



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΠΛΗΡΟΦΟΡΙΑΣ

Αναζήτηση και Εξόρυξη Πληροφορίας σε
Μεγάλες Βάσεις Αδόμητων Δεδομένων με
Μεθόδους Παράλληλης Επεξεργασίας

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΟΛΥΜΠΙΑΣ ΤΣΑΜΟΥ

Επιβλέπων: Ευάγγελος Δερματάς

Πάτρα, Δεκέμβριος 2022

Πανεπιστήμιο Πατρών, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών.

Ολυμπία Τσάμου

© 2022 – Με την επιφύλαξη παντός δικαιώματος

Το σύνολο της εργασίας αποτελεί πρωτότυπο έργο, παραχθέν από την Ολυμπία Τσάμου, και δεν παραβιάζει δικαιώματα τρίτων καθ' οιονδήποτε τρόπο. Αν η εργασία περιέχει υλικό, το οποίο δεν έχει παραχθεί από την ίδια, αυτό είναι ευδιάκριτο και αναφέρεται ρητώς εντός του κειμένου της εργασίας ως προϊόν εργασίας τρίτου, σημειώνοντας με παρομοίως σαφή τρόπο τα στοιχεία ταυτοποίησής του, ενώ παράλληλα βεβαιώνει πως στην περίπτωση χρήσης αυτούσιων γραφικών αναπαραστάσεων, εικόνων, γραφημάτων κ.λπ., έχει λάβει τη χωρίς περιορισμούς άδεια του κατόχου των πνευματικών δικαιωμάτων για την συμπερίληψη και επακόλουθη δημοσίευση του υλικού αυτού.

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με τίτλο

**Αναζήτηση και Εξόρυξη Πληροφορίας σε
Μεγάλες Βάσεις Αδόμητων Δεδομένων με
Μεθόδους Παράλληλης Επεξεργασίας**

της φοιτήτριας του Τμήματος Ηλεκτρολόγων Μηχανικών και
Τεχνολογίας Υπολογιστών

ΟΛΥΜΠΙΑΣ ΤΣΑΜΟΥ ΤΟΥ ΙΩΑΝΝΗ

Αριθμός Μητρώου: 227644

Παρουσιάστηκε δημόσια στο Τμήμα Ηλεκτρολόγων Μηχανικών και
Τεχνολογίας Υπολογιστών στις

...../...../.....

και εξετάστηκε από την ακόλουθη εξεταστική επιτροπή:

Ευάγγελος Δερματάς, Αναπληρωτής Καθηγητής, Τμήμα Μηχανικών
Η/Υ και Πληροφορικής (επιβλέπων)

Θεόδωρος Αντωνακόπουλος, Καθηγητής, Τμήμα Ηλεκτρολόγων
Μηχανικών και Τεχνολογίας Υπολογιστών (μέλος επιτροπής)

Ο Επιβλέπων

Ο Διευθυντής του Τομέα

Ευάγγελος Δερματάς
Αναπληρωτής Καθηγητής

Κυριάκος Σγάρμπας
Καθηγητής

Περίληψη

Αναζήτηση και Εξόρυξη Πληροφορίας σε Μεγάλες Βάσεις Αδόμητων Δεδομένων με Μεθόδους Παράλληλης Επεξεργασίας

Φοιτήτρια: Ολυμπία Τσάμου

Επιβλέπων: Ευάγγελος Δερματάς

Εφαρμογή τεχνικών βελτιστοποίησης σε πρόβλημα ταξινόμησης εικόνων κάνοντας
χρήση ενός Συνελικτικού Νευρωνικού Δικτύου [cs61c] [1]

Abstract

Data Mining in Large Unstructured Data Bases with Parallel Processing Methods

Student: **Olympia Tsamou**

Supervisor: **Evangellos Dermatas**

Application of optimization techniques to an image classification problem using
a Convolutional Neural Network [cs61c] [1]

Ευχαριστίες

Θα ήθελα καταρχήν να ευχαριστήσω τον καθηγητή κ. για την επίβλεψη αυτής της διπλωματικής εργασίας για την καθοδήγησή του και την εξαιρετική συνεργασία που είχαμε. Τέλος θα ήθελα να ευχαριστήσω τους γονείς μου για την καθοδήγηση και την ηθική συμπαράσταση που μου προσέφεραν όλα αυτά τα χρόνια.

Περιεχόμενα

Περίληψη	i
Abstract	iii
Περιεχόμενα	vii
1 Εισαγωγή	1
1.1 Ορισμός του Προβλήματος	1
1.2 Στόχοι της Διπλωματικής Εργασίας	1
2 Νευρωνικά Δίκτυα	3
2.1 Νευρώνες	3
2.2 Συνελικτικά νευρωνικά δίκτυα	7
2.2.1 Layers used to build ConvNets	8
3 Παράλληλος Προγραμματισμός	11
3.1 Τι είναι παράλληλος προγραμματισμός	11
4 Υλοποίηση CNN	13
5 Notes	15
5.1 Χαρακτηριστικά Υπολογιστή:	15
5.2 Κώδικας	16
5.2.1 Εκτέλεση κώδικα με GCC	17
5.2.2 Εκτέλεση Κώδικα με NVC	17
5.3 Κατανόηση του Κώδικα	21
5.3.1 Δομή φακέλου	21
5.3.2 Επισκόπηση Κώδικα	21
5.4 Μεταφορά Παραμέτρων Νευρωνικού Δικτύου στην GPU	23
6 Επίλογος	29
6.1 Συμπεράσματα	29
6.2 Μελλοντικές Επεκτάσεις	29
Α' Παράρτημα	31
Βιβλιογραφία	33

Κεφάλαιο 1

Εισαγωγή

1.1 Ορισμός του Προβλήματος

1.2 Στόχοι της Διπλωματικής Εργασίας

Γίνεται έρευνα σχετικά με τις διαφορετικές αρχιτεκτονικές νευρωνικών δικτύων και τις μεθόδους εκπαίδευσης, ωστόσο, μια άλλη κρίσιμη πτυχή των νευρωνικών δικτύων είναι, δεδομένου ενός εκπαιδευμένου δικτύου, η γρήγορη και ακριβής ταξινόμηση εικόνων. Σε αυτή τη διπλωματική, διαθέτουμε ένα εκπαιδευμένο νευρωνικό δίκτυο που ταξινομεί εικόνες 32x32 RGB σε 10 κατηγορίες. Οι εικόνες ανήκουν στο dataset CIFAR-10[2]. Μας δίνετε ο αλγόριθμος εμπρόσθιας διάδοσης του νευρωνικού δικτύου και τα τελικά βάρη του δικτύου. Σκοπός είναι η βελτίωση της ταχύτητας της εμπρόσθιας διάδοσης ώστε να γίνει ταξινόμηση με πιο γρήγορο ρυθμό.

Αναγνώριση Εικόνων

Πώς ένας υπολογιστής αναγνωρίζει εικόνες

Η ταξινόμηση εικόνων περιγράφει ένα πρόβλημα στο οποίο σε ένα υπολογιστή δίνεται μία εικόνα και πρέπει να καταλάβει τι απεικονίζει (από ένα σύνολο πιθανών κατηγοριών). Σήμερα, τα Συνελικτικά Νευρωνικά Δίκτυα (CNNs) αποτελούν μια πολύ καλή προσέγγιση αυτού το προβλήματος. Γενικά, τα νευρωνικά δίκτυα υποθέτουν πως υπάρχει κάποια συνάρτηση από την είσοδο (π.χ. εικόνες) σε μία έξοδο (π.χ. ένα σύνολο κατηγοριών εικόνων). Ενώ οι κλασικοί αλγόριθμοι προσπαθούν να κωδικοποιήσουν κάποια πληροφορία του πραγματικού κόσμου στη συνάρτηση τους, τα CNN μαθαίνουν την συνάρτηση δυναμικά από ένα σύνολο ταξινομημένων εικόνων (labelled images)—αυτή η διαδικασία ονομάζεται εκπαίδευση. Μόλις καταλήξει σε μια σταθερή συνάρτηση (δηλαδή σε μια προσέγγιση αυτής), μπορεί να εφαρμόσει τη συνάρτηση σε εικόνες που δεν έχει ξαναδεί.

Τι μπορεί να κάνει ένα νευρωνικό δίκτυο

Ένα νευρωνικό δίκτυο αποτελείται από πολλαπλά επίπεδα. Κάθε επίπεδο λαμβάνει έναν πολυδιάστατο πίνακα αριθμών ως είσοδο και παράγει έναν άλλο πολυδιάστατο πίνακα αριθμών ως έξοδο (ο οποίος στη συνέχεια γίνεται η είσοδος του επόμενου επιπέδου). Κατά την ταξινόμηση εικόνων, η είσοδος του πρώτου επιπέδου είναι η εικόνα εισόδου

(π.χ. $32 \times 32 \times 3$ αριθμοί για εικόνες 32×32 pixel με 3 κανάλια χρώματος), ενώ η έξοδος του τελευταίου επιπέδου αποτελείται ένα σύνολο πιθανοτήτων των διαφόρων κατηγοριών (π.χ., $1 \times 1 \times 10$ αριθμοί αν υπάρχουν 10 κατηγορίες).

