AI Lab 5 part B

CoEvolution

Submitted BY :

Abed Abo Hussien 208517631

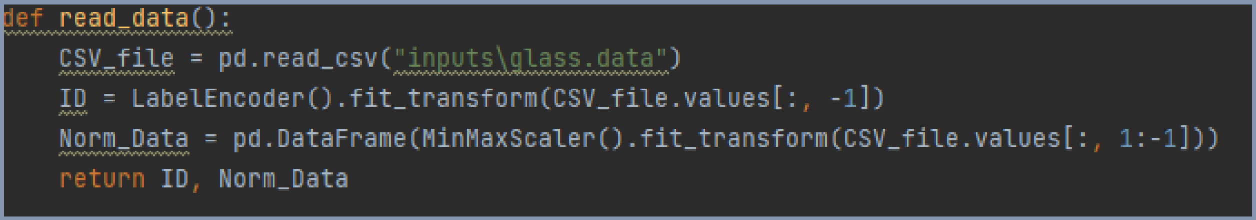
Samer Kharaoba 209050202

Section א :

In inputs folder we can find the glass.data folder

Section ב:

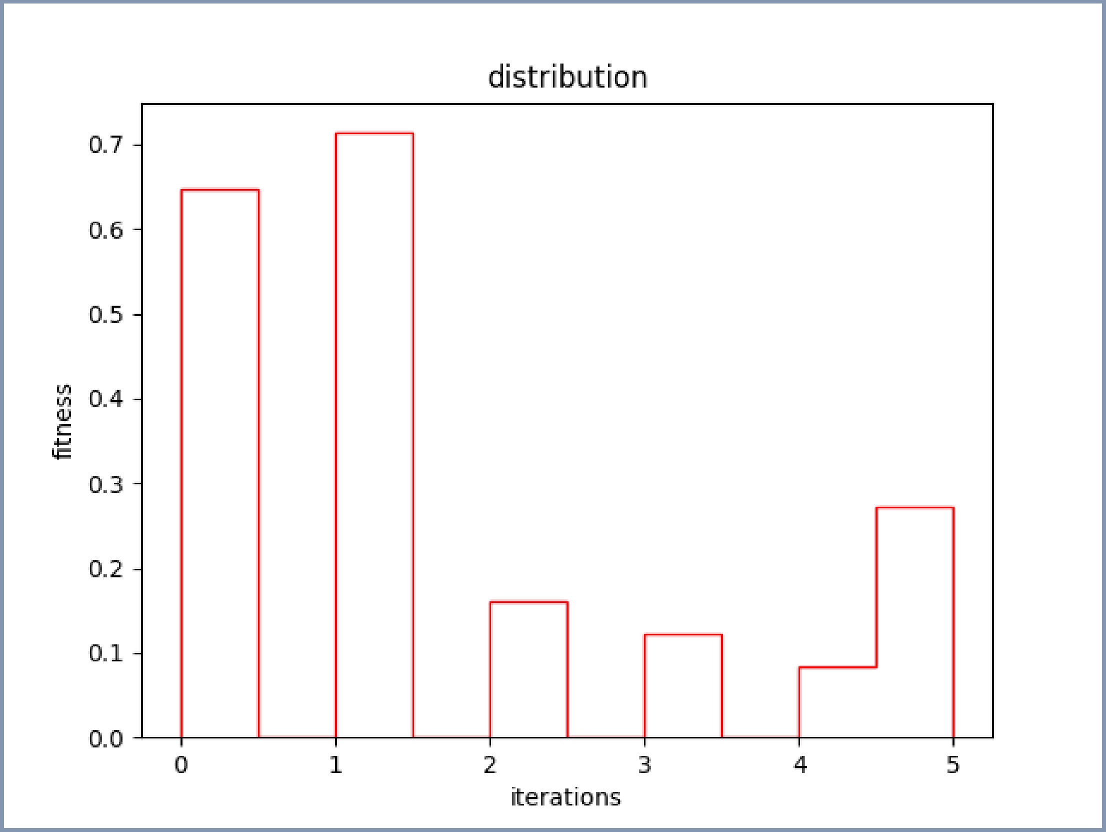
We used a built in function in pandas library :



```
def read_data():
    CSV_file = pd.read_csv("inputs\glass.data")
    ID = LabelEncoder().fit_transform(CSV_file.values[:, -1])
    Norm_Data = pd.DataFrame(MinMaxScaler().fit_transform(CSV_file.values[:, 1:-1]))
    return ID, Norm_Data
```

