Πανεπιστήμιο Ιωαννίνων
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Μελέτη και Υλοποίηση Νευρωνικών Δικτύων σε Υπολογιστικά Περιβάλλοντα Βαθειάς Μάθησης

Πτυχιακή Εργασία
του
Ιωάννη Σαμαρτζή

Επιβλέπων: Σταύρος Π. Αδάμ
Επίκουρος Καθηγητής

30 Σεπτεμβρίου 2021

Study and Implementation of Neural Networks in Deep Learning Computing Environments

Εγκρίθηκε από τριμελή εξεταστική επιτροπή
Άρτα, 30/9/21

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων Καθηγητής
   Σταύρος Π. Αδάμ
   Επίκουρος καθηγητής

2. Μέλος Επιτροπής
   Γρηγόριος Δουμένης
   Επίκουρος καθηγητής

3. Μέλος Επιτροπής
   Δημήτριος Λιαροκάπης
   Λέκτορας

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Σαμαρτζής Ιωάννης

Υπογραφή

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω θερμά τον καθηγητή κ. Σταύρο Αδάμ κυρίως για την εμπιστοσύνη που μου έδειξε, καθώς επίσης και για την πολύτιμη βοήθεια και καθοδήγηση στην επίλυση διάφορων θεμάτων κατά τη διάρκεια υλοποίησης της πτυχιακής εργασίας.

Θα ήθελα επίσης να απευθύνω τις ευχαριστίες μου στους γονείς μου, οι οποίοι στήριξαν τις σπουδές μου με διάφορους τρόπους, φροντίζοντας για την καλύτερη δυνατή μόρφωση μου.

Περίληψη

Η βαθιά μάθηση είναι μια τεχνολογία **τεχνητής νοημοσύνης** (AI) που μιμείται τη λειτουργία του ανθρώπινου εγκεφάλου, επεξεργάζεται δεδομένα και δημιουργεί πρότυπα για χρήση στη λήψη αποφάσεων. Τα πιό γνωστά περιβάλλοντα σχεδίασης και ανάπτυξης νευρωνικών δικτύων βαθιάς μάθησης βασίζονται στη γλώσσα προγραμματισμού **python**. Αν ένα πρόβλημα λυθεί σε **python**, η μεταφορά του δικτύου βαθιάς μάθηση σε κάποιο άλλο σύστημα δεν είναι εύκολη, δεδομένου ότι είναι μια γλώσσα που έχει κατασκευαστεί με σκοπό την αναγνωσιμότητα του κώδικα και η σύνταξή του να επιτρέπει στους προγραμματιστές να εκφράζουν έννοιες σε λιγότερες γραμμές κώδικα. Αυτό την καθιστά δύσκολη στο να αναλυθεί και να επεκταθεί σε κάποιο άλλο περιβάλλον. Στην παρούσα πτυχιακή γίνεται μελέτη στα **δίκτυα βαθιάς μάθησης** και συγκεκριμένα στα **συνελικτικά δίκτυα** (που χρησιμοποιούνται κυρίως σε προβλήματα ταξινόμησης εικόνων), με διάφορες βιβλιοθήκες στην **python3**. Πραγματοποιείται επίδειξη μιας υλοποίησης που δημιουργεί ένα δίκτυο στο περιβάλλον της C++ με δεδομένα από το περιβάλλον του **pytorch**. Τέλος, γίνεται εκτίμηση της εγκυρότητας και της ακρίβειας του λογισμικού σε σχέση με το **pytorch** (χρησιμοποιώντας μια τυχαία εικόνα), με στόχο τη μελέτη των προοπτικών της χρήσης του σε εφαρμογές και ενσωματωμένα συστήματα.

Λέξεις-κλειδιά: Python, Pytorch, Τεχνητή νοημοσύνη, Δίκτυα βαθιάς μάθησης, Συνελικτικά δίκτυα

Abstract

Deep learning is an **artificial intelligence** technology that mimics the functioning of the human brain, it processes data and creates patterns for decision making. The most popular deep learning neural network design and development environments are based on the **Python** programming language. If a problem is solved in python, deploying the deep learning network to another system is not straightforward and not as easy as it might sound to be, because **Python** is a language designed for code readability and also allow developers to express concepts in fewer lines of code. This makes it difficult to analyze and extend to another environment. In this dissertation we study about **deep learning networks** and in particular on the subject of **convolutional neural networks** (used mainly for image data and classification problems), applying various libraries of **python3**. Then, an implementation that creates a network in C++ with data from **pytorch** is demonstrated. Finally, the validity and accuracy of the software is evaluated in relation to pytorch (using a random image), with the aim of studying the prospects of its use in applications and embedded systems.

Keywords:python, pytorch, artificial intelligence, deep learning networks, convolutional neural networks

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1

Εισαγωγή

1.1 Στόχοι και δομή της πτυχιακής

Οι στόχοι της πτυχιακής αφορούν την μελέτη των δικτύων βαθιάς μάθησης με έμφαση στα συνελικτικά δίκτυα, στην python, την ανάλυση των στρωμάτων τους και τη περιγραφή του περιβάλλοντος ανάπτυξης. Επίσης μελετάται και αναλύεται η διαδικασία μεταφοράς του αποτελέσματος σχεδίαης ενός συνελικτικού δικτύου από ένα περιβάλλον βασισμένο σε κώδικα πψτηον ή C++ ανεξάρτητα από κάποιο συγκεκριμένο περιβάλλον. Τέλος, περιγράφεται το λογισμικό, που κατασκευάστηκε με σκοπό να ενσωματώνει συνελικτικά δίκτυα στη C++ , τα οποία έχουν εκπαιδευτεί στο pytorch .

Δομή της Πτυχιακής

Εκτός από το συγκεκριμένο πρώτο κεφάλαιο όπου περιγράφεται το θέμα που πραγματεύεται η πτυχιακή οι υπόλοιπες ενότητες έχουν ως εξής:

Το δεύτερο κεφάλαιο αφορά στο θεωρητικό υπόβαθρο της πτυχιακής. Περιγράφεται η μηχανική μάθηση μαζί με τα δίκτυα βαθιάς μάθησης και έπειτα τα συνελικτικά νευρωνικά δίκτυα. Στη συνέχεια περιγράφεται το περιβάλλον ανάπτυξης λογισμικού της python, ο τρόπος λειτουργίας της, και οι δυνατότητες που παρέχει για τα δίκτυα βαθιάς μάθησης. Τέλος γίνονται κάποιες αναφορές στις εφαρμογές που χρησιμοποιούν συνελικτικά δίκτυα.

Στο τρίτο κεφάλαιο περιγράφεται το πρόβλημα που υπάρχει με την μεταφορά εκπαιδευμένων δικτύων από ένα περιβάλλον σε κάποιο άλλο. Ενδεικτικα παρουσιάζονται διάφορες υλοποιήσεις για την επίλυση αυτού του προβλήματος, ενώ στο τέλος του κεφαλαίου παρουσιάζονται υπάρχουσες υλοποιήσεις που λύνουν αυτό το πρόβλημα. Στο τέλος του κεφαλαίου παρουσιάζεται η δική μου πρόταση για την επίλυση αυτού του προβλήματος.

Στο τέταρτο κεφάλαιο παρουσιάζονται τα εργαλεία που χρησιμοποιήθηκαν για να υλοποιηθεί ένα λογισμικό που λύνει το πρόβλημα που περιγράφεται στο τρίτο κεφάλαιο, έχουν γίνει δύο υλοποιήσεις, μια σε python και μία σε C++. Τέλος συγκρίνονται τα αποτελέσματα που βγάζουν τα λογισμικά σε σχέση με τα αρχικά αποτελέσματα που πήραμε από ένα παράδειγμα εκπαίδευσης σε pytorch.

Στο πέμπτο και τελευταίο κεφάλαιο γίνεται ανακεφαλαίωση, της μέχρι τώρα έρευνας και υλοποίησης πάνω στο αντικείμενο που απασχολεί η συγκεκριμένη εργασία. Γίνεται ανάλυση και συζήτηση των συμπερασμάτων και επίσης αναφορά σε μελλοντικές αναβαθμίσεις του ήδη υπάρχοντος λογισμικού.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 Μηχανική μάθηση και δίκτυα βαθιάς μάθησης

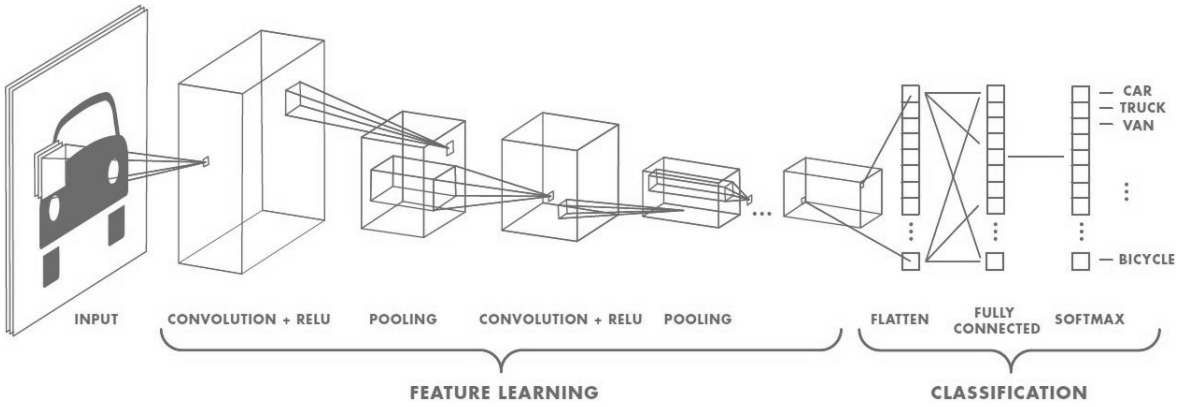
Η βαθιά μάθηση είναι ένα από τα υποσύνολα της μηχανικής μάθησης που χρησιμοποιεί αρχιτεκτονικές μηχανών μάθησης όπως τα νευρωνικά δίκτυα που χαρακτηρίζονται από ένα σημαντικό αριθμό διαφορετικών επιπέδων επεξεργασίας. Τα επίπεδα αυτά που συχνά ονομάζονται και στοιβαγμένες αρχιτεκτονικές (στασκεδ αρσηιτεστυρες) έχουν αποδειχτεί ιδιαίτερα αποτελεσματικά στην επίλυση σύνθετων προβλημάτων ταξινόμησης σύνθετων δεδομένων όπως οι εικόνες, οι εικονοροές και οι πολυπλοκες χρονοσειρές.

Συνήθως, η βαθιά μάθηση είναι μια διαδικασία εκπάιδευσης του μηχανισμού μάθησης (learner) χωρίς επίβλεψη ή με ημιεπίβλεψη. Η βαθιά μάθηση βασίζεται στη μάθηση με αναπαράσταση. Αντί να χρησιμοποιεί αλγόριθμους για συγκεκριμένες εργασίες, μαθαίνει από αντιπροσωπευτικά παραδείγματα. Για παράδειγμα, εάν θέλουμε να δημιουργήσουμε ένα μοντέλο που θα αναγνωρίζει ένα πρότυπο, θα πρέπει να προετοιμαστεί ένα σύνολο δεδομένων που περιλαμβάνει πολλές διαφορετικές εικόνες από το συγκεκριμένο πρότυπο. Η μηχανική μάθηση επιχειρεί να αποσπάσει νέες γνώσεις από ένα μεγάλο σύνολο προ επεξεργασμένων δεδομένων, ακολουθεί ένα σύνολο κανόνων που καθορίζονται από ειδικούς.

Τα δίκτυα βαθιάς μάθησης χρησιμοποιούν μεγάλες ποσότητες δεδομένων, χρειάζονται περισσότερο χρόνο για να εκπαιδευτούν και εξάγουν ακριβή συμπεράσματα από ακατέργαστα δεδομένα, χωρίς να γνωρίζουμε ποια είναι τα ιδιαίτερα χαρακτηριστικά που αντιπροσωπεύουν οι νευρώνες και μπορούν να χρησιμοποιηθούν για την επίλυση πολλών προβλημάτων. Ένα από τα πλεονεκτήματα που προσφέρουν αυτά τα δίκτυα είναι η ικανότητά τους να εντοπίσουν μοτίβα και ανωμαλίες σε μεγάλο όγκο ακατέργαστων δεδομένων, επιτρέποντάς μας να κάνουμε ανακαλύψεις σε δεδομένα ακόμη και όταν δεν είμαστε σίγουροι τι προσπαθούμε να βρούμε καθώς δεν είναι εύκολο να αιτιολογήσουμε τα συμπεράσματα που προκύπτουν. Επομένως, είναι δύσκολο να εκτιμηθεί η απόδοση του μοντέλου επειδή δεν γνωρίζουμε ποια είναι η έξοδος που πρέπει να έχει.

Τέλος η κατασκευή αλγορίθμων βαθιάς εκμάθησης είναι πολύ δαπανηρή. Χρειάζεται εξειδικευμένο προσωπικό που έχει εκπαιδευτεί να εργάζεται με εξελιγμένα μαθηματικά. Επιπλέον, η βαθιά μάθηση χρειάζεται πολλούς υπολογιστικούς πόρους. Απαιτούνται ισχυροί επεξεργαστές, κάρτες γραφικών και πολλή μνήμη για την εκπαίδευση των μοντέλων. Απαιτείται μεγάλος αποθηκευτικός χώρος για την αποθήκευση δεδομένων εισόδου, παραμέτρων βαρών και λειτουργιών ενεργοποίησης. Μερικές φορές οι αλγόριθμοι βαθιάς μάθησης γίνονται τόσο πεινασμένοι για υπολογιστική δύναμη που οι χρήστες προτιμούν άλλους αλγόριθμους, θυσιάζοντας την ακρίβεια των προβλέψεων [1].

2.2 Συνελικτικά νευρωνικά δίκτυα



Σχήμα 2.1: Οπτική αναπαράσταση συνελικτικού νευρωνικού δικτύου

Τα νευρωνικά δίκτυα όπως περιγράφεται στο [2], αποτελούν υποσύνολο της μηχανικής μάθησης και βρίσκονται στο επίκεντρο των τεχνικών βαθιάς εκμάθησης. Αποτελούνται από στρώματα κόμβων, που περιέχουν ένα επίπεδο εισόδου, ένα ή περισσότερα κρυμμένα στρώματα και ένα επίπεδο εξόδου. Κάθε κόμβος συνδέεται με έναν άλλο και έχει σχετικό βάρος και κατώφλι.

Εάν η έξοδος οποιουδήποτε μεμονωμένου κόμβου είναι πάνω από την καθορισμένη τιμή κατωφλίου, ο κόμβος αυτός ενεργοποιείται, στέλνοντας δεδομένα στο επόμενο επίπεδο του δικτύου. Διαφορετικά, δεν μεταβιβάζονται δεδομένα στο επόμενο επίπεδο του δικτύου.

Τα συνελικτικά νευρωνικά δίκτυα διακρίνονται από τα άλλα νευρωνικά δίκτυα για την ανώτερη απόδοσή τους με είσοδο δεδομένων εικόνας. Έχουν τρεις κύριους τύπους στρωμάτων, τα οποία είναι:

- Convolutional Layer
- Pooling Layer
- Fully Connected Layer

Το convolutional layer είναι το πρώτο στρώμα ενός συνελικτικού δικτύου. Μπορούν να ακολουθηθούν από πρόσθετα convolution layer ή pooling layer. Το Fully-connected layer είναι πολλαπλά στρώματα πρόσθιας τροφοδότησης. Με κάθε στρώμα, τα συνελικτικά δίκτυα αυξάνουν την πολυπλοκότητά, προσδιορίζοντας μεγαλύτερα τμήματα της εικόνας.

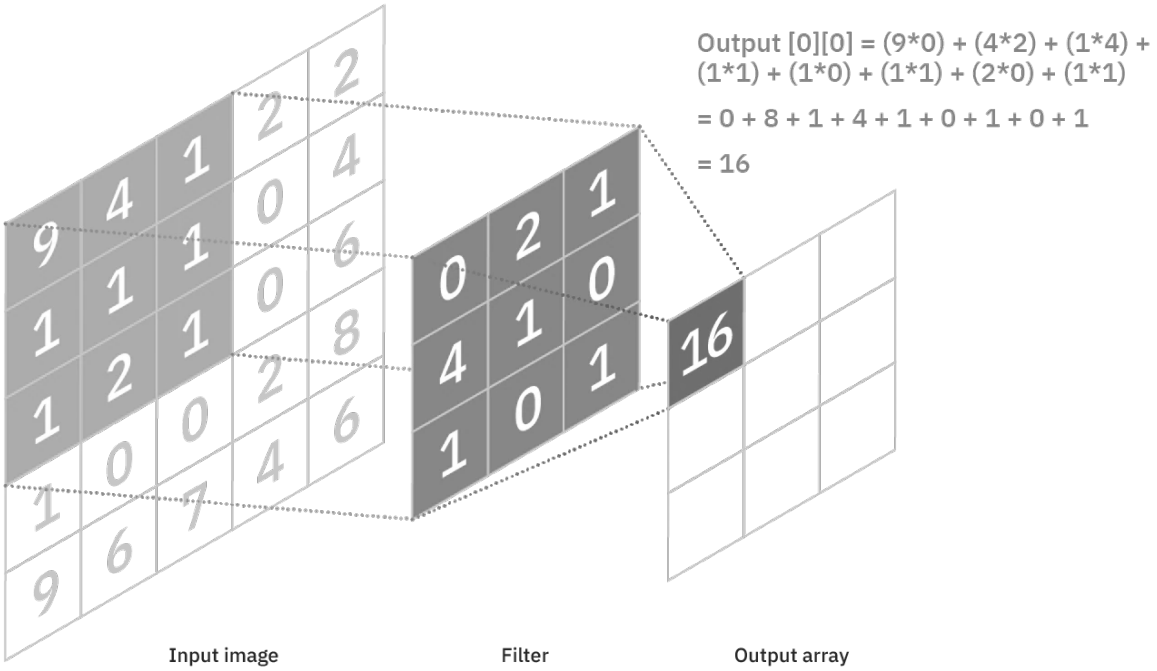
Τα πρώτα επίπεδα εστιάζουν σε απλά χαρακτηριστικά, όπως χρώματα και άκρες. Καθώς τα δεδομένα εικόνας προχωρούν μέσα από τα στρώματα του δικτύου, αρχίζουν να αναγνωρίζουν μεγαλύτερα στοιχεία ή σχήματα του αντικειμένου μέχρι να προσδιοριστεί τελικά το αντικείμενο.

2.2.1 Convolutional Layer

Το convolutional layer είναι το βασικό δομικό στοιχείο ενός συνελικτικού δικτύου και είναι εκεί όπου συμβαίνει το μεγαλύτερο μέρος της επεξεργασίας των δεδομένων εισόδου. Απαιτεί μερικά στοιχεία, τα οποία είναι τα δεδομένα εισόδου, ένα φίλτρο και ένας χάρτης χαρακτηριστικών.

Ας υποθέσουμε ότι η είσοδος είναι μια έγχρωμη εικόνα. Αυτό σημαίνει ότι η είσοδος θα έχει τρεις διαστάσεις, (δηλαδή τρία κανάλια χρώματος) που αντιστοιχούν σε RGB. Τα convolutional layers χρησιμοποιούν έναν ανιχνευτή χαρακτηριστικών γνωστός επίσης ως kernel ή φίλτρο, ο οποίος θα κινείται στα πεδία της εικόνας, ελέγχοντας αν υπάρχει κάποιο χαρακτηριστικό. Αυτή η διαδικασία είναι γνωστή ως συνέλιξη.

Ο ανιχνευτής χαρακτηριστικών είναι μια δισδιάστατη συστοιχία βαρών, η οποία αντιπροσωπεύει μέρος της εικόνας. Παρόλο που τα φίλτρα μπορούν να ποικίλουν σε μέγεθος, το σύνηθες είναι ένας πίνακας μεγέθους 3x3. Αυτό καθορίζει επίσης το μέγεθος του δεκτικού πεδίου, δηλαδη την περιοχή της εικόνας εισόδου που επηρεάζεται από το φίλτρο. Το φίλτρο εφαρμόζεται στη συνέχεια σε μια περιοχή της εικόνας και υπολογίζεται το εσωτερικό γινόμενο μεταξύ των εικονοστοιχείων εισόδου και του φίλτρου. Έπειτα ο πίνακας εξόδου τροφοδοτείται με το αποτέλεσμα του εσωτερικού γινομένου. Στη συνέχεια, το φίλτρο μετατοπίζεται κατά ένα βήμα, επαναλαμβάνοντας τη διαδικασία έως ότου περάσει απ' ολόκληρη την εικόνα. Η τελική έξοδος από τη σειρά εσωτερικών γινομένων, είναι γνωστή ως χάρτης χαρακτηριστικών.



Σχήμα 2.2: Διαδικασία συνέλιξης

Τα βάρη στον ανιχνευτή χαρακτηριστικών παραμένουν σταθερά καθώς κινείται πάνω στην εικόνα. Ορισμένες παράμετροι, όπως οι τιμές των βαρών, προσαρμόζονται κατά τη διάρκεια της εκπαίδευσης μέσω της διαδικασίας της αντίστροφης διάδοσης (back-propagation) .

Ωστόσο, υπάρχουν τρεις υπερπαράμετροι που επηρεάζουν το μέγεθος της εξόδου που πρέπει να ρυθμιστούν πριν ξεκινήσει η εκπαίδευση του νευρωνικού δικτύου.

1. Ο **αριθμός των φίλτρων** επηρεάζει το βάθος της εξόδου. Για παράδειγμα, τρία ξεχωριστά φίλτρα θα έδιναν τρεις διαφορετικούς χάρτες χαρακτηριστικών, δημιουργώντας βάθος τριών χαρακτηριστικών.

2. **Stride** ή αλλιώς βήμα, είναι η απόσταση ή ο αριθμός των εικονοστοιχείων, που διασκελίζει το φίλτρο καθώς προχωράει πάνω από τον πίνακα εισόδου. Όσο μεγαλύτερο το βήμα τόσο μικρότερος θα είναι ο χάρτης χαρακτηριστικών στην έξοδο.

3. **Padding** είναι μια λειτουργία που χρησιμοποιείται συνήθως όταν το φίλτρο δεν ταιριάζει στην εικόνα εισόδου, δηλαδή όταν ένα μέρος του βρήσκετε εκτός της εικόνας εισόδου. Η λειτουργία padding θέτει όλα τα στοιχεία που είναι εκτός της εικόνας εισόδου στο μηδέν, παράγοντας μια μεγαλύτερη ή ίσου μεγέθους έξοδο. Υπάρχουν τρεις τύποι padding:

    - **Valid padding**: Σε αυτήν την περίπτωση, η τελευταία συνέλιξη πέφτει έξω εάν οι διαστάσεις δεν ευθυγραμμίζονται.

    - **Same padding**: Διασφαλίζει ότι το επίπεδο εξόδου έχει το ίδιο μέγεθος με το επίπεδο εισόδου.

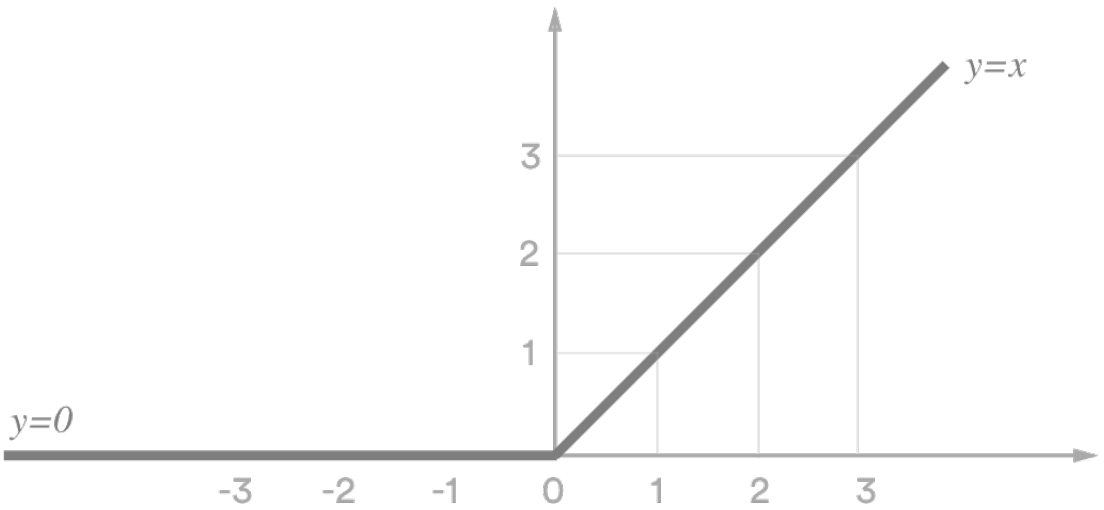
    - **Full padding**: Αυξάνει το μέγεθος της εξόδου προσθέτοντας μηδενικά στα όριο της εισόδου.

2.2.2 Rectified Linear Unit (ReLU)

Μετά από κάθε λειτουργία σύμπτυξης, ένα συνελικτικό δίκτυο εφαρμόζει έναν μετασχηματισμό με Rectified Linear Unit [3] (ReLU) (Μονάδα Γραμμικής Ανόρθωσης, ΜΓΑ) στο χάρτη χαρακτηριστικών, εισάγοντας μη γραμμικότητα στο μοντέλο.

Η μονάδα Γραμμικής Ανόρθωσης (ReLU) χρησιμοποιεί την συνάρτηση ενεργοποίησης:

$$ReLU(x) = max(0, x)$$



Σχήμα 2.3: Συνάρτηση ReLU

Δηλαδή αν η είσοδος είναι θετική περνάει όπως είναι στην έξοδο, ενώ αν είναι αρνητική παράγεται 0 στην έξοδο. Ο λόγος για τον οποίο χρησιμοποιείται η συνάρτηση ReLU, είναι ότι όταν στοιβάζονται όλο και περισσότερα στρώματα σε ένα συνελικτικό δίκτυο, έχει παρατηρηθεί εμπειρικά ότι το δίκτυο είναι πολύ πιο εύκολο και γρηγορότερο στο να εκπαιδευτεί.

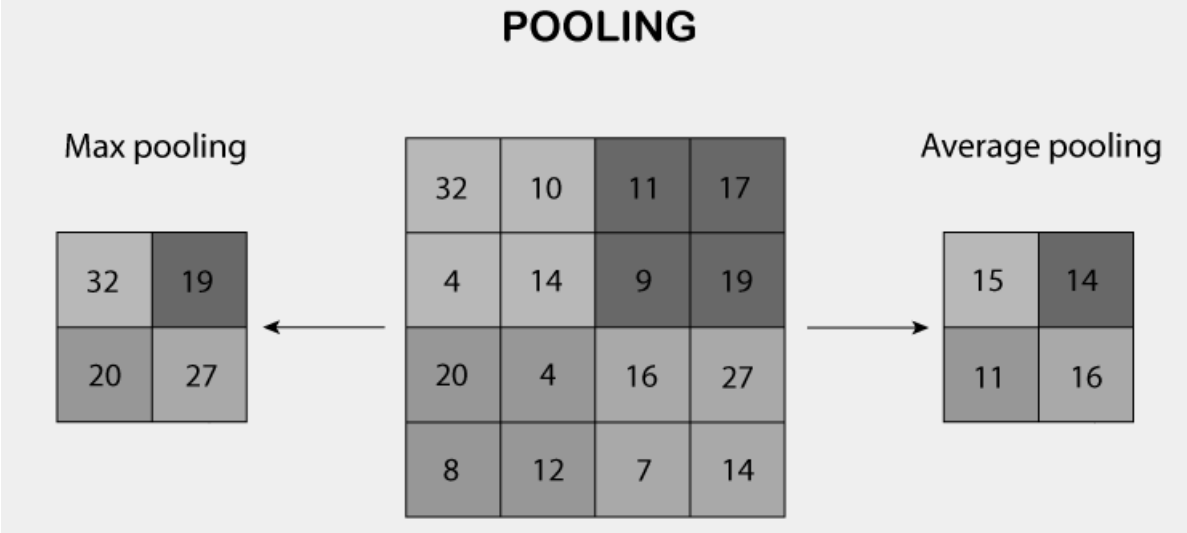
2.2.3 Pooling Layer

Το επίπεδο pooling layer γνωστό και ως downsampling, πραγματοποιεί μείωση διαστάσεων, μειώνοντας τον αριθμό των παραμέτρων στην είσοδο. Με τρόπο όμοιο με το convolutional layer, η λειτουργία pooling σαρώνει με ένα φίλτρο ολόκληρη την είσοδο, αλλά η διαφορά είναι ότι αυτό το φίλτρο δεν έχει βάρη. Αντί αυτού, ο kernel εφαρμόζει μια συνάρτηση συνάθροισης στις τιμές εντός του δεκτικού πεδίου, συμπληρώνοντας έτσι τον πίνακα εξόδου.

Υπάρχουν δύο κύριοι τύποι pooling:

- **Max Pooling**: Καθώς το φίλτρο κινείται κατά μήκος της εισόδου, επιλέγει το εικονοστοιχείο με τη μέγιστη τιμή για αποστολή στον πίνακα εξόδου.
- **Average Pooling**: Καθώς το φίλτρο κινείται κατά μήκος της εισόδου, υπολογίζει τη μέση τιμή μέσα στο δεκτικό πεδίο για αποστολή στον πίνακα εξόδου.

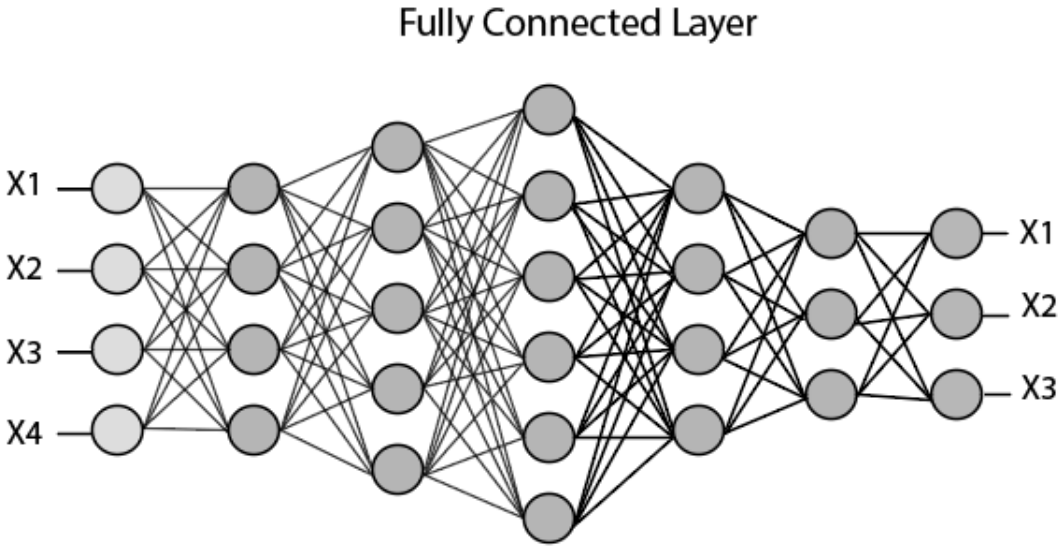
Ενώ πολλές πληροφορίες χάνονται σε αυτό το επίπεδο, έχει επίσης πολλά οφέλη. Η χρήση των pooling layer βοηθάει στη μείωση της πολυπλοκότητας του δικτύου και στον περιορισμό του κινδύνου της υπερπροσαρμογής κατά τη διάρκεια της εκπαίδευσης.



Σχήμα 2.4: Διαδικασία Pooling

2.2.4 Fully Connected Layer

Όπως αναφέραμε προηγουμένως, οι τιμές των εικονοστοιχείων της εικόνας εισόδου δεν συνδέονται άμεσα με το επίπεδο εξόδου σε μερικώς συνδεδεμένα επίπεδα. Ωστόσο, στο πλήρως συνδεδεμένο επίπεδο, κάθε κόμβος στο επίπεδο εξόδου συνδέεται απευθείας με όλους τους κόμβους του προηγούμενου επιπέδου.



Σχήμα 2.5: Πλήρως συνδεδεμένο στρώμα

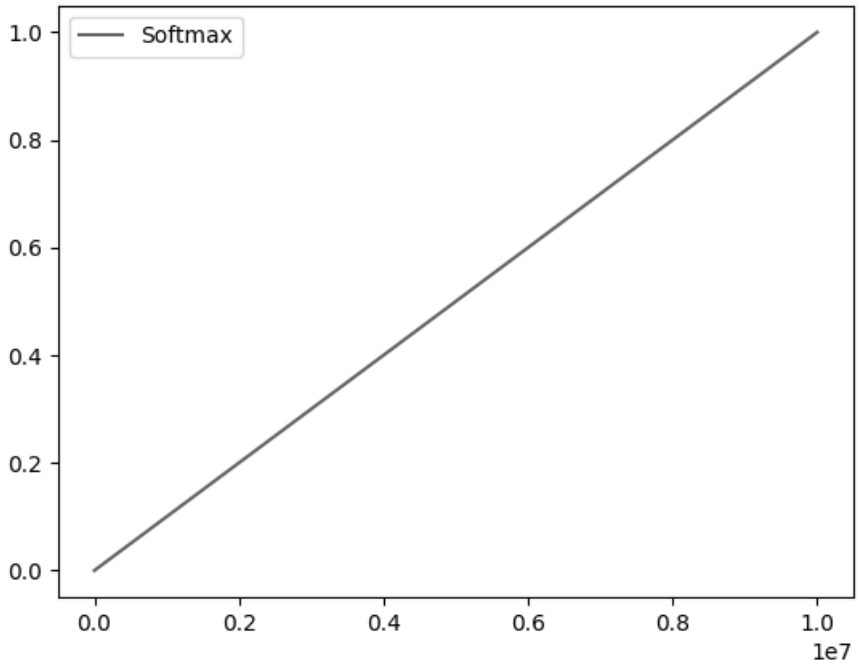
Αυτό το επίπεδο εκτελεί το έργο της ταξινόμησης με βάση τα χαρακτηριστικά που εξήχθησαν από τα προηγούμενα επίπεδα και τα διαφορετικά φίλτρα τους.

2.2.5 Softmax

Ενώ τα convolution και pooling layers τείνουν να χρησιμοποιούν συναρτήσεις ReLu, τα fully connected layers συνήθως αξιοποιούν μια συνάρτηση ενεργοποίησης softmax για να ταξινομούν κατάλληλα τις εισόδους, παράγοντας πιθανότητα στο διάστημα [0, 1].

Η συνάρτηση Softmax ορίζεται με τον παρακάτω τύπο:

$$Softmax(x_1) = \frac{exp(x_i)}{\sum_j exp(x_j)}$$



Σχήμα 2.6: Συνάρτηση Softmax

2.2.6 LogSoftmax

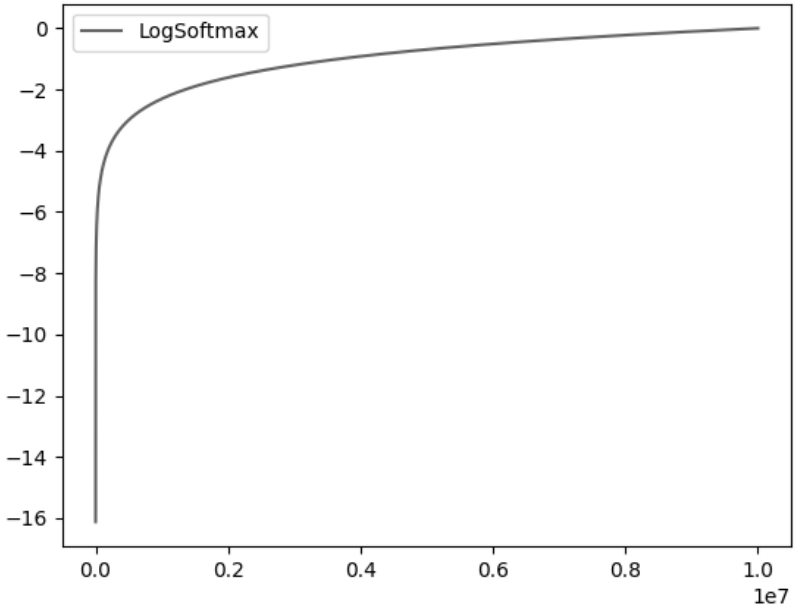
Η αναπαράσταση αριθμών στον υπολογιστή έχει ως συνέπεια αριθμοί που απαιτούν πολύ μεγάλη ακρίβεια να μην είναι δυνατό να αναπαρασταθούν αποτελεσματικά. Έτσι, όταν έχουμε γινόμενο μεταξύ πολύ μικρών ή πολύ μεγάλων πραγματικών αριθμών, το αποτέλεσμα δεν είναι πάντα δυνατό να αναπαρασταθεί και κατά συνέπεια, έχουμε το πρόβλημα underflow (για πολύ μικρούς πραγματικούς αριθμούς) και overflow (για πολύ μεγάλους πραγματικούς αριθμούς). Για να αποφύγουμε αυτά τα φαινόμενα που προκαλούν αστάθεια σε αριθμητικούς αλγορίθμους, όπως οι αλγόριθμοι εκπαίδευσης νευρωνικών δικτύων ή άλλοι αλγόριθμοι μηχανικής μάθησης σε περιπτώσεις όπου η αριθμητική συνάρτηση είναι μονότονη χρησιμοποιούμε αντί για τις αριθμητικές μεταβλητές, τον λογάριθμό τους.

Ο λογάριθμος μιας μονότονης πραγματικής συνάρτησης όπως η περίπτωση της logsoftmax παρουσιάζει την ίδια μονοτονία με αυτή της softmax. Άρα τα συμπεράσματα που βασίζονται στους λογαρίθμους ειναι αντίστοιχα με τα συμπεράσματα που θα είχαμε στην αρχική πραγματική συνάρτηση. Η τιμή της πιθανότητας μεταφέρετε από το διάστημα $[0, 1]$ στο διάστημα $(-\infty, 0]$.

Για παράδειγμα, αν η Softmax δίνει ως αποτέλεσμα την πιθανότητα $0,26 * 10^{-19}$, αν κάνουμε χρήση της LogSoftmax η τιμή του λογάριθμου της πιθανότητας θα είναι $-19,585026652$, ένας πραγματικός αριθμός αρκετά μεγαλύτερος που δέν θα προκαλεί αστάθεια κατά τη χρήση του στους αριθμητικούς αλγορίθμους.

Η συνάρτηση LogSoftmax ορίζεται με τον παρακάτω τύπο:

$$LogSoftmax(x_1) = log\left(\frac{exp(x_i)}{\sum_j exp(x_j)}\right)$$



Σχήμα 2.7: Συνάρτηση LogSoftmax

Σύμφωνα με το [4] η LogSoftmax είναι σε πλεονεκτική θέση σε σχέση με τη Softmax. Η LogSoftmax παρέχει αριθμητική σταθερότητα και αποκλείει τις εξαιρετικά εσφαλμένες κατηγοριοποιήσεις.

Όπως φαίνεται και στα σχήματα 2.5 και 2.6, η LogSoftmax έχει εκθετική φύση, σε αντίθεση με τη Softmax που είναι γραμμική. Δηλαδή όσο μικρότερη η πιθανότητα να έχει γίνει σωστή κατηγοριοποίηση, τόσο περισσότερο θα αποκλείεται η συγκεκριμένη κατηγορία.

2.3 Περιβάλλοντα ανάπτυξης

Keras

Το Keras [5] είναι μια αποτελεσματική βιβλιοθήκη για προγραμματισμό εφαρμογών νευρωνικού δικτύου υψηλού επιπέδου (API) γραμμένη σε Python. Αυτή η βιβλιοθήκη νευρωνικών δικτύων ανοιχτού κώδικα έχει σχεδιαστεί για να παρέχει γρήγορα πειράματα με βαθιά νευρωνικά δίκτυα. Το Keras επικεντρώνεται στο να είναι αρθρωτό, φιλικό προς το χρήστη και επεκτάσιμο. Δεν χειρίζεται υπολογισμούς χαμηλού επιπέδου. Αντί αυτού, βασίζεται σε μια εξειδικευμένη, βελτιστοποιημένη backend βιβλιοθήκη.Αυτή τη στιγμή, το Keras υποστηρίζει δύο βιβλιοθήκες : TensorFlow και Theano. Το Keras υιοθετήθηκε στα μέσα του 2017 και ενσωματώθηκε στο TensorFlow. Οι χρήστες μπορούν να έχουν πρόσβαση σε αυτό μέσω της μονάδας tf.keras. Ωστόσο, η βιβλιοθήκη Keras μπορεί να εξακολουθήσει να λειτουργεί ανεξάρτητα.

TensorFlow

Το TensorFlow [6] είναι ένα πλαίσιο που αναπτύχθηκε από την Google και κυκλοφόρησε το 2015. Το TensorFlow παρέχει μια συμβολική βιβλιοθήκη μαθηματικών που χρησιμοποιείται για νευρωνικά δίκτυα, ιδανική για προγραμματισμό ροής δεδομένων σε διάφορες εργασίες. Παρέχει πολλαπλά επίπεδα αφαίρεσης abstraction για μοντέλα κατασκευής και εκπαίδευσης. Το Tensorflow αποτελεί μια πολλά υποσχόμενη και ταχέως αναπτυσσόμενη είσοδος στον κόσμο της βαθιάς μάθησης, παρέχει ένα ευέλικτο και περιεκτικό οικοσύστημα κοινοτικών πόρων, βιβλιοθηκών και εργαλείων που διευκολύνουν τη δημιουργία και την ανάπτυξη εφαρμογών μηχανικής μάθησης. Επίσης, όπως αναφέρθηκε προηγουμένως, το TensorFlow χρησιμοποιεί το Keras, οπότε φαίνεται προβληματική η σύγκριση των δύο. Ακόμα, ειδικά οι χρήστες του Keras δεν χρειάζεται απαραίτητα να χρησιμοποιούν το TensorFlow.

Theano

Το Theano [7] ήταν από τις πιο δημοφιλείς βιβλιοθήκες βαθιάς εκμάθησης στο παρελθόν, ένα έργο ανοιχτού κώδικα που επιτρέπει στους προγραμματιστές να ορίσουν, να αξιολογήσουν και να βελτιστοποιήσουν μαθηματικούς τύπους όπως πολυδιάστατες συστοιχίες και αναλυτικές συναρτήσεις.Αναπτύχθηκε από το Πανεπιστήμιο του Μόντρεαλ το 2007 και είναι η κύρια βασική βιβλιοθήκη που χρησιμοποιείται για βαθιά μάθηση στην python. Θεωρείται ο γεννήτορας των πλαισίων βαθιάς μάθησης, αλλά πλέον χρησιμοποιήται μόνο για ακαδημαϊκούς σκοπούς και έχει βγεί εκτός των προτιμίσεων των περισσότερων ερευνητών.

Pytorch

Το Pytorch [8] είναι ένα σχετικά νέο περιβάλλον βαθιάς μάθησης που βασίζεται στο Torch [9], μία βιβλιοθήκη μηχανικής μάθησης ανοικτού κώδικα. Αναπτύχθηκε από την ερευνητική ομάδα AI Research lab (FAIR) του Facebook, κυκλοφόρησε στο GitHub το 2017 και χρησιμοποιήθηκε σε εφαρμογές επεξεργασίας φυσικής γλώσσας. Το Pytorch έχει φήμη για την απλότητα, την ευκολία χρήσης, την ευελιξία, την αποτελεσματική χρήση της μνήμης και τα δυναμικά υπολογιστικά γραφήματα. Καθιστά την κωδικοποίηση πιο εύκολη και ταχύτερη [10].

2.4 Εφαρμογές συνελικτικών δικτύων

Τα συνελικτικά νευρωνικά δίκτυα ενισχύουν την αναγνώριση εικόνας και τις εργασίες όρασης υπολογιστή. Η όραση υπολογιστή είναι ένα πεδίο τεχνητής νοημοσύνης (AI) που επιτρέπει στους υπολογιστές και τα συστήματα να αντλούν σημαντικές πληροφορίες από ψηφιακές εικόνες, βίντεο και άλλες οπτικές εισόδους, και με βάση αυτές τις εισόδους, να μπορούν να αναλάβουν δράση. Μερικές εφαρμογές των δικτύων [2] είναι:

- **Μάρκετινγκ**: Οι πλατφόρμες κοινωνικών μέσων παρέχουν προτάσεις σχετικά με το ποιος μπορεί να είναι στη φωτογραφία που έχει αναρτηθεί σε ένα προφίλ, διευκολύνοντας την επισήμανση φίλων σε άλμπουμ φωτογραφιών.
- **Υγειονομική περίθαλψη**: Η όραση υπολογιστή έχει ενσωματωθεί στην ακτινολογική τεχνολογία, επιτρέποντας στους γιατρούς να εντοπίσουν καλύτερα τους καρκινικούς όγκους.
- **Λιανικό εμπόριο**: Η οπτική αναζήτηση έχει ενσωματωθεί σε ορισμένες πλατφόρμες ηλεκτρονικού εμπορίου, επιτρέποντας στις εταιρείες να προτείνουν αντικείμενα που θα συμπληρώνουν μια υπάρχουσα παραγγελία.
- **Αυτοκίνητο**: Ενώ η εποχή των αυτοκινήτων χωρίς οδηγό δεν έχει ακόμη εμφανιστεί, η βασική τεχνολογία έχει αρχίσει να εισέρχεται στα αυτοκίνητα, βελτιώνοντας την ασφάλεια των οδηγών και των επιβατών μέσω χαρακτηριστικών όπως η ανίχνευση γραμμής λωρίδας και ή ανάγνωση των σημάτων οδικής κυκλοφορίας [2].
- **Μέσα κοινονικής δικτύωσης**: Η αναγνώριση προσώπων έχει ενσωματωθεί στά μέσα κοινωνικής δικτύωσης και χρησιμοποιήται για την επισήμανση ατόμων που βρήσκονται σε μία φωτογραφία. Αυτή η λειτουργία είναι ιδιαίτερα χρήσιμη όταν πρέπει να επισημανθούν πολλά άτομα σε ένα άμπουμ από πολλές φωτογραφίες.
- **Κλιματικές αλλαγές**: Τα συνελικτικά νευρωνικά δίκτυα χρησιμοποιούνται για την καταπολέμηση της κλιματικής αλλαγής, παίζουν συμαντικό ρόλο στην κατανόηση των λόγων για τους οποίους βλέπουμε τέτοιες δραστικές αλλαγές στα κλιματικά φαινόμενα.

Κεφάλαιο 3

Τρόποι υλοποίησης και εκμετάλλευσης δικτύων βαθιάς μάθησης

Στόχος αυτού του κεφαλαίου είναι η περιγραφή μερικών ήδη σχεδιασμένων λογισμικών που έχουν τη δυνατότητα να εκμεταλλευτούν εκπαιδευμένα δίκτυα από τα περιβάλλοντα που περιγράφτηκαν στο κεφάλαιο 2.3 με σκοπό την μεταφορά και ενσωμάτωσή τους σε κάποιο άλλο διαφορετικό περιβάλλον.

3.1 Σκοπιμότητα μεταφοράς εκπαιδευμένων δικτύων σε άλλα περιβάλλοντα

Με την ανάπτυξη μοντέλων μηχανικής μάθησης και νευρωνικών δικτύων έρχεται η ανάγκη να αναπτυχθούν αυτά τα μοντέλα με κάποιο τρόπο σε άλλες γλώσσες πέρα από την python, έτσι ώστε να είναι συμβατά με άλλα υπάρχοντα συστήματα. Μέσω της μεταφοράς επιτυγχάνουμε την πλήρη εκμετάλλευση των διαφόρων βιβλιοθηκών βαθιάς μάθησης που χρησιμοποιούνται στην python από διαφορετικά περιβάλλοντα που βασίζονται σε άλλες γλώσσες όπως η C++.

Το συγκεκριμένο θέμα έχει απασχολήσει πολλούς μηχανικούς στη βιομηχανία και πολλές ομάδες ερευνητών. Στη συνέχεια θα δούμε κάποιες υλοποιήσεις που έχουν γίνει και είναι σημαντικό να παρουσιαστούν.

3.1.1 Σχετικές εργασίες

Στη πλατφόρμα του github έχουν αναρτηθεί αρκετές υλοποιήσεις, μία από αυτές ονομάζεται keras2cpp [11] και μπορεί να μεταφέρει εκπαιδευμένα δίκτυα από το περιβάλλον του keras - tensorflow στη C++. Το λογισμικό αυτό αποθηκεύει στο σκληρό δίσκο σε αρχεία, ένα μοντέλο, σε δυαδική μορφή για να διαβάζεται γρήγορα. Έπειτα, από τη μεριά της C++ έχει υλοποιηθεί συνάρτηση η οποία φορτώνει το μοντέλο στη μνήμη, σε συνεχόμενο μπλοκ για καλύτερη απόδοση. Το keras2cpp παρέχει μεγάλο πλήθος στρωμάτων και συναρτήσεων ενεργοποίησης. Η συγκεκριμένη υλοποίηση δεν χρησιμοποιεί εξωτερικές συναρτήσεις παρά μόνο την standard library. Το keras2cpp είναι συμβατό με C++17, όλες τις εκδόσεις του keras2 και όλες τις εκδόσεις της Python 3.

Υπάρχει επίσης μια υλοποίηση η οποία δείχνει να μοιάζει αρκετά με το keras2cpp, ωστόσο αποτελεί μια από τις πρώτες προσπάθειες που έγιναν για να μεταφερθεί ένα δίκτυο από το keras στη C++ και ονομάζεται Kerasify [12]. Πάνω στο Kerasify βασίστηκε μια λίγο διαφορετική εργασία η οποία λέγεται pocket-tensor [13]. Η συγκεκριμένη υλοποίηση καταφέρνει να μεταφέρει το δίκτυο σε μια ενσωματωμένη συσκευή. Τα κύρια χαρακτηριστικά της είναι η χαμηλή απαίτηση σε μνήμη και η δυνατότητα να χρησιμοποιεί παραπάνω από έναν πυρήνα του μικροεπεξεργαστή για τις προβλέψεις, αυτό είναι ιδιαίτερα σημαντικό όταν μιλάμε για ενσωματωμένες συσκευές γιατί είναι αρκετά περιορισμένες σε επεξεργαστικούς πόρους.

Μια παρόμοια βιβλιοθήκη που ονομάζεται frugally-deep [14], παρέχει περισσότερα στρώματα και συναρτήσεις ενεργοποίησης σε σχέση με τις προηγούμενες υλοποιήσεις, παρέχοντας αρκετά κοντινούς χρόνους εκτέλεσης με την python.

Στο πανεπιστήμιο του Πρίνστον στις Ηνωμένες Πολιτείες Αμερικής, μια ομάδα τεσσάρων ερευνητών δημιούργησε ένα λογισμικό το οποίο ονομάζεται Keras2c [15], με σκοπό την ανάπτυξη νευρωνικών δικτύων (τα οποία έχουν εκπαιδευτεί στο keras-tensorflow) στο σύστημα ελέγχου πλάσματος (PCS) του DIII-D tokamak στο National Fusion Facility που λειτουργεί στο General Atomics στο Σαν Ντιέγκο, όπου χρησιμοποιούνται νευρωνικά δίκτυα για να προβλέψουν την εξέλιξη της κατάστασης του πλάσματος και την εμφάνιση επικίνδυνων ασταθειών. Σύμφωνα με το άρθρο αναφέρουν πως το frugally-deep μπορεί να επιλύσει τέτοιου είδους προβλήματα αλλά το γεγονός ότι βασίζεται σε μεγάλο μέγεθος εξωτερικών βιβλιοθηκών όπως η Eigen για τον υποκείμενο υπολογισμό, έχει ως συνέπεια μη ντετερμινιστική συμπεριφορά και δεν είναι ασφαλής για χρήση σε πραγματικό χρόνο.

3.2 Προτεινόμενος τρόπος μεταφοράς δικτύου σε C++

Όπως θα δούμε στην συνέχεια, θα χρειαστεί να αναπτύξουμε μία διαδικασία η οποία θα έχει την ευθύνη αποθήκευσης σε αρχείο, της αρχιτεκτονικής, των βαρών και των κατωφλιών. Έπειτα θα χρειαστεί να αναπτυχθεί ένα λογισμικό το οποίο θα υλοποιεί τις συναρτήσεις που υπάρχουν για τα συνελικτικά νευρωνικά δίκτυα, μετά θα πρέπει να γίνεται λεκτική ανάλυση στο αρχείο της αρχιτεκτονικής και ανάλογα με αυτό που αναλύεται να καλεί την αντίστοιχη συνάρτηση και να χρησιμοποιεί τα αντίστοιχα βάρη και κατώφλια με σκοπό να δημιουργήσει τα στρώματα και ως αποτέλεσμα να έχουμε ένα πιστό αντίγραφο του δικτύου. Τα αριθμητικά δεδομένα των αρχείων θα πρέπει να είναι double, έτσι το δίκτυο θα μεταφέρεται με μεγαλύτερη ακρίβεια θυσιάζοντας βέβαια χρόνο κατά την εκτέλεση λόγω του μεγαλύτερου όγκου της πληροφορίας που θα πρέπει να επεξεργαστεί.