Κάθε επίπεδο έχει ένα σύνολο από βάρη που σχετίζονται με αυτό — αυτά τα βάρη είναι που “μαθαίνει” το νευρωνικό όταν του δοθούν δεδομένα εκπαίδευσης. Ανάλογα με το επίπεδο, τα βάρη έχουν διαφορετικές ερμηνείες, αλλά δεν είναι αντικείμενο μελέτης της συγκεκριμένης διπλωματικής, φτάνει να γνωρίζουμε ότι κάθε επίπεδο λαμβάνει μία είσοδο, εκτελεί κάποια διεργασία σε αυτή, που εξαρτάται από τα βάρη και παράγει μια έξοδο. Αυτό το βήμα ονομάζεται εμπρόσθια διάδοση: παίρνουμε μία είσοδο και την προωθούμε στο δίκτυο, παράγοντας το επιθυμητό αποτέλεσμα ως έξοδο. Η εμπρόσθια διάδοση είναι το μόνο που χρειάζεται για την ταξινόμηση εικόνων σε ένα ήδη εκπαιδευμένο CNN.

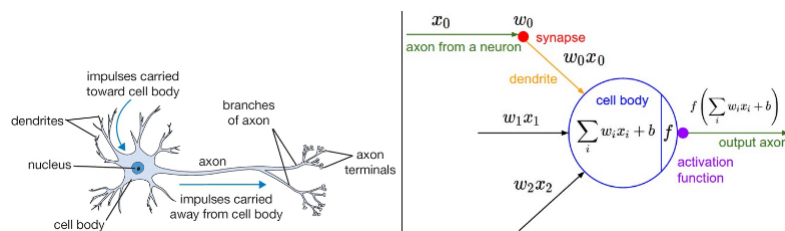
Στην πράξη, ένα νευρωνικό δίκτυο αποτελεί μια πολύ απλή μηχανή αναγνώρισης προτύπων (με εξαιρετικά περιορισμένη χωρητικότητα), αλλά μπορεί να είναι αρκετά παράξενο αυτό που καταλήγει να αναγνωρίσει. Για παράδειγμα, κάποιος μπορεί να εκπαιδεύσει ένα νευρωνικό δίκτυο να αναγνωρίζει τη διαφορά μεταξύ “σχύλων” και “λύκων”, και να δουλέψει καλά κοιτώντας το χιόνι και το δάσος στο φόντο των φωτογραφιών με τους λύκους.

Κεφάλαιο 2

Νευρωνικά Δίκτυα

Μοντελοποίηση ενός νευρώνα

Οι νευρώνες είναι εμπνευσμένοι από τους βιολογικούς νευρώνες του ανθρώπινου νευρικού συστήματος. Παρακάτω φαίνεται μια απεικόνιση ενός βιολογικού νευρώνα και η μαθηματική μοντελοποίησή του.[3]



Σχήμα 2.1: Μοντέλο Νευρώνα

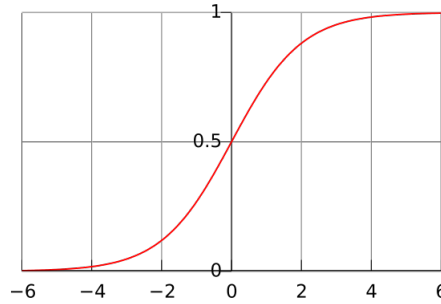
Κάθε νευρώνας λαμβάνει σήματα από τους δενδρίτες και παράγει ένα σήμα εξόδου στον άξονα. Στη συνέχεια ο άξονας διακλαδίζεται μέσω συνάψεων σε δενδρίτες άλλων νευρώνων. Στο υπολογιστικό μοντέλο τα σήματα (x_0) στον άξονα, αλληλοεπιδρούν πολλαπλασιαστικά μέσω των συνάψεων με τους δενδρίτες ($w_0 x_0$). Οι συνάψεις θεωρούμε πως είναι τα εκπαιδευσιμα στοιχεία (βάρη w) τα οποία ελέγχουν την επιρροή του ενός νευρώνα σε κάποιον άλλο. Στο κυρίως μέρος του νευρώνα, αθροίζονται τα σήματα από τους δενδρίτες. Εάν αυτό το άθροισμα ξεπερνά ένα συγκεκριμένο κατώφλι, ο νευρώνας ενεργοποιείται και στέλνει σήμα στον άξονα εξόδου. Στο μαθηματικό μοντέλο ο ακριβής χρόνος που παράγονται τα σήματα δεν έχει σημασία, η συχνότητα ενεργοποίησης του νευρώνα μας ενδιαφέρει. Ο τρόπος με τον οποίο το αναπαριστούμε αυτό είναι με μια συνάρτηση ενεργοποίησης. Η πιο συνηθισμένη συνάρτηση είναι η σιγμοειδής, η οποία έχει σαν είσοδο μια πραγματική τιμή (την ισχύ του σήματος μετά το άθροισμα) και το περιορίζει στο διάστημα μεταξύ 0 και 1. Με άλλα λόγια, κάθε νευρώνας εκτελεί το εσωτερικό γινόμενο της εισόδου με τα βάρη προσθέτοντας και μία σταθερά (bias) και εφαρμόζει μία μη γραμμική συνάρτηση, σε αυτή την περίπτωση η σιγμοειδής $\sigma(x) = 1/(1 + e^{-x})$.

2.1 Νευρώνες

Ο νευρώνας είναι μια απλή συνάρτηση. Παίρνει μία είσοδο x , υπολογίζει μια συνάρτηση $f(x)$ και τις εξόδους της. Επιλέγουμε τη λογιστική συνάρτηση:

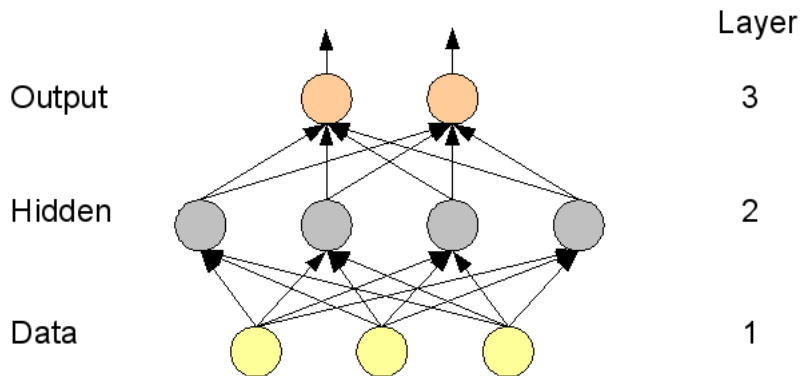
$$f(x) = \frac{1}{1 + e^{-x}}$$

Η γραφική παράστασή της φαίνεται στο Σχήμα 2.2. Είναι η πιο συχνά χρησιμοποιούμενη συνάρτηση. Έχει την ιδιότητα ότι η έξοδός της είναι ουσιαστικά γραμμική ως προς την είσοδο, εάν το μέγεθος της εισόδου είναι μικρό. Αυτό σημαίνει ότι τα νευρωνικά δίκτυα με μικρά βάρη υπολογίζουν μια γραμμική συνάρτηση και η σταδιακή αύξηση των βαρών επιτρέπει τον έλεγχο της μη γραμμικότητας. Ο βαθμός της μη γραμμικότητας ελέγχει την “χωρητικότητα” του νευρωνικού δικτύου. Για παράδειγμα, ένα δίκτυο μόνο με γραμμικούς νευρώνες μπορεί να υπολογίσει μόνο γραμμικές συναρτήσεις



Σχήμα 2.2: Η λογιστική συνάρτηση $f(x) = \frac{1}{1+e^{-x}}$

Νευρωνικό Δίκτυο Πρόσθιας Τροφοδότησης (feed forward)



Σχήμα 2.3: A feed-forward neural network with one hidden layer.

Τα feed-forward νευρωνικά δίκτυα συνδέουν μεταξύ τους επίπεδα νευρώνων. Κάθε νευρώνας στο επίπεδο l συνδέεται με κάθε νευρώνα στο επίπεδο $l + 1$, αλλά δεν υπάρχουν συνδέσεις εντός του επιπέδου. Αυτή η αρχιτεκτονική απεικονίζεται στο Σχήμα 2.3. Σε κάθε σύνδεση υπάρχει ένα βάρος με τιμές πραγματικούς αριθμούς. Ο νευρώνας k στο επίπεδο l δέχεται σαν είσοδο την τιμή:

$$x_k^l = b_k^l + \sum_{i=1}^{N_{l-1}} w_{ik}^{l-1} y_i^{l-1}$$

Όπου

- b_k^l είναι το bias στον νευρώνα k στο επίπεδο l ,

- N_{l-1} είναι ο αριθμός των νευρώνων στο επίπεδο $l - 1$,
- w_{ik}^{l-1} είναι το βάρος μεταξύ του νευρώνα i στο επίπεδο $l - 1$ και του νευρώνα k στο επίπεδο l , και
- y_i^{l-1} είναι η έξοδος του νευρώνα i στο επίπεδο $l - 1$