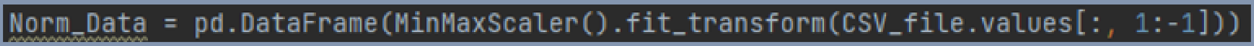
ID refers to the codes from 0-5

The given Distribution of the data is :



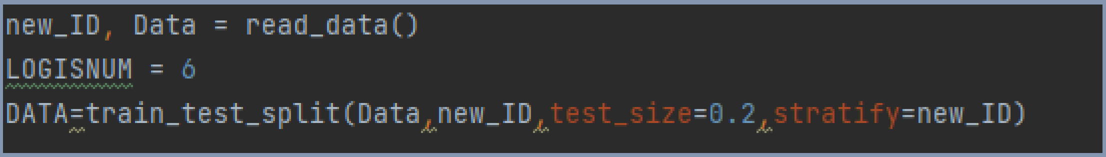
Section ד:

In read_data() function described above we used built in functions to normalize the data it’s called DataFrame of the Panda library



```
Norm_Data = pd.DataFrame(MinMaxScaler().fit_transform(CSV_file.values[:, 1:-1]))
```

Section ג:

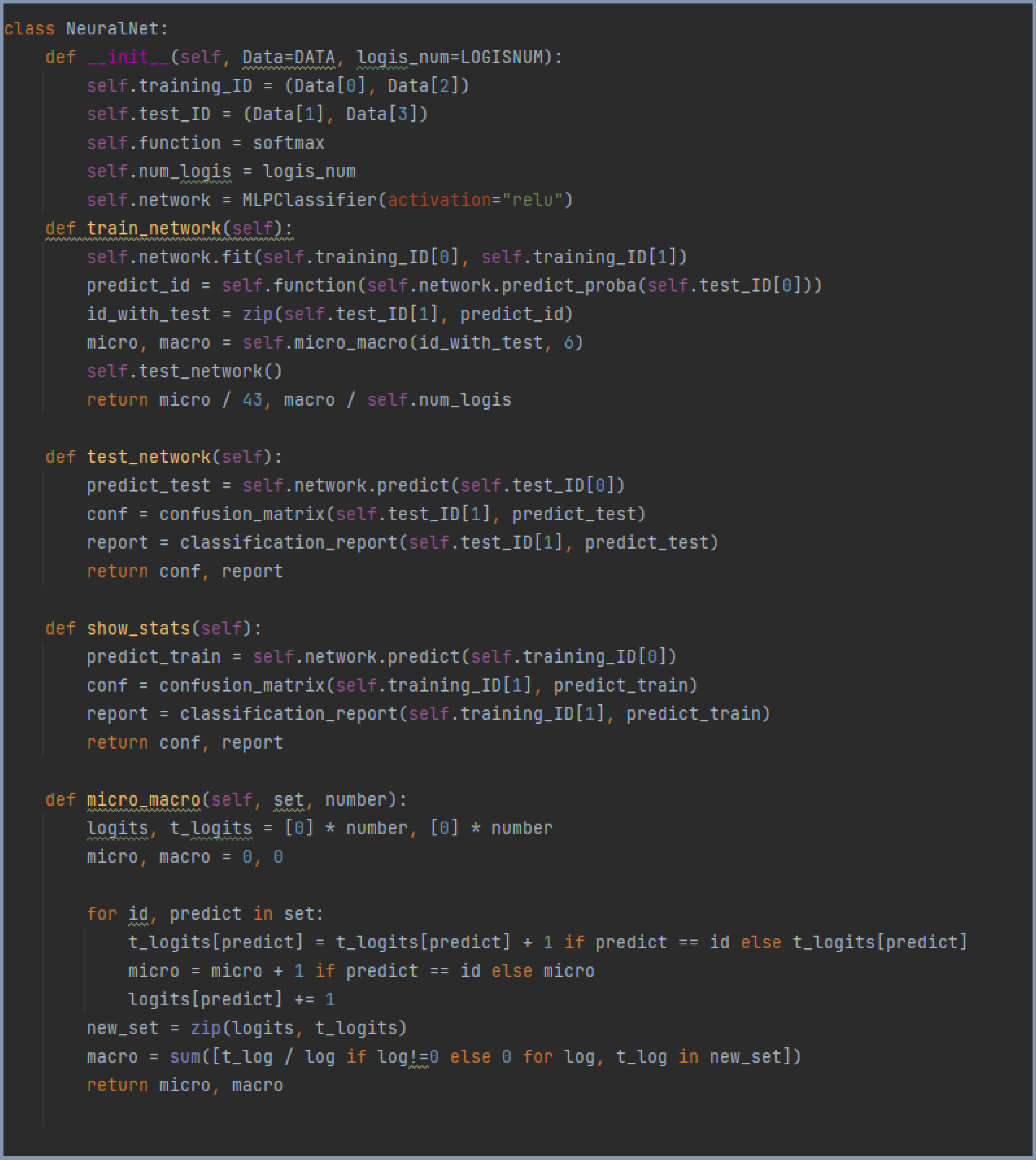


```
new_ID, Data = read_data()
LOGISNUM = 6
DATA=train_test_split(Data,new_ID,test_size=0.2,stratify=new_ID)
```

