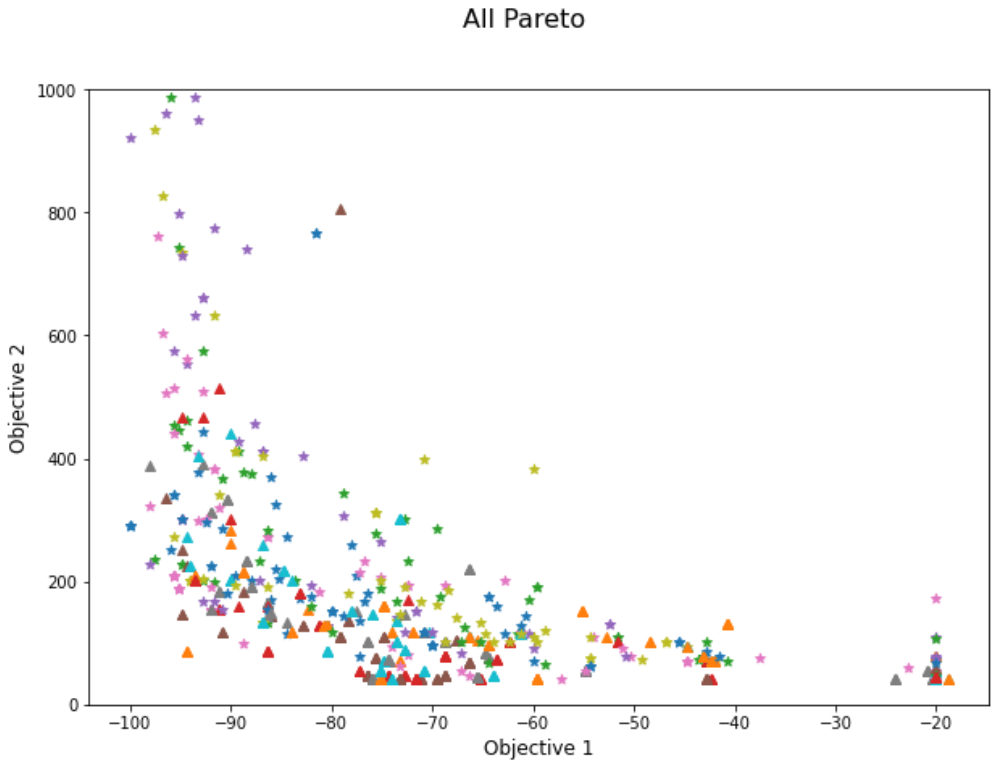
איור 9: חזית האיחוד עבור 30 הרצות MOEAD



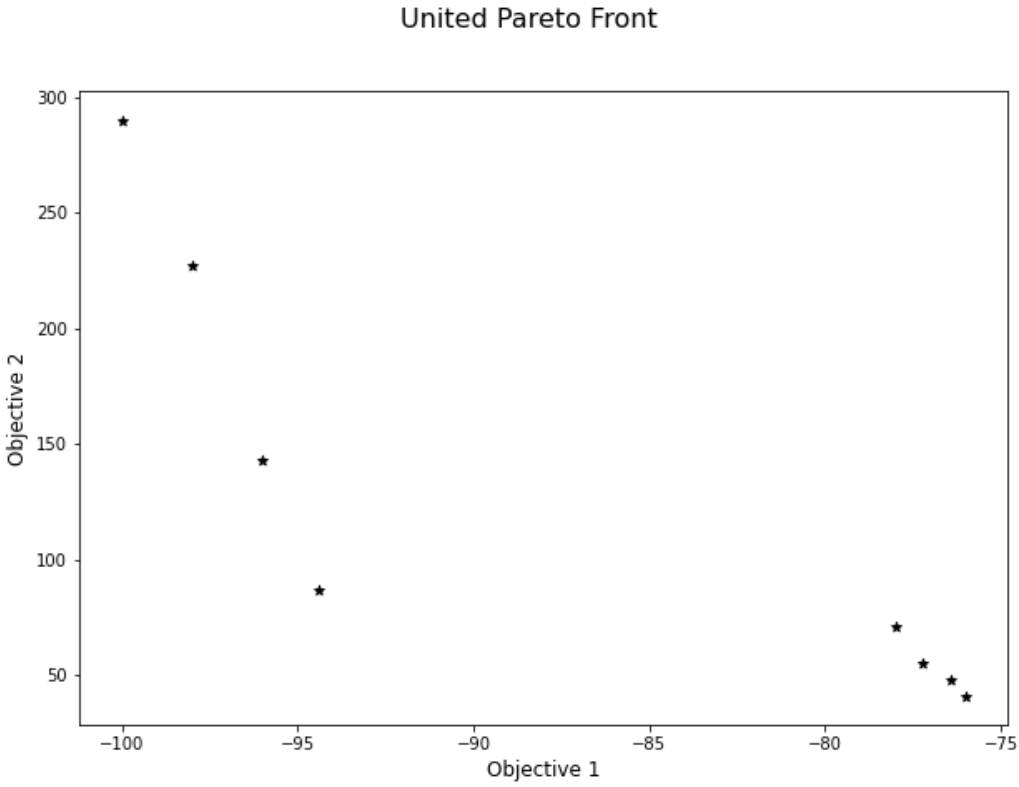
איור 10: חזית פרטו עבור 30 הרצות MOEAD

ניתן לראות כי באיור 8 יש 11 פתרונות בחזית בעוד שבאיור 10 יש 4 פתרונות בחזית.

חזית פרטו המאוחדת עבור כל 60 ההרצות עם 2 האלגוריתמים :



איור 11 : חזית האיחוד עבור 62 הרצות כוכב- NSGA II, משולש - MOEAD



איור 12 : חזית פרטו עבור 60 הרצות

באיור 11 ניתן לראות את האיחוד החזיתות של 2 האלגוריתמים שנבדקו עבור 30 דורות בנוסף ל-2 הרצות ארוכות של 30 דורות, כאשר לכל אלגוריתם סימון שונה שניתן יהיה להבדיל ביניהם, ניתן לראות כי מתקבל פיזור דומה בין 2 האלגוריתמים שנבדקו.

איור 12 היא חזית הרפרנס איתה נעשה שימוש לחישוב המדדים בהמשך. כאשר בוחנים את החזיתות שהתקבלו באיור 8 - 10 ניתן להבחין כי מתקבלות יותר נקודות בחזית עבור האלגוריתם NSGA II לעומת MOEAD.

6.2 ניתוח סטטיסטי

עבור הניתוח הסטטיסטי נעשה שימוש במדדים לבחינת טיב התוצאות והחזיתות שהתקבלו, בכדי לבצע את הניתוח יש צורך בחזית רפרנס, במקרה שלנו חזית הרפרנס היא החזית שהתקבלה כתוצאה מאיחוד 2 הרצות ארוכות של 30 דורות לכל הרצה.

המדדים שנעשה בהם שימוש:

- IGD – Inverted Generational Distance:

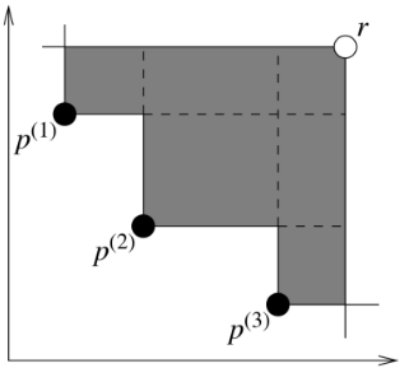
מדד זה מחשב מרחק אוקלידי בין כל נקודה Z לנקודה הקרובה ביותר A, כאשר Z היא נקודה על חזית הרפרנס וA היא נקודה אותה אנו רוצים לבחון על החזית שהתקבלה. מדד זה הוא מדד מובנה בספריית Pymoo והנוסחה לחישוב מדד זה היא:

$$IGD(A) = \frac{1}{|Z|}\left(\sum_{i=1}^{|Z|} \hat{d}_i^{\,p}\right)^{1/p}$$

כלל שערכו של מדד זה קטן יותר כך יש התאמה טובה יותר לחזית פרטו ולכן נרצה ערכים מינימליים של מדד זה.

- HV – HyperVolume:

מדד זה מחשב את השטח בין נקודת ייחוס (r) שאנו קובעים לפתרונות המתקבלים(p), ראה איור 13, מדד זה הוא מדד מובנה בספריית Pymoo גם כן.



איור 13: תיאור חישוב מדד HV

עבור מדד זה נרצה לקבל שטח כמה שיותר גדול, כלומר ככל שהשטח גדול יותר הפתרונות הנבדקים ישלטו על הפתרונות האחרים ולכן נרצה ערכים מקסימליים במדד זה.

- Spread:

מדד זה מחשב את הפיזור בין הפתרונות בחזית ונותן אינדיקציה על הפיזור שלהם. מדד זה אינו פונקציית מובנית בספריית Pymoo ולכן כתבנו פונקציה שמקבלת 2 נקודות ומחשבת את המרחק ביניהן ומבצעת מיצוע על המרחקים. במדד זה נרצה ערכים גדולים מכיוון שאלו מצביעים על פיזור בין הנקודות על החזית.

התוצאות שהתקבלו:

טבלה 1 - מדדים עבור 30 הרצות

NSGA II					MOEAD				
index	**IGD**	**HV**	**SPREAD**	**TIME [min]**	**index**	**IGD**	**HV**	**SPREAD**	**TIME [min]**
0	25.2	68894.4	89.7	41.8	**0**	399.4	0.0	673.0	49.5
1	27.9	68035.2	75.8	38.4	**1**	37.3	63632.4	53.5	30.6
2	1980.7	61281.6	3919.8	49.2	**2**	26.8	67196.8	65.8	29.7
3	23.3	67408.8	52.4	38.4	**3**	32.2	68598.0	69.2	31.0
4	33.1	65670.8	61.7	47.5	**4**	33.1	65280.4	61.1	33.8
5	313.3	59876.8	676.7	46.0	**5**	33.9	61592.4	55.4	32.5
6	472.2	61060.4	914.2	36.9	**6**	14.0	66410.0	55.9	21.6
7	1322.3	56896.4	2258.3	31.6	**7**	9.7	70281.6	57.9	21.9
8	167.1	58065.2	276.6	30.5	**8**	20.8	55474.4	46.7	26.7
9	842.0	57913.2	890.4	29.4	**9**	36.4	61692.4	61.5	20.3
10	1925.4	51786.8	3277.4	34.1	**10**	39.2	65041.6	59.9	22.6
11	8064.0	58815.2	16027.5	37.2	**11**	2395.4	63735.2	4848.5	25.4
12	222.9	62432.0	486.7	31.3	**12**	8.3	69555.2	43.2	25.3
13	170.4	62127.2	389.0	34.7	**13**	14.4	64465.6	42.1	22.1
14	1207.6	53032.0	2266.3	34.8	**14**	11.8	65838.8	98.8	25.5
15	31.8	66853.2	87.0	25.4	**15**	1611.0	64559.6	3251.8	21.5
16	13515.8	53826.0	23786.6	34.6	**16**	22.6	66206.8	73.8	24.0
17	3477.8	60369.2	4652.9	29.9	**17**	26.4	54586.0	76.2	29.9
18	27.6	63909.6	122.0	26.2	**18**	32.0	64237.2	69.6	18.1
19	58.5	60958.4	163.2	25.6	**19**	216.6	46578.0	486.7	20.2
20	24.0	66868.4	96.1	20.3	**20**	35.4	57749.2	38.0	19.6
21	460.9	64375.2	918.9	36.4	**21**	21.4	64643.2	77.2	24.9
22	582.9	63118.8	1172.4	34.0	**22**	16133.2	0.0	32440.1	48.0
23	17925.6	66990.4	35899.3	32.0	**23**	7.9	63412.0	52.8	20.6
24	4064.5	63499.2	8031.8	29.4	**24**	70.9	59906.4	183.9	20.9
25	129.7	65266.0	347.1	30.9	**25**	0.6	70466.4	17.1	23.0
26	43.4	63000.4	159.5	32.3	**26**	41.2	40417.2	26.2	36.7
27	1062.0	65630.4	2009.7	31.6	**27**	20.6	55873.2	36.0	31.8
28	15498.1	68143.6	31077.1	32.1	**28**	13.1	64614.8	77.4	30.2
29	233.7	65705.6	473.8	30.3	**29**	13.0	65283.6	54.5	20.5
average	474.5	62506.9	882.7	34.1	average	45.9	59964.2	100.5	26.4
median	196.7	63059.6	431.4	33.1	median	26.4	64465.6	59.9	24.9
std	589.2	4411.4	1058.7	6.9	std	79.5	13572.4	141.0	6.7

בטבלה 1 ניתן לראות את הניתוח הסטטיסטי שהתקבל, כאשר פתרונות שחרגו ביותר מ-5 סטיות תקן לפחות באחד המדדים נחשב כ- "outlier" והוצא מהחישוב.

