



**Πανεπιστήμιο Πειραιώς**  
**University of Piraeus**

**Πανεπιστήμιο Πειραιώς**

Σχολή Τεχνολογιών Πληροφορικής και Επικοινωνιών.

Τμήμα Πληροφορικής

**Εργασία μαθήματος**

Θέμα: Γενετικοί αλγόριθμοι και νευρωνικά δίκτυα

ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΚΑΙ ΕΜΠΕΙΡΑ ΣΥΣΤΗΜΑΤΑ

## Άσκηση 1

Για την επίλυση του προβλήματος χρησιμοποιήσαμε την γλώσσα προγραμματισμού python. Το πρόβλημα το προσεγγίσαμε ως εξής, ένας χρωματισμένος γράφος μπορεί να αναπαρασταθεί σε έναν πίνακα με 0 & 1, όπου 1 σημαίνει ότι το σημείο είναι χρωματισμένο και όπου 0 σημαίνει ότι το σημείο δεν είναι χρωματισμένο. Έτσι δημιουργήσαμε τους εξής πίνακες:

```
genetic_algorithm.py x
1 import random
2 import matplotlib.pyplot as plt
3
4 # κάθε κόμβος του grid pane αντιπροσωπεύεται από έναν αριθμό
5 # το 1 σημαίνει ότι ο κόμβος είναι χρωματισμένος και το 0 σημαίνει
6 # ότι δεν είναι χρωματισμένος.
7 N = [1, 1, 0, 0, 0, 0, 1,
8      1, 1, 0, 0, 0, 0, 1,
9      1, 1, 0, 0, 0, 0, 1,
10     1, 0, 1, 0, 0, 0, 1,
11     1, 0, 1, 0, 0, 0, 1,
12     1, 0, 0, 1, 0, 0, 1,
13     1, 0, 0, 1, 0, 0, 1,
14     1, 0, 0, 0, 1, 0, 1,
15     1, 0, 0, 0, 1, 0, 1,
16     1, 0, 0, 0, 0, 1, 1,
17     1, 0, 0, 0, 0, 1, 1]
18
19 B = [1, 1, 1, 1, 1, 0, 0,
20      1, 0, 0, 0, 0, 1, 0,
21      1, 0, 0, 0, 0, 1, 0,
22      1, 0, 0, 0, 0, 1, 0,
23      1, 0, 0, 0, 1, 0, 0,
24      1, 1, 1, 1, 0, 0, 0,
25      1, 0, 0, 0, 1, 0, 0,
26      1, 0, 0, 0, 0, 1, 0,
27      1, 0, 0, 0, 0, 1, 0,
28      1, 0, 0, 0, 0, 1, 0,
29      1, 1, 1, 1, 1, 0, 0]
30
31 K = [1, 0, 0, 0, 0, 0, 0, 0,
32      0, 0, 0, 0, 0, 0, 1, 0,
33      1, 0, 0, 0, 0, 0, 0, 0,
34      0, 0, 0, 1, 0, 0, 0, 0,
35      1, 0, 0, 0, 0, 0, 0, 0,
36      0, 1, 0, 0, 0, 0, 0, 0,
37      1, 0, 0, 0, 0, 0, 0, 1,
38      0, 0, 0, 1, 0, 0, 0, 0,
39      1, 0, 0, 0, 0, 0, 0, 0,
40      0, 0, 0, 0, 0, 0, 1, 0,
41      1, 0, 0, 0, 0, 0, 0, 0]
42
43 grid_pane = N
44
```

Βιβλιοθήκη για αναπαράσταση τυχαίων αριθμών

Βιβλιοθήκη για αναπαράσταση του γράμματος

Στη μεταβλητή gridpane βάζουμε ποιο γράμμα θέλουμε να λύσει ο αλγόριθμος

Χρησιμοποιήσαμε μονοδιάστατους πίνακες για λόγους ευκολίας, διότι σε δυσδιάστατους πίνακες ο αλγόριθμος θα αναγκαζόταν να κάνει περισσότερες επαναλήψεις οπότε θα καθυστέρουσε το πρόγραμμα σε χρόνο. Στο τέλος η λύση που βρίσκει ο αλγόριθμος μετατρέπεται σε δυσδιάστατο πίνακα (κάθε 7 στοιχεία του μονοδιάστατου πίνακα είναι μία γραμμή του δυσδιάστατου πίνακα), και αυτό γίνεται για λόγους αναπαράστασης του πίνακα σε γράφημα.

