מעבדה בבינה מלאכותית	
מטלה 5 חלק ב	
נדאא אבו סאלח	316151232
ג'ו כנג'	038163630

א) הורדנו את קובץ הדאטה מהאתר ויצרנו קובץ CSV בתוך ה colab והשתמשנו בפונקציה שקוראת את הקובץ הזה.

הורידו וקיראו את הדאטה

link text

```
[5]  target_url = ("https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data")
     df = pd.read_csv(target_url, header=None)
     df.columns = ['', '', '', '', '', '','', '', '', '', 'GlassType']
     print('Data File:')
     df.head()
```

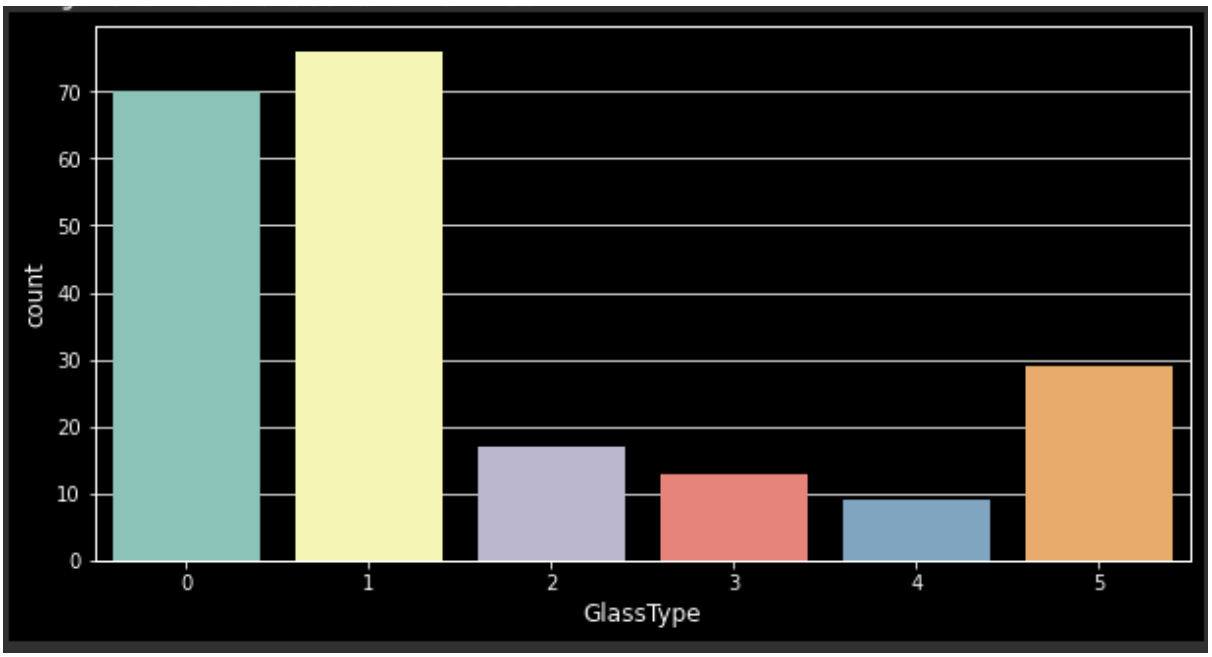
```
Data File:
```

											GlassType
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

ב)

התאימו קוד בין 0-ל5 לכל סוג זכוכית אפשרי - ציינו את ההתפלגות של הדוגמאות לפי כל סוג זכוכית.

```
[6]  plt.style.use('dark_background')
     plt.figure(figsize=(10,5))
     figure, ax = plt.subplots(1,1, figsize=(10,5))
     sns.countplot(x = 'GlassType', data=df)
     ax.set_xticklabels( ('0','1','2','3','4','5') )
     plt.show()
     df['GlassType'].value_counts()
```



COUNT:

```
2    76
1    70
7    29
3    17
5    13
6     9
Name: GlassType, dtype: int64
```

ג+ד) חלקנו את הדאטה לקבוצת אימון 80% לקבוצת בדיקה 20% תוך נרמול תוצאות.

