ΠΑΡΑΛΛΗΛΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ ΠΡΟΒΛΗΜΑΤΑ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

4η Εργαστηριακή Άσκηση

Εκπαίδευση Νευρωνικών Δικτύων με τον Αλγόριθμο Οπισθοδρομικής Διάδοσης του Σφάλματος (Error backpropagation)

Ον/μο: Γεώργιος Σκόνδρας

Τμήμα: Μηχανικών Η/Υ και Πληροφορικής

Α.Μ: 1020408

Σημείωση

Για την υλοποίηση της άσκησης χρησιμοποιήθηκε προσωπικός φορητός υπολογιστής Huawei Matebook d14 με τα παρακάτω χαρακτηριστικά

- *Επεξεργαστής: AMD Ryzen 3500U*
- *Λειτουργικό Σύστημα: (μέσω Oracle VM Virtual Box 6.1) Ubuntu 20.04.1 LTS*
- *Compiler: gcc 9.3.0 (Ubuntu 9.3.0-17ubuntu1~20.04)*

Εφόσον χρησιμοποιήθηκε Virtual Machine για την εκπόνηση της άσκησης, δώσαμε 2 φυσικούς πυρήνες που αντιστοιχούν σε 4 thread(Simultaneous Multi Threading)

*Οι κώδικες για την κάθε εργασία βρίσκονται στα αρχεία *.c στους φακέλους ex1,ex2 κλπ*

Εργασία 1

Για να γίνει δυνατή η ενεργοποίηση του Νευρωνικού Δικτύου(Ν.Δ.) χρειάστηκε εκτός από τη ζητούμενη συνάρτηση **activateNN()** να υλοποιηθούν και οι παρακάτω συναρτήσεις

- **void initializeNN()**: Συνάρτηση που αρχικοποιεί τα βάρη του Ν.Δ. με τυχαίες τιμές στο διάστημα [-1,1] και ορίζει τους σταθερούς όρους του bias με την τιμή 1.
- **void initializeInput()**: Συνάρτηση που αρχικοποιεί τις εισόδους του Νευρωνικού Δικτύου με τυχαίες τιμές στο διάστημα [-1,1] και με 1 τον σταθερό όρο του bias.
- **void printWeights()**: Βοηθητική συνάρτηση που τυπώνει τα βάρη του Ν.Δ. για κάθε επίπεδο.
- **void printNNInfo()**: Βοηθητική συνάρτηση που τυπώνει την είσοδο, τις εσωτερικές καταστάσεις και τις εξόδους του κάθε επιπέδου του Ν.Δ.

Η συνάρτηση **void activateNN(double *Vector)** είναι η συνάρτηση που παίρνει σαν παράμετρο το διάνυσμα της εισόδου Vector και υπολογίζει την αριθμητική τιμή των εσωτερικών καταστάσεων και της εξόδου όλων των νευρώνων του δικτύου. Αυτό γίνεται υπολογίζοντας κατά την σειρά των επιπέδων την εσωτερική κατάσταση κάθε νευρώνα και στη συνέχεια την έξοδο με τη χρήση σιγμοειδούς συνάρτησης.

Πιο συγκεκριμένα η έξοδος κάθε νευρώνα υπολογίζεται ως εξής:

- Αρχικά υπολογίζεται η εσωτερική κατάσταση του νευρώνα i σύμφωνα με τη σχέση
  $Internal\_State(i) = \vec{w} \cdot \vec{x} + b$
  $όπου\ \vec{x}\ το\ διάνυσμα\ εισόδων\ του\ νευρώνα\ i, \vec{w}\ το\ αντίστοιχο\ διάνυσμα\ βαρών\ και\ b\ το\ bias$
- Στη συνέχεια υπολογίζεται η έξοδος του νευρώνα χρησιμοποιώντας τη σιγμοειδή συνάρτηση $\sigma(x) = \frac{1}{1+e^{-x}}$ όπως φαίνεται παρακάτω:

$$Output(i) = \sigma\left(Internal_{State(i)}\right)$$
$$Output(i) = \sigma(\vec{w} \cdot \vec{x} + b)$$