Στη συνέχεια έχουμε μία συνάρτηση που κάνει generate έναν τυχαίο πληθυσμό μήκους της επιλογής μας.

```
# μία συνάρτηση που επιστρέφει size
# πιθανές λύσεις του προβλήματος με τυχαίο τρόπο
# δηλαδή δημιουργεί έναν τυχαίο πληθυσμό
def generate_population(size):
    population = []
    for _ in range(size):
        population.append([random.choice([0, 1]) for _ in range(11 * 7)])

    return population
```

Η επόμενη συνάρτηση είναι η συνάρτηση fitness που τη χρησιμοποιούμε για να δούμε πόσο καλό είναι ένα χρωμόσωμα.

```
# μία συνάρτηση που επιστρέφει ένα value που αντιπροσωπεύει πόσο καλή είναι
# μία πιθανή λύση, όσο πιο μεγάλο το value τόσο πιο καλή και η λύση
def fitness(individual):
    value = 0

    for i in range(11 * 7):
        if individual[i] == grid_pane[i]:
            value += 1

    return value
```

Παίρνει σαν όρισμα ένα χρωμόσωμα και επιστρέφει την βαθμολογία του. Στη συγκεκριμένη περίπτωση όσο πιο πολλές αντιστοιχίες έχει ο πίνακας individual με τον πίνακα grid\_pane (που είναι το γράμμα που θέλουμε), η βαθμολογία του αυξάνεται κατά 1.

Η επόμενη συνάρτηση είναι μια συνάρτηση που παίρνει σαν όρισμα τον πληθυσμό και επιστρέφει τον πληθυσμό ταξινομημένο (σε αύξουσα σειρά) με βάση το fitness του.

```
# μεθοδος που επιστρέφει ταξινομημένο το population με βάση το fitness σε αύξουσα σειρά
def sort_population_by_fitness(population_):
    scores = []
    for p in population_:
        scores.append(fitness(p))
    return [x for _, x in sorted(zip(scores, population_))]
```

Η συνάρτηση tournament selection είναι μία συνάρτηση που επιλέγει το καλύτερο χρωμόσωμα (με βάση το fitness του) από ένα τυχαίο μέρος του πληθυσμού και το επιστρέφει σαν parent.

```
def tournament_selection(population_passed):
    individuals = []
    parent = []
    # επιλέγω 5 τυχαία διαφορετικά χρωμοσώματα
    for _ in range(5):
        # επιλογή χρωμοσώματος με τυχαίο τρόπο
        ind = population_passed[random.randrange(0, len(population_passed))]
        # έλεγχος για να μη μπουν στη λίστα ίδια χρωμοσώματα
        while ind in individuals:
            ind = population_passed[random.randrange(0, len(population_passed))]
        # προσθέτω το χρωμόσωμα στη λίστα
        individuals.append(ind)

    # για κάθε χρωμόσωμα που επιλέχτηκε διαλέγω αυτό με την καλύτερη βαθμολογία
    for ind in individuals:
        # το len(parent) == 0 το βάζουμε για να μπει μέσα την πρώτη φορά
        if len(parent) == 0 or fitness(ind) > fitness(parent):
            parent = ind

    return parent
```

Η συνάρτηση single point crossover δέχεται σαν όρισμα 2 γονείς, και επιστρέφει δύο χρωμοσώματα που προέκυψαν από αυτούς τους γονείς

```
# μέθοδος που επιστρέφει δύο παιδιά
def single_point_crossover(individual_a, individual_b):
    # τυχαία επιλογή του σημείου που θα γίνει το crossover
    point = random.randint(1, len(individual_a) - 1)

    # δημιουργία νέων απογόνων
    for i in range(point, len(individual_a)):
        individual_a[i], individual_b[i] = individual_b[i], individual_a[i]

    return [individual_a, individual_b]
```