Με λίγα λόγια το λογισμικό θα εστιάζει στην ακρίβεια και στη δυνατότητα να μπορεί να μεταφέρει δίκτυα από πολλά περιβάλλοντα στη C++ κάτι το οποίο δεν παρατηρείται σε άλλες σχετικές εργασίες. Δεν μπορούμε να αποδείξουμε πως είναι η βέλτιστη μέθοδος, συνεπώς αναφερόμαστε σε ένα ανοιχτό πρόβλημα.

Κεφάλαιο 4

Υλοποίηση

4.1 Απαραίτητα εργαλεία

4.1.1 Περιβάλλον

Οι δύο υλοποιήσεις που ακολουθούν, αναπτύχθηκαν σε γλώσσα C++ και Python, σε λειτουργικό Linux και ειδικότερα, σε Ubuntu 20.04 LTS.

4.1.2 Hardware

Το hardware που χρησιμοποιήθηκε για την εκπαίδευση και τον υπολογισμό των πράξεων, των δικτύων που θα δούμε στη συνέχεια, ήταν ένα εικονικό σύστημα από την υπολογιστική υποδομή του Εργαστηρίου Αυτόνομων Συστημάτων (Autonomous Systems Laboratory - ASL) που βρίσκεται στο τμήμα Πληροφορικής και Τηλεπικοινωνιών στην Άρτα. Πιο συγκεκριμένα το σύστημα ήταν εξοπλισμένο με:

Hardware			
OS	CPU	RAM	STORAGE
Ubuntu 20.04 LTS	2 x Intel Xeon e5-2630	64GB ddr3	1.2TB SAS

Σχήμα 4.1: Ο υπολογιστής που χρησιμοποιήθηκε για τη λήψη των αποτελεσμάτων

4.2 Εκπαίδευση συνελικτικού δικτύου στο Pytorch

Στα παρακάτω αποσπάσματα κώδικα θα αναλύσουμε ένα παράδειγμα εκπαίδευσης συνελικτικού δικτύου με το pytorch.

Στο παρακάτω τμήμα παρουσιάζονται οι βιβλιοθήκες που απαιτούνται για χρήση των εργαλείων του pytorch (1-4) και της αποθήκευσης των δεδομένων του (5-7).

```
1  import torch
2  import torch.nn as nn
3  from torch.utils.data import DataLoader
4  from torchvision import datasets, transforms
5  import numpy as np
6  from PIL import Image
7  import script
```

Χρησιμοποιόντας τα βοηθητικά προγράμματα του pytorch (Dataloader, datasets, transforms) ενσωματώνουμε και μετασχηματίζουμε το σύνολο δεδομένων (CIFAR-10) [16]. Το σύνολο δεδομένων CIFAR-10 αποτελείται από 60000 έγχρωμες εικόνες 32x32 σε 10 κλάσεις, με 6000 εικόνες ανά κλάση. Το σύνολο των εικόνων χωρίζεται σε 50000 εικόνες εκπαίδευσης και 10000 δοκιμαστικές εικόνες.

```
transform = transforms.ToTensor()
train_data = datasets.CIFAR10( root = r"c:\path", train= True
, transform= transform)
test_data = datasets.CIFAR10( root = r"c:\path", train= False
, transform= transform)

train_loader = DataLoader(train_data, batch_size= 100,
shuffle=True, pin_memory=True, num_workers=4)
test_loader = DataLoader(test_data, batch_size = 100,
shuffle=True, pin_memory=True, num_workers=4)
```

Επιλογή του Hardware (cpu/gpu) που θέλουμε να εκτελέσει τον κώδικας.

```
device = torch.device('cpu')
```

Στο επόμενο τμήμα παρουσιάζεται η δομή των στρωμάτων, του δικτύου που χρησιμοποιήσαμε για να επιλύσουμε το πρόβλημα ταξινόμησης του συνόλου δεδομένων. Όπως είπαμε και στο θεωρητικό υπόβαθρο πρέπει να ορίσουμε τις υπερπαραμέτρους που θα επηρεάζουν την έξοδο από κάθε στρώμα.
Το αστεράκι στις παραμέτρους σημαίνει ότι έχουν προεπιλεγμένη τιμή σε περίπτωση που δεν δηλωθούν.

- nn.Conv2d(1,2,3,4*,5*):
    1. Ορίζουμε τα κανάλια εισόδου δηλαδή σε μια έγχρωμη εικόνα όπως στο παράδειγμα θα έχουμε 3 κανάλια.
    2. Ορίζουμε τον αριθμό των φίλτρων.
    3. Ορίζουμε το μέγεθος των φίλτρων.
    4. Ορίζουμε το stride που προεπιλεγμένα είναι 1.
    5. Ορίζουμε το padding που προεπιλεγμένα είναι 0.
- nn.MaxPool2d(1,2*,3*):
    1. Ορίζουμε το μέγεθος του φίλτρου.
    2. Ορίζουμε το stride που προεπιλεγμένα είναι None.
    3. Ορίζουμε το padding που προεπιλεγμένα είναι 0.
- nn.Linear(1,2):
    1. Ορίζουμε το μέγεθος κάθε δείγματος εισόδου.
    2. Ορίζουμε το μέγεθος κάθε δείγματος εξόδου.

Στο απόσπασμα που ακολουθεί περιγράφεται ο κώδικας, για τη δημιουργία του μοντέλου.

```
class convCIFAR(nn.Module):
def __init__ (self):
  super().__init__()
  self.net=nn.Sequential(
    nn.Conv2d(3,5,5,1,0).double(),
    nn.ReLU().double(),
    nn.MaxPool2d(2).double(),
    nn.Conv2d(5,20,3).double(),
    nn.ReLU().double(),
    nn.MaxPool2d(2).double(),
    nn.Flatten().double(),
    nn.Linear(720,120).double(),
    nn.ReLU().double(),
    nn.Linear(120,80).double(),
    nn.ReLU().double(),
    nn.Linear(80,10).double(),
    nn.Softmax(1).double())
def forward(self,x):
return(self.net(x))
```

Στο απόσπασμα που ακολουθεί περιγράφεται ο κώδικας pytorch που εκτελεί την εκπαίδευση του μοντέλου, όπου γίνονται διαδοχικές επαναλήψεις που ανατροφοδοτούν τις εισόδους του δίκτυο με σκοπό τη βελτιστοποίηση του μοντέλου.

```
model = convCIFAR().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 0.001)

epochs=5
train_losses=[]
train_correct=[]
for i in range(epochs):
trn_corr = 0
tst_corr = 0
for b, (X_train, y_train) in enumerate(train_loader):
X_train = X_train.to(device)
y_train = y_train.to(device)
b+=1
y_pred = model(X_train.double()).to(device)
loss = criterion(y_pred, y_train)

predicted = torch.max(y_pred.data, 1)[1]
batch_corr = (predicted == y_train).sum()
trn_corr += batch_corr

optimizer.zero_grad()
loss.backward()
optimizer.step()

if b%100 == 0:
   print(f'epoch: {i} loss:
   {loss.item()} accuracy: {trn_corr.item()*100/(100*b)}%')

train_losses.append(loss)
train_correct.append(trn_corr)
```

Το numpy [17] μπορεί να χρησιμοποιηθεί για την εκτέλεση μιας μεγάλης ποικιλίας μαθηματικών πράξεων σε πίνακες. Το PIL [18] χρησιμοποιείται για την αναπαράσταση μιας εικόνας. Η ενότητα Image που κάνουμε χρήση στο συγκεκριμένο παράδειγμα παρέχει μια σειρά λειτουργιών για τη φόρτωση εικόνων από αρχεία και τη δημιουργία νέων εικόνων.Με την χρήση τους εισάγουμε και αποθηκεύουμε στον σκληρό δίσκο, μια εικόνα γάτας η οποία θα χρησιμοποιηθεί για τα πειράματα.

```
im = Image.open("cat.jpeg").convert('RGB')
im = im.resize((32,32))
im = np.array(im)
im = im.astype('float64')
im = torch.from_numpy(im.reshape((1, 3, 32, 32)))
im = im.double()
np.savetxt("cat.txt",im.reshape(im.shape[0],-1))
```

Αφού εκπαιδεύσουμε το μοντέλο, κάνουμε μία πρόβλεψη, δίνοντας στο δίκτυο μια εικόνα. Έπειτα αποθηκεύουμε τα αποτελέσματα σε αρχείο, για να μπορέσουμε στη συνέχεια να τα συγκρίνουμε με τα αποτελέσματα που θα πάρουμε απο τις δύο υλοποιήσεις. Τέλος αποθηκεύουμε το μοντέλο του δικτύου κάνοντας χρήση της συνάρτησης savetxt η οποία είναι το σενάριο κώδικα που θα αναλύσουμε στην επόμενη ενότητα.

```
prediction = model(im).to(device)
np.savetxt("mypred.txt",prediction.detach().numpy())
script.savetxt(model,"mynet")
```

4.3 Εξόρυξη της αρχιτεκτονικής και των παραμέτρων ενός δικτύου από το pytorch

Στο τέλος της προηγούμενης ενότητας, κάναμε χρήση της συνάρτησης script.savetxt. Η συνάρτηση αυτή δέχεται δύο ορίσματα, το δίκτυο από το pytorch και το όνομα που θέλουμε να έχει ο φάκελος αποθήκευσης. Η συνάρτηση αυτή, αρχικά, εξάγει τις παραμέτρους των βαρών και τών κατωφλιών οι οποίες αποθηκεύονται σε αρχεία που έχουν για όνομα αύξουσα αριθμητική σειρά, δηλαδή για το πρώτο στρώμα τα βάρη θα αποθηκευτούν στο αρχείο 0.txt και τα κατώφλια στο αρχείο 1.txt, για το δεύτερο στρώμα τα βάρη στο 2.txt, τα κατώφλια στο 3.txt και ούτω καθεξής. Τέλος, δημιουργεί ένα αρχείο arc.txt στο οποίο αποθηκεύει την αρχιτεκτονική του δικτύου.

```
import numpy as np
import os
import shutil
def savetxt(net,name):
    if os.path.exists(name):
        shutil.rmtree(name)
    os.mkdir(name)
    i=0
    for param_tensor in net.state_dict():
        np.savetxt(name+"/"+str(i)+".txt",net.state_dict()
        [param_tensor].cpu().detach().clone().numpy().reshape
        (net.state_dict()[param_tensor].shape[0],-1).copy())
        i+=1
    with open(name+"/arc.txt", 'w') as filehandle:
        filehandle.write(str(net))
```

4.4 Πρώτη υλοποίηση

Σε αυτή την προσπάθεια υλοποίησης, αναπτύχθηκε ένα πρόγραμμα σε python το οποίο κάνει χρήση μόνο της βιβλιοθήκης numpy. Η εφαρμογή χρησιμοποιεί μόνο τα αρχεία με τις παραμέτρους του δικτύου και η αρχιτεκτονική ορίζεται από τον χρήστη κατά την χρήση των συναρτήσεων που υλοποιούν τα στρώματα του δικτύου. Έγινε κατασκευή συναρτήσεων οι οποίες λειτουργούν όπως οι συναρτήσεις του pytorch. Όλες οι συναρτήσεις επιστρέφουν έναν πίνακα εικόνας, ο οποίος έπειτα μπορεί να χρησιμοποιηθεί ως εικόνα εισόδου για κάποια άλλη συνάρτηση. Παρακάτω αναφέρονται οι συναρτήσεις που υλοποιήθηκαν.