Στη συνέχεια ο νευρώνα υπολογίζει την έξοδο

$$y_k^l = f(x_k^l)$$

Όπου f είναι οποιαδήποτε διαφορίσιμη συνάρτηση του συνόλου των εισόδων του νευρώνα (εδώ η λογιστική συνάρτηση παραπάνω). Οι νευρώνες στο επίπεδο data έχουν σαν έξοδο απλά τα δεδομένα. Τέλος, η συνάρτηση

$$E(y_1^L, \dots, y_{N_L}^L)$$

της εξόδου του νευρωνικού δικτύου που θα θέλαμε να μεγιστοποιήσουμε (μπορούμε να το θεωρήσουμε σαν ένα επιπλέον επίπεδο στην έξοδο), όπου L είναι ο αριθμός των επιπέδων στο νευρωνικό δίκτυο. Η E πρέπει να είναι διαφορίσιμη ώστε το $\frac{\partial E}{\partial y_k^L}$ να είναι εύκολα υπολογίσιμο. Training the network consists of clamping the data neurons at the data and updating the parameters. Η εκπαίδευση του νευρωνικού δικτύου περιλαμβάνει clamping τους νευρώνες δεδομένων με τα δεδομένα και ενημερώνοντας τις παραμέτρους (τα βάρη και των biases). Οι παράγωγοι υπολογίζονται με τον παρακάτω :

$$\begin{aligned} \frac{\partial E}{\partial w_{ik}^{l-1}} &= \frac{\partial E}{\partial x_k^l} y_i^{l-1} \\ \frac{\partial E}{\partial b_k^l} &= \frac{\partial E}{\partial x_k^l} \end{aligned}$$

Όπου

$$\begin{aligned} \frac{\partial E}{\partial x_k^l} &= \frac{\partial E}{\partial y_k^l} \frac{\partial y_k^l}{\partial x_k^l} \\ \frac{\partial E}{\partial y_k^l} &= \begin{cases} \frac{\partial E}{\partial y_k^L} & \text{αν } l = L \\ \sum_{i=1}^{N_{l+1}} \frac{\partial E}{\partial x_i^{l+1}} w_{ki}^l & \text{αλλιώς} \end{cases} \end{aligned}$$

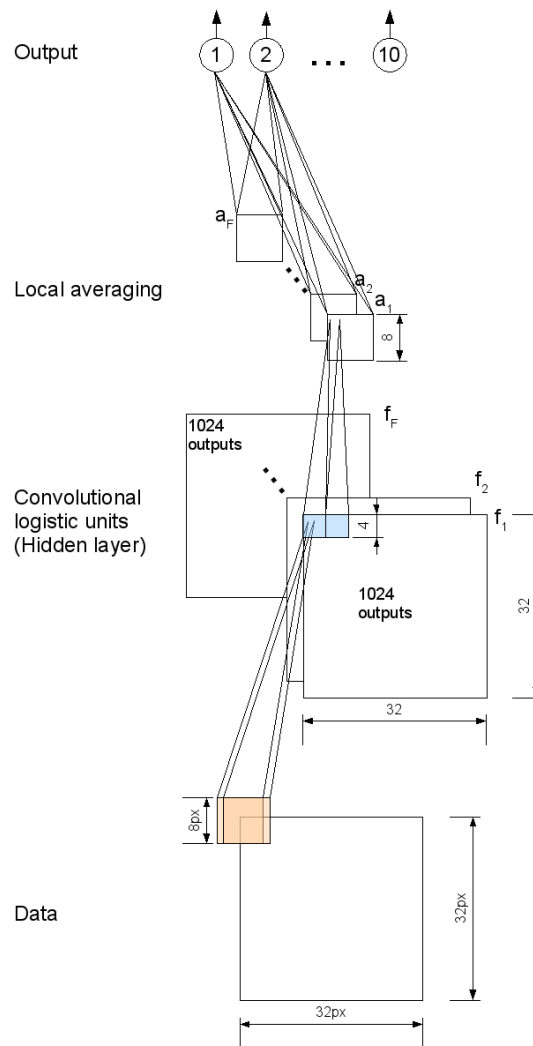
και το $\frac{\partial E}{\partial y_k^L}$ να είναι εύκολα υπολογίσιμο. Από αυτά, οι παράγωγοι ως προς τα βάρη και τα bias μπορούν να υπολογιστούν ξεκινώντας από το τελευταίο επίπεδο. Αυτό είναι γνωστό ως αλγόριθμος οπισθοδρομικής διάδοσης. Αν αυτό το δίκτυο χρησιμοποιηθεί για κατηγοριοποίηση, ο αριθμός των εξόδων ισούται με τον αριθμό των πιθανών κατηγοριών. Η έξοδος κάθε νευρώνα αντιστοιχεί στην εκτίμηση του δικτύου για κάποια κλάση.

Συνελικτικά Νευρωνικά Δίκτυα

Στα συνηθισμένα νευρωνικά δίκτυα πρόσθιας διάδοσης, κάθε νευρώνας στο επίπεδο l συνδέεται με κάθε νευρώνα του επιπέδου $l - 1$. Αυτό αρκεί αν δεν υπάρχουν τοπικές δομές στην ενεργοποίηση των νευρώνων του επιπέδου $l - 1$. Όμως στην περίπτωση των εικόνων υπάρχει κάποια δομή. Τα διπλανά pixel συσχετίζονται σε μεγαλύτερο βαθμό από αυτά που είναι μακριά. Για αυτό το λόγο κατευθύνουμε το νευρωνικό δίκτυο στην εξαγωγή τοπικών χαρακτηριστικών της εικόνας. Τα συνελικτικά δίκτυα το πετυχαίνουν αυτό συνδέοντας κάθε νευρώνα του κρυφού επιπέδου σε ένα μικρό σε ένα μικρό κομμάτι της εικόνας. Σε αυτή την περίπτωση επιλέγεται ένα κομμάτι 8×8 . Ένα συνελικτικό δίκτυο με έναν νευρώνα θα εφαρμόσει αυτό το φίλτρο 8×8 σε όλες τις 33×33 πιθανές θέσεις μίας 32×32 εικόνας (συμπληρώνοντας μηδενικά πλάτους 4 pixel σε κάθε άκρη). Οι έξοδοι των κρυμμένων επιπέδων αποτελούνται από εξόδους 33×33 αυτού του φίλτρου που εφαρμόζονται σε όλη την εικόνα. Σε αυτό το σημείο εμφανίζεται η συνέλιξη.

Το να έχουμε μόνο ένα νευρώνα είναι περιοριστικό, παρόλο που το φίλτρο εφαρμόζεται σε όλη την εικόνα και ο νευρώνας έχει πάνω από χίλιες εξόδους. Θα θέλαμε να έχουμε πολλούς νευρώνες που μαθαίνουν διαφορετικά φίλτρα, καθένα εφαρμόζεται σε μια εικόνα. Αυτό το σενάριο απεικονίζεται στην εικόνα 3. Στην εικόνα υπάρχουν Φ κρυφοί νευρώνες συνολικά, και κάθε 32×32 πλάκα αναπαριστά τις εξόδους 32×32 που παράγονται από ένα κρυφό νευρώνα όπου κάνει συνέλιξη το φίλτρο 8×8 με την εικόνα και εφαρμόζει την λογιστική συνάρτηση. Υπάρχουν 32×32 εξοδοί αντί για 33×33 επειδή στις GPU προτιμάμε αριθμούς όπως το 32 παρά το 33, και το νευρωνικό δίκτυο δεν θα χάσει αυτές τις λίγες επιπλέον εξόδους που λείπουν στις άκρες της εικόνας. Σε ένα τυπικό νευρωνικό δίκτυο για να πάρουμε 32×32 εξόδους από το κρυφό επίπεδο, θα πρέπει να έχουμε 32×32 νευρώνες σε αυτό, και αυτό επιτυγχάνεται εδώ με ένα μόνο νευρώνα. Έτσι μπορούμε να θεωρήσουμε τα συνελικτικά νευρωνικά δίκτυα σαν κανονικά νευρωνικά δίκτυα, αλλά με τον περιορισμό ότι ορισμένες ομάδες νευρώνων μοιράζονται τα βάρη (και επίσης οι αυτοί νευρώνες είναι μόνο τοπικά συνδεδεμένες με την εικόνα).

Οι εξοδοί ενός κρυφού νευρώνα όταν εφαρμόζεται συνέλιξη σε κοντινές περιοχές της εικόνας θα είναι παρόμοιες. Επομένως, η τιμές σε κοντινές περιοχές της ίδιας πλάκας 32×32 των εξόδων της κρυφής μονάδας (Σχήμα 2.4) στο κρυφό στρώμα είναι παρόμοιες. Έτσι, τα συνεπτυγμένα δίκτυα συχνά περιλαμβάνουν ένα στρώμα υποδειγματοληψίας ακριβώς πάνω από το συνεπτυγμένο στρώμα, για να υπολογίζεται τοπικά ο μέσος όρος των αποκρίσεων των κοντινών κρυφών μονάδων. Στο δίκτυό μου, οι μονάδες υπολογισμού του μέσου όρου υπολογίζουν τον μέσο όρο 4×4 μη επικαλυπτόμενων επιφανειών των εξόδων των κρυφών μονάδων. Το στρώμα υπολογισμού του μέσου όρου μειώνει το μέγεθος των πλακών των αποκρίσεων των κρυφών μονάδων από 32×32 σε 8×8 . Μπορεί επίσης να υποστηριχθεί ότι η απώλεια κάποιας ακρίβειας όσον αφορά την ακριβή θέση των χαρακτηριστικών στην εικόνα είναι στην πραγματικότητα πλεονεκτητική, επειδή επιτυγχάνεται μεγαλύτερος βαθμός αναλλοίωτης λειτουργίας. Στην όραση αποδεικνύεται συχνά ότι η ακριβής θέση ενός χαρακτηριστικού δεν είναι τόσο σημαντική όσο η κατά προσέγγιση θέση του σε σχέση με άλλα χαρακτηριστικά.