Η συνάρτηση `make_next_generation` δέχεται σαν όρισμα έναν πληθυσμό και επιστρέφει έναν καινούργιο πληθυσμό.

```
# μέθοδος που δημιουργεί μία νέα γενιά
def make_next_generation(previous_population):
    next_generation = []

    for _ in range(len(previous_population)):
        # επιλογή δύο γονέων
        parent1 = tournament_selection(previous_population)
        parent2 = tournament_selection(previous_population)
        # ελέγχουμε ότι οι γονείς δεν αποτελούν το ίδιο χρωμόσωμα,
        # διαφορετικά δεν έχει νόημα να γίνει το crossover
        while parent1 == parent2:
            parent1 = tournament_selection(previous_population)
            parent2 = tournament_selection(previous_population)
        # δημιουργία απογόνων
        child1 = single_point_crossover(parent1, parent2)[0]
        child2 = single_point_crossover(parent1, parent2)[1]
        # επιλέγουμε το πιο δυνατό παιδί
        if fitness(child1) > fitness(child2):
            next_generation.append(child1)
        else:
            next_generation.append(child2)

    return next_generation
```

Από τα δύο παιδιά που προκύπτουν έχουμε επιλέξει να κρατάμε το πιο "ικανό".

Η συνάρτηση `fitness all population` επιστρέφει μία λίστα με τις βαθμολογίες όλου του πληθυσμού, δηλαδή το πρώτο στοιχείο της λίστας είναι η βαθμολογία του πρώτου χρωμοσώματος του πληθυσμού.

```
# μας επιστρέφει την βαθμολογία μιας γενιάς
def fitness_all_population(population_):
    scores = []
    # προσθέτουμε τις βαθμολογίες όλων των χρωμοσωμάτων μιας γενιάς
    for p in population_:
        scores.append(fitness(p))
    return scores
```

Στη συνέχεια έχουμε το κυρίως πρόγραμμα

```
# πόσες γενιές θα δημιουργήσουμε
generations = 60
# πόσος θα είναι ο πληθυσμός σε κάθε γενιά
population_size = 10000
# δημιουργία πληθυσμού
population = generate_population(population_size)
find = False

for i in range(generations):
    print(f"=====GENERATION {i + 1} ===== FITNESS: {sum(fitness_all_population(population))}===== ")

    # αν δημιουργηθούν όλες οι γενιές σταματάμε το loop
    if i == generations:
        break
    # ψάχνουμε αν βρέθηκε η λύση
    for p in population:
        # αν το fitness του p είναι 77 τότε σημαίνει ότι
        # βρέθηκε ένας πίνακας που είναι ακριβώς ίδιος με τον
        # πίνακα που ψάχνουμε
        if fitness(p) == 77:
            find = True
            break
    # αν βρεθεί η λύση σταματάμε το loop
    if find:
        break

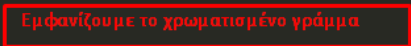
    # δημιουργία νέας γενιάς
    population = make_next_generation(population)

# παίρνουμε το καλύτερο χρωμόσωμα της γενιάς το οποίο βρίσκεται στο τέλος της λίστας
# καθώς είναι ταξινομημένη σε αύξουσα σειρά
best_individual = sort_population_by_fitness(population)[-1]
print(f"\nFINAL RESULT FITNESS: {fitness(best_individual)}")

# εκτυπώνουμε τον πίνακα σε κατάλληλη μορφή
cnt = 0
row = []
best_individual_2d = []
for i in range(len(best_individual)):
    cnt += 1
    row.append(best_individual[i])
    if cnt % 7 == 0:
        cnt = 0
        print(row)
        best_individual_2d.append(row[:])
        row.clear()

print("Done!")

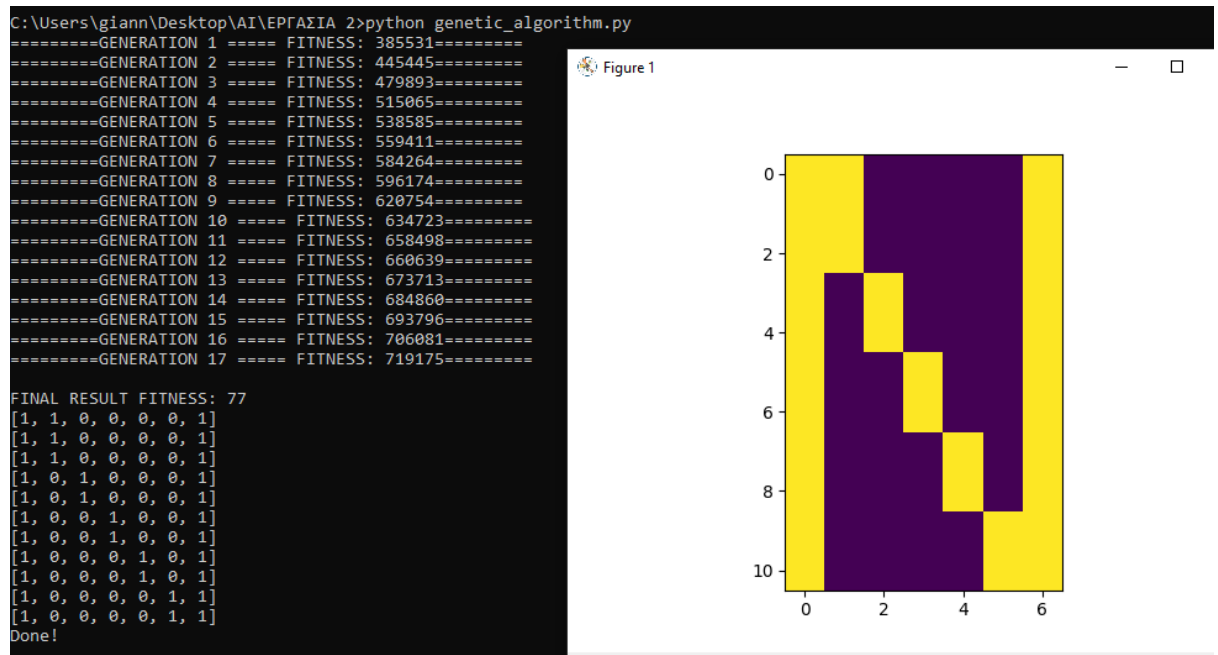
plt.imshow(best_individual_2d)
plt.show()
```



