MACHINE (DEEP) LEARNING IN SISTEMI DI TRIGGER

Stefano Giagu
INFN Sezione di Napoli - 21-22.4.2022

PROGRAMMA DEL CORSO

- oggi:
    - **mattina:** richiami elementi essenziali ANN, DNN in sistemi di trigger, tecniche di compressione di reti neurali, acceleratori FPGA per infrenza AI a media letenza, l’ambiente VitisAI
    - **pomeriggio:** (hands-on) training di un modello CNN per un trigger a media latenza su acceleratore FPGA: implementazione e training del modello non compresso, compressione tramite pruning (con tensorflow) e quantizzazione fixed-point INT8 (con tensorflow e con VitisAI) del modello, compilazione su acceleratore Xilinix Alveo U50 con VitisAI
- domani:
    - **mattina:** tecniche di compressione di reti neurali (2nda parte), inferenza AI ultrarapida su FPGA, la libreria hls4ml e le sue limitazioni, use-case applicativo: trigger hw muonico per l’upgrade di fase2 dell’esperimento ATLAS
    - **pomeriggio:** (hands-on) training di un modello CNN per la ricostruzione a bassissima latenza di muoni nel trigger di livello-0 dell’esperimento ATLAS, quantizzazione estrema con Qkeras, distillazione del modello con teacher-student knowledge-distillation, sintesi del modello con hls4ml vs implementazione custom CNN in VHDL

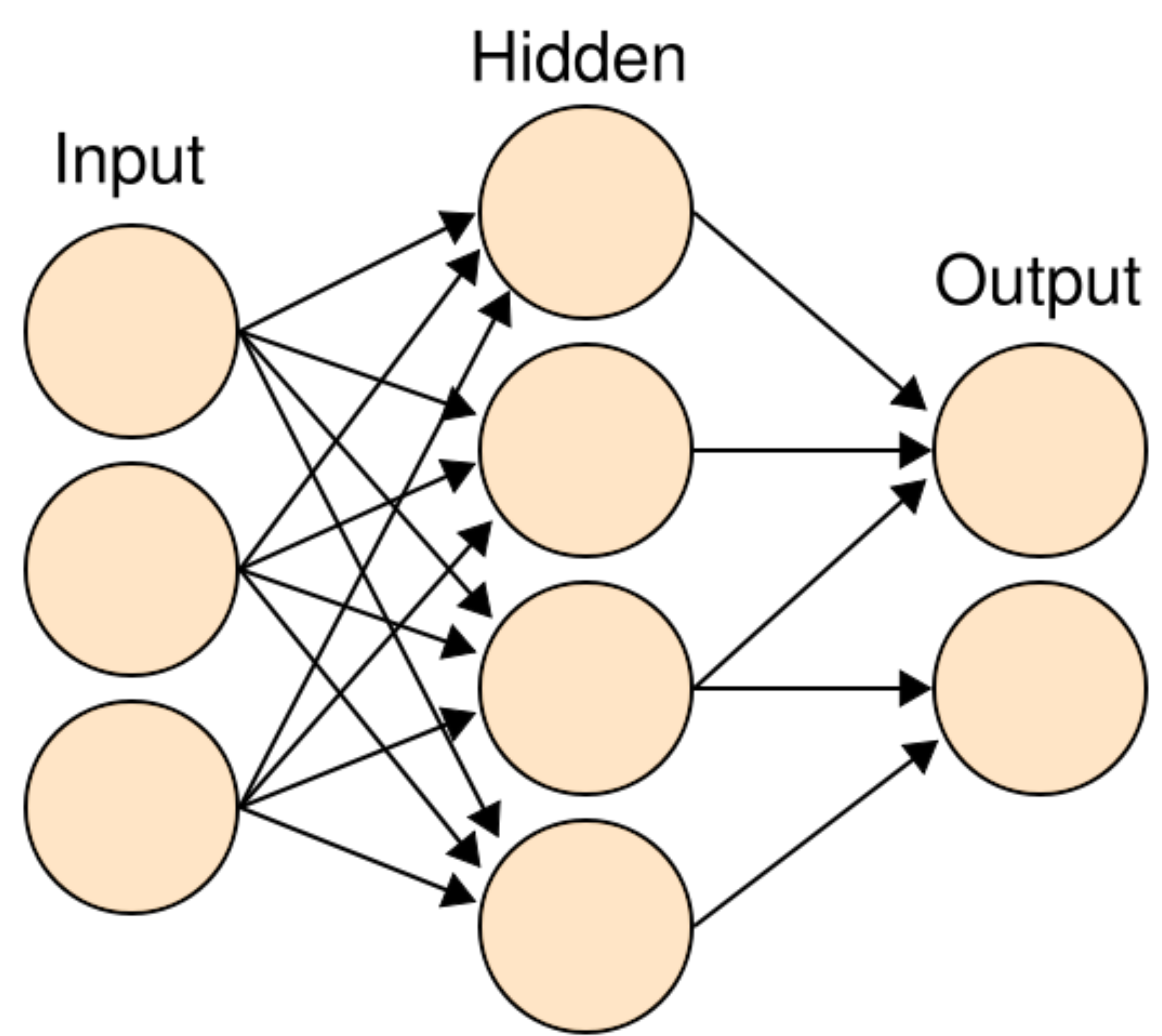
RICHIAMI ANN

ARTIFICIAL NEURAL NETWORKS

- costituiscono l'esempio più noto di modelli non-lineari e sono alla base delle moderne tecniche di DL
- ANN: modello matematico in grado di approssimare con grande precisione forme funzionali del tipo:

$$f : R^n \to R^m : y = f(x) \longrightarrow \text{ANN F: } \hat{y} = \text{F}(x)$$

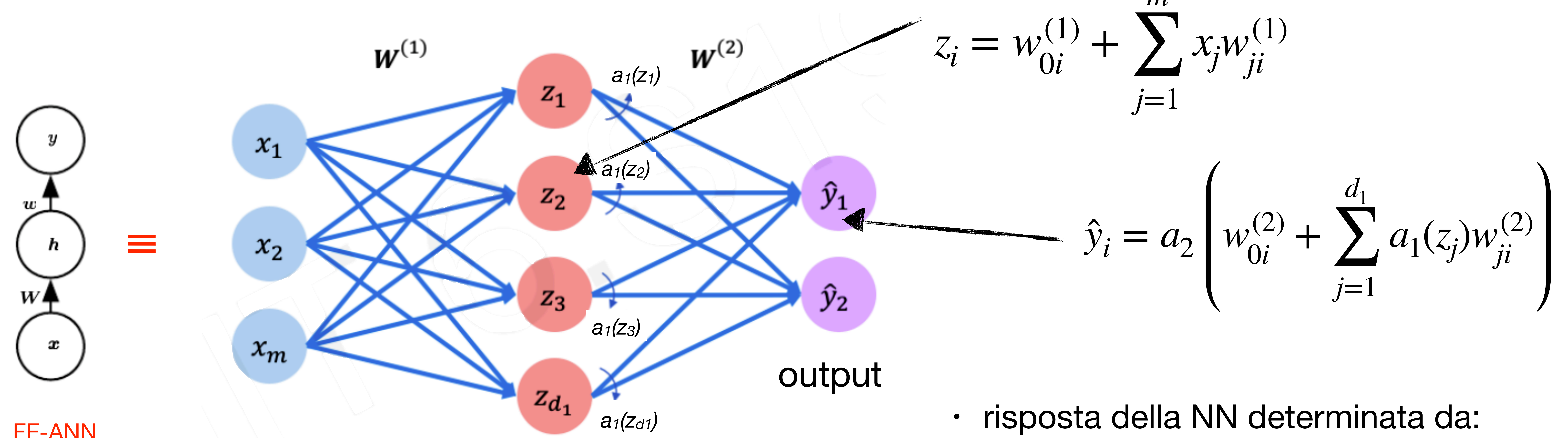
  - introdotto in analogia con le reti neurali biologiche, in modo più formale e preciso un DNN è una composizione di funzioni connesse in catene descritte da grafi (es. Feed-Forward NN possono essere rappresentati come grafi diretti aciclici)



- architettura: gruppo interconnesso fi unità di calcolo semplici (neuroni)
- analizza le informazioni in input con un approccio di tipo connessionista → azioni eseguite in modo collettivo ed in parallelo dai neuroni
- apprendimento: si comporta come un sistema adattivo, modifica la struttura basandosi sull'insieme di informazioni che fluiscono attraverso la rete durante la fase di apprendimento

MULTILAYER PERCEPTRONS FEED-FORWARD NN

- l'architettura base per le ANN
  - neuroni organizzatu in layers: input, hidden-1, ... , hidden-K, output
  - possibili solo connessioni dei neuroni di un dato layer verso il successivo: direct acyclic graph
  - sono presenti tutte le possibili connessioni (layer densi)

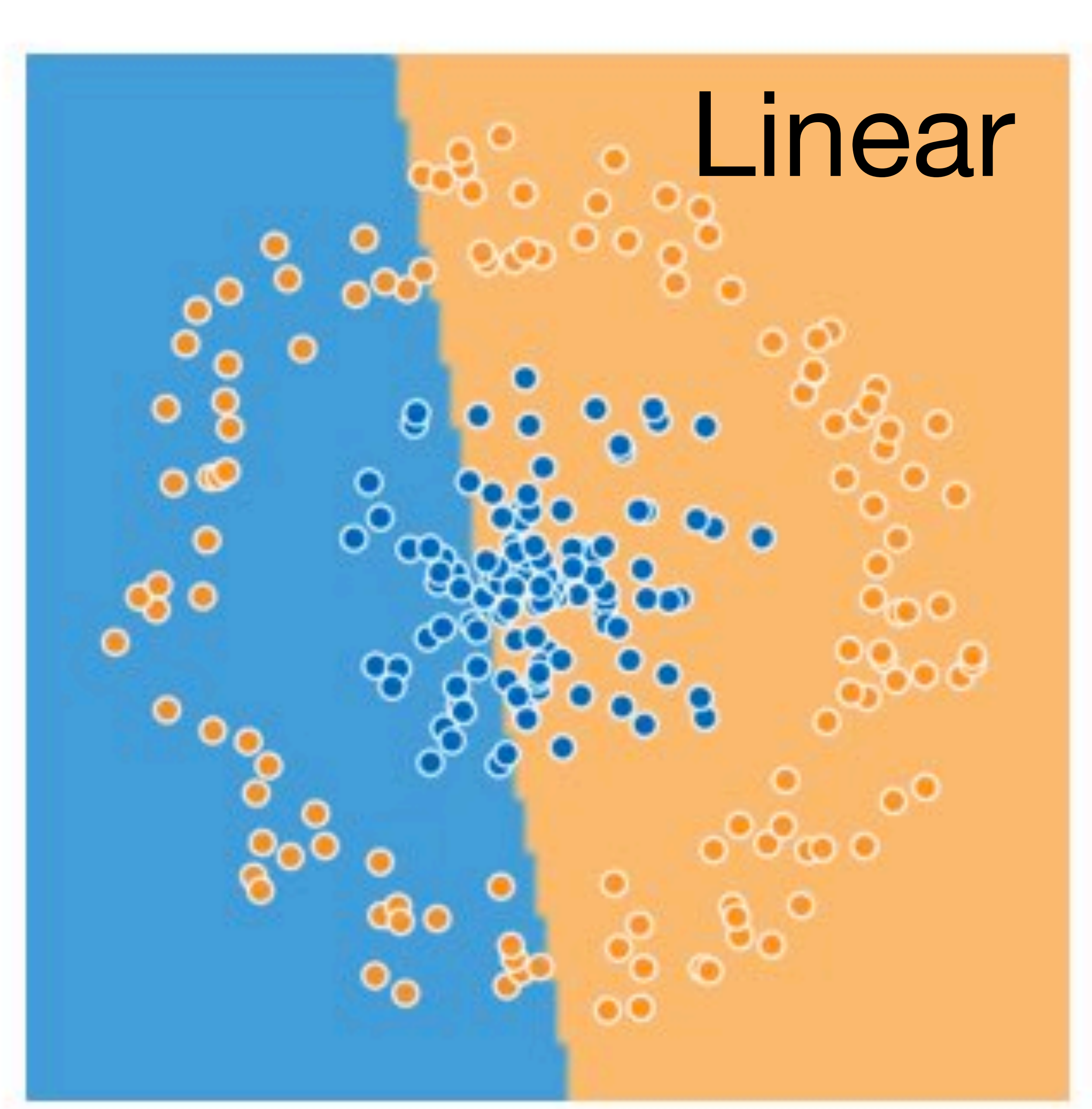


$$z_i = w_{0i}^{(1)} + \sum_{j=1}^{m} x_j w_{ji}^{(1)}$$

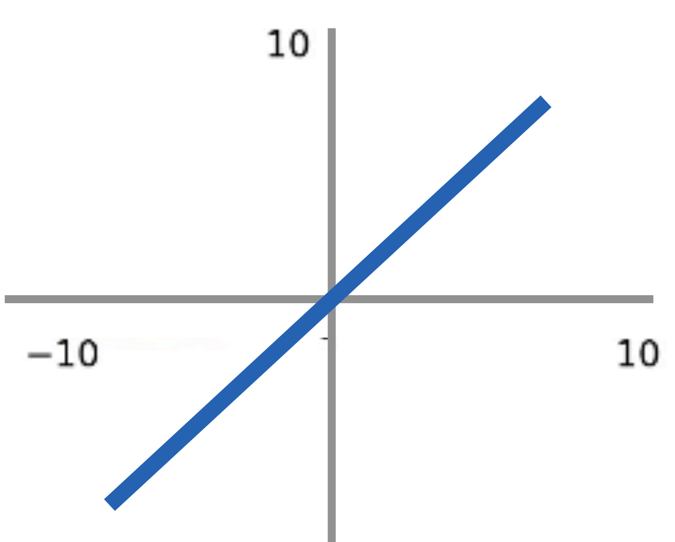
$$\hat{y}_i = a_2\left(w_{0i}^{(2)} + \sum_{j=1}^{d_1} a_1(z_j) w_{ji}^{(2)} \right)$$

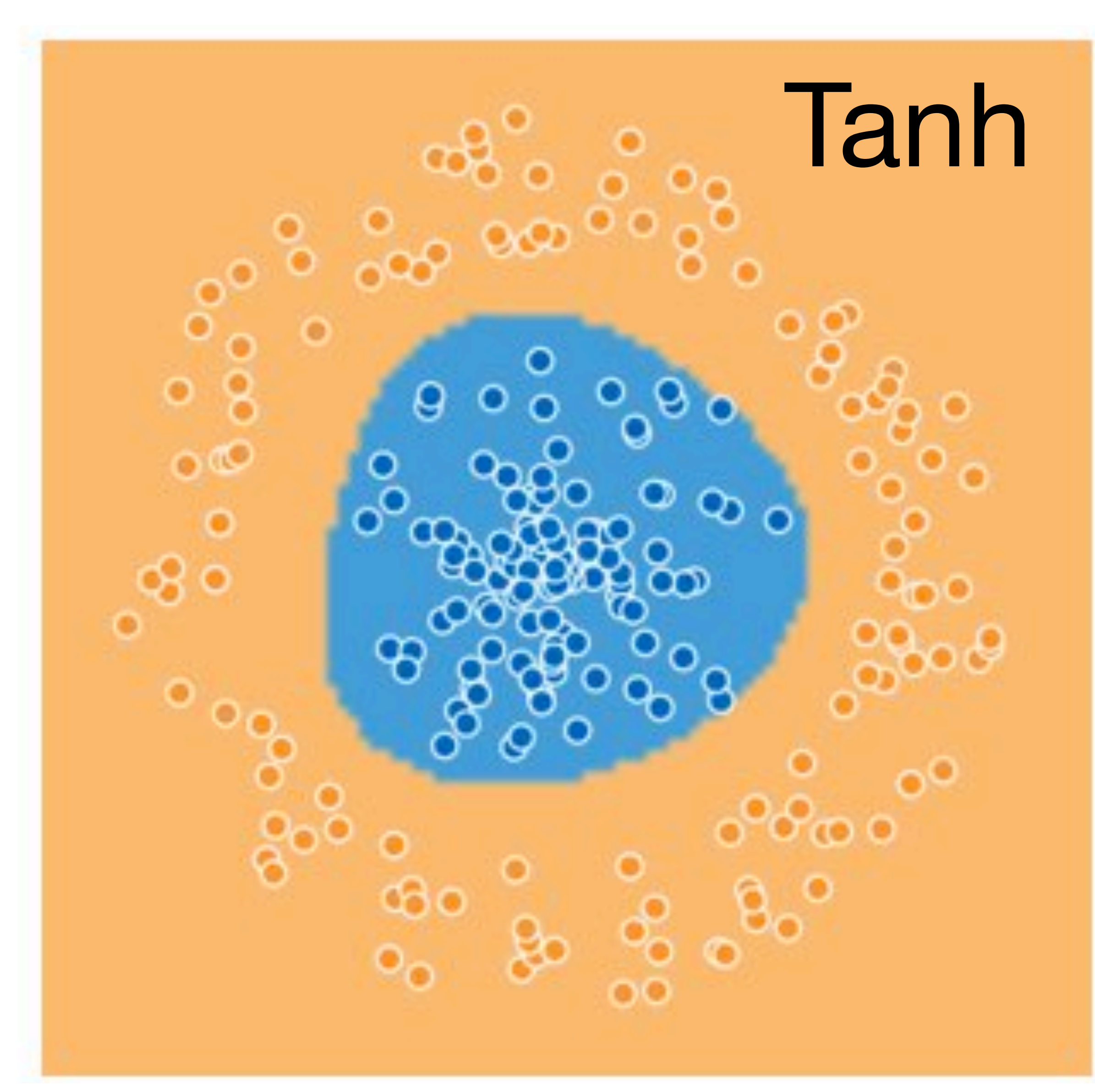
- risposta della NN determinata da:
  - weights w_{ij}
  - network topology (#layers, size of each layer, ...)
  - activation function of each layer

attivazioni non-lineare permettono di apprendere pattern complessi e non lineari …