Again here we used a function from the Panda library that splits the data that we got from the CVS file into a training set and a test set , all were put in a variable named DATA

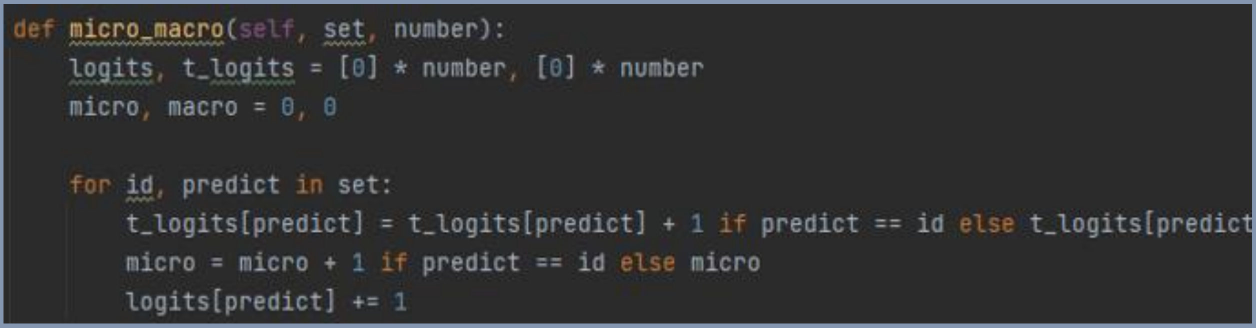
Section ה:

We built a class to create Neural Networks , we used a MLPClassifier that defines the basics of a Neural network and runs according to our wishes:



```python
class NeuralNet:
    def __init__(self, Data=DATA, logis_num=LOGISNUM):
        self.training_ID = (Data[0], Data[2])
        self.test_ID = (Data[1], Data[3])
        self.function = softmax
        self.num_logis = logis_num
        self.network = MLPClassifier(activation="relu")
    def train_network(self):
        self.network.fit(self.training_ID[0], self.training_ID[1])
        predict_id = self.function(self.network.predict_proba(self.test_ID[0]))
        id_with_test = zip(self.test_ID[1], predict_id)
        micro, macro = self.micro_macro(id_with_test, 6)
        self.test_network()
        return micro / 43, macro / self.num_logis

    def test_network(self):
        predict_test = self.network.predict(self.test_ID[0])
        conf = confusion_matrix(self.test_ID[1], predict_test)
        report = classification_report(self.test_ID[1], predict_test)
        return conf, report

    def show_stats(self):
        predict_train = self.network.predict(self.training_ID[0])
        conf = confusion_matrix(self.training_ID[1], predict_train)
        report = classification_report(self.training_ID[1], predict_train)
        return conf, report

    def micro_macro(self, set, number):
        logits, t_logits = [0] * number, [0] * number
        micro, macro = 0, 0

        for id, predict in set:
            t_logits[predict] = t_logits[predict] + 1 if predict == id else t_logits[predict]
            micro = micro + 1 if predict == id else micro
            logits[predict] += 1
        new_set = zip(logits, t_logits)
        macro = sum([t_log / log if log!=0 else 0 for log, t_log in new_set])
        return micro, macro
```