Στην αρχή δηλώνουμε πόσες γενιές θέλουμε να δημιουργήσουμε και τι πληθυσμό θα έχουν. Στην συνέχεια κάνουμε ένα loop, κάθε φορά εκτυπώνουμε σε ποια γενιά βρισκόμαστε και ποια είναι η συνολική βαθμολογία της γενιάς. Το loop θα σταματήσει αν βρεθεί λύση ή αν δημιουργηθούν όλες οι γενιές.

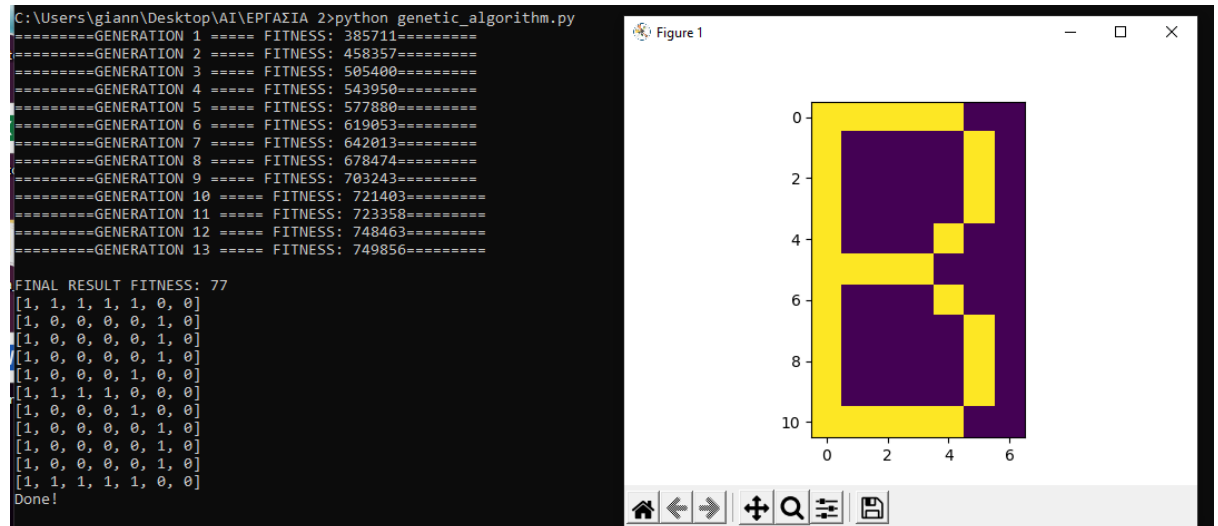
Μετά το loop παίρνουμε το καλύτερο χρωμόσωμα, το εκτυπώνουμε μαζί με την βαθμολογία του και μετά το μετατρέπουμε σε δυσδιάστατο πίνακα, αφού δημιουργηθεί ο δυσδιάστατος πίνακας εμφανίζουμε το γράφημα.