Σχήμα 2.4: The architecture of the convolutional neural network.

2.2 Συνελικτικά νευρωνικά δίκτυα

(CNNs / ConvNets)

πηγή: <https://cs231n.github.io/convolutional-networks/>

Τα Συνελικτικά νευρωνικά δίκτυα μοιάζουν πολύ με τα συνηθισμένα Νευρωνικά Δίκτυα: αποτελούνται από νευρώνες που έχουν εκπαιδευσιμα βάρη και σταθερές (biases). Κάθε νευρώνας λαμβάνει κάποιες εισόδους, εκτελεί ένα εσωτερικό γινόμενο και προαιρετικά ακολουθεί μια μη γραμμικότητα. Ολόκληρο το δίκτυο εκφράζει μια ενιαία διαφορίσιμη συνάρτηση βαθμολογίας(score): από τα pixel μιας εικόνας στο ένα άκρο έως τις βαθμολογίες της κλάσης στο άλλο. Ακολουθεί μια συνάρτηση απωλειών (π.χ.SVM/Softmax) στο τελευταίο (πλήρως συνδεδεμένο) επίπεδο.

Οι αρχιτεκτονικές ConvNet κάνουν τη ρητή παραδοχή ότι οι εισοδοί είναι εικόνες, γεγονός που μας επιτρέπει να κωδικοποιήσουμε ορισμένες ιδιότητες στην αρχιτεκτονική. Αυτές στη συνέχεια καθιστούν πιο αποτελεσματική την υλοποίηση της συνάρτησης forward και μειώνουν κατά πολύ τον αριθμό των παραμέτρων στο δίκτυο.

Επισκόπηση Αρχιτεκτονικής

Τα απλά Νευρωνικά Δίκτυα λαμβάνουν μια είσοδο (ένα απλό διάνυσμα) και τη μετασχηματίζουν μέσω μιας σειράς κρυφών επιπέδων. Κάθε κρυφό επίπεδο αποτελείται από ένα σύνολο νευρώνων, όπου κάθε νευρώνας συνδέεται πλήρως με όλους τους νευρώνες του προηγούμενου επιπέδου και όπου οι νευρώνες ενός επιπέδου λειτουργούν εντελώς ανεξάρτητα και δεν μοιράζονται καμία σύνδεση. Το τελευταίο πλήρως συνδεδεμένο επίπεδο ονομάζεται 'επίπεδο εξόδου' και αντιπροσωπεύει τις βαθμολογίες των κλάσεων.

Τα απλά Νευρωνικά Δίκτυα δεν προσαρμόζονται καλά (don't scale) σε πλήρεις εικόνες (full images). Στο CIFAR-10, οι εικόνες έχουν μέγεθος μόνο $32 \times 32 \times 3$ (32 πλάτος, 32 ύψος, 3 χρωματικά κανάλια), οπότε ένας μόνο πλήρως συνδεδεμένος νευρώνας στο πρώτο κρυφό επίπεδο ενός κανονικού Νευρωνικού Δικτύου θα είχε $32 \times 32 \times 3 = 3072$ βάρη. Αυτό το ποσό φαίνεται ακόμα διαχειρίσιμο, αλλά αυτή η πλήρως συνδεδεμένη δομή δεν κλιμακώνεται (does not scale) σε μεγαλύτερες εικόνες. Για παράδειγμα, μια εικόνα πιο μεγάλου μεγέθους, π.χ. $200 \times 200 \times 3$, θα οδηγούσε σε νευρώνες που έχουν $200 \times 200 \times 3 = 120.000$ βάρη. Επιπλέον, θα χρειαζόμασταν πολλούς τέτοιους νευρώνες, οπότε οι παράμετροι θα αυξάνονταν κατά πολύ. Η πλήρης συνδεσιμότητα σε αυτή την περίπτωση κοστίζει σε όγκο και ο μεγάλος αριθμός παραμέτρων μπορεί να οδηγήσει σε overfitting.

Τα Συνελικτικά Νευρωνικά Δίκτυα εκμεταλλεύονται το γεγονός ότι η είσοδος αποτελείται από εικόνες και χρησιμοποιούν την αρχιτεκτονική με τρόπο που έχει νόημα. Πιο συγκεκριμένα, σε αντίθεση με τα απλά Νευρωνικά Δίκτυα, τα επίπεδα του ConvNet έχουν νευρώνες οργανωμένους σε 3 διαστάσεις: πλάτος, ύψος, βάθος. (Με το βάθος αναφερόμαστε στην τρίτη διάσταση του όγκου ενεργοποίησης, όχι το βάθος ολόκληρου του Νευρωνικού Δικτύου, το οποίο αναφέρεται στον συνολικό αριθμό των επιπέδων στο δίκτυο.) Για παράδειγμα, οι εικόνες εισόδου στο CIFAR-10 αποτελεί τον όγκο εισόδου των ενεργοποιήσεων, και ο όγκος έχει διαστάσεις $32 \times 32 \times 3$ (πλάτος, ύψος, βάθος αντίστοιχα).

Επιπλέον: <https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html>

2.2.1 Layers used to build ConvNets

<https://cs231n.github.io/convolutional-networks/>

Ένα απλό ConvNet είναι μια ακολουθία επιπέδων και κάθε επίπεδο ενός ConvNet μετατρέπει έναν όγκο ενεργοποιήσεων (activations) σε έναν άλλο μέσω μιας διαφορίσιμης συνάρτησης. Χρησιμοποιούμε τρεις βασικούς τύπους επιπέδων για την κατασκευή αρχιτεκτονικών ConvNet : Convolutional Layer, Pooling Layer, και Fully-Connected Layer (όπως και στα κανονικά νευρωνικά δίκτυα). Για παράδειγμα. Ένα απλό ConvNet για ταξινόμηση του CIFAR-10, θα μπορούσε να έχει την παρακάτω αρχιτεκτονική [INPUT - CONV - RELU - POOL - FC], όπου:

- INPUT (Είσοδος) $[32 \times 32 \times 3]$ περιέχει τις τιμές των pixel της εικόνας, στην περίπτωση μας μία εικόνα πλάτους 32, ύψους 32 και τρία κανάλια χρωμάτων R,G,B.
- CONV layer (Συνελικτικό επίπεδο) υπολογίζει την έξοδο των νευρώνων που συνδέονται τοπικά σε περιοχές της εικόνας εισόδου, καθένας υπολογίζει το εσωτερικό γινόμενο από τα βάρη και μια μικρή περιοχή που συνδέεται στην είσοδο του

όγκου. Αυτό θα έχει σαν αποτέλεσμα έναν όγκο $[32 \times 32 \times 12]$ εάν αποφασίσουμε να χρησιμοποιήσουμε 12 φίλτρα.

- RELU layer (επίπεδο συνάρτησης ReLu) σε κάθε στοιχείο εφαρμόζεται μια συνάρτηση ενεργοποίησης, όπως η $\max(0, x)$ με κατώφλι στο 0. Αυτό το επίπεδο αφήνει το μέγεθος του όγκου το ίδιο ($[32 \times 32 \times 12]$).
- POOL layer αυτό το επίπεδο εκτελεί μια πράξη υποδειγματοληψίας κατά μήκος των χωρικών διαστάσεων (πλάτος, ύψος), με αποτέλεσμα έναν όγκο $[16 \times 16 \times 12]$.
- FC (i.e. fully-connected) layer (Πλήρως διασυνδεδεμένο επίπεδο) θα υπολογίσει τις βαθμολογίες των κλάσεων, με αποτέλεσμα έναν όγκο μεγέθους $[1 \times 1 \times 10]$, όπου κάθε ένας από τους 10 αριθμούς αντιστοιχεί στη βαθμολογία μιας κλάσης αναμεσα στις 10 κατηγορίες του CIFAR-10. Όπως και στα συνηθισμένα Νευρωνικά Δίκτυα και όπως υποδηλώνει το όνομα, κάθε νευρώνας σε αυτό το επίπεδο συνδέεται με όλους τους αριθμούς του προηγούμενου όγκου.

