Ομάδα 7 **Project Advisor:** Παναγιώτης Μπάκος
Αραπίδης Μανώλης 03116071
Κουτρούμπας Αθανάσιος 03116073

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ

Αναφορά Εργασίας

Inference acceleration on FPGAs - Focus on CNNs

Ακ. έτος 2020-2021, 9ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

Περιεχόμενα

Εισαγωγή

Εισαγωγή για τα Νευρωνικά

Την τελευταία δεκαετία, το μεγαλύτερο κομμάτι της τεχνητής νοημοσύνης έχει στραφεί προς την χρήση των των Νευρωνικών Δικτύων (Neural Networks - NN), είτε σε βιομηχανικό είτε σε ακαδημαϊκό επίπεδο, έναντι άλλων τεχνικών μηχανικής μάθησης. Η συνεχής και αυξανόμενη χρήση τους στην σύγχρονη εποχή έχει κάνει επιτακτική την ανάγκη για την όλο και πιο γρήγορη εκπαίδευση και εκτέλεσή τους.
Ειδικότερα τα Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks - CNN) έχουν αποδειχθεί από τα πιό ακριβή δίκτυα για εφαρμογές επεξεργασίας εικόνας, βίντεο και αναγνώριση φωνής. Αυτή η μεγαλύτερη απόδοση όμως, έχει ώς αποτέλεσμα μεγάλο υπολογιστικό κόστος. Κάποια CNN απαιτούν έως και 38 GOP/s για να κάνουν classification ενός μόνο frame [2].

Οπότε έχει δημιουργηθεί η ανάγκη για εξειδικευμένο hardware που να μπορεί να επιταχύνει την βαριά επεξεργασία που χρειάζονται αυτά τα δίκτυα. Από την επεξεργασία των νευρωνικών στις CPU (10-100 GOP/s) έχουμε πλέον περάσει στην επεξεργασία πάνω σε GPU οι οποίες μπορούν να προσφέρουν μέχρι και 10 TOP/s στο peak performance τους. Επίσης έχουν χρησιμοποιηθεί και τεχνολογίες όπως τα TPUs από την Google, τα οποία είναι ASICs φτιαγμένα ειδικά για Neural Networks και Machine Learning Acceleration, τα οποία μπορούν να φτάσουν μέχρι και τα 92 TOP/s στο peak performance τους [7].
Τα τελευταία χρόνια όμως, έχει αρχίσει και ερευνάται η χρησιμοποιήση των FPGAs στο κομμάτι της επιτάχυνσης εφαρμογών Μηχανικής Μάθησης και ειδικότερα Νευρωνικών Δικτύων.

Εισαγωγή για τα FPGA

Τα FPGA (Field-Programmable Gate Array) είναι ολοκληρωμένα κυκλώματα, τα οποία μπορούν να προγραμματιστούν στο επίπεδο του υλικού από έναν σχεδιαστή, μετά την κατασκευή τους στο εργοστάσιο [9]. Αυτό επιτρέπει να κατασκευαστούν κυκλώματα υψηλής παραλληλίας, το οποία επιτυγχάνει επιδόσεις πολύ μεγαλύτερες από αυτές μιας κοινής επεξεργαστικής μονάδας.
Προγραμματίζονται συνήθως από γλώσσες περιγραφής υλικού (Hardware Description Language - HDL), όπως VHDL, Verilog, αλλά μπορούν και να προγραμματιστούν από γλώσσες υψηλού επιπέδου, όπως C/C++, SystemC, MATLAB, με την μέθοδο Σύνθεσης υψηλού επιπέδου (High Level Synthesis - HLS).

Αρχιτεκτονική FPGA

Το FPGA αποτελείται από έναν πίνακα προγραμματιζόμενων λογικών μονάδων (Configurable Logic Block - **CLB** σε όρους Xilinx και Adaptive Logic Module - **ALM** σε όρους Intel), οι οποίες μπορούν να υλοποιήσουν διάφορες λογικές συναρτήσεις. Όλες αυτές οι λογικές μονάδες περιβάλλονται γύρω από ένα σύστημα διασυνδέσεων (Fabric), το οποίο δρομολογεί τα σήματα μεταξύ των CLB, ώστε να μπορούν να συνδεθούν μεταξύ τους. Επίσης στο FPGA περιλαμβάνονται διάφορες μονάδες εξειδικευμένου υλικού όπως SRAM,

DSP, καθώς και I/O μονάδων οι οποίες επιτρέπουν την επικοινωνία του FPGA με άλλες συσκευές (όπως CPU) και τον προγραμματισμό του.

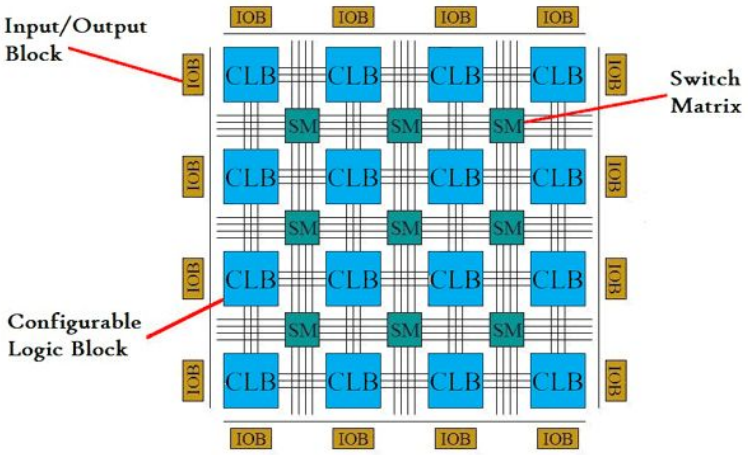


Fig. 1: Αρχιτεκτονική FPGA

CLB

Τα CLBs (ή ALM) συνήθως αποτελούνται από πίνακες αναζήτησης (Lookup Tables - LUTs), κάποια στοιχεία αποθήκευσης (D Flip-Flop), αθροιστές (Full Adders - FA) και πολυπλέκτες, οι οποίοι τους επιτρέπουν πραγματοποιούν λογικές πράξεις, αριθμητικές πράξεις, καθώς και να αποθηκεύουν δεδομένα. Τα CLBs όπως αναφέραμε συνδέονται μέσω του συστήματος διασύνδεσης για να μπορέσουν να συνθέσουν μεγαλύτερα λογικά κυκλώματα, από αυτά που τους επιτρέπουν τα στοιχεία ενός μόνο CLB.

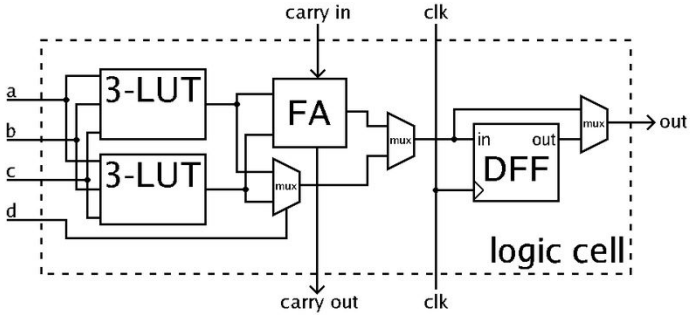


Fig. 2: Συνήθως τα CLBs περιέχουν LUTs 4-εισόδων. Στην εικόνα τα LUTs έχουν χωριστεί σε δύο 3-εισόδων LUTs. Σε κανονική λειτουργία συνδυάζονται μέσω του αριστερού πολυπλέκτη για να συνθέσουν ένα 4-εισόδων LUT.

CPU - FPGA System Architecture

Τα FPGA πλέον συνδέονται σε υπολογιστικά συστήματα ή ενσωματώνονται σε ένα design με μια general purpose CPU για να μπορέσει να γίνει προγραμματισμός σε Software και σε Hardware ώστε να μπορέσει να επιτευχθεί επιτάχυνση της εφαρμογής σε μία συσκευή. Αυτό επιτρέπει τον προγραμματισμό εφαρμογών σε γλώσσες υψηλού επιπέδου, και στα κομμάτια που είναι υπολογιστικά ακριβά να γίνεται προγραμματισμός στο Hardware.
Μια γενική αρχιτεκτονική που συνδυάζει αυτές τις δύο πλατφόρμες χρειάζεται αρχικά ένα δίαυλο επικοινωνίας μεταξύ FPGA-CPU. Συνήθως αυτός γίνεται μέσω του PCIe bus ή του

AXI bus (συνήθως σε SoC πακέτα). Ο κάθε επεξεργαστής έχει πρόσβαση στην δική του μνήμη, αλλά και πρόσβαση στη μνήμη του άλλου. Τα περισσότερα design ενός NN επιταχυντή, χρησιμοποιούν το FPGA για την επιτάχυνση της εφαρμογής και την CPU για τον έλεγχο και συντονισμό του FPGA και τις μεταφορές δεδομένων.

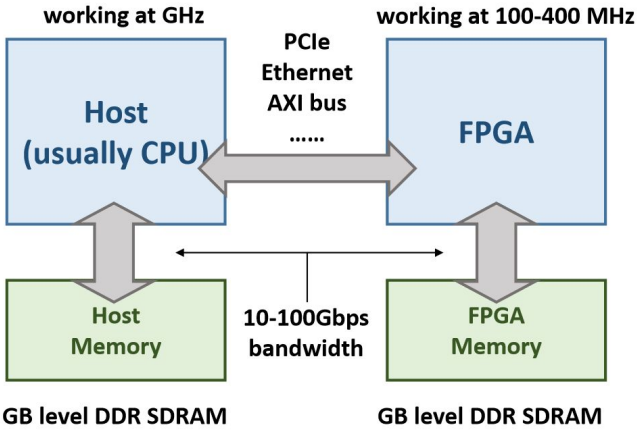


Fig. 3:Γενικότερο παράδειγμα τυπικής αρχιτεκτονικής ενός FPGA-based NN accelerator σε συνεργασία με μία CPU και την system Memory.

Soc

Πλέον, τα FPGA μπορούν να τοποθετούνται μαζί με μικροεπεξεργαστές (όπως ARM) σε ένα κοινό τσιπ, σχηματίζοντας ένα SoC (System On Chip). Αυτό επιτρέπει την ταχύτερη επικοινωνία μεταξύ FPGA-CPU καθώς και γρηγορότερη μεταφορά δεδομένων μεταξύ τους. Τέτοια SoC είναι επίσης είναι αρκετά δημοφιλή σε Embedded συστήματα, όπου ένα χαμηλών δυνατοτήτων embedded υπολογιστικό σύστημα θα μπορεί να εκτελεί βαριές υπολογιστικά εφαρμογές πολύ πιο γρήγορα και με λιγότερη κατανάλωση ενέργειας μέσω του επιταχυντή.

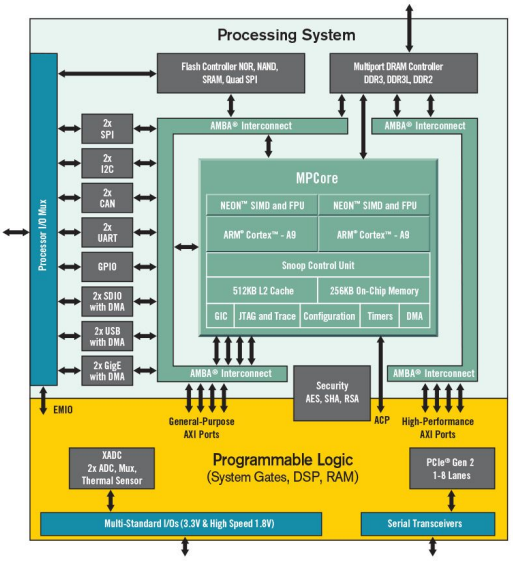


Fig. 4: Ένα τέτοιο παράδειγμα είναι η αρχιτεκτονική Zynq-7000 της Xilinx, η οποία προσφέρει ένα SoC, που περιέχει έναν επεξεργαστή ARM Cortex-A9, μαζί με ένα κομμάτι προγραμματιζόμενης λογικής (FPGA) τεχνολογίας 28nm Artix-7.

Προγραμματισμός FPGA

Ο προγραμματισμός ενός FPGA, όπως αναφέραμε μπορεί να γίνει με μία γλώσσα περιγραφής υλικού (HDL), όπως η Verilog ή η VHDL. Αυτές οι γλώσσες προσφέρουν την μέγιστη ελευθερία προγραμματισμού πάνω σε ένα FPGA, καθώς περιγράφουν την συμπεριφορά και την δομή των κυκλωμάτων (π.χ. το πως θα γίνουν οι διασυνδέσεις μεταξύ στοιχείων). Με τις γλώσσες αυτές μπορεί να επιτευχθεί η μέγιστη αξιοποίηση των πόρων ενός FPGA καθώς και να έχουμε μέγιστες επιδόσεις. Όμως ο χρόνος παραγωγής ενός design σε μία HDL, μπορεί να είναι πάρα πολύ μεγάλος και επίσης δεν είναι και ιδιαίτερα εύκολη η σχεδίαση σε μία γλώσσα περιγραφής υλικού.

Για αυτό το λόγο, τα τελευταία χρόνια με την σχετική άνοδο των FPGA έχει αναπτυχθεί ιδιαίτερα η High Level Synthesis (HLS) σε ένα FPGA με μία γλώσσα υψηλού επιπέδου όπως C/C++ η οποία θα μετατρέψει αυτόματα τον κώδικα αυτό σε μία γλώσσα περιγραφής υλικού. Ο προγραμματισμός σε HLS, επιτρέπει την πολύ πιο εύκολη και γρήγορη σχεδίαση ενός design, σε σχέση με τον αντίστοιχο προγραμματισμό με μία HDL, το οποίο όμως έρχεται πολλές φορές με το κόστος του μη βέλτιστου design. Οι κατασκευαστές FPGA, αλλά και άλλες εταιρείες, παρέχουν πλέον πολλα προγράμματα για τον σχεδιασμό σε HLS. Με αυτά το προγράμματα, ο σχεδιαστής πολλές φορές αρκεί να επιλέξει μια συνάρτηση που θέλει να επιταχύνει και με τα κατάλληλα directives στο πρόγραμμα να ορίσει τις βελτιστοποιήσεις που θα γίνουν στο hardware (όπως Loop Unrolling, Pipelining, Array Partitioning κτλπ).
Κάποια από τα προγράμματα για HLS σχεδιασμό σε FPGA είναι τα παρακάτω:

- Από την Xilinx τα SDSoC και SDAccel, και το Vitis HLS (ex-Vivado HLS)
- Από την Intel υπάρχει ο Intel® High Level Synthesis Compiler
- Από την LegUp Computing το LegUp HLS
- Από την Mathworks για χρήση της MATLAB το HDL Coder from Mathworks
- Από την Khronos Group το OpenCL, το οποίο γενικά προσφέρεται για ετερογενή προγραμματισμό (CPU, GPU, FPGA)

Training και Inference

Οι αλγόριθμοι νευρωνικών δικτύων τρέχουν σε δύο διαφορετικές φάσεις, στην φάση της εκπαίδευσης (training) και αφού εκπαιδευτούν περνούν στην φάση εξαγωγής συμπερασμάτων (inference). Η διαδικασία του training έχει κοινά σημεία με αυτή του inference [6].

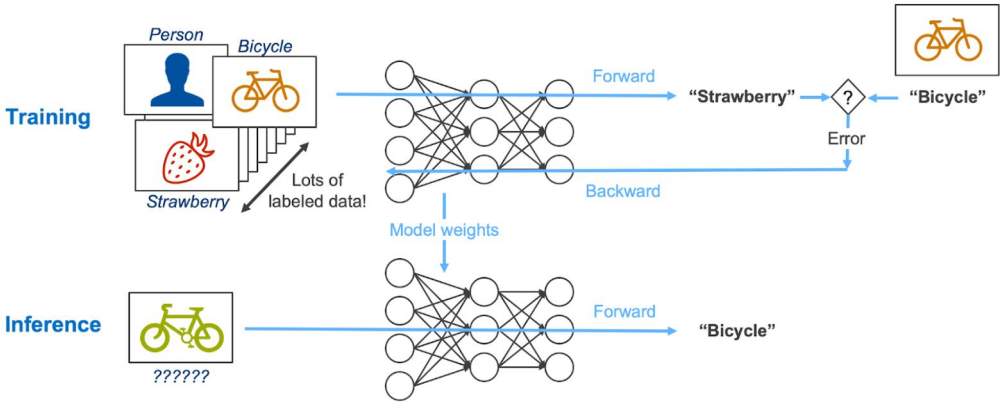


Fig. 5: Training και Inference σε ένα Νευρωνικό Δίκτυο

Training

Πιο συγκεκριμένα κατα την διάρκεια της εκπαίδευσης ενός νευρωνικού δικτύου ένα σύνολο από δεδομένα, που το έχουμε ήδη κατηγοριοποιήσει, δίνεται ως είσοδο. Το training χωρίζεται σε στάδια. Το πρώτο είναι το forward propagation, κατα το οποίο η είσοδος περνάει από τα διαδοχικά layers του δικτύου και εν τέλει παράγει μια έξοδο.Το δεύτερο είναι το backward propagation, η πρόβλεψη που δίνεται ως έξοδο συγκρίνεται με την ήδη γνωστή. Η απόκλιση τους μεταδίδεται προς τα πίσω, διαδοχικά μεταξύ των layers, και χρησιμοποιείται για να ανανεωθούν τα βάρη σε καθένα από αυτά.
Τα παραπάνω επαναλαμβάνονται διαδοχικά μέχρι να φτάσουμε στην επιθυμητή απόκλιση ή να σταματήσουμε να παρατηρούμε βελτίωση στο accuracy που πετυχαίνουμε.

Inference

Η διαδικασία του inference χρησιμοποιεί ένα ήδη εκπαιδευμένο νευρωνικό δίκτυο. Εκτελεί μόνο forward propagation, δέχεται άγνωστες εισόδους και επιχειρεί να τις κατηγοριοποιήσει.

Διαφορές Training - Inference

Μια ακόμα διαφορά τους είναι ότι το training γίνεται συνήθως μόνο μία φορά, ενώ ο σκοπός του inference είναι να επαναλαμβάνεται διαρκώς. Επιπλέον, η εκπαίδευση είναι πιο απαιτητική διαδικασία και μπορεί να διαρκέσει από ώρες μέχρι και μέρες, κάτι που θεωρείται αποδεκτό δεδομένου ότι θα γίνει μόνο μια φορά. Αντίθετα το inference, ειδικά για real-life applications, είναι κρίσιμο να διαρκεί όσο το δυνατόν λιγότερο (πχ συστήματα αυτόματης οδήγησης).

Διαφορές FPGA και GPU ως Επιταχυντές

Η GPU είναι πιο διαδεδομένη πλατφόρμα που χρησιμοποιείται για την επιτάχυνση του inference για τα CNN. Τα FPGA φαίνεται να αποκτούν όλο και περισσότερη δημοφιλία για την παραπάνω εφαρμογή, καθώς έχουν καταφέρει να πετύχουν επιδόσεις πολύ κοντά σε αυτές των GPU ενώ υπερτερούν σε άλλους τομείς όπως την κατανάλωση ενέργειας και την ευελιξία της αρχιτεκτονικής του υλικού. Ακολουθεί ανάλυση για καθένα από τα παραπάνω.

Υπολογιστική Δύναμη

Η ευρέως χρήση των GPU οφείλεται στις υψηλές υπολογιστικές ικανότητες τους, σε κάποιες περιπτώσεις μέχρι και 11 TFLOP/s [2]. Παρόλο αυτά, η όλο και μεγαλύτερη ανάπτυξη των FPGA τους έχει επιτρέψει να πλησιάσουν αρκετά τις δυνατότητες των GPU, καταγράφοντας αποδόσεις μέχρι και 9.2 TFLOP/s [2].

Την παραπάνω διαφορά έρχεται να μετριάσει το μεγάλο πλήθος on-chip μνήμης των FPGA. Οι προσβάσεις σε εξωτερική μνήμη μπορεί να είναι bottleneck για την επιταχυνση των CNN και μειώνοντας αυτή την καθυστέρηση πετυχαίνουμε καλύτερες επιδόσεις. Πέρα από την μεγαλύτερη ταχύτητα που εξασφαλίζει η on-chip μνήμη, η πρόσβαση σε αυτή έχει σημαντικά λιγότερες απαιτήσεις σε ενέργεια.

Η ανάπτυξη των νευρωνικών δικτύων είναι ραγδαία και παρατηρείται η τάση να χρησιμοποιούνται τύποι δεδομένων, όπως ternary, binary μέχρι και custom datatypes έναντι συνηθέστερων και μεγαλύτερης ακρίβειας (FP32 ή INT16). Η ευελιξία των FPGA να μπορούν να υλοποιήσουν απευθείας στο υλικό πληθώρα datatypes, τα κάνει ιδανικά για τέτοιες εφαρμογές. Σε αντίθεση οι κατασκευαστές των GPU θα πρέπει να αλλάξουν τις υλοποιησεις των αρχιτεκτονικών για να μπορούν να συμβαδίζουν με τα παραπάνω, κάτι εξαιρετικά χρονοβόρο και κοστοβόρο.

Κατανάλωση ενέργειας

Τα FPGA είναι γνωστά για την χαμηλή κατανάλωση ενέργειας, καθώς πολλές φορές έχουν ως άξονα σχεδιασμού την χρήση σε ενσωματωμένα συστήματα (π.χ. αυτόματη οδήγηση). Η [10] δείχνει ότι η Xilinx Virtex Ultrascale+ έχει 4 φορές μικρότερη κατανάλωση από την NVidia Tesla V100 σε γενικού σκοπού υπολογισμούς. Ο κύριος λόγος που οι GPU απαιτούν περισσότερη ενέργεια, είναι ότι οι υπολογιστικές μονάδες τους έχουν μεγαλύτερη πολυπλοκότητα στο υλικό.

Ευελιξία αρχιτεκτονικής

Όπως αναφέραμε και παραπάνω οι υλοποιήσεις των νευρωνικών δικτύων αλλάζουν διαρκώς, οπότε υπάρχει ανάγκη η πλατφόρμα να μπορεί αλλάζει το hardware για να ταιριάζει το software. Το παραπάνω το καταφέρνουν τα FPGA χάριν την μεγάλης ευελιξία τους, καθώς αποτελούνται από συνδυασμό προγραμματιζόμενης λογικής, DSPs και BRAM blocks. Η διασύνδεση μεταξύ των υπολογιστικών μονάδων καθώς και η πορεία των δεδομένων είναι προγραμματιζόμενη, ακόμα και κατά το runtime, μειώνοντας το latency χωρίς υστέρηση από μεριά παραλληλίας.

Παρόλο αυτά ο προγραμματισμός των FPGA παρουσιάζει κάποιες δυσκολίες, καθώς απαιτείται ο προγραμματιστής να έχει γνώση του hardware. Αντίθετα ο προγραμματισμός των GPU δεν έχει την παραπάνω ιδιαιτερότητα και γίνεται αρκετά εύκολα μέσω CUDA ή OpenCL. Για την αντιμετώπιση αυτού του προβλήματος έχει παραχθεί το μοντέλο HLS (High Level Synthesis) για προγραμματισμό σε γλώσσες όπως C/C++. Τέλος η [3] μας δείχνει ότι υπάρχουν προγράμματα, όπως το LegUp, για απευθείας μετατροπη παραλοποιημένου κώδικα (με χρήση OpenMP ή Pthreads) σε HLS κώδικα, κάνοντας πιο εύχρηστα τα FPGA.

Συνελικτικά Νευρωνικά Δίκτυα - CNN

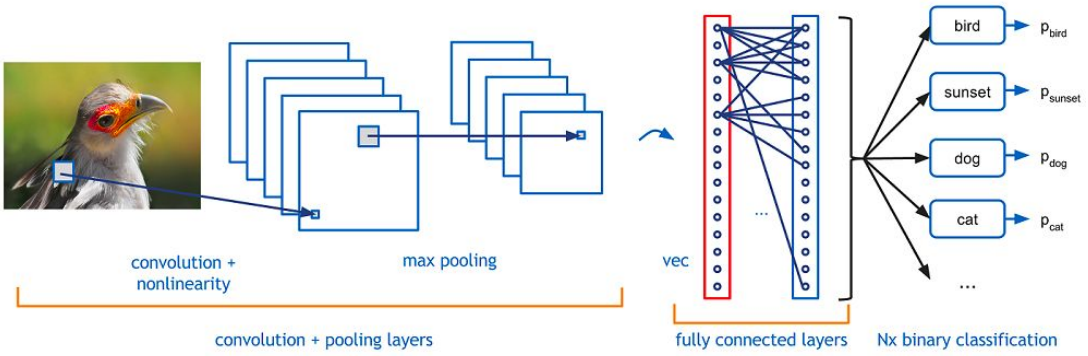


Fig. 6: Αφαιρετική αναπαράσταση αρχιτεκτονικής CNN

Όπως αναφέραμε, τα CNN είναι από τα πιο πολυχρησιμοποιημένα Βαθιά Νευρωνικά Δίκτυα για χρήσης Μηχανικής Μάθησης, καθώς προσφέρουν εξαιρετικές αποδόσεις σε εφαρμογές με επεξεργασία εικόνας, όμως έχουν βαρύ υπολογιστικό κόστος, οπότε και για αυτό έχουν απασχολήσει περισσότερο για την επιτάχυνση τους.

Όσον αφορά την λειτουργία των CNN, γενικά έχουν εμπνευστεί από τον τρόπο με τον οποίο ο άνθρωπος λαμβάνει και εξάγει πληροφορία μέσω τον βιολογικών οπτικών νευρώνων. Γενικώς αποδίδουν καλύτερα από τα κλασικά Νευρωνικά δίκτυα σε εφαρμογές που έχουν ως εισόδους εικόνες, καθώς πραγματοποιούν εξαγωγή χαρακτηριστικών, ώστε από εξαγωγή low level χαρακτηριστικών (όπως γεωμετρικά σχήματα) να μπορεί να πραγματοποιηθεί εξαγωγή πιο high level χαρακτηριστικών (όπως χέρια, πόδια, μάτια).

Η βασικές διαφορές τους με τα κλασικά Νευρωνικά Δίκτυα, είναι ότι οι νευρώνες κάθε επιπέδου δεν είναι πλήρως συνδεδεμένοι με τους νευρώνες των γειτονικών επιπέδων, και ότι οι νευρώνες μοιράζονται κοινά βάρη μεταξύ τους, αντί να έχει κάθε νευρώνας και ξεχωριστό βάρος. Επίσης, ανάμεσα στα επιμέρους επίπεδα των νευρώνων υπάρχουν διατάξεις που πραγματοποιούν δειγματοληψία ώστε να μειωθούν η διαστατικότητα τις εισόδου.

Αρχιτεκτονική CNN

Τα CNN γενικά αποτελούνται από ένα ή περισσότερα από **Convolution layers** (Συνελικτικό επίπεδο) μαζί με **Activation Layers** (Επίπεδο Ενεργοποίησης) και **Pooling layers** (Επίπεδο υποδειγματοληψίας). Τέλος υπάρχει ένα ή περισσότερα **Fully Connected Layers** (Πλήρως συνδεδεμένα επίπεδα), τα οποία είναι υπεύθυνα για την τελική ταξινόμηση/απόφαση του δικτύου. Επίσης, πλέον χρησιμοποιούνται και κάποια **Batch-Normalization Layers** για την επιτάχυνση της διαδικασίας του training.
Επιγραμματικά:

1. Convolution layers (Συνελικτικά Layers)
2. Activation Layers (Layer Ενεργοποίησης)
3. Pooling layers (Layer υποδειγματοληψίας)
4. Batch-Normalization Layers
5. Fully Connected Layers (Πλήρως συνδεδεμένα Layer)

Παρακάτω αναφερόμαστε αναλυτικά στα στάδια για την διαδικασία του inference στα CNN, δηλαδή στην διαδικασία που που γίνονται feed-forward **Β το πλήθος εικόνες-είσοδοι (Batch Size)** στα **L layers** του νευρωνικού.

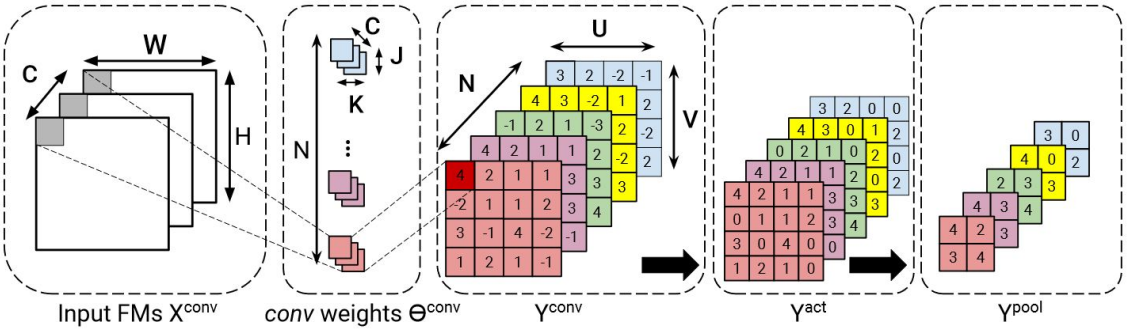


*Fig. 7: Feed forward propagation στα **Conv**, **Activation** και **Pooling** layers*

Convolution Layer

Τα convolution layers, είναι υπεύθυνα για την εξαγωγή χαρακτηριστικών από τις εισόδους, μέσω συνέλιξης κατάλλων 3D φίλτρων (kernels) σε αυτές. Κάθε είσοδος έχει ένα **βάθος C** (π.χ. C=3 για RGB εικόνες ή C=4 για CMYK εικόνες), και μπορεί να είναι είτε μία εικόνα είτε η έξοδος που έχει προκύψει από κάποιο προηγούμενο layer. Η εφαρμογή ενός 3D φίλτρου σε μία 3D εικόνα έχει ως αποτέλεσμα ένα 2D Feature Map (FM).

Σε κάθε convolution layer, έχουμε **Ν αριθμό φίλτρων**, οπότε και παράγονται Ν δισδιάστατα FM. Τα output FM, έχουν συνήθως διαστάσεις μικρότερες τις εισόδου (εκτός και να προστεθεί κάποιο padding, ώστε να μην υπάρχει απώλεια διάστασης) και έχουν βάθος όπως αναφέραμε ίδιο με το πλήθος των **Ν φίλτρων**.

Input FM

Τα input FMs έχουν διαστάσεις $H \times W \times C$ (ή πιο σωστά $B \times H \times W \times C$, αν λάβουμε υπόψιν μας και το Batch Size).

Output FM

Τα output FMs έχουν διαστάσεις $V \times U \times N$ (ή πιο σωστά $B \times V \times U \times N$, αν λάβουμε υπόψιν μας και το Batch Size).

Φίλτρα-Kernels

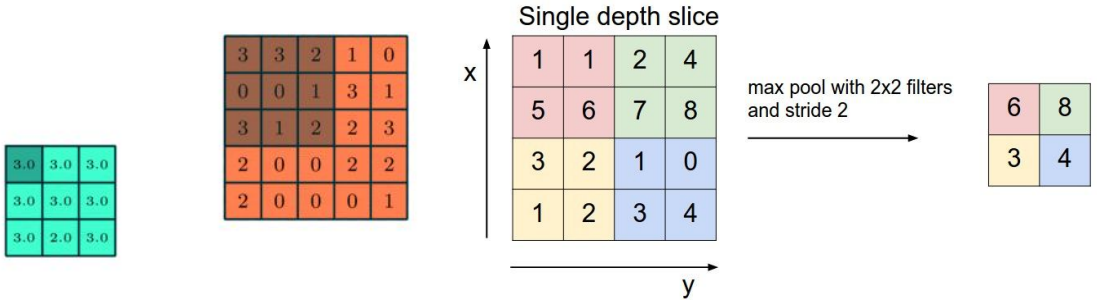
Τα φίλτρα έχουν διαστάσεις $J \times K \times C$ (ή πιο σωστά $B \times J \times K \times C$, αν λάβουμε υπόψιν μας και το Batch Size).

Activation Layer

Μετά από κάθε convolution layer, συνήθως ακολουθείτε ένα activation layer, το οποίο εφαρμόζει μια μη-γραμμική συνάρτηση σε κάθε τιμή του Output FM. Στα αρχικά CNN, εφαρμόζονταν συναρτήσεις σαν την TanH και την Σιγμοειδή. Όμως επειδή αυτές αντιστοιχίζουν την είσοδο σε ένα κλειστό διάστημα όπως (-1,1) ή (0,1) αυτό προκαλεί προβλήματα σε βαθιά δίκτυα, όπως την σταδιακή απώλεια της αλλαγής στην πρόβλεψη με αποτέλεσμα την πολύ αργή σύγκλιση. Επίσης, αυτές οι συναρτήσεις είναι και αρκετά υπολογιστικά ακριβές για να υπολογιστούν, οπότε πολλές φορές υλοποιούνται με την χρήση Look-up tables, με προϋπολογισμένες τιμές για να μειωθεί ο χρόνος υπολογισμού.
Για να αποφευχθούν τέτοια προβλήματα, πλέον εφαρμόζεται συναρτήσεις όπως η ReLU (Rectified Linear Unit), η οποία αντιστοιχίζει την είσοδο x στο ανοιχτό διάστημα $(0,\ max(x))$. Έτσι με αυτό τον τρόπο εισάγεται μη-γραμμικότητα και παράλληλα η είσοδος δεν περιορίζεται σε ένα κλειστό διάστημα. Αυτό επιτρέπει ταχύτερους χρόνους εκπαίδευσης και πολύ λιγότερη υπολογιστική πολυπλοκότητα [11].

Pooling Layer

Μετά από ένα ή παραπάνω διαδοχικά convolution layer (και το αντίστοιχο activation), υπάρχει συνήθως ένα pooling layer, το οποίο εκτελεί μια υποδειγματοληψία στο κάθε channel του input FM του επόμενου layer, επιλέγοντας την μέση (Average Pooling) ή την μέγιστη τιμή (Max Pooling) μιας δοσμένης περιοχής από τιμές. Αυτό έχεις ως αποτέλεσμα να μειωθούν οι διαστάσεις του input FM, και για το νευρωνικό να μπορέσει κάνει τα FM να αναγνωρίζουν τα χαρακτηριστικά που έχουν βρει, ανεξάρτητα την θέση που έχουν στην εικόνα. Αυτό κάνει πιο αποτελεσματική και αξιόπιστη την πρόβλεψη.



(a) *(b)*

Fig. 8: (a) Max Pooling με 3x3 φίλτρα (και stride 1) σε FM 5x5
(b) Max Pooling με 2x2 φίλτρα (και stride 2) σε FM 4x4

Fully Connected Layer

Αυτό είναι το τελευταίο επίπεδο των CNN, και χρησιμοποιείται ώστε να γίνει το τελικό classification task από το νευρωνικό. Πρόκειται για κάποια layers, με πλήρως συνδεδεμένους νευρώνες από το κάθε layer, και χωρίς μοιραζόμενα βάρη.

Batch-Normalization Layer

Τα Batch-Normalization Layers, έχουν εισαχθεί για να επιταχύνουν την διαδικασία του training, κανονικοποιώντας ένα batch εισόδων ώστε να έχουν μηδενική μέση τιμή και διακύμανση ένα, ώστε να σταθεροποιήσει την διαδικασία μάθησης και να μειώσει δραματικά των αριθμό των εποχών μάθησης [8].

Συνέλιξη στα CNN

3D Συνέλιξη

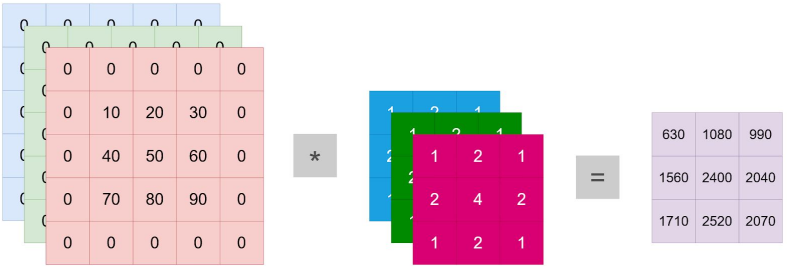


Fig. 9: 3D Convolution: 3D Convolution 1 input FM με 1 φίλτρο για παραγωγή Output FM

Συνολικά σε ένα Conv Layer έχουμε N το πλήθος 3D συνελίξεις του input FM με τα N φίλτρα. Αυτό θα οδηγήσει στην δημιουργία N το πλήθος output FM.

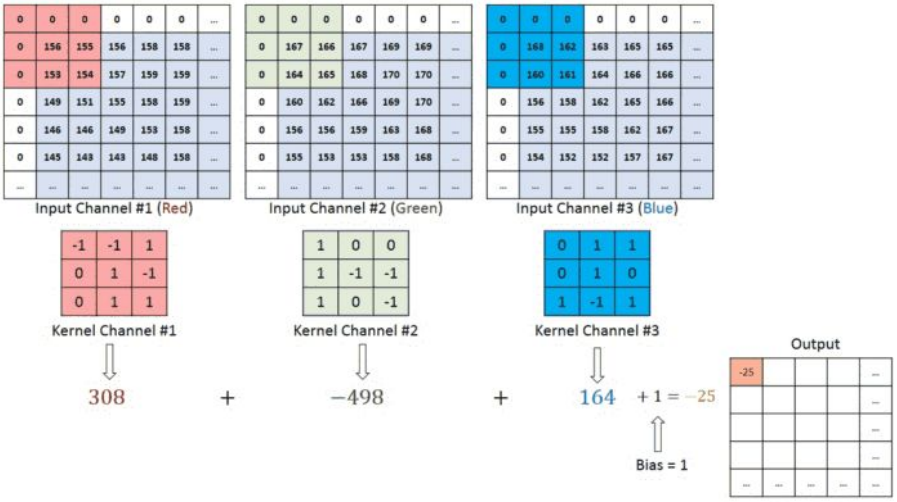


Fig. 10: 3D Convolution με ένα φίλτρο: 2D convolution και άθροισμα από κάθε channel για παραγωγή Output FM

Η συνέλιξη που συμβαίνει μεταξύ ενός input FM και ενός φίλτρου, πρόκειται για μια 3D συνέλιξη, όπου έχουμε 2D συνέλιξεις για κάθε επίπεδο της διάστασης C, και τα αποτελέσματα της συνέλιξης από κάθε επίπεδο αθροίζονται και αποτελούν την τελική τιμή της 3D συνέλιξης για κάθε κελί του ενός input FM.

2D Συνέλιξη

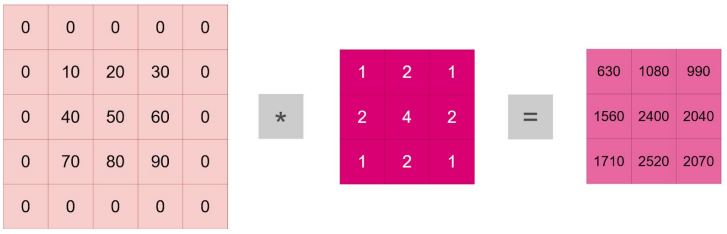


Fig. 11: 2D Convolution: 2D Convolution 1 input FM με 1 φίλτρο για παραγωγή Output FM

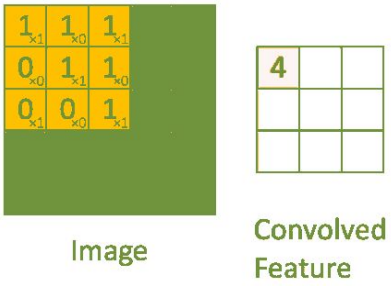


Fig. 12: 2D Convolution: Element-wise πολλαπλασιασμός ενός φίλτρου 3x3 με 5x5 Input FM και άθροισμα γινομένων για παραγωγή τελικού Output FM

Η 2D συνέλιξη του input FM με το ένα φίλτρο, είναι στην ουσία ένας element-wise πολλαπλασιασμός πινάκων, των τιμών του φίλτρου με τις τιμές του input FM πάνω στις οποίες εφαρμόζεται το φίλτρο. Αυτά τα γινόμενα αθροίζονται και αποτελούν το αποτέλεσμα της συνέλιξης για ένα στοιχείο του τελικού πίνακα.

Stride

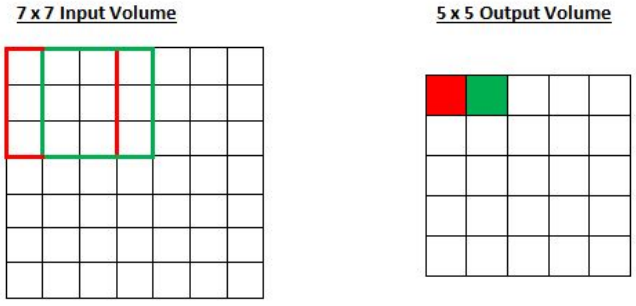


Fig. 13: (a) Stride = 1, για convolution πίνακα εισόδου 7x7 έχουμε πίνακα εξόδου 5x5

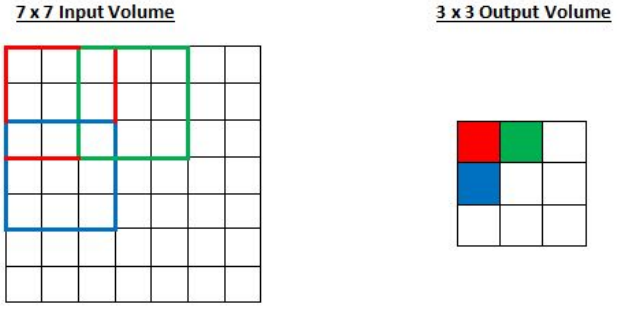
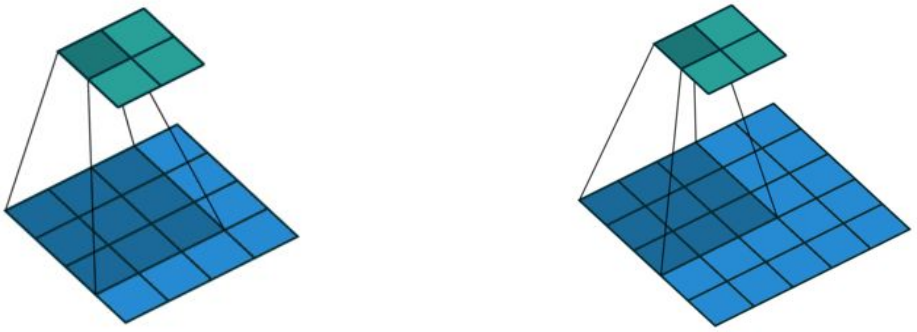


Fig. 13: (b) Stride = 2, για convolution πίνακα εισόδου 7x7 έχουμε πίνακα εξόδου 3x3. Σε περίπτωση που Stride = 3, δεν θα χωρούσαν κάποιες τιμές του φίλτρου στον πίνακα εισόδου κατά την 2η ολίσθηση.

Για να υπολογιστούν όλες οι τιμές του τελικού πίνακα, το φίλτρο ολισθαίνει πάνω στον πίνακα εισόδου κατά μία σταθερή τιμή που ονομάζεται stride, και μπορεί να είναι ίση με 1 (για ολίσθηση κατά μια θέση όπως στην εικόνα) ή μεγαλύτερης τιμής. Για να έχουμε μεγαλύτερη τιμή stride, πρέπει να προσέξουμε ώστε το φίλτρο να χωράει ολόκληρο στον πίνακα εισόδου και να μην βγαίνουν τιμές του φίλτρου εκτός των ορίων του πίνακα. Επίσης προφανώς όσο αυξάνουμε το stride, καταλήγουμε να έχουμε μικρότερες διαστάσεις στον πίνακα εξόδου.



(a) *(b)*

Fig. 14: (a) Stride = 1 σε πίνακα εισόδου 4x4 με φίλτρο 3x3 και πίνακα εξόδου 2x2
(b) Stride = 2 σε πίνακα εισόδου 5x5 με φίλτρο 3x3 και πίνακα εξόδου 2x2

Padding και Διαστάσεις FM

Γενικά στα CNN μετά το κάθε επίπεδο συνέλιξης οι διαστάσεις των Output FM θα μειώνονται σταδιακά. Σε πολυεπίπεδα δίκτυα, αυτό μπορεί να αποτελέσει πρόβλημα καθώς δεν θα ύστερα από κάποια επίπεδα οι διαστάσεων μπορεί να μικρύνουν τόσο που δεν θα διευκολύνουν την μάθηση. Για να μειώσουμε το φαινόμενο μείωσης των διαστάσεων του αρχικού πίνακα, χρησιμοποιείται η τεχνική του Padding. Με αυτήν την τεχνική, προστίθενται όσες γραμμές και στήλες από μηδενικά χρειάζονται ώστε να έχουμε τις επιθυμητές διαστάσεις για τον πίνακα εξόδου. Τα μηδενικά αυτά δεν επηρεάζουν το αποτέλεσμα της συνέλιξης καθώς κατά τον πολλαπλασιασμό θα δίνουν τιμή 0, οπότε καμία επιρροή στο τελικό άθροισμα.
Η εξίσωση που υπολογίζει τις διαστάσεις του πίνακα εξόδου μετά από ένα convolution layer:

$$O = \frac{(N-F+2P)}{S} + 1$$

όπου:

- O, οι διαστάσεις του πίνακα εξόδου
- N, οι διαστάσεις του πίνακα εισόδου
- F, οι διαστάσεις του φίλτρου
- P,το padding
- S, το stride.

Για παράδειγμα:
Αν πίνακας εισόδου 7x7 ($N = 7$) και φίλτρο 3x3 ($F = 3$), χωρίς padding ($P = 0$) και stride 1 ($S = 1$), τότε διαστάσεις πίνακα εξόδου:

$$O = \frac{(N-F+2P)}{S} + 1 = \frac{7-3}{1} + 1 = 5$$

όπως στο από πάνω παράδειγμα.

Για παράδειγμα:
Αν πίνακας εισόδου 32x32 ($N = 32$) και φίλτρο 5x5 ($F = 5$), χωρίς padding ($P = 0$) και stride 1 ($S = 1$), τότε διαστάσεις πίνακα εξόδου:

$$O = \frac{(N-F+2P)}{S} + 1 = \frac{32-5}{1} + 1 = 28$$

Για να μείνουν οι διαστάσεις εισόδου, ίδιες με τις διαστάσεις εξόδου τι padding πρέπει να βάλουμε;

$$O = 32 \Rightarrow \frac{(N-F+2P)}{S} + 1 = 32 \Rightarrow \frac{32-5+2P}{1} + 1 = 32 \Rightarrow P = 2$$

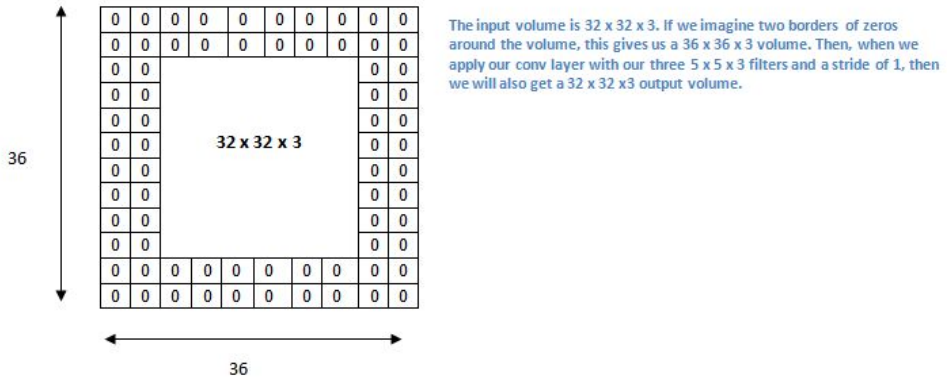


Fig. 15: Padding = 2, σε πίνακα 32x32 με φίλτρο 5x5 και stride 1, για να έχουμε ίδια διάσταση πίνακα εξόδου (32x32)

Παραλληλίες στα CNN

Λόγω του μεγάλου αριθμού υπολογισμών που γίνονται ώστε να πραγματοποιηθεί η διαδικασία του inference στα CNN είναι αναγκαίο να εκμεταλλευτούμε τις παραλληλίες που εμφανίζονται κατά την επεξεργασία τους. Παρακάτω παραθέτουμε τις παραλληλίες που διαπιστώνονται όπως παρουσιάζονται και στο paper [2]:

Batch Parallelism

Παράλληλο inferring πολλών εισόδων

- Επειδή τα CNN μπορούν να κάνουν ταξινόμηση πολλών εικόνων ταυτόχρονα ομαδοποιώντας τα σε ένα batch μεγέθους B, μπορούμε να έχουμε επαναχρησιμοποίηση των ίδιων φίλτρων σε κάθε layer και έτσι να έχουμε ελαχιστοποίηση των accesses στην εξωτερική μνήμη.

Inter-Layer Parallelism

Παραλληλία μεταξύ των layer

- Στα CNN λόγω της feed-forward ροής τους, υπάρχει εξάρτηση δεδομένων μεταξύ των layer, λόγο της feed-forward ιεραρχικής δομής τους. Οπότε η επεξεργασία των layers μπορεί να γίνει pipelined, δηλαδή μπορεί να ξεκινήσει η επεξεργασία του layer (l) πριν ακόμα τελειώσει η επεξεργασία του layer ($l-1$). Πιο αναλυτικά για την 3η διάσταση όλοι οι N πίνακες σε κάθε layer είναι ανεξάρτητοι μεταξύ τους. Για παράδειγμα, το activation του 1ου FM μπορεί να γίνει με το που τελειώσει το convolution του 1ου φίλτρου, ταυτόχρονα με το convolution του 2ου φίλτρου. Αντίστοιχα το pooling του 1ου Activated FM μπορεί να γίνει με το που τελειώσει το activation του 1ου φίλτρου, ταυτόχρονα με activation του 2ου φίλτρου και το convolution του 3ου φίλτρου κλπ.

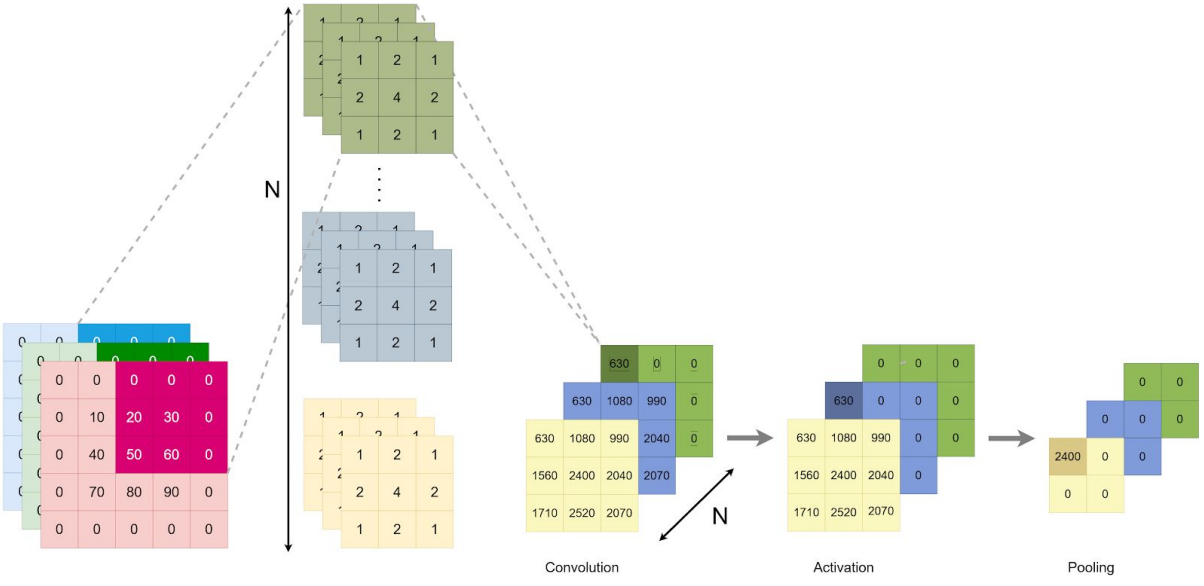


Fig. 16: Inter-Layer Parallelism

Inter-FM Parallelism

Παραλληλία μεταξύ των επεξεργασιών των Output FM

- Κάθε FM που προκύπτει ως έξοδος από το conv layer μπορεί να υπολογιστεί ανεξάρτητα από τα υπόλοιπα. Άρα εφόσων έχουμε N το πλήθος των φίλτρων, τα N Feature Maps μπορούν να επεξεργαστούν παράλληλα.
    - Άρα έχουμε N μέγεθος παραλληλίας.

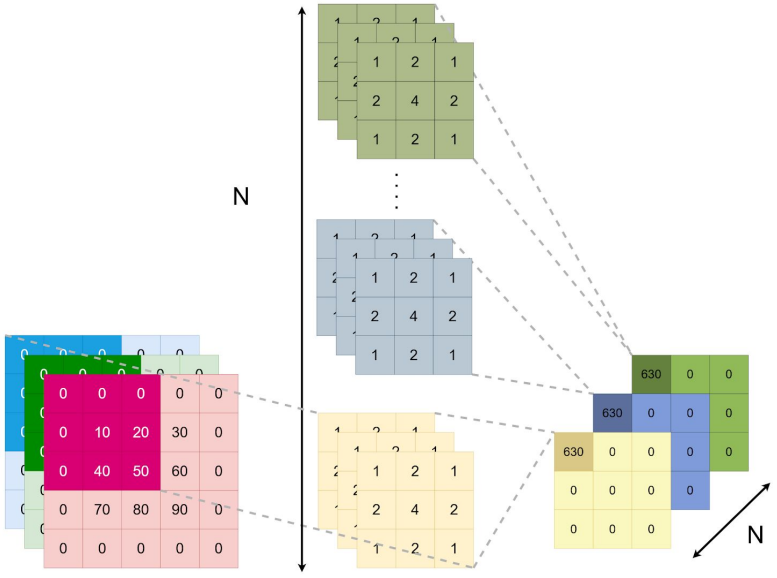


Fig. 17: Inter-FM Parallelism

Intra-FM Parallelism

Παραλληλία ενδιάμεσα της επεξεργασίας ενός Output FM

- Σε κάθε FM το κάθε pixel του μπορεί να υπολογιστεί ταυτόχρονα με τα υπόλοιπα, αφού ο υπολογισμός του κάθε pixel είναι ανεξάρτητος του υπολογισμού των υπόλοιπων.
    - Άρα έχουμε $V \times U$ μέγεθος παραλληλίας.

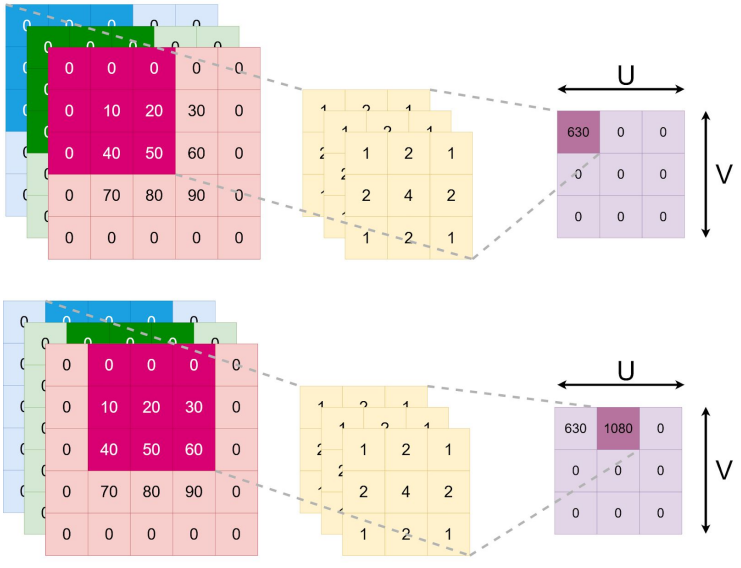


Fig. 18: Intra-FM Parallelism

Inter-Convolution Parallelism

Παραλληλία μεταξύ των συνέλιξεων των Output FM

- Σε κάθε conv layer έχουμε 3D συνέλιξη C input images με C filters (π.χ. C=3, αν έχουμε RGB εικόνες, οπότε 1 εικόνα για κάθε RGB κανάλι). Αυτές οι συνελίξεις μπορούν να εκφραστούν ως ένα άθροισμα των 2D συνελίξεων της κάθε input εικόνας με το αντίστοιχο φίλτρο (από τις C). Άρα αυτές οι C το πλήθος 2D συνελίξεις μπορούν να γίνουν παράλληλα.
    - Άρα έχουμε C μέγεθος παραλληλίας.

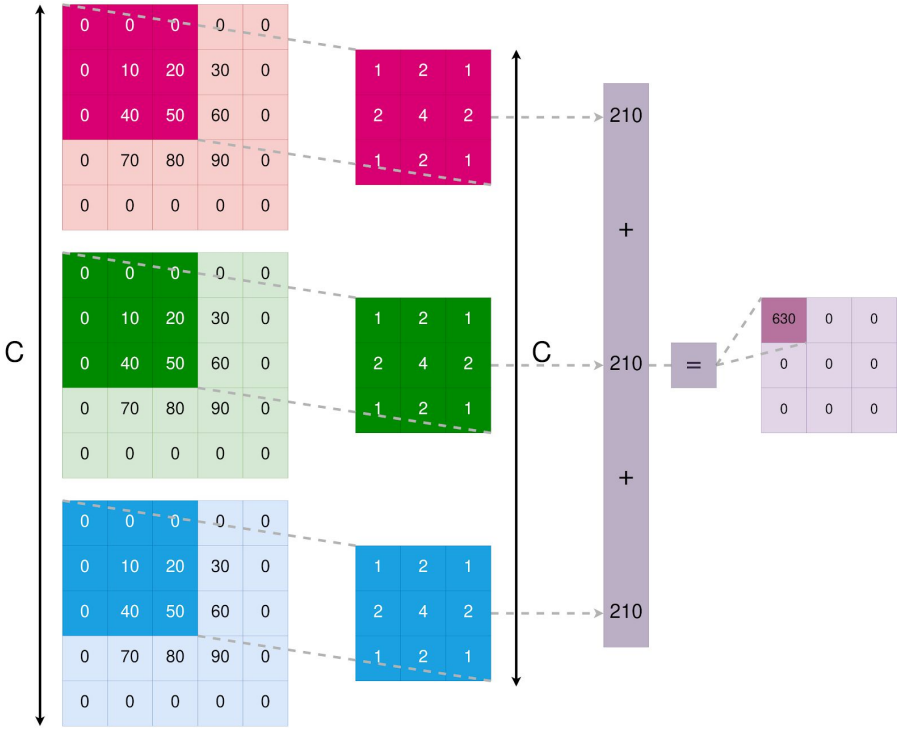


Fig. 19: Inter-Convolution Parallelism

Intra-Convolution Parallelism

Παραλληλία ενδιάμεσα της συνέλιξης ενός Output FM

- Η κάθε 2D συνέλιξη που συμβαίνει σε κάθε conv layer, μπορεί να γίνει pipelined. Η 2D συνέλιξη είναι ένα άθροισμα γινομένων, οπότε με τη χρήση MAC operations μπορεί να γίνει pipelined όπως αντίστοιχα τέτοια operations.
    - Άρα έχουμε $P_J \times P_K$ μέγεθος παραλληλίας.

Επιτάχυνση των CNN στα FPGA

Περιορισμένοι Πόροι FPGA

Όπως αναφέραμε τα CNN προσφέρουν πολλές επιλογές για παραλληλισμό. Όμως, λόγω του περιορισμού των πόρων του FPGA, σπάνια μπορούν να εκμεταλλευτούν όλες αυτές τις παραλληλίες, ειδικά με τις τοπολογίες των βαθιών νευρωνικών δικτύων. Πολλές φορές δεν μπορεί να γίνει πλήρως “Unrolled” ούτε ένα μόνο Conv layer [2].

Περιορισμένοι on-chip Μνήμη FPGA

Ένα άλλο πρόβλημα που παρουσιάζεται είναι ο διαθέσιμος χώρος αποθήκευσης στα on-chip memory units (όπως registers και SRAM) είναι περιορισμένος και σπάνια μπορεί να χωρέσει ολόκληρα τα μεγέθη των μοντέλων των βαριών NN. Για παράδειγμα τυπικές υλοποιήσεις CNN χρειάζονται 100-1000 MB μνήμης ενός το μέγεθος της SRAM του μεγαλύτερου FPGA chip είναι μόλις 50 MB [1]. Οπότε για να καλυφθεί αυτό το κενό χώρο, συνήθως εμπλέκονται και off-chip DDR memories. Όμως το χαμηλότερο bandwidth και η υψηλότερη ενεργειακή κατανάλωση των DDR μνημών περιορίζει κατά πολύ την απόδοση του συστήματος.

Παρακάτω παρουσιάζεται και ένα διάγραμμα που δείχνει την διαφορά μεταξύ της διαθέσιμης μνήμης on-chip στο FPGA και του απαιτούμενου χώρου για διάφορα γνωστά CNN μοντέλα. Οι μπάρες αντιπροσωπεύουν την μνήμη σε registers (με μπλε) και σε SRAM (με πορτοκαλί). Οι διακεκομμένες γραμμές, αναπαριστούν τον απαιτούμενο χώρο για όλες τις παραμέτρους των μοντέλων με 32-bit floating point τύπους δεδομένων [1].

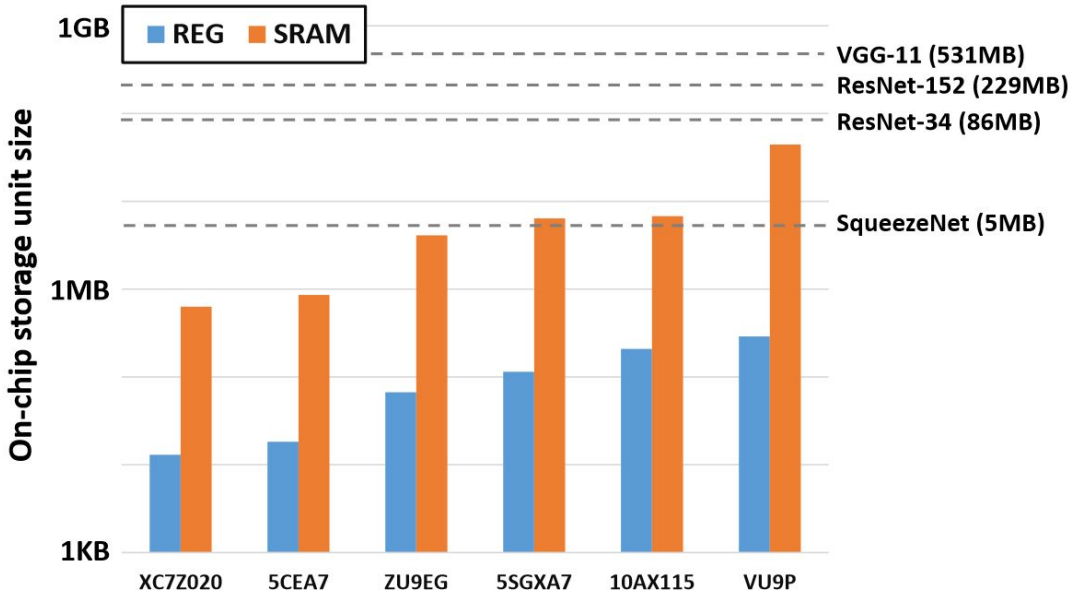


Fig. 20: Διάγραμμα διαφοράς διαθέσιμης μνήμης στο FPGA και απαιτούμενης από γνωστά μοντέλα βαθιών νευρωνικών δικτύων.

Αντιμετώπιση Περιορισμών FPGA

Όπως φαίνεται ξεκάθαρα, αυτό αποτελεί ένα bottleneck για την απόδοση και πρέπει να αντιμετωπιστεί. Γενικότερα βέβαια το θέμα των μειωμένων πόρων του FPGA αποτελεί ένα από τα μεγαλύτερα προβλήματα τους για την εύκολη και ταυτόχρονα αποδοτική σχεδίαση ενός μοντέλου. Για αυτό το λόγο όμως υπάρχουν πάρα πολλές αρχιτεκτονικές και βελτιστοποιήσεις προτεινόμενες στην βιβλιογραφία. Στα παρακάτω κεφάλαια θα εστιάσουμε σε αυτές τις αρχιτεκτονικές και τεχνικές βελτιστοποίησεις.
Παρακάτω αναφέρουμε επιγραμματικά τις διάφορες υλοποιήσεις και βελτιστοποιήσεις που συναντώνται στην βιβλιογραφία σχετικά με την επιτάχυνση των CNN στα FPGA:

1. Η μία αφορά την **αρχιτεκτονική** που θα υλοποιηθεί στο FPGA για το μοντέλο και προσπαθεί να αυξήσει την επαναχρησιμοποίηση της μνήμης και να κάνει τους απαραίτητους μετασχηματισμούς για την μέγιστη αξιοποίηση των πόρων του FPGA.
2. Η άλλη αντιμετώπιση έχει να κάνει με τον **προσεγγιστικό υπολογισμό** και προσπαθεί να μετασχηματίσει ή να μειώσει τα δεδομένα που χρειάζεται το μοντέλο χωρίς όμως να χάνει πολύ από την ακρίβεια πρόβλεψης του.
3. Υπάρχουν επίσης και οι αλγοριθμικοί τρόποι για την βελτίωση των συνελίξεων στα CNN, όπως οι μετασχηματισμοί **GEMM**, ο **Winograd** μετασχηματισμός καθώς και ο **Fast Fourier Transform** (FFT), με τους οποίους δεν θα ασχοληθούμε όμως περισσότερο καθώς αποτελούν γενικότερους τρόπους βελτιστοποίησεις της συνέλιξης και όχι συσχετιζόμενους μοναδικά με τα FPGA, παρόλο που προφανώς χρησιμοποιούνται και σε αυτά πολλές φορές.

Αρχιτεκτονικές για FPGA Επιταχυντές για CNN

Για να αντιμετωπιστούν τα προβλήματα που αντιμετωπίζουν τα FPGA που αναφέρθηκαν προηγουμένως, πρέπει να σχεδιαστούν καλές αρχιτεκτονικές που να αξιοποιούν τα χαρακτηριστικά των FPGA.

Processing Elements (PE) - Computation Units (CU)

Πολλές αρχιτεκτονικές της βιβλιογραφίας χρησιμοποιούν πολλαπλά Processing Elements (PE), τα οποία συναντώνται και με άλλες ονομασίες όπως: Computation Units (CU). Στην ουσία πρόκεινται, για βασικές μονάδες επεξεργασίας οι οποίες μπορούν να εκτελούν τις ίδιες πράξεις και επαναχρησιμοποιούνται συνεχώς. Σκοπός τους είναι να είναι όσο το δυνατόν μικρότερα, ώστε να χρησιμοποιηθούν περισσότερα από αυτά και να επιτευχθεί high peak performance. Πολλές φορές επιλέγεται και το PE να αντιστοιχεί σε ένα ή καποια CLB του FPGA για μέγιστη αξιοποίηση των πόρων του.
Ένα καλά σχεδιασμένο PE, μπορεί να αυξήσει την συχνότητα λειτουργίας όλου του συστήματος, επιτυγχάνοντας έτσι υψηλή απόδοση.

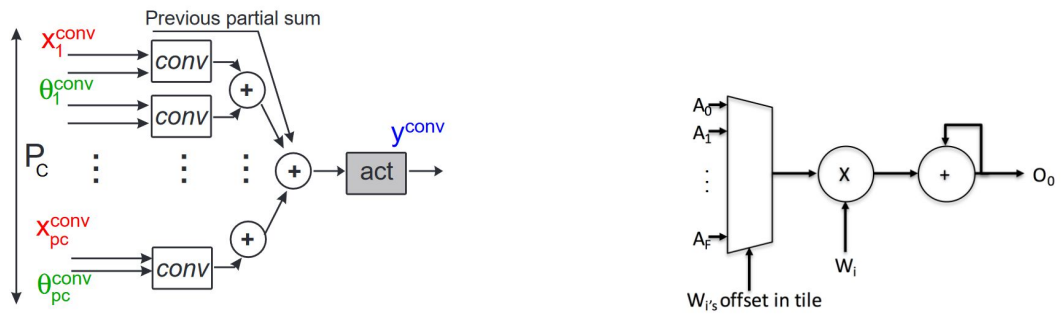


Fig. 21: Παραδείγματα δομής ενός PE

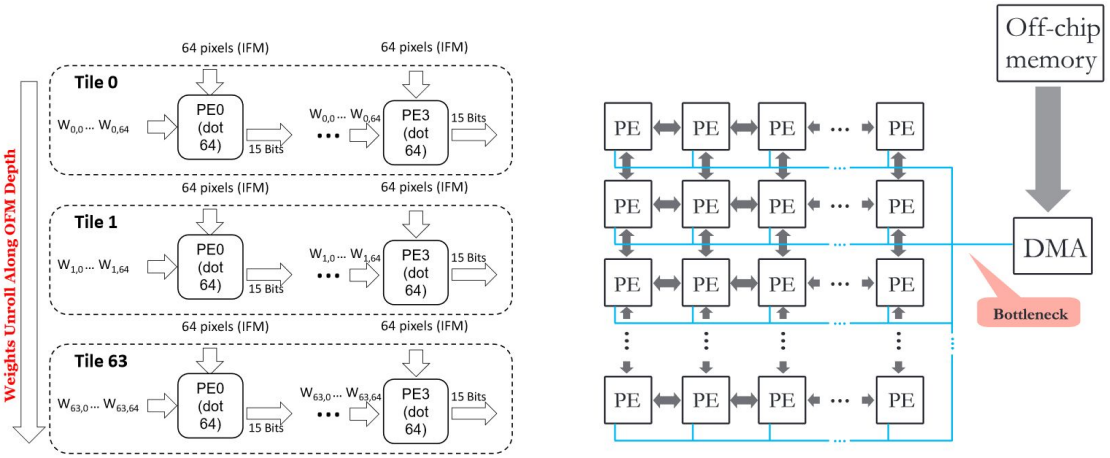


Fig. 22: Παραδείγματα αρχιτεκτονικών που κάνουν χρήση των PE

Systolic Arrays

Μια από τις αρχικές αρχιτεκτονικές στα FPGA, ήταν αυτό των Systolic Arrays, φτιαγμένες για την επιτάχυνση των 2D συνελίξεων στα Conv Layers [2]. Η τεχνική αυτή συνήθως αξιοποιεί τα PE τα οποία τοποθετούνται σε μια δισδιάστατη τοπολογία, όπου το κάθε ενώνεται με τα γειτονικά του και επιτρέπει έτσι στα δεδομένα να επαναχρησιμοποιηθούν περνώντας μέσα από το κάθε PE, μειώνοντας αντίστοιχα τις προσβάσεις στην μνήμη.

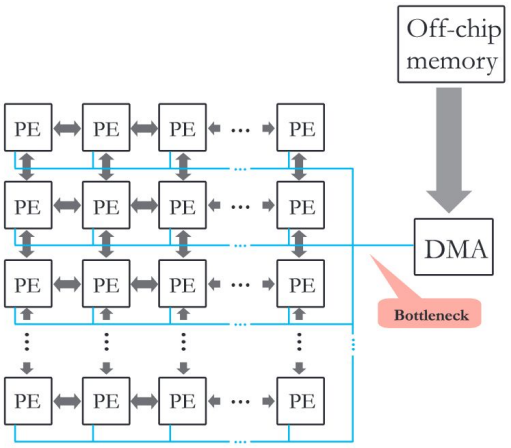


Fig. 23: Παράδειγμα τοπολογίας PE σε Systolic Array

Η τοπολογία αυτή συνήθως λειτουργεί και συντονίζεται από την CPU και τα δεδομένα μεταφέρονται στα PE μέσω κάποιου DMA. Η τοπολογία αυτή είναι ανεξάρτητη του CNN μοντέλου και τις παραμέτρους του.

Αυτού του είδους όμως οι τοπολογίες μπορούν να υποστηρίξουν συνελίξεις με φίλτρα μεγέθους μέχρι έναν αριθμό K_m (όπου εξαρτάται από την κάθε αρχιτεκτονική και για παράδειγμα μπορεί να είναι $K_m = 7$ ή και $K_m = 10$ όπως αναφέρεται και στην [2]). Επίσης αναφέρεται ότι σε συνελίξεις με φίλτρα μεγέθους K αρκετά μικρότερου του K_m ($K \ll K_m$), μόνο ένα κομμάτι της υπολογιστικής ισχύς της αρχιτεκτονικής αξιοποιείται.
Για παράδειγμα, σε τοπολογία με $K_m = 7$, και συνελίξεις με φίλτρα 3×3, αξιοποιείται μόνο το 9% των DSP Blocks.

Επίσης τα Systolic Arrays, δεν μπορούν να υλοποιήσουν τεχνικές caching των δεδομένων, οπότε είναι υποχρεωμένα να παίρνουν τις εισόδους τους από κάποια off-chip μνήμη. Οπότε, η απόδοση τους τελικά είναι αρκετά memory bound, και για αυτό τον λόγο εφαρμόζεται ελάχιστα πλέον.

SIMD Accelerators and Loop Optimizations

Η απάντηση στα μη αποδοτικά Systolic Arrays ήταν τα SIMD (Single Instruction - Multiple Data) Accelerators για τα CNN στα FPGA. Η γενική ροή δεδομένων και επεξεργασίας είναι αυτή που φαίνεται στην παρακάτω φωτογραφία, όπου τα Input FMs και τα Weights γίνονται fetched από την DRAM στους on-chip buffers (X^{conv} και Θ^{conv} αντίστοιχα). Τα δεδομένα αυτά περνάνε μετά στα PE και μετά την επεξεργασία τους σε αυτά, τα αποτελέσματα μεταφέρονται πίσω στους on-chip buffers (Y^{conv}) και αν χρειαστεί πίσω στην εξωτερική μνήμη, ώστε να χρησιμοποιηθούν από τα επόμενα layers [2].

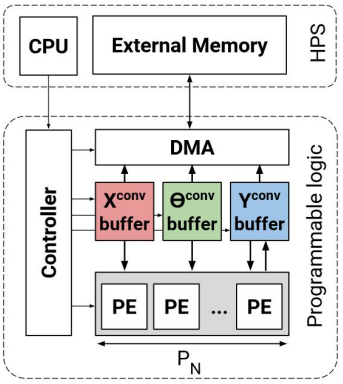


Fig. 24: Παράδειγμα ενός γενικού SIMD επιταχυντή

Το κάθε PE, είναι διαμορφώσιμο και μπορεί να έχει δικές του μονάδες επεξεργασίας όπως DSP Blocks καθώς και δικούς του on-chip registers για data caching σε PE επίπεδο.

Σε αυτό το design το κάθε PE είναι σχεδιασμένο για την υλοποίηση των πράξεων που απαιτούνται για την 3D συνέλιξη ενός Input FM με ένα φίλτρο σε ένα Conv Layer (2D Convolution, Sum of 2D Results, Activation).

Οπότε για τις N 3D συνελίξεις με τα N φίλτρα που απαιτούνται σε ένα Conv Layer χρειάζονται N PE.

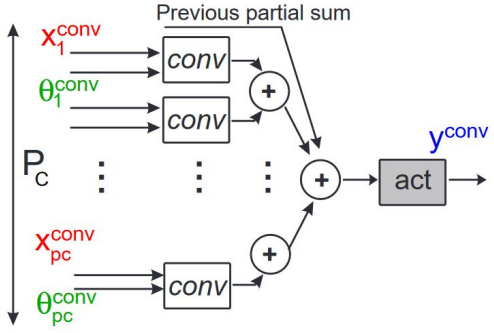


Fig. 25: Παράδειγμα δομής ενός PE

Οπότε με αυτήν την αρχιτεκτονική, πλέον το να υλοποιηθεί ο επιταχυντής για ένα CNN δίκτυο είναι απλά θέμα εύρεσης της βέλτιστης διαμόρφωσης των PE, καθώς και θέμα

βέλτιστου χρονοπρογραμματισμού της ροής των δεδομένων ώστε να έχουμε μέγιστο computational throughput.

Μερικά παραδείγματα παραμέτρων για εύρεση βέλτιστου design είναι:

- Αριθμός PE
- Αριθμός DSP Blocks σε κάθε PE
- Μέγεθος των data caches

Ειδικότερα για τα Convolutional Layers, η επεξεργασία των δεδομένων μπορεί να γραφτεί ως κώδικας με πολλαπλά και εμφωλευμένα loop, οπότε η εύρεση του βέλτιστου PE design, μπορεί να αναχθεί σε ένα **Loop Optimization Problem**. Μπορούμε δηλαδή να εφαρμόσουμε τεχνικές βελτιστοποίησεις των εμφωλευμένων loop, όπως:

- Loop Unrolling
- Loop Tiling
- Loop Interchange

Οπότε θέτωντας το κατάλληλο Unroll και Tiling Factor (P_i, T_i αντίστοιχα) καθορίζεται και ο αριθμός των PE, ο αριθμός των υπολογιστικών πόρων και on-chip μνήμης που χρειάζεται το κάθε PE καθώς και τα μεγέθη των on-chip buffers και το σύνολο των προσβάσεων στην DRAM εξωτερική μνήμη.

Παρακάτω παρατίθεται ένας πίνακας ο οποίος αντιστοιχίζει τις παράλληλίες των CNN που έχουμε αναφέρει με το αντίστοιχο factor P_i, T_i που γίνεται optimized:

Parallelism	Intra-layer	Inter-FM	Intra-FM		Inter-Convolution	Intra-Convolution	
Loop	L_L	L_N	L_V	L_U	L_C	L_J	L_K
Unroll factor	P_L	P_N	P_V	P_U	P_C	P_J	P_K
Tiling Factor	T_L	T_N	T_U	T_U	T_C	T_J	T_K

Table 1: Πίνακας Loop Optimization για P_i και T_i και την αντίστοιχη παραλληλία που αξιοποιούν

Περιγραφή Δομής του PE

Το κάθε PE, όπως αναφέραμε εκτελεί την 3D συνέλιξη ενός Block του Input FM (διαστάσεων του φίλτρου) με ένα φίλτρο, και η δομή του επιτρέπει την αξιοποίηση του Inter-Convolution Parallelism.

- Στο κάθε PE έχουμε C εισόδους (π.χ. $C = 3$ για RGB, $C = 4$ για CMYK) για το Input FM (X_{conv_i}) και για το Kernel-Filter (Θ_{conv_i}).
    - Για κάθε κανάλι C, γίνεται το 2D convolution του ενός block (διαστάσεων του kernel) του input FM με του Kernel-Filter για αυτό το Block του Input FM (διαστάσεων του φίλτρου) με ένα φίλτρο, και η δομή του επιτρέπει την αξιοποίηση του Inter-Convolution Parallelism.
    - Στο κάθε PE έχουμε C εισόδους για το Input FM (Xconvi) και για το Kernel-Filter ($conv_i$).
    - Για κάθε κανάλι C, γίνεται το 2D convolution του ενός block (διαστάσεων του kernel) του input FM με του Kernel-Filter για αυτό το κανάλι.

    - Ανά δύο προστίθενται τα αποτελέσματά του 2D convolution από το κάθε κανάλι C.
    - Το άθροισμα αυτό προστίθενται σε ένα partial sum, όπου θα προστεθούν τελικά όλα τα μερικά αθροίσματα από τα ανα δύο κανάλια.
    - Τελικά μετά το 2ο στάδιο αθροίσματος θα έχουμε το συνολικό άθροισμα όλων των C καναλιών, άρα και το αποτέλεσμα του 3D convolution που θα περάσει στην activation function.
- Αυτό γίνεται για κάθε block του input FM διαστάσεων του Kernel-Filter.
- Οπότε σε κάθε έξοδο των PE έχουμε το αποτέλεσμα ενός κελιού των Output FM.

Loop Unrolling

Κάνοντας Loop Unroll ενισχύουμε την παράλληλη εκτέλεση του του αλγορίθμου με αποτέλεσμα την επιτάχυνση του με κόστος όμως στους διαθέσιμους πόρους του FPGA.
Όπως φαίνεται και στον παραπάνω πίνακα, μπορεί να γίνει Loop Unrolling με factor P_i για οποιοδήποτε από τους παραλληλισμούς που έχουμε αναφέρει:

$$P_i \leq i,\ i \in \{L, N, V, U, C, J, K\}$$

Για έναν SIMD επιταχυντή που περιγράψαμε παραπάνω:

- Το Unrolling Factor P_N καθορίζει τον αριθμό των PE που χρειαζόμαστε.
- Τα Unrolling Factor P_C, P_K, P_J καθορίζουν των αριθμό των πολλαπλασιαστών, αθροιστών και το μέγεθος των registers που θα υπάρχουν μέσα σε κάθε PE.

Loop Tiling

Γενικα η on-chip μνήμη των FPGA δεν είναι αρκετά μεγάλη για να χωρέσει όλα τα βάρη και τα ενδιάμεσα FMs από όλα τα layers του νευρωνικού. Οπότε πολλές φορές πρέπει τα δεδομένα να αποθηκεύονται στην εξωτερική DRAM που όπως είδαμε είναι αρκετά κοστοβόρο από θέμα καθυστέρησης και ενεργειακής απόδοσης.
Οπότε πρέπει τα δεδομένα πρέπει να αποθηκεύονται ιδανικά μόνο στους on-chip buffers και τους registers, ώστε κάθε δεδομένα που έρχεται από την DRAM να επαναχρησιμοποιείται όσο το δυνατόν γίνεται περισσότερο.

Το Loop Tiling, μπορεί να βοηθήσει σε αυτό καθώς χωρίζουμε τα FMs και τα βάρη του κάθε Conv Layer σε Tiles τα οποία θα χωράνε στους on-chip buffers. Οπότε όλη η επεξεργασία θα γίνεται πάνω στα on-chip δεδομένα μειώνοντας την εξάρτηση από το bandwidth των μνημών. Το μέγεθος του Tile, εξαρτάται από το μέγεθος της on-chip μνήμης και καθορίζει αντίστοιχα το πόσα memory accesses θα γίνουν. Δηλαδή ένα μεγάλο Tile απαιτεί μεγαλύτερη on-chip μνήμη, αλλά ελαχιστοποιεί τις φορές που θα ζητήσουμε δεδομένα από την εξωτερική μνήμη.
Επίσης γειτονικά Tile, μπορούν να μοιράζονται μέρος των δεδομένων, όπως σε ένα Conv Layer όπου γειτονικά Tile μπορούν να μοιράζονται το Input FM ή τα βάρη.

Για τον SIMD επιταχυντή που περιγράψαμε παραπάνω, τα μεγέθη των buffers που περιέχουν τα Input FM, τα βάρη και τα Output FM καθορίζονται από τα Tiling Factor που φαίνονται στον πίνακα παραπάνω, σύμφωνα με την παρακάτω εξίσωση:

$$M_{conv} = T_C T_H T_W + T_N T_C T_J T_K + T_N T_V T_U$$

Ένα παράδειγμα σε High Level κώδικα για το πως θα γίνει το Loop Tiling με όλα τα παραπάνω Tiling Factor είναι το εξής:

```
// Ll: Layer
for (int l=0;l<L,l++){
// Lb : Batch
for (int b=0;b<B,l++){
// Ln: Y Depth
for (int n=0;n<N;n++){
// Lv: Y Columns
for (int v=0;v<V,v++){
// Lu: Y Raws
for (int u=0;u<U,u++){
// Lc: X Depth
for (int c=0;n<C;c++){
// Lj: Theta Columns
for (int j=0;j<J,j++){
// Lk: Theta Raws
for (int k=0;k<K,k++){
  Y[b,l,n,v,u] += X[b,l,c,v+j,u+k] *
      Theta[l,n,c,j,k]
}}}}}}}
```

(a)

```
for (int n=0;n<N;n+=Tn){
for (int v=0;v<V,v+=Tv){
for (int u=0;u<U,u+=Tu){
for (int c=0;n<C;c+=Tc){
  // DRAM: Load in on-chip buffers the tiles:
  // X[l,c:c+Tc,v:v+Tv,u:u+Tu]
  // Theta [l,n:n+Tn,c:c+Tc,j,k]
  // Process on-chip tiles
  for (int tn=0;tn<Tn;tn++){
  for (int tv=0;tv<Tv,tv++){
  for (int tu=0;tu<Tu,tu++){
  for (int tc=0;tn<Tc;tc++){
  for (int j=0;j<J,j++){
  for (int k=0;k<K,k++){
    Y[l,tn,tv,tu] += X[l,tc,tv+j,tu+k] *
        Theta[l,tn,tc,j,k];
  }}}}}}
  // DRAM: Store output tile
}}}}
```

(b)

Fig. 26: Loop Tiling στα Conv Layers: (a) Πριν το tiling (b) Μετά το tiling

Και αντίστοιχα φαίνεται το Loop Tiling με όλα τα παραπάνω Tiling Factor, εποπτικά στα Input FM, βάρη και Output FM:

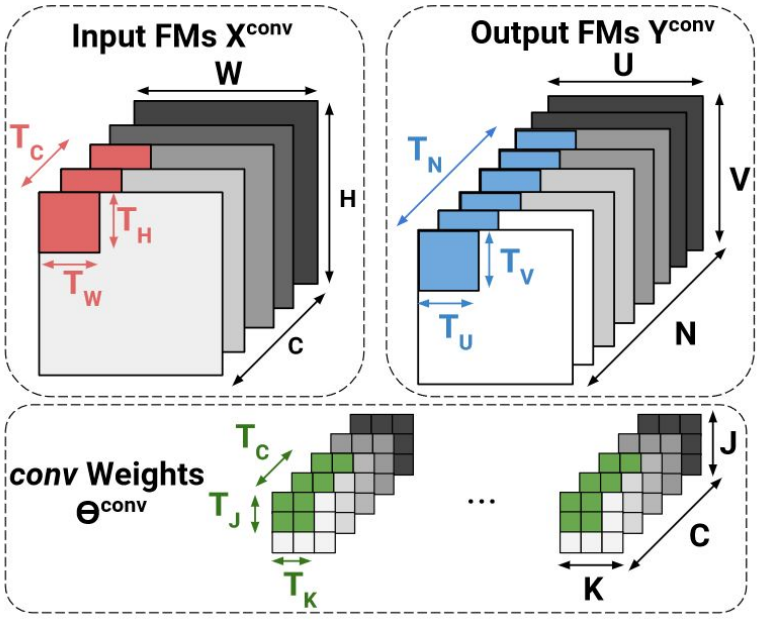


Fig. 27: Loop Tiling στα Conv

Προσεγγιστικος Υπολογισμός

Approximate Computing

Πέρα από τους υπολογιστικούς μετασχηματισμούς και τις βελτιστοποιήσεις στην ροή των δεδομένων (data-path), προκειμένου να πετύχουμε καλύτερη επιτάχυνση του inference των CNN εφαρμόζουμε προσεγγιστικό υπολογισμό. Πιο συγκεκριμένα, επιλέγουμε να ανταλλάξουμε ένα μέρος από την ακρίβεια για πετύχουμε μεγαλύτερες ταχύτητες. Υπάρχουν δύο μέθοδοι. Η πρώτη κάνει αριθμητικές προσεγγίσεις μειώνοντας την ορθότητα (precision) ενώ η δεύτερη στοχεύει στο να μειώσει το πλήθος των πράξεων που συμβαίνουν.

Αριθμητική προσέγγιση

Έρευνες έχουν δείξει [2] ότι είναι δυνατόν να μειώσουμε το precision στους τελεστές και τους τελεστέους των επιπέδων των CNN χωρίς σημαντικές αποκλίσεις στο accuracy. Λέμε αυτή την τεχνική κβαντισμό (Quantization) και μπορεί να εφαρμοστεί στα εξής:

Στην είσοδο **Στα βάρη** **Στα FMs**

Floating Point

Αναφέρουμε αρχικά την αναπαράσταση των κλασσικών Floating Points για να γίνει πιο ξεκάθαρη μετά η αναπαράσταση των Fixed Points.
Οι floating point αριθμοί αποτελούνται από ένα πρόσημο (sign) του ενός bit, έναν exponent και τη mantissa. Ο exponent είναι υπεύθυνος για την αναπαράσταση μεγαλύτερου εύρους πραγματικών αριθμών και η mantissa για την καλύτερη ακρίβεια τους [12].

Το πλήθος των bits που δίνεται σε κάθε μέρος των floating point φαίνεται στον παρακάτω πίνακα, ο οποίος δείχνει για 3 διαφορετικά format Floating Point:

Format	Total bit-width	Exponent bit-width	Mantissa bit-width
Double Floating Points	64	11	52
Single Floating Point	32	8	23
Half Floating Point	16	5	10

Table 2: Πίνακας του του layout και πλήθος bits για το κάθε component των Floating Point 16, 32, 64bit

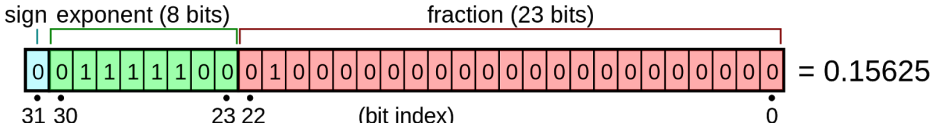


Fig. 28: Floating Point 32 bit - Παράδειγμα του layout και πλήθος bits για το κάθε component

Μπορούμε να υπολογίσουμε την τιμή ενός floating point αριθμού χρησιμοποιώντας την παρακάτω εξίσωση:

$$value = (-1)^{sign} \times (1 + \frac{mantissa}{2^{23}}) \times 2^{(exponent-127)}$$

Fixed Point αριθμητική

Ο σύνηθες τύπος για αντιπροσώπευση των δεδομένων είναι οι Floating-Point αριθμοί (32 bit). Αν και πετυχαίνουν μεγάλη ακρίβεια, οι πράξεις με αυτούς του τύπους δεδομένων είναι υπολογιστικά ακριβές.

Static Fixed Point

Οι fixed point έχουν το ίδιο πλήθος bits (bit-width) το οποίο ορίζει το επιθυμητό εύρος αριθμών και precision. Πιο συγκεκριμένα χρησιμοποιούνται οι Static Fixed Point (SFP), οι οποίοι έχουν καθορισμένο μήκος από bits για την mantissa και μοιράζονται τον ίδιο exponent. Ο exponent καθορίζει το εύρος και η mantissa το precision. Παρακάτω φαίνεται μια αναπαράσταση για τους floating και fixed point αριθμούς:

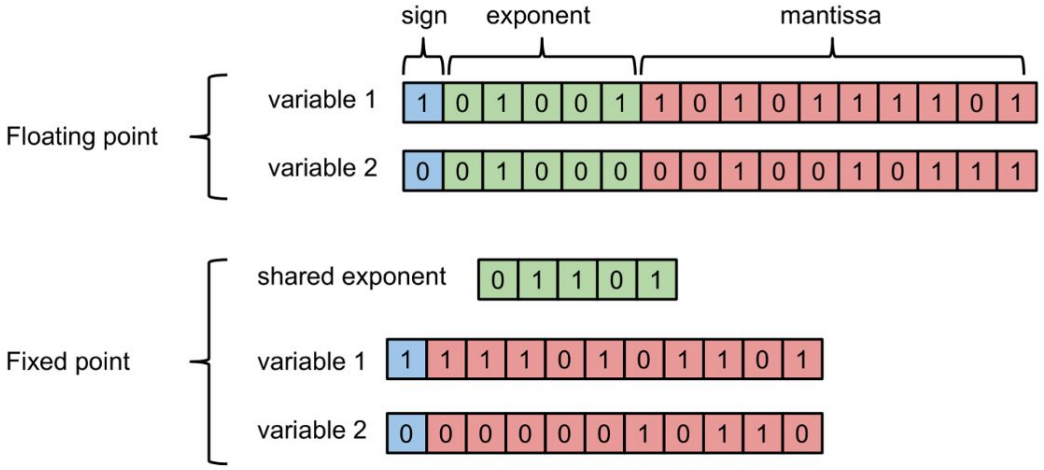


Fig. 29: Floating και Fixed Point Layout

Συγκρίνοντας με τους FP οι SFP πετυχαίνουν καλύτερη αξιοποίηση του υλικού και μικρότερη κατανάλωση ενέργειας. Για παράδειγμα, σε ένα DSP αντί για ένα πολλαπλασιασμό 32-bit FP μπορούμε να εκτελέσουμε 2 $18 \times 19\ bits$, έχοντας μεγαλύτερο throughput [2].

Dynamic Fixed Point

Σε βαθιά νευρωνικά δίκτυα πολλά επίπεδα έχουν διαφορετικές απαιτήσεις από το αριθμητικό εύρος των δεδομένων. Αυτό παρατηρείται στα FMs όπου όσο προχωράμε βαθύτερα τείνουν να αποτελούνται από αριθμούς μεγαλύτερου εύρους από ότι τα βάρη. Όπως γίνεται εμφανές ο τύπος SFM είναι ακατάλληλος για τέτοιες τοπολογίες, καθώς το εύρος των που μπορεί να αναπαραστήσει είναι σταθερό.

Μία λύση σ' αυτό το πρόβλημα είναι να χρησιμοποιήσουμε Dynamic Fixed Point (DFP) τύπους δεδομένων. Ο DFP επιτρέπει για κάθε επίπεδο του νευρωνικού, τα input και output FMs και τα βάρη να έχουν διαφορετικά διαμοιραζόμενα exponent. Αυτό επιτρέπει κάθε κομμάτι του δικτύου να αναπαρίσταται όπως χρειάζεται, προσφέροντας ευελιξία και ταυτόχρονα διατηρώντας υψηλό throughput, λόγω μείωσης του precision. Προκειμένου να βρεθεί η βέλτιστη αναπαρασταση συνήθως χρησιμοποιείται κάποια ευριστική ή γίνεται εξαντλητική αναζήτηση.

Ακόμα ένα όφελος που έχουμε από την μείωση των bit που χρησιμοποιούμε για τα δεδομένα είναι ότι καταναλώνουν μικρότερο μέγεθος την τοπικής μνήμης. Αυτό μας επιτρέπει να έχουμε περισσότερα τοπικα δεδομένα, ενισχύοντας τις δυνατότητες επαναχρησιμοποίησης του και μειώνοντας τις δαπανηρές, από μεριάς χρόνου, προσβάσεις στην εξωτερική μνήμη.

Binary Representation

Η δυαδική αναπαράσταση αποτελεί την ακραία περίπτωση της αριθμητικής προσέγγισης. Τα βάρη ή τα FMs ή και τα δύο αναπαριστούμε δυαδικά, όπως είναι αναμενόμενο πετυχαίνουμε πολύ μεγάλη επιτάχυνση. Εργασίες πάνω στο BinaryConnect [13] δείχνουν ότι με χρήση δυαδικών βάρων στο δίκτυο ImageNet, οι απαιτήσεις του bandwidth μειώθηκαν κατά 3200% ενώ η ακρίβεια κατά 19,2%. Στην ίδια έρευνα χρησιμοποιούν δυαδικούς αριθμούς τόσο για τα βάρη όσο και για τα FMs και παρατηρήθηκε 29,8% μικρότερη ακρίβεια [2]. Είναι εμφανές ότι οι παραπάνω βελτιστοποιήσεις έχουν ως τίμημα την μειωμένη ακρίβεια. Παρόλα αυτά, στην [5] χρησιμοποιείται Binary NN και 8 bit για τα activations, με μόλις 1% απόκλιση από την αρχική ακρίβεια με floating points.

Με την χρήση δυαδικών αριθμών για τα βάρυ και τα FMs, η συνέλιξη τους μπορεί να γίνει με ένα XNOR layer ακολουθούμενο από ένα μετρητή. Αρχικά γίνεται η λογική πράξη XNOR μεταξύ της εισόδου και του αντίστοιχου βάρους και το παραγόμενο αποτέλεσμα (0 ή 1) αυξάνει, στην περίπτωση του 1, ένα μετρητή. Επειδή η πράξη της συνέλιξης απλοποιήθηκε αρκετά δεν χρειάζεται να γίνεται μέσω των DSPs, αλλά μπορεί να πραγματοποιηθεί μέσω του προγραμματιζόμενου υλικού, προσφέροντας ακόμα μεγαλύτερη ταχύτητα.

Η παραπάνω διαδικασία φαίνεται από την ακόλουθη εικόνα.

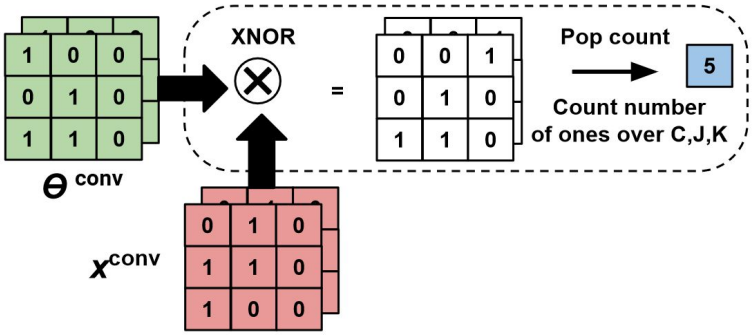


Fig. 30: Συνέλιξη δυαδικού πίνακα με δυαδικό φίλτρο με χρήση XNOR operator και pop counter.

Ternary Representation

Μία άλλη αναπαράσταση που δεν είναι τόσο αφαιρετική όσο η δυαδική, είναι μια τριαδική αναπαράσταση των βαρών. Συνήθως συνδυάζεται με κάποια μεγαλύτερη αναπαράσταση για τα activations. Αφού χρησιμοποιούνται ternary βάρη (-1, 0, +1) προσφέρεται η δυνατότητα, όλα οι MAC (Multiply-and-Accumulate) πράξεις να αντικατασταθούν πλέον από αθροίσματα. Στην [4] χρησιμοποιείται μια αναπαράσταση της μορφής INT-8-2 (όπου 8-bit για activation και 2-bit για τα βάρη) και φτάνει σε επιδόσεις πολύ μεγαλύτερες αυτών των CPU και GPU αγγίζοντας νούμερα των ASIC (TPU).

Low Bit-width Implementations (Logic vs. DSP)

Hardware Resources

Το μέγεθος ενός PE ή CU εξαρτάται άμεσα από το bit-width που θα επιλεχθεί για τα δεδομένα. Στην [1] επιλέγονται για σύγκριση 3 διαφορετικές υλοποιήσεις για διάφορα μεγέθη Floating και Fixed Point αριθμών.
Οι 3 διαφορετικές υλοποιήσεις βασίζονται σε:

1. Υλοποίηση πολλαπλασιασμών-αθροίσεων, χρησιμοποιώντας λογικούς πόρους (LUT, FF) σε Xilinx FPGA.
2. Υλοποίηση πολλαπλασιασμών-αθροίσεων, χρησιμοποιώντας DSP units σε Xilinx FPGA.
3. Υλοποίηση πολλαπλασιασμών-αθροίσεων, χρησιμοποιώντας DSP units σε Altera FPGA.

Παρακάτω φαίνεται ο συγκριτικός πίνακας για την χρήση των πόρων του FPGA για τις διάφορες υλοποιήσεις με διάφορους τύπους δεδομένων:

	Xilinx Logic				Xilinx DSP			Altera DSP	
	multiplier		adder		multiply & add			multiply & add	
	LUT	FF	LUT	FF	LUT	FF	DSP	ALM	DSP
fp32	708	858	430	749	800	1284	2	1	1
fp16	221	303	211	337	451	686	1	213	1
fixed32	1112	1143	32	32	111	64	4	64	3
fixed16	289	301	16	16	0	0	1	0	1
fixed8	75	80	8	8	0	0	1	0	1
fixed4	17	20	4	4	0	0	1	0	1

Table 3: Συγκριτικός πίνακας χρήσης πόρων του FPGA για 2 διαφορετικές υλοποιήσεις multiply-add κυκλωμάτων με διάφορους τύπους δεδομένων

Αρχικά για την Logic-only υλοποίηση, βλέπουμε ότι συμπιέζοντας τα βάρη και τα activations από 32-bit floating-point αριθμούς σε 8-bit fixed point, τα resources πέφτουν περίπου στο 1/10 για τους πολλαπλασιαστές και 1/50 για τους αθροιστές. Χρησιμοποιώντας 4-bit αναπαράσταση (και χαμηλότερα) μπορούμε να έχουμε ακόμα μεγαλύτερη μείωση χρήσης πόρων, αλλά πλέον έχουμε μεγάλη μείωση της ακρίβειας πρόβλεψης.

Για την υλοποίηση με DSP units, έχουμε αποτελέσματα και για τις δύο πλατφόρμες της Xilinx και της Altera. Πάλι σε σύγκριση με την 32-bit floating-point αναπαράσταση, η χαμηλότερου bit-width αναπαραστήσεις έχουν αρκετή μείωση των resources, όμως στο Altera FPGA δεν είναι ιδιαίτερα εμφανής διότι τα DSP units του, παρέχουν native υποστήριξη για πράξεις με floating points.
Έπειτα φαίνεται ότι για οποιοδήποτε μήκος αναπαράστασης κάτω των 16-bit χρησιμοποιείται μόνο ένα 1 DSP unit και στις δύο πλατφόρμες, οπότε δεν υπάρχει καμία βελτίωση στην μείωση του hardware άμα πάμε κάτω σε αναπαραστάσεις κάτω των 16-bit.

Το πρόβλημα είναι ότι οι wide πολλαπλασιαστές και αθροιστές αυτών των DSP units, σε αυτές τις περιπτώσεις υπολειτουργούν, μη αξιοποιώντας τις πλήρης δυνατότητες τους. Μία λύση σε αυτό που προτείνεται στο [14] είναι η χρήση ενός DSP unit για την υλοποίηση δύο ανεξάρτητων πολλαπλασιασμών χαμηλής αναπαράστασης.

Οπότε, οι δύο πολλαπλασιασμοί $A \times C$ και $B \times C$, που έχουν ένα κοινό τελεστή μπορούν να εκτελεστούν όπως παρακάτω:

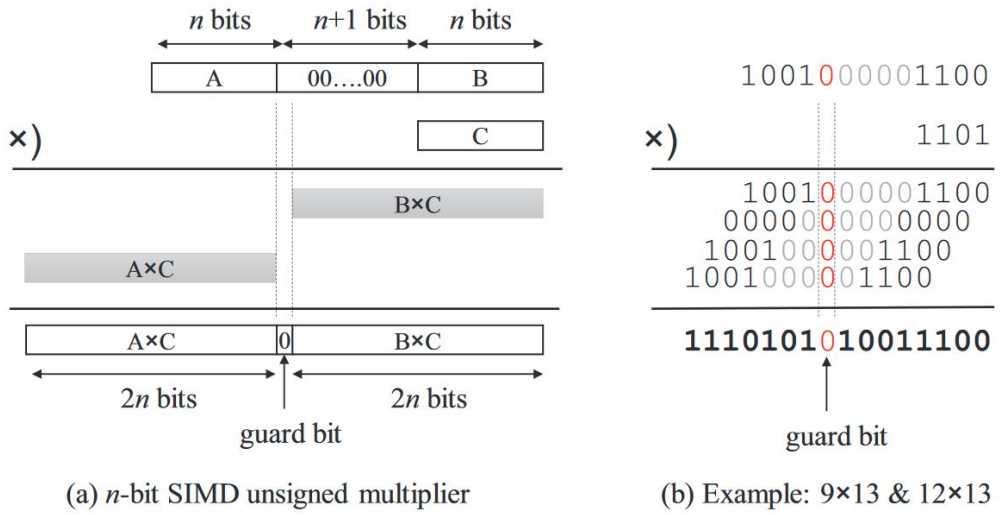


Fig. 31: (a) n-bit SIMD unsigned πολλαπλασιαστής (b) Παράδειγμα πολλαπλασιασμού 9x13 και 12x13 bit

Θα πρέπει ο τελεστής, C που πολλαπλασιάζουμε τις εισόδους A και B να είναι ο ίδιος και η αναπαράσταση των A και B να είναι όπως φαίνεται στην εικόνα (a). Οφείλουν να έχουν $n+1$ bits κενό μεταξύ τους ώστε τα ενδιάμεσα πηλίκα των πράξεων $A \times C$ και $B \times C$ να αθροίζονται ξεχωριστά, όπως φαίνεται στην εικόνα (b).

Για να μπορέσουν να γίνουν ταυτόχρονα δύο n-bit πολλαπλασιασμοί χρειάζεται ένας $(3n+1)\times n$ πολλαπλασιαστής.

Για παράδειγμα:

- Με ένα DSP unit με 25×18 πολλαπλασιαστή
- Μπορούν να γίνουν στο ίδιο unit, δύο 8-bit πολλαπλασιασμοί

Performance Results

Στην [4] χρησιμοποιείται μια αναπαράσταση της μορφής INT-8-2 (όπου **8-bit για activation και 2-bit για τα βάρη**) το οποίο είναι μια αναπαράσταση που δεν μπορεί να υποστηριχθεί natively από άλλες CPU, GPU ή και ASIC προς το παρών. Με αυτήν την αναπαράσταση φτάνει σε επιδόσεις **5 AI-TOPS** για το Arria 10 FPGA και **76 AI-TOPS στα 0.7 TOPS/W** για το Stratix 10 FPGA.

Αφού χρησιμοποιούνται ternary βάρη (-1, 0, +1) προσφέρεται η δυνατότητα, όλες οι MAC (Multiply-and-Accumulate) πράξεις να αντικατασταθούν πλέον από αθροίσματα. Αυτό επιτρέπει να χρησιμοποιηθούν τα λογικά block των FPGA (CLB ή ALM) για τις MAC πράξεις από ότι τα DSP που χρησιμοποιούνται συχνά. Αυτό είναι ιδιαίτερα κρίσιμο, διότι οι επιδόσεις των DSP είναι η ίδια ή και χειρότερη με αυτή των DSP σε μία CPU ή GPU. Επίσης η πλειονότητα της αρχιτεκτονικής του FPGA περιέχει CLB ή ALM, και όχι DSP.

Όπως είδαμε αυτή η υλοποίηση προσφέρει από τις καλύτερες επιδόσεις που έχουμε για FPGA, οπότε η διερεύνηση αρχιτεκτονικών που αξιοποιούν τα Logic Blocks του FPGA αντί των DSP σίγουρα προσφέρει την δυνατότητα για μεγαλύτερες επιδόσεις.

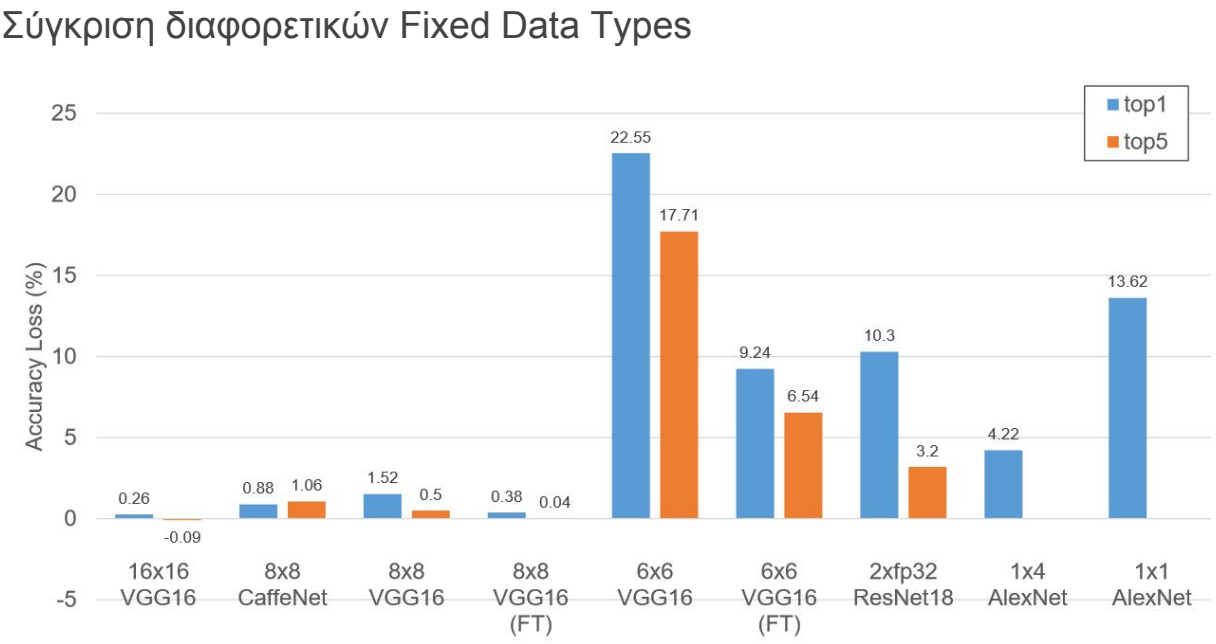


Fig. 32: Συγκριτικό διάγραμμα μεταξύ διαφορετικών τρόπων κβάντισης των δεδομένων. Το κάθε configuration συμβολίζει (weight bit-width)x(activation bit-width)

Μείωση πράξεων στα CNNs

Για υλοποιήσεις πάνω σε FPGA υπάρχουν δύο κύριες μέθοδοι, η πρώτη είναι η μείωση των βαρών (weight pruning) και η δεύτερη είναι τα low rank φίλτρα, για να πετύχουμε λιγότερους πολλαπλασιασμούς.

Weight Pruning

Τα φίλτρα των CNNs έχουν πολλές παραμέτρους και κάποιες από αυτές μπορούν να αφαιρεθούν χωρίς να υπάρχει μεγάλη απόκλιση στην ακρίβεια. Μια μέθοδος που χρησιμοποιείται για pruning είναι τα βάρη που έχουν τιμές πολύ κοντά στο 0 να παίρνουν την τιμή 0. Μια πιο σύγχρονη προσέγγιση, για τις ανάγκες μικρότερης κατανάλωσης ενέργειας, είναι η αφαίρεση των βαρών. Είναι σημαντικό τα βάρη που θα απομείνουν απο τις παραπάνω μεθόδους να επεξεργαστούν περαιτέρω (fine-tuning) για να ελαχιστοποιηθεί το περιθώριο λάθους στην κατηγοριοποίηση. Από την [3] βλέπουμε ότι για μείωση κατά 53% και 85% στα βάρη των Convolution και FC επίπεδων έχουμε πτώση μόνο κατά 0,5% της ακρίβειας.

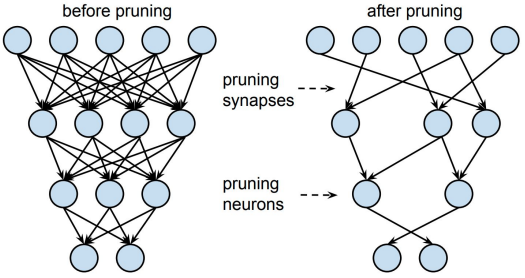


Fig. 33: Αναπαράσταση νευρώνων ενός δικτύου: (a) Πριν το Weight (b) Μετά το Weight Pruning:

Low Rank Approximation

Ένας εναλλακτικός τρόπος για να μειωθούν οι πράξεις είναι με την χρήση seperatable filters. Έστω ένα τέτοιο 2D φίλτρο A με διαστάσεις $n \times m$, το οποίο μπορεί να αναπαρασταθεί απο τα φιλτρα B και Γ με διαστάσεις $n \times 1 \text{ και } 1 \times m$ αντίστοιχα, $\mathrm{A} = \mathrm{B} \times \Gamma$.

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Fig. 34: Παράδειγμα Sobel φίλτρου 3x3 διασπασμένο σε γινόμενο ενός 3x1 και 1x3 πινάκων

Έχει αποδειχθεί ότι η πράξη της συνέλιξης πάνω σε μία εικόνα μεγέθους $\mathrm{N} \times \mathrm{M}$, με το αρχικό πίνακα A έχει πολυπλοκότητα $O(N \times M \times n \times m)$ ενώ με την χρήση των διασπασμένων πινάκων B και Γ έχει $O(N \times M \times (n + m))$ (για ανάλυση της πολυπλοκότητας, για κάθε περίπτωση βλέπε Appendix A). Οπότε υπάρχει αλγοριθμική μείωση της πολυπλοκότητας, αντικαθιστώντας έναν επιπλέον πολλαπλασιασμό της τάξης των μεγεθών του φίλτρου με ένα άθροισμα αντίστοιχα των μεγεθών του φίλτρου.
Πέρα από την παραπάνω βελτιστοποίησης του χρόνου εκτέλεσης της συνέλιξης, οι διασπασμένοι πίνακες απαιτούν λιγότερη μνήμη, επιτρέποντας καλύτερη αξιοποίηση της τοπικής μνήμης.

Αποτελέσματα Υλοποιήσεων

Σε αυτήν την ενότητα παρουσιάζουμε μερικές υλοποιήσεις που εφαρμόζουν τις τεχνικές που παρουσιάσαμε στις προηγούμενες ενότητες και τις μετρικές τους.

Μετρικές

Οι μετρικές με τις οποίες αξιολογούμε τις υλοποιήσεις είναι οι εξής:

- Ακρίβεια πρόβλεψης του μοντέλου (Accuracy)
- Υπολογιστική Δύναμη (Computational Throughput)
    - Μετριέται σε GOP/s (Giga-Operations per second) ή TOP/s (Tera-Operations per second)
- Κατανάλωση Ενέργειας (Power)
    - Μετριέται σε W
- Υπολογιστική Δύναμη προς Κατανάλωση Ενέργειας (Comp. Through. / Power)
    - Μετριέται σε $GOP/s/W$ ή $TOP/s/W$

Όπως θα γίνει κατανοητό, στις υλοποιήσεις στα FPGA είναι σημαντικός παράγοντας η κατανάλωση ενέργειας και δίνεται έμφαση σε αυτήν την μετρική όπως και στον λόγο υπολογιστικής δύναμης προς την κατανάλωση ενέργειας, καθώς μας επιτρέπεται να συγκρίνουμε μοντέλα διαφορετικών υπολογιστικών δυνατοτήτων καθαρά ως προς την ενεργειακή τους απόδοση.

Υλοποιήσεις με Loop Optimizations

Θα εξετάσουμε αρχικά κάποιες υλοποιήσεις που παρουσιάζονται συγκριτικά στον Πίνακα 4 και προέρχεται από την δουλειά του [2]. Αυτές οι υλοποίησεις εφαρμόζουν κατά κύριο λόγο Loop Optimization τεχνικές, αλλά κάποιες εφαρμόζουν και άλλες τεχνικές, όπως μειωμένη αναπαράσταση με Fixed Points. Σε αυτές τις υλοποιήσεις δεν εστιάζουμε πάντα στην μετρική της ακρίβειας καθώς οι περισσότερες υλοποιήσεις δεν εφαρμόζουν υπερβολικό προσεγγιστικό υπολογισμό.

Σημείωση: *Η αρίθμηση των υλοποιήσεων αναφέρεται στον αριθμό του reference που έχει η [2] για κάθε paper, οπότε ο αναγνώστης μπορεί να ανατρέξει στα References της [2] με τον αριθμό της υλοποίησης για περισσότερες πληροφορίες.*

	Network	Network Workload		Bitwidth	Desc.	Device	Freq	Through	Power	LUT	DSP	Memory
		Comp. (GOP)	Param. (M)				(MHz)	(GOPs)	(W)	(K)		(MB)
[39]	AlexNet-C	1.3	2.3	Float 32	HLS	Virtex7 VX485T	100	61.62	18.61	186	2240	18.4
[9]	VGG16SVD-F	30.8	50.2	Fixed 16	HDL	Zynq Z7045	150	136.97	9.63	183	780	17.5
[28]	AlexNet-C	1.3	2.3	Fixed 16	OpenCL	Stratix5 GSD8	120	187.24	33.93	138	635	18.2
	AlexNet-F	1.4	61.0					71.64		272	752	30.1
	VGG16-F	31.1	138.0					117.9		524	1963	51.4
[77]	AlexNet-C	1.3	2.3	Float 32	HLS	Virtex7 VX485T	100	75.16		28	2695	19.5
[65]	AlexNet-F	1.4	61.0	Fixed 16	HLS	Virtex7 VX690T	150	825.6	126.00		14400	
	VGG16-F	31.1	138.0					1280.3	160.00		21600	
[78]	NIN-F	2.2	61.0	Fixed 16	HDL	Stratix5 GXA7	100	114.5	19.50	224	256	46.6
	AlexNet-F	1.5	7.6					134.1	19.10	242	256	31.0
[36]	AlexNet-F	1.4	61.0	Fixed 16		Virtex7 VX690T	156	565.94	30.20	274	2144	34.8
[79]	AlexNet-C	1.3	2.3	Float 32	HLS	Virtex7 VX690T	100	61.62		273	2401	20.2
[80]	VGG16-F	31.1	138.0	Fixed 16	HDL	Arria10 GX1150	150	645.25	50.00	322	1518	38.0
[43]	AlexNet-F	1.4	61.0	Fixed 16	OpenCL	Arria10 GT1150	239.6	360.4		700	1290	47.2
	VGG-F	31.1	138.0				221.65	460.5		708	1340	49.3
	VGG-F	31.1	138.0				231.85	1171.3		626	1500	33.4
[42]	AlexNet-C	1.3	2.3	Fixed 16	HDL	Cyclone5 SEM	100	12.11		22	28	0.2
		1.3	2.3			Virtex7 VX485T	100	445			2800	
[84]	NiN	20.2	7.6	Fixed 16	HDL	Stratix5 GXA7	150	282.67		453	256	30.2
	VGG16-F	31.1	138.0					352.24		424	256	44.0
	ResNet-50	7.8	25.5					250.75		347	256	39.3
	NiN	20.2	7.6			Arria10 GX1150	200	587.63		320	1518	30.4
	VGG16-F	31.1	138.0					720.15		263	1518	44.5
	ResNet-50	7.8	25.5					619.13		437	1518	38.5
[85]	AlexNet-F	1.5	7.6	Float 32		Virtex7 VX690T	100	445.6	24.80	207	2872	37
	VGG16SVD-F	30.8	50.2					473.4	25.60	224	2950	47

Table 4: FPGA-based CNN επιταχυντές υλοποιώντας Loop Optimizations

Loop Unroll και Tilling στα LC, LN Loops

Αυτές οι αρχιτεκτονικές κάνουν Loop Unroll στα L_C , L_N loops, οπότε εκμεταλλεύονται κυρίως τις Inter-FM και Inter-Convolution παραλληλίες.

Αρχικά η πρώτη υλοποίηση που αξιοποίηση τεχνικές Loop Unrolling είναι η [39] η οποία φτάνει επιδόσεις των **61.62 GOPs** στο AlexNet με κατανάλωση **18.61 W**. Αυτές είναι πολύ ικανοποιητικές επιδόσεις, όμως δεν είναι βέλτιστες καθώς είχε υλοποιηθεί με HLS εργαλεία και χρησιμοποιεί 32 bit floating points.

Η δουλειά στην [78] χρησιμοποιεί παρόμοιες τεχνικές Unrolling τις οποίες εφαρμόζει όχι μόνο στα Conv Layers αλλά και στα Fully Connected Layers. Επίσης έχει υλοποιηθεί με RTL σχεδίαση και χρησιμοποιεί 16 bits Fixed Point. Για αυτό και καταφέρνει και σχεδόν 2-πλάσια βελτίωση στην υπολογιστική επίδοση (**134.1 GOPs**) με σχεδόν την ίδια κατανάλωση (**19.1 W**).

Η δουλειά στην [65] αποτελεί μια ανανέωση της δουλειά της [39], από τους ίδιους τους συγγραφείς όπου πάλι χρησιμοποιώντας τις ίδιες τεχνικές Unrolling και Tiling και χρησιμοποιώντας HLS εργαλεία, κατάφεραν παραπάνω από 10-πλάσια βελτίωση στην υπολογιστική επίδοση (**825.6 GOPs**) στο AlexNet με κατανάλωση **126 W**. Αυτό επιτεύχθει χρησιμοποιώντας ένα Deeply Pipelined cluster 4 Virtex 7-VC709 FPGA ελεγχόμενα από ένα 1 Zynq ZC706 FPGA και 16 bits Fixed Point αναπαράσταση αυτή τη φορά.

Για μία πιο αναλυτική σύγκριση αυτής της υλοποίησης με μία αντίστοιχη με μία GPU Titan X, μπορούμε να αναφερθούμε στον Πίνακα 5 (**Design A** vs. **GPU+CPU**), ο οποίος προέρχεται από το αντίστοιχο paper της υλοποίησης [65] (ή το Reference [15] στην δική μας εργασία). Παρατηρούμε ότι, και οι δύο υλοποιήσεις πετυχαίνουν παρόμοιο accuracy έχοντας όμως στο FPGA χαμηλότερη αναπαράσταση αριθμών (16 bit Fixed Point έναντι 32 bit Floats στην GPU). Επίσης παρόλο που η υλοποίηση με FPGA έχει 40% χαμηλότερο Throughput (**1385.5 GOPs** η GPU) είναι περίπου 50% πιο Energy Efficient (**6.55 GOPs/W** έναντι **4.22 GOPs/W** της GPU).

	CPU	GPU+CPU	Work [12]	Work [13]	Design A	Design B	Design C	Design D	Design E
Device	AMD A10	NVIDIA Titan X	Zynq XC7Z045	Stratix-V GSD8	Virtex-7 VX690t	Virtex-7 VX690t	Virtex-7 VX690t	Virtex-7 VX690t	Virtex-7 VX690t
Technology	32nm	28nm	28nm	28nm	28nm	28nm	28nm	28nm	28nm
CNN Model	AlexNet	AlexNet	VGG & AlexNet	VGG& AlexNet	**Alex**	**Alex**	**VGG**	**VGG**	**VGG**
Precision	float	float	fixed(16b)	fixed(8-16b)	fixed(16b)	fixed(16b)	fixed(16b)	fixed(16b)	fixed(16b)
Accuracy Top-1	-	54.3%	68.02%	66.58%	52.4%	52.4%	66.51%	66.52%	66.51%
Accuracy Top-5	-	78.7%	87.94%	87.48%	77.83%	77.83%	86.89%	86.92%	86.88%
Frequency (MHz)	3,800	~ 1,000	150	120	150	150	150	150	150
Power (Watt)	87.3	328.3	9	19.1	126	126	35	35	160
Objective	-	-	-	Throughput	**Energy***	**Latency***	**Energy***	**Latency***	**Through.***
Batch Size	16	256	1	1	16	1	2	1	2
# of FPGAs	-	-	1	1	**1+4‡**	**1+4‡**	**1+1‡**	**1+1‡**	**1+6‡**
Through. (GOPS)	**34.23**	**1385.5**	**137.0**	**117.8**	**825.6**	**128.8**	**290**	**203.9**	**1280.3**
Latency (ms)	**83.6**	**89.7**	**224.6**	**262.9**	**104**	**30.6**	**213.6**	**151.8**	**200.9**
E.-E.(GOPS/J)	**0.39**	**4.22**	**15.2**	**6.17**	**6.55†**	**1.02†**	**8.28†**	**5.83†**	**8.00†**

* Optimization objective for 'Energy' uses the metric of 'GOPS/J' and objective for 'Latency' uses 'second/image.'
‡ 1+N represents 1 Zynq board plus N VC709 boards.
† The power consumption also include the ZC706 board; therefore the overall energy-efficiency is less than reported in Figure 4.

Table 5: Σύγκριση υλοποίησης FPGA [65] (Design A) με CPU, GPU

Loop Unroll και Tilling στα LN, LC, LJ, LK Loops

Αυτές οι αρχιτεκτονικές κάνουν Loop Unroll και στα L_J, L_K πέρα από τα L_C, L_N loops, για παράλληλη επεξεργασία και με βάση τις διαστάσεις των φίλτρων.

Στις προηγούμενες υλοποιήσεις δεν αξιοποιήθηκε αυτή η παραλληλία καθώς οι διαστάσεις των φίλτρων συνήθως είναι πολύ μικρές, οπότε δεν υπάρχει περιθώριο για μεγάλες αυξήσεις στην επίδοση. Επίσης, το άλλο πρόβλημα που έχουν είναι ότι τα PEs πρέπει να παραμετροποιηθούν διαφορετικά για κάθε Layer, οπότε αυτό αυξάνει την πολυπλοκότητα του σχεδιασμού.

Από τις υλοποιήσεις [9], [28], [36] που εφαρμόζουν αυτή την τεχνική την καλύτερη καθαρή υπολογιστική επίδοση την έχει η [36] (**565.94 GOPs**) πάνω στο AlexNet με ενεργειακή κατανάλωση **30.20 W** αντίστοιχη της [28] που έχει όμως χειρότερη επίδοση (**187.24 GOPs**) για το AlexNet. Επίσης η [36] έχει και την καλύτερη ενεργειακή απόδοση από αυτές τις 3 υλοποιήσεις (**18.7 GOPs/W,** έναντι **5.5 GOPs/W** της [28] και **14.2 GOPs/W** της [9]).

Loop Unroll και Tiling στα LN, LU, LV Loops

Αυτές οι αρχιτεκτονικές κάνουν Loop Unroll και στα L_U, L_V πέρα από τα L_N loops, για παράλληλη επεξεργασία και με βάση τις διαστάσεις του Output FM.

Επειδή οι διαστάσεις των Output FM καθώς και ο αριθμός των φίλτρων, κάνουν αρκετά ακριβό το Unroll στο Hardware. Μόνο συσκευές με μεγάλες υπολογιστικές ικανότητες (π.χ. μεγάλο πλήθος DSP Block) μπορούν να αξιοποιήσουν αποδοτικά αυτήν την παραλληλία. Για παράδειγμα η [77] καταφέρνει επίδοση **75.16** GOPs στο AlexNet με ενεργειακή κατανάλωση **33.93 W**, και άρα ενεργειακή απόδοση **2.2 GOPs/W**. Συγκρίνοντας και με τις προηγούμενες υλοποιήσεις βλέπουμε ότι δεν υπερτερεί σε καμία μετρική.

Loop Unroll και Tiling για βέλτιστα Factors στα LN, LC, LJ, LK Loops

Σε όλες τις προηγούμενες αρχιτεκτονικές τα loop L_C, L_N, L_J, L_K γίνονται Unroll με τον ίδιο τρόπου που γίνονται Tiled, δηλαδή ($T_i = P_i$). Σε αυτές τις αρχιτεκτονικές εξερευνάται η εύρεση του βέλτιστου Unroll και Tiling Factor.

Πιο συγκεκριμένα έχει φανεί ότι τα Input FMs και τα βάρη επαναχρησιμοποιούνται πιο αποδοτικά όταν γίνονται unrolled οι υπολογισμοί για ένα μόνο Input FM (δηλαδή όταν $P_C = P_J = P_K = 1$). Επίσης τα Tiling Factors ορίζονται ώστε τα δεδομένα που χρειάζονται για τον υπολογισμό μιας εξόδου να μπορούν να χωρέσουν στους on-chip buffers ($T_C = C,\ T_K = K,\ T_J = J$). Η υλοποίηση της [80] εφαρμόζοντας αυτές τις τεχνικές καταφέρνει επίδοση **645.25 GOPs** στο VGG16 με ενεργειακή κατανάλωση **50.0 W** και άρα ενεργειακή απόδοση **12.9 GOPs/W**. Συγκρίνοντας με τις προηγούμενες, αποτελεί μια από τις πιο δυνατές υπολογιστικά αλλά και ενεργειακά υλοποίηση.

Υλοποιήσεις με Approximate Computing

Θα εξετάσουμε αρχικά κάποιες υλοποιήσεις που παρουσιάζονται συγκριτικά στον Πίνακα 6 και Πίνακα 7 και προέρχεται από την δουλειά του [2]. Αυτές οι υλοποίησεις εφαρμόζουν κατά κύριο λόγο Approximate Υπολογισμό και Low Rank Approximation αντίστοιχα. Όπως και προηγούμενες, κάποιες από αυτές εφαρμόζουν και άλλες τεχνικές, όμως δεν είναι αυτές που τις χαρακτηρίζουν.

		Dataset	Network Workload		Bitwidth				Acc	Device	Freq	Through.	Power	LUT	DSP	Memory
			Comp. (GOP)	Param. (M)	In/Out	FMs	W-C	W-FC			(MHz)	(GOPs)	(W)	(K)		(MB)
FP32	[61]	ImageNet	30.8	138.0	32	32	32	32	90.1	Arria10 GX1150	370	866	41.7	437	1320	25.0
FP16	[30]	ImageNet	1.4	61.0	16	16	16	16	79.2	Arria10 GX1150	303	1382	44.3	246	1576	49.7
DFP	[80]	ImageNet	30.8	138.0	16	16	8	8	88.1	Arria10 GX1150	150	645		322	1518	38.0
	[84]	ImageNet	30.8	138.0	16	16	16	16		Arria10 GX1150	200	720		132	1518	44.5
	[61]	ImageNet	30.8	138.0	16	16	16	16		Arria10 GX1150	370	1790		437	2756	29.0
BNN	[104]	Cifar10	1.2	13.4	20	2	1	1	87.7	Zynq Z7020	143	208	4.7	47	3	
	[52]	Cifar10	0.3	5.6	20/16	2	1	1	80.1	Zynq Z7045	200	2465	11.7	83		7.1
		MNIST	0.0	9.6	8	2	1	1	98.2			5905		364	20	
	[106]	Cifar10	1.2	13.4	8	8	1	1	86.3	Stratix5 GSD8	150	9396	26.2	438	20	44.2
		ImageNet	2.3	87.1	8	32	1a	1	66.8			1964		462	384	
TNN		Cifar10	1.2	13.4					89.4			10962	13.6	275		39.4
	[107]	SVHN	0.3	5.6	8	2	2	2	97.6	Xilinx7 VX690T	250	86124	7.1	155		12.2
		GTSRB	0.3	5.6					99.0			86124	6.6	155		12.2

Table 6: FPGA-based CNN επιταχυντές υλοποιώντας Approximate υπολογισμό

		Dataset	Network Workload		Removed	Bitwidth	Acc	Device	Freq	Through.	Power	LUT	DSP	Memory
			Comp. (GOP)	Param. (M)	Param. (%)		(%)		(MHz)	(GOPs)	(W)	(K)		(MB)
SVD	[9]	ImageNet	30.5	138.0	63.6	16 Fixed	87.96	Zynq 7Z045	150	137	9.6	183	780	17.5
Pruning	[45]	Cifar10	0.3	132.9	89.3	8 Fixed	91.53	Kintex 7K325T	100	8621	7.0	17	145	15.1
	[7]	ImageNet	1.4	61.0	85.0	32 Float	79.70	Stratix 10	500	12000	141.2			

Table 7: FPGA-based CNN επιταχυντές υλοποιώντας Low Rank Approximation

Approximate Υπολογισμός

Αυτές οι αρχιτεκτονικές κάνουν χρήση προσεγγιστικών υπολογισμών χρησιμοποιώντας πολύ μικρότερες αναπαραστάσεις από τις συνηθισμένες. Πιο συγκεκριμένα, θα εστιάσουμε στις Binary (1 bit) και Ternary (2 bit) υλοποιήσεις.

Αρχικά η δουλειά στην [104], χρησιμοποιεί 20 bits για την αναπαράσταση των εισόδων-εξόδων, 2 bit για τα FMs και 1 bit για τα βάρη. Στο Cifar10 πετυχαίνει υπολογιστική επίδοση **208 GOPs** με κατανάλωση **4.7 W**, και άρα **44.2 GOPs/W** ενεργειακή απόδοση. Επίσης η ακρίβεια του μοντέλου είναι στα **87.7%**. Άρα συγκρίνοντας με τις προηγούμενες υλοποιήσεις, βλέπουμε ότι έχουμε υψηλό Throughput, αλλά το πιο σημαντικό είναι η πολύ καλύτερη ενεργειακή απόδοση από όλα τα προηγούμενα design, και με μικρή απώλεια της ακρίβειας.

Η δουλειά στην [52] έχοντας την ίδια αναπαράσταση με την [104] αλλά χρησιμοποιώντας καλύτερο network configuration και ένα πιο δυνατό FPGA καταφέρνει υπολογιστική επίδοση **2465 GOPs** (= **2.5 TOPs**) στο Cifar10 με κατανάλωση **11.7 W**, και άρα **210.7 GOPs/W** ενεργειακή απόδοση. Αυτές οι πολύ καλές επιδόσεις έρχονται όμως σε κόστος μικρότερης κατά 7% σχεδόν ακρίβειας με το προηγούμενο design (**80.1% Accuracy**).

Η δουλειά στην [106], χρησιμοποιεί 8 bits για την αναπαράσταση των εισόδων-εξόδων, αλλά και 8 bit για τα FMs και 1 bit για τα βάρη. Πετυχαίνει την καλύτερη επίδοση από όλες τις υλοποιήσεις με Binary αναπαράσταση των βαρών (**9396 GOPs = 9.3 TOPs**), την καλύτερη ενεργειακή απόδοση (**358.6 GOPs/W**) και με ακρίβεια σχεδόν όσο και η υλοποίηση της [104] (**86.3% Accuracy**).

Τέλος, η δουλειά στην [107], διαφοροποιείται από τις άλλες καθώς χρησιμοποιεί τριαδική αναπαράσταση για τα βάρη. Πιο αναλυτικά, χρησιμοποιεί 8 bits για την αναπαράσταση των εισόδων-εξόδων, 2 bit για τα FMs και 2 bit για τα βάρη. Παρόλο που χρησιμοποιεί 1 bit παραπάνω αναπαράστασης για τα βάρη καταφέρνει την καλύτερη υπολογιστική επίδοση από όλες τις Binary υλοποιήσεις (**10962 GOPs = 10.9 TOPs**), την μακράν καλύτερη ενεργειακή απόδοση (**1661 GOPs/W**) και με καλύτερη ακρίβεια από όλες (**89.4% Accuracy**).

Low Rank Approximation

Αυτές οι αρχιτεκτονικές κάνουν χρήση τεχνικών Weight Pruning και Low Rank Approximation. Πιο αναλυτικά θα δούμε δύο υλοποιήσεις που εφαρμόζουν αυτές τις δύο τεχνικές.

Η δουλειά στην [9], χρησιμοποιεί Low Rank Approximation στα Fully Connected Layers για να μειώσει σημαντικά τον αριθμό των βαρών. Χρησιμοποιώντας το VGG16, παράγουν ένα μοντέλο VGG16-SVD το οποίο έχει 63% λιγότερες παραμέτρους και πετυχαίνει **87.96% Accuracy** στο ImageNet. Επίσης χρησιμοποιεί 16 bits fixed point αναπαράσταση. Οι επιδόσεις του δεν είναι και από τις καλύτερες που έχουμε συναντήσει (**136.97 GOPs**) όπως αντίστοιχα και η ενεργειακή του απόδοση (**14.2 GOPs/W**), αλλά το μοντέλο καταφέρνει και πιάνει πολύ λιγότερη μνήμη (**17 MB**) σε σχέση με κάποιες υλοποιήσεις με Binary αναπαράσταση (π.χ. [106] στο ImageNet **44.2 MB**).

Η δουλειά στην [7], χρησιμοποιεί την τεχνική του Weight Pruning και πιο συγκεκριμένα έχει έναν zero skip scheduler, ο οποίος αναγνωρίζει τα μηδενικά στοιχεία και δεν τα δρομολογεί ποτέ για MAC επεξεργασία. Με αυτή την τεχνική καταφέρνει την καλύτερη επίδοση που υπάρχει σε αυτό το paper (**12000 GOPs**) με το AlexNet μοντέλο στο ImageNet, χωρίς όμως να είναι το πιο ενεργειακά αποδοτικό (**85 GOPs/W**). Επίσης δεν έχει και την καλύτερη ακρίβεια από τα μοντέλα (**79.7% Accuracy**), αλλά είναι αρκετά ικανοποιητική και συγκρίσιμη με των υπόλοιπων υλοποιήσεων.

Υλοποίηση FPGA Accelerator Υψηλής Απόδοσης

Τέλος παρουσιάζουμε αποτελέσματα της υλοποίησης της [4], η οποία αποτελεί μια από τις καλύτερες υλοποιήσεις που βρήκαμε για FPGA επιταχυντή για Βαθιά Νευρωνικά Δίκτυα, όσον αφορά και τις τρεις μετρικές που σχολιάζουμε.

Η δουλειά αυτή, χρησιμοποιεί μία Ternary αναπαράσταση για τα βάρη και πιο αναλυτικά έχει 8 bit για τα Activations και 2 bit για τα βάρη. Υλοποιεί Dynamic Fixed Point αναπαράσταση, ώστε να υπάρχει υψηλότερη ακρίβεια στα απαιτούμενα Layer, και χαμηλότερη ακρίβεια στα πιο υπολογιστικά απαιτητική Layers. Για αυτόν τον λόγο κιόλας, χρησιμοποιεί την CPU για την επεξεργασία του πρώτου και τελευταίου Layer σε υψηλότερο precision ώστε να επιτευχθεί καλή ακρίβεια, χωρίς να περιορίζεται πολύ η απόδοση.

Χρησιμοποιώντας μικρή ακρίβεια αναπαράστασης, σχεδιάζεται αντίστοιχα μία αρχιτεκτονική με PE, όπου χρησιμοποιείται αποκλειστικά λογικά Blocks (ALMs σε Altera ορολογία) για την υλοποίηση των πράξεων και σχεδόν καθόλου DSP. Το FPGA δοκιμάστηκε για το ResNet50 μοντέλο και έτρεξε σε δύο FPGA boards, το Arria 10 και το Stratix 10 το οποίο έχει περισσότερους πόρους από το πρώτο.

Στο Arria 10 η καλύτερη υλοποίηση (στα 400 MHz) πετυχαίνει **10 TOPs** (= **10000 GOPs**), το οποίο σύμφωνα με τους συγγραφείς ήταν η καλύτερη ως τότε υλοποίηση για το ResNet50 στο Arria 10. Η ενεργειακή του απόδοση είναι λίγο παραπάνω από **200 GOPs/W**, και η ακρίβεια του στα **71.1%**.

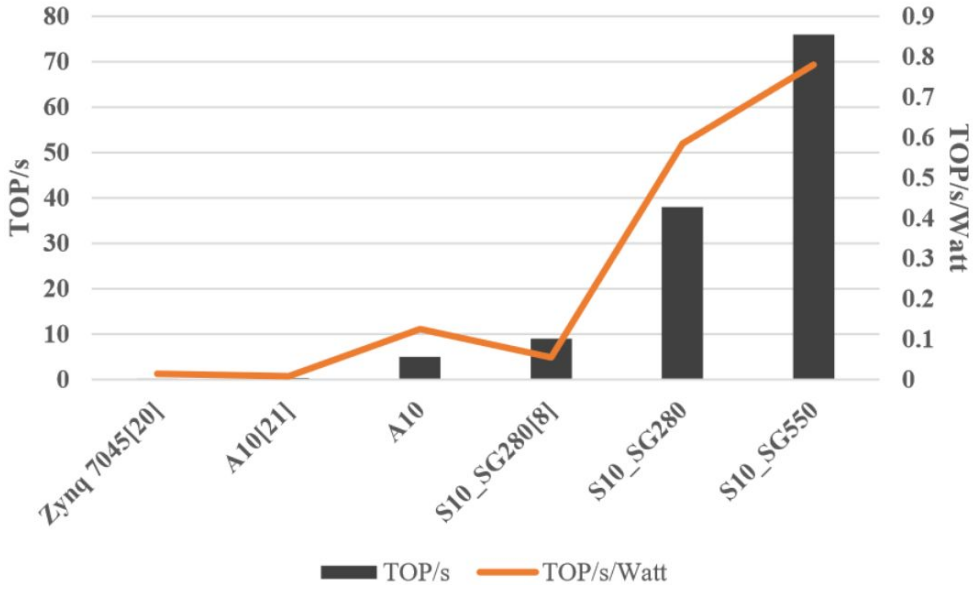


*Fig. 35: **Performance/Watt** για τις καλύτερες υλοποιήσεις στα Arria 10 (A10) και Stratix 10 (S10) FPGA για το ResNet50 δίκτυο μαζί με προηγούμενες δουλειές.*

Για το Stratix 10, κοιτώντας το διάγραμμα (Fig. 35) που φαίνεται παρακάτω, περιέχει τις 3 καλύτερες υλοποιήσεις για το Stratix 10 καθώς και υλοποιήσεις για το Arria 10 αλλά και άλλη δουλειά πάνω στο Xilinx Zynq Z-7045 SOC. Η καλύτερη υλοποίηση πετυχαίνει επίδοση **76 TOPs** (= **76000 GOPs**) και ενεργειακή απόδοση **780 GOPs/W**, κάνοντας αυτήν την υλοποίηση από τις καλύτερες υπολογιστικά και από τις πιο ενεργειακά αποδοτικές. Μάλιστα

οι επιδόσεις του, ξεπερνάνε επιδόσεις πολλών GPU και προσεγγίζουν αυτές μίας TPU (**96 TOPS**, **1200 GOPs/W**). Η ακρίβεια του μοντέλου είναι κοντά στα 70%, το οποίο είναι αρκετά αποδεκτό.

Επίσης κοιτάζοντας τον Πίνακα 8, και βλέποντας τις υπολογιστικές επιδόσεις και ενεργειακές αποδόσεις διαφόρων μοντέλων GPU της Nvidia, βλέπουμε ότι η μόνη που έχει καλύτερη επίδοση είναι η Nvidia V100 που πρόκειται για την Top-of-the-line GPU της Nvidia για AI Applications. Παρόλο που έχει κατά 65% μεγαλύτερο Throughput (**125 TOPs**) έχει 46% σχεδόν χειρότερη ενεργειακή απόδοση (**417 GOPs/W**).

GPU	TOP/s	Power (W)	Precision	TOP/s/Watt
V100	125	300	FP16	0.416667
P4	22	75	INT8	0.293333
P40	47	250	INT8	0.188
P100	18.7	250	FP16	0.0748

Table 8: Θεωρητικά Peak TOP/s και TOP/s/Watt τελευταίων GPU της NVIDIA για Inferencing με χαμηλό Precision

Συμπεράσματα

Σε αυτό την εργασία εκθέσαμε τις δυνατότητες παραλληλίας των συνεκτικών νευρωνικών δικτύων και εξετάσαμε τις τεχνικές που μας επιτρέπουν να τις εκμεταλλευτούμε. Παρουσιάσαμε διάφορες αρχιτεκτονικές που μας δίνουν την δυνατότητα να εκμεταλλευτούμε την υπολογιστική δύναμη και ευελιξία του FPGA. Ακόμα εστιάσαμε στις αλλαγές που μπορούμε να κάνουμε στα δεδομένα που επεξεργαζόμαστε και κατά πόσο αυτές επηρεάζουν την ταχύτητα αλλά και την ακρίβεια του συστήματος μας.

Καταλήγουμε στο συμπέρασμα ότι τα FPGA μπορούν να ακολουθήσουν πιστά τις εξελίξεις στα CNN και να αποτελέσουν ένα πολύ καλό υποψήφιο για inference αυτών. Καθώς συνδυάζει υψηλές υπολογιστικές ικανότητες και μικρή κατανάλωση ενέργειας, αντίθετα με των ανταγωνισμό (CPU και GPU) που δεν μπορεί να πετύχει ισορροπία μεταξύ των δύο.

References

[1] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang and Huazhong Yang. 2017. *[DL] A Survey of FPGA-Based Neural Network Inference Accelerator*. ACM Trans. Recong. Technol. Syst.9, 4, Article 11 (December 2017). *arXiv*: arXiv:1712.08934

[2] Kamel Abdelouahab, Maxime Pelcat, Jocelyn Sérot and François Berry, F. 2018. *Accelerating CNN inference on FPGAs: A Survey*. *arXiv*: arXiv:1806.01683

[3] Jin Hee Kim, Brett Grady, Ruolong Lian, John Brothers, Jason H. Anderson. 2017. *FPGA-based CNN inference accelerator synthesized from multi-threaded C software*. 30th IEEE International System-on-Chip Conference (SOCC), Munich, pp. 268-273. *arXiv*: arXiv:1807.10695

[4] Sudarshan Srinivasan, *et al*. 2019. *High Performance Scalable FPGA Accelerator for Deep Neural Networks*. *arXiv*: arXiv:1908.11809

[5] Richard Chamberlain, BittWare. *FPGA Acceleration of Binary Weighted Neural Network Inference*. White Paper. Available: FPGA Acceleration of Binary Weighted Neural Network Inference

[6] Bhavesh Patel. 2018. *Where The FPGA Hits The Server Road For Inference Acceleration*. Available: Where The FPGA Hits The Server Road For Inference Acceleration

[7] Norman P. Jouppi, *et al*. 2017. *In-datacenter performance analysis of a tensor processing unit.* ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA), Toronto, ON, 2017, pp. 1-12, *arXiv*: arXiv:1704.04760

[8] Jason Brownlee. 2019. *A Gentle Introduction to Batch Normalization for Deep Neural Networks*. Available: A Gentle Introduction to Batch Normalization for Deep Neural Networks

[9] Abhiemanyu Pandit. 2019. *Introduction to FPGA and It's Programming Tools*. Available: What is FPGA: Introduction, Architecture & Programming Tools

[10] Cathal Murphy and Yao Fu. 2017. *Xilinx All Programmable Devices: A Superior Platform for Compute-Intensive Systems*. White Paper: FPGAs/SoCs. Available: Xilinx All Programmable Devices: A Superior Platform for Compute-Intensive Systems (WP492)

[11] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. 2012. *ImageNet Classification with Deep Convolutional Neural Networks.* Imagenet classification with deep convolutional neural networks. 1097-1105, Available: ImageNet Classification with Deep Convolutional Neural Networks

[12] Matthieu Courbariaux, Yoshua Bengio, Jean-Pierre David. 2015. *Training deep neural networks with low precision multiplications*. Accepted as a workshop contribution at ICLR 2015. *arXiv:* arXiv:1412.7024

[13] Matthieu Courbariaux, Yoshua Bengio, Jean-Pierre David. 2015. *BinaryConnect: Training Deep Neural Networks with binary weights during propagations*. Accepted at NIPS 2015. *arXiv:* arXiv:1511.00363

[14] Dong Nguyen, Daewoo Kim and Jongeun Lee. 2017. *Double MAC: Doubling the performance of convolutional neural networks on modern FPGAs*., Available: Double MAC: Doubling the Performance of Convolutional Neural Networks on Modern FPGAs

[15] Chen Zhang, Di Wu, Jiayu Sun, Guangyu Sun, Guojie Luo, Jason Cong. 2016. *Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster*. Available: Energy-Efficient CNN Implementation on a Deeply Pipelined FPGA Cluster

Appendix A

Πολυπλοκότητα Separable Filters

Separable Filter Dimensions:

- Input Dimensions: $N \times M$
- Kernel Dimensions: $n \times m$

2D Convolution:

Total 2D Conv **Complexity**: $O((N-n+1) \times (M-m+1) \times n \times m)$

$$When\ N \gg n\ and\ M \gg m,$$
$$O(N \times M \times n \times m)$$

Total Spatial **Complexity**: $O((N-n+1) \times (M-m+1))$

1D Convolution:

1D Filters Dimensions:

- 1D-Vertical Dimensions: $n \times 1$
- 1D-Horizontal Dimensions: $1 \times m$

1D-Vertical **Complexity** : $O((N-n+1) \times M \times n)$
Intermediate Buffer Size Spatial **Complexity**: $O((N-n+1) \times M)$

1D-Horizontal **Complexity** : $O(N \times (M-m+1) \times m)$
Total 1Ds Conv **Complexity** : $O(N \times (M-m+1) \times m + (N-n+1) \times M \times n)$

$$When\ N \gg n\ and\ M \gg m,$$
$$O(N \times M \times m + N \times M \times n) =$$
$$O(N \times M \times (n+m))$$

Output Array size Spatial **Complexity**: $O((N-n+1) \times (M-m+1))$
For in place algorithm,
Total Spatial **Complexity**: $O((N-n+1) \times M)$