Τώρα τρέχοντας αυτό το πρόγραμμα για το γράμμα N θα πάρουμε το ακόλουθο αποτέλεσμα:



Χρειάστηκαν να δημιουργηθούν 17 γενιές για να βρει τη λύση.

Τώρα το τρέχουμε για το γράμμα B.



Χρειάστηκαν 13 γενιές να βρει τη λύση.

## Άσκηση 2

Πρώτο βήμα για την λύση αυτής της άσκησης ήταν να βρούμε ένα κατάλληλο dataset με εικόνες από χειρόγραφα γράμματα. Αυτό που βρήκαμε είναι το handwritten dataset που περιέχει πάνω από 300 χιλιάδες δεδομένα με εικόνες 28 \* 28 pixel με γράμματα λατινικού αλφαβήτου.

Link of dataset: [A-Z Handwritten Alphabets in .csv format | Kaggle](#)

Το συγκεκριμένο dataset στην πρώτη στήλη έχει κωδικοποιημένα τα γράμματα με νούμερα 0-25, με 0=A, 1=B κτλ. Στις επόμενες στήλες υπάρχουν οι πληροφορίες για τα pixel της εικόνας.

Γνωρίζοντας πως είναι στημένα τα δεδομένα, ξεκινήσαμε την διαδικασία τακτοποιώντας τα στο πρόγραμμα μας.

```
genetic_algorithm.py  x  nn.py  x
1 import tensorflow as tf # θα μας βοηθήσει για να δημιουργήσουμε το neural network
2 import pandas as pd     # θα μας βοηθήσει για το διάβασμα δεδομένων και την επεξεργασία τους
3 from sklearn.model_selection import train_test_split # θα μας βοηθήσει για να χωρίζουμε τα δεδομένα
4
5 # διαβάζουμε τα δεδομένα μας από τα αρχεία
6 df = pd.read_csv('Handwritten Data.csv')
7 # ξέρουμε ότι η πρώτη στήλη είναι οι αριθμοί από 0-25 που
8 # κωδικοποιούν το λατινικό αλφάβητο (δηλ. 0=A, 1=B κτλ)
9 # τα αποθηκεύουμε σε μία μεταβλητή x, και στοιχεία για κάθε γράμμα
10 # τα αποθηκεύουμε σε μία μεταβλητή y
11 x, y = df.drop(columns=['0']), df['0']
```

Έτσι στη μεταβλητή x έχουμε τις πληροφορίες της εικόνας και στη μεταβλητή y έχουμε τα κωδικοποιημένα γράμματα (νούμερα από 0-25).

Στη συνέχεια χωρίσαμε τα δεδομένα σε train letters που είναι τα δεδομένα με τα οποία θα κάνουμε train το μοντέλο, σε train labels που είναι τα κωδικοποιημένα γράμματα του πίνακα train letters, σε test letters που είναι τα δεδομένα με τα οποία θα τεστάρουμε το μοντέλο και σε test labels.

```
13 # χωρίζουμε τα δεδομένα σε train_letters και test_letters
14 # τα label περιέχουν αριθμούς από 0-25, για κάθε γράμμα της αλφαβήτου
15 # και τα train_letters/test_letters περιέχουν πληροφορίες για φωτογραφίες των 28*28 pixels
16 train_letters, test_letters, train_labels, test_labels = train_test_split(x, y)
```

Στη συγκεκριμένη μέθοδο το default split για τα test\_letters είναι 25% των συνολικών δεδομένων.