חלקו את הדאטה לקבוצת אימון 80% וקבוצת בדיקה 20%

```python
label = df.values[:, -1]
x_train, x_test, y_train, y_test = train_test_split(df, label, stratify=label, test_size=0.2, random_state=1)
print("Train part:")
print("Train X:")
print(x_train)
print("Train Y:")
print(y_train)
print("-----------------------------")
print("Test part:")
print("Test X:")
print(x_test)
print("Test Y:")
print(y_test)
```

```
Train part:
Train X:
                                                                                \
96    97  1.51841  13.02  3.62  1.06  72.34  0.64   9.13  0.00  0.15
98    99  1.51689  12.67  2.88  1.71  73.21  0.73   8.54  0.00  0.00
194  195  1.51683  14.56  0.00  1.98  73.29  0.00   8.52  1.57  0.07
106  107  1.53125  10.73  0.00  2.10  69.81  0.58  13.30  3.15  0.28
209  210  1.51623  14.14  0.00  2.88  72.61  0.08   9.18  1.06  0.00
..   ...      ...    ...   ...   ...    ...   ...    ...   ...   ...
99   100  1.51811  12.96  2.96  1.43  72.92  0.60   8.79  0.14  0.00
124  125  1.52177  13.20  3.68  1.15  72.75  0.54   8.52  0.00  0.00
208  209  1.51640  14.37  0.00  2.74  72.85  0.00   9.45  0.54  0.00
153  154  1.51610  13.42  3.40  1.22  72.69  0.59   8.32  0.00  0.00
23    24  1.51751  12.81  3.57  1.35  73.02  0.62   8.59  0.00  0.00

     GlassType
96           2
98           2
194          7
106          2
209          7
..         ...
99           2
124          2
208          7
153          3
23           1
```

```
Train Y:
[2. 2. 7. 2. 7. 2. 2. 1. 2. 2. 1. 2. 3. 1. 7. 7. 7. 7. 6. 1. 2. 1. 5. 1.
 1. 2. 2. 7. 2. 1. 3. 2. 1. 2. 1. 3. 2. 2. 6. 1. 3. 3. 1. 1. 2. 7. 2. 2.
 2. 2. 2. 7. 5. 1. 5. 1. 2. 7. 2. 1. 2. 3. 1. 3. 1. 2. 1. 3. 2. 1. 2. 1.
 2. 7. 2. 5. 3. 2. 2. 7. 2. 2. 1. 7. 1. 3. 1. 1. 2. 1. 1. 1. 5. 1. 5. 1.
 7. 5. 2. 6. 7. 1. 2. 2. 3. 1. 1. 1. 7. 2. 7. 1. 3. 2. 2. 6. 2. 1. 1. 2.
 2. 2. 7. 1. 5. 2. 6. 7. 6. 2. 6. 7. 2. 1. 2. 1. 5. 2. 1. 1. 1. 1. 2. 1.
 2. 2. 2. 1. 2. 1. 7. 2. 2. 5. 2. 1. 1. 1. 7. 1. 1. 1. 2. 3. 1. 1. 2. 2.
 7. 3. 1.]
-----------------------------
```

	GlassType
78	2
104	2
135	2
85	2
191	7
50	1
80	2
168	5
12	1
6	1
155	3
45	1
20	1
97	2
68	1
164	5
127	2
108	2
169	5
64	1
205	7
148	3
198	7
213	7
5	1
9	1
95	2
212	7
183	6
7	1
196	7
1	1
107	2
144	2
37	1
25	1
137	2
16	1
93	2
117	2
181	6
152	3
102	2

Test X:

78	79	1.51613	13.92	3.52	1.25	72.88	0.37	7.94	0.00	0.14
104	105	1.52410	13.83	2.90	1.17	71.15	0.08	10.79	0.00	0.00
135	136	1.51789	13.19	3.90	1.30	72.33	0.55	8.44	0.00	0.28
85	86	1.51625	13.36	3.58	1.49	72.72	0.45	8.21	0.00	0.00
191	192	1.51602	14.85	0.00	2.38	73.28	0.00	8.76	0.64	0.09
50	51	1.52320	13.72	3.72	0.51	71.75	0.09	10.06	0.00	0.16
80	81	1.51592	12.86	3.52	2.12	72.66	0.69	7.97	0.00	0.00
168	169	1.51666	12.86	0.00	1.83	73.88	0.97	10.17	0.00	0.00
12	13	1.51589	12.88	3.43	1.40	73.28	0.69	8.05	0.00	0.24
6	7	1.51743	13.30	3.60	1.14	73.09	0.58	8.17	0.00	0.00
155	156	1.51646	13.04	3.40	1.26	73.01	0.52	8.58	0.00	0.00
45	46	1.51900	13.49	3.48	1.35	71.95	0.55	9.00	0.00	0.00
20	21	1.51750	12.82	3.55	1.49	72.75	0.54	8.52	0.00	0.19
97	98	1.51743	12.20	3.25	1.16	73.55	0.62	8.90	0.00	0.24
68	69	1.52152	13.12	3.58	0.90	72.20	0.23	9.82	0.00	0.16
164	165	1.51915	12.73	1.85	1.86	72.69	0.60	10.09	0.00	0.00
127	128	1.52081	13.78	2.28	1.43	71.99	0.49	9.85	0.00	0.17
108	109	1.52222	14.43	0.00	1.00	72.67	0.10	11.52	0.00	0.08
169	170	1.51994	13.27	0.00	1.76	73.03	0.47	11.32	0.00	0.00
64	65	1.52172	13.48	3.74	0.90	72.01	0.18	9.61	0.00	0.07
205	206	1.51732	14.95	0.00	1.80	72.99	0.00	8.61	1.55	0.00
148	149	1.51670	13.24	3.57	1.38	72.70	0.56	8.44	0.00	0.10
198	199	1.51531	14.38	0.00	2.66	73.10	0.04	9.08	0.64	0.00
213	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.00
5	6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.00	0.26
9	10	1.51755	13.00	3.60	1.36	72.99	0.57	8.40	0.00	0.11
95	96	1.51860	13.36	3.43	1.43	72.26	0.51	8.60	0.00	0.00
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.00
183	184	1.51969	14.56	0.00	0.56	73.48	0.00	11.22	0.00	0.00
7	8	1.51756	13.15	3.61	1.05	73.24	0.57	8.24	0.00	0.00
196	197	1.51556	13.87	0.00	2.54	73.23	0.14	9.41	0.81	0.01
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.00
107	108	1.53393	12.30	0.00	1.00	70.16	0.12	16.19	0.00	0.24
144	145	1.51660	12.99	3.18	1.23	72.97	0.58	8.81	0.00	0.24
37	38	1.51797	12.74	3.48	1.35	72.96	0.64	8.68	0.00	0.00
25	26	1.51764	12.98	3.54	1.21	73.00	0.65	8.53	0.00	0.00
137	138	1.51711	12.89	3.62	1.57	72.96	0.61	8.11	0.00	0.00
16	17	1.51784	12.68	3.67	1.16	73.11	0.61	8.70	0.00	0.00
93	94	1.51590	13.24	3.34	1.47	73.10	0.39	8.22	0.00	0.00
117	118	1.51708	13.72	3.68	1.81	72.06	0.64	7.88	0.00	0.00
181	182	1.51888	14.99	0.78	1.74	72.50	0.00	9.95	0.00	0.00
152	153	1.51779	13.64	3.65	0.65	73.00	0.06	8.93	0.00	0.00
102	103	1.51820	12.62	2.76	0.83	73.81	0.35	9.42	0.00	0.20

```
Test Y:
[2. 2. 2. 2. 7. 1. 2. 5. 1. 1. 3. 1. 1. 2. 1. 5. 2. 2. 5. 1. 7. 3. 7. 7.
 1. 1. 2. 7. 6. 1. 7. 1. 2. 2. 1. 1. 2. 1. 2. 2. 6. 3. 2.]
```

