# 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

- oggi:
  - **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
- domani:
  - **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

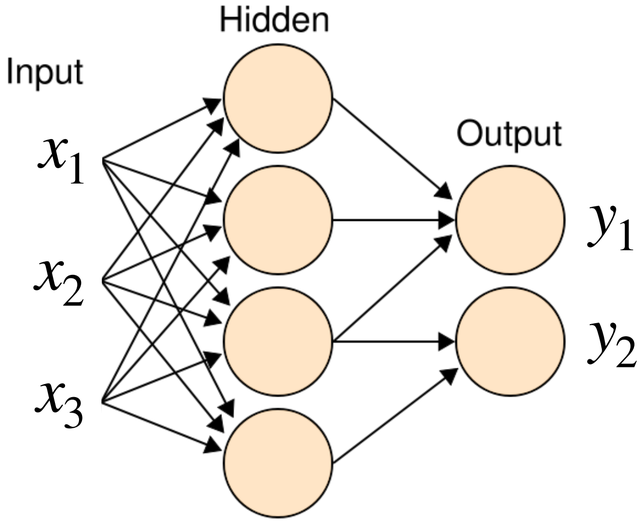
# RICHIAMI ANN/CNN

# ARTIFICIAL NEURAL NETWORKS

- costituiscono l'esempio più noto di modelli di ML non-lineari e sono alla base delle moderne tecniche di DL
- ANN: modello matematico in grado di approssimare con grande precisione una generica funzione:

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

  - originariamente introdotte in analogia con le reti neurali biologiche, in un linguaggio più matematico/computazionale una DNN è una composizione di funzioni connesse tra loro in catene descritte da grafi (es. Feed-Forward NN possono essere rappresentati come grafi diretti aciclici)

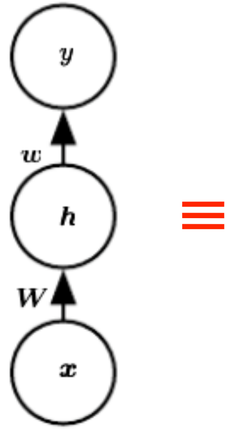



$$x = (x_1, x_2, x_3) \longrightarrow \quad y = (y_1, y_2)$$

- architettura: gruppo interconnesso di 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 (training) della rete

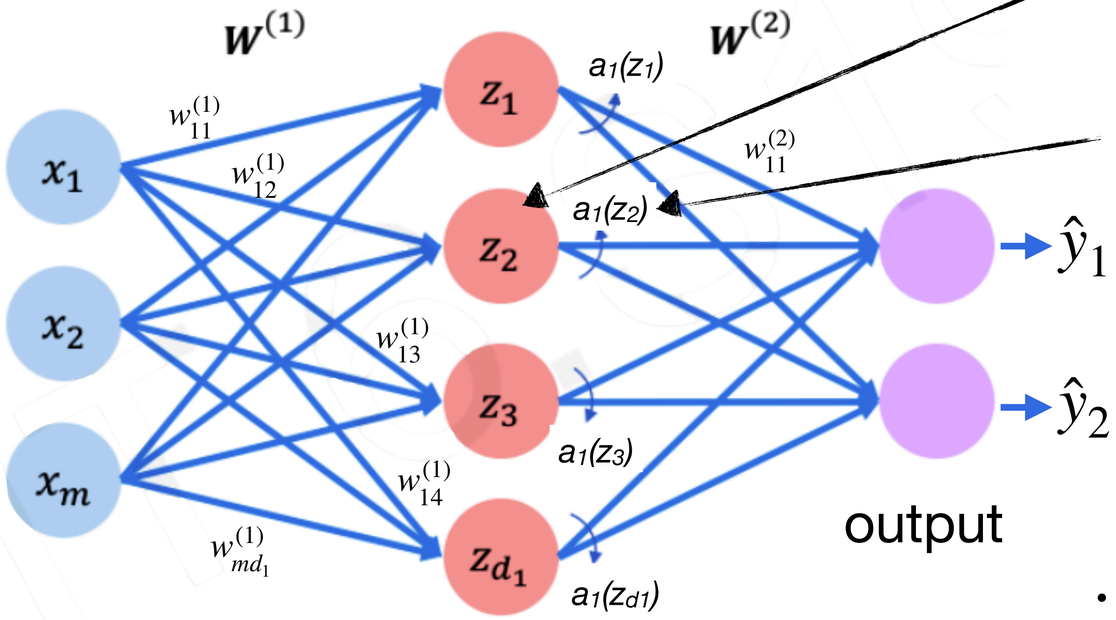
# MULTILAYER PERCEPTRONS FEED-FORWARD NN

- l'architettura base per le ANN
  - neuroni organizzati 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)




