Τίτλος διπλωματικής: "Απεικόνιση αλγορίθμων Βαθιάς Μάθησης στην πλατφόρμα Graphcore IPU"

Επιλεγμένοι αλγόριθμοι: **HOG,ERT**

Μηχάνημα που θα χρησιμοποιηθεί για την υλοποίηση: **IPU** της Graphcore

Αρχιτεκτονική του μηχανήματος Intelligence Processing Unit (IPU)

Το IPU χρησιμοποιείται ως επιταχυντής για έναν κεντρικό υπολογιστή. Ο κεντρικός επεξεργαστής μπορεί να δημιουργήσει κώδικα IPU για εκτέλεση σε μία ή περισσότερες IPU. Ο κεντρικός υπολογιστής επικοινωνεί με τις IPU μέσω της διεπαφής PCI Express (PCIe). Αυτό επιτρέπει στον κεντρικό υπολογιστή να εκφορτώνει εργασίες υπολογισμού στο IPU και το IPU να μεταφέρει δεδομένα από και προς τη μνήμη του κεντρικού υπολογιστή. Πολλά IPU μπορούν να χρησιμοποιηθούν μαζί σε μία μόνο εργασία.Σε αυτήν την περίπτωση επικοινωνούν μέσω προσαρμοσμένων καλωδίων διασύνδεσης IPU-Link.

Το μηχάνημα IPU έχει πολλά ανεξάρτητα processing units τα οποία ονομάζονται **Tiles**. Τα tiles αυτα συνδέονται μεταξύ τους με ένα σύστημα ιστού, πολύ γρήγορο all-to-all τύπου επικοινωνίας, που ονομάζεται exchange. Στην περίπτωση που χρησιμοποιούνται περισσότερα του ενός IPU τότε αυτο το σύστημα επεκτείνεται σε όλα τα tiles όλων των IPU's.

Το κάθε tile αποτελείται απο έναν επεξεργαστή και την τοπική του μνήμη. Ουσιαστικά πρόκειται για έναν ολοκληρωμένο επεξεργαστή που μπορεί να εκτελέσει ανεξάρτητα προγράμματα με αυθαίρετη ροή ελέγχου.

Ο επεξεργαστής υποστηρίζει έναν καθορισμένο αριθμό νημάτων υλικού, τα οποία εξυπηρετούνται σε σειρά round-robin.Ο κώδικας μπορεί να εκτελεστεί με δύο τρόπους: supervisor or worker.. Ο κώδικας του supervisor ελέγχει την εκτέλεση εργασιών των worker, οι οποίες εκτελούν υπολογισμό κυμαινόμενου σημείου.

Κάθε tile έχει την δική του τοπική μνήμη SRAM. Αυτή η τοπική μνήμη είναι και η μόνη που είναι άμεσα πρόσβασιμη από τα tile instructions. Τα δεδομένα και ο κώδικας του εκάστοτε tile μοιράζονται αυτήν την μνήμη. Επίσης δεν υπάρχει πρόσβαση σε διαμοιραζόμενη μνήμη (shared-memory) μεταξύ διαφορετικών tiles.

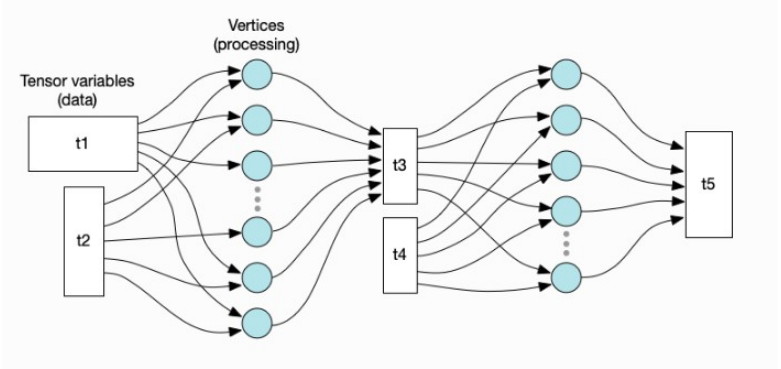
Κάθε IPU έχει 1216 tiles και το κάθε tile έχει 256 kilobytes SRAM. Αρα συνολικά έχουμε περίπου 300MB μνήμη στο μηχάνημα.

Προγραμματιστικο Μοντέλο στο μηχάνημα IPU

Όταν το IPU εκτελεί ένα πρόγραμμα, όλα τα tiles συνεργάζονται παράλληλα στην εκάστοτε εργασία. Κάθε tile μπορεί να εκτελέσει διαφορετικό κώδικα και λειτουργεί σε δεδομένα που είναι αποθηκευμένα στην τοπική του μνήμη. Τα tiles μπορούν να ανταλλάσσουν δεδομένα στο τέλος κάθε εργασίας υπολογισμού και πρέπει να συγχρονιστούν για να επιτευχθεί αυτή η επικοινωνία.

Όσο αναφορά τα δεδομένα αποθηκεύονται σε multi-dimensional arrays που ονομάζονται tensors. Ένα πρόγραμμα που τρέχει σε IPU μπορει να διαβάσει/γράψει απο και σε πολλά διαφορετικα tensors. Τα tiles μπορούν να επεξεργαστούν τα tensors αυτά (πολλά tiles επεξεργάζονται ενα η πολλά διαφορετικά tensors) και κάθε επεξεργασία δεδομένων αποθηκέυεται στην τοπική μνήμη του κάθε tile.

Ο συνολικός υπολογισμός του εκάστοτε προγράμματος μας μπορεί να αναπαρασταθεί ως ένας **γράφος (Graph)**. Οι **κορυφές (vertex)** του γράφου αντιπροσωπεύουν τον κώδικα που εκτελείται από ένα tile. Οι **ακμές (edges)** αναπαριστούν τα δεδομένα στα οποία εκτελείται ο κώδικας του vertex.Εάν τα δεδομένα βρίσκονται στη μνήμη του tile που εκτελεί τον κώδικα του vertex, τότε αυτές οι άκρες αντιπροσωπεύουν τις αναγνώσεις και τις εγγραφές στην τοπική μνήμη. Εάν υπάρχουν μεταβλητές αποθηκευμένες σε άλλο tile, τότε οι άκρες αντιπροσωπεύουν επικοινωνία μέσω του ιστού ανταλλαγής(exchange fabric).

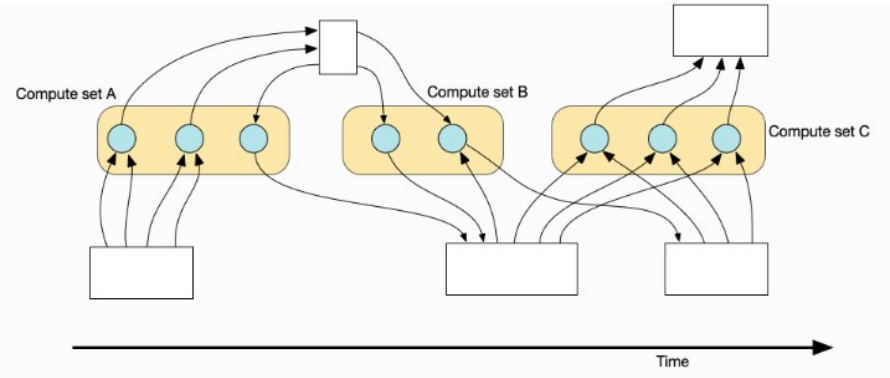


Η λειτουργία που εκτελείται από μια κορυφή μπορεί να είναι οτιδήποτε, από μια απλή αριθμητική λειτουργία έως την αναδιαμόρφωση / μεταφορά δεδομένων του tensor ή την εκτέλεση μιας N-διαστατικής συνέλιξης.

Παράλληλη εκτέλεση κώδικα

Ο γράφος μπορεί να διανεμηθεί σε όλη την IPU έτσι ώστε τα vertices να εκτελούνται παράλληλα. Κάθε tile εκτελεί ένα ή περισσότερα vertices, λειτουργώντας σε δεδομένα που είναι αποθηκευμένα τοπικά και στη συνέχεια μεταδίδει τα αποτελέσματα σε άλλα tiles.

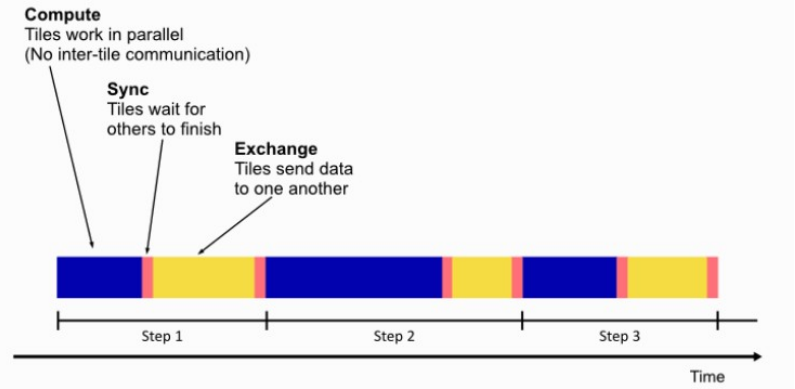
Το σύνολο κορυφών που εκτελούν παράλληλα είναι γνωστό ως **σύνολο υπολογισμών(compute set)**. Η σειρά με την οποία εκτελούνται τα βήματα καθορίζεται από ένα πρόγραμμα ελέγχου που φορτώνεται σε κάθε tile.



Η IPU χρησιμοποιεί το μοντέλο εκτέλεσης bulk-synchronous parallel (BSP), όπου η εκτέλεση μιας εργασίας χωρίζεται σε βήματα. Κάθε βήμα αποτελείται από τις ακόλουθες φάσεις:

->τοπικός υπολογισμός (local compute)
->παγκόσμιος συγχρονισμός (global synchronisation)
->ανταλλαγή δεδομένων (data exchange)

Στη φάση υπολογισμού, όλα τα tiles εκτελούνται παράλληλα, λειτουργώντας στα τοπικά τους δεδομένα. Αφού ολοκληρωθεί η εκτέλεση κάθε tile, μπαίνει στην κατάσταση συγχρονισμού. Όταν όλα τα tiles έχουν φτάσει στην κατάσταση συγχρονισμού, το IPU εισέρχεται στη φάση ανταλλαγής όπου αντιγράφονται δεδομένα μεταξύ των tiles.



Μετά τη φάση ανταλλαγής, η διαδικασία επαναλαμβάνεται: τα tiles μετακινούνται σε μια νέα φάση υπολογισμού, εκτελώντας υπολογισμούς χρησιμοποιώντας τα τοπικά τους δεδομένα και τα νέα δεδομένα που λαμβάνονται κατά τη διάρκεια της ανταλλαγής. Το πρόγραμμα συνεχίζεται εκτελώντας μια σειρά από τέτοια βήματα, εναλλάσσοντας μεταξύ φάσεων ανταλλαγής και υπολογισμού.

Για να προσδιοριστεί η σειρά των βημάτων που θα εκτελεστούν, κάθε tile περιέχει ένα **πρόγραμμα ελέγχου (control program)**. Αυτό φορτώνεται σε κάθε tile, από τον κεντρικό υπολογιστή, για τον έλεγχο της εκτέλεσης των ακολουθιών υπολογισμού και ανταλλαγής.

Διαθέσιμο Μηχάνημα

Εμείς έχουμε στην διάθεση μας 16 μηχανηματα IPU όπου το κάθε ένα απο αυτά έχει 1216 tiles

Σύντομη Περιγραφή Βημάτων των Αλγορίθμων

HOG algorithm

Ο ανιχνευτής προσώπου HOG της βιβλιοθήκης Dlib βασίζεται στη μέθοδο εξαγωγής χαρακτηριστικών των Felzenszwalb και πέντε ταξινομητές εκπαιδεύτηκαν σε 3000 εικόνες της βάσης δεδομένων Labeled Faces in the Wild. Οι πέντε ταξινομητές εκπαιδεύονται σε πέντε διαφορετικές περιστροφές του προσώπου για να κάνουν τον ανιχνευτή προσώπου πιο περιστρεφόμενο και μια εικόνα σαρώνεται σε πολλαπλές κλίμακες για να κάνει την ανίχνευση προσώπου πιο αμετάβλητη.

Συνοπτικά τα βήματα του αλγορίθμου:

Αρχικά αφου φορτώσουμε την αρχική πρωτότυπη εικόνα προς επεξεργασία θέλουμε να δημιουργήσουμε διάφορα scales της εικόνας προς τα κάτω (downscaling) ώστε να μπορούμε να ανιχνεύουμε ανθρώπινα πρόσωπα όλων τον μεγεθών (κυρίως για πρόσωπα μικρού μεγέθους). Ένας καλός και γρήγορος τρόπος για να γίνει αυτό είναι με την χρήση Bilinear Interpolation (https://en.wikipedia.org/wiki/Bilinear_interpolation). Ο αριθμός των layers αυτων της εικόνας μας εξαρτάται απο το detection window που χρησιμοποιεί ο αλγόριθμος μας, στην δική μας περίπτωση είναι 80x80 pixels, άρα κάνουμε downscaling μέχρι η εικόνας μας να φτάσει στο μέγεθος του detection window. Κάθε βήμα του αλγορίθμου μετα το downscaling θα εφαρμοστεί σε όλα τα layers της εικονας που δημιουργήσαμε.

Στην συνέχεια υπολογίζουμε για κάθε pixel της εικόνας τον προσανατολισμο κλίσης (gradient computation) στον οριζόντιο και κάθετο άξονα καθώς επίσης και τη διαβάθμιση (gradient magnitude). Τα gradients αυτά αφού υπολογιστούν κατηγοριοποιούνται σε 18 signed directions.

Έπειτα η εικόνα μας χωρίζεται σε κελιά μεγέθους 8x8 pixels και κάθε κελί περιγράφεται απο ένα ιστόγραμμα (histogram bins) 18 τιμών/κάδων που αντιστοιχίζονται με τα κατηγοριοποιημένα gradients που έχουμε ήδη υπολογίσει. Κάθε pixel συμβάλει στα 4 γειτονικά του pixel για τον υπολογισμό του ιστογράμματος του κάθε κελιού. Το πόσο συμβάλει κάθε pixel στα γειτονικά του εξαρτάται απο το gradient magnitude και την αποσταση του μέσα στο κελί.

Επόμενο βήμα είναι η κανονικοποίηση των κάδων ιστογράμματος. Ο λόγος που κάνουμε κανονικοποίηση ειναι ότι οι κλίσεις μιας εικόνας είναι ευαίσθητες στον συνολικό φωτισμό. Εάν κάνουμε την εικόνα πιο σκοτεινή διαιρώντας όλες τις τιμές των pixel με 2, το μέγεθος της διαβάθμισης θα αλλάξει κατά το ήμισυ και επομένως οι τιμές του ιστογράμματος θα αλλάξουν κατά το ήμισυ. Στην ιδανική περίπτωση, θέλουμε ο περιγραφέας μας να είναι ανεξάρτητος από τις παραλλαγές φωτισμού. Με άλλα λόγια, θα θέλαμε να «ομαλοποιήσουμε» το ιστόγραμμα έτσι ώστε να μην επηρεάζονται από παραλλαγές φωτισμού. Έτσι οι κάδοι κάθε κελιού ομαλοποιούνται με βάση την ενεργειακή αξία του κάθε κελιού και των 8 γειτόνων του. Η ενεργειακή αξία υπολογίζεται απο τη σχέση energy = $\Sigma^{8}_{n=0}(\text{hist bin}_n + \text{hist bin}_{n+9})^{\wedge 2}$.

Προχωράμε στην εξαγωγή των χαρακτηριστικών του αλγορίθμου (feature extraction), όπου προκύπτει ένα τελικό διάνυσμα χαρακτηριστικών για κάθε κελί που περιέχει 31 χαρακτηριστικά:18 signed normalized bins, 9 unsigned normalized bins καθώς και 4 gradient energy features.

Γραμμικό φίλτρο ταξινόμησης και ανίχνευση: Ο ταξινομητής που χρησιμοποιούμε έχει εκπαιδευτεί με μέγεθος προσώπου 80 × 80 pixel ή 10 × 10 κελιά. Αυτό σημαίνει ότι τα χαρακτηριστικά μιας περιοχής 10 × 10 κελιών, 3100 συνολικά, χρησιμοποιούνται ως είσοδος για τον ταξινομητή. Κάθε ένα από αυτά τα χαρακτηριστικά πολλαπλασιάζεται με ένα συγκεκριμένο βάρος, και όταν το άθροισμα αυτών των πολλαπλασιασμένων δυνατοτήτων υπερβαίνει ένα όριο, η περιοχή ταξινομείται ως πρόσωπο. Αυτό έγινε για κάθε περιοχή 10 × 10 κελιών στην εικόνα χαρακτηριστικών. Υπάρχουν πέντε ταξινομητές, ένας για κάθε περιστροφή του προσώπου, έτσι αυτή η διαδικασία επαναλαμβάνεται πέντε φορές.

Τελευταίο βήμα του αλγορίθμου αυτού είναι η χρήση Non-maximum suppression για να μειώσουμε τις πολλαπλες ανιχνεύσεις που γίνονται στα διάφορα scales και κρατάμε αυτήν με το καλύτερο score.

ERT algorithm

Ο αλγόριθμος ανίχνευσης ορόσημων ERT της βιβλιοθήκης Dlib είναι με βάση τον αλγόριθμο που περιγράφεται από τους Kazemi και Sullivan και εκπαιδεύτηκε στο σύνολο δεδομένων ορόσημο iBUG 300-W.

Ο αλγόριθμος αυτός είναι μια επαναληπτική διαδικασία. Ονομάζεται επίσης μιαπροσέγγιση καταρρακτού-παλινδρόμησης. Κάθε επανάληψη, ή επίπεδο του καταρράκτη, βελτιώνει την εκτίμηση των τοποθεσιών-ορόσημων.

Ο αλγόριθμος ξεκινά με μια αρχική εκτίμηση των τοποθεσιών ορόσημων που βασίζεται στο μέσο σχήμα όλων των εικόνων με τις οποίες έχει εκπαιδευτεί, στο κέντρο της μέσης της εικόνας προσώπου. Σε κάθε ένα από τα 15 επίπεδα του καταρράκτη, συνολικά 500 δέντρα παλινδρόμησης υπολογίζουν το καθένα μια μετατόπιση αυτών των ορόσημων για να κάνει την ανίχνευση πιο ακριβή. Κάθε δέντρο παλινδρόμησης διασχίζεται χρησιμοποιώντας τη διαφορά μεταξύ δύο τιμών pixel σε κάθε διάσπαση, αλλά το κρίσιμο σημείο αυτής της προσέγγισης είναι ότι αυτές οι θέσεις pixel ευρετηριάζονται σε σχέση με την εκτίμηση σχήματος ορόσημου για το τρέχον επίπεδο καταρράκτη. Τα αποτελέσματα όλων των δέντρων παλινδρόμησης σε ένα επίπεδο προστίθενται στην τρέχουσα εκτίμηση ορόσημων, η οποία οδηγεί στην εκτίμηση σχήματος ορόσημου για το επόμενο επίπεδο.

Συνοπτικά τα βήματα του αλγορίθμου:

Αρχικοποίηση: Αρχικοποιούμε την εκτίμηση σχήματος ορόσημου. Αυτός είναι ο μέσος όρος όλων των μορφών ορόσημων με τα οποία εκπαιδεύτηκε ο ανιχνευτής.

Υπολογισμός χαρακτηριστικών: Υπολογίζουμετον μετασχηματισμό ομοιότητας μεταξύ της τρέχουσας εκτίμησης σχήματος και της αρχικής εκτίμησης σχήματος μέσου. Χρησιμοποιούμε αυτόν τον μετασχηματισμό για να υπολογίσσουμετη νέα θέση των σημείων χαρακτηριστικών για τα δέντρα παλινδρόμησης.

Εκτίμηση δέντρου παλινδρόμησης: Διασχίζουμε καθένα από τα 500 δέντρα παλινδρόμησης με βάση τις διαφορές έντασης pixel και προσθέτουμε τα αποτελέσματά τους στην τρέχουσα εκτίμηση σχήματος ορόσημου.

Επανάληψη: Επαναλαμβάνουμε το βήμα υπολογισμού των χαρακτηριστικών και του βήματος παλινδρόμησης για κάθε επίπεδο του καταρράκτη

Implementation Aproaches

1)Naive εκδοση του συνολικού προτζεκτ με χρήση έτοιμου κώδικα απο βιβλιοθήκες που υποστηρίζει το IPU

Μια αρχική προσέγγιση είναι να φτιάξουμε μία βασική υλοποίηση των αλγορίθμων αυτών, χρησιμοποιώντας έτοιμο κώδικα ο οποίος τρέχει σε κάποιο απο τα υποστιρηζόμενα frameworks της Graphcore (tensorflow, pytorch etc). Αντιμετωπίζοντας τους υλοποιημένους αλγορίθμους ως μαύρα κουτιά ο στόχος είναι να τα κάνουμε port στο IPU μηχάνημα, να τα συνδέσουμε μεταξύ τους και να τα τρέξουμε όλα μαζί. Έτσι θα έχουμε ένα σημείο αναφοράς για σύγκριση χρόνου εκτέλεσης με την τελική optimized μορφή.