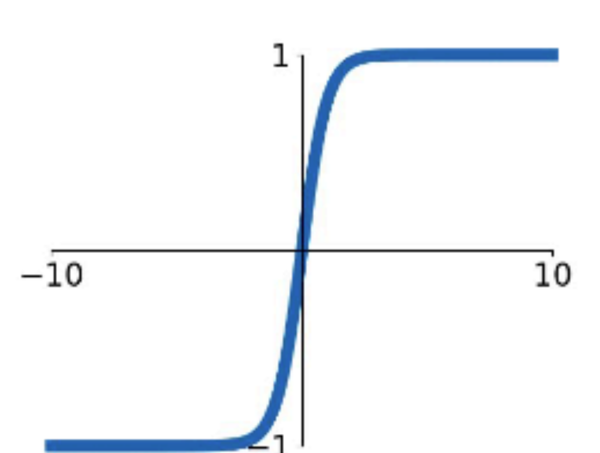


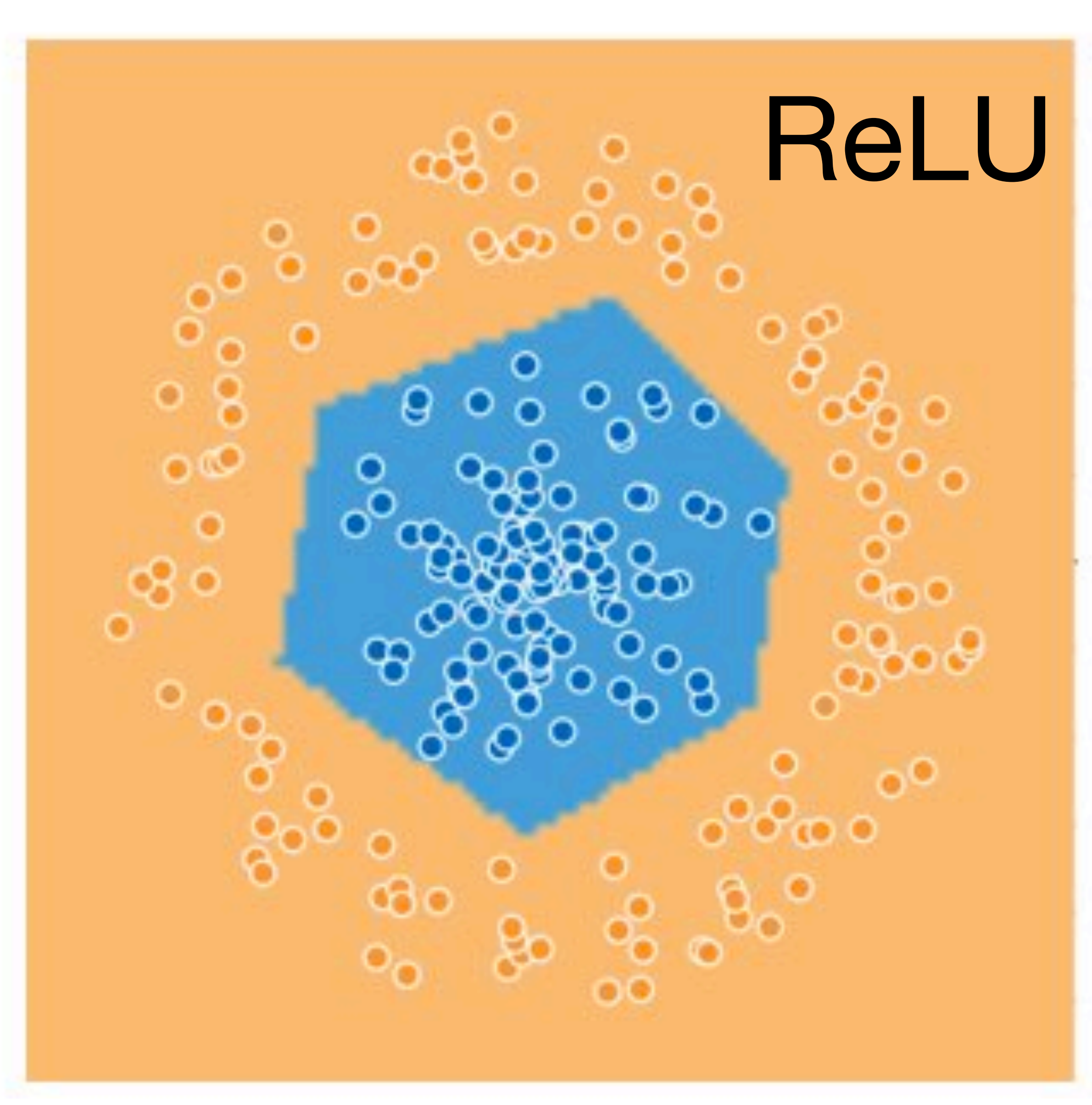
$$a(z) = z$$



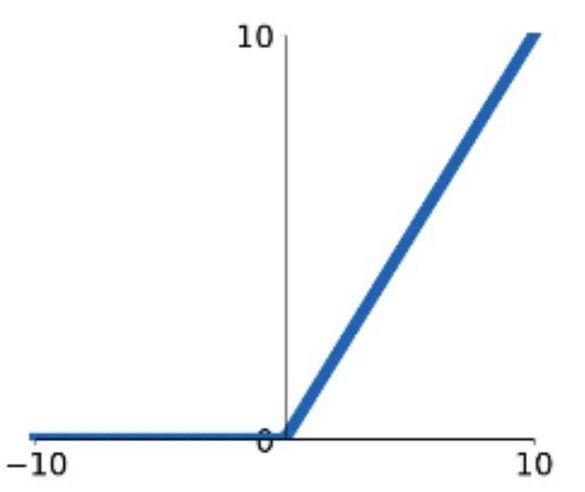


$$a(z) = \tanh[z]$$





$$a(z) = \max[0,z]$$



TRAINING

- addestrare la rete consiste nell'aggiustare il valore dei pesi (e di tutti gli altri iperparametri) in accordo ad una data loss function che misura le prestazioni del modello rispetto ad uno specifico compito
- tecnica più utilizzata per ottimizzare i parametri: discesa lungo il gradiente con back-propagation
- ottimizzazione degli iperparametri: tipicamente tramite approcci euristici (autoML: grid o random search, bayesian-opt, …)

n_h: numero di neuroni nel layer hidden

Output per un ANN con:
- un singolo hidden layer con att: tanh
- un output layer con att: lineare
- nessun bias

navi: numero di neuroni nel layer di input

$$\hat{y} = \sum_{j=1}^{n_h} \tanh[z_j]\, w_{j1}^{(2)} = \sum_{j=1}^{n_h} \tanh[\sum_{i=1}^{n_{var}} x_i w_{ij}^{(1)}]\, w_{j1}^{(2)}$$

peso associato alla connessione tra il j-esimo neurone del hidden layer e il neurone di output

peso associato alla connessione tra l'i-esimo neurone del input layer e il j-esimo neurone del hidden layer

NOTA: il modello matematicamente corrisponde ad una composizione di funzioni: $f(x) = f^{(2)}(f^{(1)}(x))$ con $f^{(i)}$ i-esimo layer … per un deep-NN con d-layer nascosti: $f(x) = f^{(d+1)}(\cdots f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(x)))))$

CALCOLO DEI GRADIENTI DURANTE IL TRAINING

- durante il training vengono presentati alla rete N esempi: $T\{x^{(i)}\}$ (i=1,…,N)
- i pesi della rete vengono inizializzati a valori random (piccoli e intorno a zero): ~N(0,σ)
- per ogni evento viene calcolato l'output del modello $\hat{y}(x^{(i)})$ e confrontato con il target atteso $y^{(i)}$ tramite una opportuna funzione di perdita che misura la "distanza" tra $\hat{y}(x^{(i)})$ e $y^{(i)}$:

Esempio MSE

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} L_i\left(y^{(i)}, \hat{y}^{(i)}(x^{(i)};\mathbf{w})\right)$$

$$L_i\left(y^{(i)}, \hat{y}^{(i)}(x^{(i)};\mathbf{w})\right) = \frac{1}{2}\left(y^{(i)} - \hat{y}^{(i)}(x^{(i)};\mathbf{w})\right)^2$$

- il vettore di pesi viene scelto come quello che minimizza L: $\mathbf{w}^* = \underset{\mathbf{w}}{argmin}[L(\mathbf{w})]$
- il minimo viene cercato con tecniche di GD/SGD …

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta\nabla_w L(T|\mathbf{w})$$

per aggiornare i pesi di tutti i layer della rete è necessario calcolare il gradiente di funzioni complicate per ogni neurone della rete e di valutarne il valore numerico ⇒

procedura di Back-Propagation

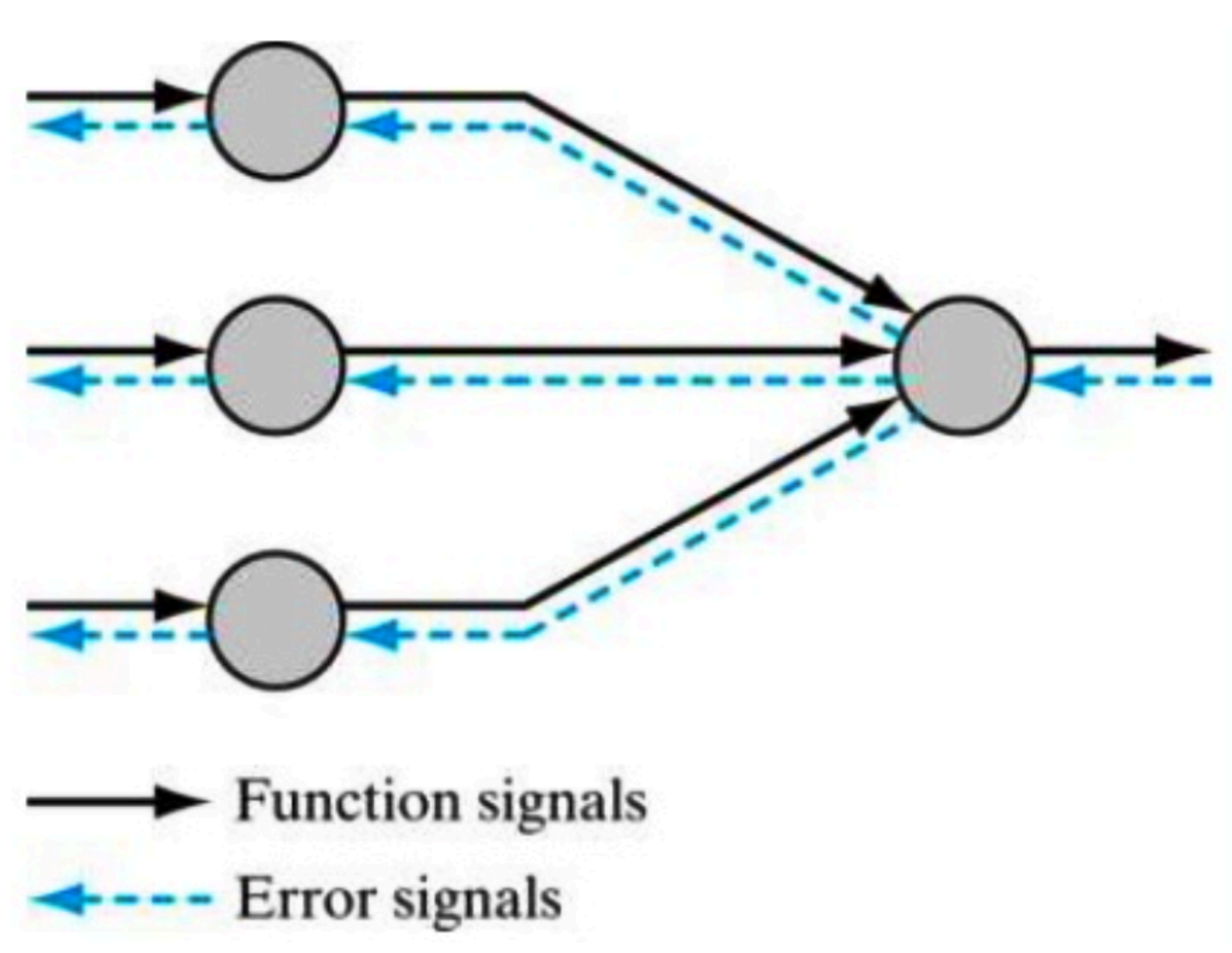
```
import tensorflow as tf

weights = tf.Variable([tf.random.normal()])

while True:
  with tf.GradientTape() as g:
    loss = compute_loss(weights)
    gradient = g.gradient(loss, weights)
  weights = weights - lr * gradient
```

BACK-PROPAGATION

- il training di un NN avviene in due fasi distinte che si ripetono ad ogni iterazione:
  - Forward phase: i pesi sono fissati e il vettore di input viene propagato layer per layer fino ai neuroni di output (function signal)
  - Backward phase: viene calcolato l'errore Δ confrontando l'output con il target e il risultato viene propagato indietro, ancora layer per layer (error signal)

- ogni neurone (hidden o di output) riceve e confronta i segnali di funzione e di errore

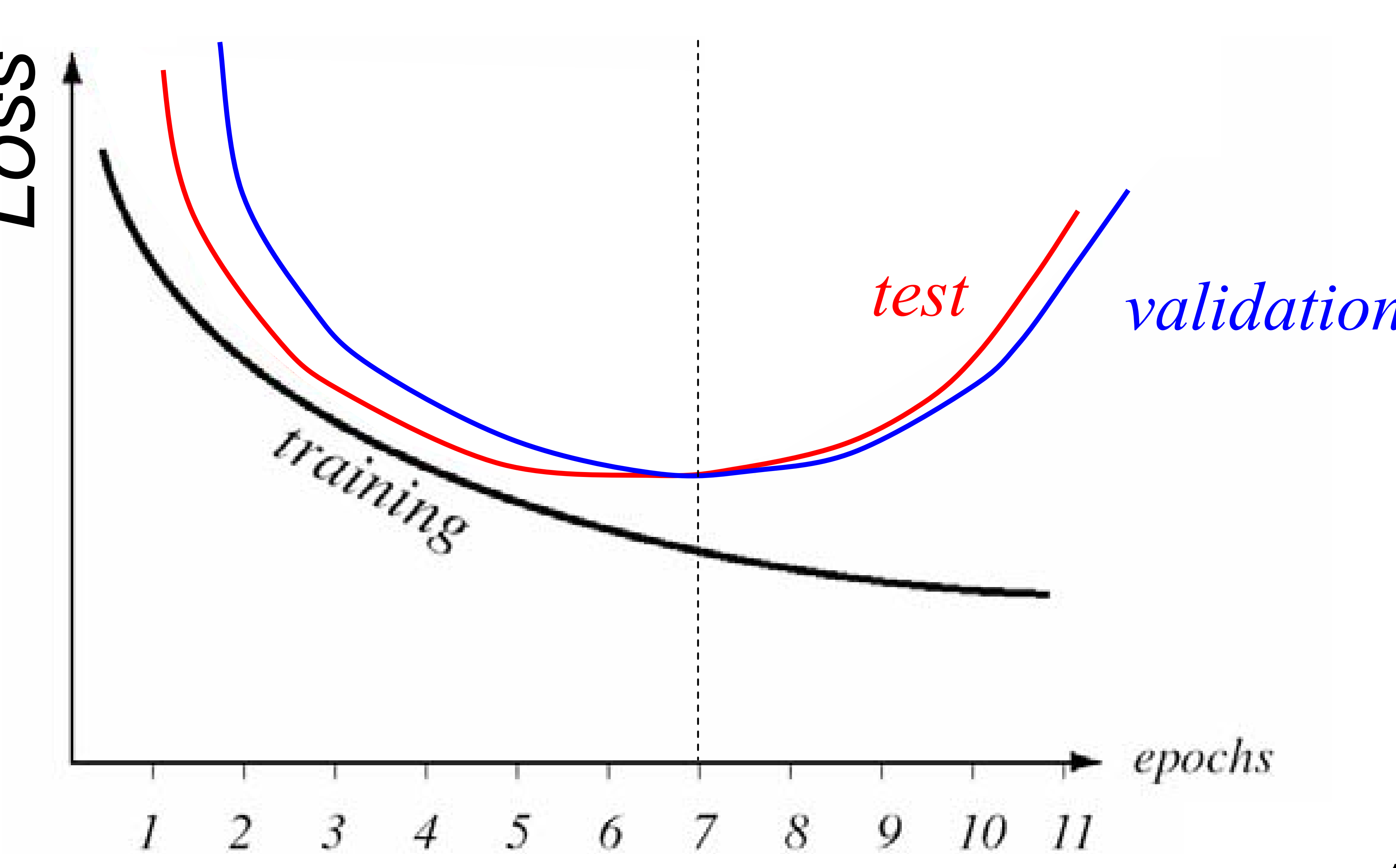
- la back-propagation consiste in una semplificazione del calcolo del gradiente ottenuta applicando in modo ricorsivo la regola di derivazione di funzioni composte (chain rule)