Με αυτό τον τρόπο, τα ConvNets μετατρέπουν μια εικόνα, layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

In summary:

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/ FC do, RELU/ POOL don't)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/ FC/ POOL do, RELU doesn't)

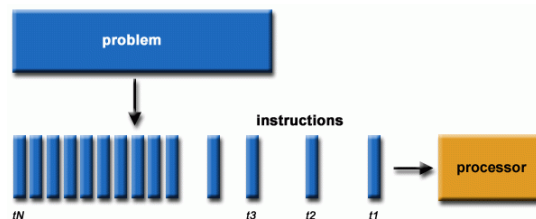
Κεφάλαιο 3

Παράλληλος Προγραμματισμός

3.1 Τι είναι παράλληλος προγραμματισμός

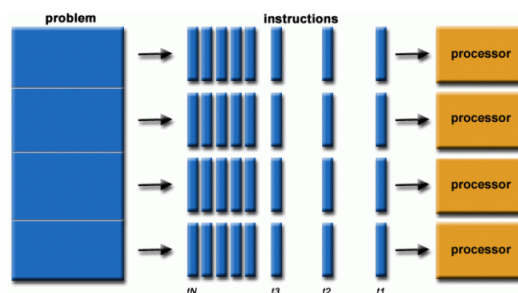
<https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial>

Παραδοσιακά, τα προγράμματα γράφονται για σειριακή εκτέλεση και υπολογισμούς. Ένα πρόβλημα αναλύεται σε μία σειρά διακριτών εντολών οι οποίες εκτελούνται διαδοχικά ή μία μετά την άλλη από έναν μόνο επεξεργαστή. Μόνο μία εντολή μπορεί να εκτελεστεί ανά πάσα στιγμή.



Σχήμα 3.1: Παράδειγμα σειριακής υπολογιστικής

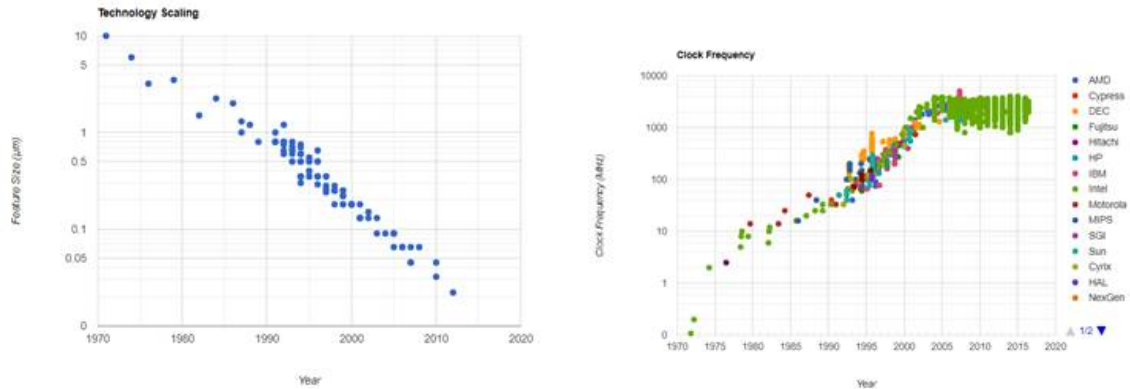
Ο παράλληλος προγραμματισμός αναφέρεται στην ταυτόχρονη χρήση πολλαπλών υπολογιστικών πόρων για την επίλυση ενός υπολογιστικού προβλήματος. Το πρόβλημα χωρίζεται σε διακριτά μέρη που μπορούν να επιλυθούν ταυτόχρονα. Κάθε μέρος αναλύεται περαιτέρω σε μια σειρά εντολών, οι οποίες εκτελούνται ταυτόχρονα σε διαφορετικούς επεξεργαστές. Επιπλέον, υπάρχει ένας κεντρικός μηχανισμός που συντονίζει και ελέγχει την εκτέλεση όλων αυτών.[4]



Σχήμα 3.2: Παράδειγμα παράλληλης υπολογιστικής

Γιατί Παράλληλος Προγραμματισμός·

Στο σημερινό κόσμο, στον τομέα της επιστήμης και της τεχνολογίας, ο όγκος των δεδομένων που χρειάζεται να επεξεργαστούμε αυξάνεται εκθετικά. Οι απαιτήσεις των εφαρμογών που δίνουν λύση σε πολλά σύγχρονα προβλήματα επιβάλλουν όλο και καλύτερες επιδόσεις στους επεξεργαστές. Ειδικότερα στον τομέα της τεχνητής νοημοσύνης και της μηχανικής μάθησης.



Σχήμα 3.3: Transistor Scaling and Clock Frequency[5]

Η αύξηση της απόδοσης των σειριακών επεξεργαστών έχει σταματήσει, καθώς οι σχεδιαστές επεξεργαστών έχουν φθάσει στα όρια της σμίχρυνσης, της συχνότητας ρολογιού, της ισχύος, ακόμη και της θερμότητας. Με την πάροδο του χρόνου, το μέγεθος των τρανζίστορ έχει μειωθεί. Λειτουργούν ταχύτερα, καταναλώνουν λιγότερη ενέργεια και μπορούμε να τοποθετήσουμε περισσότερα σε ένα ολοκληρωμένο κύκλωμα. Η αύξηση της συχνότητας του ρολογιού σε έναν επεξεργαστή, έχει σταματήσει λόγω της μεγάλης κατανάλωσης ενέργειας και την έκλυση θερμότητας που αυτή συνεπάγεται.

Καθώς η κατανάλωση ισχύος είναι από τους πιο σημαντικούς παράγοντες στον σχεδιασμό σύγχρονων επεξεργαστών, η αύξηση της υπολογιστικής ισχύος, σήμερα, επιτυγχάνεται κυρίως μέσω της προσθήκης όλο και περισσότερων υπολογιστικών πυρήνων στους επεξεργαστές. Αυτή η τάση της αύξησης του αριθμού των πυρήνων αντί της ταχύτητας του ρολογιού φαίνεται να επιτυγχάνει την ιδανικότερη απόδοση από άποψη κατανάλωσης αλλά και επεξεργαστικής ισχύος. Για να μπορέσουμε να αξιοποιήσουμε αυτού του είδους του hardware πρέπει να ξεφύγουμε από τον σειριακό τρόπο σκέψης στον προγραμματισμό και να αρχίσουμε να σκεφτόμαστε παράλληλα.

Κεφάλαιο 4

Υλοποίηση CNN

Κεφάλαιο 5

Notes

5.1 Χαρακτηριστικά Υπολογιστή:

Λειτουργικό σύστημα: Ubuntu 20.04.5 LTS

Επεξεργαστής: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz

Πληροφορίες :

```
$ nvcpuuid
vendor id      : GenuineIntel
model name     : Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
cpu family     : 6
model          : 61
name           : Broadwell Core gen 5 M-5xxx
stepping       : 4
processors     : 4
threads        : 2
clflush size   : 8
L2 cache size  : 256KB
L3 cache size  : 3072KB
flags          : acpi aes apic avx erms avx2 md_clear ibrs stibp l1d_flush
flags          : ssdb cflush cmov cplds cx8 cx16 de dtes f16c ferr fma fpu
flags          : fxsr ht lm mca mce mmx monitor movbe msr mtrr nx osxsave
flags          : pae pat pdcm pge popcnt pse pseg36 selfsnoop speedstep sep
flags          : sse sse2 sse3 ssse3 sse4.1 sse4.2 syscall tm tm2 tsc vme
flags          : xsave xtptr
default target : -tp haswell
```

Κάρτα Γραφικών: NVIDIA GeForce 920M

Πληροφορίες :

```
$ nvidia-smi
CUDA Driver Version:      11040
NVRM version:             NVIDIA UNIX x86_64 Kernel Module  470.141.03
                          Thu Jun 30 18:45:31 UTC 2022
Device Number:            0
Device Name:              NVIDIA GeForce 920M
Device Revision Number:   3.5
```

```

Global Memory Size:          2101739520
Number of Multiprocessors:    2
Concurrent Copy and Execution: Yes
Total Constant Memory:       65536
Total Shared Memory per Block: 49152
Registers per Block:         65536
Warp Size:                   32
Maximum Threads per Block:   1024
Maximum Block Dimensions:    1024, 1024, 64
Maximum Grid Dimensions:     2147483647 x 65535 x 65535
Maximum Memory Pitch:        2147483647B
Texture Alignment:           512B
Clock Rate:                  954 MHz
Execution Timeout:           Yes
Integrated Device:           No
Can Map Host Memory:         Yes
Compute Mode:                default
Concurrent Kernels:          Yes
ECC Enabled:                 No
Memory Clock Rate:           900 MHz
Memory Bus Width:            64 bits
L2 Cache Size:               524288 bytes
Max Threads Per SMP:         2048
Async Engines:               1
Unified Addressing:          Yes
Managed Memory:             Yes
Concurrent Managed Memory:    No
Default Target:              cc35

```

Εκδόσεις διαθέσιμων Compiler:

NVIDIA nvc 22.3-0 64-bit target on x86-64 Linux -tp haswell

```

GNU gcc (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
    gcc-10 (Ubuntu 10.3.0-1ubuntu1~20.04) 10.3.0

```

```

LLVM clang version 10.0.0-4ubuntu1
      Target: x86_64-pc-linux-gnu
      Thread model: posix

```

5.2 Κώδικας

Πήραμε τον κώδικα από το Github: <https://github.com/61c-teach/sp19-proj4-starter>

Site από αντίστοιχο μάθημα στο Berkeley: Project 4: Performance Programming

5.2.1 Εκτέλεση κώδικα με GCC

Απλή εκτέλεση του κώδικα χωρίς καμία αλλαγή.