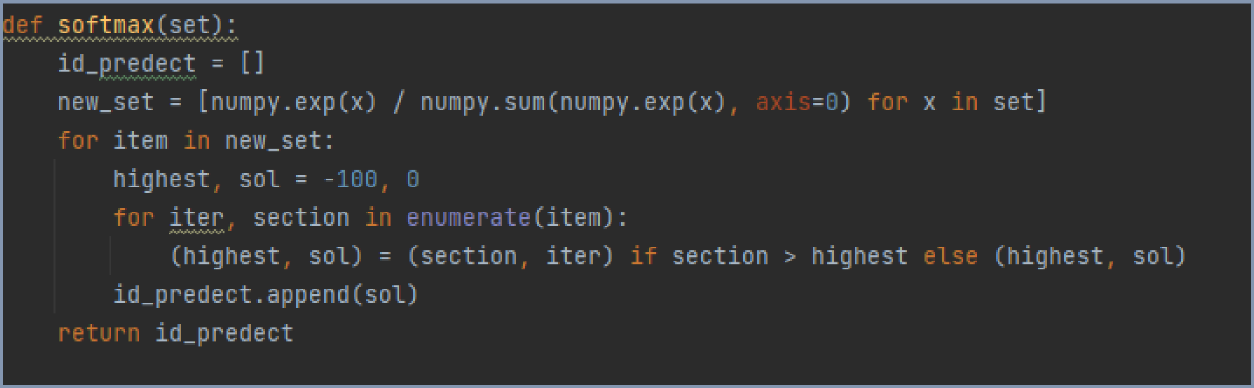
1. __init__(self,DATA,Logis_num):
    - This class gets Data (devided into 2 groups) and the number of logits that we would like it to have :
2. Test network:
    - Uses the test set on the network and returns a report on the outcome
3. Train network:
    - Uses the training set to train the network
4. Show stats :
    - Returns a report on the training set and it's confusion matrix
5. Micro macro:
    - Calculates macro and micro values of the softmax evaluation function
    - The 6 logits that are sent to the softmax function are in the micro macro function



```
def micro_macro(self, set, number):
    logits, t_logits = [0] * number, [0] * number
    micro, macro = 0, 0

    for id, predict in set:
        t_logits[predict] = t_logits[predict] + 1 if predict == id else t_logits[predict
        micro = micro + 1 if predict == id else micro
        logits[predict] += 1
```

Section ١:

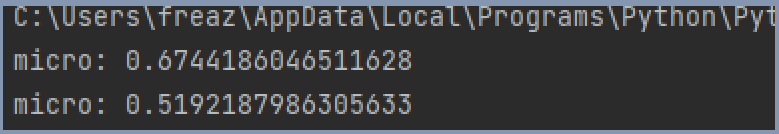
Firstly we created a matrix that contains all values of softmax function then worked on all of them to find the solution



```
def softmax(set):
    id_predect = []
    new_set = [numpy.exp(x) / numpy.sum(numpy.exp(x), axis=0) for x in set]
    for item in new_set:
        highest, sol = -100, 0
        for iter, section in enumerate(item):
            (highest, sol) = (section, iter) if section > highest else (highest, sol)
        id_predect.append(sol)
    return id_predect
```

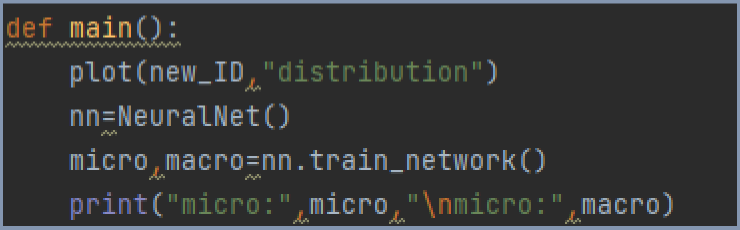
Section ٢ :

We tested on one Neural network described by the class above and the total results were :



```
C:\Users\freaz\AppData\Local\Programs\Python\Pyt
micro: 0.6744186046511628
micro: 0.5192187986305633
```