CURVE DI APPRENDIMENTO

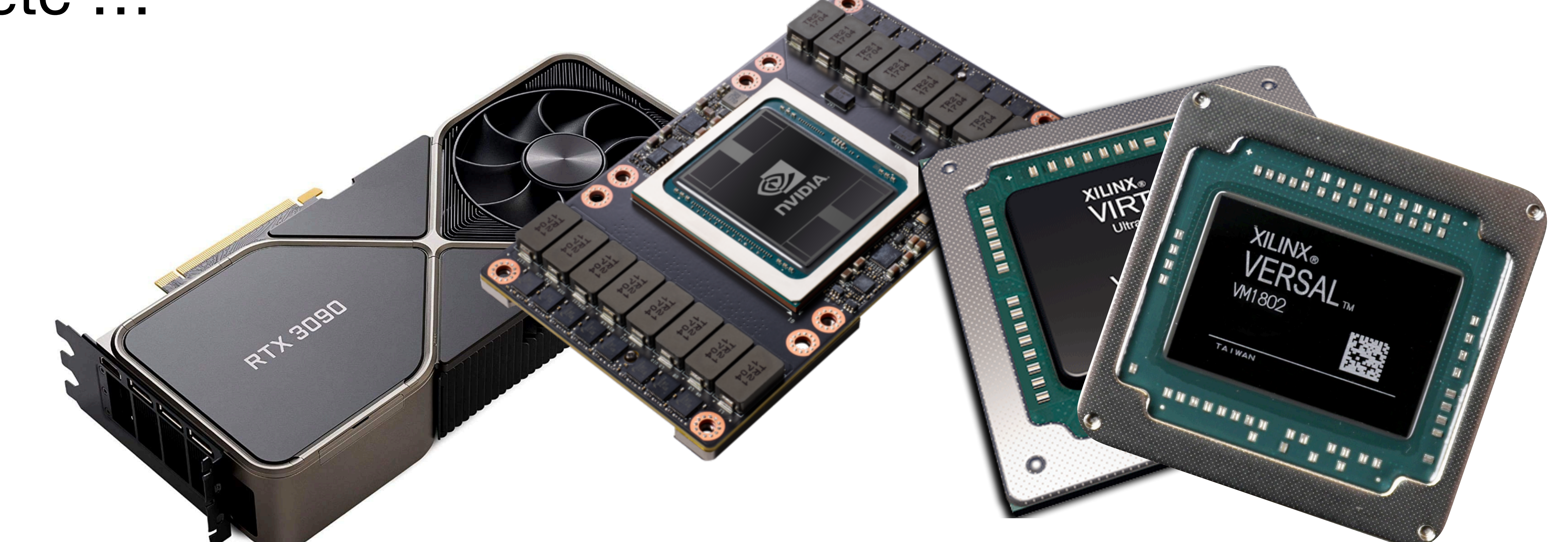
- se si grafica il valore della loss function calcolata sul training set questo tipicamente risulta grande all'inizio dell'ottimizzazione quando i pesi della rete sono stati inizializzati in modo casuale (a piccoli valori random)
- con il passare del numero di iterazioni (epoche) l'errore decresce fino a raggiungere (tipicamente) un valore di plateau che dipende da: dimensioni del training set, l'architettura della rete, il valore iniziale dei pesi, gli iperparametri …
- il progresso nel training viene visualizzato tramite curve di apprendimento (learning curve) che riportano il valore della loss (o l'accuracy o qualsiasi metrica utile per valutare l'apprendimento)

- come in tutti gli algoritmi di ML, per addestrare la rete sono necessari più dataset (e/o l'uso di tecniche di cross validation):
  - per il training
  - per ottimizzare gli iperparametri, decidere il criteri di stop
  - ed infine per valutare le prestazioni del modello addestrato …



ASPETTI CRITICI DI UNA RETE NEURALE

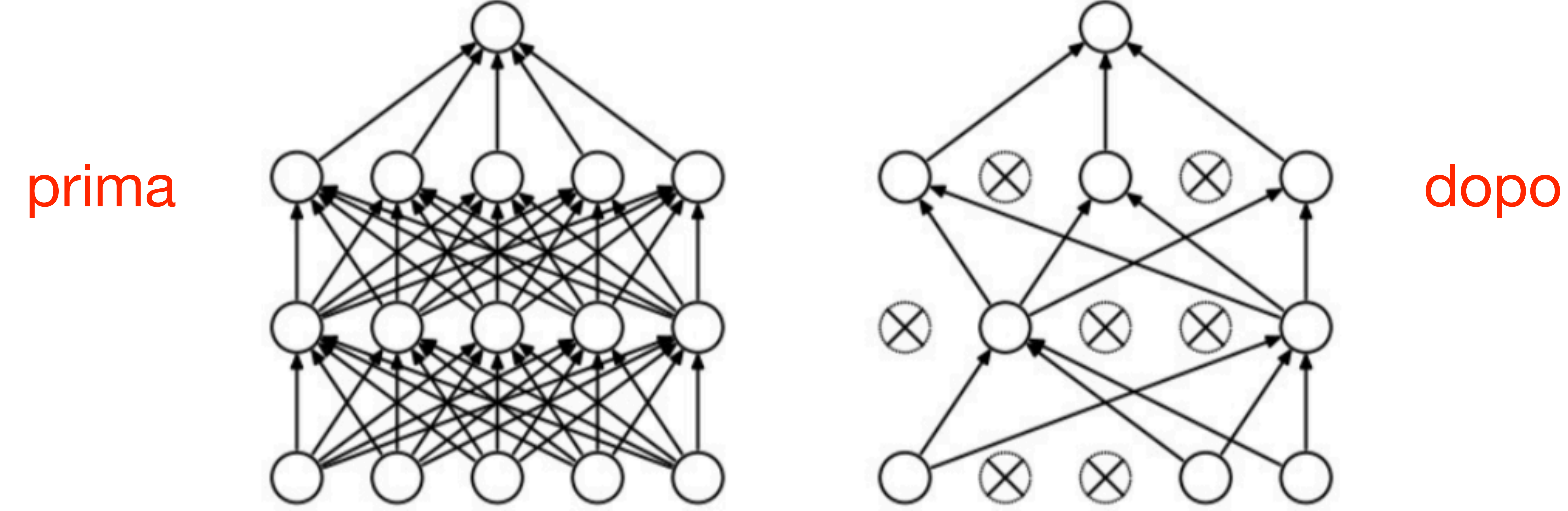
- velocità (di training e di inferenza):
  - varie tecniche per accelerare la convergenza:
    - momentum, adaptive learning rate (Adam o RMSProp), etc …
    - funzioni di attivazione che non saturano (tipo ReLU)
    - batch normalisation
    - compressione/semplificazione delle ANN
    - processori dedicati (GPUs, FPGAs, TPUs (ASICS), …)



- hardcore overfitting:
  - è il problema principale delle reti neurali con architettura complessa con molti layer e molti pesi
  - conseguenza inevitabile del trade-off tra varianza e bias (generalizzazione)
  - soluzione: tecniche di regolarizzazione (dropout, L1/L2/L1+2, …) che vincolano la complessità del modello neurale

DROPOUT

- tecnica molto popolare e potente per prevenire l'overfitting in architecture di reti neurali profonde
  - le connessioni tra i neuroni vengono eliminate in base ad una probabilità definita
  - forza il modello a non basarsi eccessivamente su particolari set di feature



usata di routine nel contesto delle reti ConvNET in cui riesce ad aumentare drasticamente le prestazioni sul campione di test, può anche essere utilizzata come compressore del modello…

ARCHITETTURE PER ANALISI DI IMMAGINI: CONVNET/CNN

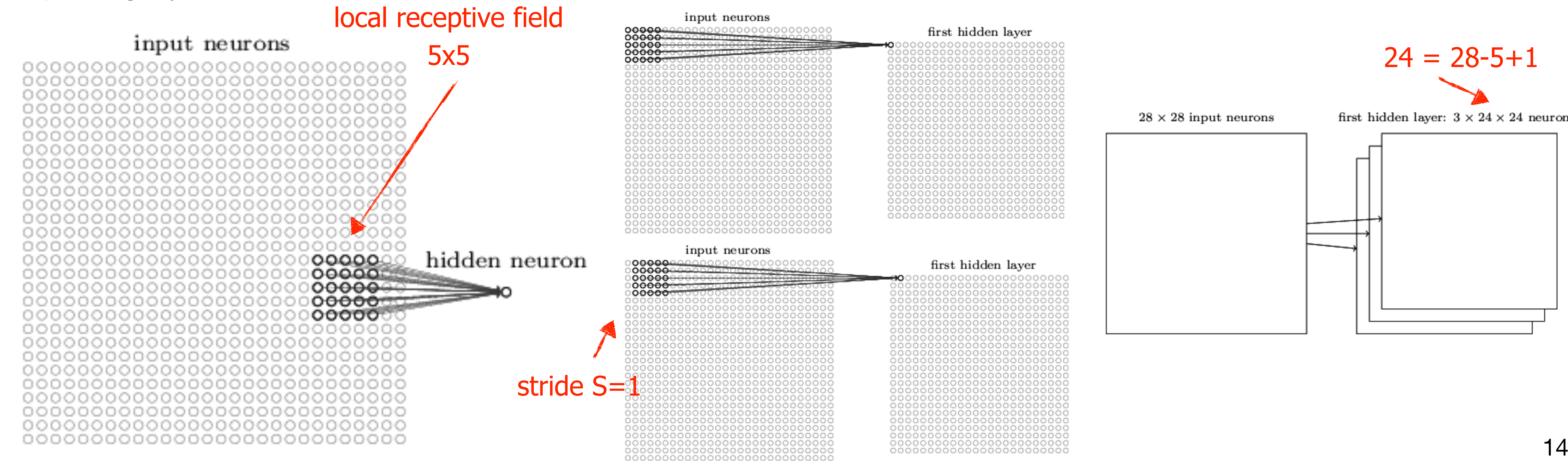
- CNN sono specifiche DNN progettate per eccellere in task di pattern recognition su immagini
- operano direttamente sulle immagini (informazioni raw dei pixels)
- implementano bias induttivi specifici delle immagini fotografiche:
    - struttura spaziale simmetrica dell'input: pixel organizzati in mesh
    - equivarianza per traslazioni: sub-feature nell'immagine rimagono le stesse in different punti dell'immagine
    - auto-similarità: due o più sub-feature simili presenti nell'immagine possono essere riconosciute con un unico filtro in grado di identificare una delle sub-feature
    - composizionalità: una feature complessa composta da varie sub-feature può essere riconosciuta identificando alcune delle sub-feature
    - località delle feature: identificare una sub-feature richiede bastano pochi pixel concentrati in una piccola porzione dell'immagine stessa

CONVOLUTIONAL FEATURE EXTRACTION LAYER

- sono layers di neuroni usati per identificare caratteristiche simili in differenti posizioni dell'immagine

- basati su tree idee di base:
  - local receptive field
  - shared-weights (kernels)
  - pooling layers

- i neuroni di input (uno per ognuno degli NxN pixel dell'immagine) NON sono fully connected con tutti i neuroni del layer nascosto. Le connessioni esistono solo per una piccola regione spaziale dell'immagine
- il local receptive field viene fatto scorrere attraverso l'intera immagine: ad ogni shift viene associato un neurone nascosto nel layer nascosto



l'operazione di convoluzione

local receptive field o convolutional kernel

W_i

X_i

$\sum X_i * W_i$

5x5

1 ×1	1 ×0	1 ×1	0	0
0 ×0	1 ×1	1 ×0	1	0
0 ×1	0 ×0	1 ×1	1	1
0	0	1	1	0
0	1	1	0	0

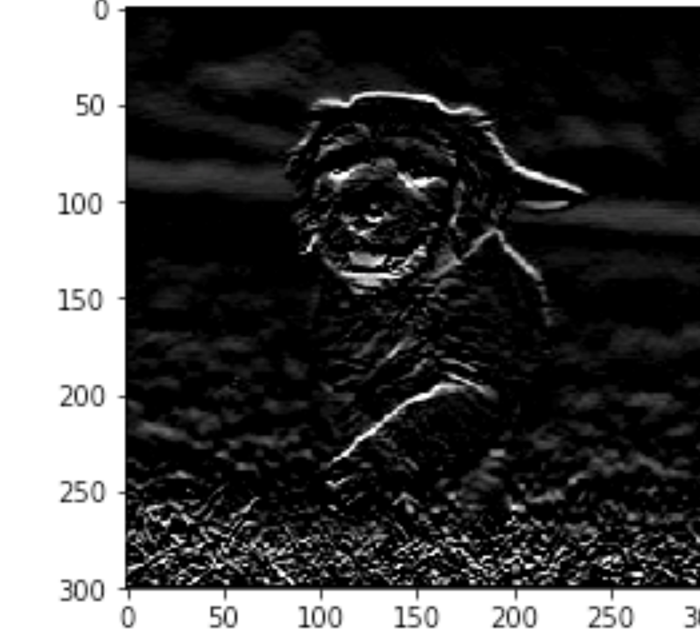
Image

3x3 (5-3+1)

4		

Convolved Feature

```
# convert the image to gray color
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)

# declaring sobel filter
sobel = np.array([[ -1, -2, -1],
                  [ 0, 0, 0],
                  [ 1, 2, 1]])

# applying sobel filter
filtered_image = cv2.filter2D(gray, -1, sobel_y)
# plotting the image
plt.imshow(filtered_image, cmap='gray')
```

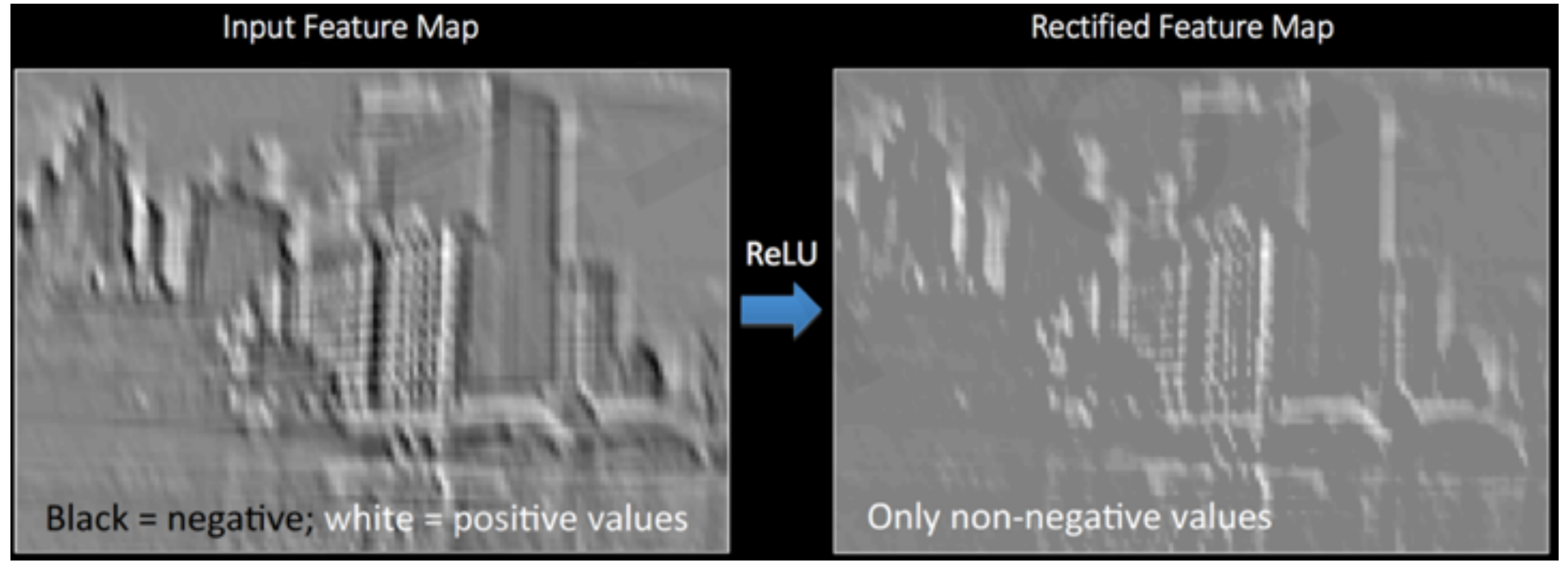


- shared-weights:
  - tutti i neuroni di un dato hidden layer condividono gli stessi pesi → tutti i neuroni dello stesso hidden layer identificano la stessa caratteristica dell'immagine, solo in punti differenti dell'immagine
  - ASSUNZIONE CRUCIALE: invarianza traslazionale degli oggetti nell'immagine
  - poiché la CNN deve identificare molte caratteristiche elementari, ci saranno numerosi kernel ognuno con associato il relativo hidden layer
  - enorme vantaggio rispetto ad una DNN densa: un numero molto minore di parametri (pesi) da apprendere …