Στο Makefile χρησιμοποιώ τα εξής Flags:

```
CFLAGS?=-Wall -Wno-unused-result -march=native -fopenmp -O3
```

Εκτέλεση:

```
$ make clean && make benchmark && ./benchmark benchmark
rm -f *.o
rm -f benchmark
rm -f benchmark_baseline
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -c benchmark.c
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -c network.c
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -c layers.c
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -c volume.c
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -o benchmark benchmark.o
    network.o layers.o volume.o -lm
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
Loading batches...
Loading input batch 0...
Running classification...
78.250000% accuracy
19103874 microseconds
```

Χρόνος Εκτέλεσης: 19,10 sec

5.2.2 Εκτέλεση Κώδικα με NVC

Εκτέλεση του κώδικα χωρίς καμία αλλαγή, με τον compiler της NVIDIA:

Makefile

```
CFLAGS?=-Wall -Minfo -mp -march=native -O3
CC=nvc
```

Στο Makefile αφαιρέθηκε το `-Wno-unused-result` καθώς δεν αναγνωρίζεται από τον nvc compiler. Προσθέσαμε όμως το flag `-Minfo` ώστε να παίρνουμε πληροφορίες σχετικά με βελτιστοποιήσεις που γίνονται.

```
$ make clean && make benchmark && ./benchmark benchmark
rm -f *.o
rm -f benchmark
rm -f benchmark_baseline
nvc -Wall -Minfo=opt -mp -march=native -O3 -c benchmark.c
run_classification:
    141, Memory zero idiom, loop replaced by call to __c_mzero8
nvc -Wall -Minfo=opt -mp -march=native -O3 -c network.c
```

```
net_classify:
    147, Loop unrolled 10 times (completely unrolled)
nvc -Wall -Minfo=opt -mp -march=native -O3 -c layers.c
conv_forward:
    109, FMA (fused multiply-add) instruction(s) generated
fc_forward:
    283, FMA (fused multiply-add) instruction(s) generated
softmax_forward:
    361, Loop unrolled 8 times
nvc -Wall -Minfo=opt -mp -march=native -O3 -c volume.c
nvc -Wall -Minfo=opt -mp -march=native -O3 -o benchmark benchmark.o network.o lay
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
Loading batches...
Loading input batch 0...
Segmentation fault (core dumped)
```

το σφάλμα Segmentation fault (core dumped) δεν συμβαίνει εάν αφαιρέσουμε το `optimazation -O3`

```
nvc -Wall -Minfo=opt -mp -march=native -c benchmark.c
nvc -Wall -Minfo=opt -mp -march=native -c network.c
nvc -Wall -Minfo=opt -mp -march=native -c layers.c
conv_forward:
    109, FMA (fused multiply-add) instruction(s) generated
fc_forward:
    283, FMA (fused multiply-add) instruction(s) generated
nvc -Wall -Minfo=opt -mp -march=native -c volume.c
nvc -Wall -Minfo=opt -mp -march=native -o benchmark benchmark.o network.o
layers.o volume.o -lm
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
Loading batches...
Loading input batch 0...
Running classification...
78.250000% accuracy
30988236 microseconds
```

Το optimization που δημιουργεί το πρόβλημα είναι το `-Mvect`. Ο κώδικας τρέχει κανονικά με `-O3 -Mnovect`.

Χρήση του Valgrind

Ελέγχουμε την εκτέλεση του κώδικα με το Valgrind για σφάλματα Memory Access και Leaks.

```
make CFLAGS="-Wall -Wno-unused-result -march=native -fopenmp -g -O0"
valgrind --leak-check=full --show-leak-kinds=all ./benchmark benchmark
```

!?! Με `-O0` και `-g` ο gcc και ο nvc δεν βγάζουν κάποιο πρόβλημα.

GCC Compiler

```
make CFLAGS="-Wall -Wno-unused-result -march=native -fopenmp -g -O3"  
valgrind --leak-check=full --show-leak-kinds=all ./benchmark benchmark
```

```
==11846== Memcheck, a memory error detector  
==11846== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.  
==11846== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info  
==11846== Command: ./benchmark benchmark  
==11846==  
RUNNING BENCHMARK ON 1200 PICTURES...  
Making network...  
Loading batches...  
Loading input batch 0...  
Running classification...  
78.250000% accuracy  
1335761832 microseconds  
==11846==  
==11846== HEAP SUMMARY:  
==11846==      in use at exit: 0 bytes in 0 blocks  
==11846==    total heap usage: 21,439 allocs, 21,439 frees, 247,283,464 bytes allocated  
==11846==  
==11846== All heap blocks were freed -- no leaks are possible  
==11846==  
==11846== For lists of detected and suppressed errors, rerun with: -s  
==11846== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

NVC Compiler

```
make CFLAGS?="-Wall -Minfo=opt -mp -march=native -g -O3"  
valgrind --leak-check=full --show-leak-kinds=all ./benchmark benchmark
```

```
==14505== Memcheck, a memory error detector  
==14505== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.  
==14505== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info  
==14505== Command: ./benchmark benchmark  
==14505==  
RUNNING BENCHMARK ON 1200 PICTURES...  
Making network...  
Loading batches...  
Loading input batch 0...  
==14505== Invalid write of size 8  
==14505==    at 0x404C56: volume_set (volume.c:23)  
==14505==    by 0x401ACC: run_classification (benchmark.c:91)  
==14505==    by 0x402CCA: main (benchmark.c:223)  
==14505== Address 0x62bda30 is 0 bytes after a block of size 24,576 allocated  
==14505==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/  
vgpreload_memcheck-amd64-linux.so)  
==14505==    by 0x404CD3: make_volume (volume.c:28)
```

```

==14505==      by 0x401A67: run_classification (benchmark.c:82)
==14505==      by 0x402CCA: main (benchmark.c:223)
==14505==
--14505-- VALGRIND INTERNAL ERROR: Valgrind received a signal 11 (SIGSEGV) -
      exiting
--14505-- si_code=128; Faulting address: 0x0; sp: 0x1002ca9e10
valgrind: the 'impossible' happened:
      Killed by fatal signal
host stacktrace:
==14505==      at 0x5804F830: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/
      memcheck-amd64-linux)
==14505==      by 0x58051B46: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/
      memcheck-amd64-linux)
==14505==      by 0x58005BF7: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/
      memcheck-amd64-linux)
==14505==      by 0x580A7204: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/
      memcheck-amd64-linux)
==14505==      by 0x580F5FD4: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/
      memcheck-amd64-linux)
sched status:
      running_tid=1
Thread 1: status = VgTs_Runnable (lwpid 14505)
==14505==      at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-gnu/valgrind/
      vgpreload_memcheck-amd64-linux.so)
==14505==      by 0x404CAD: make_volume (volume.c:27)
==14505==      by 0x401A67: run_classification (benchmark.c:82)
==14505==      by 0x402CCA: main (benchmark.c:223)
client stack range: [0x1FFEFFFC000 0x1FFEFFFFF] client SP: 0x1FFEFFD650
valgrind stack range: [0x1002BAA000 0x1002CA9FFF] top usage: 18696 of 1048576

```

Το πρόβλημα εντοπίζεται στο κομμάτι κώδικα `benchmark.c:91` στην συνάρτηση `volume_set()`

`benchmark.c:87-94`

```

int outp = 1;
for (int d = 0; d < 3; d++) {
    for (int y = 0; y < 32; y++) {
        for (int x = 0; x < 32; x++) {
            volume_set(batchdata[i], x, y, d, ((double)data[outp++])/255.0-0.5);
        }
    }
}

```

! Παρατήρηση 1:

Στο παραπάνω κομμάτι κώδικα, αν προσθέσουμε μια εντολή `if()` ανάμεσα στα `loop`, η οποία δεν επηρεάζει την αρχική λειτουργία του κώδικα, το πρόγραμμα τρέχει κανονικά.

```

int outp = 1;
for (int d = 0; d < 3; d++) {

```

```
for (int y = 0; y < 32; y++) {  
    for (int x = 0; x < 32; x++) {  
        volume_set(batchdata[i], x, y, d, ((double)data[outp++])/255.0-0.5);  
    }  
}  
if(outp==4) printf(".");  
}
```

! Παρατήρηση 2:

το πρόγραμμα λειτουργεί κανονικά με παλιότερη έκδοση του NVC compiler nvc 21.3 οι τιμές των environment variables για κάθε έκδοση βρίσκονται στα παρακάτω αρχεία

nvc-show-v21.3.txt

nvc-show-v22.3.txt

5.3 Κατανόηση του Κώδικα

5.3.1 Δομή φακέλου

Ο φάκελος περιλαμβάνει τα παρακάτω αρχεία:

```
Makefile  
layers.h  
snapshot/  
layers_baseline.c  
test/  
benchmark.c  
network.c  
volume.c  
network.h  
volume.h  
huge_test.sh  
network_baseline.c  
layers.c  
run_test.sh
```


Τα μόνα αρχεία που επεμβαίνουμε είναι:

- layers.c
- network.c
- volume.c

Τα αρχεία layers_baseline.c, network_baseline.c, και volume_baseline.c περιλαμβάνουν την σειριακή έκδοση του κώδικα σαν σημείο αναφοράς για μέτρηση της επιτάχυνσης.

5.3.2 Επισκόπηση Κώδικα

Στον κώδικα χρησιμοποιούνται διάφοροι τύποι δεδομένων. Με τον όρο τύποι δεδομένων, εννοούμε τις δομές που με την typedef έχουν μετονομαστεί ώστε να έχουν νόημα. Επίσης στα header files υπάρχουν περιγραφές για το τι κάνει κάθε συνάρτηση.



images/network.png

Πρώτος τύπος δεδομένων `volume_t`, περιλαμβάνει έναν τρισδιάστατο πίνακα (ή όγκο) μεταβλητών `double`. Χρησιμοποιείται για την αναπαράσταση των αποτελεσμάτων κάθε επιπέδου, καθώς και για τα βάρη κάποιων επιπέδων.

Στη συνέχεια, έχουμε τα διαφορετικά είδη επιπέδων: `conv`, `relu`, `pool`, `fc`, και `softmax`. Για το καθένα από αυτά υπάρχει:

- Μία δομή δεδομένων που περιέχει την περιγραφή των παραμέτρων του επιπέδου. Κάθε επίπεδο έχει συγκεκριμένο αριθμό παραμέτρων που δεν αλλάζει κατά την διάρκεια της εκτέλεσης του προγράμματος. Για παράδειγμα, το μέγεθος των όγκων της εισόδου και εξόδου ενός επιπέδου δηλώνεται στον ορισμό του CNN.
- Μία συνάρτηση `*_forward` η οποία εκτελεί την κύρια λειτουργία του επιπέδου. Αυτές οι συναρτήσεις παίρνουν τη δομή δεδομένων του επιπέδου, και έναν πίνακα `pointers` των όγκων των εισόδων και των εξόδων. Επιπλέον οι μεταβλητές `start` και `end` δείκτες σε αυτό τον πίνακα. Αυτό επιτρέπει σε κάθε επίπεδο να επεξεργάζεται μια ομάδα εισόδων. Για παράδειγμα, μπορεί να ορίσουμε `start = 5` και `end = 9` για την επεξεργασία των εισόδων 5,6,7,8,9 ταυτόχρονα. Τα αποτελέσματα επιστρέφονται στους αντίστοιχους πίνακες εξόδων.
- Δύο συναρτήσεις, την `make_*` και `*_load`. Η πρώτη παράγει ένα επίπεδο και με συγκεκριμένο σύνολο παραμέτρων, και η δεύτερη φορτώνει τα βάρη του επιπέδου από αρχείο `*.txt`.

Τελευταία σημαντική δομή δεδομένων είναι η `network_t`, η οποία περιέχει όλες τις πληροφορίες που περιγράφουν το CNN. Αυτό περιλαμβάνει όλα τα επίπεδα, και μία παράμετρο που αφορά στους διάφορους ενδιάμεσους όγκους (`volumes`). Οι όγκοι (`volumes`) δεν χρησιμοποιούνται για την αποθήκευση δεδομένων, αυτό γίνεται με τα `batches`. Χρησιμοποιούνται για να υπάρχουν διαθέσιμες οι διαστάσεις του κάθε όγκου.

Όλα τα ενδιάμεσα δεδομένα αναπαρίστανται ως `batches`. Περιλαμβάνουν τα ενδιάμεσα δεδομένα που σχετίζονται με ένα σύνολο εικόνων εισόδου. Το `batch_t` είναι συντομογραφία του `volume_t***`, πρόκειται δηλαδή για δισδιάστατους πίνακες δεικτών σε `volumes`. Η πρώτη διάσταση δηλώνει το επίπεδο που ανήκει το `volume` (π.χ., `V0`, `V1`, κλπ. στο σχήμα) και η δεύτερη διάσταση δηλώνει την είσοδο. Στο αρχικό πρόγραμμα τα `batches` έχουν μέγεθος 1 (π.χ., επεξεργαζόμαστε ένα παράδειγμα κάθε φορά), για παραλληλοποίηση θα χρειαστούν μεγαλύτερα `batches`, τα οποία κάνουμε `allocate` στη μνήμη με την συνάρτηση `make_batch` και απελευθερώνουμε από τη μνήμη με την `free_batch`.

Τέλος, η συνάρτηση `net_forward` δέχεται ένα `batch` (καθώς και δείκτες `start/end`) και εφαρμόζει το CNN για κάθε είσοδο από την αρχή μέχρι το τέλος καλώντας τις συναρτήσεις `forward` για κάθε επίπεδο. Αυτή η συνάρτηση χρησιμοποιείται από την `net_classify`. Παίρνει ένα σύνολο εικόνων, βάζει κάθε μία από αυτές στο `volume V0` ενός στοιχείου του `batch`, και στη συνέχεια τρέχει το δίκτυο σε αυτά. Αποθηκεύουμε τα `likelihoods` (δηλαδή τις τιμές στο τελευταίο επίπεδο του δικτύου) σε έναν 2D πίνακα τύπου `double` ώστε να δούμε που θα ταξινομούσε το δίκτυο την εικόνα.

5.4 Μεταφορά Παραμέτρων Νευρωνικού Δικτύου στην GPU

Το `volume`, χρησιμοποιείται για την αναπαράσταση ενεργοποιήσεων (δηλαδή των καταστάσεων) μεταξύ των διαφορετικών επιπέδων του CNN. Έχει τρεις διαστάσεις. Η ερμηνεία τους εξαρτάται από το επίπεδο που τα παράγαγε. Στο πρώτο επίπεδο, το `volume` περιέχει τα δεδομένα των εικόνων που θέλουμε να ταξινομήσουμε (όπου βάθος - `depth` είναι τα τρία χρωματικά κανάλια). Στο τελευταίο επίπεδο το `volume` περιέχει τις πιθανότητες μία εικόνα να ανήκει σε μία κατηγορία. Τα βάρη - `weights` αναπαρίστανται σαν array μίας διάστασης με μέγεθος [πλάτοςύψοςβάθος].

```
typedef struct volume {  
    int width;  
    int height;  
    int depth;  
    double *weights;  
} volume_t;
```

Η αρχιτεκτονική του νευρωνικού δικτύου ορίζεται ως εξής.

```
#define NUM_LAYERS 11  
  
#define NUM_CLASSES 10  
typedef struct network {  
    volume_t *layers[NUM_LAYERS + 1];  
  
    conv_layer_t *10;  
    relu_layer_t *11;  
    pool_layer_t *12;  
    conv_layer_t *13;  
    relu_layer_t *14;  
    pool_layer_t *15;  
    conv_layer_t *16;  
    relu_layer_t *17;  
    pool_layer_t *18;  
    fc_layer_t *19;  
    softmax_layer_t *110;  
} network_t;
```

Αποτελείται από 11 επίπεδα (`NUM_LAYERS 11`), στο `volume_t *layers[]` είναι αποθηκευμένα οι είσοδοι/ έξοδοι κάθε επιπέδου. Το `layers[0]` είναι η εικόνα εισόδου στο νευρωνικό δίκτυο, μεγέθους 32x32x3 και στο `net->layers[0]->weights` βρίσκονται αποθηκευμένα τα pixel της εικόνας. Τα `layers[1]-layers[11]` περιλαμβάνουν τις εξόδους του κάθε επιπέδου αντίστοιχα. Οι μεταβλητές 10 - 110 περιλαμβάνουν τις παραμέτρους του κάθε επιπέδου ανάλογα με τη λειτουργία του καθενός.

- 10,13 και 16 συνελικτικά επίπεδα, `conv_layer_t`
- 11,14 και 17 επίπεδα ReLU, `relu_layer_t`
- 12,15 και 18 επίπεδα Pooling, `pool_layer_t`
- 19 πλήρως διασυνδεδεμένο επίπεδο, `fc_layer_t`

- 110 επίπεδο Softmax, softmax_layer_t

Η συνάρτηση `make_network()` δεσμεύει χώρο μνήμης για το νευρωνικό δίκτυο και αρχικοποιεί βασικές παραμέτρους.

Για να γίνει η μεταφορά των δεδομένων στην κάρτα γραφικών - device, χρειάζεται να κάνουμε *allocate* τη δομή `net` στο device.

```
network_t *make_network() {
    network_t *net = (network_t *) malloc(sizeof(network_t));
    #pragma acc enter data create (net[0:1])
    net->layers[0] = make_volume(32, 32, 3, 0.0);
    net->l0 = make_conv_layer(32, 32, 3, 5, 16, 1, 2);
    //Code...
}
```

Η *directive* `enter data create` χρησιμοποιείται για τον ορισμό μιας μεταβλητής στη συσκευή και μπορεί να χρησιμοποιηθεί αμέσως μετά την εκχώρηση μιας δυναμικής μεταβλητής μέσω `new`, `malloc` ή `allocate` για να ταιριάζει με τη δημιουργία της μεταβλητής του *host*. Παρομοίως, η *directive* `exit data delete` τοποθετείται λίγο πριν τις εντολές `freed`, `deleted` ή `deallocated` στον *host*. [robFab]

Για κάθε μεταβλητή μέλος του `net`, η απαραίτητη δέσμευση μνήμης στο *device* γίνεται μέσω των αντίστοιχων συναρτήσεων `make_*`.

Στη συνάρτηση `make_volume()` τοποθετούμε `enter data create` μετά το `malloc` για το `new_vol` και ξεχωριστά για την μεταβλητή `weights` καθώς πρόκειται για δυναμική μεταβλητή. Τέλος, με την `update device(new_vol[0:1], new_vol->weights[0:(width * height * depth)])` αντιγράφουμε στη συσκευή τα δεδομένα του `new_vol`.

```
volume_t *make_volume(int width, int height, int depth, double value) {
    volume_t *new_vol = malloc(sizeof(struct volume));
    #pragma acc enter data create (new_vol[0:1])
    new_vol->weights = malloc(sizeof(double) * width * height * depth);
    #pragma acc enter data create (new_vol->weights[0:(width * height * depth)])
    new_vol->width = width;
    new_vol->height = height;
    new_vol->depth = depth;

    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            for (int d = 0; d < depth; d++) {
                volume_set(new_vol, x, y, d, value);
            }
        }
    }

    #pragma acc update device (new_vol[0:1], \
        new_vol->weights[0:(width * height * depth)])
    return new_vol;
}
```

Για την αποδέσμευση της μνήμης, στη συνάρτηση `free_volume()` προσθέτουμε `exit data delete(v->weights, v)`

```
void free_volume(volume_t *v) {
    free(v->weights);
    free(v);
    #pragma acc exit data delete(v->weights,v)
}
```

Για κάθε συνάρτηση `make_*`(), τοποθετούμε αντίστοιχα *directive* μετά από κάθε `malloc` και αντιγράφουμε τα δεδομένα στο τέλος της συνάρτησης. Όπου έχουμε κλήση της `make_volume()` δεν χρειάζεται κάποιο επιπλέον *directive*.

Convolutional (layers.c:21-54)

```
conv_layer_t *make_conv_layer(int input_width, int input_height, int input_depth,
    int filter_width, int num_filters, int stride, int pad) {
    conv_layer_t *l = (conv_layer_t *) malloc(sizeof(conv_layer_t));
    #pragma acc enter data create(l[0:1])
    l->output_depth = num_filters;

    //other parameters....
    #pragma acc update device(l[0:1])

    l->filters = malloc(sizeof(volume_t *) * num_filters);
    #pragma acc enter data create(l->filters[0:num_filters])
    for (int i = 0; i < num_filters; i++) {
        l->filters[i] = make_volume(l->filter_width, l->filter_height,
            l->input_depth, 0.0);
    }

    l->bias = 0.0;
    #pragma acc update device(l->bias)
    l->biases = make_volume(1, 1, l->output_depth, l->bias);

    return l;
}
```

ReLU (layers.c:167-179)

```
relu_layer_t *make_relu_layer(int input_width, int input_height,
    int input_depth) {
    relu_layer_t *l = (relu_layer_t *) malloc(sizeof(relu_layer_t));
    #pragma acc enter data create(l[0:1])
    l->input_depth = input_depth;
    // other parameters...
    #pragma acc update device(l[0:1])
    return l;
}
```

Pool (layers.c:196-213)

```
pool_layer_t *make_pool_layer(int input_width, int input_height, int input_depth,
    int pool_width, int stride) {
    pool_layer_t *l = (pool_layer_t *) malloc(sizeof(pool_layer_t));
    #pragma acc enter data create(l[0:1])
    l->pool_width = pool_width;
    //other parameters...
    #pragma acc update device(l[0:1])
    return l;
}
```

Fully Connected (layers.c:261-284)

```
fc_layer_t *make_fc_layer(int input_width, int input_height, int input_depth,
    int num_neurons) {
    fc_layer_t *l = (fc_layer_t *) malloc(sizeof(fc_layer_t));
    #pragma acc enter data create(l[0:1])
    l->output_depth = num_neurons;
    //other parameters...
    #pragma acc update device(l[0:1])
    l->filters = (volume_t **) malloc(sizeof(volume_t *) * num_neurons);
    #pragma acc enter data create(l->filters[0:num_neurons])
    for (int i = 0; i < l->output_depth; i++) {
        l->filters[i] = make_volume(1, 1, l->num_inputs, 0.0);
    }

    l->bias = 0.0;
    #pragma acc update device(l->bias)
    l->biases = make_volume(1, 1, l->output_depth, l->bias);

    return l;
}
```

Softmax (layers.c:331-347)

```
softmax_layer_t *make_softmax_layer(int input_width, int input_height,
    int input_depth) {
    softmax_layer_t *l = (softmax_layer_t *) malloc(sizeof(softmax_layer_t));
    #pragma acc enter data create(l[0:1])
    l->input_depth = input_depth;
    // other parameters...
    #pragma acc update device(l[0:1])
    l->likelihoods = (double *) malloc(sizeof(double) * l->output_depth);
    #pragma acc enter data create(l->likelihoods[0:l->output_depth])

    return l;
}
```


Στην συνάρτηση `free_network()` προσθέτουμε *directives* `exit data delete` πριν από κάθε `free`.

```
void free_network(network_t *net) {
    for (int i = 0; i < NUM_LAYERS + 1; i++){
        free_volume(net->layers[i]);

        // Free each conv layer's filters and biases
        for (int f = 0; f < net->l0->output_depth; f++) {
            free_volume(net->l0->filters[f]);
        }
        #pragma acc exit data delete(net->l0->filters)
        free(net->l0->filters);
        free_volume(net->l0->biases);

        for (int f = 0; f < net->l3->output_depth; f++) {
            free_volume(net->l3->filters[f]);
        }
        #pragma acc exit data delete(net->l3->filters)
        free(net->l3->filters);
        free_volume(net->l3->biases);

        for (int f = 0; f < net->l6->output_depth; f++) {
            free_volume(net->l6->filters[f]);
        }
        #pragma acc exit data delete(net->l6->filters)
        free(net->l6->filters);
        free_volume(net->l6->biases);

        // Free FC layer filters and biases
        for (int f = 0; f < net->l9->output_depth; f++) {
            free_volume(net->l9->filters[f]);
        }
        #pragma acc exit data delete(net->l9->filters)
        free(net->l9->filters);
        free_volume(net->l9->biases);

        // Free softmax layer likelihoods
        #pragma acc exit data delete(net->l10->likelihoods)
        free(net->l10->likelihoods);

        #pragma acc exit data delete(net->l0,net->l1,net->l2,net->l3,net->l4,\
            net->l5,net->l6,net->l7,net->l8,net->l9,net->l10)
        free(net->l0);
        //free l1-l10...
        #pragma acc exit data delete(net)
        free(net);
    }
}
```

Επιπλέον, στις συναρτήσεις `copy_volume`, `conv_load` και `fc_load` έχουν προστεθεί τα απαραίτητα *directives* για ενημέρωση των παραμέτρων.

```
convload(layers.c:128 – 165)

void conv_load(conv_layer_t *l, const char *file_name) {
    int filter_width, filter_height, depth, filters;

    FILE *fin = fopen(file_name, "r");
    //Code...
    //Update Weights and Biases on Device
    int we=filter_width*filter_height*depth;
    for(int f = 0; f < filters; f++) {
#pragma acc update device(l->filters[f]->weights[0:we])
    }

#pragma acc update device(l->biases->weights[0:l->output_depth])

    fclose(fin);
}

fcload(layers.c:305 – 329)

void fc_load(fc_layer_t *l, const char *filename) {
    FILE *fin = fopen(filename, "r");
    //...Code...

    //Update Weights and Biases on Device
    for(int f = 0; f < l->output_depth; f++) {
#pragma acc update device(l->filters[f]->weights[0:l->num_inputs])
    }
#pragma acc update device(l->biases->weights[0:l->output_depth])
    fclose(fin);
}
```

Εκτέλεση Κώδικα

Χωρίς -acc

```
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
16 972 microseconds
```

Με -acc

```
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
167 979 microseconds
```

Κεφάλαιο 6

Επίλογος

6.1 Συμπεράσματα

6.2 Μελλοντικές Επεκτάσεις

Παράρτημα Α΄

Παράρτημα

Απενδιζ γοες ηερε...

Βιβλιογραφία

- [1] CS 61C. *Project 4: Performance Programming*. URL: <https://inst.eecs.berkeley.edu/~cs61c/sp19/projects/proj4/>.
- [2] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. *The CIFAR-10 dataset*. URL: <https://www.cs.toronto.edu/~kriz/cifar.html>.
- [3] CS231n. *Convolutional Neural Networks for Visual Recognition*. URL: <http://cs231n.github.io>.
- [4] HPC @ LLNL. *Introduction to parallel computing tutorial*. URL: <https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial>.
- [5] Stanford Edu. *CPU Database*. URL: <http://cpudb.stanford.edu/>.
- [6] Alex Krizhevsky, Geoffrey Hinton, et al. «Learning multiple layers of features from tiny images». 2009.
- [7] Alex Krizhevsky. *Convolutional Neural Networks for Object Classification in CUDA*. 2009.
- [8] Udacity. *CS344: Intro to Parallel Programming*. URL: <https://youtube.com/playlist?list=PLAwTw4SYaPm0z11jGTXRF7RuEEAgsIwH>.
- [9] GitHub. *Starter code for CS 61C Project 4 (Performance Programming) for the Spring 2019 semester*. URL: <https://github.com/61c-teach/sp19-proj4-starter>.