Μετα απο αναζήτηση στο διαδίκτυο βρήκαμε τις εξής υλοποιήσεις που θα μπορούσαν να δοκιμαστούν:

HOG

https://github.com/cprakashagr/hog-svm-tf (με χρήση tensorflow για SVM training, Opencv lib για HOG)
https://github.com/preethampaul/HOG (tensorflow, numpy, pillow, python 3.5, matplotlib)

ERT

https://github.com/ibarakaiev/face-recognition (python, dlib, matplotlib, numpy, skimage)
https://github.com/jjrCN/ERT-GBDT_Face_Alignment (C++ , opencv, boost)
https://github.com/JiaShun-Xiao/face-alignment-ert-2D (python , menpo, numpy)

2) Custom έκδοση

Σε αυτήν την έκδοση θέλουμε να σπάσουμε το υπολογιστικό κομμάτι του κάθε αλγορίθμου σε μικρότερα κομμάτια τα οποία με custom κώδικα πρέπει να προσαρμοστούν στην αρχιτεκτονική μας. Ο στόχος είναι η παραλληλοποίηση αυτών των μικρότερων κομματιών στο μηχάνημα μας.

Βέβαια θα πρέπει να λάβουμε υπόψη τα tradeoffs, απο την άποψη ποιά κομμάτια αξίζει να παραλληλοποιηθούν καθώς το κόστος επικοινωνίας και μεταφοράς δεδομένων μπορεί να είναι ασύμφορο.

Achieved Progress

Μέχρι 23/01/2021

- Έχει στηθεί ένας σκελετός του κώδικα για να τρέξουμε κώδικα σε IPU μηχάνημα
- Έχει επιτευχθει η σύνδεση της Dlib βιβλιοθήκης με το μηχάνημα και μπορούμε να τρέξουμε τους αλγορίθμους με χρήση CPU
- Μια πρώτη προσπάθεια για πραλληλοποίηση των πρώτων 2 βημάτων του αλγορίθμου HOG με χρήση του IPU (για την custom υλοποίηση)
- Έχουν βρεθεί υλοποιήσεις των αλγορίθμων μας σε υποστηριζόμενες βιβλιοθήκες του μηχανήματος IPU και προσπαθώ να τα συνδέσω ώστε να έχω μια naive έκδοση.

Σχόλια και Απορίες

Μετά απο περίπου 3 μήνες ασχολίας με το μηχάνημα IPU και τους αλγορίθμους που περιγράφτηκαν πιστεύω ότι είναι βασικό να υπάρξει σύντομα μια naive έκδοση του συνολικού προτζεκτ μας. Παρόλα αυτα θέλω να σχολιάσω ότι έχει πάρα πολυ πληροφορία που πρέπει να αφομοιωθεί, θέτοντας το απο την σκοπιά εργαλείων που δεν έχω ξαναχρησιμοποιήσει και πρέπει να εξοικειωθώ καθώς και απο την άποψη υλικού και αυτής της νέας custom αρχιτεκτονικής.

Όσο αναφορά την naive έκδοση, όπως περιγράφω και πιο πάνω έχουν βρεθεί κάποιες έτοιμες υλοποιήσεις (κυρίως σε tensorflow για τον αλγόριθμο HOG) τα οποία δεν έχω καταφέρει ακόμα να κάνω port στο μηχάνημα IPU καθώς είναι και η πρώτη φορα που έρχομαι σε επαφη με το tensorflow framework. Απο το documentation της graphcore προκύπτει αναγκαία η εξοικείωση με το framework αυτο για να επιτευχθεί το port του έτοιμου κώδικα.

Σε αυτό το σημείο λοιπόν προκύπτει το εξής ερώτημα απο μέρους μου:

Να εστιάσω στο να καταφέρω να κάνω port και στην εξοικείωση του tensorflow? Αυτό σημαίνει ότι μπαίνει τελείως στην άκρη η ενασχόληση με την custom έκδοση μας (η οποία μπορεί να είναι και μια απλή sequential υλοποίηση στην αρχή). Η custom έκδοση μας χρησιμοποιεί custom βιβλιοθήκες της graphcore με βασικότερη την poplar δηλαδή ένα ακόμα εργαλείο που θέλει εξοικείωση.

(The Poplar library provides classes and functions to implement and deploy parallel programs on the IPU. It uses a graph-based method to describe programs and, although it can be used to describe arbitrary accelerated computation, it has been designed specifically to suit the needs of artificial intelligence and machine learning applications)

Κυρίως θέλω να μου δώσετε μια κατεύθυνση/γνώμη για το που είναι καλύτερο να εστιάσω γιατί τώρα ασχολούμε λίγο με όλα και το κυρίως θέμα είναι ότι δεν έχω και την κατάρτηση/εμπειρία να κρίνω προς τα που πρέπει να κινηθώ.