טבלה 2 : השוואת מדדים סטטיסטיים בין האלגוריתמים

	IGD [%]	HV [%]	SPREAD [%]	TIME [%]
average	90.3	4.1	88.6	22.4
median	86.6	-2.2	86.1	24.8
std	86.5	-207.7	86.7	2.3

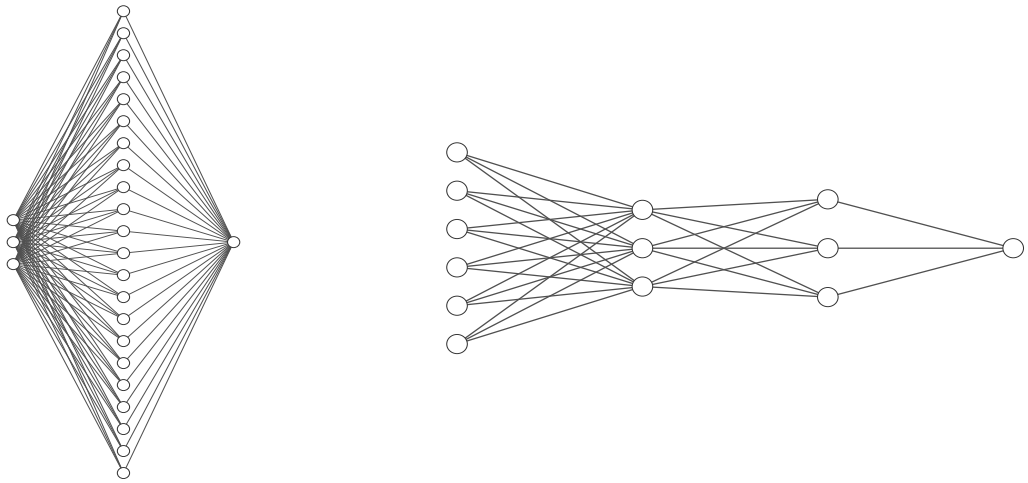
בטבלה 2 יש השוואה בין המדדים הסטטיסטיים עבור שני האלגוריתמים, כאשר הערכים המוצגים בטבלה הם השינוי מהמדדים שהתקבלו באלגוריתם NSGA II למדדים שהתקבלו בMOEAD, כאשר אחוז חיובי מעיד על שיפור במדד לטובת MOEAD ואחוז שלילי מעיד על שיפור במדד לטובת NSGA II.
וניתן לראות בבירור שיפור בכל המדדים למעט HV לטובת MOEAD, כלומר במדדי האיכות בהם השתמשנו להעריך את הפתרונות רוב הפתרונות שהתקבלו מאלגוריתם MOEAD טובים יותר.
יש לציין כי יש לשקול את מדדים אלו מכיוון שכמות הפתרונות אשר התקבלו בחזית פרטו בין השני האלגוריתמים היא גדולה כפי שצויין קודם.
עבור הזמן ניתן לראות כי משתנה זה אחיד באופן יחסי בין שני האלגוריתמים ועם סטיית תקן קטנה.

6.3 הצגת פתרונות אופניים

בחלק זה נציג פתרונות אופניים מחזית פרטו המאוחדת, נזכיר כי מטרת האופטימיזציה הייתה למצוא את הארכיטקטורה המינימלית של הרשת עם ביצועי הדיוק הגבוהים ביותר, כאשר הדיוק הוא על מקרי ה-test ולא ה-train, כך שמדד זה מציג באופן נאמן את הדיוק של הרשת.

טבלה 3 : תוצאות הפתרונות האופייניים

Solution number	Accuracy	Neural network size
1	100%	290
2	98%	227
3	94.4%	89
4	77.2%	55



איור 14 : פתרונות אופייניים, משמאל פתרון 1, מימין פתרון 2

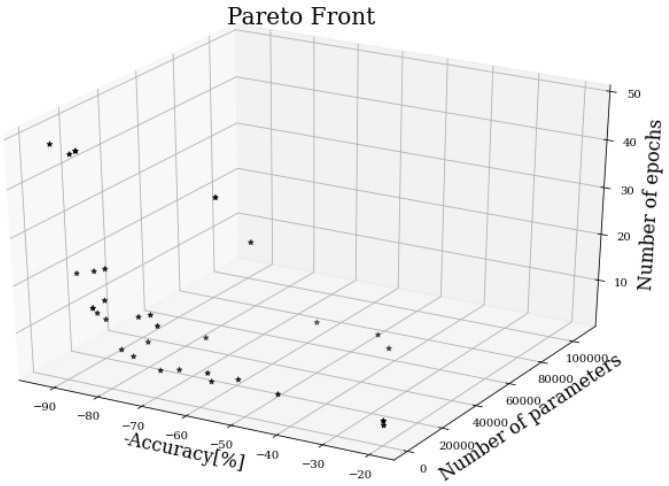
איור 15 : פתרונות אופייניים, משמאל פתרון 3, מימין פתרון 4

ניתן לראות באיור 14 ו15 חלק מהפתרונות האופייניים של תהליך האופטימיזציה, כפי שניתן היה לצפות ככל שגדול הרשת גדל כך יגדל גם הדיוק וכוח החישוב הנדרש.

6.4 בעיה תלת ממדית (בונוס)

בחלק זה הוספנו פרמטר תכן אשר משמש גם כמטרה, על מנת להפוך את הבעיה לתלת מימדית. פרמטר זה הינו מספר epochs, כלומר מספר מחזורי האימון שהרשת מבצעת, כאשר השאיפה היא לצמצם את מספר מחזורי האימון בכדי לצרוך פחות כוח חישובי באימון הרשת וגם להקטין את זמן אימון הרשת.

לאופטימיזציה זו עשינו שימוש באלגוריתם NSGA II כפי שנלמד בכיתה, עם גודל אוכלוסייה של 60 פריטים ולאורך של 20 דורות.



איור 16 : חזית פרטו תלת מימד

באיור 16 ניתן לראות את חזית פרטו עם 3 מטרות, כאשר ציר Z הוא הציר שהתווסף לאופטימיזציה. נשים לב כי יש פריסה של פתרונות לאורך ציר זה ויש לו משמעות, כמו כן, פריסת הפתרונות לאורך מטרה זו הוא יותר משמעותי מפריסת הפתרונות לאורך הציר שמייצג את המטרה השנייה באופטימיזציה (מספר הפרמטרים ברשת), וניתן לומר כי עבור אופטימיזציה זו מטרת המיניזציה של Number of Epochs היא בעלת משקל גדול יותר מהמטרה של Number of Parameters.

ניתן לראות בנספח – הקוד המצורף, עוד פלוטים עבור הבעיה התלת מימדית.

7 סיכום ומסקנות

במסגרת פרויקט זה בוצעה אופטימיזציה לרשת נוירונים שמטרתה חיזוי וקלסיפיקציה בין חמישה מצבים שונים. ייעדי האופטימיזציה, כלומר מרחב המטרות, כללו דיוק מקסימלי לצד גודל רשת מינימאלי. על מנת לבצע את האופטימיזציה, הוגדרו שלושה פרמטרי תכן הכוללים את מספר השכבות החבויות ברשת, גודלן של השכבות וכן גודל השכבה הראשונה. האלגוריתמים שנבחרו לביצוע האופטימיזציה הם NSGA II ו-MOEAD.

במסגרת ביצוע הפרויקט עלו מספר מסקנות חשובות:

- ניתן לקבוע שבעבור שני האלגוריתמים התקבלו תוצאות טובות, ובפרט התקבלו חזיתות פרטו איכותיות המציגות פתרונות מדויקים (Accuracy) ובגדלי רשת קטנים יחסית.
- מהסתכלות על קצב התקדמות הפתרונות במרחב המטרות ניתן לומר כי הן ב-2D והן ב-3D מגיעים להתכנסויות מהירות לחזית פרטו אופטימלית.
- האלגוריתם NSGA II הפיק מספר גדול יותר של פתרונות בלתי נשלטים בהשוואה ל- MOEA/D.
- מבחינת המדדים הסטטיסטיים ניתן ללמוד שאלגוריתם MOEAD הציג ביצועים טובים יותר.
- למרות ההמדדים הסטטיסטים שנתו לטובת MOEAD , ניתן לראות כי יש חשיבות רבה להסתכלות ויזואלית על החזית המתקבלת.
- בסעיף הבונוס ניתן לראות שמספר ה epochs הוא פרמטר משמעותי שיכול לסייע באופטימיזציה של רשתות.
- ניתן לראות שכמה מהפתרונות האופייניים שונים זה מזה משמעותית, לכן נדרשת מערכת קבלת החלטות נוספת על מנת לבצע בחירת פתרון ספציפי.

בשל הביצועים האיכותיים שהוצגו בדוח זה ובשל העובדה שהמאמץ המחשובי הדרוש לשימוש באלגוריתמים אלו אינו גדול, שימוש בהם (או באלגוריתמים שונים) בעת עבודה עם רשתות עצביות עשויה לסייע בייעול מבנה הרשת ושיפור הביצועים שלה. בשל כך, ובשל העובדה שייעדי הפרויקט הפרטניים הושגו אנו סבורים שמטרתו הכוללת הושגה.

8 מקורות

1. Matplotlib - Python library for data visualization. (n.d.). Retrieved from https://matplotlib.org/
2. NumPy - Mathematical Python library. (n.d.). Retrieved from https://numpy.org/
3. PYMOO - Python library for Multi Objective Optimization. (n.d.). Retrieved from https://pymoo.org/index.html
4. Scikit-learn - python library for ML&AI Programing. (n.d.). Retrieved from https://scikit-learn.org/stable/
5. Q. Zhang and H. Li, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," in *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712-731, Dec. 2007, doi: 10.1109/TEVC.2007.892759.

Optimization Project

Import libraries

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy
from scipy import stats
import pickle

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

from google.colab import data_table # Enables rendering of pandas dataframes into interactive displays
data_table.enable_dataframe_formatter()

from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import initializers
from tensorflow.random import set_seed
```

In [ ]:

```
# !pip install -U pymoo
```

In [ ]:

```
from pymoo.algorithms.moo.nsga2 import NSGA2
from pymoo.problems import get_problem
from pymoo.optimize import minimize
from pymoo.visualization.scatter import Scatter
from pymoo.core.problem import Problem


from pymoo.core.problem import ElementwiseProblem
from pymoo.core.variable import Real, Integer, Choice, Binary

from pymoo.operators.sampling.rnd import IntegerRandomSampling

from pymoo.util.display.column import Column
from pymoo.util.display.output import Output
from pymoo.core.callback import Callback

from pymoo.algorithms.moo.moead import MOEAD
from pymoo.util.ref_dirs import get_reference_directions

from pymoo.indicators.igd import IGD
from pymoo.indicators.hv import HV
```

Load Data

In [ ]:

```
from IPython.display import Image

Image(filename = f'WhatsApp Image 2022-12-11 at 15.35.55.jpeg', width = 1000, height = 300)
```

Out[1]:

In [ ]:

```
# Load the dataset
train_path = "train_data_norm.csv"
test_path = "test_data_norm.csv"
train_df = read_csv(train_path)
test_df = read_csv(test_path)
```

In [ ]:

```
train_df.iloc[:,1:].head()
```

```
Warning: Total number of columns (29) exceeds max_columns (20). Falling back to pandas display.
```

Out[12]:

	0	1	2	3	4	5	6	7	8	9	...	19	20	21	2
0	0.777778	0.921875	0.654206	0.565517	0.594595	0.802083	0.847162	0.377358	0.446429	0.195719	...	0.660920	0.615819	0.255319	0.4961
1	0.777778	0.927083	0.682243	0.565517	0.574324	0.802083	0.851528	0.377358	0.452381	0.195719	...	0.678161	0.638418	0.239362	0.4847
2	0.777778	0.932292	0.691589	0.572414	0.574324	0.802083	0.851528	0.381132	0.446429	0.198777	...	0.683908	0.638418	0.244681	0.4885
3	0.777778	0.932292	0.682243	0.572414	0.574324	0.802083	0.851528	0.381132	0.452381	0.201835	...	0.689655	0.644068	0.244681	0.4885
4	0.785185	0.932292	0.682243	0.572414	0.574324	0.802083	0.851528	0.381132	0.452381	0.198777	...	0.672414	0.638418	0.244681	0.4885

5 rows × 29 columns

```
Warning: Total number of columns (29) exceeds max_columns (20) limiting to first (20) columns.
```

In [ ]:

```
train_df.iloc[:,1:].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 29989 entries, 0 to 29988
Data columns (total 29 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   0       29989 non-null  float64
 1   1       29989 non-null  float64
 2   2       29989 non-null  float64
 3   3       29989 non-null  float64
 4   4       29989 non-null  float64
 5   5       29989 non-null  float64
 6   6       29989 non-null  float64
 7   7       29989 non-null  float64
 8   8       29989 non-null  float64
 9   9       29989 non-null  float64
 10  10      29989 non-null  float64
 11  11      29989 non-null  float64
 12  12      29989 non-null  float64
 13  13      29989 non-null  float64
 14  14      29989 non-null  float64
 15  15      29989 non-null  float64
 16  16      29989 non-null  float64
 17  17      29989 non-null  float64
 18  18      29989 non-null  float64
 19  19      29989 non-null  float64
 20  20      29989 non-null  float64
 21  21      29989 non-null  float64
 22  22      29989 non-null  float64
 23  23      29989 non-null  float64
 24  24      29989 non-null  float64
 25  25      29989 non-null  float64
 26  26      29989 non-null  float64
 27  27      29989 non-null  float64
 28  labels  29989 non-null  int64
dtypes: float64(28), int64(1)
memory usage: 6.6 MB
```

In [ ]:

```
train_df.iloc[:,1:].describe()
```

```
Warning: Total number of columns (29) exceeds max_columns (20). Falling back to pandas display.
```

Out[9]:

	0	1	2	3	4	5	6	7	8	
count	29989.000000	29989.000000	29989.000000	29989.000000	29989.000000	29989.000000	29989.000000	29989.000000	29989.000000	29989.0
mean	0.544960	0.688247	0.540248	0.523526	0.507787	0.612537	0.748747	0.404003	0.408470	0.5
std	0.223028	0.168670	0.219905	0.209107	0.236347	0.216479	0.146647	0.184760	0.156300	0.1
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	0.385185	0.562500	0.336449	0.351724	0.297297	0.458333	0.681223	0.264151	0.315476	0.4
50%	0.533333	0.708333	0.570093	0.537931	0.520270	0.614583	0.772926	0.392453	0.386905	0.5
75%	0.740741	0.828125	0.719626	0.696552	0.675676	0.807292	0.860262	0.524528	0.470238	0.6
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

8 rows × 29 columns

In [ ]:

```
# Reduction and shuffle the data
train_df = train_df.iloc[:,1:]
test_df = test_df.iloc[:,1:]
train_data = pd.DataFrame(train_df.iloc[0,:])
test_data = pd.DataFrame(test_df.iloc[0,:])
train_data = train_data.T
test_data = test_data.T
for i in range(len(train_df)):
  if(i%30 == 0):
    train_data = train_data.append(train_df.iloc[i,:])
for j in range(len(test_df)):
  if(j%30 == 0):
    test_data = test_data.append(test_df.iloc[j,:])
train_data = train_data.iloc[1:,:]
test_data = test_data.iloc[1:,:]
#shuffle:
train_data = train_data.sample(frac=1).reset_index(drop=True)
test_data = test_data.sample(frac=1).reset_index(drop=True)
# train data = 1000 samples
# test data = 250 samples
```

In [ ]:

```
# encode the data
X_train = train_data.iloc[:, :-1].values
y_train = train_data.iloc[:, -1].values

X_test = test_data.iloc[:, :-1].values
y_test = test_data.iloc[:, -1].values
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(1000, 28) (1000,) (250, 28) (250,)
```

Optimization problem

Objective Function

In [ ]:

```
# Objective Function - build neural network, fit the net, and check accuracy on test data
def obj_func(n_first_layer ,n_hidden ,s_hidden):
  set_seed(5)
  # define model
  model = Sequential()
  model.add(Dense(n_first_layer, input_dim=28, activation='relu', input_shape=(28,))) #,kernel_initializer=initializers.Zeros
  for i in range(n_hidden):
    model.add(Dense(s_hidden, activation='relu'))
  model.add(Dense(5, activation='softmax'))
  # model.summary()
  trainableParams = np.sum([np.prod(v.get_shape()) for v in model.trainable_weights])
  nonTrainableParams = np.sum([np.prod(v.get_shape()) for v in model.non_trainable_weights])
  totalParams = trainableParams + nonTrainableParams

  # compile the model
  model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
  # fit the model
  model.fit(X_train, y_train, epochs=50, batch_size=20, verbose=0)
  # evaluate the model
  loss, acc = model.evaluate(X_test, y_test, verbose=0)

  return(totalParams,round(acc*100,2))
```

Define utility classes

In [ ]:

```
# print in every iteration at the minimize
class MyOutput(Output):

    def __init__(self):
        super().__init__()
        self.acc_ = Column("best acc", width=10)
        self.acc_w = Column("worst acc", width=10)
        self.par_ = Column("best params", width=10)
        self.par_w = Column("worst params", width=10)
        self.columns += [self.acc_, self.acc_w, self.par_, self.par_w]

    def update(self, algorithm):
        super().update(algorithm)
        res = algorithm.pop.get("F")
        res = np.round(res, 2)
        self.acc_.set(f'{np.min(res[:,0]):.2f}')
        self.acc_w.set(f'{np.max(res[:,0]):.2f}')
        self.par_.set(f'{np.min(res[:,1]):.0f}')
        self.par_w.set(f'{np.max(res[:,1]):.0f}')
        plt.scatter(res[:,0],res[:,1])
        plt.draw() # show()
```

In [ ]:

```
# save data for plots after the minimize
class MyCallback(Callback):

    def __init__(self) -> None:
        super().__init__()
        self.data["best"] = []
        self.index = 1
        self.data_history = pd.DataFrame(columns=['Gen' ,'Design_1', 'Design_2', 'Design_3', 'Objective_1', 'Objective_2'])

    def notify(self, algorithm):
        self.data["best"].append(algorithm.pop.get("F").min())
        self.data_history = self.data_history.append({'Gen': self.index, 'Design_1':  np.round(algorithm.pop.get("X")[:,0],0),
                                                       'Design_2':  np.round(algorithm.pop.get("X")[:,1],0),
                                                       'Design_3':  np.round(algorithm.pop.get("X")[:,2],0),
                                                       'Objective_1':  algorithm.pop.get("F")[:,0],
                                                       'Objective_2':  algorithm.pop.get("F")[:,1]}, ignore_index=True)
        self.index = self.index + 1
```

Define The Problem

- design parameters = s_first_layer [1,14], n_hidden [1,50] , s_hidden[1,100]
- objective = total parameters[41,<500K] , accuracy[0,100]

In [ ]:

```
# problem class
class Net_Struct_Problem(Problem):
      def __init__(self, **kwargs):
          super().__init__(n_var=3, n_obj=2, n_ieq_constr=0,xl=[1, 1, 1], xu=[14,50,100], vtype=int)

      def _evaluate(self, X, out, *args, **kwargs):
          num_params = []
          acc_ = []
          X = np.round(X)
          for x in X:
            f, n, s = int(x[0]), int(x[1]), int(x[2])
            t_param, acc = obj_func(f, n, s)
            acc_.append(acc)
            num_params.append(t_param)
          acc_ = np.array(acc_)
          num_params = np.array(num_params)
          out['F'] =  [-acc_, num_params]
```

Define algorithm parameters

In [ ]:

```
num_of_generations = 1
population_size = 10
```

Run The NSGA2 Algorithm (1 gen 10 pop)

In [ ]:

```
problem = Net_Struct_Problem()

algorithm = NSGA2(pop_size=population_size)

stop_criteria = ('n_gen', num_of_generations)

results_NSGA2 = minimize(
    problem=problem,
    algorithm=algorithm,
    callback=MyCallback(),
    output=MyOutput(),
    termination=stop_criteria,
    save_history=True,
    verbose=True
)
```

Data

In [ ]:

```
def from_pymoo_to_des_obj(results):
  des_1 = results.algorithm.callback.data_history['Design_1']
  des_1 = des_1[0][:]
  des_2 = results.algorithm.callback.data_history['Design_2']
  des_2 = des_2[0][:]
  des_3 = results.algorithm.callback.data_history['Design_3']
  des_3 = des_3[0][:]
  obj_1 = results.algorithm.callback.data_history['Objective_1']
  obj_1 = obj_1[0][:]
  obj_2 = results.algorithm.callback.data_history['Objective_2']
  obj_2 = obj_2[0][:]
  return des_1,des_2,des_3,obj_1,obj_2
```

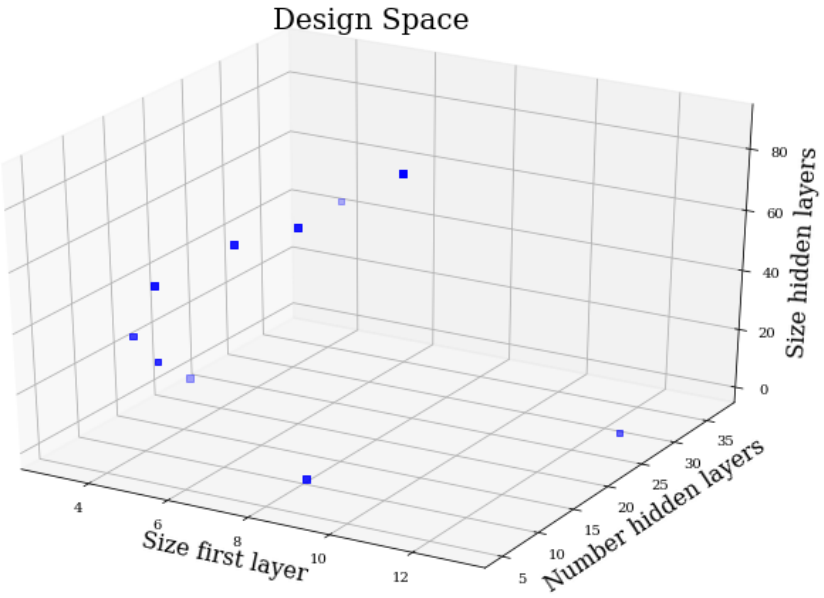
In [ ]:

```
des_1,des_2,des_3,obj_1,obj_2 = from_pymoo_to_des_obj(results_NSGA2)
```

Design Space

In [ ]:

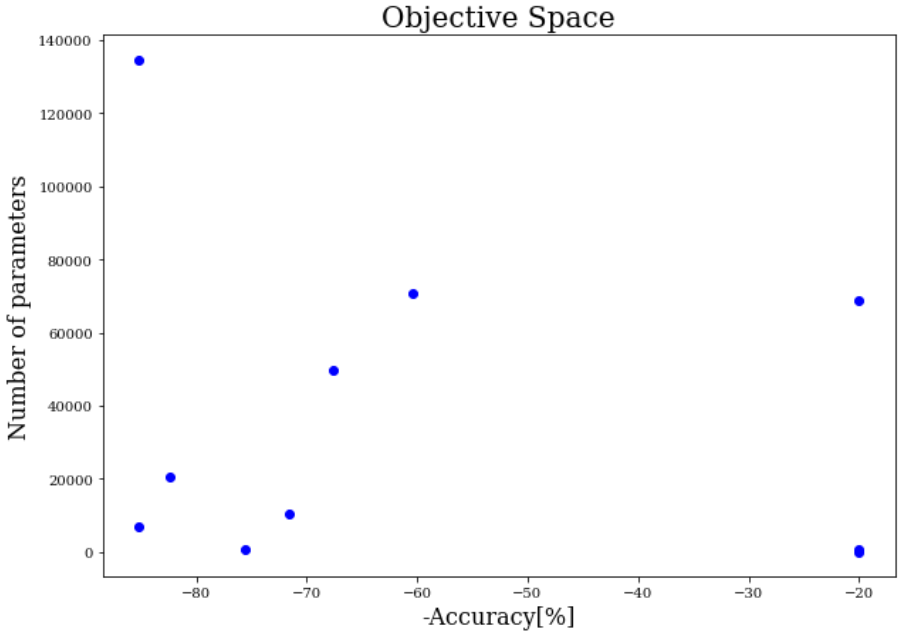
```
# Create the figure and axes for the 3D plot
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111,projection='3d')
ax.scatter(des_1, des_2, des_3,color=['blue'], marker="s")
ax.set_xlabel('Size first layer',fontsize=16)
ax.set_ylabel('Number hidden layers',fontsize=16)
ax.set_zlabel('Size hidden layers',fontsize=16)
ax.set_title('Design Space',fontsize=20) #NSGA2\n
plt.savefig('nsga_des.png')
plt.show()
```



Objective Space

In [ ]:

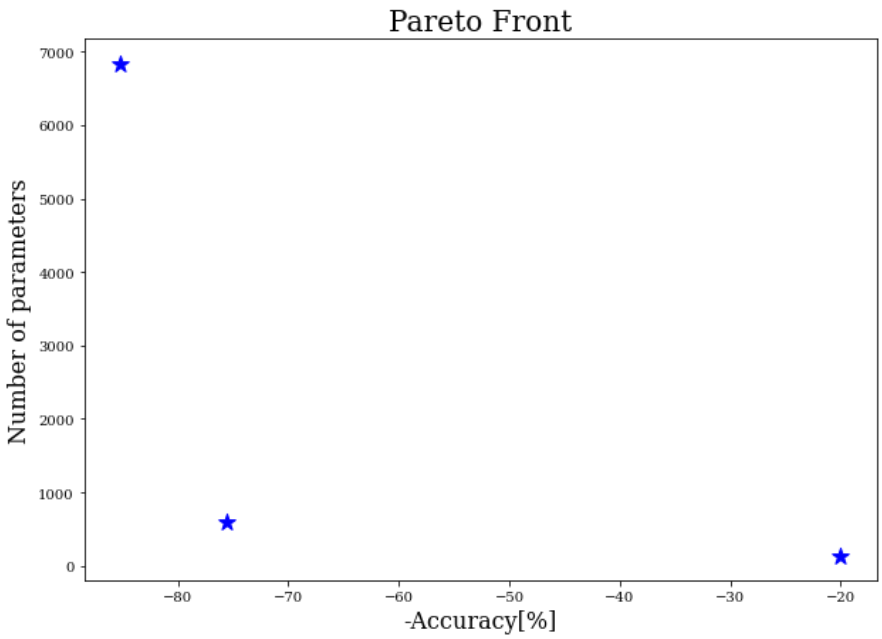
```
# Create the figure and axes for the 3D plot
fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(111)
ax.scatter(obj_1,obj_2, color=['blue'])
ax.set_xlabel('-Accuracy[%]',fontsize=16)
ax.set_ylabel('Number of parameters',fontsize=16)
ax.set_title('Objective Space',fontsize=20)
plt.savefig('nsga_obj.png')
plt.show()
```



Pareto Front

In [ ]:

```
# Create the figure and axes for the 3D plot
fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(111)
ax.scatter(results_NSGA2.F[:,0],results_NSGA2.F[:,1], color=['blue'], marker="*", s=150)
ax.set_xlabel('-Accuracy[%]',fontsize=16)
ax.set_ylabel('Number of parameters',fontsize=16)
ax.set_title('Pareto Front',fontsize=20)
plt.savefig('nsga_pareto.png')
plt.show()
```



Check another algorithm MOEAD (1 gen)

In [ ]:

```
problem = Net_Struct_Problem()

ref_dirs = get_reference_directions("uniform", 2, n_partitions=9)

algorithm_m = MOEAD(
    ref_dirs,
    n_neighbors=15,
    prob_neighbor_mating=0.7,
)

results_MOEAD = minimize(problem,
                algorithm_m,
                termination=stop_criteria,
                # seed=1,
                callback=MyCallback(),
                save_history=True,
                verbose=True)

Scatter().add(results_MOEAD.F).show()
# Scatter().add(ref_dirs).show()
```

Data

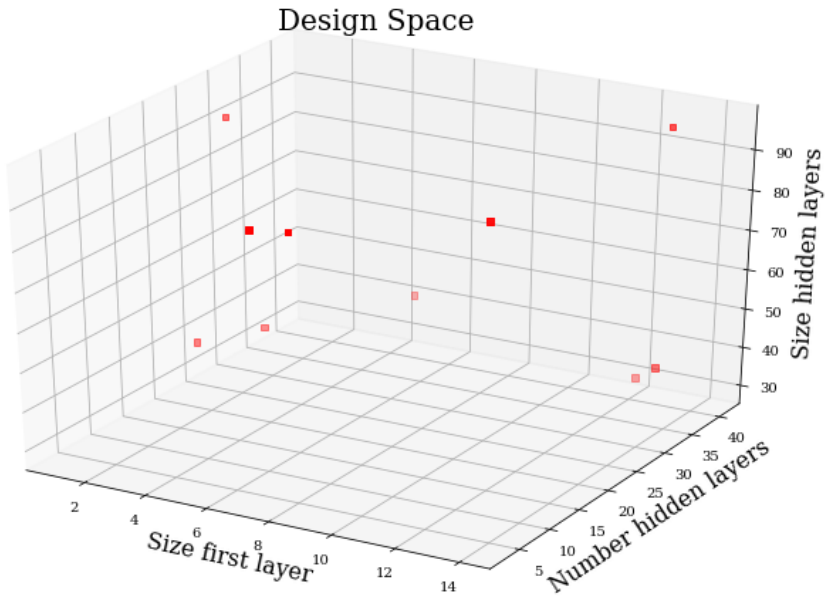
In [ ]:

```
des_1,des_2,des_3,obj_1,obj_2 = from_pymoo_to_des_obj(results_MOEAD)
```

Design Space

In [ ]:

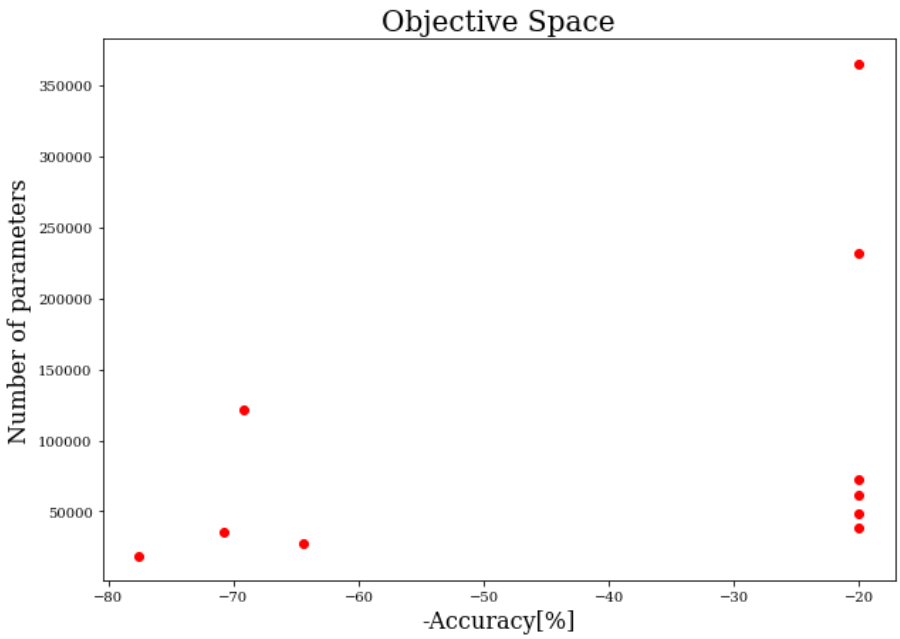
```
# Create the figure and axes for the 3D plot
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111,projection='3d')
ax.scatter(des_1, des_2, des_3,color=['red'], marker="s")
ax.set_xlabel('Size first layer',fontsize=16)
ax.set_ylabel('Number hidden layers',fontsize=16)
ax.set_zlabel('Size hidden layers',fontsize=16)
ax.set_title('Design Space',fontsize=20)
plt.savefig('moead_des.png')
plt.show()
```



Objective Space

In [ ]:

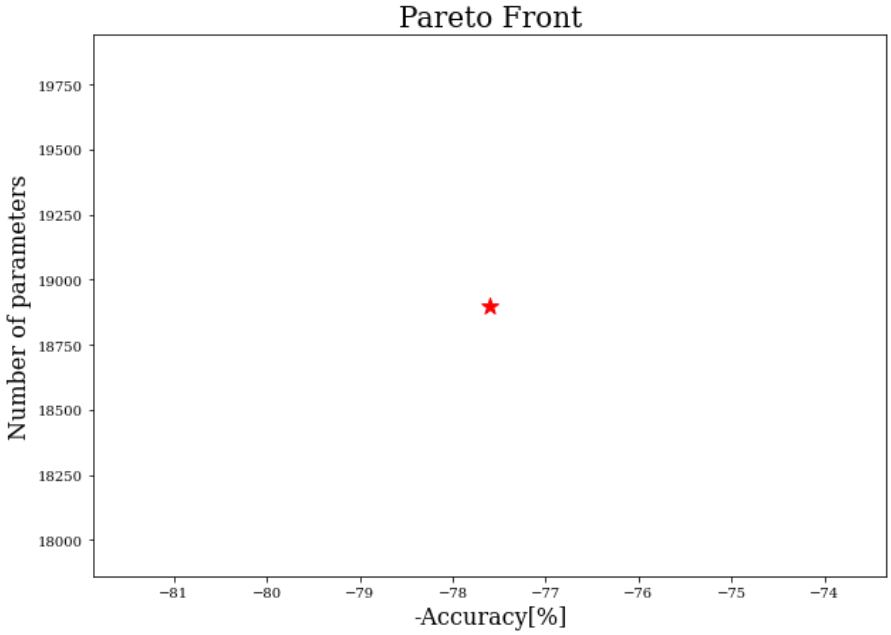
```
# Create the figure and axes for the 3D plot
fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(111)
ax.scatter(obj_1,obj_2, color=['red'])
ax.set_xlabel('-Accuracy[%]',fontsize=16)
ax.set_ylabel('Number of parameters',fontsize=16)
ax.set_title('Objective Space',fontsize=20)
plt.savefig('moead_obj.png')

plt.show()
```



Pareto Front

In [ ]:

```
# Create the figure and axes for the 3D plot
fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(111)
ax.scatter(results_MOEAD.F[:,0],results_MOEAD.F[:,1], color=['red'], marker="*", s=150)
ax.set_xlabel('-Accuracy[%]',fontsize=16)
ax.set_ylabel('Number of parameters',fontsize=16)
ax.set_title('Pareto Front',fontsize=20)
plt.savefig('moead_pareto.png')
plt.show()
```



Load all the runs data

In [ ]:

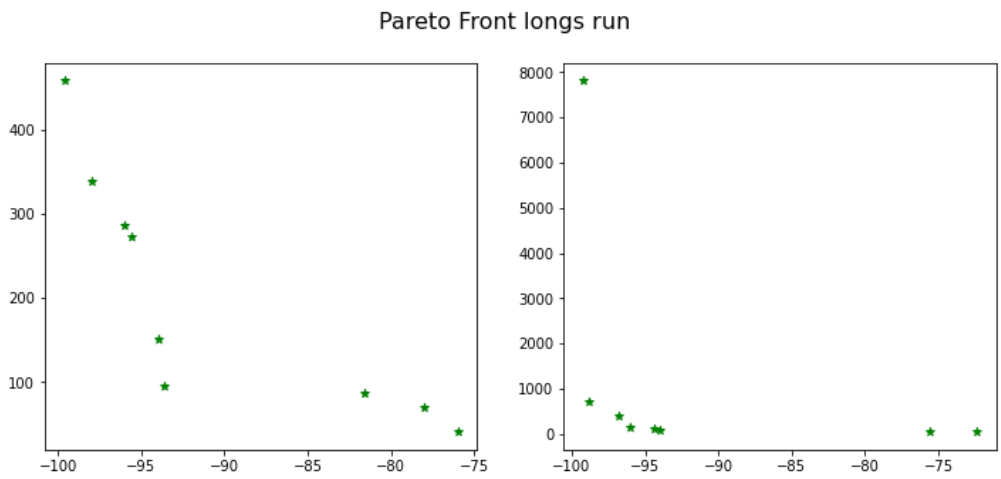
```
def open_res(run_number):
  # Load the object from the file using pickle
  with open(f'res_moead_{run_number}.pkl', 'rb') as f:
    moead_data = pickle.load(f)
  # Load the object from the file using pickle
  with open(f'res_nsga_{run_number}.pkl', 'rb') as l:
    nsga_data = pickle.load(l)
  return nsga_data, moead_data
```

In [ ]:

```
with open('/content/res_nsga_long_run.pkl', 'rb') as f:
  long_data = pickle.load(f)
with open('res_nsga_long_run_2.pkl', 'rb') as f:
  long_data_2 = pickle.load(f)
with open('/content/res_nsga_3_obj.pkl', 'rb') as f:
  long_3_obj = pickle.load(f)
```

In [ ]:

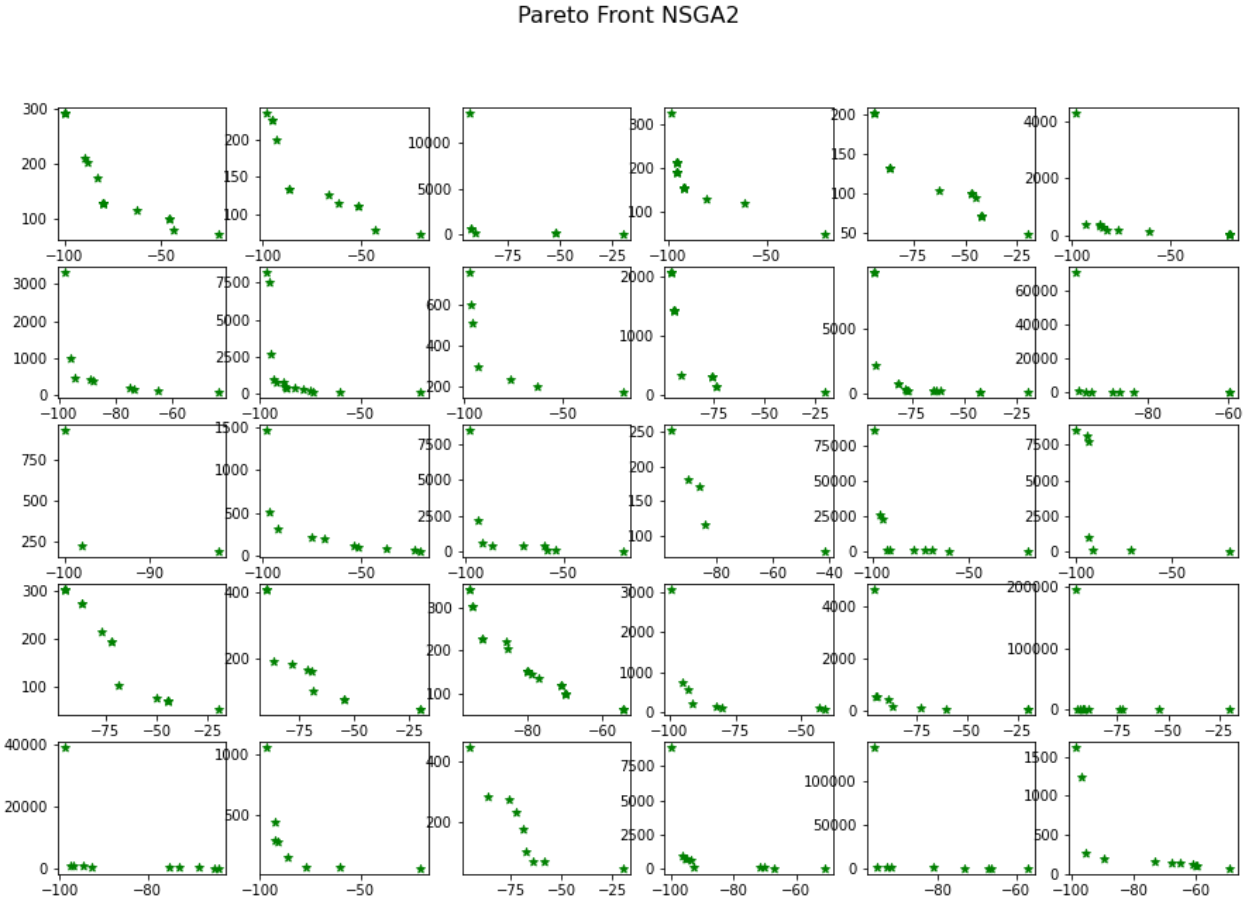
```
fig, axs = plt.subplots(1,2, figsize=(12,5))
axs[0].scatter(long_data.F[:,0],long_data.F[:,1], color=['green'], marker="*")
axs[1].scatter(long_data_2.F[:,0],long_data_2.F[:,1], color=['green'], marker="*")
fig.suptitle('Pareto Front longs run', fontsize=16)
plt.show()
```



Pareto Front

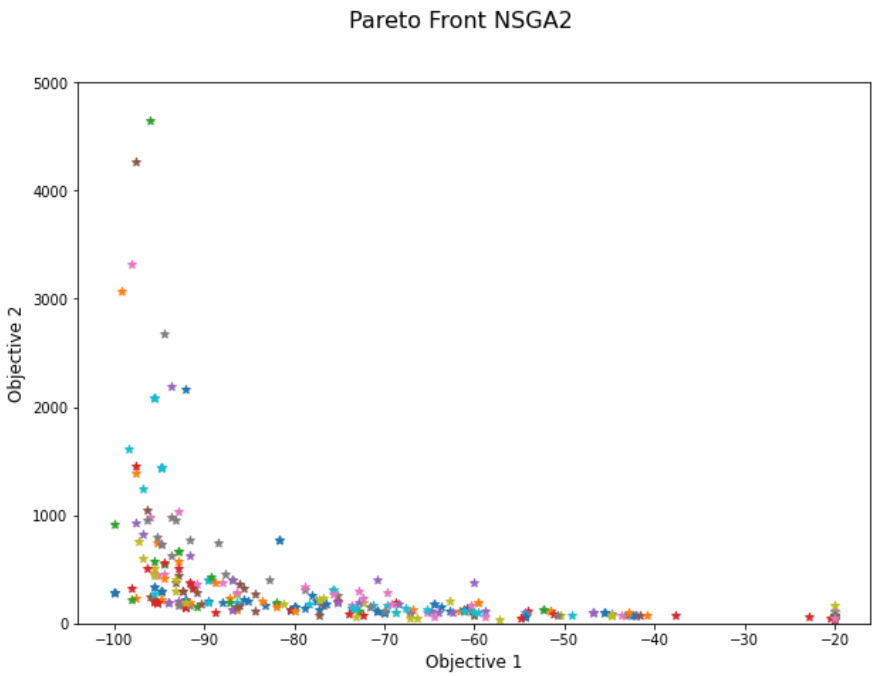
In [ ]:

```
fig, axs = plt.subplots(5,6, figsize=(15,10))
k = 0
for i in range(5):
  for j in range(6):
    nsga_data,moead_data = open_res(k)
    axs[i,j].scatter(nsga_data.F[:,0],nsga_data.F[:,1], color=['green'], marker="*")
    k += 1
fig.suptitle('Pareto Front NSGA2', fontsize=16)
plt.show()
```



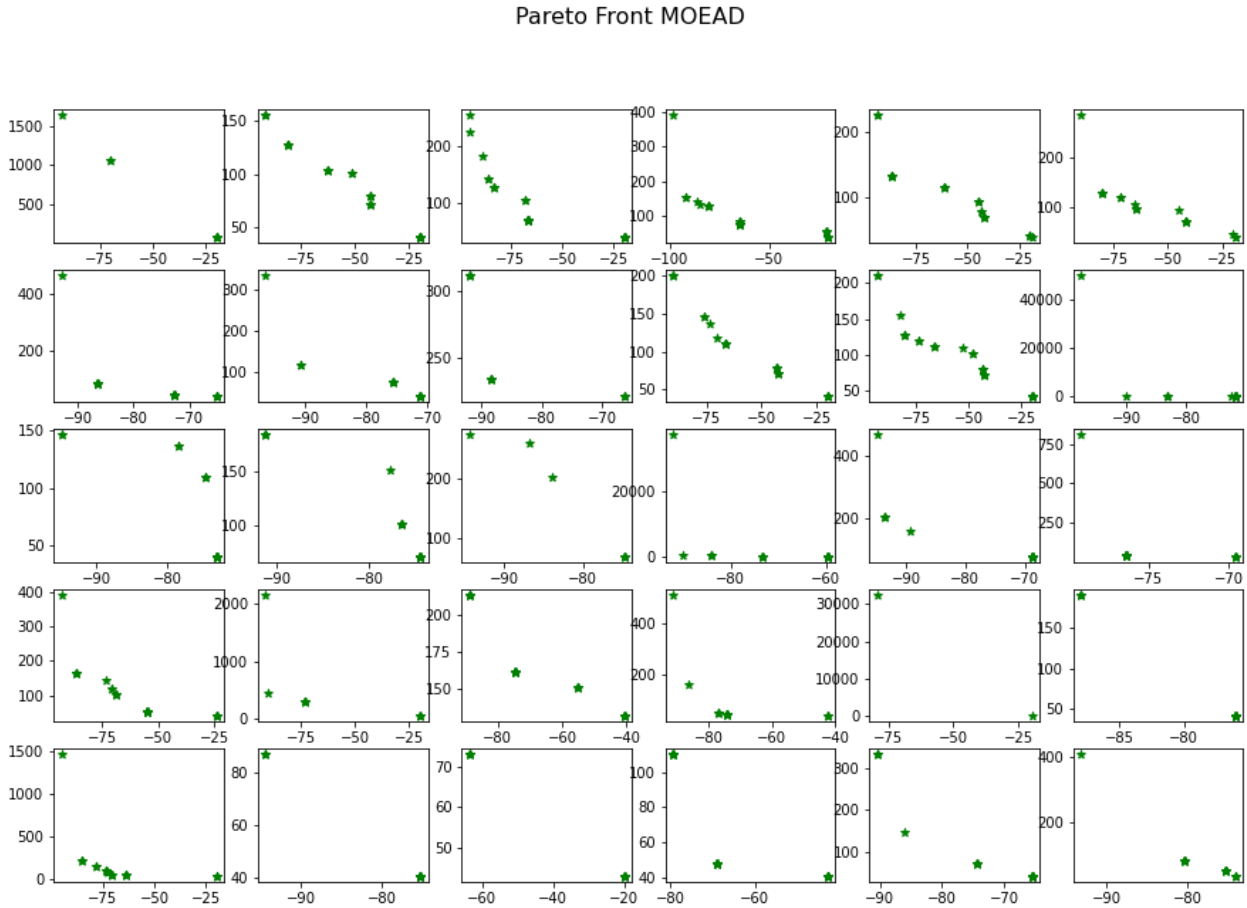
In [ ]:

```
fig, axs = plt.subplots(figsize=(10,7))
k = 0
for i in range(5):
  for j in range(6):
    nsga_data,moead_data = open_res(k)
    axs.scatter(nsga_data.F[:,0],nsga_data.F[:,1], marker="*")
    k += 1
fig.suptitle('Pareto Front NSGA2', fontsize=16)
axs.set_ylim([0,5000])
axs.set_xlabel("Objective 1", fontsize = 12)
axs.set_ylabel("Objective 2", fontsize = 12)

plt.show()
```



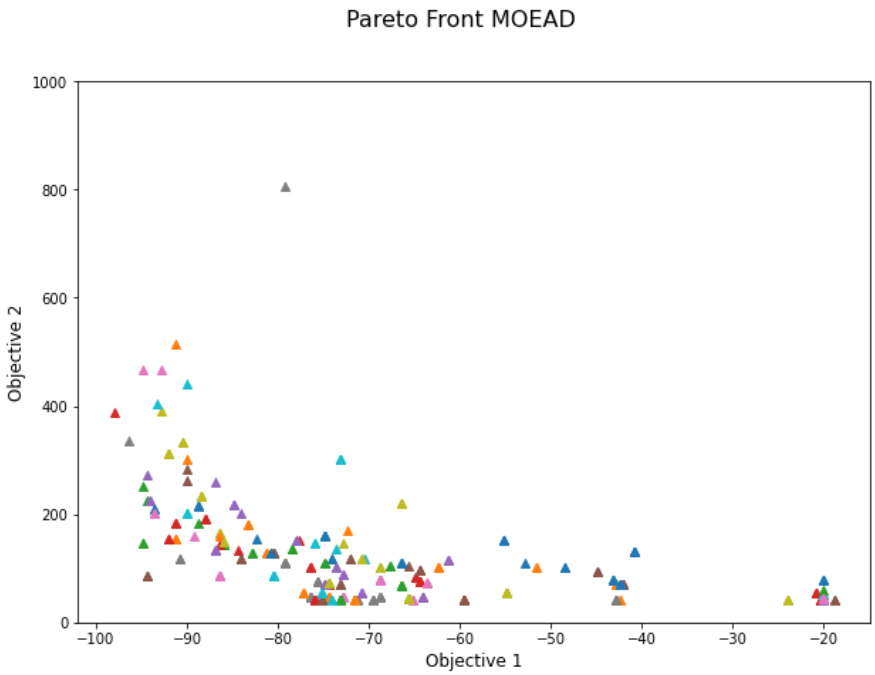
In [ ]:

```
fig, axs = plt.subplots(5,6, figsize=(15,10))
k = 0
for i in range(5):
  for j in range(6):
    nsga_data,moead_data = open_res(k)
    axs[i,j].scatter(moead_data.F[:,0],moead_data.F[:,1], color=['green'], marker="*")
    k += 1
fig.suptitle('Pareto Front MOEAD', fontsize=16)

plt.show()
```



In [ ]:

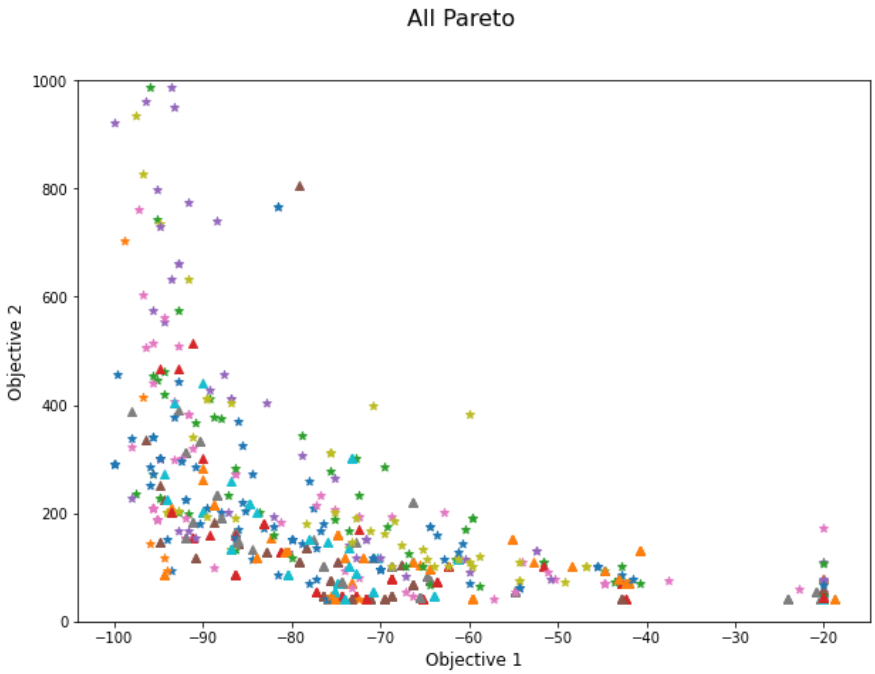
```
fig, axs = plt.subplots(figsize=(10,7))
k = 0
for i in range(5):
  for j in range(6):
    nsga_data,moead_data = open_res(k)
    axs.scatter(moead_data.F[:,0],moead_data.F[:,1], marker="^")
    k += 1
fig.suptitle('Pareto Front MOEAD', fontsize=16)
axs.set_ylim([0,1000])
axs.set_xlabel("Objective 1", fontsize = 12)
axs.set_ylabel("Objective 2", fontsize = 12)
plt.show()
```



United Pareto Front

In [ ]:

```
# show united pareto
fig, axs = plt.subplots(figsize=(10,7))
k = 0
all_points = pd.DataFrame(columns=['Acc' ,'Params', 'des_1', 'des_2', 'des_3'])
for i in range(5):
  for j in range(6):
    nsga_data,moead_data = open_res(k)
    axs.scatter(nsga_data.F[:,0],nsga_data.F[:,1], marker="*")
    axs.scatter(moead_data.F[:,0],moead_data.F[:,1], marker="^")
    for l in range(len(nsga_data.F[:,0])):
      all_points = all_points.append({'Acc': nsga_data.F[l,0], 'Params':  nsga_data.F[l,1],
                                      'des_1':  np.round(nsga_data.X[l,0],0), 'des_2':  np.round(nsga_data.X[l,1],0),
                                      'des_3':  np.round(nsga_data.X[l,2],0)}, ignore_index=True)
    for p in range(len(moead_data.F[:,0])):
      all_points = all_points.append({'Acc': moead_data.F[p,0], 'Params':  moead_data.F[p,1],
                                      'des_1':  np.round(moead_data.X[p,0],0), 'des_2':  np.round(moead_data.X[p,1],0),
                                      'des_3':  np.round(moead_data.X[p,2],0)}, ignore_index=True)
    k += 1

for l in range(len(long_data.F[:,0])):
  all_points = all_points.append({'Acc': long_data.F[l,0], 'Params':  long_data.F[l,1],
                                  'des_1':  np.round(long_data.X[l,0],0), 'des_2':  np.round(long_data.X[l,1],0),
                                  'des_3':  np.round(long_data.X[l,2],0)}, ignore_index=True)
for p in range(len(long_data_2.F[:,0])):
  all_points = all_points.append({'Acc': long_data_2.F[p,0], 'Params':  long_data_2.F[p,1],
                                  'des_1':  np.round(long_data_2.X[p,0],0), 'des_2':  np.round(long_data_2.X[p,1],0),
                                  'des_3':  np.round(long_data_2.X[p,2],0)}, ignore_index=True)

axs.scatter(long_data.F[:,0],long_data.F[:,1], marker="*")
axs.scatter(long_data_2.F[:,0],long_data_2.F[:,1], marker="*")

fig.suptitle('All Pareto', fontsize=16)
axs.set_xlabel("Objective 1", fontsize = 12)
axs.set_ylabel("Objective 2", fontsize = 12)
axs.set_ylim([0,1000])
plt.show()
```



In [ ]:

```
all_points
```

Out[29]:

	Acc	Params	des_1	des_2	des_3
0	-100.0	290.0	3.0	1.0	22.0
1	-100.0	290.0	3.0	1.0	22.0
2	-80.4	128.0	3.0	1.0	4.0
3	-80.4	128.0	3.0	1.0	4.0
4	-88.0	201.0	4.0	1.0	8.0
...	...	...	...	...	...
985	-94.4	119.0	3.0	1.0	3.0
986	-96.0	143.0	3.0	3.0	3.0
987	-96.8	415.0	7.0	2.0	9.0
988	-99.2	7812.0	11.0	2.0	78.0
989	-72.4	41.0	1.0	1.0	1.0

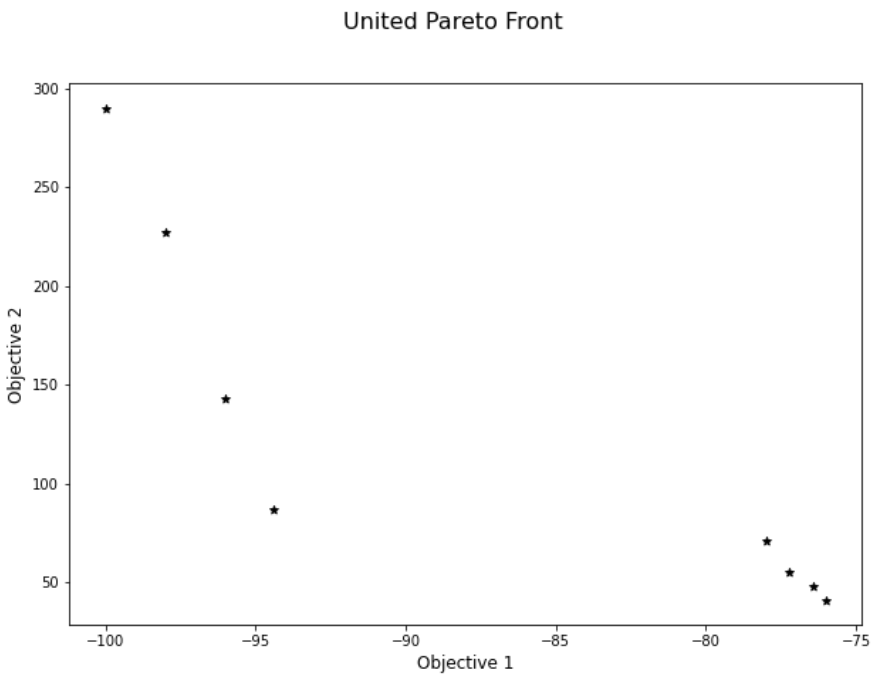
990 rows × 5 columns

In [ ]:

```
def pareto_front(points):
    pareto_points = []

    for point in points:
        is_pareto = True
        for other in points:
            if((point[0] != other[0] or point[1] != other[1]) and (point[0] >= other[0] and point[1] >= other[1])):
                is_pareto = False
                break
        if is_pareto:
            pareto_points.append(point)
    pareto_points = np.unique(np.array(pareto_points), axis=0)
    return pareto_points
```

In [ ]:

```
poi = np.array(all_points)
pareto_points = pareto_front(poi)
fig, axs = plt.subplots(figsize=(10,7))
axs.scatter(pareto_points[:,0],pareto_points[:,1], color=['black'], marker="*")
fig.suptitle('United Pareto Front', fontsize=16)
axs.set_xlabel("Objective 1", fontsize = 12)
axs.set_ylabel("Objective 2", fontsize = 12)
plt.show()
```



In [ ]:

```
pareto_points
```

Out[32]:

```
array([[-100. ,  290. ,    3. ,    1. ,   22. ],
       [ -98. ,  227. ,    6. ,    2. ,    3. ],
       [ -96. ,  143. ,    3. ,    3. ,    3. ],
       [ -94.4,   87. ,    2. ,    1. ,    3. ],
       [ -78. ,   71. ,    2. ,    1. ,    1. ],
       [ -77.2,   55. ,    1. ,    1. ,    3. ],
       [ -76.4,   48. ,    1. ,    1. ,    2. ],
       [ -76. ,   41. ,    1. ,    1. ,    1. ]])
```

Metrics

In [ ]:

```
def pareto_front_spread(points):

    # Calculate the distance between all pairs of points
    distances = []
    for i in range(len(points)):
        for j in range(i+1, len(points)):
            d = np.linalg.norm(np.array(points[i]) - np.array(points[j]))
            distances.append(d)

    # Calculate the spread metric as the average distance between points
    spread = np.mean(distances)

    return spread
```

In [ ]:

```
def get_metrics(results, united_fornt, ref_point = np.array([-20, 1000])):

  ind_1 = IGD(results.F) # pf - single pareto front
  igd = ind_1(united_fornt)

  ind_2 = HV(ref_point=ref_point)
  hv = ind_2(results.F)

  # calculate spread
  sp = pareto_front_spread(results.F)# /np.shape(results.F)

  # time
  time = results.exec_time/60 # min

  return igd, hv, sp ,time
```

In [ ]:

```
metrics_nsga = pd.DataFrame(columns=['IGD' ,'HV', 'SPREAD', 'TIME'])
metrics_moead = pd.DataFrame(columns=['IGD' ,'HV', 'SPREAD', 'TIME'])
united_fornt = pareto_points
for i in range(30):
  nsga_data,moead_data = open_res(i)
  igd, hv, sp ,time = get_metrics(nsga_data, united_fornt, ref_point = np.array([-20, 1000]))
  metrics_nsga = metrics_nsga.append({'IGD': igd, 'HV': hv, 'SPREAD': sp, 'TIME': time}, ignore_index=True)
  igd, hv, sp ,time = get_metrics(moead_data, united_fornt, ref_point = np.array([-20, 1000]))
  metrics_moead = metrics_moead.append({'IGD': igd, 'HV': hv, 'SPREAD': sp, 'TIME': time}, ignore_index=True)
```

Metrics Statistic

In [ ]:

```
metrics_nsga
```

Out[51]:

	IGD	HV	SPREAD	TIME
0	25.190392	68894.4	89.663660	41.813838
1	27.917401	68035.2	75.838859	38.387946
2	1980.744445	61281.6	3919.834991	49.173865
3	23.308062	67408.8	52.442044	38.401822
4	33.088666	65670.8	61.746820	47.484846
5	313.344441	59876.8	676.723379	45.996864
6	472.213766	61060.4	914.195887	36.938655
7	1322.339576	56896.4	2258.312833	31.556316
8	167.088807	58065.2	276.625040	30.525996
9	841.952642	57913.2	890.416000	29.372219
10	1925.435441	51786.8	3277.380938	34.100406
11	8064.003311	58815.2	16027.493057	37.187526
12	222.925696	62432.0	486.667485	31.296801
13	170.377288	62127.2	389.009893	34.713414
14	1207.557435	53032.0	2266.303810	34.778916
15	31.776177	66853.2	86.976856	25.407296
16	13515.810995	53826.0	23786.635311	34.575162
17	3477.831695	60369.2	4652.936897	29.929511
18	27.591297	63909.6	121.993818	26.233144
19	58.532720	60958.4	163.205275	25.606549
20	24.049467	66868.4	96.111025	20.327375
21	460.910496	64375.2	918.905746	36.402756
22	582.914581	63118.8	1172.449446	34.003160
23	17925.640710	66990.4	35899.322388	31.971061
24	4064.455023	63499.2	8031.796927	29.379196
25	129.679735	65266.0	347.120290	30.937805
26	43.354438	63000.4	159.543161	32.286144
27	1061.958492	65630.4	2009.739277	31.629486
28	15498.141180	68143.6	31077.053601	32.148323
29	233.690331	65705.6	473.772481	30.341903

In [ ]:

```
metrics_moead
```

Out[52]:

	IGD	HV	SPREAD	TIME
0	399.361607	0.0	673.012361	49.484956
1	37.279385	63632.4	53.548910	30.556569
2	26.787605	67196.8	65.833392	29.727450
3	32.226065	68598.0	69.153706	30.960574
4	33.112744	65280.4	61.058756	33.810084
5	33.862540	61592.4	55.440702	32.541557
6	13.981099	66410.0	55.869667	21.632449
7	9.716963	70281.6	57.927470	21.921216
8	20.794726	55474.4	46.656917	26.746250
9	36.414934	61692.4	61.511847	20.337495
10	39.179520	65041.6	59.932583	22.570903
11	2395.410745	63735.2	4848.515981	25.416795
12	8.262241	69555.2	43.226860	25.346910
13	14.401222	64465.6	42.113678	22.149386
14	11.807386	65838.8	98.790817	25.481356
15	1611.016306	64559.6	3251.849240	21.514183
16	22.619750	66206.8	73.767364	23.950881
17	26.372375	54586.0	76.200946	29.946124
18	31.952182	64237.2	69.601681	18.093724
19	216.647154	46578.0	486.675856	20.239724
20	35.375802	57749.2	38.042321	19.573843
21	21.374870	64643.2	77.241655	24.923297
22	16133.173203	0.0	32440.054017	48.041584
23	7.865912	63412.0	52.799731	20.599054
24	70.893530	59906.4	183.897422	20.865532
25	0.640000	70466.4	17.090113	23.027772
26	41.215175	40417.2	26.171257	36.748941
27	20.619664	55873.2	36.012802	31.788794
28	13.076963	64614.8	77.384020	30.201269
29	13.010572	65283.6	54.491426	20.521172

In [ ]:

```
# statistic of the metrics
metrics_nsga.describe()
```

Out[53]:

	IGD	HV	SPREAD	TIME
count	30.000000	30.000000	30.000000	30.000000
mean	2464.460824	62393.680000	4688.673907	33.763610
std	4804.346481	4597.911927	9376.198224	6.419642
min	23.308062	51786.800000	52.442044	20.327375
25%	47.149009	59999.900000	160.458690	30.387926
50%	387.127469	63059.600000	783.569690	32.217234
75%	1774.661474	65696.900000	3024.611656	36.804680
max	17925.640710	68894.400000	35899.322388	49.173865

In [ ]:

```
metrics_moead.describe()
```

Out[54]:

	IGD	HV	SPREAD	TIME
count	30.000000	30.000000	30.000000	30.000000
mean	712.615075	58244.280000	1441.795783	26.957328
std	2957.196589	17136.355009	5944.820529	7.674433
min	0.640000	0.000000	17.090113	18.093724
25%	14.086130	58288.500000	52.987026	21.543749
50%	29.369893	64351.400000	61.285302	25.135103
75%	38.704486	65700.000000	77.348429	30.467744
max	16133.173203	70466.400000	32440.054017	49.484956

Video

In [ ]:

```
Image(filename = f'/content/Pareto Video, long data.gif', width = 500, height = 500)
```

Out[5]:

In [ ]:

```
# fig, axs = plt.subplots( figsize=(10,7))


# res_nsga_long_run,res_nsga_long_run_2 = open_res(1)
# axs.scatter(res_nsga_long_run_2.F[:,0],res_nsga_long_run_2.F[:,1], color=['green'], marker="*")

# fig.suptitle('Pareto Front NSGA2 30 generations', fontsize=16)
# plt.show()
```

In [ ]:

```
#download data from colab to computer

# df_vid = long_3_obj.algorithm.callback.data_history
# df_vid
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

In [ ]:

In [ ]:

```
# import matplotlib.animation as animation

# obj_1 = df_vid['Objective_1']
# obj_2 = df_vid['Objective_2']

# #Clean data from space and get a list of values
# def obj_2_lst(obj):
#     obj_lst = []
#     for epoch in obj:
#         help_lst = []
#         curr_obj = epoch[1:-1].split(' ')
#         for word in curr_obj:
#             try:
#                 help_lst.append(float(word))
#             except:
#                 continue
#         obj_lst.append(help_lst)
#     obj_lst = np.array(obj_lst)
#     return(obj_lst)

# obj_lst_1 = obj_2_lst(obj_1)
# obj_lst_2 = obj_2_lst(obj_2)

# #Define the Video parameters
# frames = len(obj_lst_1)
# points = len(obj_lst_1[0])

# # init the figure & size
# fig, ax = plt.subplots(figsize=(7,7))

# #Main function, plots a new plot for every itteration
# def update(i):
#     ax.clear()
#     ax.scatter(obj_lst_1[i], obj_lst_2[i], label ="Generation: {}".format(i), c = 'coral', marker = '^' )
#     ax.legend()
#     ax.set_xlabel('Objective 1: Accuracy')
#     ax.set_ylabel('Objective 2: NN size')
#     ax.set_title('Pareto Front - Objective space, 30 Genertion')
#     ax.set_xlim(-100, 0)
#     #ax.set_ylim(0, 250000)

# ani = animation.FuncAnimation(fig, update, frames=frames, interval=400)
# ani.save('Pareto Video, long data 2.gif', writer='pillow')
# plt.show()
```

Build the best model and try it on all the data

- Design Parameters:
  - Size of first layer: 3
  - Number of hidden layers: 1
  - Size of hidden layers: 22
- Objectives on all the data:
  - Number of parameters: 290
  - Test Accuracy: 97.7%

In [ ]:

```
# build the optimal model after the optimization
def build_model(n_first_layer ,n_hidden ,s_hidden):
  set_seed(5)
  # define model
  model = Sequential()
  model.add(Dense(n_first_layer, input_dim=28, activation='relu', input_shape=(28,))) #,kernel_initializer=initializers.Zeros
  for i in range(n_hidden):
    model.add(Dense(s_hidden, activation='relu'))
  model.add(Dense(5, activation='softmax'))
  # model.summary()
  trainableParams = np.sum([np.prod(v.get_shape()) for v in model.trainable_weights])
  nonTrainableParams = np.sum([np.prod(v.get_shape()) for v in model.non_trainable_weights])
  totalParams = trainableParams + nonTrainableParams

  # compile the model
  model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
  # fit the model
  model.fit(X_train, y_train, epochs=50, batch_size=20, verbose=0)
  # evaluate the model
  loss, acc = model.evaluate(X_test, y_test, verbose=0)

  return model, totalParams, round(acc*100,2)
```

In [ ]:

```
model, params, acc = build_model(3,1,22)
print(f'Number of parameterss:  {params}')
# print(f'Accuracy:  {acc}%')
```

```
Number of parameterss:  290.0
```

In [ ]:

```
all_X_test = test_df.iloc[:, :-1].values
all_y_test = test_df.iloc[:, -1].values
print(all_X_test.shape, all_y_test.shape)
```

```
(7498, 28) (7498,)
```

In [ ]:

```
loss, acc = model.evaluate(all_X_test, all_y_test)
```

```
235/235 [==============================] - 1s 2ms/step - loss: 0.1075 - accuracy: 0.9768
```

In [ ]:

```
# check the final model

class_names = ['class_0','class_1','class_2','class_3','class_4']
class estimator:
  _estimator_type = ''
  classes_=[]
  def __init__(self, model, classes):
    self.model = model
    self._estimator_type = 'classifier'
    self.classes_ = classes
  def predict(self, X):
    y_prob= self.model.predict(X)
    y_pred = y_prob.argmax(axis=1)
    return y_pred

classifier = estimator(model, class_names)
```

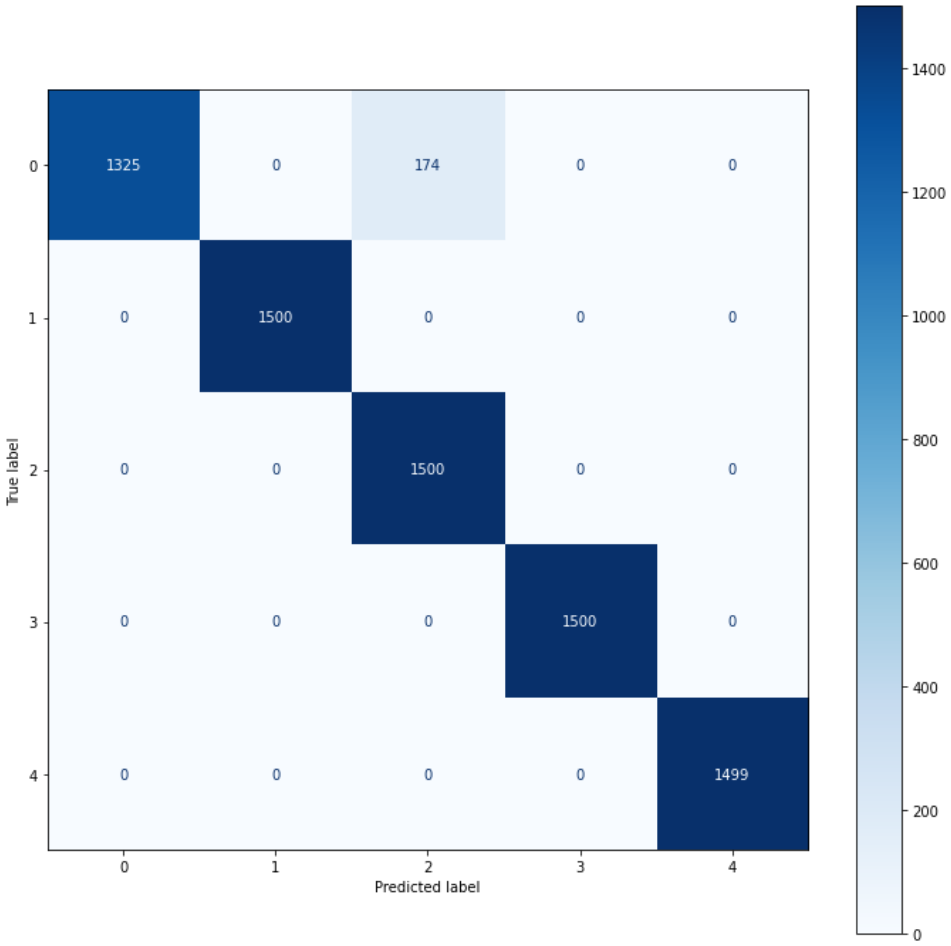
In [ ]:

```
from sklearn.metrics import plot_confusion_matrix
figsize = (12,12)
plot_confusion_matrix(estimator=classifier, X=all_X_test, y_true=all_y_test, cmap='Blues', normalize=None, ax=plt.subplots(figsiz
```

```
 25/235 [==>.........................] - ETA: 0s

/usr/local/lib/python3.8/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusion_matr
ix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the
class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)

235/235 [==============================] - 0s 2ms/step
```