Στη συνέχεια θα πρέπει να επεξεργαστούμε τα train letters και τα test letters έτσι ώστε οι τιμές τους να είναι ανάμεσα σε 0 και 1, διαφορετικά



το μοντέλο που θα δημιουργήσουμε δε θα είναι σε θέση να προπονηθεί. Γνωρίζοντας ότι μια τιμή του pixel μπορεί να είναι από 0 έως 255, διαιρούμε τις μεταβλητές με το 255.

```
18 # επεξεργασία δεδομένων
19 # για να προπονήσουμε ή να τεστάρουμε τα data, θα πρέπει τα δεδομένα μας
20 # να αναπαριστούνται σε νούμερα από 0 έως 1. Κάθε pixel ξέρουμε ότι αναπαριστάται από
21 # ένα νούμερο 0-255 οπότε κάθε νούμερο το διαιρούμε με 255 έτσι ώστε να πάρουμε ένα νούμερο από 0-1
22 train_letters = train_letters / 255.0
23 test_letters = test_letters / 255.0
```

Στη συνέχεια δημιουργούμε την αρχιτεκτονική του μοντέλου μας

```
25 # κατασκευή μοντέλου
26 # χρησιμοποιούμε ένα sequential μοντέλο διότι τα sequential μοντέλα
27 # είναι κατάλληλα όταν κάθε layer έχει μία είσοδο (στη συγκεκριμένη περίπτωση εικόνα)
28 # και μία έξοδο (στη συγκεκριμένη περίπτωση ένα γράμμα)
29 model = tf.keras.Sequential([
30     # Για input layer χρησιμοποιούμε το flatten layer με input shape
31     # τα pixel της εικόνας (28*28).
32     tf.keras.layers.Flatten(input_shape=(28, 28)), # input layer (1)
33     # το dense layer ή full connected layer είναι ένα κρυφό στρώμα που έχουμε
34     # επιλέξει πως θα έχει 200 νευρώνες και για activation function να έχει τη relu
35     tf.keras.layers.Dense(200, activation='relu'), # hidden layer (2)
36     # το output layer είναι και αυτό dense, έχει 26 νευρώνες (έναν για κάθε
37     # γράμμα της αλφαβήτου) και έχουμε επιλέξει για activation function την softmax
38     tf.keras.layers.Dense(26, activation='softmax') # output layer (3)
39 ])
```

Το κάνουμε compile

```
41 # compile μοντέλου
42 # για optimizer χρησιμοποιούμε τον adam, και για υπολογισμό του loss του
43 # μοντέλου μας χρησιμοποιούμε τη μέθοδο sparse_categorical_crossentropy
44 model.compile(optimizer='adam',
45               loss='sparse_categorical_crossentropy',
46               metrics=['accuracy'])
```

Το προπονούμε

```
48 # εκπαιδεύουμε το μοντέλο μας πάνω στα δεδομένα μας,
49 # το εκπαιδεύουμε 2 φορές
50 model.fit(train_letters, train_labels, epochs=2)
```

Το τεστάρουμε

```
52 # υπολογίζουμε το loss και το accuracy πάνω στα test data που κρατήσαμε
53 loss, accuracy = model.evaluate(test_letters, test_labels)
54 print(f"loss: {loss}")
55 print(f"accuracy: {accuracy}")
```

Και το αποθηκεύουμε για να το τεστάρουμε σε δικές μας εικόνες που σχεδιάσαμε με το paint

Screenshots από τη κατασκευή του μοντέλου

Ξεκινάει το πρώτο epoch

Epoch 1/2

```
WARNING:tensorflow:Model was constructed with shape (None, 28, 28) for input KerasTensor(type_s
WARNING:tensorflow:Model was constructed with shape (None, 28, 28) for input KerasTensor(type_s
4613/8730 [=====>.....] - ETA: 7s - loss: 0.5679 - accuracy: 0.8443
```