Επιβεβαιώνουμε την καλή λειτουργία της συνάρτησης **activateNN()** για ένα δίκτυο με 2 νευρώνες στο πρώτο επίπεδο και δύο στο επίπεδο εξόδου με εισόδους και βάρη τυχαίες τιμές στο διάστημα [-1,1] όπως φαίνεται παρακάτω.

```
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex1$ ./ex1
Level 1 weights

Neuron 0
W0X0: 1.000000
W0X1: 0.566198
W0X2: 0.596880

Neuron 1
W1X0: 1.000000
W1X1: 0.823295
W1X2: -0.604897

Level 2 weights

Neuron 0
W0X0: 1.000000
W0X1: -0.329554
W0X2: 0.536459

Neuron 1
W1X0: 1.000000
W1X1: -0.444451
W1X2: 0.107940



**** Neural Network Activation ****

Input vector

X0: 1.000000
X1: 0.680375
X2: -0.211234

Internal Neuron State

Level 1 Internal State
Neuron 0: 1.259146
Neuron 1: 1.687925

Level 2 Internal State
Neuron 0: 1.196062
Neuron 1: 0.744923
Neuron Outputs

Level 1 Outputs
Neuron 0: 1.00000000000000000000
Neuron 1: 0.77887907588492943933
Neuron 2: 0.84395101740470002127

Level 2 Outputs
Neuron 0: 0.76782352856690738729
Neuron 1: 0.67807139743693056744
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex1$
```

Εργασία 2

Για την υλοποίηση του αλγορίθμου Error Back Propagation χρησιμοποιήθηκαν οι παρακάτω πίνακες:

- **double INPUT[INPUT_SIZE+1]** : Το διάνυσμα εισόδου, 12 τιμές +1 για το bias
- **double DESIRED_OUTPUT[LEVEL_2_NEURONS]** : Το διάνυσμα επιθυμητής εξόδου, μεγέθους, όσοι και οι νευρώνες στο τελευταίο επίπεδο
- **double WL1[LEVEL_1_NEURONS][INPUT_SIZE+1]** : Ο πίνακας με τα βάρη του πρώτου επιπέδου
- **double WL2[LEVEL_2_NEURONS][LEVEL_1_NEURONS+1]** : Ο πίνακας με τα βάρη του δεύτερου επιπέδου +1 διάσταση για τα βάρη του bias
- **double DL1[LEVEL_1_NEURONS]** : Το διάνυσμα εσωτερικών καταστάσεων του πρώτου επιπέδου
- **double DL2[LEVEL_2_NEURONS]** : Το διάνυσμα εσωτερικών καταστάσεων του δευτέρου επιπέδου
- **double OL1[LEVEL_1_NEURONS+1]** : Το διάνυσμα εξόδων του πρώτου επιπέδου +1 στοιχείο για το bias
- **double OL2[LEVEL_2_NEURONS]** : Το διάνυσμα εξόδων του δεύτερου επιπέδου

- **double D2[LEVEL_2_NEURONS]** : Το διάνυσμα με τις τιμές του δέλτα του δεύτερου επιπέδου
- **double D1[LEVEL_1_NEURONS]** : Το διάνυσμα με τις τιμές του δέλτα του πρώτου επιπέδου

Επιπλέον χρησιμοποιήθηκαν οι παρακάτω βοηθητικές συναρτήσεις:

- **void initializeInputOutput()** : Συνάρτηση που αρχικοποιεί το διάνυσμα εισόδου και επιθυμητής εξόδου
- **double sigmoid(double x)** : Συνάρτηση που υπολογίζει τη σιγμοειδή συνάρτηση σύμφωνα με τον τύπο $\sigma(x) = \frac{1}{1+e^{-x}}$
- **double mse(double *a, double *b, int size)** : Συνάρτηση που υπολογίζει το μέσο τετραγωνικό σφάλμα ανάμεσα στο διάνυσμα εξόδου του Νευρωνικού Δικτύου και στο επιθυμητό διάνυσμα εξόδου σύμφωνα με τον τύπο:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

- **void printOutputs()** : Συνάρτηση που τυπώνει τις εξόδους του Νευρωνικού Δικτύου και τις επιθυμητές εξόδους
- **void backwordPass()** : Συνάρτηση που υλοποιεί τον Αλγόριθμο Οπισθοδρομικής Διάδοσης Σφάλματος (Error Back Propagation)

Η βασική ιδέα του αλγορίθμου είναι η εξής. Υπολογίζονται πρώτα οι τιμές του δέλτα για το τελευταίο(δεύτερο) επίπεδο σύμφωνα με τη διαφορά της εξόδου για κάθε νευρώνα i με την επιθυμητή έξοδο για το νευρώνα αυτόν, δηλαδή:

$$Delta_2(i) = Output(i) - Desired\_Output(i)$$

Στη συνέχεια ανανεώνονται τα βάρη στο δεύτερο επίπεδο προσθέτοντας για κάθε βάρος την ποσότητα ΔW_{ij}. Η ποσότητα αυτή υπολογίζεται για κάθε βάρος που συνδέει τον νευρώνα i του πρώτου επιπέδου με το νευρώνα j του δεύτερου σύμφωνα με τον παρακάτω τύπο:

$$W^{ij}{}_2 = -a \cdot Delta_2(j) \cdot Output(i)$$

Στη συνέχεια υπολογίζονται οι τιμές του δέλτα στο πρώτο επίπεδο για κάθε νευρώνα i του πρώτου επιπέδου σύμφωνα με την παράγωγο της εξόδου του νευρώνα i πολλαπλασιάζοντας και με ένα άθροισμα για τις τιμές των βαρών και των δέλτα του δεύτερου επιπέδου τόσο για τους νευρώνες, όσο και για το bias, όπως φαίνεται στον παρακάτω τύπο:

$$Delta_1(i) = O_2{}'(i) \cdot \sum_j w_2^{ij} \cdot Delta_2(j)$$

Επειδή σαν συνάρτηση ενεργοποίησης έχουμε επιλέξει την σιγμοειδή συνάρτηση

$\sigma(x) = \frac{1}{1+e^{-x}}$, έχουμε ότι $O_2{}'(i) = O_2(i) \cdot (1 - O_2(i))$

Στη συνέχεια ανανεώνονται και τα βάρη στο πρώτο επίπεδο με τον ίδιο τρόπο όπως και στο δεύτερο προσθέτοντας για κάθε βάρος την ποσότητα ΔW_{ij}. Η ποσότητα αυτή υπολογίζεται για κάθε βάρος που συνδέει την είσοδο i με το νευρώνα j του πρώτου επιπέοδυ σύμφωνα με τον παρακάτω τύπο:

$$W^{ij}{}_1 = -a \cdot Delta_1(i) \cdot Input(j)$$

Σαν παράμετρος μάθησης επιλέχθηκε η τιμή: $\alpha = 0.1$

Για να δούμε τις δυνατότητες του αλγόριθμου, τοποθετούμε στην είσοδο και την έξοδο τυχαίες τιμές στο διάστημα [-1,1] με τη βοήθεια της συνάρτησης **initializeInputOutput()**.Επαναλάβετε την διαδικασία εκπαίδευσης για 10 επαναλήψεις με το ίδιο ζευγάρι εισόδου-επιθυμητής εξόδου. Παρακάτω βλέπουμε τα αποτελέσματα

```
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex2$ gcc ex2.c -o ex2 -lm
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex2$ ./ex2
Neural Network Training

Iteration       MSE
1               0.176916:
2               0.028397:
3               0.008110:
4               0.001063:
5               0.000356:
6               0.000149:
7               0.000054:
8               0.000029:
9               0.000012:
10              0.000008:

Network Output  Desirted Output
0.364747        0.364784
0.515437        0.513401
0.951307        0.952230
0.916724        0.916195
0.635944        0.635712
0.717299        0.717297
0.140919        0.141603
0.608057        0.606969
0.022265        0.016301
0.242823        0.242887
Final MSE 0.000004:
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex2$
```

Παρατηρούμε ότι η διαφορά ανάμεσα στην έξοδο του Ν.Δ. και τις επιθυμητές τιμές ελαττώνεται συνεχώς και να τείνει στο μηδέν. Αυτό φαίνεται από την τιμή του Μέσου Τετραγωνικού Σφάλματος το οποίο μειώνεται σταθερά και τείνει στο 0. Αυτή η συμπεριφορά παρατηρήθηκε για οποιαδήποτε τυχαία διανύσματα εισόδου και επιθυμητής εξόδου.

Εργασία 3

Για να βρούμε τα τμήματα του κώδικα που αναλώνεται ο περισσότερος χρόνος εκτέλεσης κάνουμε χρήση του GNU profiler gprof.

Αρχικά κάνουμε compile με παράμετρο -pg ώστε να δώσουμε οδηγία στον compiler να δημιουργήσει επιπλέον κώδικα ώστε να γίνει η κατάλληλη καταγραφή πληροφοριών για την ανάλυση που απαιτεί ο gprof. Στη συνέχεια εκτελούμε τον κώδικα και με τη χρήση του gprof βλέπουμε την ανάλυση παρακάτω:

```
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex3$ gcc ex3_serial.c -o ex3 -lm -pg
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex3$ ./ex3
Training for 1000 epochs took total 0.715027 seconds to execute
Average time per epoch 0.000715 seconds
Final MSE 0.000000:
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex3$ gprof ex3_serial gmon.out > analysis.txt
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex3$ cat analysis.txt
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 97.48      0.68     0.68     1000   682.36   682.36  backwordPass
  2.87      0.70     0.02                             sigmoid
  0.00      0.70     0.00   310110     0.00     0.00  mse
  0.00      0.70     0.00     1001     0.00     0.00  activateNN
  0.00      0.70     0.00        2     0.00     0.00  initializeNN
  0.00      0.70     0.00        1     0.00     0.00  printOutputs
```

Παρατηρούμε ότι η συνάρτηση **backwordPass()** που υλοποιεί τον Αλγόριθμο Οπισθοδρομικής διάδοσης σφάλματος καταναλώνει το 97.48% του χρόνου εκτέλεσης. Συνεπώς είναι η συνάρτηση που θα επιδιώξουμε να παραλληλοποιήσουμε για να βελτιώσουμε τη συνολική απόδοση του κώδικά μας. Επίσης παρατηρούμε ότι ο μέσος χρόνος για την εκπαίδευση του Ν.Δ. για ένα epoch είναι **0.715msec**

Όσον αφορά τη βελτίωση τη βελτιστοποίηση του σειριακού κώδικα μπορούμε να κάνουμε χρήση Vector Processing για να εκμεταλλευτούμε τους Wide Registers που μας προσφέρει η αρχιτεκτονική. Αυτό μπορούμε να το πετύχουμε είτε με δηλώσεις **#pragma omp simd**, είτε χρησιμοποιώντας τις επιλογές που βελτιστοποίησης που μας παρέχει ο compiler. Θα κάνουμε χρήση της επιλογής βελτιστοποίησης **-O3** και εκτός από τις διάφορες βελτιστοποιήσεις που εφαρμόζει ο gcc, όπως το loop unrolling, και το inlining παρακάτω βλέπουμε σε ποιες γραμμές κώδικα γίνεται Vector Processing

```
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex3$ gcc ex3_serial.c -o ex3 -lm -O3 -fopt-info-vec
ex3_serial.c:120:9: optimized: loop vectorized using 16 byte vectors
ex3_serial.c:120:9: optimized: loop vectorized using 16 byte vectors
ex3_serial.c:153:13: optimized: loop vectorized using 16 byte vectors
ex3_serial.c:141:9: optimized: loop vectorized using 16 byte vectors
ex3_serial.c:135:5: optimized: loop vectorized using 16 byte vectors
ex3_serial.c:174:2: optimized: loop vectorized using 16 byte vectors
ex3_serial.c:174:2: optimized: loop vectorized using 16 byte vectors
```

Με τις βελτιστοποιήσεις που εφαρμόσαμε ρίξαμε το χρόνο εκπαίδευσης για ένα epoch, από **0.715msec** στα **0.224msec** όπως φαίνεται παρακάτω:

```
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex3$ ./ex3_serial
Training for 1000 epochs took total 0.224396 seconds to execute
Average time per epoch 0.000224 seconds
Final MSE 0.000000:
```

Στη συνέχεια γίνεται χρήση οδηγιών της OpenMP για παραλληλοποίηση του κώδικα για τα μεγέθη εισόδων, εξόδων και νευρώνων που ζητούνται.

Για να λάβουμε ακριβή μέτρηση μέσου χρόνου εκπαίδευσης του Ν.Δ. για ένα epoch στην παράλληλη υλοποίηση αντί για χρήση τη συνάρτησης **clock()** της **time.h** που έγινε στον σειριακό κώδικα θα γίνει χρήση της συνάρτησης **omp_get_wtime()** της **omp.h**

Επειδή τα μεγέθη που ζητούνται για αυτό το ερώτημα είναι μικρά, βελτίωση της απόδοσης του κώδικα έγινε μόνος με την παραλληλοποίηση του τριπλού for-loop για τον υπολογισμό των παραμέτρων δέλτα του πρώτου επιπέδου, όπως φαίνεται πρακάτω:

```
#pragma omp parallel for
for(int i=0; i<LEVEL_1_NEURONS; i++){
    for(int j=0; j<LEVEL_2_NEURONS; j++){
        for(int k=0; k<LEVEL_1_NEURONS+1; k++){
            D1[i]+= D2[j] * WL2[j][k];
        }
    }
    D1[i]*=sigmoid(DL1[i])*(1-sigmoid(DL1[i]));
}
```

Χρειάζεται απλά μία δήλωση **#pragma omp parallel for** καθώς ο τρόπος που είναι γραμμένα τα loop καθιστά τις πράξεις μεταξύ των thread ανεξάρτητες και χωρίς race condition χωρίς καμία περεταίρω οδηγία συγχρονισμού.

Έγινε προσπάθεια να παραλληλοποιηθούν και άλλα τμήματα του κώδικα, αλλά ο συνολικός χρόνος αυξανόταν καθώς το overhead για create και join των thread ήταν αρκετά μεγάλο και η σειριακή εκτέλεση τους ήταν γρηγορότερη. Αυτό οφείλεται στο γεγονός πως το διάνυσμα εισόδου που ζητείται είναι αρκετά μικρό, όπως και ο αριθμός νευρώνων στο δεύτερο επίπεδο.

Με την παραλληλοποίηση με τη χρήση 4 thread καταφέραμε να ρίξουμε το συνολικό χρόνο εκπαίδευσης του Ν.Δ. για ένα epoch από τα **0.224msec** στα **0.097msec**. Πλέον η εκπαίδευση γίνεται 2.3 φορές πιο γρήγορα.

Να σημειωθεί σε αυτό το σημείο πως μετά από πειράματα για μεγαλύτερο αριθμό εισόδων, εξόδων και νευρώνων σε κάθε επίπεδο, καθώς και παραλληλοποιώντας περισσότερα τμήματα του αλγορίθμου η επιτάχυνση ήταν καλύτερη. Βέβαια αυτές οι δοκιμές ξεφεύγουν από τα ζητούμενα της άσκησης.

Εργασία 4

Για τα ζητούμενα αυτής της άσκησης θα χρειαστεί να υλοποιήσουμε και κάποιες ακόμα συναρτήσεις, αναλυτικά:

- **void loadTrainingData()**: Συνάρτηση που φορτώνει το αρχείο του training set και τοποθετεί τα δεδομένα των εικόνων και των κατηγοριών στους κατάλληλους πίνακες. Γίνεται χρήση των συναρτήσεων **fopen(),fgets()** και **strtok()**
- **void loadTestData**: Συνάρτηση παρόμοια με την **loadTrainingData()**, αλλά αφορά το test set
- **int classDecision(double *output)**: Συνάρτηση που με είσοδο το διάνυσμα εξόδου του Ν.Δ, επιστρέφει ποια είναι η κατηγορία που επιλέγεται ανάμεσα στις 10, ανάλογα με το ποιος νευρώνας έχει τη μεγαλύτερη τιμή.
- **int desiredDecision(int input, int train_flag)**: Συνάρτηση που παίρνει σαν είσοδο τον αριθμό της εικόνας που μας ενδιαφέρει και ένα flag, για το αν πρόκειται για εικόνα του train ή του test set και επιστρέφει την επιθυμητή κατηγορία του ανατρέχοντας στον κατάλληλο πίνακα με τα labels.
- **int decisionTruth(int output, int desired)**: Συνάρτηση που ελέγχει αν η κατηγορία που επέλεξε το Ν.Δ είναι ίδια με την επιθυμητή κατηγορία σύμφωνα με το dataset

Επίσης για το debugging χρησιμοποιήθηκαν οι παρακάτω συναρτήσεις

- **printData()**: Βοηθητική συνάρτηση για να βεβαιωθούμε ότι τα δεδομένα που φορτώνονται από τα csv files είναι σωστά.
- **printOutputs()**: Βοηθητική συνάρτηση που τυπώνει τις εξόδους του Ν.Δ

Όσον αφορά τις παραμέτρους του Ν.Δ μετά από πειράματα έγιναν οι εξής επιλογές.

- Στα βάρη δόθηκαν τιμές double ανάμεσα στο [-1,1]. Ο λόγος που δόθηκαν μικρές τιμές στα βάρη είναι για να επιτύχουμε σταθερότητα και μικρότερη πιθανότητα το Ν.Δ να κάνει overfit. Ουσιαστικά το Ν.Δ έτσι έχει καλύτερη δυνατότητα γενίκευσης και έχει καλή απόδοση και σε καινούρια δεδομένα.
- Όσον αφορά την παράμετρο μάθησης μετά από πειράματα επιλέχθηκε η τιμή $\alpha = 0.0001$ Με αυτή την τιμή επιτυγχάνουμε αργή και σταθερή εκπαίδευση στο Ν.Δ. Με μεγαλύτερες παραμέτρους μάθησης παρατηρήθηκε αυξομείωση του accuracy από τους πρώτους κιόλας γύρους εκπαίδευσης και το accuracy παρέμενε αρκετά χαμηλά. Αντίθετα με μικρές τιμές για το α το accuracy ήταν μεγαλύτερο και αυτή η αυξομείωση παρατηρήθηκε περίπου μετά από 15 κύκλους εκπαίδευσης. Επιπλέον με μικρή τιμή για την παράμετρο μάθησης τα βάρη εξακολουθούν να διατηρούν τις μικρές τιμές τους και το Ν.Δ διατηρεί τις ικανότητες γενίκευσης που επιθυμούμε.
- Ο αριθμός κρυφών επιπέδων και νευρώνων στο κρυφό επίπεδο επιλέχθηκαν σύμφωνα με αυτούς που δίνονται από την εκφώνηση. Να σημειωθεί πως στη γενίκευση συμβάλει και το γεγονός ότι έχουμε πολύ λίγους νευρώνες(100) στο κρυφό επίπεδο και ένα μόνο κρυφό επίπεδο.

Η διαδικασία της εκπαίδευσης και αξιολόγησης γίνεται στη main. Να σημειωθεί πως ο κώδικας στη main δεν είναι αρκετά «καθαρός» και πολλά πράγματα θα μπορούσαν να γίνουν πιο όμορφα με κάποιες επιπλέον συναρτήσεις. Αυτό δεν έγινε δυστυχώς λόγω πίεσης χρόνου και εξεταστικής.

Αρχικά φορτώνονται τα δεδομένα του training set γίνεται η εκπαίδευση τυπώνοντας το accuracy για κάθε γύρο. Στη συνέχεια τυπώνονται τα συνολικά αποτελέσματα για το accuracy καθώς και πληροφορίες για το συνολικό χρόνο εκπαίδευσης και το μέσο χρόνο εκπαίδευσης για κάθε γύρο. Στη συνέχεια φορτώνεται το test σετ και με τα βάρη που έχουν διαμορφωθεί από την εκπαίδευση ενεργοποιείται το Ν.Δ και υπολογίζεται το accuracy για το test set. Τα αποτελέσματα φαίνονται στην εικόνα «Τελικά Αποτελέσματα» που παρατίθεται στο τέλος της εργασίας.

Η επιλογή του αριθμού των γύρων εκπαίδευσης έγινε μετά από πειράματα. Για τις παραμέτρους που επιλέξαμε παραπάνω παρατηρήθηκες γρήγορη αύξηση του accuracy από τους 4 πρώτους γύρους εκπαίδευσης φτάνοντας στο 50%. Στους 11 κύκλους εκπαίδευσης το Ν.Δ φτάνει σε accuracy 59%. Για τους επόμενους γύρους μέχρι και τον γύρο 33 υπάρχει αυξομείωση στο accuracy σε τιμές ανάμεσα στο 60-65%. Πιθανότατα να μπορούσαμε να λύσουμε το πρόβλημα αυτό αλλάζοντας την τιμή της παραμέτρου μάθησης μετά από κάποιους γύρους εκπαίδευσης, κάτι που δεν υλοποιήθηκε στα πλαίσια αυτής της υλοποίησης.

Παρατηρώντας την παρακάτω εικόνα με τα τελικά αποτελέσματα παρατηρούμε ότι ο **συνολικός χρόνος εκπαίδευσης** για τις 60.000 εικόνες για 33 γύρους ήταν **302sec** δηλαδή **5 λεπτά**. Ο **μέσος χρόνος για έναν γύρο εκπαίδευσης** είναι **9.1sec**.

Όσον αφορά την τελική ακρίβεια πρόβλεψης πετύχαμε:

- **Accuracy 65.8%** για το **training set**
- **Accuracy 65.2%** για το **test set**

Τα αποτελέσματα είναι αρκετά ικανοποιητικά και βλέπουμε ότι το accuracy για το test set είναι αρκετά κοντά σε αυτό του training set. Αυτό σημαίνει πως η επιλογή που κάναμε στις παραμέτρους για να πετύχουμε γενίκευση ήταν αρκετά καλή.

```
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex4$ gcc ex4.c -o ex4 -O3 -lm -fopenmp
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex4$ time ./ex4
Beginning Training
Epoch: 1 accuracy 14.66333%
Epoch: 2 accuracy 32.42000%
Epoch: 3 accuracy 41.35667%
Epoch: 4 accuracy 50.88833%
Epoch: 5 accuracy 52.07333%
Epoch: 6 accuracy 52.66833%
Epoch: 7 accuracy 49.71667%
Epoch: 8 accuracy 52.42333%
Epoch: 9 accuracy 58.63000%
Epoch: 10 accuracy 58.48000%
Epoch: 11 accuracy 59.86500%
Epoch: 12 accuracy 62.20500%
Epoch: 13 accuracy 62.54167%
Epoch: 14 accuracy 63.86833%
Epoch: 15 accuracy 62.24667%
Epoch: 16 accuracy 61.31167%
Epoch: 17 accuracy 62.72167%
Epoch: 18 accuracy 62.59500%
Epoch: 19 accuracy 64.53167%
Epoch: 20 accuracy 63.70667%
Epoch: 21 accuracy 61.12500%
Epoch: 22 accuracy 63.16500%
Epoch: 23 accuracy 62.85000%
Epoch: 24 accuracy 61.26000%
Epoch: 25 accuracy 61.51333%
Epoch: 26 accuracy 63.46333%
Epoch: 27 accuracy 65.74167%
Epoch: 28 accuracy 64.79667%
Epoch: 29 accuracy 63.93833%
Epoch: 30 accuracy 61.39833%
Epoch: 31 accuracy 64.12000%
Epoch: 32 accuracy 65.69833%
Epoch: 33 accuracy 64.28833%

Total Training for 60000 images
Epochs: 33
Training time: 302.519229 sec
Training time per epoch: 9.167249 seconds

Final Training Accuracy: 65.79833%

Testing for 10000 images
Final Test Accuracy: 65.26000%

real    5m11,240s
user    19m58,605s
sys     0m2,919s
osboxes@osboxes:~/Desktop/parProg/Labs/lab4/ex4$
```

Τελικά Αποτελέσματα

ΤΕΛΟΣ