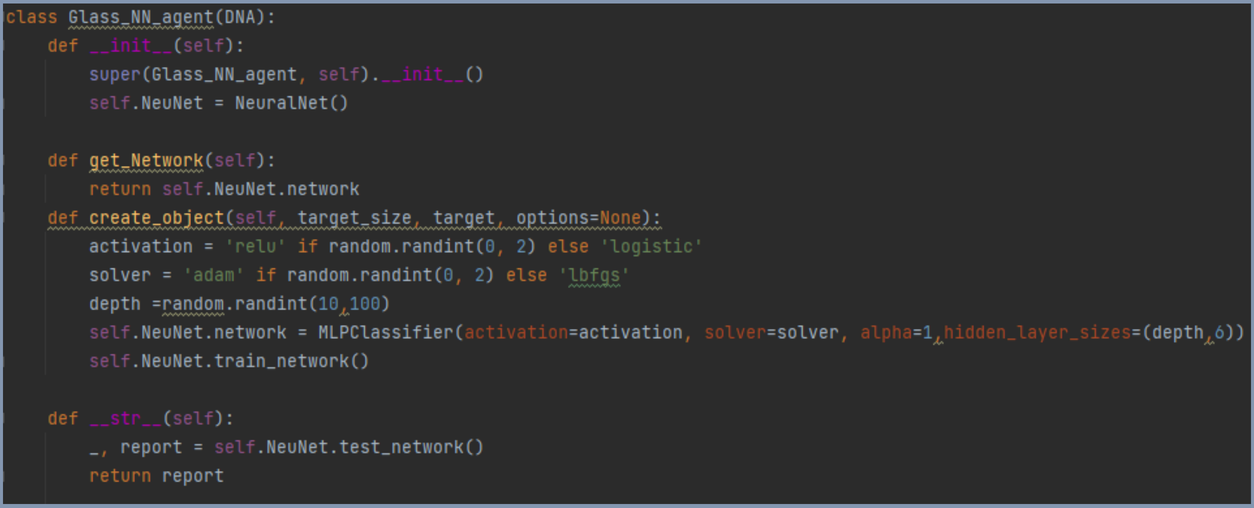
By using this simple code :



```
def main():
    plot(new_ID,"distribution")
    nn=NeuralNet()
    micro,macro=nn.train_network()
    print("micro:",micro,"\nmicro:",macro)
```

Section ח :

Firstly we created a new agent that has a Neural Network :



```
class Glass_NN_agent(DNA):
    def __init__(self):
        super(Glass_NN_agent, self).__init__()
        self.NeuNet = NeuralNet()

    def get_Network(self):
        return self.NeuNet.network
    def create_object(self, target_size, target, options=None):
        activation = 'relu' if random.randint(0, 2) else 'logistic'
        solver = 'adam' if random.randint(0, 2) else 'lbfgs'
        depth =random.randint(10,100)
        self.NeuNet.network = MLPClassifier(activation=activation, solver=solver, alpha=1,hidden_layer_sizes=(depth,6))
        self.NeuNet.train_network()

    def __str__(self):
        _, report = self.NeuNet.test_network()
        return report
```

- Note: DNA is a variation of our Agent

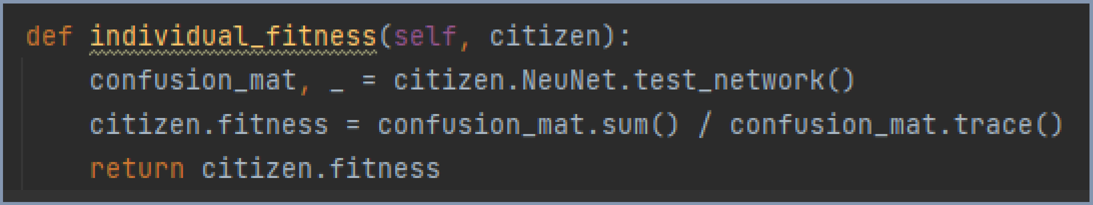
Here we made it so that the neural networks that are created in the population are different from one another :

1. By activation scheme
2. By hidden layer size ,that is decided by random and dictates the depth of the network
3. By the type of solver

Note : logistic returns the sigmoid function

4. Alpha gives the L2 penalty

Fitness function :



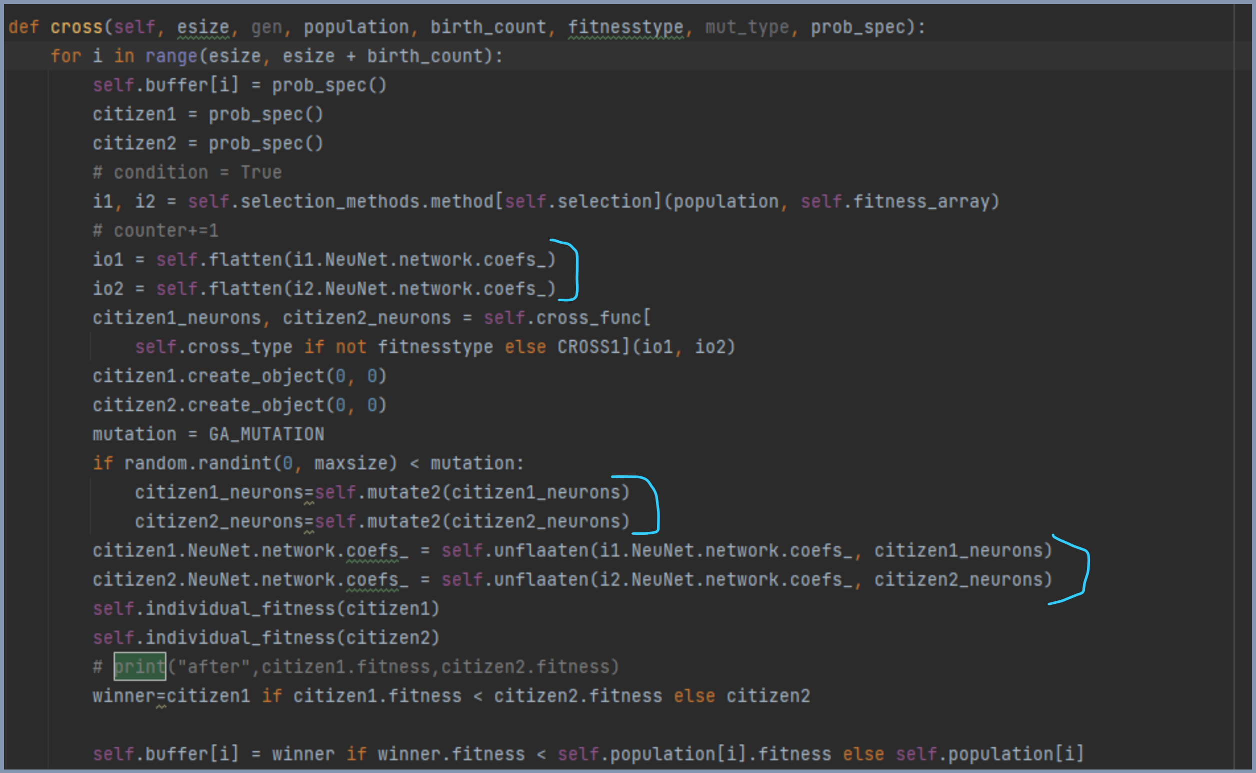
```python
def individual_fitness(self, citizen):
    confusion_mat, _ = citizen.NeuNet.test_network()
    citizen.fitness = confusion_mat.sum() / confusion_mat.trace()
    return citizen.fitness
```

We used the confusion mat to calculate the fitness so basically so that the alpha value that we put in the neural network would have an effect according to the f1 criteria .

As a reminder alpha is the penalty that we used that uses L2 normal on the weights

Sections ט&י :

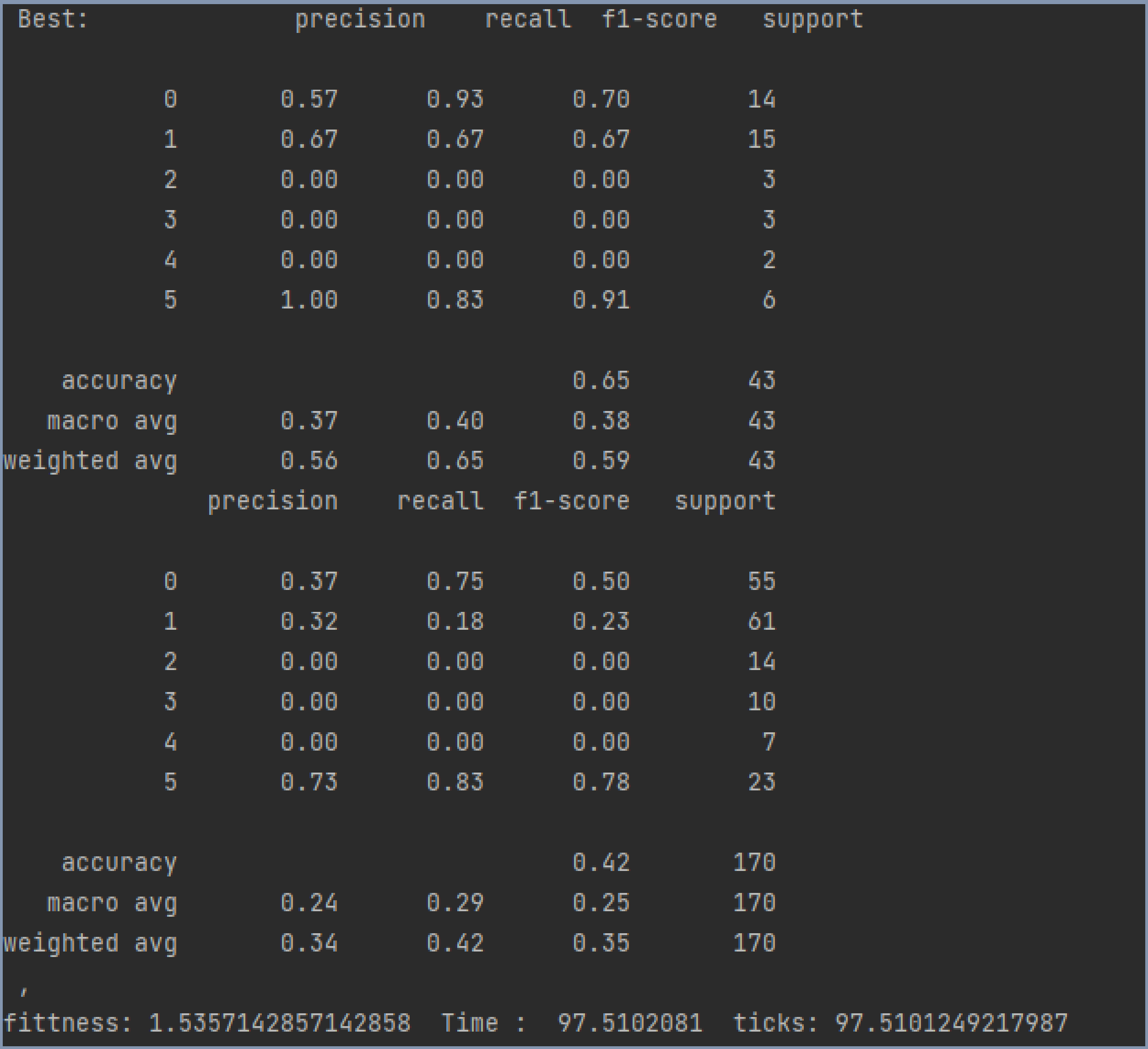
Firstly we tuned the cross function so that it manipulates the layers of the network:



```
def cross(self, esize, gen, population, birth_count, fitnesstype, mut_type, prob_spec):
    for i in range(esize, esize + birth_count):
        self.buffer[i] = prob_spec()
        citizen1 = prob_spec()
        citizen2 = prob_spec()
        # condition = True
        i1, i2 = self.selection_methods.method[self.selection](population, self.fitness_array)
        # counter+=1
        io1 = self.flatten(i1.NeuNet.network.coefs_)
        io2 = self.flatten(i2.NeuNet.network.coefs_)
        citizen1_neurons, citizen2_neurons = self.cross_func[
            self.cross_type if not fitnesstype else CROSS1](io1, io2)
        citizen1.create_object(0, 0)
        citizen2.create_object(0, 0)
        mutation = GA_MUTATION
        if random.randint(0, maxsize) < mutation:
            citizen1_neurons=self.mutate2(citizen1_neurons)
            citizen2_neurons=self.mutate2(citizen2_neurons)
        citizen1.NeuNet.network.coefs_ = self.unflaaten(i1.NeuNet.network.coefs_, citizen1_neurons)
        citizen2.NeuNet.network.coefs_ = self.unflaaten(i2.NeuNet.network.coefs_, citizen2_neurons)
        self.individual_fitness(citizen1)
        self.individual_fitness(citizen2)
        # print("after",citizen1.fitness,citizen2.fitness)
        winner=citizen1 if citizen1.fitness < citizen2.fitness else citizen2

        self.buffer[i] = winner if winner.fitness < self.population[i].fitness else self.population[i]
```

1. we used flatten to flatten the hidden layers (positions of the weights) that flattens the array of weights into one array
2. Used One point cross function that was Implemented before
3. Used Random Mutate to a random number
4. Unflattened the weights matrix (Called coefs_)
5. Calculated fitness for both new weights

The results :

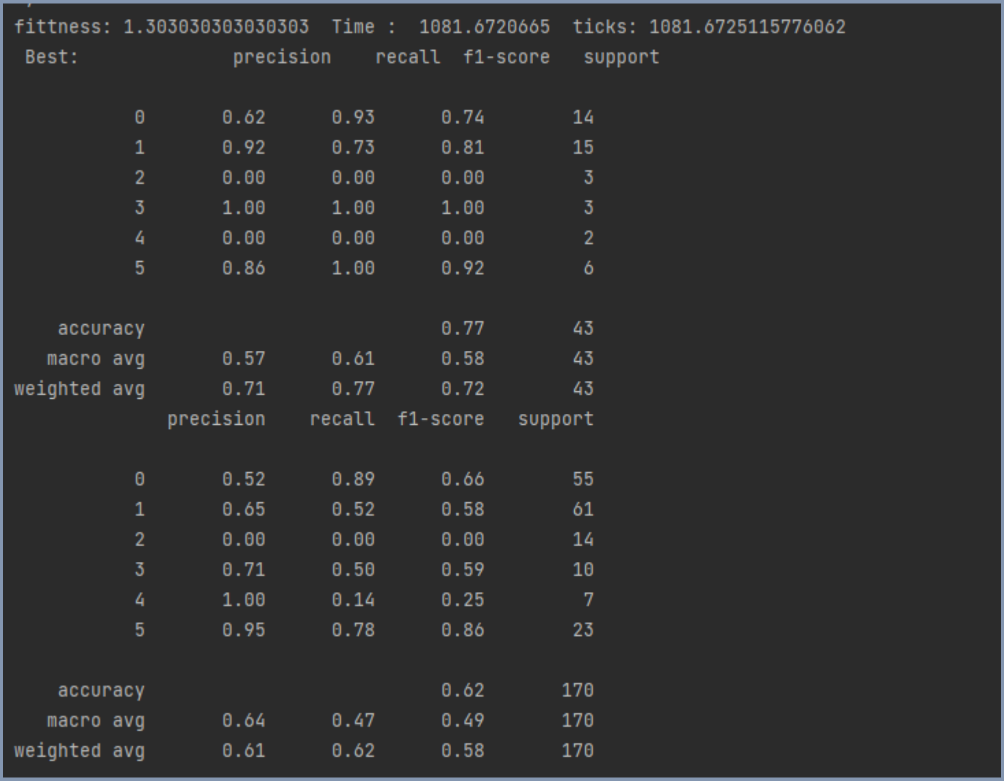


```
 Best:              precision    recall  f1-score   support

           0       0.57      0.93      0.70        14
           1       0.67      0.67      0.67        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       1.00      0.83      0.91         6

    accuracy                           0.65        43
   macro avg       0.37      0.40      0.38        43
weighted avg       0.56      0.65      0.59        43
              precision    recall  f1-score   support

           0       0.37      0.75      0.50        55
           1       0.32      0.18      0.23        61
           2       0.00      0.00      0.00        14
           3       0.00      0.00      0.00        10
           4       0.00      0.00      0.00         7
           5       0.73      0.83      0.78        23

    accuracy                           0.42       170
   macro avg       0.24      0.29      0.25       170
weighted avg       0.34      0.42      0.35       170
 ,
fittness: 1.5357142857142858  Time :  97.5102081  ticks: 97.5101249217987
```

The above one is the training set

The bottom one is the test set

Best individual :



```
fittness: 1.303030303030303  Time :  1081.6720665  ticks: 1081.6725115776062
 Best:              precision    recall  f1-score   support

           0       0.62      0.93      0.74        14
           1       0.92      0.73      0.81        15
           2       0.00      0.00      0.00         3
           3       1.00      1.00      1.00         3
           4       0.00      0.00      0.00         2
           5       0.86      1.00      0.92         6

    accuracy                           0.77        43
   macro avg       0.57      0.61      0.58        43
weighted avg       0.71      0.77      0.72        43
              precision    recall  f1-score   support

           0       0.52      0.89      0.66        55
           1       0.65      0.52      0.58        61
           2       0.00      0.00      0.00        14
           3       0.71      0.50      0.59        10
           4       1.00      0.14      0.25         7
           5       0.95      0.78      0.86        23

    accuracy                           0.62       170
   macro avg       0.64      0.47      0.49       170
weighted avg       0.61      0.62      0.58       170
```

Last section :

Parallelization :

The parallelization process would allow us to run more iterations and allows the evolution phase specifically the fitness calculation in those stages to run much faster as long as the memory overhead doesn't become too much as the parallelization process requires handling threads via a thread pool that can take a respectable amount of memory.

Caching snippets of similar networks:

the caching of snippets of a network can help reduce the time it takes to calculate the fitness for similar individuals as such we would have similar sections (fitness values) for those sections on both networks.

Using integers instead of floating points:

Using integers does indeed reduce the time of calculations , i.e. faster fitness calculations but using this method we would lose precision thus losing the whole point of Neural Networks .
so , in these tests we wouldn't recommend using integers.

Smart initialization of Weights:

This one doesn't help with the overhead from the fitness but would help us converge faster to a solution .thus we sure think that it would be a great to have feature in the algorithm.

Instruction :

Enter the values you are asked to enter

Final thoughts :

- Improve the 3 points alluded to in the last section

References :

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

https://python-course.eu/machine-learning/neural-networks-with-scikit.php

https://www.pluralsight.com/guides/machine-learning-neural-networks-scikit-learn