Out[47]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f1af0957100>
```



Run algo with 4 des and 3 obj

Define The Problem

- design parameters = n_epochs[3,50], s_first_layer [1,14], n_hidden [1,50] , s_hidden[1,100]
- objective = total parameters[41,<500K] , accuracy[0,100], n_epochs[3,50]

In [ ]:

```
# Objective Function - build neural network, fit the net, and check accuracy on test data
def obj_func_2(n_first_layer ,n_hidden ,s_hidden, n_epochs):
  set_seed(5)
  # define model
  model = Sequential()
  model.add(Dense(n_first_layer, input_dim=28, activation='relu', input_shape=(28,)))
  for i in range(n_hidden):
    model.add(Dense(s_hidden, activation='relu'))
  model.add(Dense(5, activation='softmax'))
  # model.summary()
  trainableParams = np.sum([np.prod(v.get_shape()) for v in model.trainable_weights])
  nonTrainableParams = np.sum([np.prod(v.get_shape()) for v in model.non_trainable_weights])
  totalParams = trainableParams + nonTrainableParams

  # compile the model
  model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
  # fit the model
  model.fit(X_train, y_train, epochs=n_epochs, batch_size=20, verbose=0)
  # evaluate the model
  loss, acc = model.evaluate(X_test, y_test, verbose=0)

  return(totalParams, round(acc*100,2), n_epochs)
```

In [ ]:

```
# print in every iteration at the minimize
class MyOutput_2(Output):

    def __init__(self):
        super().__init__()
        self.acc_ = Column("best acc", width=10)
        self.acc_w = Column("worst acc", width=10)
        self.par_ = Column("best params", width=10)
        self.par_w = Column("worst params", width=10)
        self.ep_ = Column("best epochs", width=10)
        self.ep_w = Column("worst epochs", width=10)
        self.columns += [self.acc_, self.acc_w, self.par_, self.par_w, self.ep_, self.ep_w]

    def update(self, algorithm):
        super().update(algorithm)
        res = algorithm.pop.get("F")
        res = np.round(res, 2)
        self.acc_.set(f'{np.min(res[:,0]):.2f}')
        self.acc_w.set(f'{np.max(res[:,0]):.2f}')
        self.par_.set(f'{np.min(res[:,1]):.0f}')
        self.par_w.set(f'{np.max(res[:,1]):.0f}')
        self.ep_.set(f'{np.min(res[:,2]):.0f}')
        self.ep_w.set(f'{np.max(res[:,2]):.0f}')
        # plt.scatter(res[:,0],res[:,1])
        # plt.draw() # show()
```

In [ ]:

```
# save data for plots after the minimize
class MyCallback_2(Callback):

    def __init__(self) -> None:
        super().__init__()
        self.data["best"] = []
        self.index = 1
        self.data_history = pd.DataFrame(columns=['Gen' ,'Design_1', 'Design_2', 'Design_3', 'Design_4', 'Objective_1', 'Objectiv

    def notify(self, algorithm):
        self.data["best"].append(algorithm.pop.get("F").min())
        self.data_history = self.data_history.append({'Gen': self.index, 'Design_1':  np.round(algorithm.pop.get("X")[:,0],0),
                                                       'Design_2':  np.round(algorithm.pop.get("X")[:,1],0),
                                                       'Design_3':  np.round(algorithm.pop.get("X")[:,2],0),
                                                       'Design_4':  np.round(algorithm.pop.get("X")[:,3],0),
                                                       'Objective_1':  algorithm.pop.get("F")[:,0],
                                                       'Objective_2':  algorithm.pop.get("F")[:,1],
                                                       'Objective_3':  algorithm.pop.get("F")[:,2]}, ignore_index=True)
        self.index = self.index + 1
```

In [ ]:

```
# problem class
class Net_Struct_Problem_2(Problem):
      def __init__(self, **kwargs):
          super().__init__(n_var=4, n_obj=3, n_ieq_constr=0,xl=[3, 1, 1, 1], xu=[50, 14,50,100], vtype=int)

      def _evaluate(self, X, out, *args, **kwargs):
          num_params = []
          acc_ = []
          ep_ = []
          X = np.round(X)
          for x in X:
            e, f, n, s = int(x[0]), int(x[1]), int(x[2]), int(x[3])
            t_param, acc, ep = obj_func_2(f, n, s, e)
            acc_.append(acc)
            num_params.append(t_param)
            ep_.append(ep)
          acc_ = np.array(acc_)
          num_params = np.array(num_params)
          ep_ = np.array(ep_)
          out['F'] =  [-acc_, num_params, ep_]
```

In [ ]:

```
problem_2 = Net_Struct_Problem_2()

algorithm_n_2 = NSGA2(pop_size=60) # 60*5 = 50min

stop_criteria_2 = ('n_gen', 5)

results_NSGA2_3_obj = minimize(
    problem=problem_2,
    algorithm=algorithm_n_2,
    callback=MyCallback_2(),
    output=MyOutput_2(),
    termination=stop_criteria_2,
    save_history=True,
    verbose=True
)
```

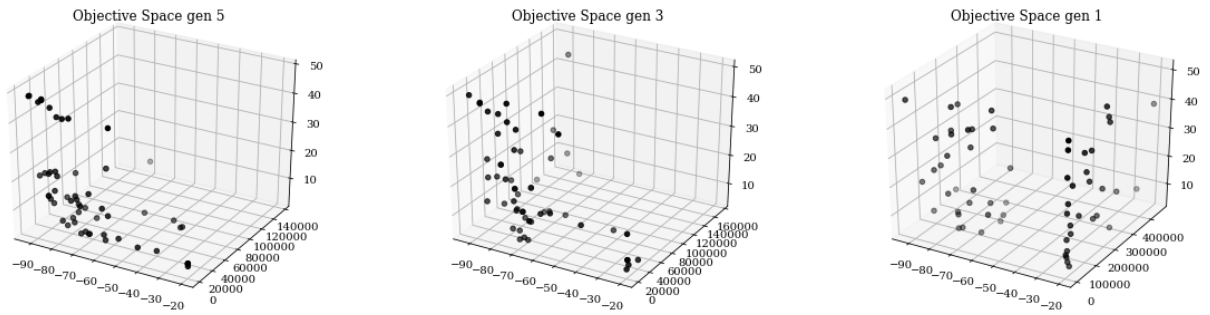
```
=====================================================================================================
n_gen  |  n_eval  |  best acc  | worst acc  | best params | worst params | best epochs | worst epochs
=====================================================================================================
     1 |       60 |     -92.80 |     -20.00 |        1801 |       457253 |           5 |           50
     2 |      120 |     -92.80 |     -20.00 |        1399 |       230073 |           5 |           49
     3 |      180 |     -92.80 |     -20.00 |         507 |       154111 |           5 |           49
     4 |      240 |     -92.80 |     -20.00 |         289 |       154111 |           3 |           49
     5 |      300 |     -92.80 |     -20.00 |         289 |       136422 |           3 |           48
```

In [ ]:

```
def get_obj(res, i):
  ddes_1 = res.algorithm.callback.data_history['Design_1']
  ddes_1 = ddes_1[i][:]
  ddes_2 = res.algorithm.callback.data_history['Design_2']
  ddes_2 = ddes_2[i][:]
  ddes_3 = res.algorithm.callback.data_history['Design_3']
  ddes_3 = ddes_3[i][:]
  ddes_4 = res.algorithm.callback.data_history['Design_4']
  ddes_4 = ddes_4[i][:]
  oobj_1 = res.algorithm.callback.data_history['Objective_1']
  oobj_1 = oobj_1[i][:]
  oobj_2 = res.algorithm.callback.data_history['Objective_2']
  oobj_2 = oobj_2[i][:]
  oobj_3 = res.algorithm.callback.data_history['Objective_3']
  oobj_3 = oobj_3[i][:]
  return oobj_1, oobj_2, oobj_3
```

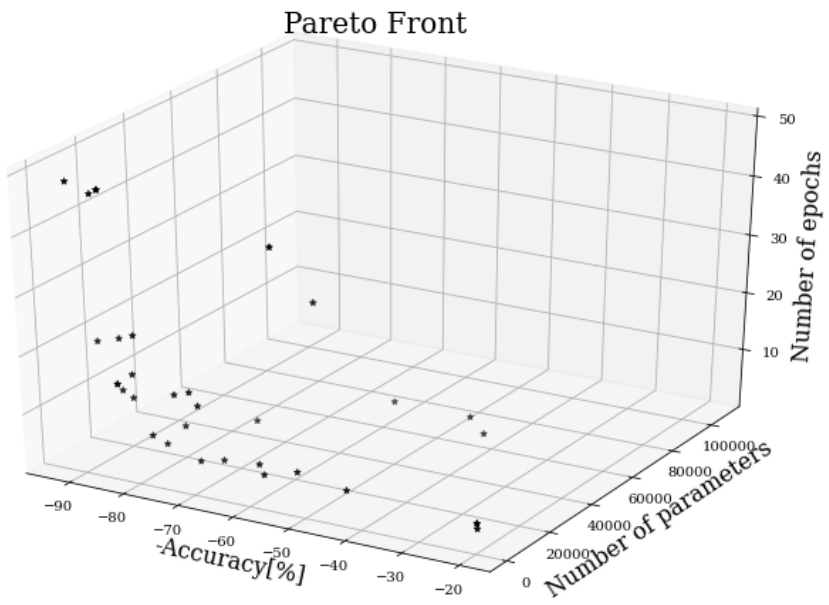
In [ ]:

```
fig = plt.figure(figsize=(20,5))
ax = fig.add_subplot(131,projection='3d')
oobj_1, oobj_2, oobj_3 = get_obj(results_NSGA2_3_obj, 4)
ax.scatter(oobj_1, oobj_2, oobj_3,color=['black'])
# ax.set_xlabel('-Accuracy[%]',fontsize=4)
# ax.set_ylabel('Number of parameters',fontsize=4)
# ax.set_zlabel('Number of epochs',fontsize=4)
ax.set_title('Objective Space gen 5',fontsize=12)
ax2 = fig.add_subplot(132,projection='3d')
oobj_1, oobj_2, oobj_3 = get_obj(results_NSGA2_3_obj, 2)
ax2.scatter(oobj_1, oobj_2, oobj_3,color=['black'])
ax2.set_title('Objective Space gen 3',fontsize=12)
ax3 = fig.add_subplot(133,projection='3d')
oobj_1, oobj_2, oobj_3 = get_obj(results_NSGA2_3_obj, 0)
ax3.scatter(oobj_1, oobj_2, oobj_3,color=['black'])
ax3.set_title('Objective Space gen 1',fontsize=12)


plt.show()
```



In [ ]:

```
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111,projection='3d')
ax.scatter(results_NSGA2_3_obj.F[:,0],results_NSGA2_3_obj.F[:,1], results_NSGA2_3_obj.F[:,2],color=['black'], marker="*")
ax.set_xlabel('-Accuracy[%]',fontsize=16)
ax.set_ylabel('Number of parameters',fontsize=16)
ax.set_zlabel('Number of epochs',fontsize=16)
ax.set_title('Pareto Front',fontsize=20)

plt.show()
```



Video from long and big run

In [ ]:

```
Image(filename = f'/content/3 Objectives.gif', width = 600, height = 500)
```

Out[7]: