MACHINE (DEEP) LEARNING IN SISTEMI DI TRIGGER/REAL-TIME

Stefano Giagu (Sapienza Università di Roma e INFN sezione di Roma)

Corso di Formazione Nazionale “Introduzione alle reti neurali ed applicazioni sui dispositivi elettronici”,INFN Sezione di Napoli - 6-7.12.2022

INFN

Istituto Nazionale di Fisica Nucleare

PROGRAMMA DEL CORSO

- ieri:
  - **mattina (h10:00-12:00):** richiami elementi essenziali ANN, DNN in sistemi di trigger, tecniche di compressione di reti neurali
  - **pomeriggio (h14:00-…):** sessione hands-on: training, pruning e quantizzazione a 8bit di un semplice modello di rete neurale di tipo convoluzionale utilizzando la libreria Tensorflow/Keras
- oggi:
  - **mattina (h10:00-12:00):** tecniche di compressione di reti neurali (2-nda parte), inferenza AI su FPGA, l'ambiente di sviluppo Xilinx VitisAI, la libreria hls4ml, use-case applicativo: trigger hw muonico per l'upgrade di fase2 dell'esperimento ATLAS
  - **pomeriggio (h14:00-…):** sessione hands-on:
    - quantizzazione a 8bit del modello studiato nell'hands-on precedente e compilazione usando la libreria Vitis AI, run su scheda acceleratrice Xilinx Alveo U50
    - training di un modello CNN per la ricostruzione a bassissima latenza di muoni nel trigger di livello-0 dell'esperimento ATLAS, quantizzazione spinta con la libreria Qkeras, ottimizzazione delle prestazioni del modello quantizzato attraverso knowledge-distillation, sintesi del modello con hls4ml

TECHNICHE DI COMPRESSIONE

- **pruning:** vengono eliminati i pesi del modello che hanno minore effetto sulla risposta del modello oppure che risultano più piccoli in modulo
- **weight sharing (o clustering):** gruppi di pesi vicini vengono raggruppati in cluster e il loro valore viene sostituito con un valore unico
- **quantizzazione:** può essere applicata ai pesi e/o alle uscite delle funzioni di attivazione dei layer della rete neurale. Consiste nel ridurre la precisione con cui si rappresentano le diverse grandezze (64/32bit →16bit, 8bit, 4bit …)
- **Binary / Ternary net:** versioni estreme della quantizzazione (reti con pesi e attivazioni binarie [0,1] o ternarie [-1,0,1] → possono essere implementate su FPGA in modo efficiente e compatto, ma risultano molto sensibili al rumore indotto dalla quantizzazione

TECHNICHE DI COMPRESSIONE

- **sparse regularisation:** appartiene allo stesso gruppo delle tecniche di pruning (training aware pruning) e di dropout (post-training pruning), in cui la sparsità del modello è ottenuta applicando tecniche di regolarizzazione L1, L2 o L1+L2 durante il training dell rete stesse

- **distillazione via teacher-student knowledge transfer:** si usa un modello complesso pre-addestrato, come guida durante l'addestramento di un modello semplificato

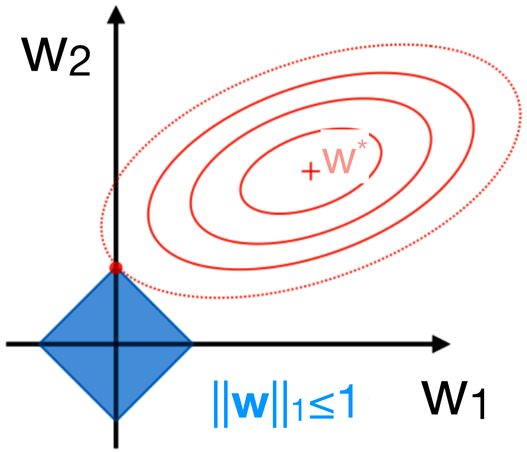
- **low-rank factorisation:** riduce la dimensionalità delle matrici dei pesi applicate ad ogni layer della rete per velocizzare le operazioni di consolazione nei primi layer (che dominano i tempi di inferenza in una deep CNN)

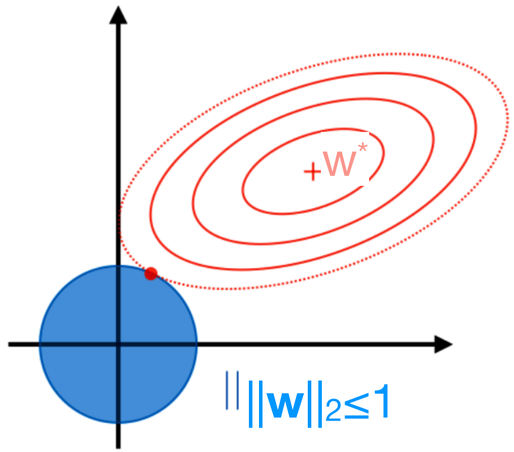
- **3x3 winograd convolutions (NVIDIA):** versione più efficiente della convoluzione in CNN che riduce il numero di moltiplicazioni floating-point necessarie per ogni convocazione

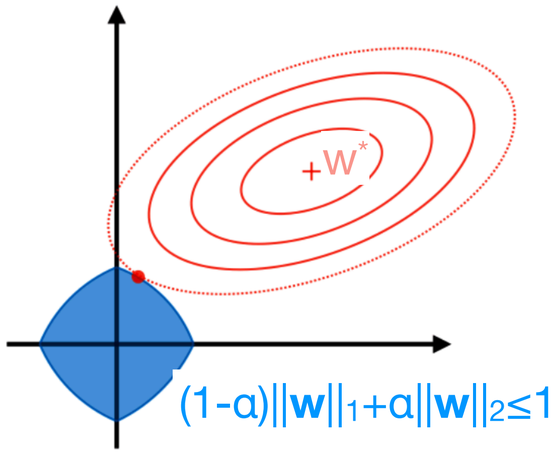
SPARSIFICAZIONE VIA REGOLARIZZAZIONE L1/L2/L3

- idea: vincolare la complessità del modello penalizzando valori grandi dei pesi a meno che non sia fortemente richiesto dai dati stessi
- si aggiunge una penalty alla loss function: L(**w**) → L(**w**) + Ω(**w**)

L1 **LASSO**	L2 **weight decay**	L1+L2 **Elastic Net**
• Shrinks coefficients to 0 • Good for variable selection	Makes coefficients smaller	Tradeoff between variable selection and small coefficients







$$\Omega(\mathbf{w}) = \alpha \parallel \mathbf{w} \parallel_1 = \sum |w_k|$$

$$\Omega(\mathbf{w}) = \frac{\alpha}{2} \parallel \mathbf{w} \parallel_2^2 = \frac{\alpha}{2}\mathbf{w}^t\mathbf{w} = \sum w_k^2$$

$$\Omega(\mathbf{w}) = \lambda[(1-\alpha) \parallel \mathbf{w} \parallel_1 + \alpha \parallel \mathbf{w} \parallel_2^2]$$

ESEMPIO: GD CON REGOLARIZZAZIONE L2

loss regolarizzata: $$L_R(\mathbf{w}) = L(\mathbf{w}) + \frac{\alpha}{2}\mathbf{w}^t\mathbf{w}$$

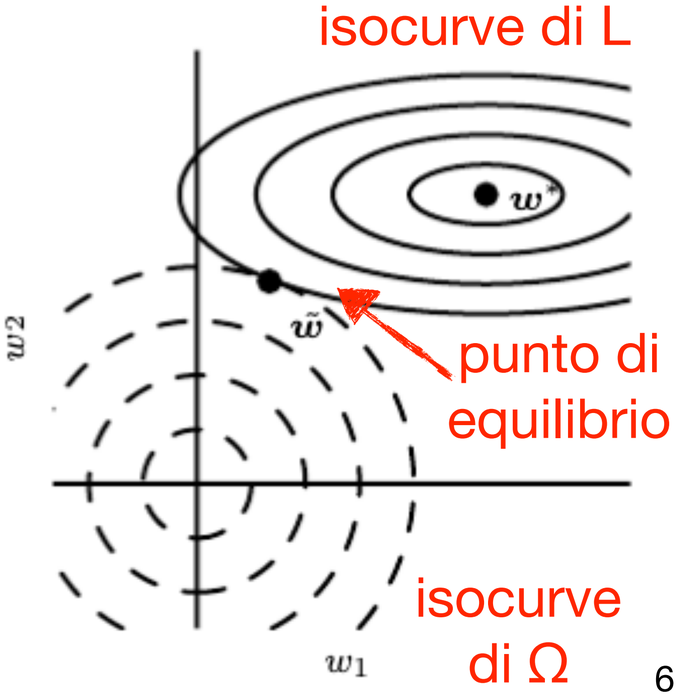
gradiente: $$\nabla_{\mathbf{w}} L_R(\mathbf{w}) = \nabla_{\mathbf{w}} L(\mathbf{w}) + \alpha\mathbf{w}$$

aggiornamento dei pesi: $$\mathbf{w} \leftarrow \mathbf{w} - \eta[\nabla_{\mathbf{w}} L(\mathbf{w}) + \alpha\mathbf{w}] = (1 - \eta\alpha)\mathbf{w} - \eta\nabla_{\mathbf{w}} L(\mathbf{w})$$

l'effetto di L2 sulla singola iterazione è quello di ridurre il vettore di pesi per un certo fattore

fatto su tutti gli step del training si può dimostrare che l'effetto complessivo è quello di riscalare le componenti del vettore soluzione **w*** del problema non regolarizzato proporzionalmente a $\lambda_i/(\lambda_i+\alpha)$ con λ_i gli autovalori della matrice Hessiana di L

- componenti nelle direzioni di **w** poco sensibili ($\lambda_i \ll \alpha$) → grande effetto di riduzione
- componenti nelle direzioni sensibili ($\lambda_i \gg \alpha$) → invariate



ESEMPIO: GD CON REGOLARIZZAZIONE L1

loss function regolarizzata: $L_R(\mathbf{w}) = L(\mathbf{w}) + \alpha \sum_i |w_i|$

gradiente: $\nabla_{\mathbf{w}} L_R(\mathbf{w}) = \nabla_{\mathbf{w}} L(\mathbf{w}) + \alpha \text{sign}[\mathbf{w}]$

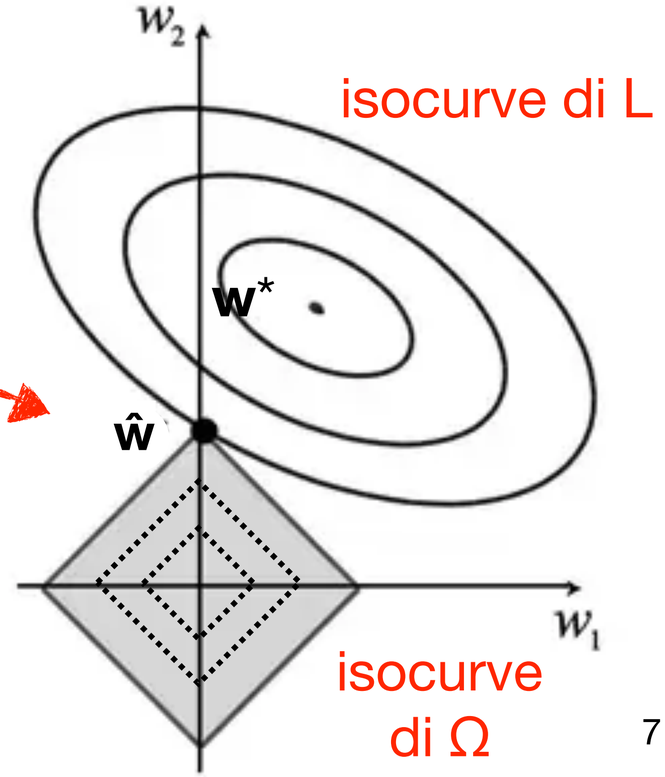
effetto molto diverso da quello di L2 e a causa delle singolarità soluzione non facile da trovare analiticamente

si può dimostrare che per una loss L con minimo quadratico e hessiano diagonale H = diag[$\lambda_1 \dots \lambda_n$]:

$$\hat{w}_i = \text{sign}[w_i^*] \max[\,|w_i^*| - \frac{\alpha}{\lambda_i}, 0]$$

con **w*** vettore soluzione del problema non regolarizzato

- $w_i^* \leq \alpha/\lambda_i \Rightarrow \hat{w}_i = 0$
- $w_i^* > \alpha/\lambda_i \Rightarrow \hat{w}_i$ scalato di α/λ



L1 agisce come feature selector / sparsificatore

WEIGHT SHARING O CLUSTERING

- l'algoritmo di clustering, o di condivisione dei pesi, riduce il numero pesi della rete che hanno valore univoco/simile, sostituendoli a gruppi con un peso unico rappresentativo del cluster
  - raggruppa i pesi di ogni layer in N cluster usando algoritmi di clustering (tipicamente density based come dbscan)
  - condivide il valore del centroide del cluster per tutti i pesi appartenenti al cluster
  - spesso si accompagna ad una quantizzazione dei pesi
- ha vantaggi simili a quelli del pruning
- NOTA: empiricamente si osserva che non funziona bene per layer che sono preceduti da batch normalisation o layer normalisation

SCHEMA WEIGHT CLUSTERING

Weight matrix

0.86	-1.88	-1.49	-0.51
-1.7	1.58	0.12	-2.38
1.9	-2.19	1.37	-2.79
1.92	0.46	-2.85	-0.61

clustering

0.86	-1.88	-1.49	-0.51
-1.7	1.58	0.12	-2.38
1.9	-2.19	1.37	-2.79
1.92	0.46	-2.85	-0.61

calcola centroidi

Centroid	Index
-2.85	0
-1.26	1
0.33	2
1.92	3

sostituisci la matrice dei pesi con una matrice di indici che puntano ai centroidi

Pull indices

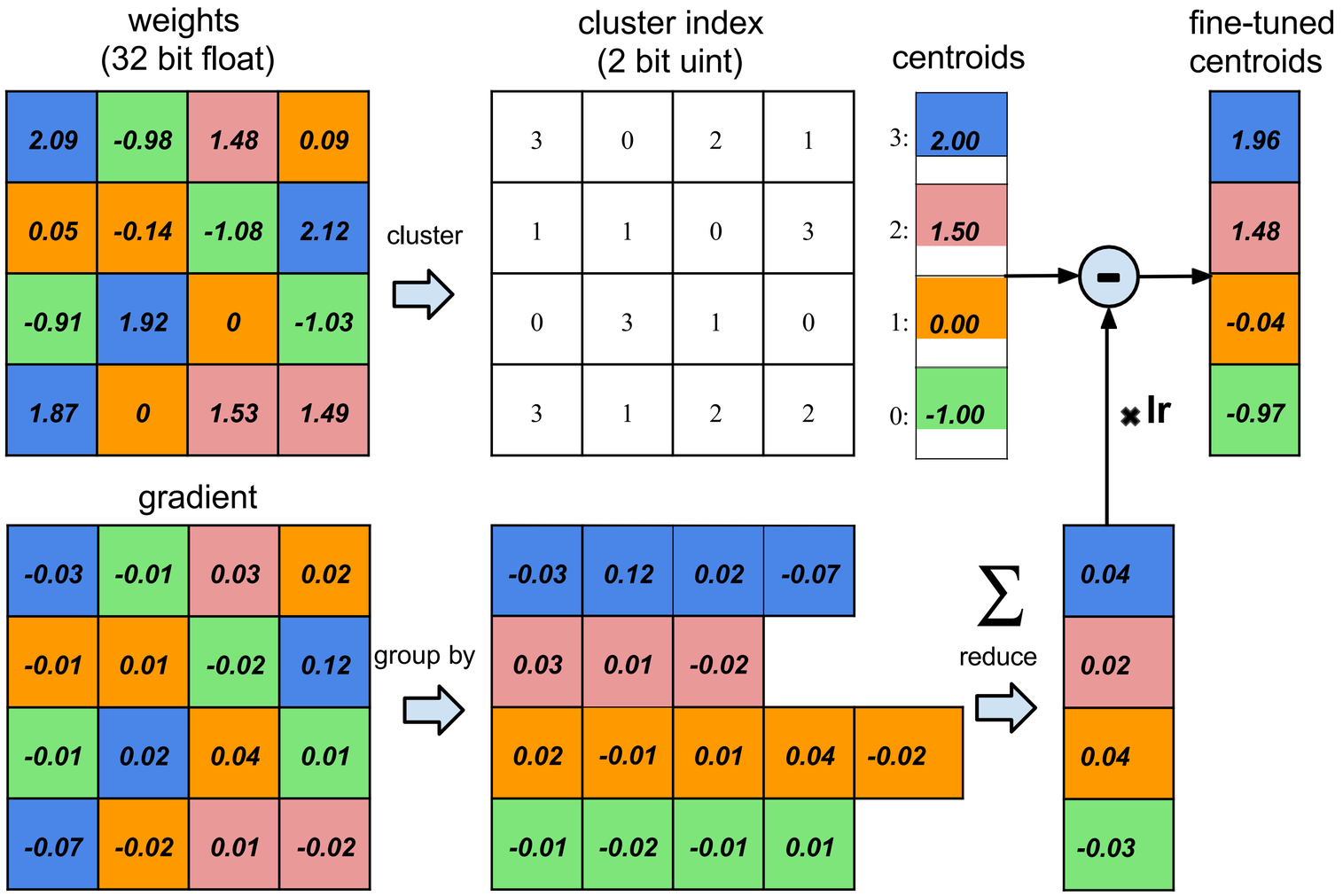
2	1	1	1
1	3	2	0
3	0	3	0
3	2	0	1

=

0.33	-1.26	-1.26	-1.26
-1.26	1.92	0.33	-2.85
1.92	-2.85	1.92	-2.85
1.92	0.33	-2.85	-1.26

NOTA: anche storando fp32 si stanno salvando 4 valori invece che 16

VARIAZIONE SUL METODO



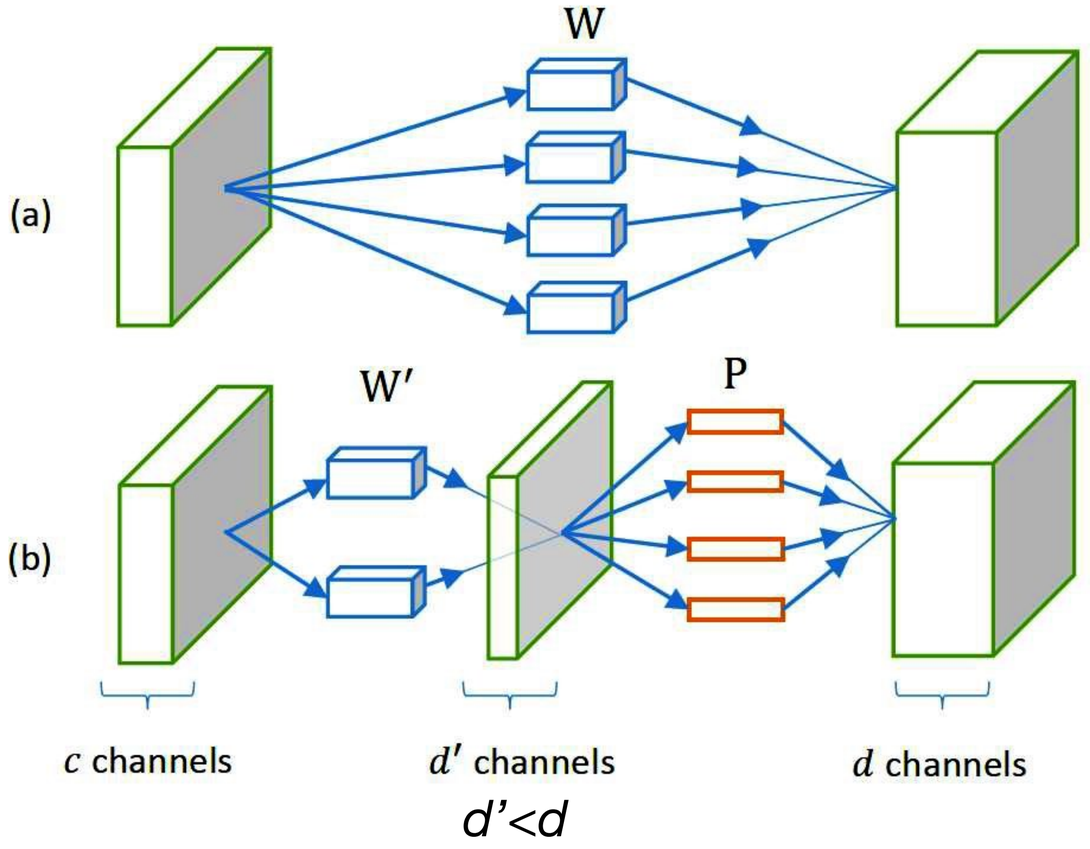
fine-tune i centroidi durante il training sottraendo i gradienti aggregati associati ad ogni cluster

S. Han et al. arXiv:1510.00149

LOW RANK FACTORIZATION

- riduce la dimensionalità delle matrici dei pesi applicate ad ogni layer convoluzionale della rete
- sfrutta la ridondanza (overparametrizzazione) presente nei filtri convoluzionali per derivare approssimazioni molto più veloci da calcolare (x2 speedup)

- Decompose a convolutional layer with d filters with filter size $k \times k \times c$ to
  - A layer with d' filters ($k \times k \times c$)
  - A layer with d filter ($1 \times 1 \times d'$)



Complessità

$$O(dk^2c)$$

$$O(d'k^2c) + O(d'd)$$

rapporto $\sim \frac{d'}{d}$

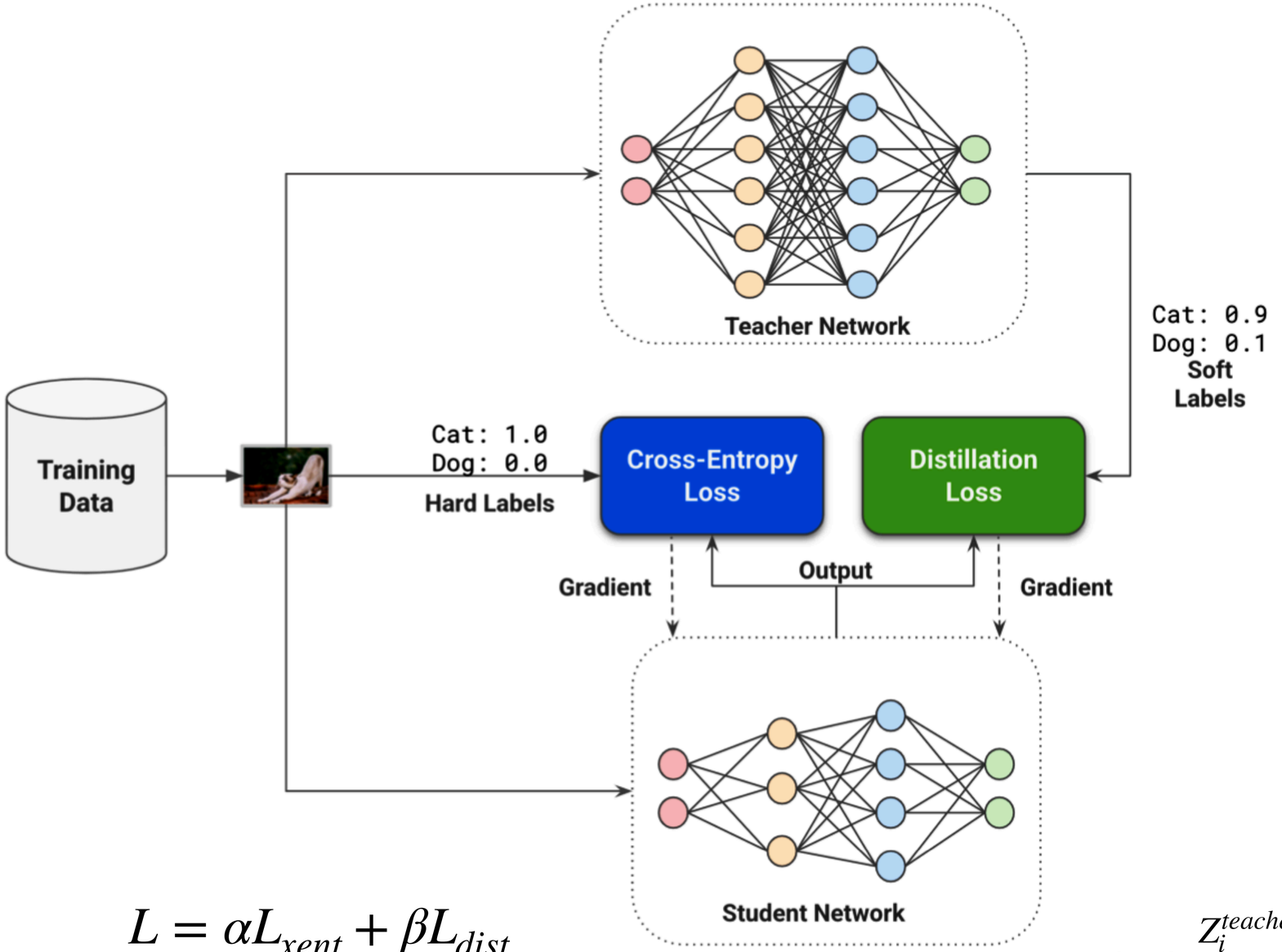
Zhang et al, arXiv:1411.4229
Denton et al, arXiv:1404.0736

KNOWLEDGE DISTILLATION

- Hinton e collaboratori, in un lavoro seminale del 2014 (arxiv:1503.02531) hanno esplorato la possibilità di insegnare a reti neurali lightweight (student) ad estrarre “dark knowledge" da un singolo modello insegnate complesso o anche da un ensemble di insegnanti. L’idea di Hinton (proposta per task di classificazione) era quella di usare un modello teacher per generare label software (soft label) per i dati di training (simile in qualche modo alla data augmentation in cui si creano dati sintetici per migliorare la capacità di generalizzazione di una rete neurale)

- le soft label sostituiscono i valori delle label ground-truth con le probabilità che l’input appartenga a ciascuna classe, probabilità stimate dal teacher

- durante il training il modello student può quindi apprendere sia dalle label hard (ground truth) che da quelle soft prodotte dal teacher



- sia il teacher che lo student ricevono lo stesso input
- lo studente viene addestrato utilizzando la normale x-entropy loss incrociata con le etichette hard, ed utilizzando la loss di distillazione che utilizza le soft label
- durante il training i pesi del teacher sono congelati e quindi non vengono aggiornati durante la backpropagation

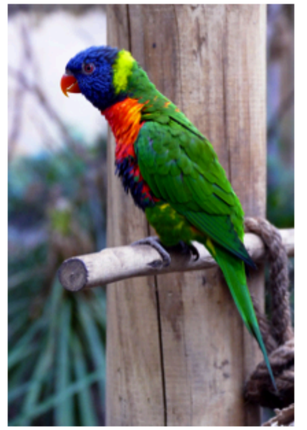
$$L = \alpha L_{xent} + \beta L_{dist}$$

$$L_{dist} = \text{x-entropy}(\hat{y}, y^{soft}; \mathbf{w})$$

$$y_i^{soft} = \frac{\exp \frac{Z_i^{teacher}}{T}}{\sum_j \exp \frac{Z_j^{teacher}}{T}}$$

T: parametro di temperatura che agisce da smooting per la distribuzione delle soft label

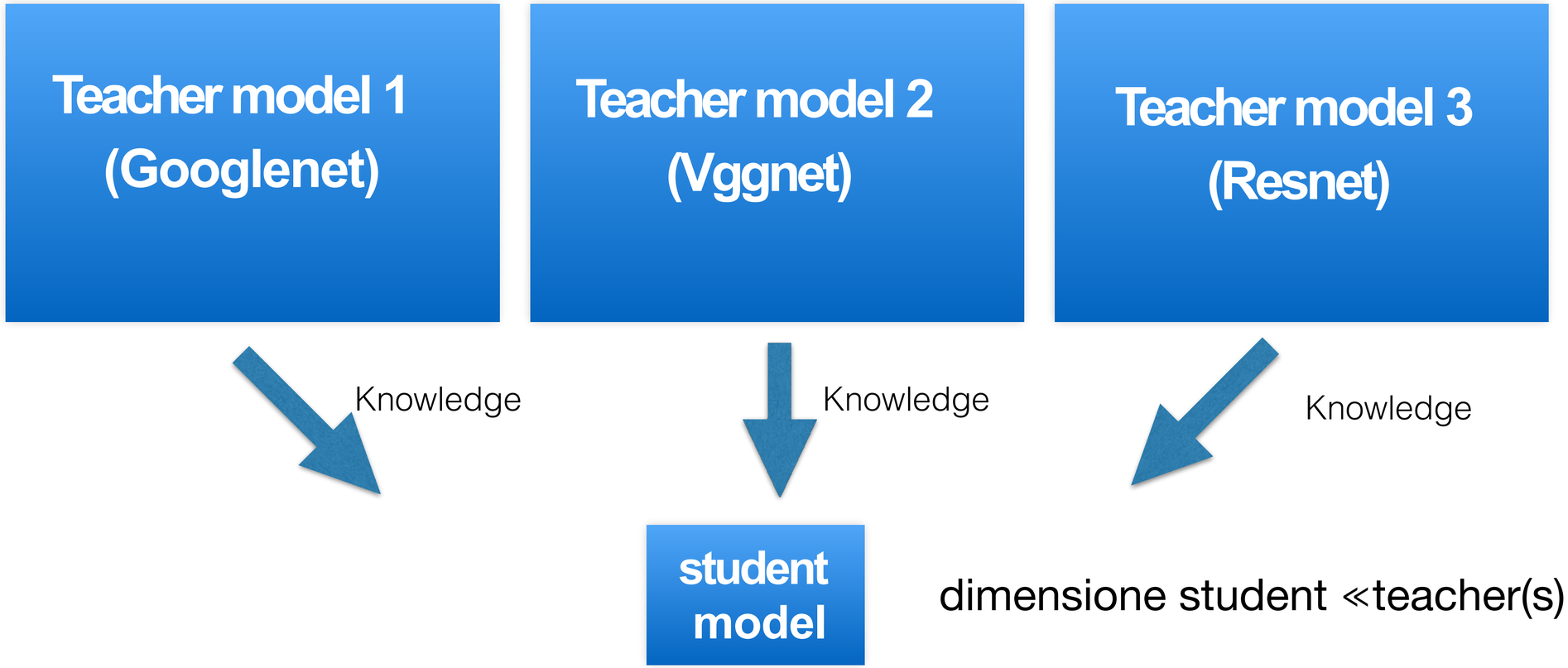
esempio:

Labels	Images			
				
Hard[19]	[1, 0, 0, 0]	[0, 1, 0, 0]	[0, 0, 1, 0]	[0, 0, 0, 1]
Soft	[.80, .15, .03, .02]	[.15, .75, .05, .05]	[.03, .02, .85, .10]	[.05, .05, .20, .79]

- il teacher pensa che un gatto sia più simile ad un cane che ad un piccione o a un pappagallo, e che questi ultimi siano più simil tra loro che al gatto o al cane
- la distillazione permette allo student di catturare relazioni tra le classi che non sono rappresentate nelle hard label del dataset di training

KNOWLEDGE DISTILLATION PER LA COMPRESSIONE DI DNN

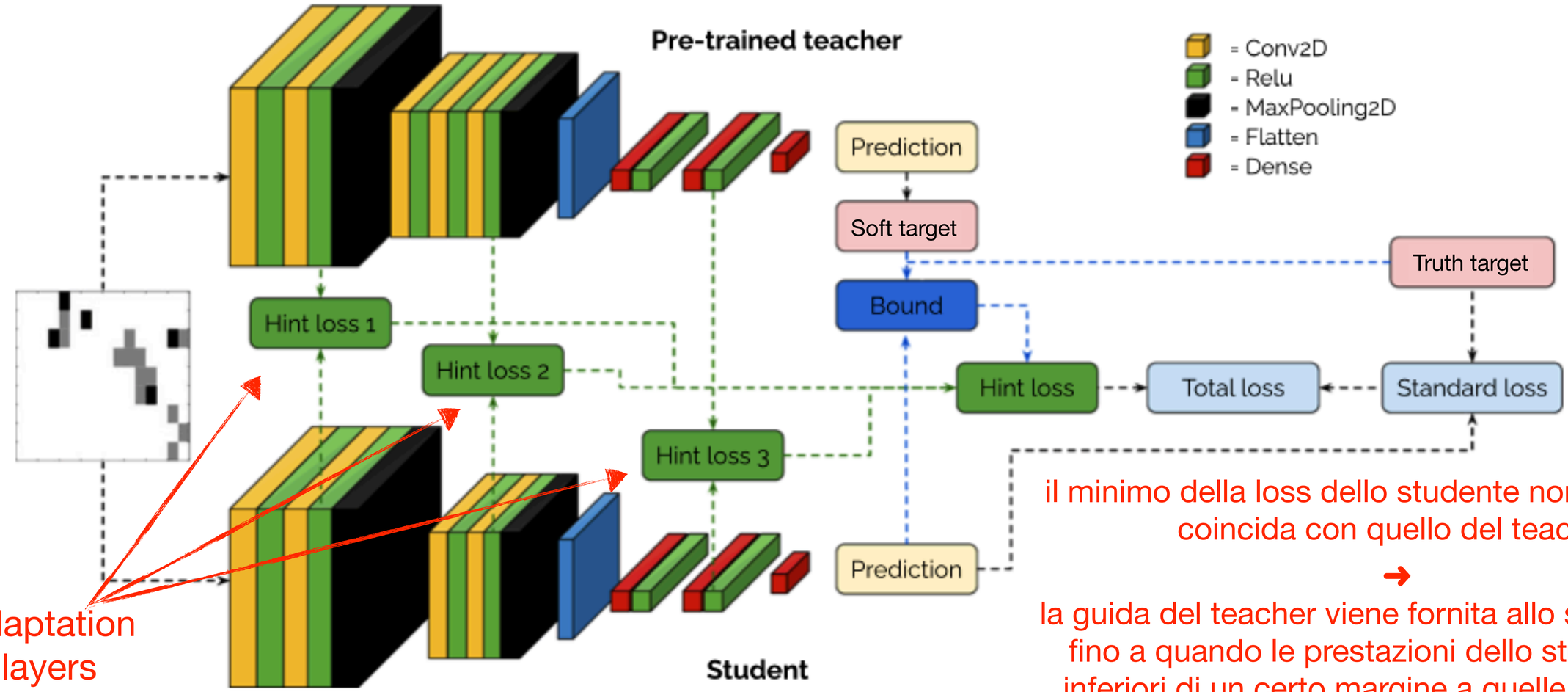
idea: trasferire la conoscenza appresa da uno o più DNN di grande dimensione e potere espressivo, pre-addestrato per la stessa task, ad un modello lightweight eventualmente anche compresso con i metodi visti in precedenza



dimensione student «teacher(s)

Hinton et al, arxiv:1503.02531
Romero et al, arxiv:1412.6550

ADVANCED KNOWLEDGE DISTILLATION



il minimo della loss dello studente non è detto che coincida con quello del teacher

➜

la guida del teacher viene fornita allo studente solo fino a quando le prestazioni dello student sono inferiori di un certo margine a quelle del teacher

permette di addestrare modelli molto leggeri che sarebbe altrimenti impossibile far convergere in assenza del teacher

Romero et al, arxiv:1412.6550

S. Francescato, S.Giagu, F. Riti, G.Russo, L.Sabetta, F.Tortonesi, Eur. Phys. J. C (2021) 81:969

ESEMPIO LOSS “GUIDATA”

- la guida del teacher viene fornita allo studente solo fino a quando le prestazioni dello student sono inferiori di un certo margine a quelle del teacher, il margine è controllato da un opportuno iperparametro γ

total loss $L(y, y_S, y_T)$

$$= \begin{cases} \|y - y_S\|^2 + \sum_{i=1}^{3} \gamma_i H_i & \text{if } \|y - y_T\|^2 < \gamma \|y - y_S\|^2 \\ \|y - y_S\|^2 & \text{Otherwise} \end{cases}$$

$$H_i = \|A_i - T_i^H\|^2$$

hint loss (adaptation layer)

y_S: predizione dello studente

y_T: predizione del teacher

γ: iperparametro per controllare l’effetto di guida del teacher

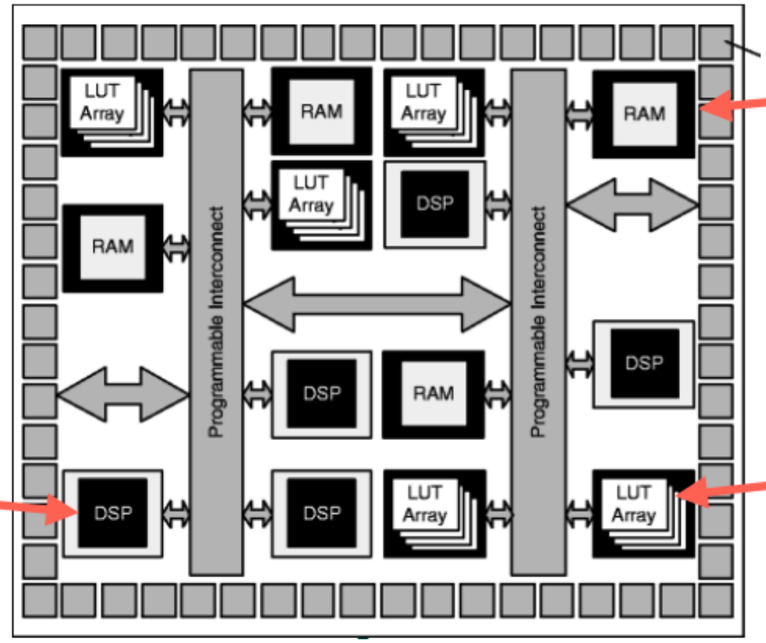
$\|\cdot\|^2$: morma L2

DNN SU FPGAs

- Field Programmable Gate Arrays sono circuiti integrati la cui logica può essere riprogrammata
- oggi hanno prestazioni confrontabili a quelle degli ASIC ma risultano molto più versatili, inoltre sono molto più efficienti dal punto di vista energetico rispetto a CPU e GPU
- sono generalmente strutturate come array di building blocks (celle logiche) (DSPs, LUT, FlipFlops, RAM) che possono essere interconnesse tra loro a nostro piacimento per svolgere funzioni complesse
- la configurazione viene specificata tramite un linguaggio di programmazione di alto livello Hardware Description Language

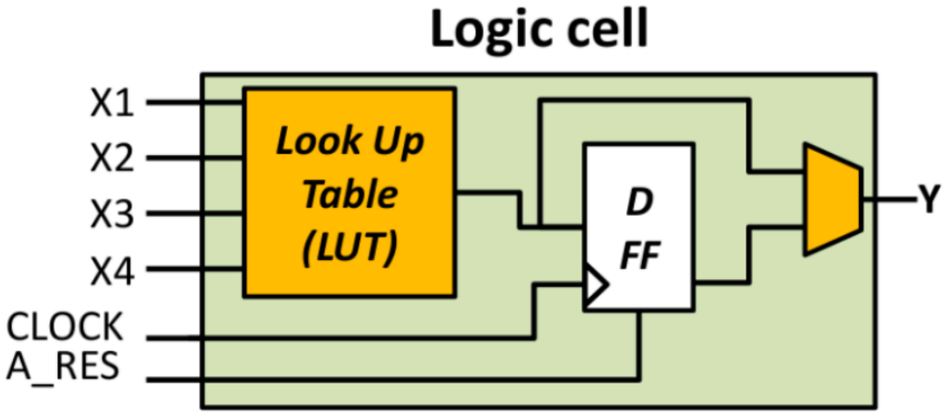
Available resources:

- **Digital signal processors (DSPs):** specialised units for multiplication and arithmetic
  Used a lot in DNNs and easily become scarce for big networks!



- **memory (BRAM)**
- **Logic cells/lookup tables (LUTs):** perform arbitrary functions (boolean ops, arithmetic, small memory)
- **flip-flops (FF):** registers data in time with clock pulse to keep OPs synchronised

ARCHITETTURA FPGA



General reference Cell model

Cells details can change for different vendors or FPGA models.

esempio di una cella logica

Look-Up Table

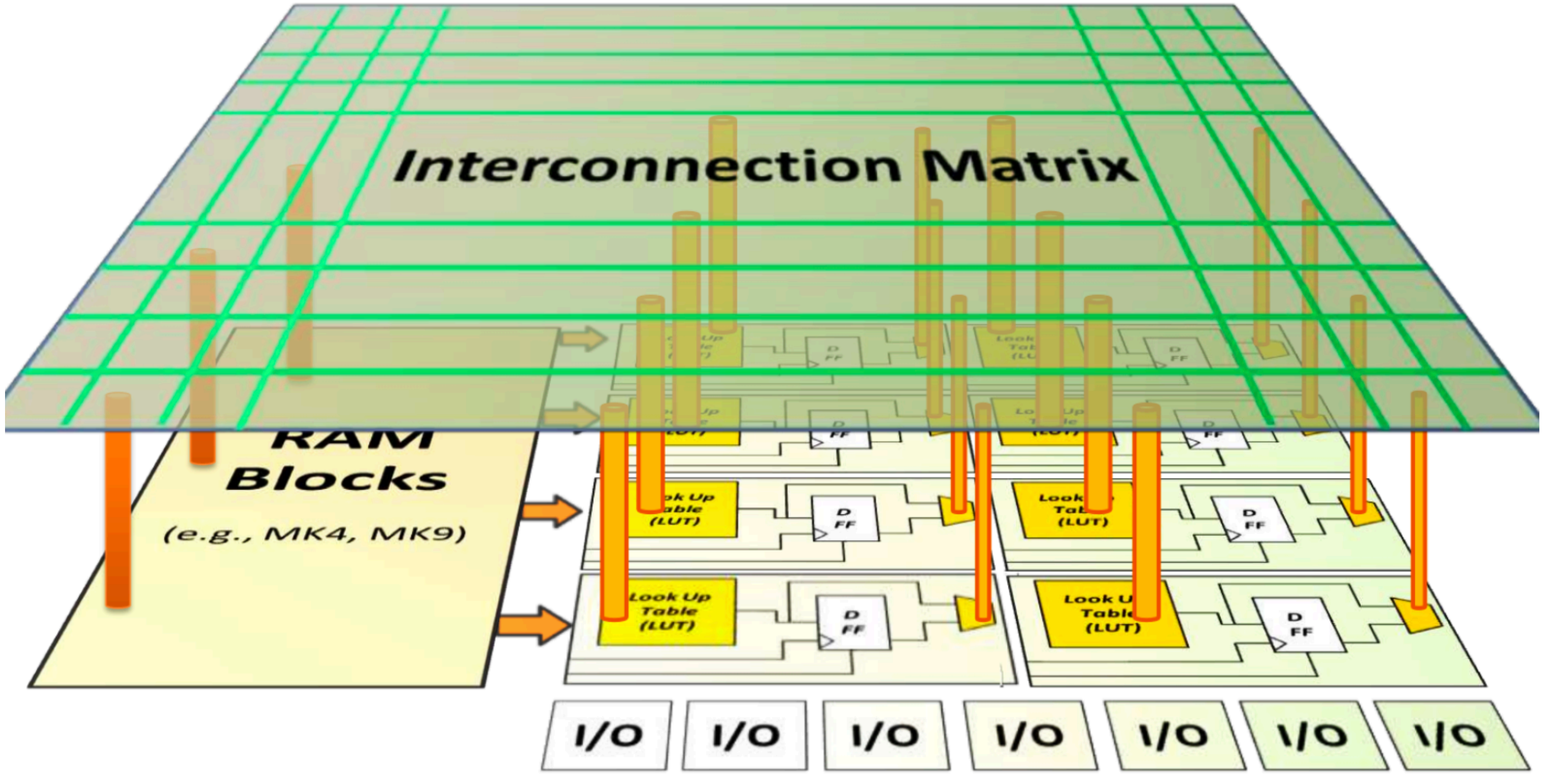
X1	X2	X3	X4	Y
0	0	0	1	?
0	0	1	0	?
...	...	...	...	?
1	1	1	1	?

A Cell usually contains:

- A **look-up table**: allow to map any combinatorial function of 4 inputs and 1 output.
- a **FF of type D**: to store persistent data.
- A **mux 2 -> 1**: to eventually bypass the FF for purely combinatorial cells.

Programmable element

ARCHITETTURA FPGA



interconnessione tra blocchi logici

VANTAGGI E SVANTAGGI FPGAs NEL COMPUTING

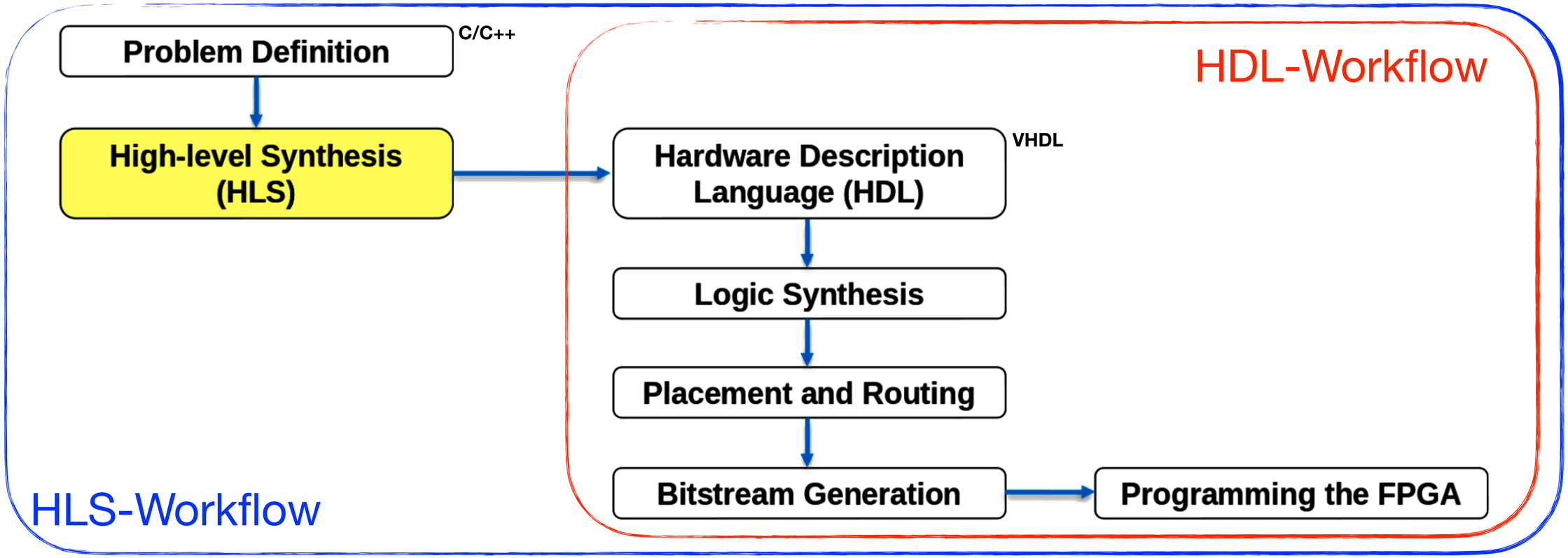
- uso delle FPGA nel calcolo in crescita:
  - ottimizzate per parallelismo massivo → potenziale guadagno in prestazioni
  - possono accelerare task per le quali i sistemi uni-processore non funzionano bene (ANN, DL)
  - possono gestire in modo efficiente tipi di dati non standard

CPU	GPU	FPGA
Fixed architecture	Fixed architecture	Adaptable Architecture
Predefined instruction set	Predefined instruction set	No fixed instruction set
Fixed memory hierarchy	Fixed memory hierarchy	Customizable memory hierarchy
Thread-level parallelism	SIMD parallelism	Excels at all types of parallelism

Difficoltà programmazione

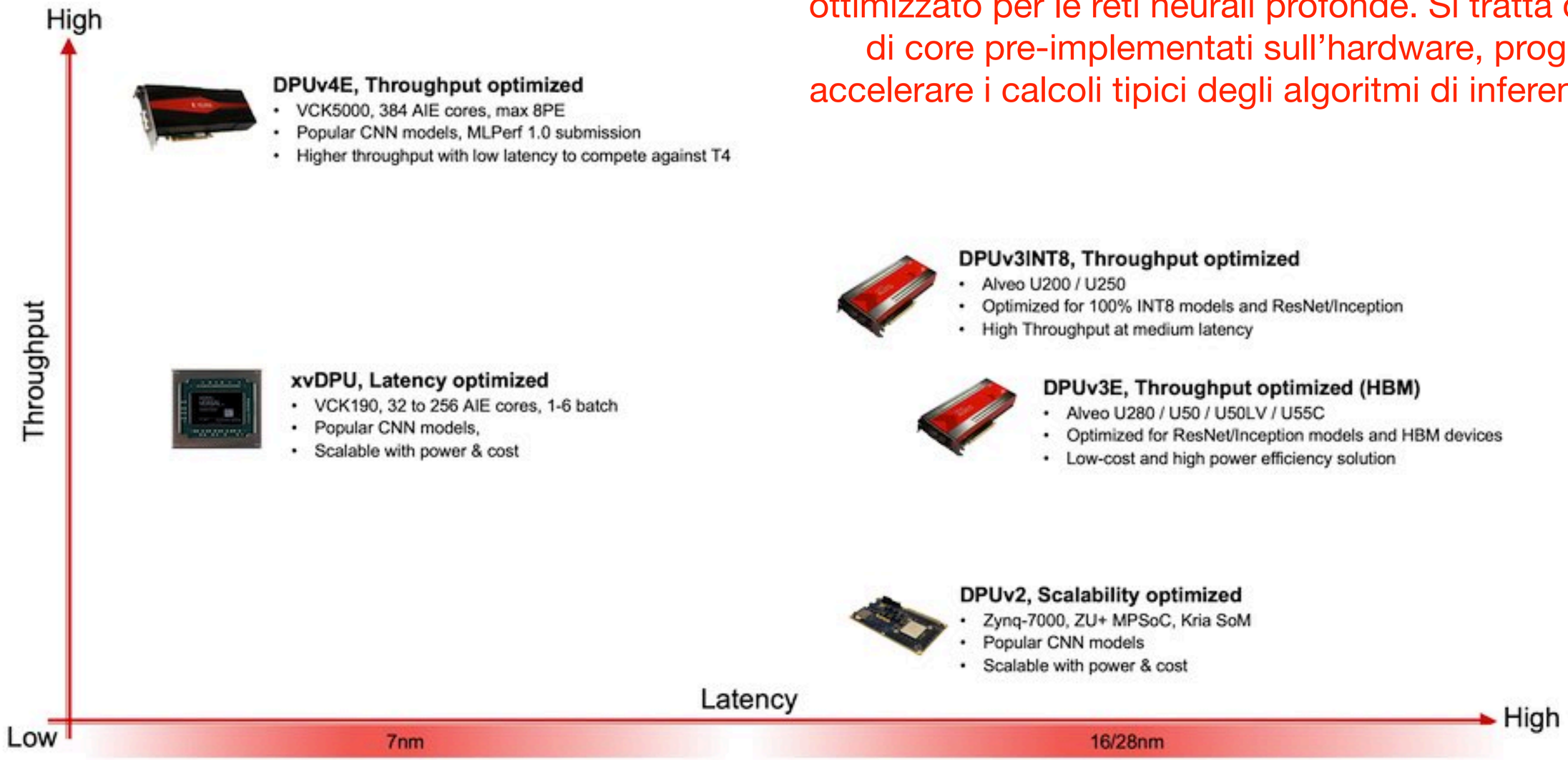
	CPU	GPU	FPGA (HLS)	FPGA (HDL)
Engineering time	short	medium	medium	Very long
Compilation time	short	short	Very long	Very long
Debugging	easy	medium	medium	hard
Maintainability	easy	medium	medium	hard

FPGA WORKFLOW PROGRAMMAZIONE



ACCELERATORI FPGA PER APPLICAZIONI A MEDIA LATENZA

DPU: DL processor unit, è un motore programmabile ottimizzato per le reti neurali profonde. Si tratta di un gruppo di core pre-implementati sull'hardware, progettati per accelerare i calcoli tipici degli algoritmi di inferenza nei DNN.

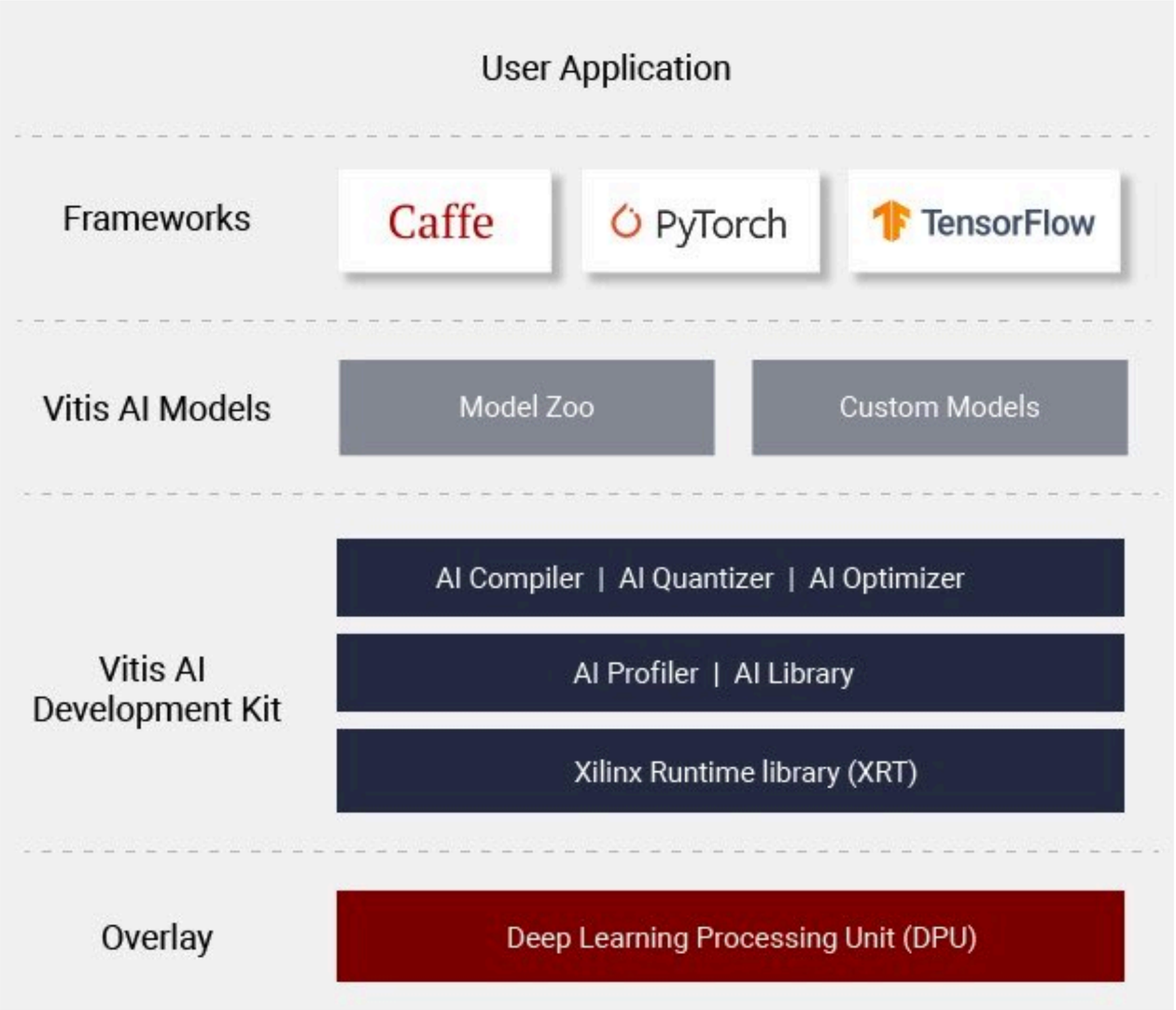


L’AMBIENTE DI SVILUPPO VitisAI

- Sviluppato da AMD-Xilinix per supportare il deployment rapido di reti neurali su acceleratori hardware Xilinx (Edge, Alveo, Acap)
  - https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html
- punti di forza:
  - relativamente semplice da utilizzare (la parte complicata è l’installazione degli acceleratori, lo sviluppo dei modelli è semplice anche se a volte risulta un pò involuto)
  - sfrutta bene le DPU degli acceleratori Xilinx
  - compatibile con tensorlfow/keras, pytorch, …
  - fornisce una libreria di modelli pre-addestrati e ottimizzati sui diversi hw utilizzabile per fine tuning sul proprio dataset
- limitazioni:
  - compressione dei modelli ottimizzata per applicazioni a media latenza ed alto throughput (non per design all-on-fpga)
    - latenze confrontabili con quelle ottenibili su GPU: 0.1-1 ms/inferenza, OK per trigger di alto livello non per trigger hw
  - non tutte le tipologie di layer e attivazioni neurali sono ancora state implementate → non tutte le architetture di ANN implementatili
  - alcune funzioni della libreria interessanti e utili non sono free (pruner, profiler)

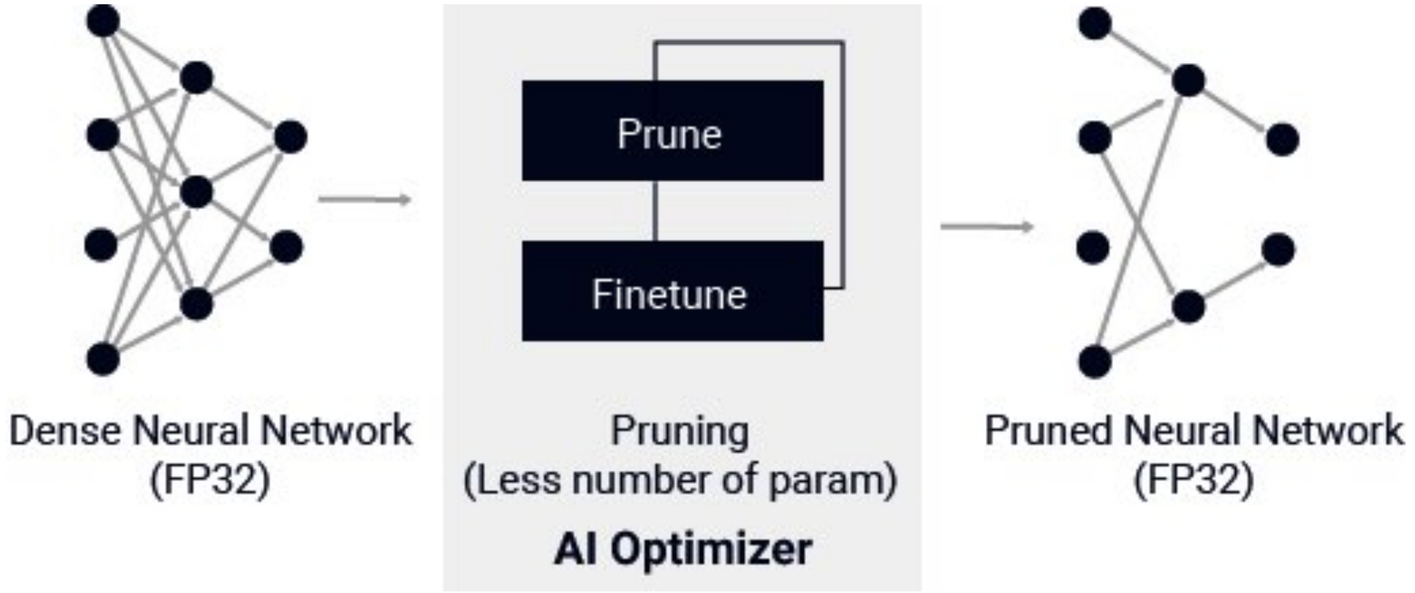
NOTA: per h/w intel/altera esiste un ambiente equivalente chiamato OpenVINO

VitisAI STACK



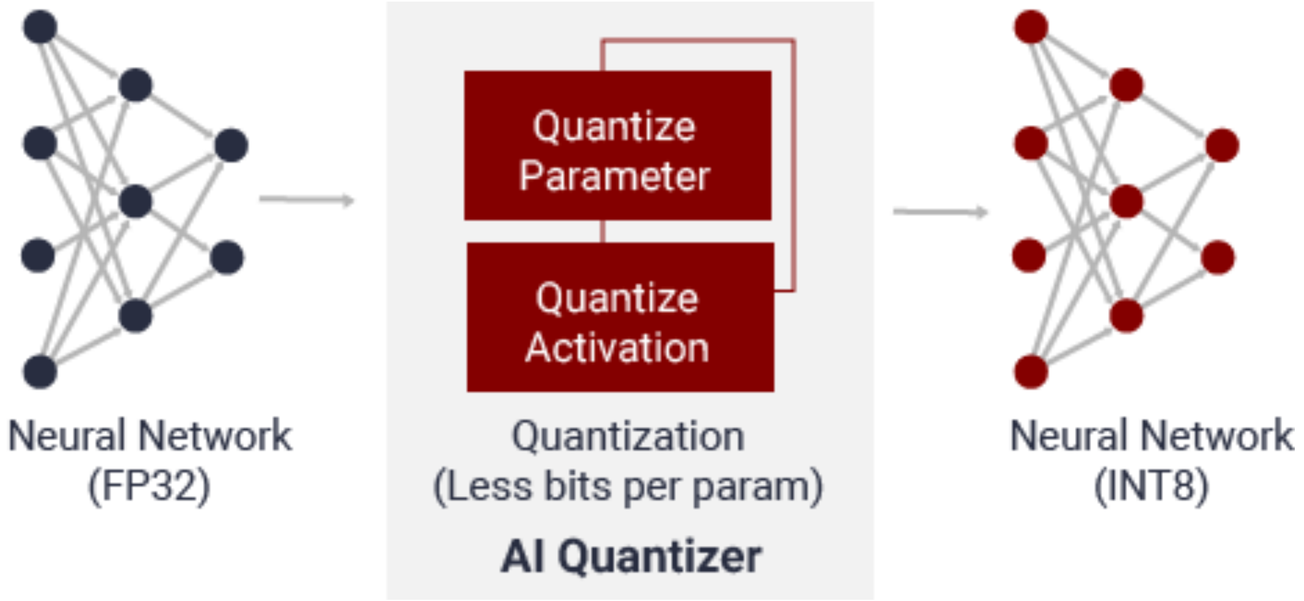
AI Optimiser

- viene chiamato un pò pomposamente Optimiser ma in realtà applica solo compressione via pruning
- disponibile solo nella versione non free di VitisAI, vedremo nell'hands-on come sia possibile ottenere gli stessi o migliori risultati usando le librerie native di tensorlfow …



AI Quantizer

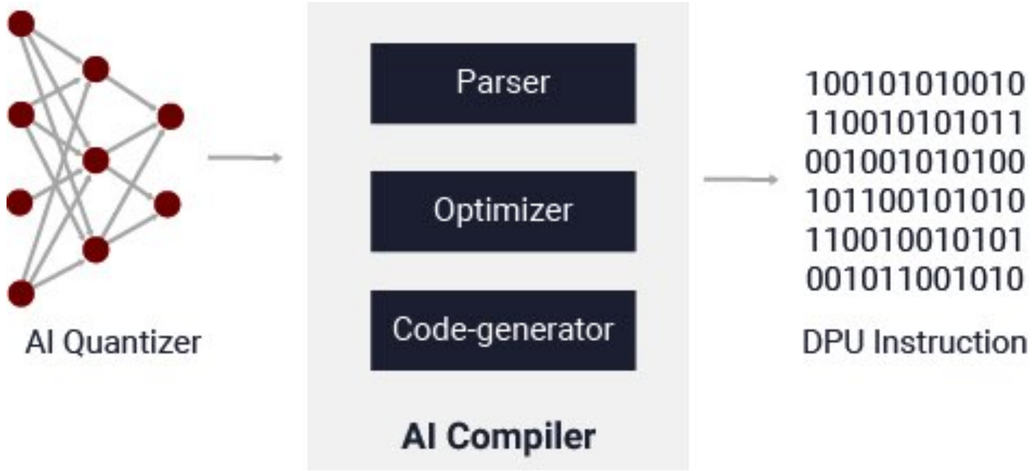
- converte i pesi delle reti e le attivazioni da floating point f32 a fixed-point INT8
- migliori risultati rispetto ad utilizzare le librerie tensorflow (inserisce nel modello quantizzato meta-informazioni che vengono sfruttate durante la compilazione del modello per lo specifico hardware)

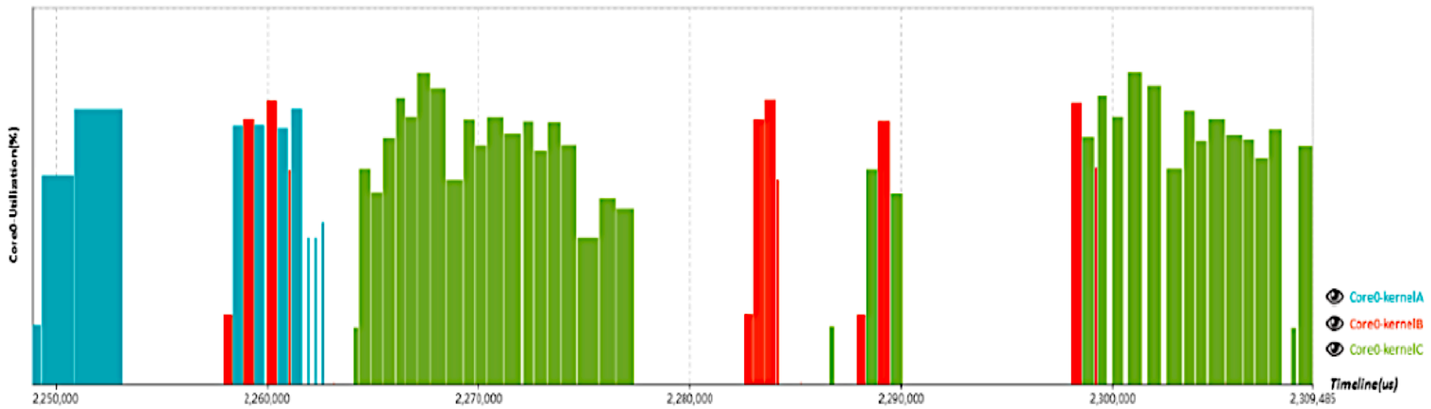


sia QAT che PQT

AI Compiler e Profiler

- compiler: mappa il modello quantizzato ad un set ottimizzato di istruzioni e data flow della DPU selezionata. Ottimizza le prestazioni e l'utilizzo delle risorse tramite layer fusion, scheduling delle istruzioni, on-chip memory reuse, etc…





- profiler (non free): permette di profilare l'utilizzo delle risorse e del data-flow sullo specifico hw

VitisAI Model Zoo

Rich Models from Tensorflow, Caffe and Pytorch

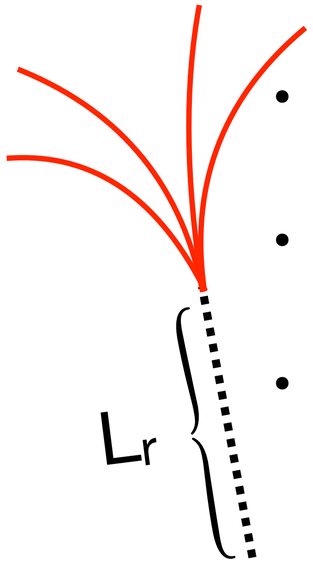
Open and Free on Github for All Developers

Advanced Optimization, Including Pruning, Applied

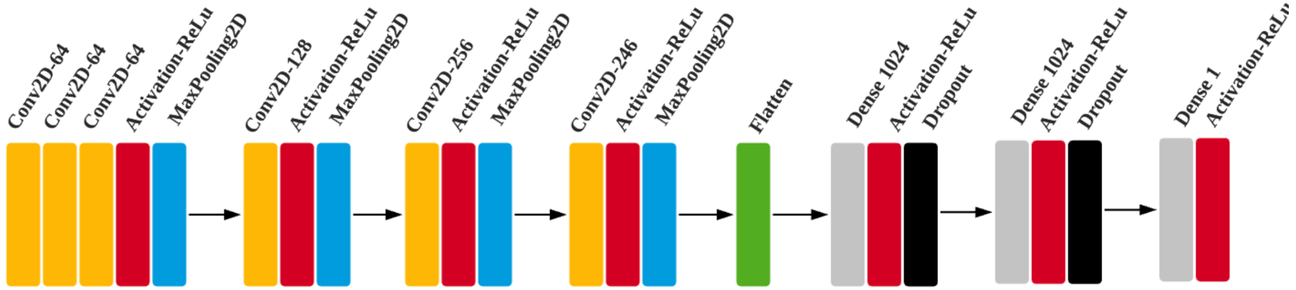
Retrainable with Custom Dataset

ESEMPIO USO: AI-ASSISTED MUON TRIGGERS FOR LLP

High Level Muon Trigger: modello DNN addestrato ad identificare particelle a lunga vita media a partire dei pattern di hit rilasciati in un rivelatore dello spettrometro muonico

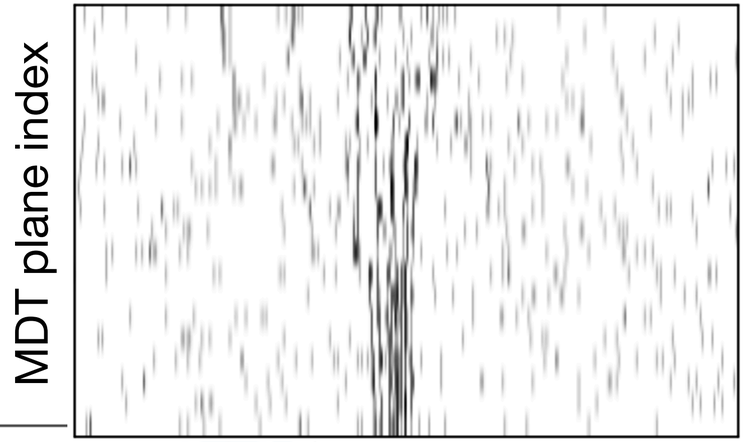


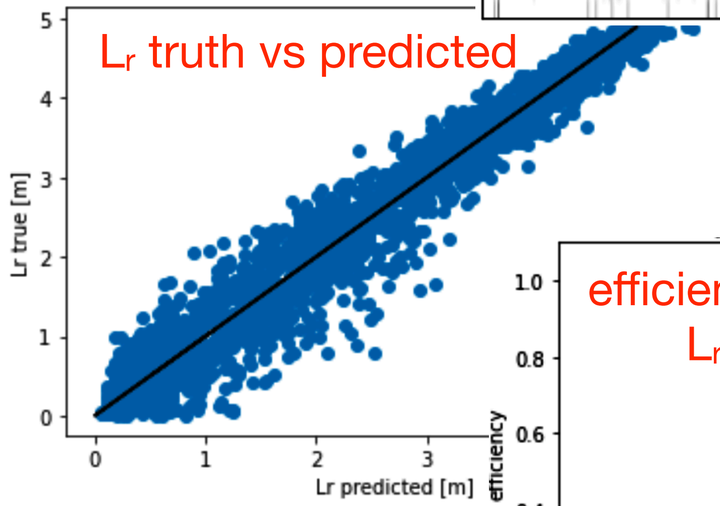
- benchmark: particelle neutre che decadono in multi-muoni a differenti distanze dal vertice primario di interazione (LLP)
- hits pattern nello spettrometro rappresentati come immagini binarie
- immagini usate per addestrare una CNN a predire la distanza di decadimento radiale (L_r) della LLP

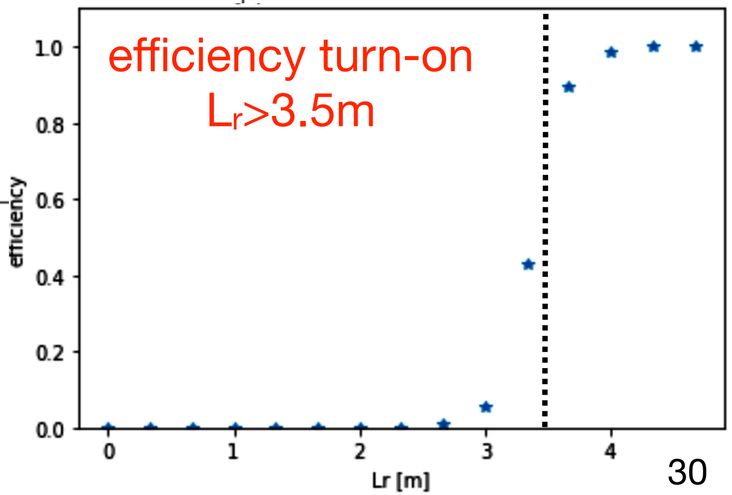


VGG Convolutional Neural Network architecture

example decay with 10 tracks in the MS

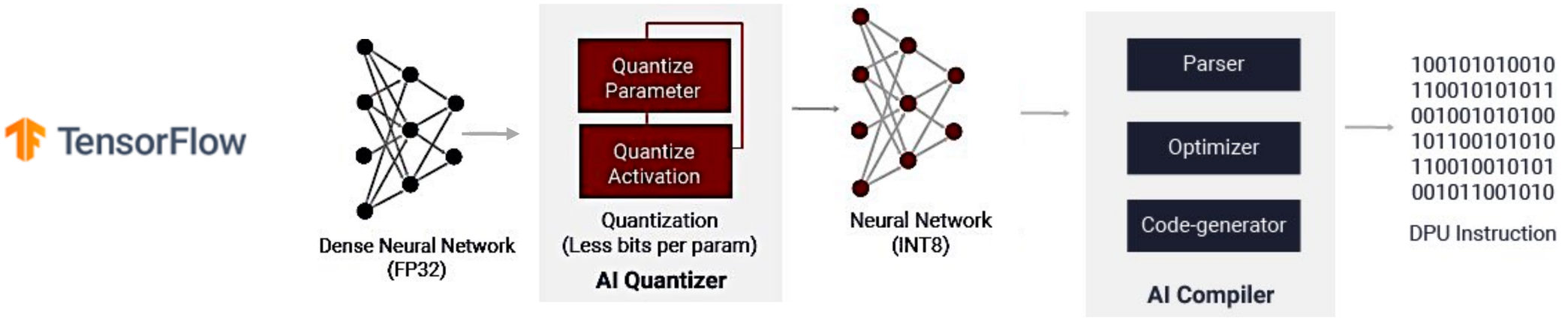






IMPLEMENTAZIONE SU ACCELERATORI FPGA ALVEO U50

modello addestrato con Tensorflow/keras, quantizzato e compilato con VitisAI



weight e activation quantisation a 8-bit
leverage FPGA DeepLearning processor
e ottimizza il memory transfer

compilazione
sulla FPGA

Memory occupancy (MB):	10.9	2.8

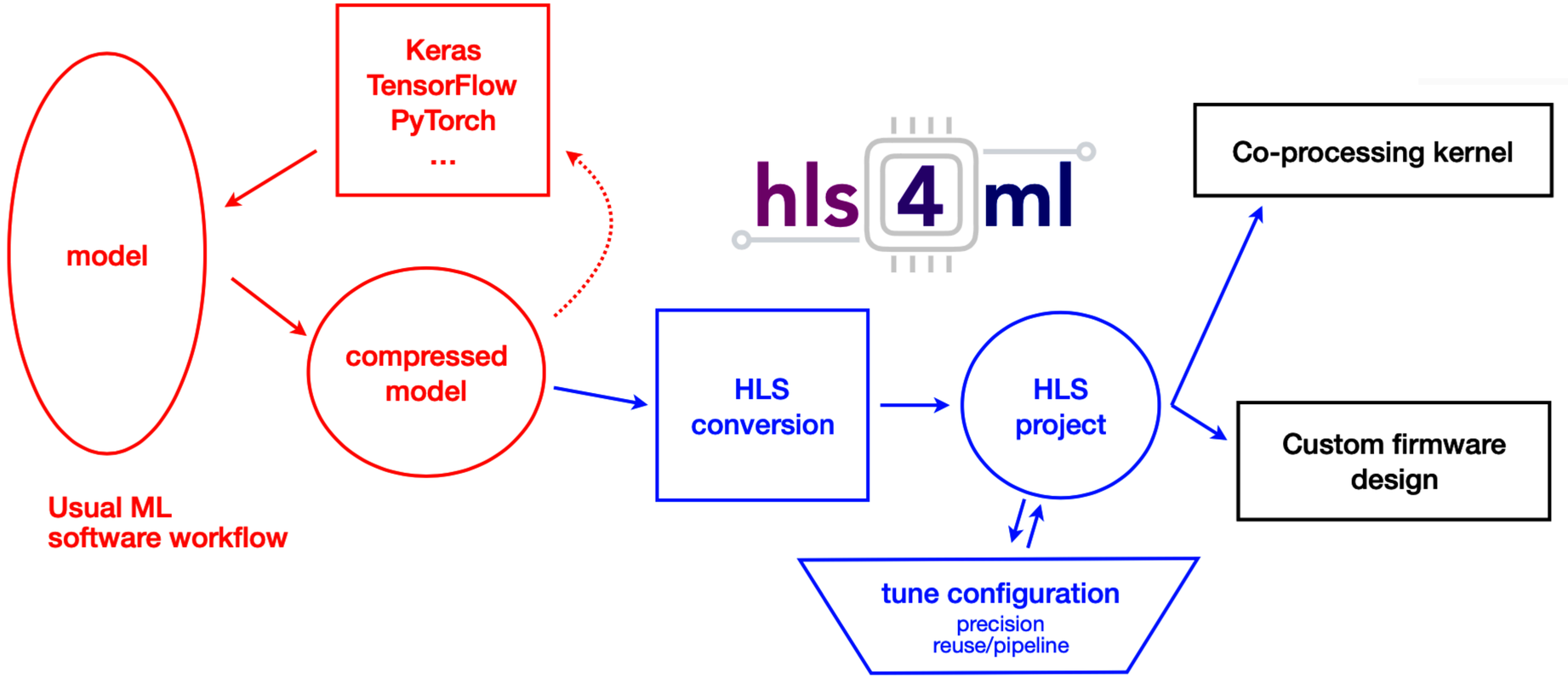
performances:
CPU/GPU FP32
FPGA INT8

	CPU Xeon E5-2698	GPU TESLA V100	FPGA ALVEO U50
predictions/sec	~350	~1 10^3	~2.5 10^3
power cons. [W]	135	250	75

~x7/x2.5 larger event rate
sustainable wrt CPUs/GPUs with
50/30% power consumption

HLS4ML

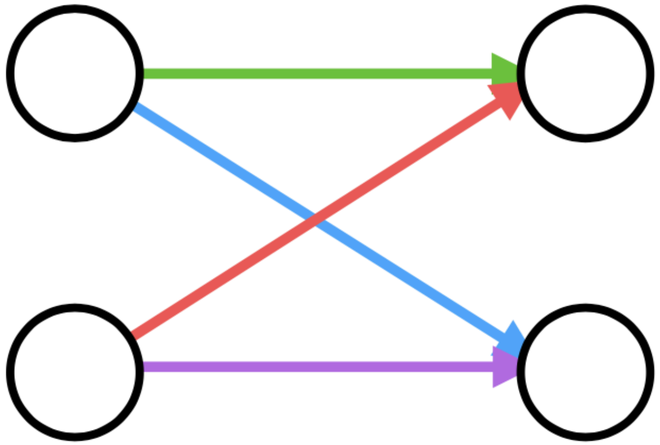
- https://github.com/fastmachinelearning/hls4ml
- libreria per la traslazione automatica di modelli DNN in HLS per FPGA Xilinx

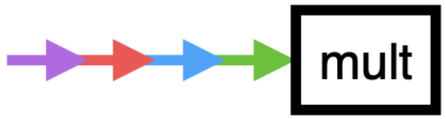


permette di controlare durante la sintesi il tradeoff tra utilizzo risorse e latenza/throughput tramite un “Reuse Factor” che controlla quanta parte delle operazioni parallelizzare vs quanta serializzare

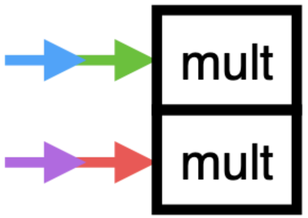
PARALLELIZZAZIONE VS SERIALIZZAZIONE

- **ReuseFactor**: how much to parallelize

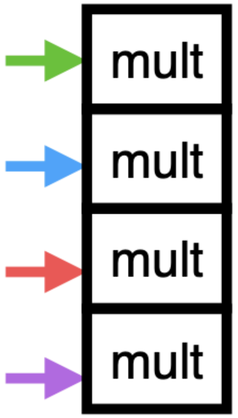




reuse = 4
use 1 multiplier 4 times



reuse = 2
use 2 multipliers 2 times each



reuse = 1
use 4 multipliers 1 time each

LIMITAZIONI (CORRENTI) DI HLS4ML

- CNN: input solo seriale (latenza cresce linearmente con la dimensione dell’input)
- non tutte le tipologie di layer, attivazioni etc. supportate
- con input grandi, modelli non semplici spesso fallisce la sintesi

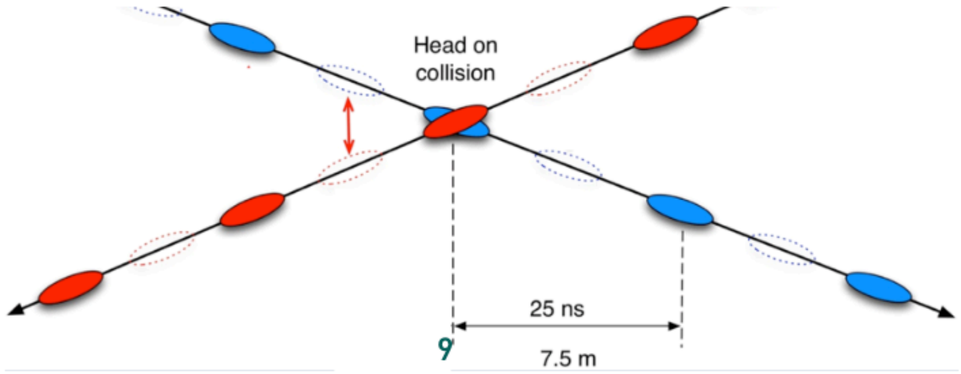
- OK per DNN e per CNN con architetture molto semplici e un numero di parametri limitato

DNN IMPLEMENTATE “ALL ON FPGA”

- vantaggi:
  - permettono un alto parallelismo e quindi consentono di implementare DNN con bassissima latenza (le componenti sequenziali della rete possono in ogni caso essere accelerate sfruttando il parallelismo tramite pipeline)
  - sono intrinsecamente deterministiche: la latenza nel processamento dei dati è ripetibile e predicibile (requirements per sistemi di trigger e online in esperimenti con acceleratori)
  - sono più efficienti delle GPU in termini di potenza consumata (migliore rapporto prestazioni/watt)

- prospettiva molto attraente per provare ad implementare un’algoritmo basato su DNN in uno dei trigger hw degli esperimenti LHC

es. ATLAS@HL-LHC
L0 trigger total decision time O(1μs)
latency per l’algoritmo di filtro: <400ns



DNN E RISORSE FPGA

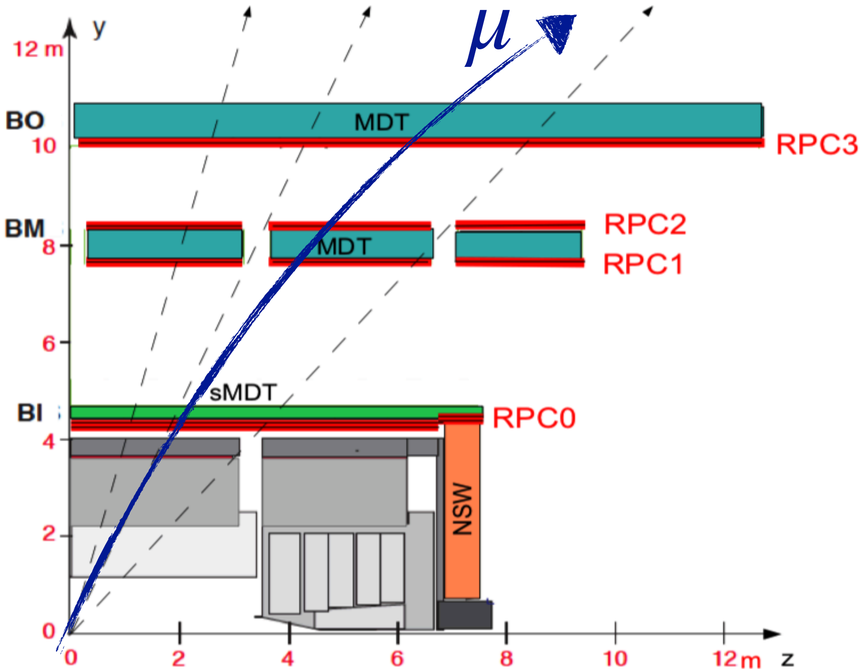
- l'inferenza con un ANN consiste sostanzialmente in operazioni di moltiplicazioni tra matrici che sono implementabili in modo molto efficiente nei DSP delle FPGA
- i DSP diventano la risorsa più preziosa e il vincolo principale quando si vuole sintetizzare il modello neurale nella FPGA
- le ultime generazioni di FPGA hanno 2k-12k DSP (ex. Xilinx Virtex Ultrascale+ XCV2P-12P)
- questo fissa la dimensione massima del modello implementabile (tenendo anche conto che una frazione non trascurabile delle risorse a disposizione deve essere necessariamente usata per le altre funzioni richieste dallo specifico trigger)

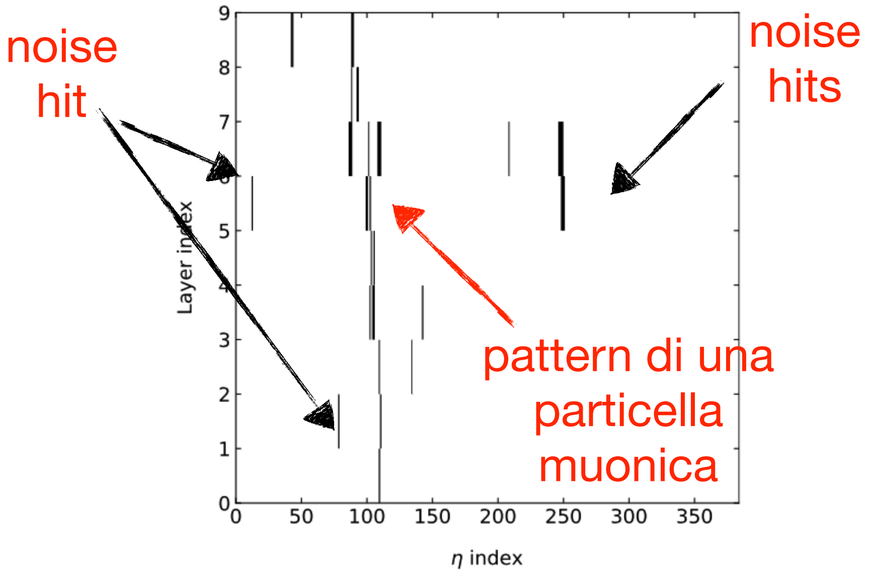
USE CASE: ULTRAFAST CNN SU FPGA PER IL TRIGGER DI L0 MUONICO DI ATLAS@HL-LHC

- Goal: ricostruire pt e pseudorapidità di tracce muoniche a partire degli hit nei rivelatori RPC **in meno di 400 ns** (3 ordini di grandezza più veloce dei più veloci modelli di AI su CPUe GPU)

competizione tra latenza e occupazione delle risorse in una FPGA, mentre le prestazioni di un DNN dipendono fortemente dalle sue dimensioni

Strategia: multi-stage **AI model compression** e semplificazione basata su **quantizzazione aggreessiva** + **knowledge distillation** per minimizzare la perdita di prestazioni



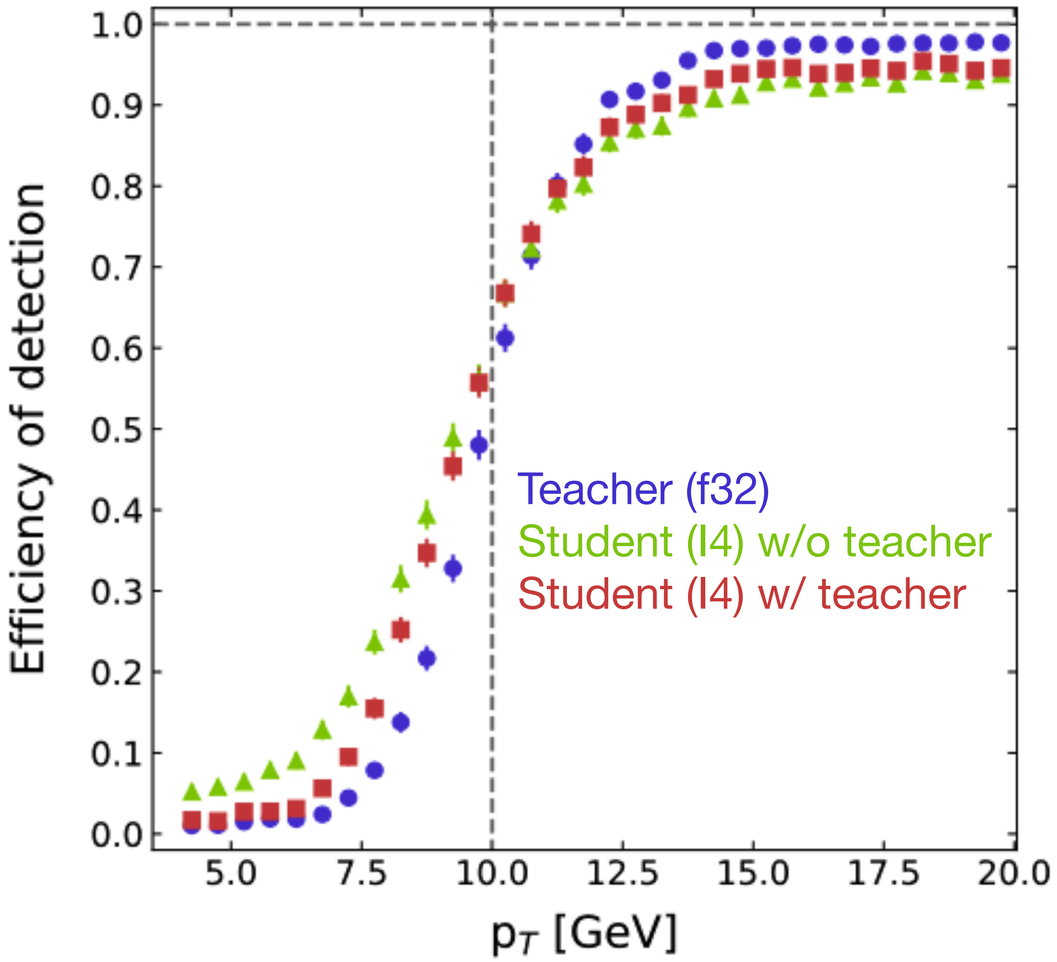


PIPELINE DI IMPLEMENTAZIONE DEL MODELLO

- progetto e ottimizzazione di un modello CNN tipo VGG tradizionale allenato al task di predire pt e pseudorapidità dei muoni (~80k parametri)
- semplificazione del numero di parametri del modello via frammentazione dell'input (sliding window che cerca muoni in patch ridotte dell'immagine originale) (~11k parametri)
- costruzione di un modello CNN semplificato (~700 parametri) e quantizzato (pesi e funzioni di attivazione, tranne output layer) fixed point a 4 bit (I4)
- training del modello semplificato via teacher-student knowledge-transfer
- conversione del modello addestrato in linguaggio VHDL (due strategie testate):
    - hls4ml (modello TF/keras → C++) + VivadoHLS (C++ → VHDL)
    - HLD: scrittura del modello di CNN addestrato direttamente VHDL
- sintesi e generazione del firmware per la FPGA con Vivado

PRESTAZIONI PRELIMINARI

curva efficienza single muon trigger
soglia nominale pt 10 GeV



FPGA resource occupation

Table 3 Percentage occupancy relative to the total FPGA available resources (model xcvu13p-fhga2104-2L-e [14])

Model (9 × 16)	BRAM	DSPs	FF	LUT
Teacher (%)	20.9 %	258.0 %	69.4 %	15.3 %
Student 32 bit (%)	3.2	31.0	8.4	2.7
QStudent 4 bit (%)	0.2	0.05	0.4	1.7

Inference time per event on FPGA
Xilinx Ultrascale+ XCV13P

- Teacher fp32: 5 ms (Tesla V100 GPU)
- Student 4 bit: 438 ns (hls4ml implementation)
- Student 4 bit: 84 ns (our VHDL implementation)

S. Francescato, S.Giagu, F. Riti, G.Russo, L.Sabetta, F.Tortonesi, Eur. Phys. J. C (2021) 81:969

PRESTAZIONI VS LIVELLO DI QUANTIZZAZIONE