נרמלו את מאפייני הדאטה להתפלגות נורמלית סטנדרטית (ממוצע 0 וסטית תקן 1) שיהווה קלט לרשת

```
print("Data before Normalizing : ")
print(df)
normalized = pd.DataFrame((preprocessing.MinMaxScaler()).fit_transform(df))
print("Data after Normalizing : ")
print(normalized)
```

```
Data before Normalizing :
                                                                    GlassType
0      1  1.52101  13.64  4.49  1.10  71.78  0.06  8.75  0.00  0.0          1
1      2  1.51761  13.89  3.60  1.36  72.73  0.48  7.83  0.00  0.0          1
2      3  1.51618  13.53  3.55  1.54  72.99  0.39  7.78  0.00  0.0          1
3      4  1.51766  13.21  3.69  1.29  72.61  0.57  8.22  0.00  0.0          1
4      5  1.51742  13.27  3.62  1.24  73.08  0.55  8.07  0.00  0.0          1
..   ...      ...    ...   ...   ...    ...   ...   ...   ...  ...        ...
209  210  1.51623  14.14  0.00  2.88  72.61  0.08  9.18  1.06  0.0          7
210  211  1.51685  14.92  0.00  1.99  73.06  0.00  8.40  1.59  0.0          7
211  212  1.52065  14.36  0.00  2.02  73.42  0.00  8.44  1.64  0.0          7
212  213  1.51651  14.38  0.00  1.94  73.61  0.00  8.48  1.57  0.0          7
213  214  1.51711  14.23  0.00  2.08  73.36  0.00  8.62  1.67  0.0          7

[214 rows x 11 columns]
Data after Normalizing :
           0         1         2         3         4         5         6   \
0    0.000000  0.432836  0.437594  1.000000  0.252336  0.351786  0.009662
1    0.004695  0.283582  0.475188  0.801782  0.333333  0.521429  0.077295
2    0.009390  0.220808  0.421053  0.790646  0.389408  0.567857  0.062802
3    0.014085  0.285777  0.372932  0.821826  0.311526  0.500000  0.091787
4    0.018779  0.275241  0.381955  0.806236  0.295950  0.583929  0.088567
..        ...       ...       ...       ...       ...       ...       ...
209  0.981221  0.223003  0.512782  0.000000  0.806854  0.500000  0.012882
210  0.985915  0.250219  0.630075  0.000000  0.529595  0.580357  0.000000
211  0.990610  0.417032  0.545865  0.000000  0.538941  0.644643  0.000000
212  0.995305  0.235294  0.548872  0.000000  0.514019  0.678571  0.000000
213  1.000000  0.261633  0.526316  0.000000  0.557632  0.633929  0.000000

           7         8    9    10
0    0.308550  0.000000  0.0  0.0
1    0.223048  0.000000  0.0  0.0
2    0.218401  0.000000  0.0  0.0
3    0.259294  0.000000  0.0  0.0
4    0.245353  0.000000  0.0  0.0
..        ...       ...  ...  ...
209  0.348513  0.336508  0.0  1.0
210  0.276022  0.504762  0.0  1.0
211  0.279740  0.520635  0.0  1.0
212  0.283457  0.498413  0.0  1.0
213  0.296468  0.530159  0.0  1.0

[214 rows x 11 columns]
```

ה) רשת נירונית פשוטה בעלת קישוריות מליאה.

יצרו רשת נוירונים פשוטה MLP בעלת קישוריות מלאה FULLY CONNECTED

דו-שכבתית המקבלת קלטים של 9 המאפיינים מסווגת אותם ומוציאה שישה פלטים (logits)

```
from sklearn import neural_network
from sklearn.metrics import classification_report,accuracy_score,confusion_matrix
import numpy as np
mlp = neural_network.MLPClassifier(max_iter=50000,hidden_layer_sizes=(9,6))
```

ו + ז) פונקצית ה-SOFTMAX לפי קריטריון F1:

את ששת הפלטים יש לאגד לפונקצית הסתברות באמצעות פונקצית SOFTMAX

[116]

```python
def softmax(x):
    """Compute softmax values for each sets of scores in x."""
    e_x = np.exp(x - np.max(x))
    return e_x / e_x.sum(axis=0) # only difference


print(mlp)
mlp.fit(x_train,y_train)
print(mlp.predict_proba(x_test[:1]))
print(softmax(mlp.predict_proba(x_test[:1])))
print(mlp.predict(x_test))
#print(y_test)
#print(mlp.score(x_test, y_test))
#Predict
y_pred = mlp.predict(x_test)
#y_pred = softmax(x_pred)

#x_pred = mlp.predict_proba(x_test[:1])
#y_pred = softmax(x_pred)

#Score
acu = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test,y_pred)
print(CGREEN + "Confusion matrix:\n",cm)

print(CBOLD + "\n Accuracy of ANN:",acu)
print("\n"+ CEND)
print(classification_report(y_test, y_pred))
```

```
Confusion matrix:
 [[ 9  5  0  0  0  0]
 [ 2 12  0  1  0  0]
 [ 2  1  0  0  0  0]
 [ 0  0  0  0  0  3]
 [ 0  1  0  0  0  1]
 [ 0  0  0  1  0  5]]

 Accuracy of ANN: 0.6046511627906976


              precision    recall  f1-score   support

           0       0.69      0.64      0.67        14
           1       0.63      0.80      0.71        15
           2       0.00      0.00      0.00         3
           3       0.00      0.00      0.00         3
           4       0.00      0.00      0.00         2
           5       0.56      0.83      0.67         6

    accuracy                           0.60        43
   macro avg       0.31      0.38      0.34        43
weighted avg       0.52      0.60      0.56        43
```

הרחיבו והתאימו את המנוע הגנטי שלכם לפי העקרונות שתוארו בהרצאה כדי לייצר רשתות נוירונים עמוקות

[54]

```python
#from the Lecture:

import random
import numpy as np
from IPython.display import clear_output

# sigmoid and ReLU function
def sigmoid(x):
    return 1/(1+np.exp(-x))
def ReLU(x):
    return x * ( x > 0)
```

```python
p = random.randint(0, 1)
if p == 0:
    activ = ReLU
else:
    activ = sigmoid
```

Best Solution:
1.0
8
1
0
8
6
1
8
4
5
8
2
3
8
5
2
8
2
0
8
0
5
8
6
2
Best Solution depth: 72
Best Solution layers: [141, 197, 131, 183, 197, 131, 141, 197, 131, 183, 197, 131, 141, 197, 131, 141, 197, 131, 183, 197, 131, 183, 197, 141, 197, 197, 131, 141, 197, 131, 183, 197, 131, 183, 131, 183, 197, 141, 197, 197, 131, 141, 197, 131, 183, 197, 131, 183, 197, 183, 197,

```
131, 183, 197, 131, 141, 141, 141, 197, 141, 141, 141, 197, 131, 141,
[197, 131, 183, 197, 131, 131, 119
```

א - ע״י מקביליות

אלגוריתמים גנטיים הם אלגוריתמים שאפשר למקבל בקלות יחסית; במיוחד כשמדובר ברשתות נוירוניות לפיכך, שני סוגים של מקביליות אפשרית, מקביליות נתונים מקביליות מבוקרת. ההקבלה של אלגוריתמים גנטיים מביאה יתרונות רבים. הסיווג של אלגוריתמים אלה מבוסס לרוב על סוג מודל המחשוב, אסטרטגיית הליכה ומכונות המחשוב של המשתמש.

אנחנו יכולים פשוט לתת למעבדים מרובים ליצור אוכלוסיות משלהם ולמזג אותן מאוחר יותר במרחב משותף

ב - ע"י קשינג של תוצאות נשנות של חלקים דומים בין רשתות

אלגוריתמים גנטיים על רשתות נוירונים דורשים מספר רב של דגימות ממרחב הפתרונות. יש להעריך כל דגימה/פתרון באמצעות פונקציית fitness היעד המבצעת אופטימיזציה. חישובי fitness הם צוואר בקבוק מבחינת זמן בשיטות דגימה כמו אלגוריתמים גנטיים. caching של תוצאות חלקיות ממחשבי fitness אלו יכולה לשפר את זה.

ג - ע"י הקטנת השימוש בPOINT FLOATING וחישוב בשלמים במקום

ככל שפונקצית fitness מורכבת יותר וגודל האוכלוסייה גדול יותר, כך העלות החישובית גדולה יותר מבחינת floating point ככל שיהיה פחות שימוש ונרצה פחות דיוק אז העלות החישובית יורדת

ד - ע"י איתחול חכם של המשקולות

איכות האוכלוסייה הראשונית של פרטים משחקת תפקיד משמעותי בקביעת פתרון אופטימלי או כמעט אופטימלי ולכן איתחול חכם של משקולות רשתות נוירונים יכול לעזור מאוד בשיפור הפיטניס