NON LINEARITÀ

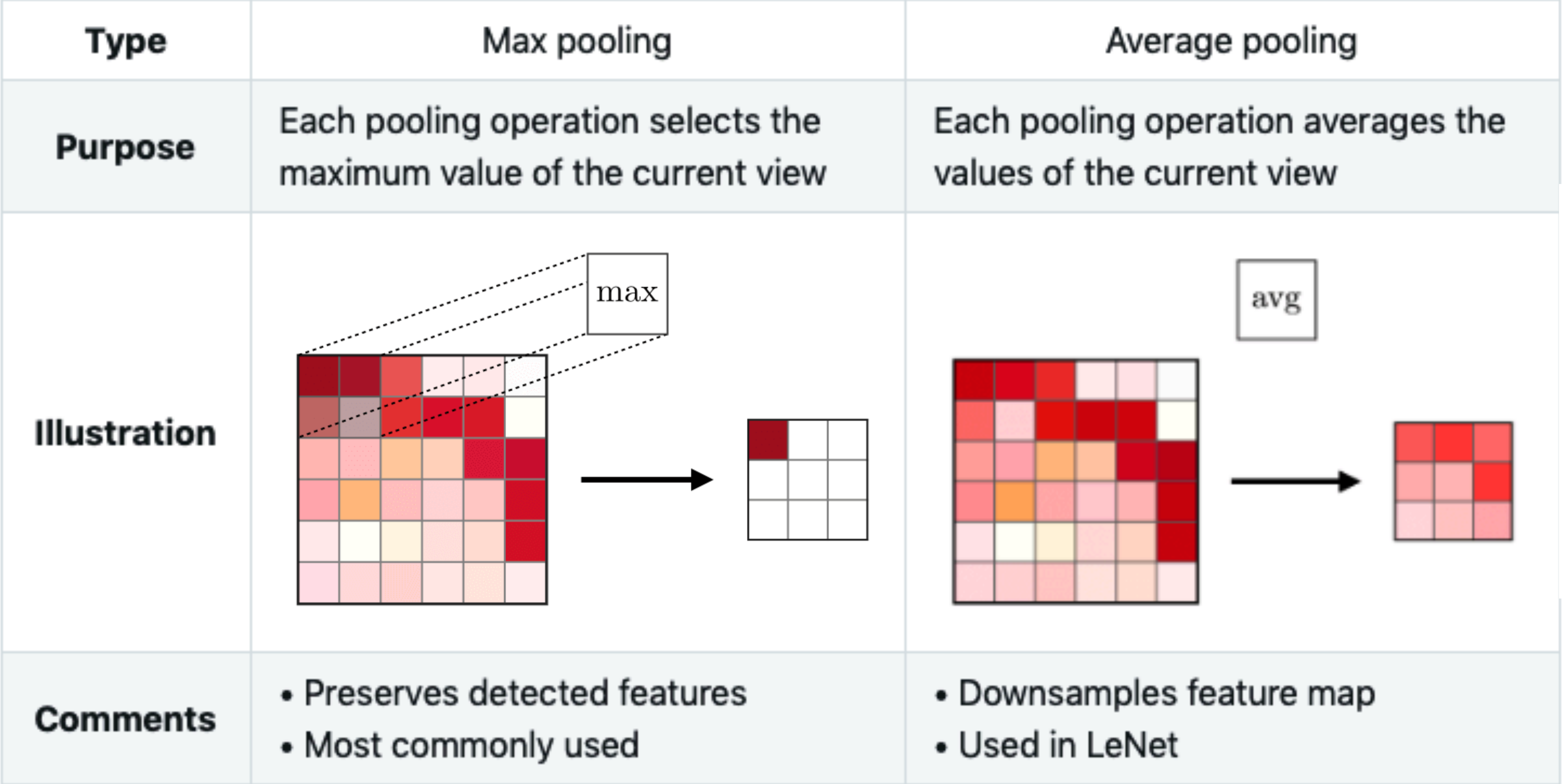
dopo l'operazione di convoluzione ad ogni pixel (neurone) dell'immagine filtrata viene applicata la funzione di attivazione (ex. ReLU: tutti i valori negativi vengono messi a zero)

- enfatizzate solo alcune delle caratteristiche dominanti delle sub-feature selezionate dal filtro

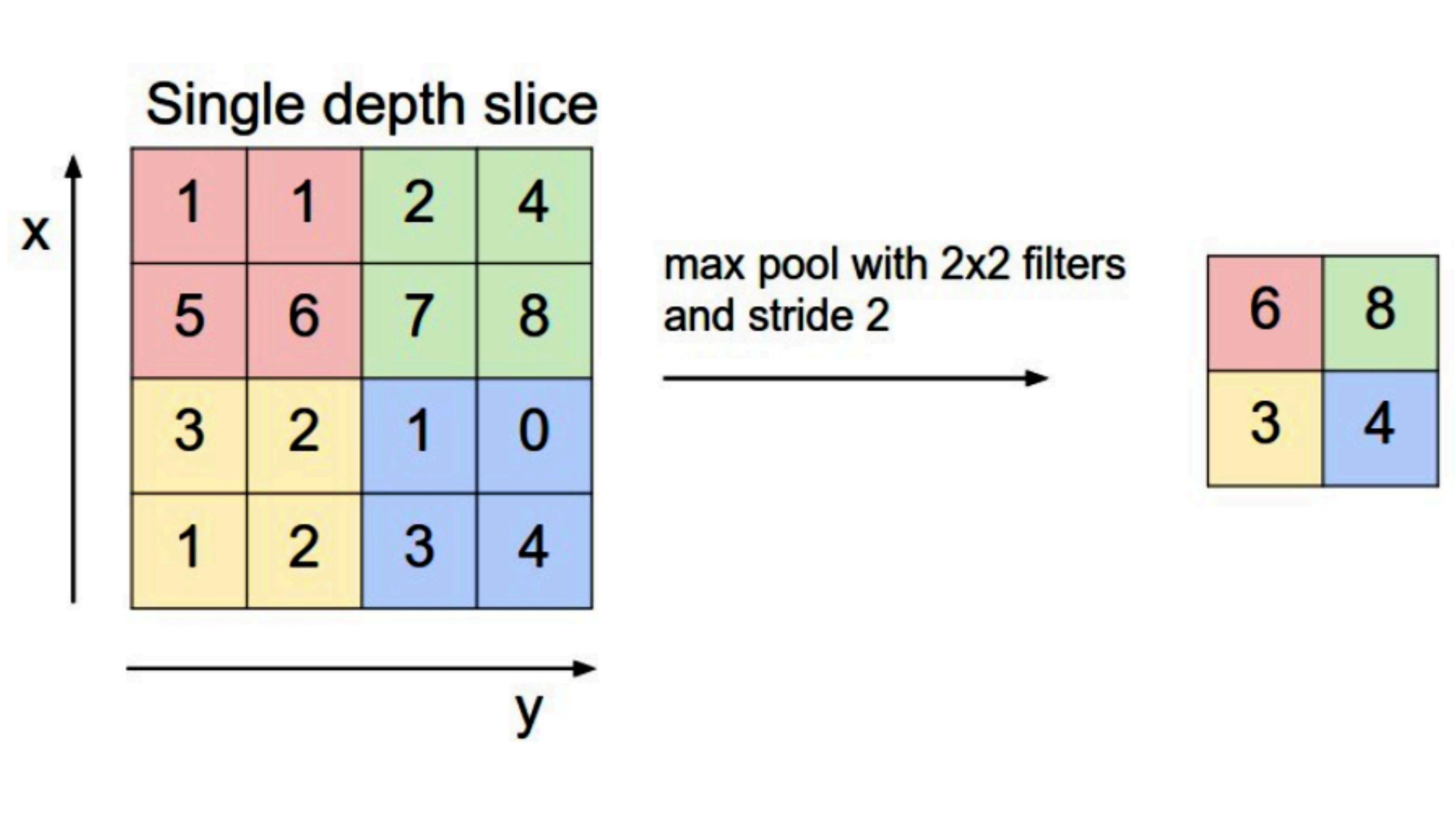


before ReLU after ReLU

- pooling layers:
    - in aggiunta ai layer convolutional una CNN è anche equipaggiata con layer detti di pooling, tipicamente applicati dopo ogni gruppo di layer convolutionali. Lo scopo è quello di applicare un'operazione di downsampling: i.e. semplificare l'informazione in output dal layer convoluzionale riducendo il numero di pesi, e allo stesso tempo rendendo la rete neurale meno sensibile a piccole traslazioni dell'immagine
    - l'idea è quella che una volta identificata una sub-feature, conoscerne la posizione esatta non è cosi importante quanto conoscerne la posizione relativa rispetto alle altre sub-feature presenti nell'immagine

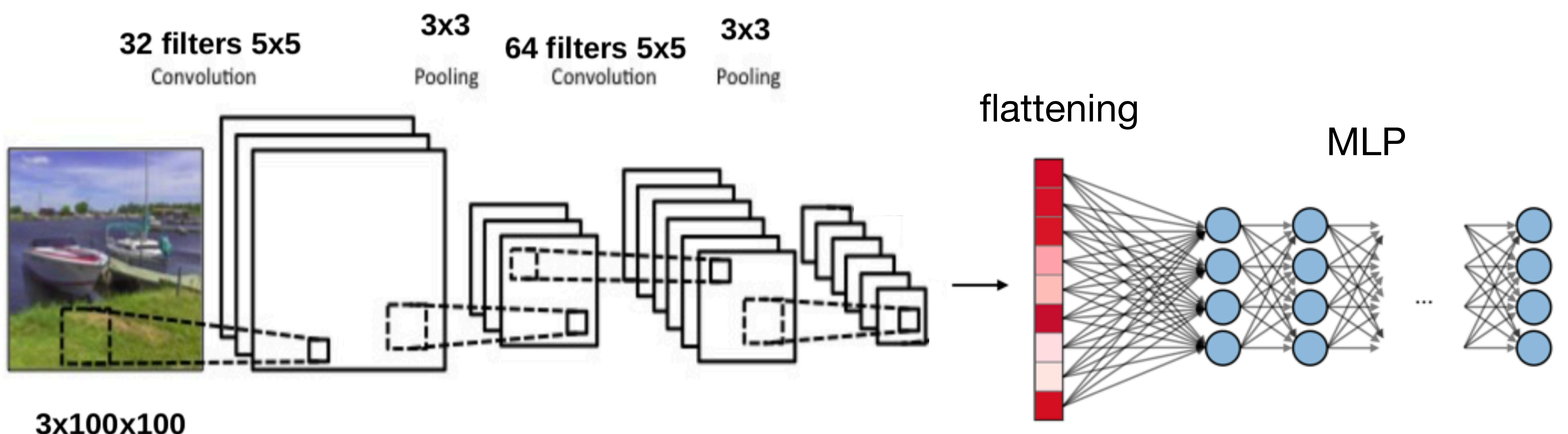


Type	Max pooling	Average pooling
Purpose	Each pooling operation selects the maximum value of the current view	Each pooling operation averages the values of the current view
Illustration	max	avg
Comments	• Preserves detected features • Most commonly used	• Downsamples feature map • Used in LeNet



BLOCCO FINALE MLP IN UNA CNN

- generalmente l’output dei layer convoluzionali viene connesso, via un layer di flattening, a uno o più layer di tipo denso che sono utilizzati per ottimizzare la funzione di loss rispetto alla task specifica della rete (classificazione, regressione, …) etc..

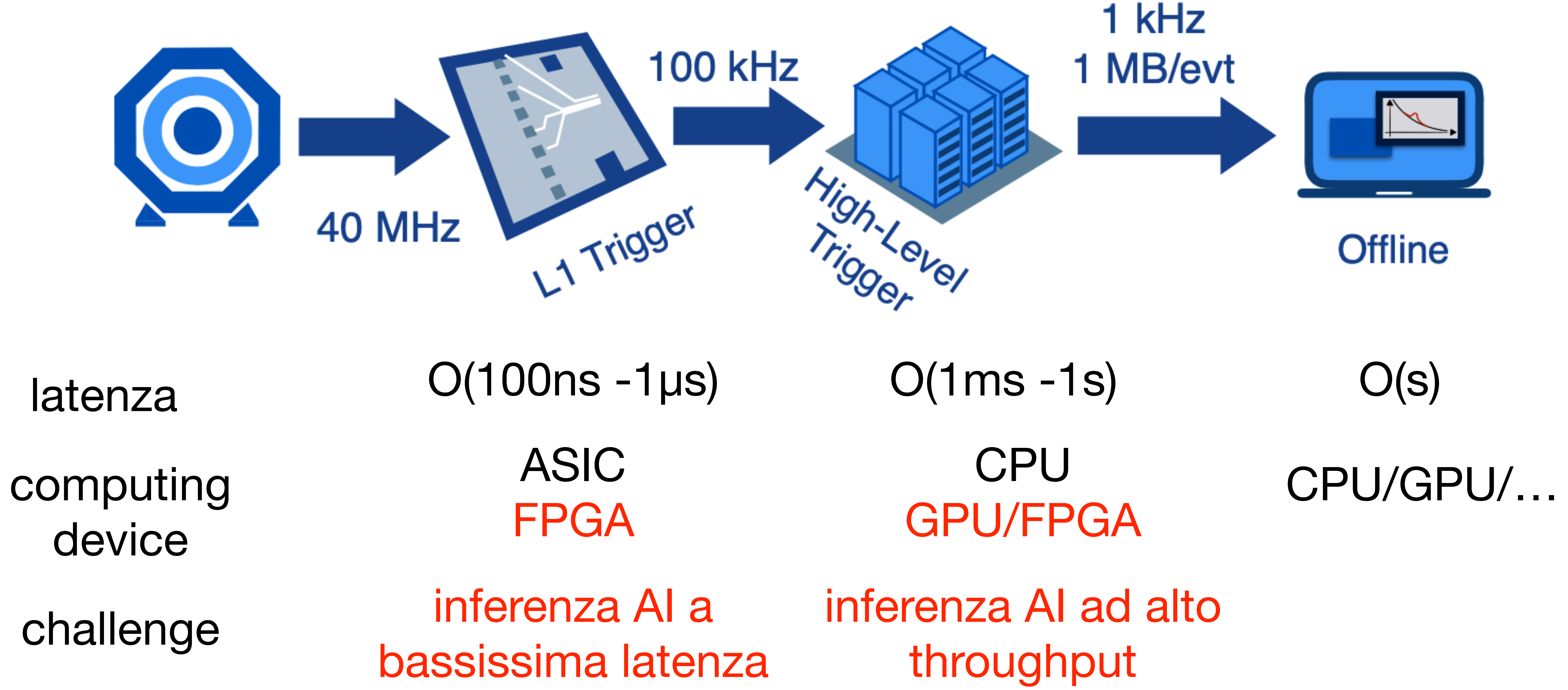


DNN NEI TRIGGER DI ESPERIMENTI DI FISICA DELLE ALTE ENERGIE

- la potenza delle DNN già da alcuni anni ha consentito lo sviluppo di nuove tecniche di rivelazione, ampliando il potenziale di misura e scoperta dei grandi esperimenti di fisica delle alte energie
- una delle linee di sviluppo più interessanti è legata alla possibilità di estendere l'uso di questi algoritmi all'online degli esperimenti, in particolare nei sistemi di trigger
    - molto più flessibili degli algoritmi tradizionali
    - possibilità di sfruttare l'informazione raw dei detector senza la perdita di informazione legata al feature engineering
    - potrebbero fornire la chiave per superare le sfide sperimentali della prossima generazione di esperimenti (a partire da HL-LHC)

DNN NEI TRIGGER DI ESPERIMENTI DI FISICA DELLE ALTE ENERGIE

- tuttavia i sistemi di trigger (in generale tutti i sistemi real-time a latenza ridotta) hanno vincoli molto più stringenti dell'offline e risorse spesso limitate



PROBLEMI DELLE ARCHITETTURE DNN CONVENZIONALI

- oggi le architetture di reti neurali con prestazioni allo stato dell'arte hanno decuplicato le dimensioni rispetto ai modelli di pochi anni fa (AlexNet), e il trend non sembra rallentare
  - es. Google PalM NLP (architettura multitask basata su transformer): 520 miliardi di parametri
- dimensioni così grandi implicano diversi problemi:
  - **occupazione di memoria:**
    - il modello + i dati devono essere copiati fisicamente sulla memoria del processore durante il training → limitazioni nelle dimensioni dell'input/batch size
    - il trasferimento processore-memoria esterna è di norma il collo di bottiglia computazionale nella fase di inferenza → maggiore latenza durante l'inferenza
    - non sono adatti all'utilizzo su small device (smartphone per esempio) o device con poca memoria (FPGA senza memoria esterna)

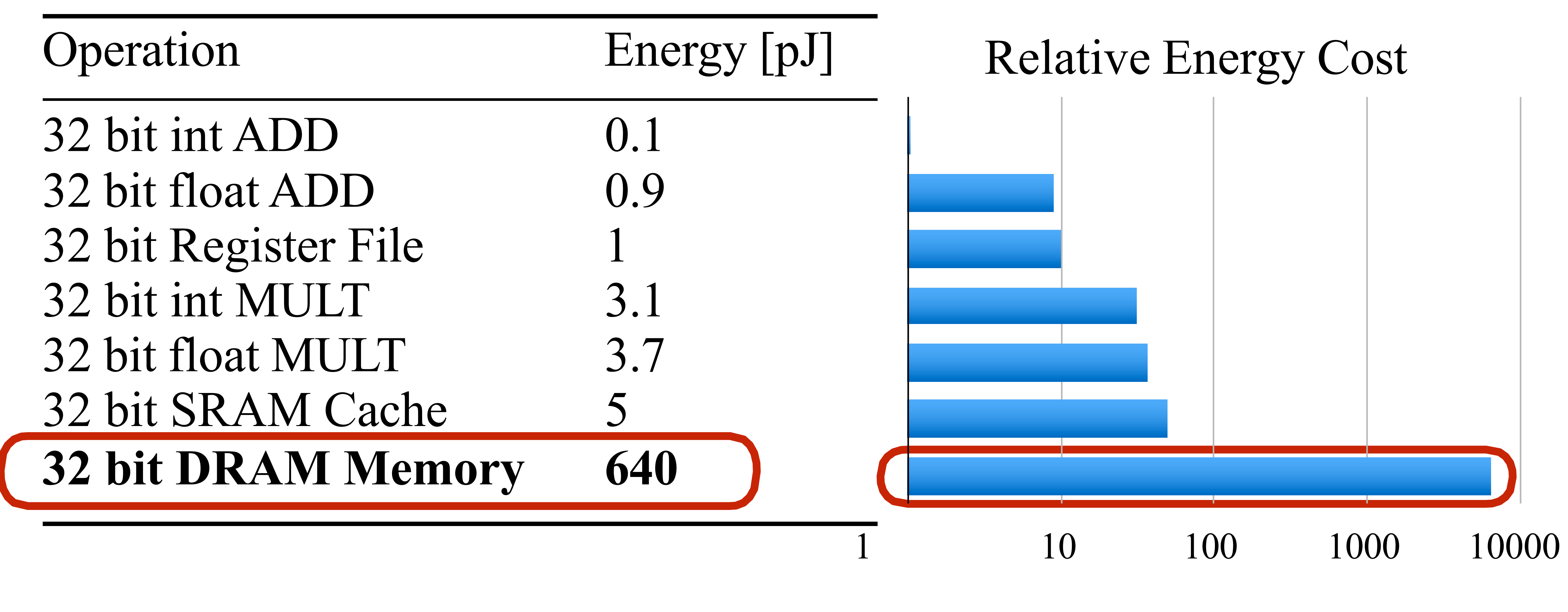
- **costo computazionale:**
  - sia per il training:

	Error rate	Training time
ResNet18:	10.76%	2.5 days
ResNet50:	7.02%	5 days
ResNet101:	6.21%	1 week
ResNet152:	6.16%	1.5 weeks

  - sia per l'inferenza: **non sono adatti per essere utilizzati in sistemi real-time a bassa latenza**

- **costo energeticp:**
  - DeepMind GPT-3: 12M USD per essere addestrato
  - già nel 2016 alphaGO richiedeva 3k USD per game solo nella fase di inferenza
  - reti piccole invece possono essere più facilmente portate e utilizzate su processori più efficienti dal punto di vista energetico (ARM, FPGA, ACAP, …)

DOVE VIENE CONSUMATA L'ENERGIA?



Operation	Energy [pJ]
32 bit int ADD	0.1
32 bit float ADD	0.9
32 bit Register File	1
32 bit int MULT	3.1
32 bit float MULT	3.7
32 bit SRAM Cache	5
32 bit DRAM Memory	**640**

1 = 1000

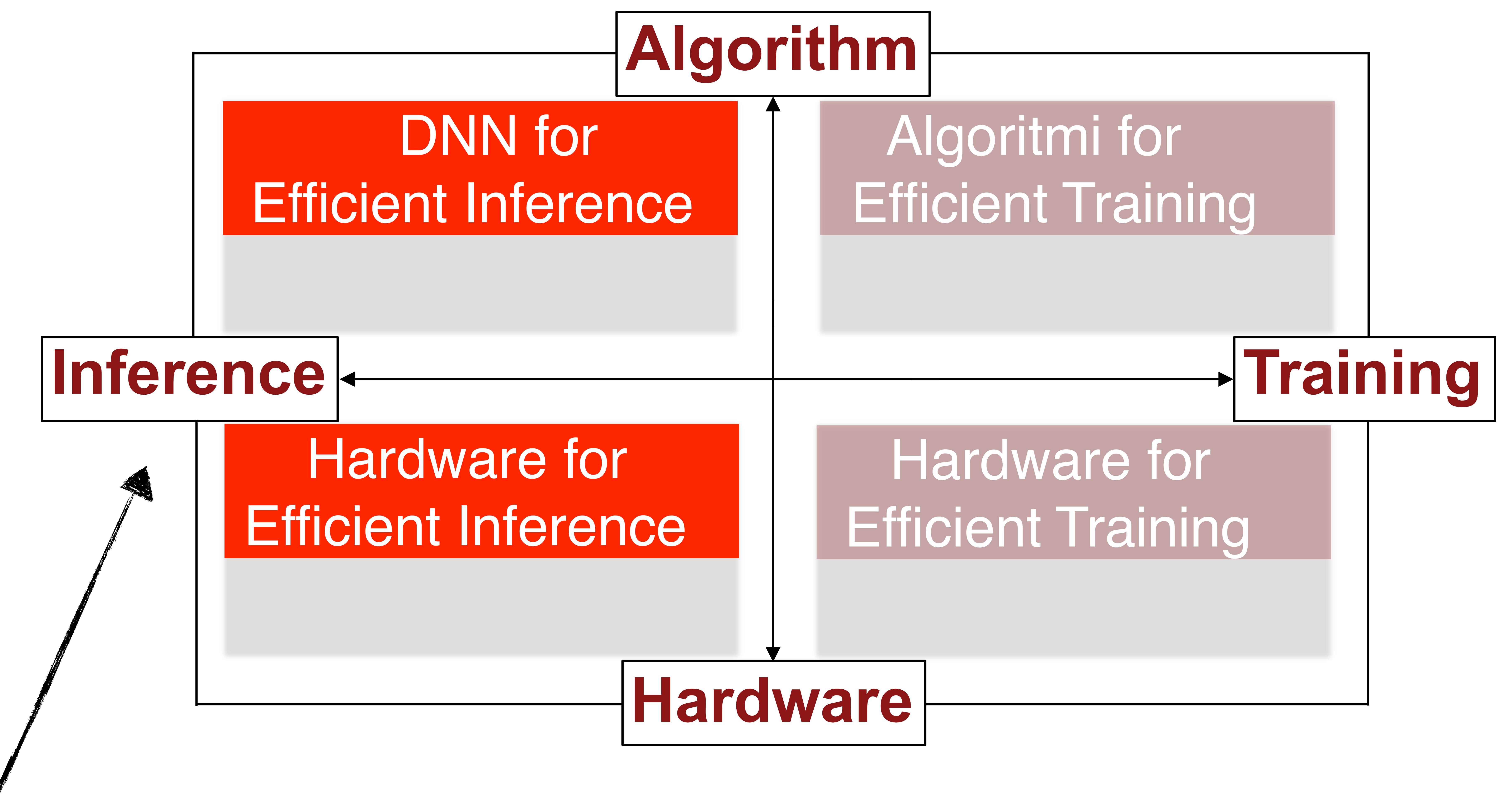
modelli più grandi implicano più memoria e quindi molta più energia

in sintesi: la dimensione della DNN è il problema principale sia per la latenza, sia per l'utilizzo delle risorse che per il consumo energetico

soluzione

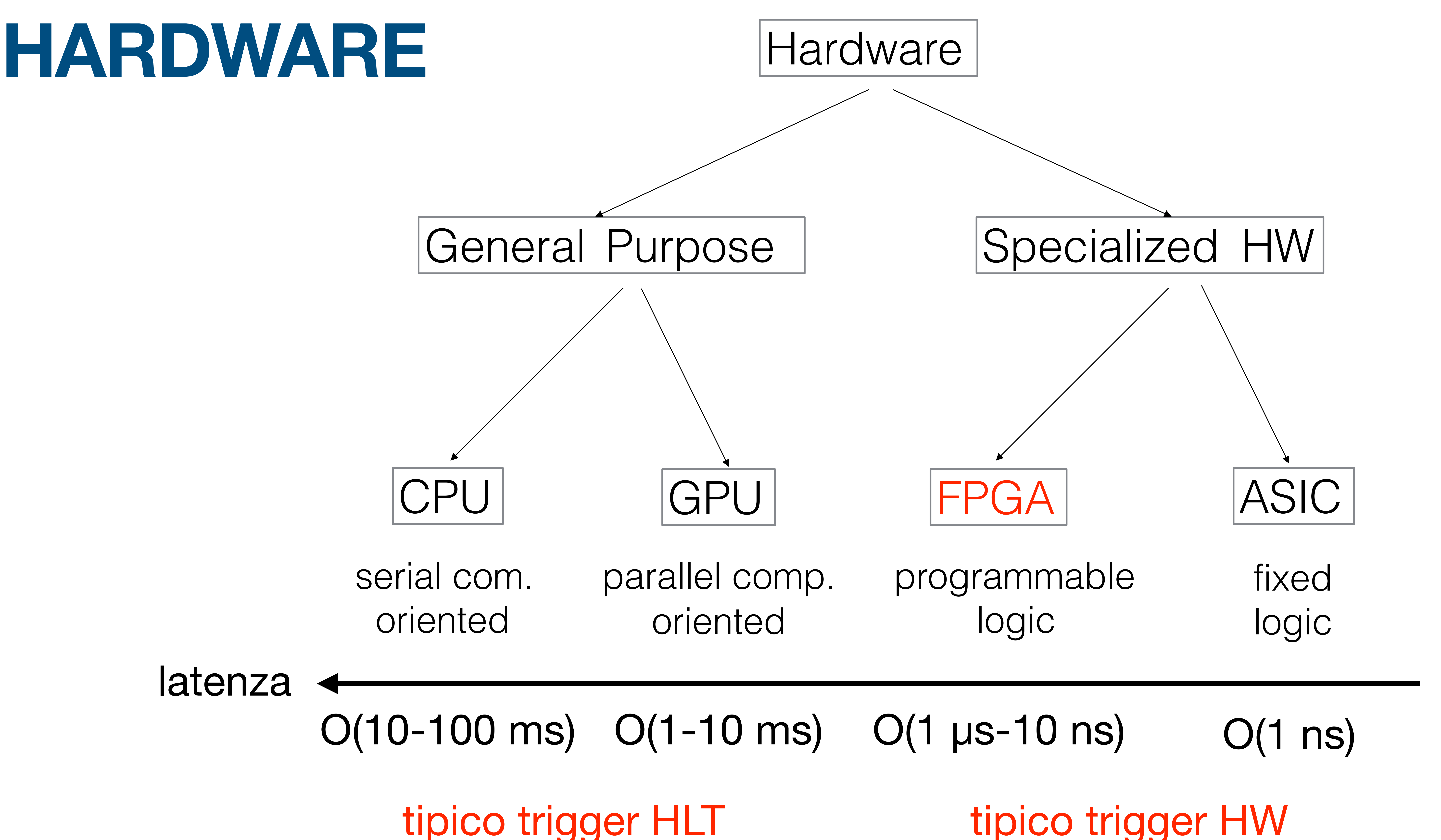
reti neurali ottimizzate per inferenza efficiente

hw ottimizzato per inferenza efficiente e veloce (FPGA)



regione rilevante per le applicazioni
di DNN a sistemi di trigger

HARDWARE



ALGORITMI DI MODEL COMPRESSION

- GOAL: realizzare modelli di reti neurali lightweight e che siano:
  - veloci
  - efficienti dal punto di vista dell'uso della memoria
  - efficienti dal punto di vista energetico

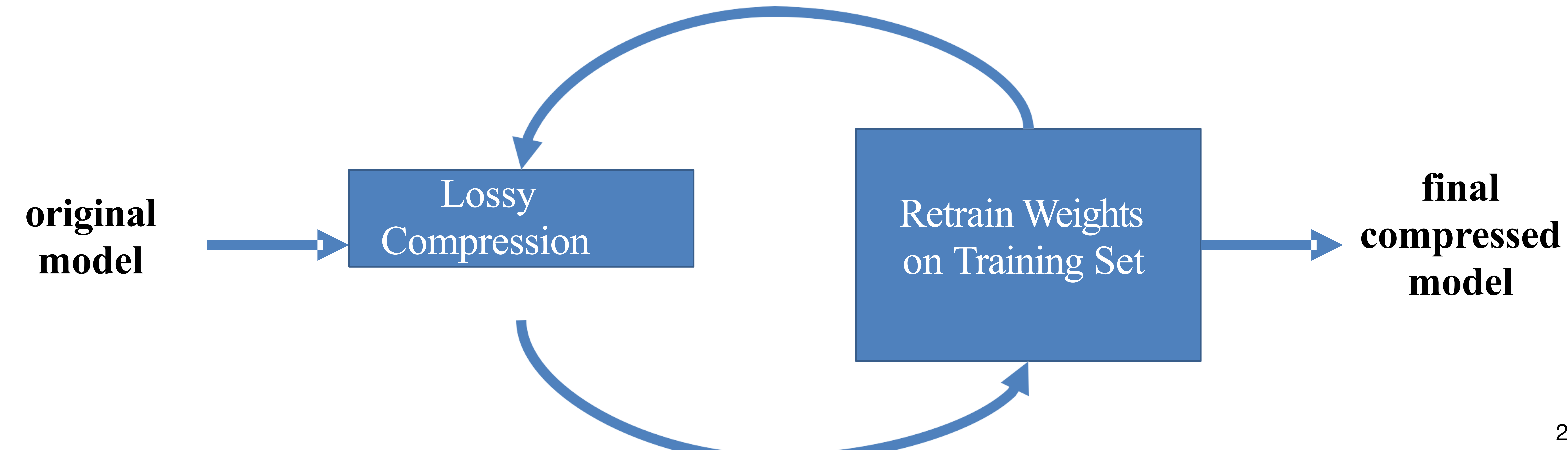
- Due tecniche di compressione utilizzate e diversi paradigmi di addestramento:
  - training aware compression: si addestra direttamente un modello lightweight
  - post training compression: a partire da un modello complesso con prestazioni allo stato dell'arte si ottiene un modello lightweight eliminando parte delle informazioni nel modello originale

COMPRESSIONE TRAINING AWARE E FINE TUNING

idea semplice ma efficace: si applica la compressione lossy al modello addestrato e si riaddestra il modello per migliorarne le prestazioni

- full training
- fine tuning: si addestra per poche epoche (2-5 epoche max) con bassi valori del LR

spesso il fine tuning fornisce prestazioni migliori di un full training



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
- **quantization:** 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 quantità (16bit, 8bit, 4bit … invece che gli usuali 32bit o 64bit)
- **Binary / Ternary net:** versione estrema della quantizzazione (reti con pesi e ativazioni binarie [0,1] o ternarie [-1,0,1]

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

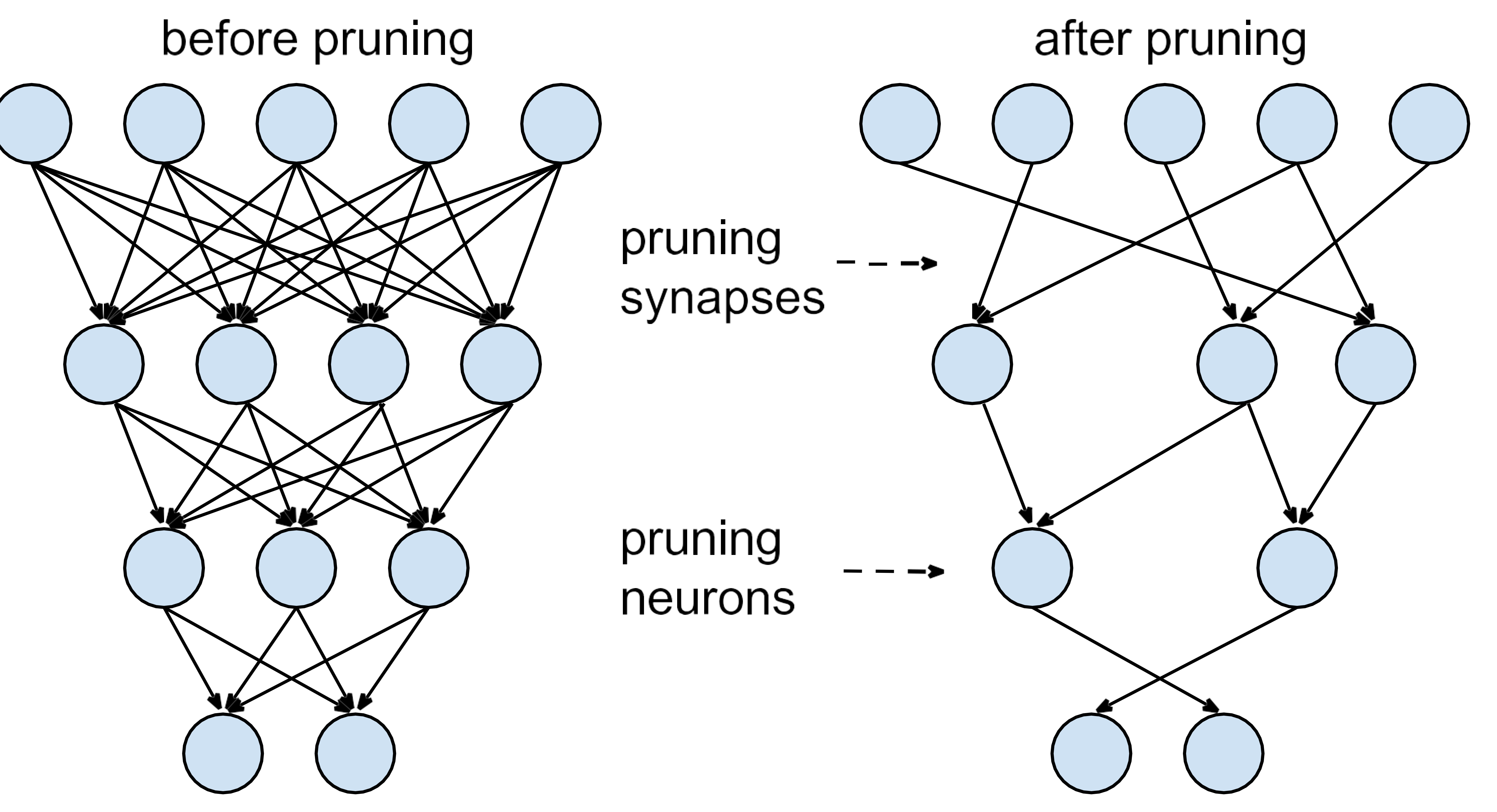
- **low-rank factorisation:** riduce la dimensionalità delle matrici dei pesi applicate ad ogni layer della rete per velocizzare le operazioni di consolazione nei primi layer (che dominano i tempi di inferenza in una deep CNN)

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

PRUNING

- la tecnica di pruning (LeCun, NIPS ’89, Han, NIPS 2015) basata sulla grandezza dei pesi, azzera gradualmente i pesi del modello durante il training per ottenere un predefinito livello di sparsità del modello
- un modello sparso può poi essere sia compresso facilmente con tecniche di compressione tradizionali (e.g. gzip), sia usato direttamente (con i pesi posti a zero) saltando gli zeri durante l’inferenza per diminuire il tempo necessario per l’inferenza

  - si possono ottenere compressioni anche di fattori x5 con una perdita di prestazioni minima nel modello
  - funziona meglio con layer densi, meno bene con layer convoluzionali



MOTIVATA DA STUDI SUL CERVELLO ANIMALE

1000 Trillion Synapses

50 Trillion Synapses

500 Trillion Synapses

This image is in the public domain

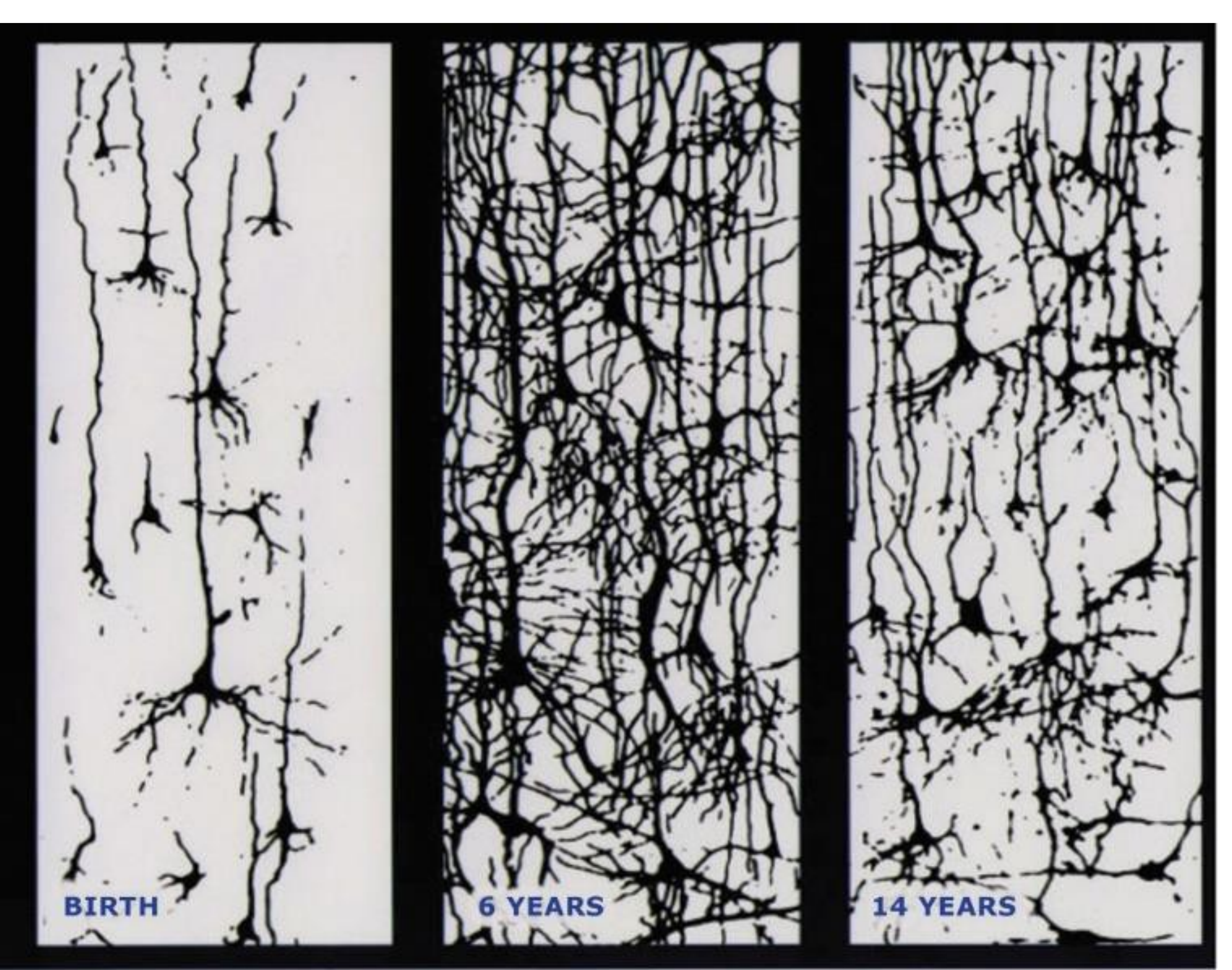
Newborn

This image is in the public domain

1 year old

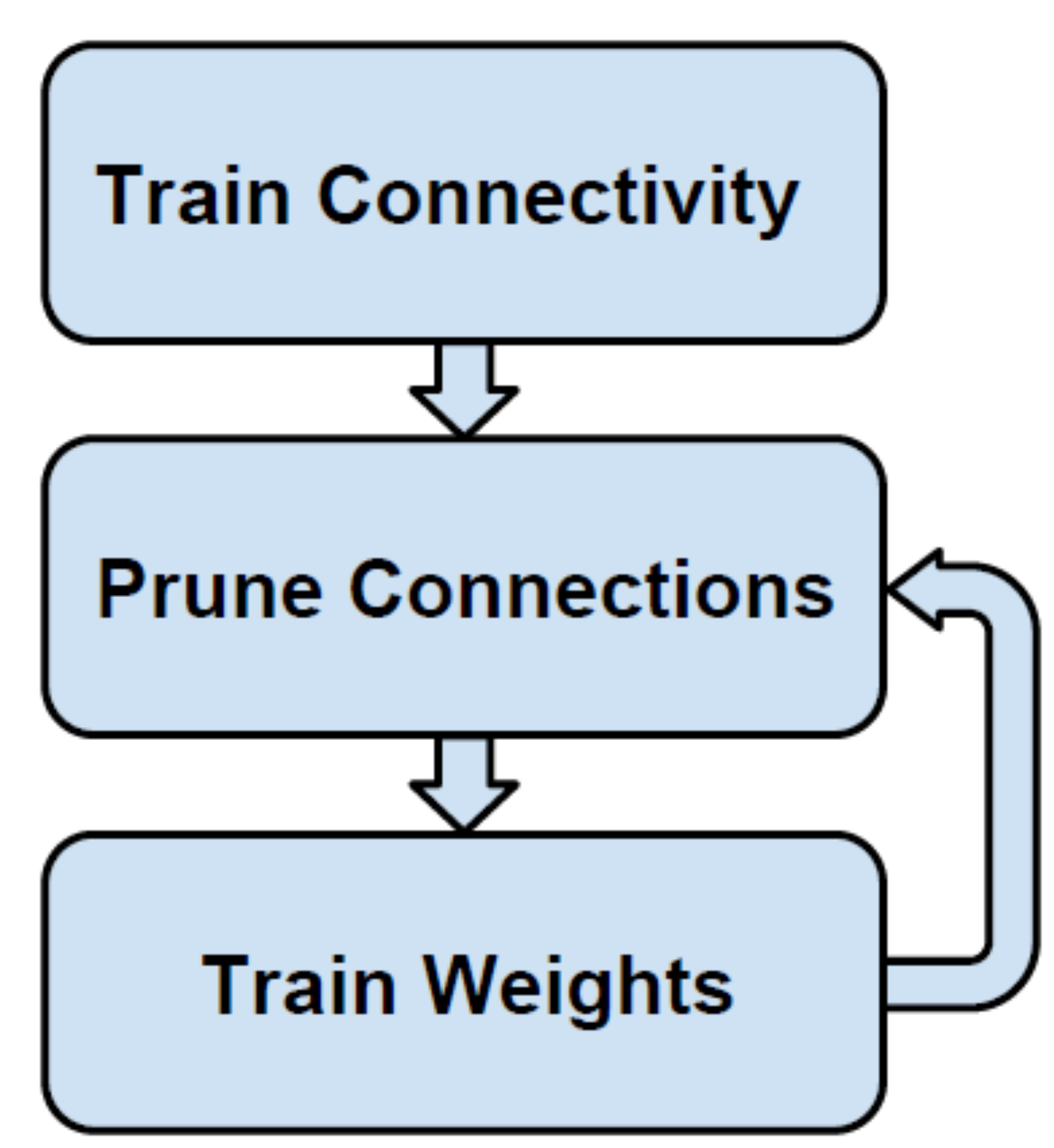
This image is in the public domain

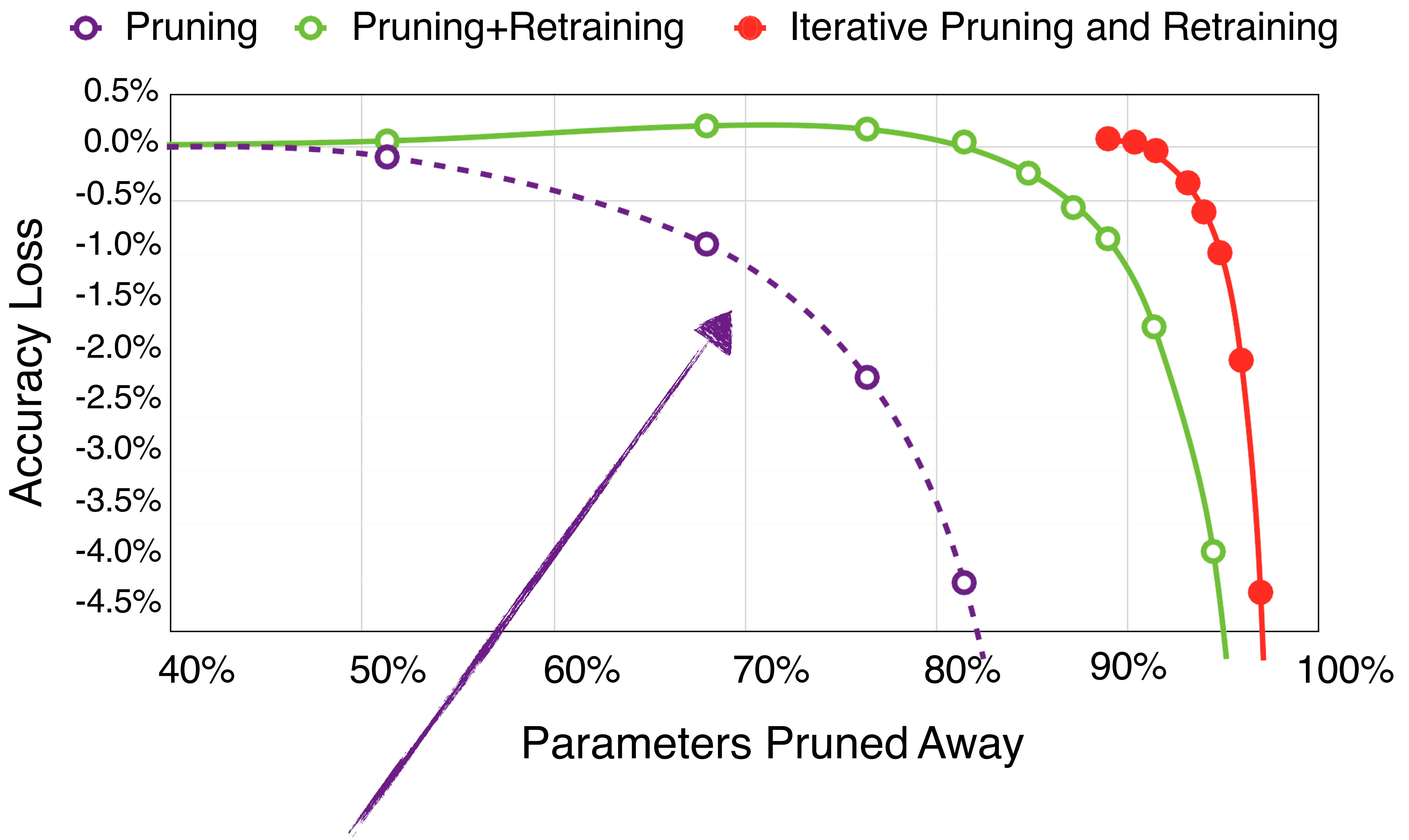
Adolescent



Christopher A Walsh. Peter Huttenlocher (1931-2013). Nature, 502(7470):172–172, 2013.

PROCEDURA DI TRAINING AWARE PRUNING

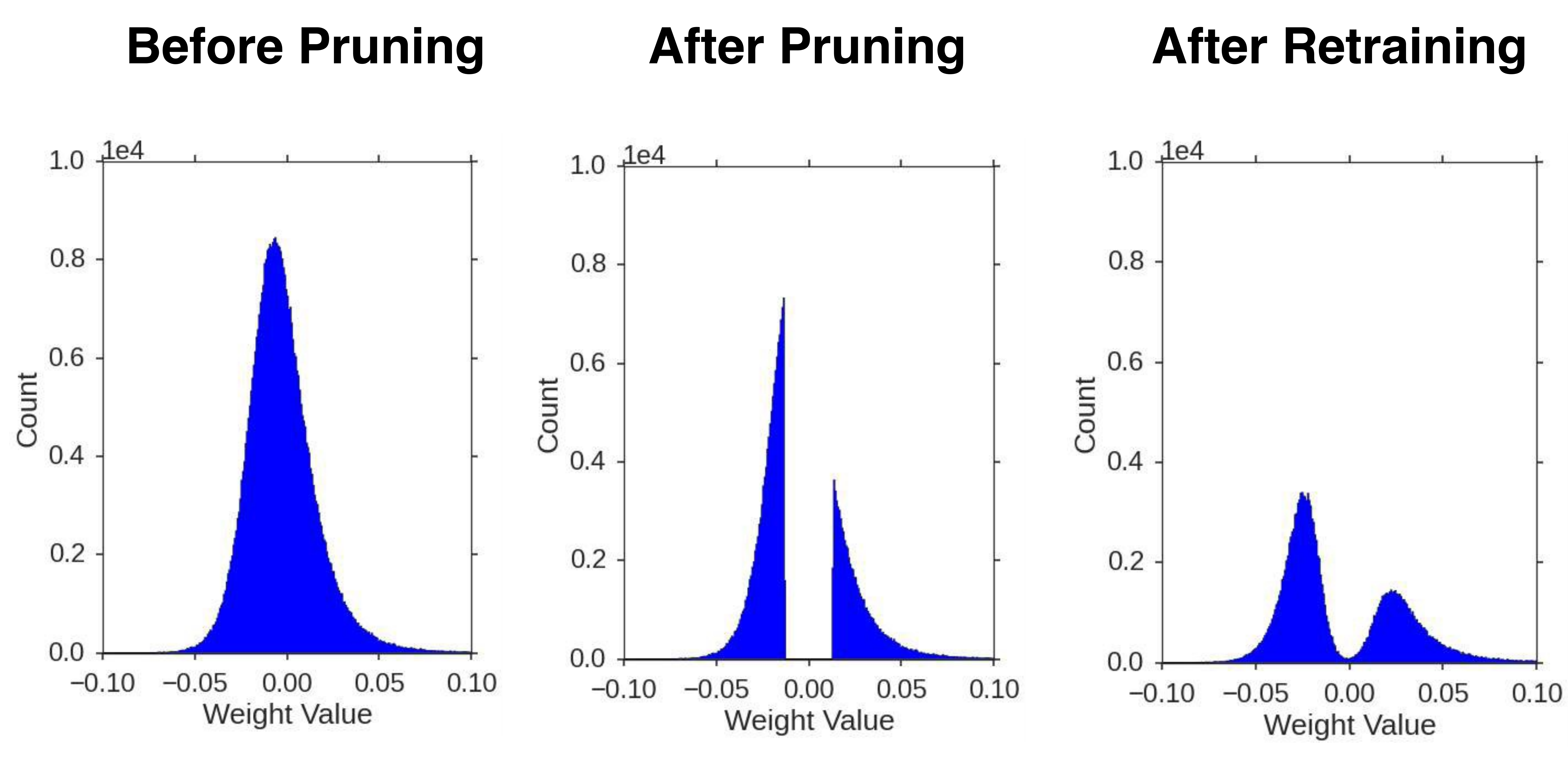




- remove weights which $weight < threshold$
- retrain after pruning weights
- learn effective connections by iterative pruning

pruning after training (dropout) non è una tecnica molto efficace per forti compressioni

DISTRIBUZIONE DEI PESI



Han et al, NIPS 2015 - Conv5 layer of Alexnet. Representative for other network layers as well.

COMPRESSIONE VIA QUANTIZZAZIONE

- riduce la dimensione con cui i pesi e/o le attivazioni della rete vengono rappresentate (64/32 bit → 16, 8, … fino a 2 o 1 bit (ternary e binary networks))
- due tipologie di quantizzazione in DNN:
  - PTQ: post training quantisation, più semplice ma spesso con perdita di prestazioni non recuperabile con fine tuning (a meno di non usare bassi livelli di quantizzazione: 16/8 bit)
  - TAQ: training aware quantisation, richiede il riaddestramento del modello quantizzato ma porta spesso a prestazioni allo stesso livello (a volte superiori in termini di generalizzazione) a quelle del modello di partenza
- può essere applicata in modo diverso in ogni layer della rete (dynamic quantisation)
- è uno step obbligatorio quando:
  - si vogliono sfruttare specifiche unità di accelerazione (es. DPU nelle schede Xilinx Alveo richiedono fixed point INT8)
  - si vogliono sintetizzare modelli neurali su FPGA con latenze ultra-piccole (O(100 ns))

QUANTIZZAZIONE

- training con float
- quantizzazione dei pesi e delle attivazioni:
  - si accumula statistica per le distribuzioni
  - si sceglie l’intervallo dinamico più appropriato
- fine-tuning con float
- conversione a formato fixed-point

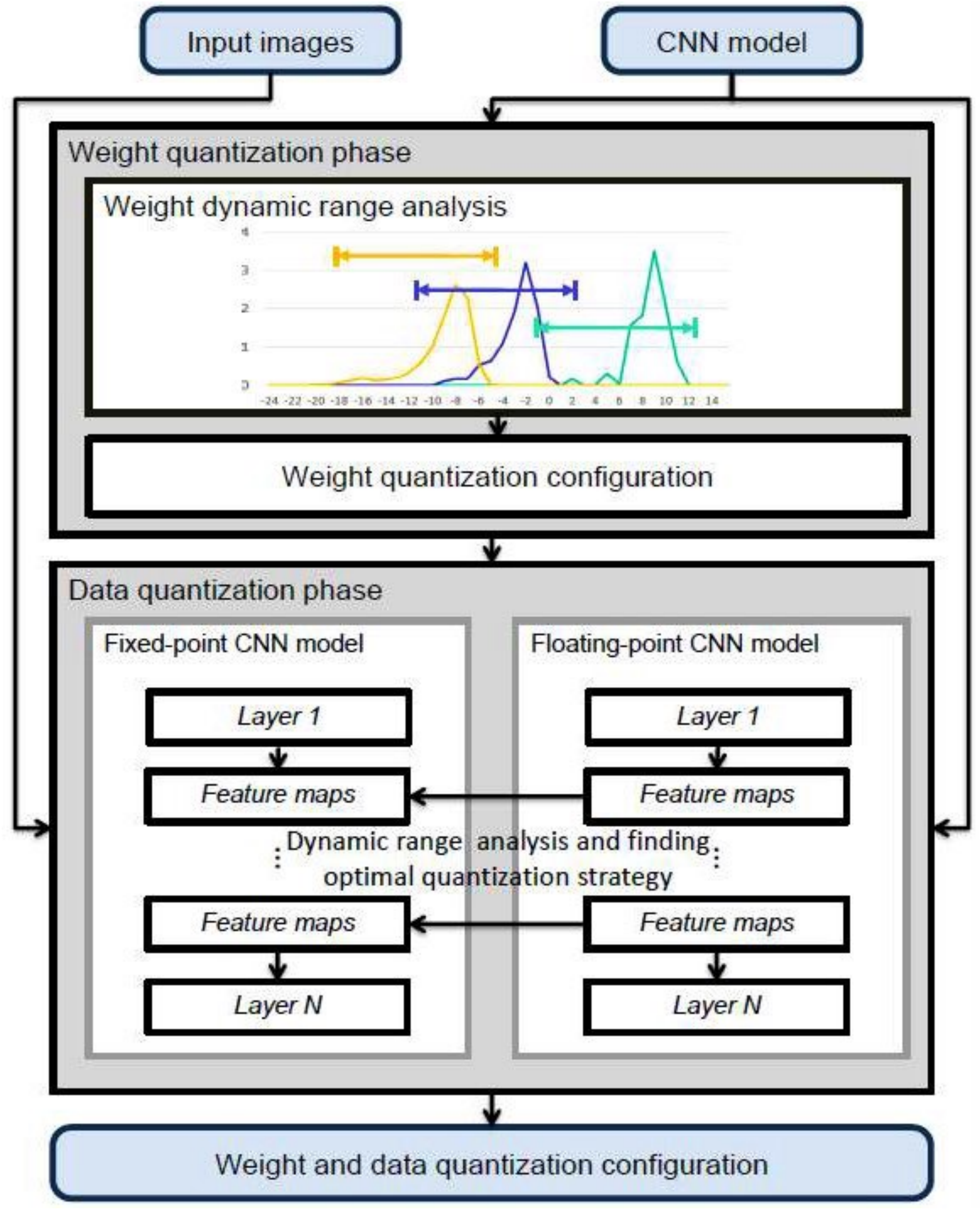
ap_fixed<width,integer>

0101.1011101010

integer fractional

width

NOTA: spesso si opta per una procedura non dinamica (stessa quantizzazione per tutti i pesi/attivazioni della rete



Qiu et al. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network, FPGA’16

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.

High

Throughput

DPUv4E, Throughput optimized
- VCK5000, 384 AIE cores, max 8PE
- Popular CNN models, MLPerf 1.0 submission
- Higher throughput with low latency to compete against T4

DPUv3INT8, Throughput optimized
- Alveo U200 / U250
- Optimized for 100% INT8 models and ResNet/Inception
- High Throughput at medium latency

xvDPU, Latency optimized
- VCK190, 32 to 256 AIE cores, 1-6 batch
- Popular CNN models,
- Scalable with power & cost

DPUv3E, Throughput optimized (HBM)
- Alveo U280 / U50 / U50LV / U55C
- Optimized for ResNet/Inception models and HBM devices
- Low-cost and high power efficiency solution

DPUv2, Scalability optimized
- Zynq-7000, ZU+ MPSoC, Kria SoM
- Popular CNN models
- Scalable with power & cost

Low — Latency — High

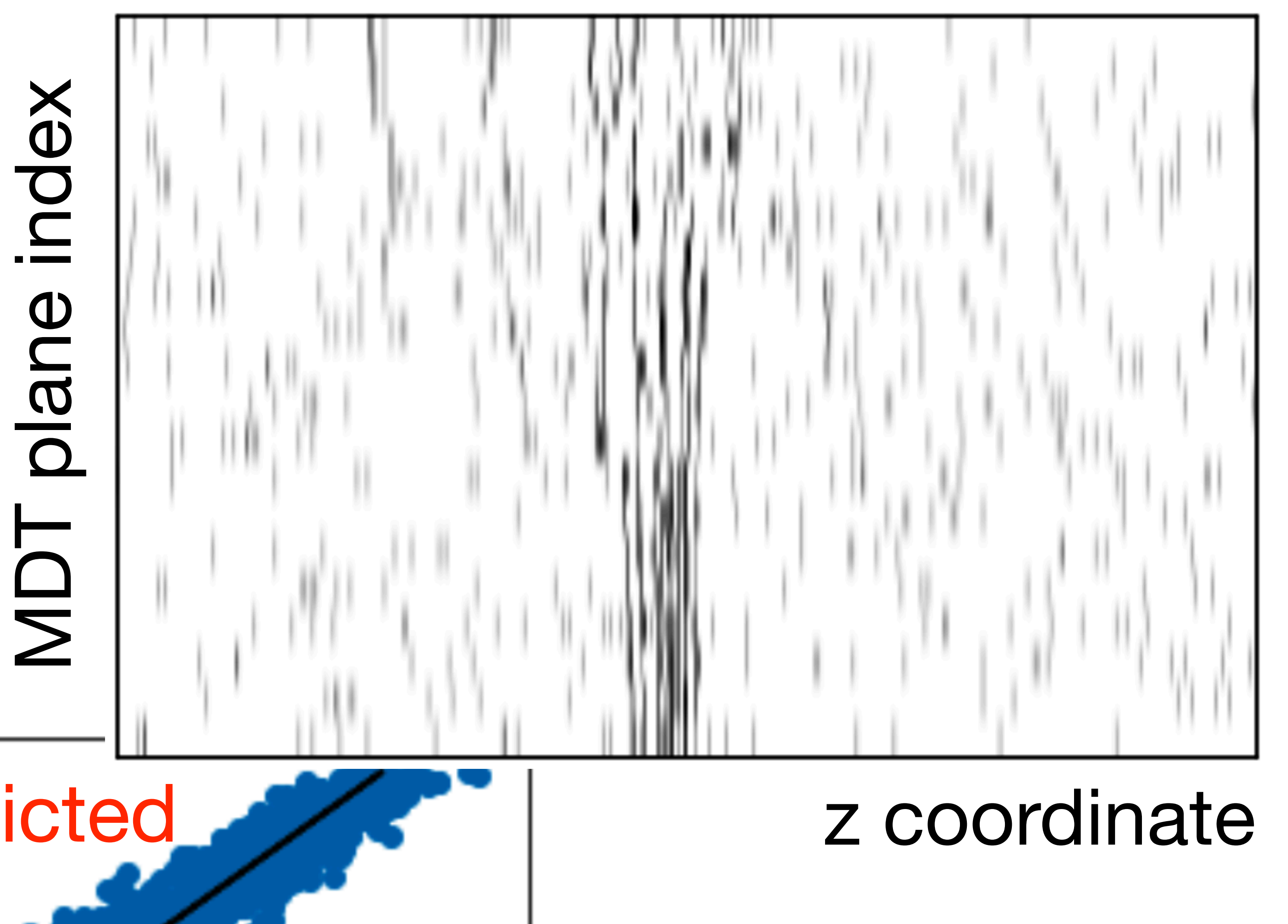
7nm — 16/28nm

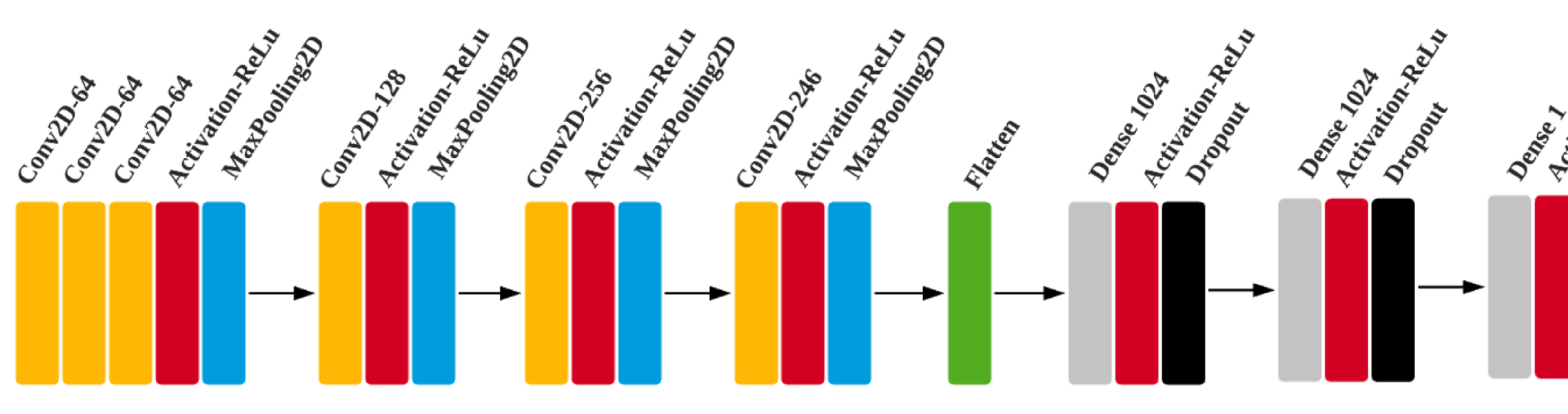
ESEMPIO USO: AI-ASSISTED MUON TRIGGERS FOR LLP

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

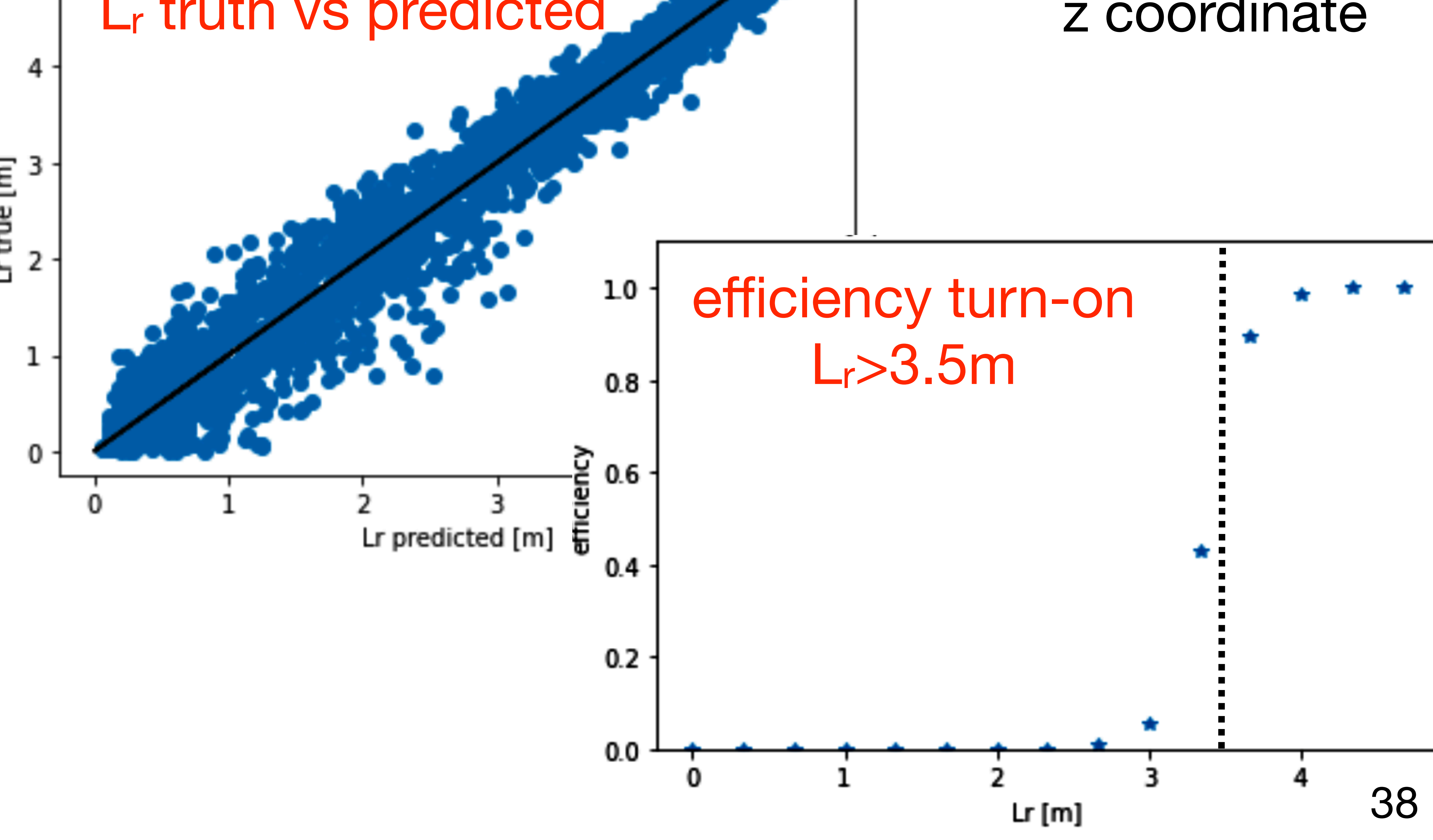
- benchmark: LLP neutre che decadono in multi-muoni a differenti distanze dal vertice primario di interazione
- hits pattern nello spettrometro rappresentati come immagini binarie
- immagini usare per addestrare una CNN a predire la distanza di decadimento radiale (L_r) della LLP

L_r



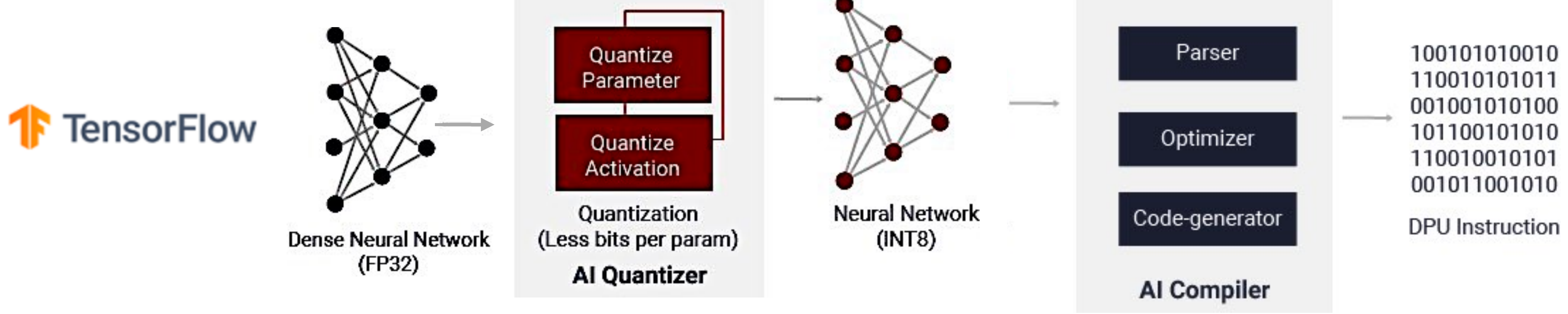


VGG Convolutional Neural Network architecture



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

performances:
CPU/GPU FP32
FPGA INT8

	CPU Xeon E5-2698	GPU TESLA V100	FPGA ALVEO U50
ms/prediction	~3	0.14	0.075
predictions/sec	~350	~7 10^3	~13.3 10^3

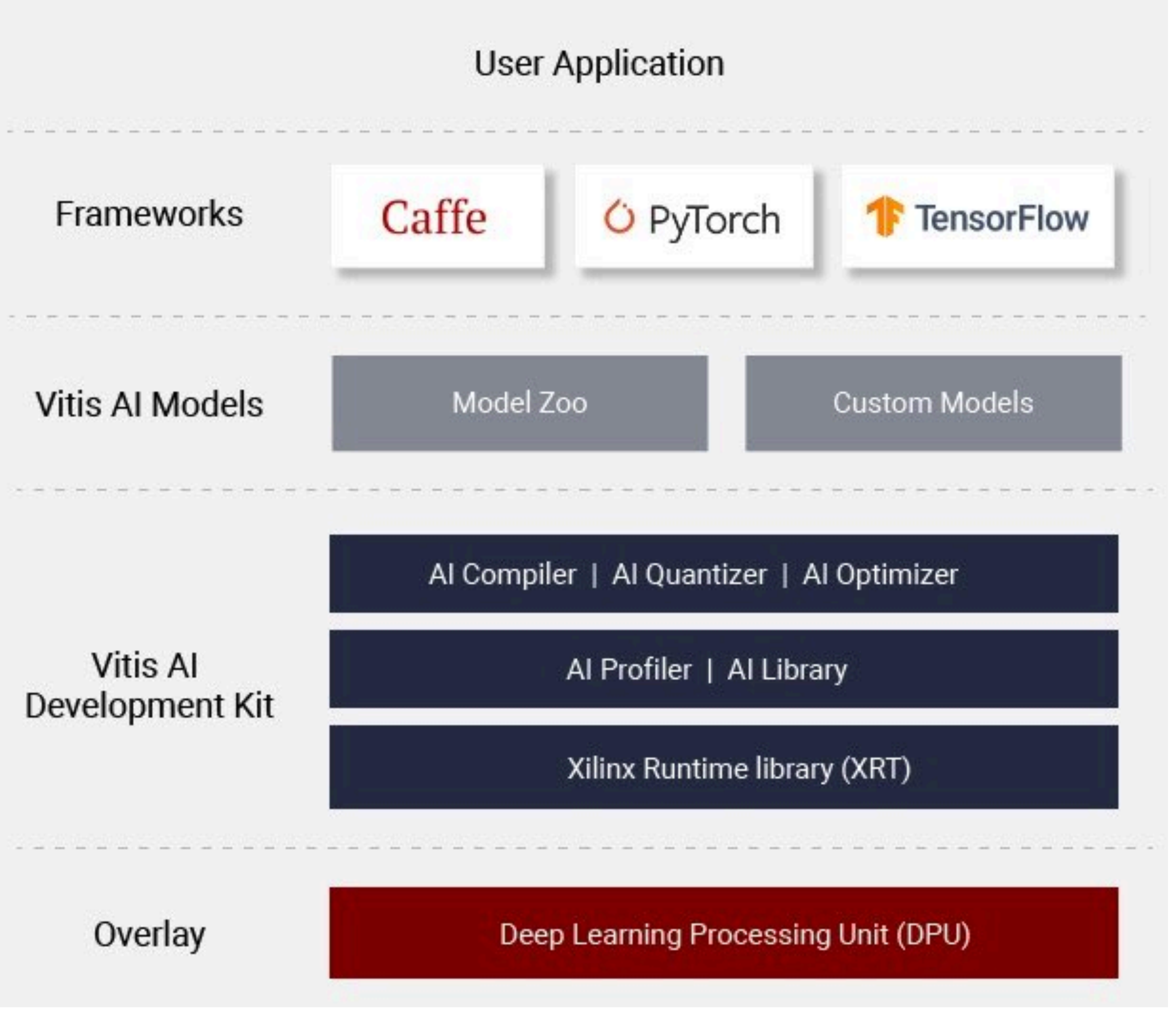
~x40 larger event rate sustainable wrt CPUs

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 molto semplice anche se a volte un pò involuto)
  - sfrutta bene le DPU degli acceleratori Xilinx
  - compatibile con tensorlfow/keras, pytorch, …
  - fornisce una libreria di modelli preaddestrati 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
    - latenze tipiche simili a quelle ottenibili con GPU: 0.1-1 ms/inferenza, OK per trigger di alto livello non per trigger hw
  - non tutte le tiplogie di layer e attivazioni neurali sono ancora state implementate
  - alcune funzioni interessanti e utili non sono free (pruner, profiler)

NOTA: per h/w intel/altera esiste l'ambiente equivalente OpenVINO

VitisAI STACK



VitisAI Model Zoo

Rich Models from Tensorflow, Caffe and Pytorch

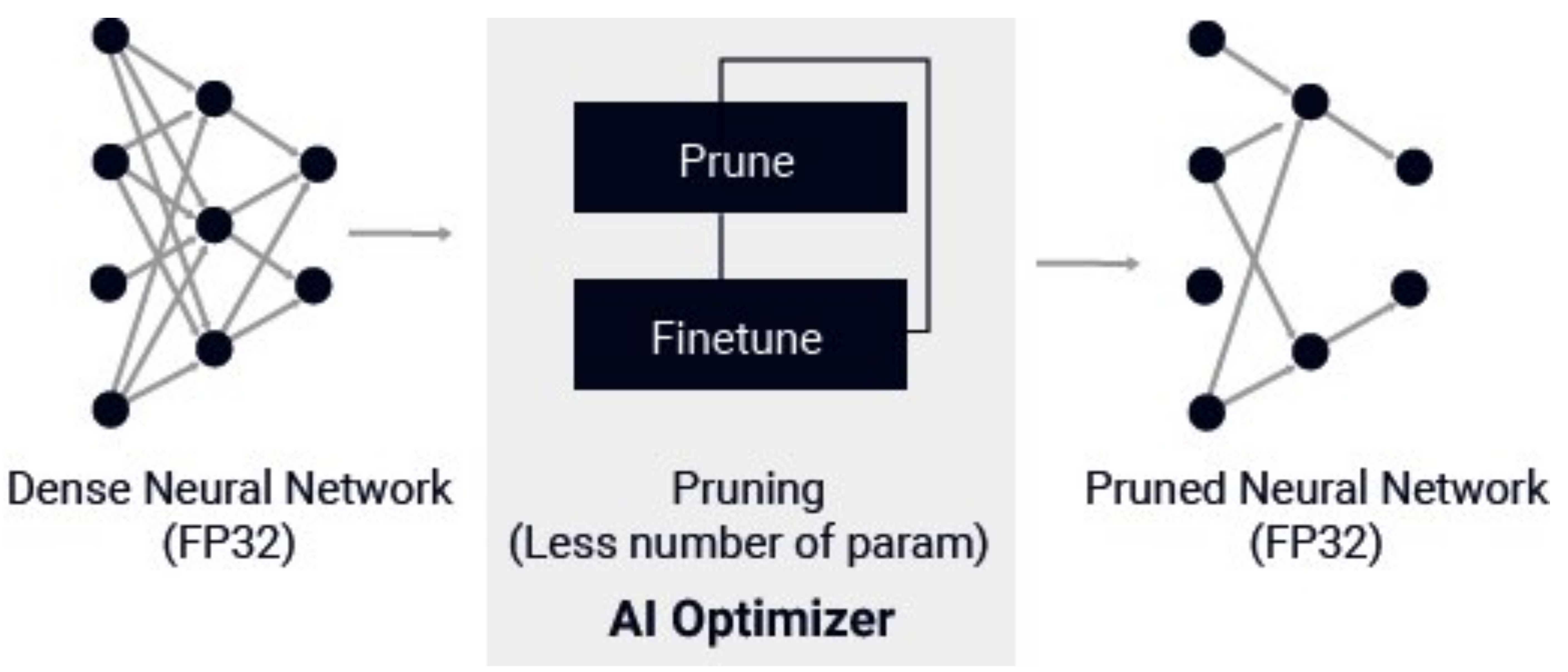
Open and Free on Github for All Developers

Advanced Optimization, Including Pruning, Applied

Retrainable with Custom Dataset

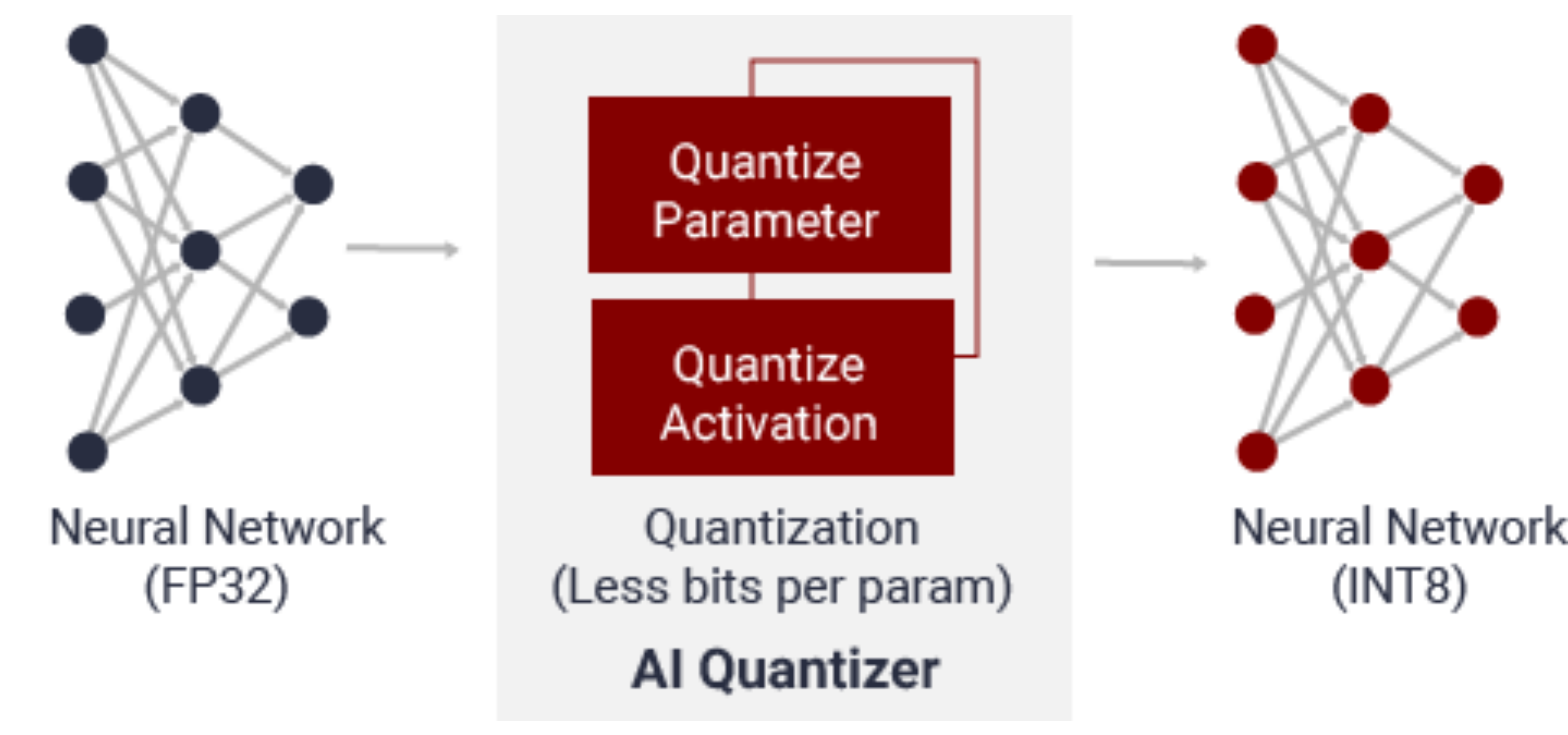
AI Optimiser

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



AI Quantizer

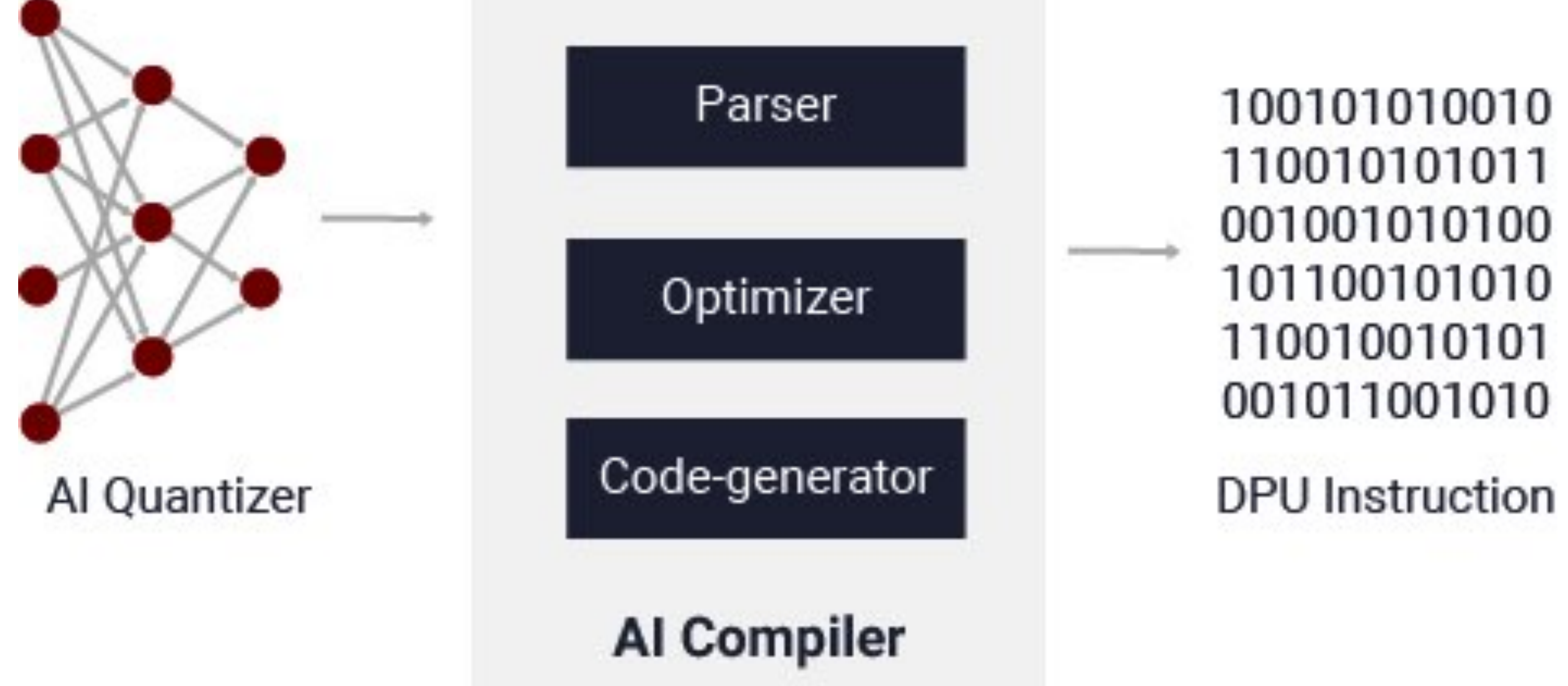
- converte i pesi delle reti e le attivazioni da floating point f32 a fixed-point INT8
- migliori risultati rispetto ad utilizzare le librerie tensorflow (inserisce nel modello quantizzato meta-informazioni che vengono sfruttate durante la compilazione del modello per lo specifico hardware)

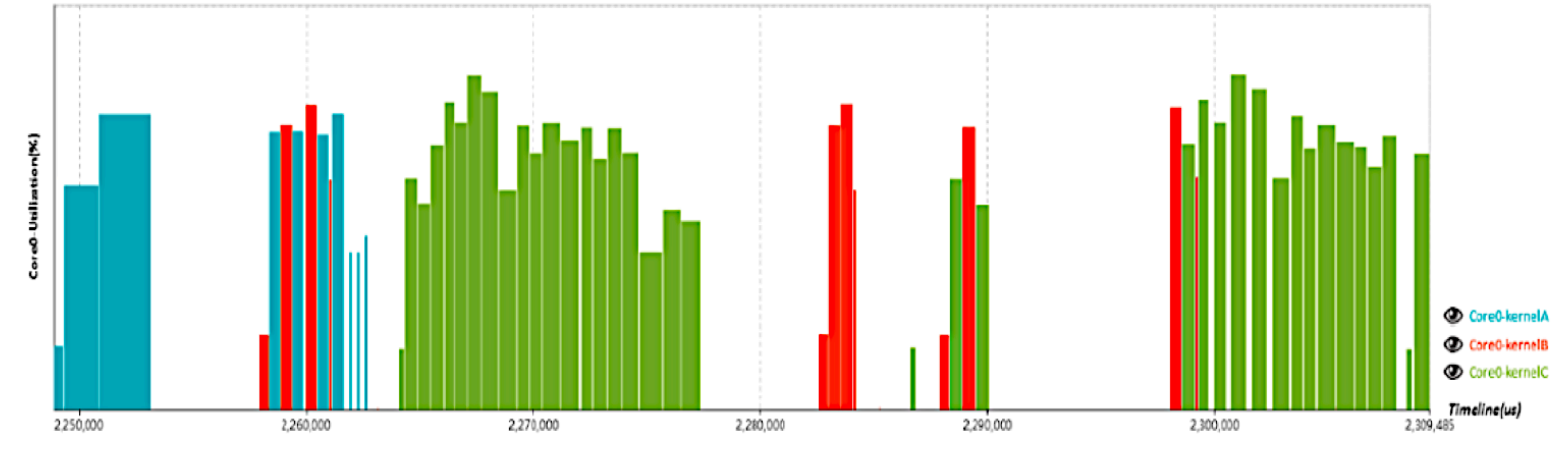


sia QAT che PQT

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