Epoch 1/2

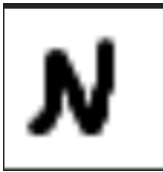
```
WARNING:tensorflow:Model was constructed with shape (None, 28, 28) for input KerasTensor(type_spec=TensorSpec(shape=(None, 28, 28), dtype=tf.float32, name='flatten_
WARNING:tensorflow:Model was constructed with shape (None, 28, 28) for input KerasTensor(type_spec=TensorSpec(shape=(None, 28, 28), dtype=tf.float32, name='flatten_
8730/8730 [=====] - 16s 2ms/step - loss: 0.4353 - accuracy: 0.8808 ← ΠΡΩΤΟ ΕPOCH
Epoch 2/2
8730/8730 [=====] - 14s 2ms/step - loss: 0.1174 - accuracy: 0.9670 ← ΔΕΥΤΕΡΟ ΕPOCH
WARNING:tensorflow:Model was constructed with shape (None, 28, 28) for input KerasTensor(type_spec=TensorSpec(shape=(None, 28, 28), dtype=tf.float32, name='flatten_
2910/2910 [=====] - 3s 1ms/step - loss: 0.1068 - accuracy: 0.9701 ← ΤΕΣΤΑΡΙΣΜΑ MONTEADY
loss: 0.10678169131278992
accuracy: 0.9700579047203064
2021-05-01 18:10:21.524091: W tensorflow/python/util/util.cc:348] Sets are not currently considered sequences, but this may change in the future, so consider avoidi
Process finished with exit code 0
```

Πρόγραμμα για τεστάρισμα σε φωτογραφίες των γραμμάτων N και B:

```
genetic_algorithm.py x nn.py test.py x
1 import numpy as np
2 import tensorflow as tf
3 import cv2
4
5 # δημιουργούμε ένα λεξικό με τα γράμματα των επιθέτων μας
6 surname_letters = {1: 'B',
7                    13: 'N',
8                    }
9
10 # κάνουμε load το μοντέλο που δημιουργήσαμε
11 model = tf.keras.models.load_model('recognize_letters.model')
12
13 # στο input βάζουμε το όνομα της εικόνας
14 input_img = 'N2.png'
15 # φορτώνουμε την εικόνα σε κατάλληλη μορφή (gray scale)
16 img = cv2.imread(input_img)[: , : , 0]
17
18 # μετατρέπουμε την εικόνα σε array πίνακα για να μπορούμε
19 # να την εισάγουμε στο μοντέλο μας
20 img2 = np.invert(np.array([img]))
21
22 # κανουμε την πρόβλεψη και την εκτυπώνουμε
23 prediction = model.predict(img2)
24
25 # αν ο αριθμός υπάρχει στο λεξικό εκτυπώνουμε την αντίστοιχη τιμή του λεξικού
26 # διαφορετικά εκτυπώνουμε το κωδικοποιημένο γράμμα
27 if int(np.argmax(prediction)) in surname_letters.keys():
28     print(f"Input: {input_img}\nPrediction: {surname_letters[int(np.argmax(prediction))]}")
29 else:
30     print(f"Input: {input_img}\nPrediction: {np.argmax(prediction)}")
```

Screenshot αποτελέσματος:

Εικόνα N1.png



```
2021-05-01 18:20:19.073891: I tensorflow/compiler,
Input: N1.png
Prediction: N

Process finished with exit code 0
```

Εικόνα B1.png



```
2021-05-01 18:22:55.831964: I tensorflow/compiler,
Input: B1.png
Prediction: B

Process finished with exit code 0
```

Εικόνα N2.png



```
2021-05-01 18:30:39.495991:
Input: N2.png
Prediction: N
```

Εικόνα B2.png



```
2021-05-01 18:34:53.236536: I tensorflow
Input: B2.jpg
Prediction: B

Process finished with exit code 0
```

Ωστόσο όπως είναι και αναμενόμενο, κάποιες φορές το μοντέλο μας κάνει λάθη, για παράδειγμα για την εικόνα N11.png



```
2021-05-01 18:38:18.062989: I tensorflow
Input: N11.png
Prediction: 0
```

Την αναγνωρίζει σαν A.