Το αστεράκι στις παραμέτρους σημαίνει ότι έχουν προεπιλεγμένη τιμή σε περίπτωση που δεν δηλωθούν.

- convol2D(1,2,3,4*,5*,6*) : Η συνάρτηση αυτή είναι αντίστοιχη με την nn.Conv2d.
    1. Ορίζουμε τον πίνακα εικόνας εισόδου.
    2. Ορίζουμε τον πίνακα με τις παραμέτρους των βαρών.
    3. Ορίζουμε τον πίνακα με τις παραμέτρους των κατωφλιών.
    4. Ορίζουμε το padding που προεπιλεγμένα είναι 0.
    5. Ορίζουμε το stride που προεπιλεγμένα είναι 1.
    6. Ορίζουμε τη συνάρτιση ενεργοποίησης που προεπιλεγμένα είναι relu.

- mpl2D(1,2*,3*) : Η συνάρτηση αυτή είναι αντίστοιχη με την nn.MaxPool2d.
    1. Ορίζουμε τον πίνακα εικόνας εισόδου.
    2. Ορίζουμε το μέγεθος του φίλτρου.
    3. Ορίζουμε το stride που προεπιλεγμένα είναι 1.

- apl2D(1,2*,3*) : Η συνάρτηση αυτή είναι αντίστοιχη με την nn.AvgPool2d.
    1. Ορίζουμε τον πίνακα εικόνας εισόδου.
    2. Ορίζουμε το μέγεθος του φίλτρου.
    3. Ορίζουμε το stride που προεπιλεγμένα είναι 1.

- fcl(1,2,3,4) : Η συνάρτηση αυτή είναι αντίστοιχη με την nn.Linear.
    1. Ορίζουμε τον πίνακα εικόνας εισόδου.
    2. Ορίζουμε τον πίνακα με τις παραμέτρους των βαρών.
    3. Ορίζουμε τον πίνακα με τις παραμέτρους των κατωφλιών.
    4. Ορίζουμε τη συνάρτιση ενεργοποίησης που προεπιλεγμένα είναι relu.

- relux(1) : Η συνάρτηση αυτή είναι αντίστοιχη με την nn.ReLU.
    1. Ορίζουμε τον πίνακα εικόνας εισόδου.

- flat(1) : Η συνάρτηση αυτή είναι αντίστοιχη με την nn.Flatten.
    1. Ορίζουμε τον πίνακα εικόνας εισόδου.

- sm(1) : Η συνάρτηση αυτή είναι αντίστοιχη με την nn.Softmax.
    1. Ορίζουμε τον πίνακα εικόνας εισόδου.

- lsm(1) : Η συνάρτηση αυτή είναι αντίστοιχη με την nn.LogSoftmax.
    1. Ορίζουμε τον πίνακα εικόνας εισόδου.

Αφού είδαμε τις συναρτήσεις που κατασκευάστηκαν, τώρα θα δούμε και πως χρησιμοποιούνται. Τα αποσπάσματα κώδικα που θα παρουσιαστούν αφορούν μόνο το παράδειγμα κώδικα που αναλύθηκε στο κεφάλαιο 4.2 .

Αρχικά, εισάγονται τα αρχεία με τις παραμέτρους των βαρών και των κατωφλιών σε πίνακες. Για τα convolution layers γίνεται ένας μετασχηματισμός των πινάκων με τις παραμέτρους των βαρών, έτσι ώστε να ταιριάζουν με την συνάρτηση convol2D.

```
layer1 = np.loadtxt("mynet/0.txt")
layer1 = layer1.reshape(5,3,5,5)
layer1b = np.loadtxt("mynet/1.txt")

layer2 = np.loadtxt("mynet/2.txt")
layer2 = layer2.reshape(20,5,3,3)
layer2b = np.loadtxt("mynet/3.txt")

layer3 = np.loadtxt("mynet/4.txt")
layer3b = np.loadtxt("mynet/5.txt")

layer4 = np.loadtxt("mynet/6.txt")
layer4b = np.loadtxt("mynet/7.txt")

layer5 = np.loadtxt("mynet/8.txt")
layer5b = np.loadtxt("mynet/9.txt")
```

Κατόπιν εισάγουμε την ίδια εικόνα που χρησιμοποιήσαμε στην ενότητα 4.2 . Αντίστοιχα καλούμε τις συναρτήσεις με την ίδια σειρά και τα ίδια ορίσματα όπως στο pytorch. Τον πίνακα που θα επιστρέφει η κάθε συνάρτηση θα τον εισάγουμε στη μεταβλητή output, την οποία θα ξαναχρησιμοποιούμε ως εικόνα εισόδου της επόμενης συνάρτησης εώς ότου φτάσουμε στο τελευταίο στρώμα. Τέλος, αποθηκεύουμε τον πίνακα που παρήγαγε η τελευταία συνάρτηση, που είναι στην ουσία η έξοδος του δικτύου, έτσι ώστε να γίνει σύγκριση με τα αποτελέσματα που κρατήσαμε απο την ενότητα 4.2 .

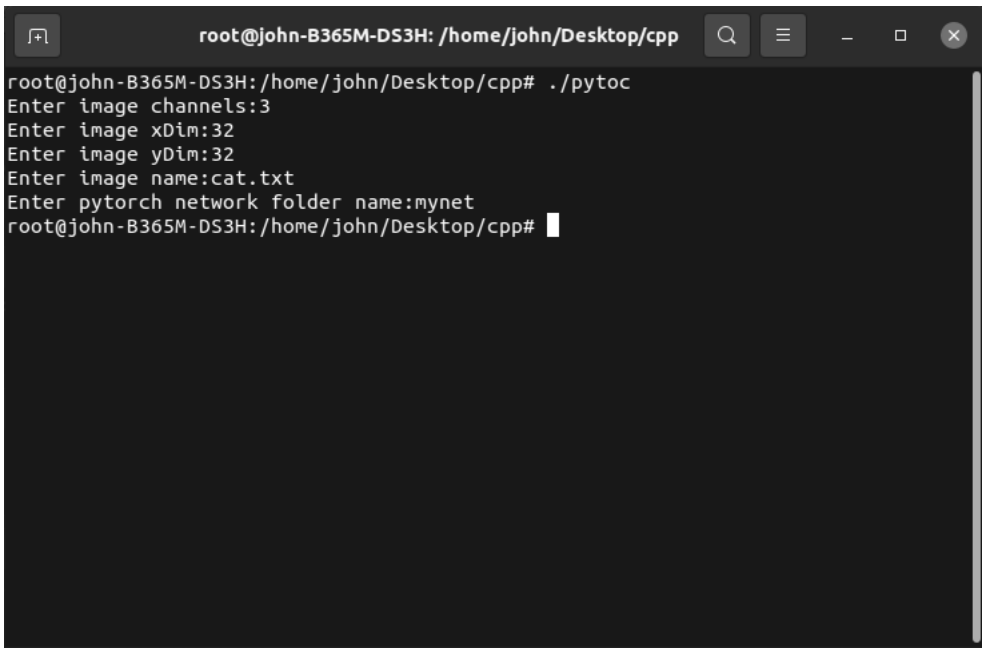
```
im = Image.open("cat.jpeg").convert('RGB')
im = im.resize((32,32))
im = np.array(im)
im = im.astype('float64')
im = np.reshape(im,(3,32,32))

output = convol2D(im, layer1,layer1b,strides=1,padding=0)

output = mpl2D(output,strides=2)

output = convol2D(output, layer2,layer2b,strides=1,padding=0)

output = mpl2D(output,strides=2)

output = flat(output)

output = fcl(output,layer3,layer3b,activation='relu')

output = fcl(output,layer4,layer4b,activation='relu')

output = fcl(output,layer5,layer5b,activation='logsoftmax')

np.savetxt("mypred2.txt",output)
```

4.5 Δεύτερη υλοποίηση

Η συγκεκριμένη υλοποίηση έγινε με σκοπό να αποφύγουμε την python. Από την πρώτη υλοποίηση αποκτήσαμε την τεχνογνωσία πάνω στην κατασκευή συναρτήσεων για συνελικτικά νευρωνικά δίκτυα. Έτσι, πήραμε την απόφαση και υλοποιήσαμε το ίδιο λογισμικό σε γλώσσα προγραμματισμού C++. Σε σχέση με την προηγούμενη, αυτή η υλοποίηση εκμεταλεύεται επίσης το αρχείο της αρχιτεκτονικής για να αναγνωρίσει τα επίπεδα του δικτύου με αποτέλεσμα την αυτόματη δημιουργία του δικτύου. Χρησιμοποιήθηκαν οι βιβλιοθήκες regex και fstream για να γραφτεί κώδικας, ο οποίος αναλύει κάθε γραμμή του αρχείου της αρχιτεκτονικής με κανονικές εκφράσεις βρίσκοντας έτσι ποιά συνάρτηση αντιστοιχεί και ποιές τιμές πρέπει να έχουν οι υπερπαραμέτροι της. Επίσης έγινε χρήση της βασικής βιβλιοθήκης iostream και της βιβλιοθήκης cmath για εκθετικές πράξεις και υπολογισμό λογαρίθμων στις συναρτήσεις ενεργοποίησης LogSoftmax και Softmax.

Οι συναρτήσεις λειτουργούν λίγο διαφορετικά σε σχέση με την πρώτη υλοποίηση.
Η διαφορά βρίσκεται στο όρισμα της εικόνας εισόδου, όπου οι συναρτήσεις παίρνουν έναν δείκτη σε πίνακα και επεξεργάζονται τον πίνακα χωρίς να επιστρέφουν κάτι. Για τις συναρτήσεις πριν τα πλήρως συνδεδεμένα στρώματα χρησιμοποιήθηκαν τριπλοί δείκτες σε πίνακα. Γιάυτό το λόγο έγινε υπερφόρτωση της συνάρτησης relux. Από την υπερφόρτωση αυτή προέκυψαν δύο εκδοχές, η μία δέχεται σκέτο δείκτη σε πίνακα ενώ η άλλη τριπλό δείκτη. Κατασκευάστηκε επίσης μία συνάρτηση για την εισαγωγή της πρώτης εικόνας εισόδου που θα χρησιμοποιεί το δίκτυο. Ονομάζεται loadim και έχει σαν ορίσματα το όνομα της εικόνας και τον τριπλό δείκτη όπου θα μεταφορτωθεί.

Αφού παρουσιάστηκαν οι δυνατότητες που παρέχει αυτή η υλοποίηση, ήρθε η ώρα να δούμε πώς χρησιμοποιείται. Αρχικά ορίζουμε τα κανάλια και τις διαστάσεις της εικόνας, έπειτα το αρχείο της εικόνας και τέλος το όνομα του φακέλου που περιέχει τα αρχεία των παραμέτρων και της αρχιτεκτονική. Το λογισμικό θα περάσει την εικόνα μέσα από το δίκτυο και θα αποθηκεύσει τα αποτελέσματα σε ένα αρχείο.



Σχήμα 4.2: Χρήση εφαρμογής

Κεφάλαιο 5

Συμπεράσματα

5.1 Αποτελέσματα

Έχοντας συγκεντρώσει τα αποτελέσματα από κάθε υλοποίηση και από το παράδειγμα του pytorch στα σχήματα 5.1, 5.2, 5.3, 5.4 μπορούμε να πούμε πώς τα αποτελέσματα είναι ίδια. Παρατηρούμε πως όταν κάνουμε χρήση μεταβλητών τύπου float προκύπτουν διαφορές μεγέθους μικρότερες από 10^{-6}, ενώ όταν χρησιμοποιούμε double έχουμε διαφορές μικρότερες από 10^{-15}. Αυτές οι διαφορές προκύπτουν γιατί σύμφωνα με το [19], κάθε τύπος μεταβλητής έχει ένα όριο ψηφίων που μπορεί να υπάρχει ακρίβεια.

Συμπεραίνουμε λοιπόν, πως το λογισμικό όσον αφορά στο θέμα της ακρίβειας των αποτελεσμάτων, έχει φτάσει τα όρια του κάθε τύπου μεταβλητής.

Βλέποντας τους χρόνους εκτέλεσης στο σχήμα 5.5, τα αποτελέσματα είναι τα αναμενόμενα, δεδομένου ότι η C++ θεωρείται μία από τις γρηγορότερες γλώσσες στο χρόνο εκτέλεσης εντολών.

	Pytorch	**Python**	**C++**
plane	-5.412998657226562500e+2	-5.412999074985007155e+2	-5.412999074985007053e+2
car	-0.714776916503906250e+2	-0.714777198781957850e+2	-0.714777198781957350e+2
bird	-8.140858154296875000e+2	-8.140858831598329743e+2	-8.140858831598328286e+2
cat	-5.722955932617187500e+2	-5.722956749028778631e+2	-5.722956749028777304e+2
deer	-7.866286621093750000e+2	-7.866286596207249886e+2	-7.866286596207250131e+2
dog	-7.111199340820312500e+2	-7.111200352374510203e+2	-7.111200352374508306e+2
frog	-4.948321533203125000e+2	-4.948322020736276272e+2	-4.948322020736275172e+2
horse	-4.860205688476562500e+2	-4.860205976423773109e+2	-4.860205976423773329e+2
ship	-5.710444335937500000e+2	-5.710445125904906263e+2	-5.710445125904906213e+2
truck	0	0	0

Σχήμα 5.1: Τα αποτελέσματα από καθε υλοποίηση για μεταβλητές τύπου float.

	Python - C++	**Pytorch - Python**	**Pytorch - C++**
plane	0.277265e-15	1.1355851887910000e-6	1.135585188513736e-6
car	13.50714e-15	7.6731129823683289e-6	7.673112968861187e-6
bird	3.960537e-15	1.8410962368167730e-6	1.841096232856237e-6
cat	3.607160e-15	2.2192367927147330e-6	2.219236789107573e-6
deer	0.665979e-15	0.0676485210338300e-6	0.067648520367851e-6
dog	5.156581e-15	2.7496893941175330e-6	2.749689388960953e-6
frog	2.990110e-15	1.3252525058740520e-6	1.325252502883942e-6
horse	0.598022e-15	0.7827216701539140e-6	0.782721670751936e-6
ship	0.135914e-15	2.1473540455196371e-6	2.147354045383723e-6
truck	0	0	0

Σχήμα 5.2: Διαφορές αποτελεσμάτων float

	Pytorch	**Python**	**C++**
plane	0	0	0
car	-5.774618605914413365e+2	-5.774618605914409954e+2	-5.774618605914411743e+2
bird	-3.330680357287754418e+2	-3.330680357287754987e+2	-3.330680357287754814e+2
cat	-8.684758584257143639e+2	-8.684758584257140228e+2	-8.684758584257140981e+2
deer	-7.463746075390824899e+2	-7.463746075390826036e+2	-7.463746075390828123e+2
dog	-9.260200197223632586e+2	-9.260200197223634689e+2	-9.260200197223631683e+2
frog	-13.464214413828629738e+2	-13.464214413828629511e+2	-13.464214413828629654e+2
horse	-9.992753027018256944e+2	-9.992753027018258081e+2	-9.927530270182536725e+2
ship	-1.229615062310435292e+2	-1.229615062310431881e+2	-1.229615062310432665e+2
truck	-10.55922490527958416e+2	-10.55922490527958062e+2	-10.55922490527958351e+2

Σχήμα 5.3: Τα αποτελέσματα από καθε υλοποίηση για μεταβλητές τύπου double.

	Python - C++	**Pytorch - Python**	**Pytorch - C++**
plane	0	0	0
car	4.863006e-15	9.272059e-15	4.409053e-15
bird	0.470262e-15	1.546702e-15	1.076440e-15
cat	2.046866e-15	9.272059e-15	7.225193e-15
deer	5.673055e-15	3.090686e-15	8.763741e-15
dog	3.261939e-15	5.716547e-15	2.454608e-15
frog	3.887142e-15	6.170499e-15	2.283357e-15
horse	5.803532e-15	3.090686e-15	8.894218e-15
ship	2.131133e-15	9.272059e-15	7.140926e-15
truck	7.855835e-15	9.622718e-15	1.766883e-15

Σχήμα 5.4: Διαφορές αποτελεσμάτων double

	Python	**C++**
Milliseconds	287	45

Σχήμα 5.5: Χρόνος εκτέλεσης

5.2 Ανοιχτά προβλήματα

Αρχικά από το pytorch, πρέπει να βρούμε τρόπο να εξάγουμε και αναδρομικά μοντέλα. Επίσης, όπως παρατηρήθηκε και σε άλλες σχετικές εργασίες, οι παραμέτροι των βαρών και των κατωφλίων, θα πρέπει να αποθηκεύονται σε δυαδική μορφή και ύστερα όταν θα μεταφέρονται στη C++ να βρίσκονται σε ένα συνεχόμενο μπλόκ μνήμης. Σαν αποτέλεσμα θα μειωθεί σε μεγάλο βαθμό ο χρόνος εκτέλεσης.

Ο κώδικας του λογισμικού, ενώ έχει έναν στοιβαρό σχεδιασμό, έχει σημεία όπου ο τρόπος χρήσης των δεικτών θα μπορούσε να γίνει πιο αποδοτικά.

Ιδιαίτερα καλό θα ήταν να φτιάχναμε μια βιβλιοθήκη συναρτήσεων, όπου σε κάθε συνάρτηση θα αντιστοιχεί ένα αρχείο είτε σε μορφή κώδικα C++ είτε σε δυαδική μορφή (binary code). Με τον τρόπο αυτό δημιουργείται μια βιβλιοθήκη συναρτήσεων έτοιμη είτε για include σε επίπεδο πηγαίου κώδικα, είτε για σύνδεση (link) με τον τελικό κώδικα C++. Έτσι, θα μπορούσαμε πιο εύκολα να τις διαχειριστούμε και να τις συντηρήσουμε.

Θεωρώ επίσης σημαντικό, την ενσωμάτωση κώδικα συναρτήσεων, για περισσότερα στρώματα και συναρτήσεις ενεργοποίησης, έτσι ώστε να καλύπτονται περισσότερα μοντέλα.

Το επόμενο μεγάλο βήμα θα είναι, να μπορεί να συνεχίζεται η εκπαίδευση του μοντέλου στην C++ από το σημείο που σταμάτησε να εκπαιδεύεται στο προηγούμενο περιβάλλον.

Τέλος, σε συνδιασμό με όλα τα προηγούμενα, αν μπορούσαμε να εκμεταλευτούμε και την επεξεργαστική ισχύ που παρέχουν οι κάρτες γραφικών θα είχαμε ένα ταχύτατο και επεκτάσιμο λογισμικό για την επίλυση προβλημάτων βαθειάς μάθησης και όχι μονο.

Βιβλιογραφία

[1] Y. Gavrilova, "A guide to deep learning and neural networks." [Online]. Available: https://serokell.io/blog/deep-learning-and-neural-network-guide

[2] I. C. Education, "Convolutional neural networks." [Online]. Available: https://www.ibm.com/cloud/learn/convolutional-neural-networks

[3] D. Liu, "A practical guide to relu." [Online]. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

[4] B. Basnet, "Log softmax vs softmax." [Online]. Available: https://deepdatascience.wordpress.com/2020/02/27/log-softmax-vs-softmax

[5] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[7] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: http://arxiv.org/abs/1605.02688

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[9] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[10] J. Terra, “Keras vs tensorflow vs pytorch: Understanding the most popular deep learning frameworks,” 2021. [Online]. Available: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article

[11] G. Perevozchikov, “Keras2cpp,” 2019. [Online]. Available: https://github.com/gosha20777/keras2cpp

[12] R. Rose, “Kerasify,” 2016. [Online]. Available: https://github.com/moof2k/kerasify

[13] G. Valiente, “Pocket-tesnor,” 2018. [Online]. Available: https://github.com/GValiente/pocket-tensor

[14] T. Hermann, “Frugally-deep,” 2016. [Online]. Available: https://github.com/Dobiasd/frugally-deep

[15] R. Conlin, K. Erickson, J. Abbate, and E. Kolemen, “Keras2c: A library for converting keras neural networks to real-time compatible c,” *Engineering Applications of Artificial Intelligence*, vol. 100, p. 104182, 2021.

[16] A. Krizhevsky, V. Nair, and G. Hinton, “Cifar-10 (canadian institute for advanced research).” [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[17] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[18] A. Clark, “Pillow (pil fork) documentation,” 2015. [Online]. Available: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf

[19] C. Robertson, K. Sharkey, M. Jones, G. Hogenson, S. Cai, and N. Turn, “Type float,” 2016. [Online]. Available: https://docs.microsoft.com/en-us/cpp/c-language/type-float?view=msvc-160