≡

FF-ANN acyclic direct graph




MLP con 1 HL

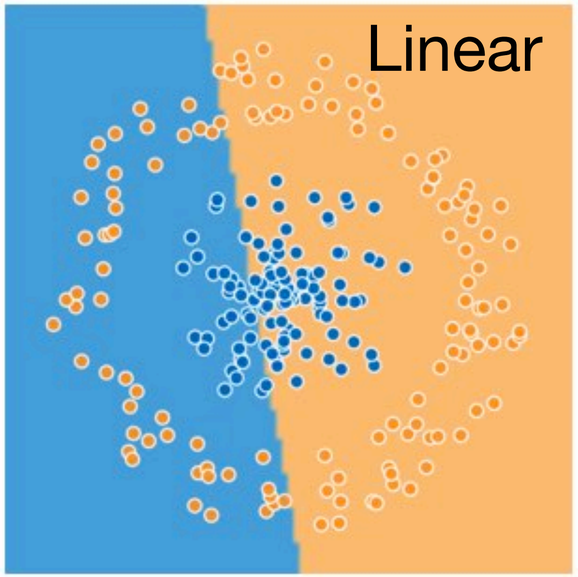
$$z_i = w_{0i}^{(1)} + \sum_{j=1}^{m} x_j w_{ji}^{(1)}$$ somma sinaptica

$$z_i' = a_1(z_i)$$ $a_1(\cdot)$: funzione di attivazione

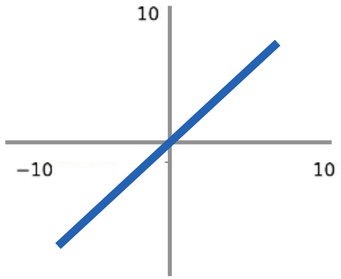
$$\hat{y}_i = a_2\left(w_{0i}^{(2)} + \sum_{j=1}^{d_1} a_1(z_j) w_{ji}^{(2)}\right)$$

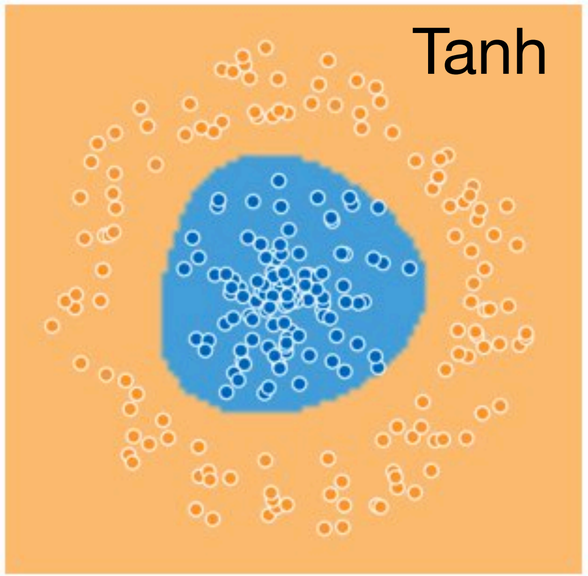
- risposta della NN determinata da:
  - pesi: (weight) $w_{ij}$
  - topologia della rete (#layers, size of each layer, …)
  - funzioni di attivazione di ogni layer

# funzioni di attivazione non-lineari permettono di apprendere pattern complessi…

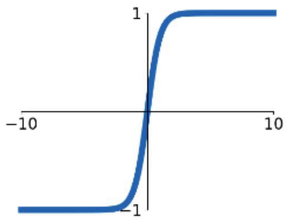


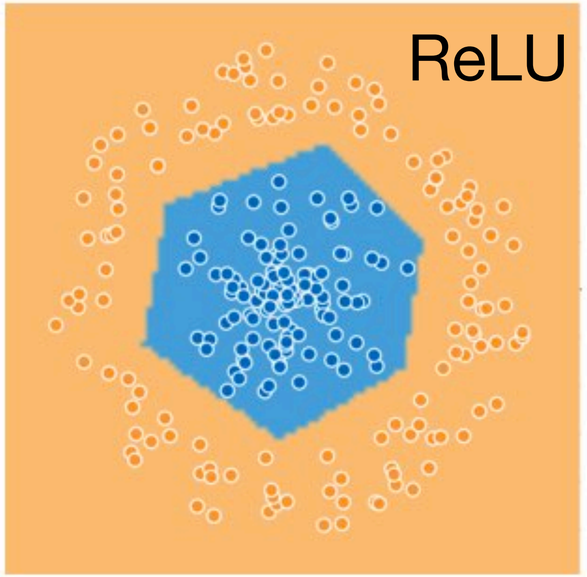

$$a(z) = z$$



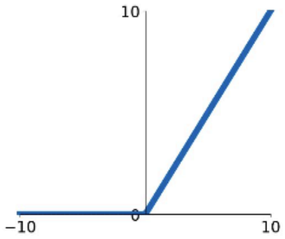




$$a(z) = \tanh[z]$$




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



# TRAINING DI UNA ANN

- 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 utilizzata per ottimizzare i pesi: discesa lungo il gradiente con back-propagation
- ottimizzazione degli iperparametri: tipicamente tramite approcci euristici (grid o random search, bayesian-opt, …)

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

$n_h$: numero di neuroni nel layer hidden

$n_{var}$: numero di neuroni nel layer di input

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

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 (loss) che misura la "distanza" tra $\hat{y}(x^{(i)})$ e $y^{(i)}$:

Esempio MSE

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

- 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

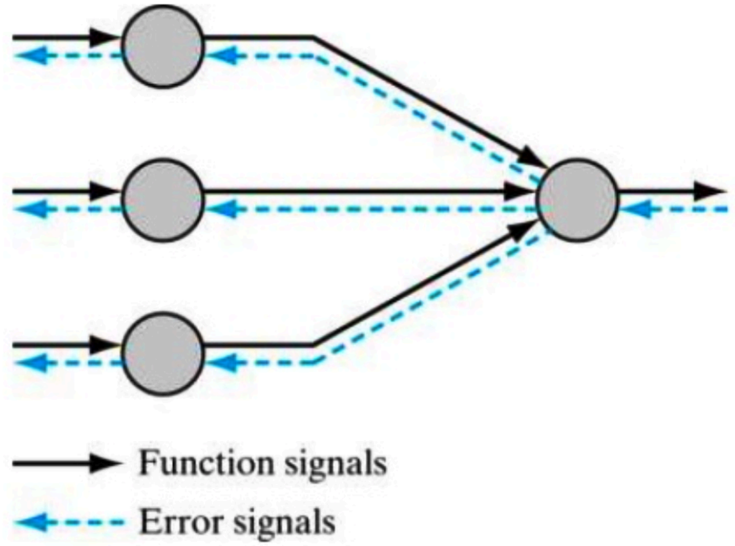
⇒ procedura di Back-Propagation

```
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 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)

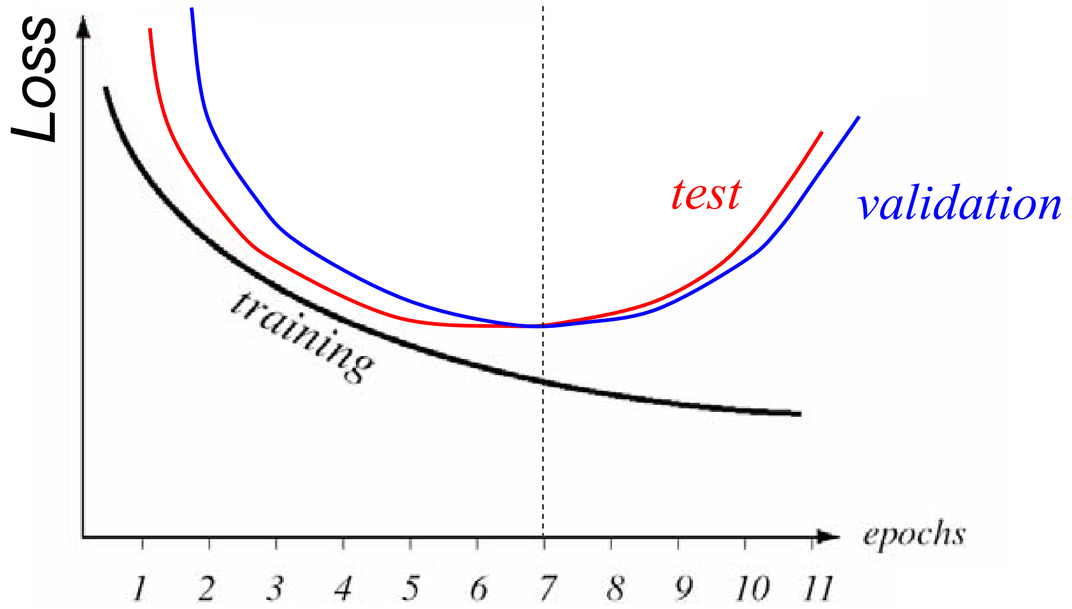
- ogni neurone (hidden o di output) riceve e confronta i segnali di funzione e di errore
- la back-propagation consiste in una semplificazione del calcolo del gradiente ottenuta applicando in modo ricorsivo la regola di derivazione di funzioni composte (chain rule)



# 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 e il modello non ha ancora appreso dai dati
- 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 e per ottenere il migliore compromesso tra varianza e bias 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

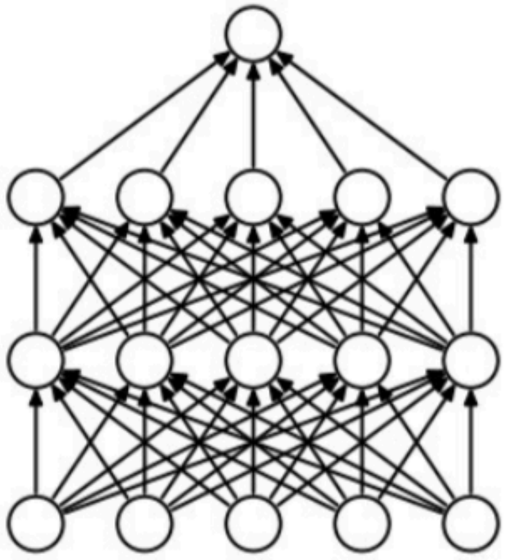
- velocità (di training e di inferenza):
  - varie tecniche per accelerare la convergenza:
    - momentum, adaptive learning rate (Adam o RMSProp), etc …
    - funzioni di attivazione che non saturano (tipo ReLU)
    - batch/layer normalisation
    - compressione/semplificazione delle ANN
    - processori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

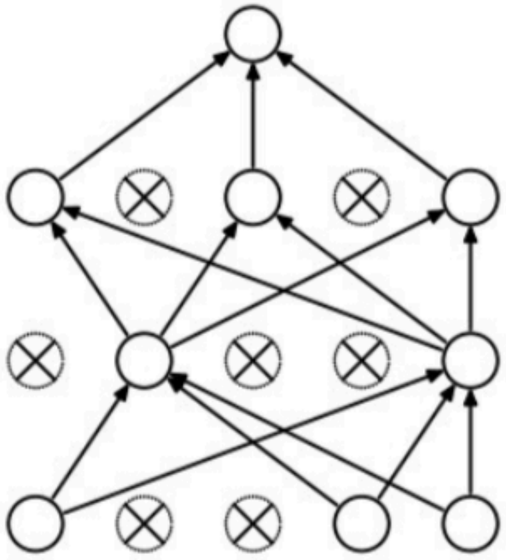

- hardcore overfitting:
  - è il problema principale delle reti neurali con architettura complessa con molti layer e molti pesi
  - conseguenza inevitabile del trade-off tra varianza e bias (generalizzazione)
  - soluzione: tecniche di regolarizzazione (dropout, L1/L2/L1+2, …), e.g. vengono introdotti appositi vincoli che limitano la complessità del modello neurale (rendono il compito di apprendimento/adattamento al campione di training più difficile)

# ESEMPIO: REGOLARIZZAZIONE TRAMITE 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

prima





dopo

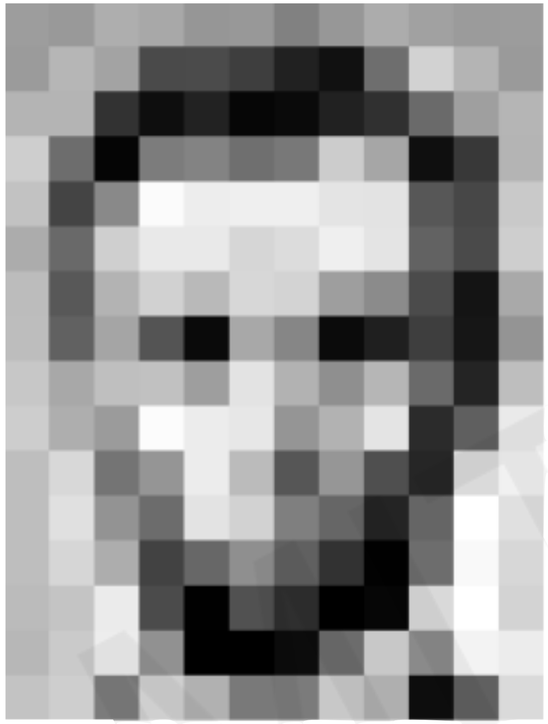
usata di routine nel contesto delle reti ConvNET in cui riesce ad aumentare drasticamente le prestazioni sul campione di test, può anche essere utilizzata come compressore del modello…

# ARCHITETTURE PER ANALISI DI IMMAGINI: CNN

- Le Convolutional NN 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
    - località delle feature: identificare una sub-feature richiede pochi pixel concentrati in una porzione limitata dell'immagine stessa
    - 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

# COME UNA ANN “VEDE” UN’IMMAGINE

le immagini dal punto rivista del computer sono sostanzialmente matrici (tensori) di numeri



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 157 | 153 | 174 | 168 | 150 | 152 | 129 | 151 | 172 | 161 | 155 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 105 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 96 | 50 | 2 | 109 | 249 | 215 |
| 187 | 196 | 235 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 123 | 207 | 177 | 121 | 123 | 200 | 175 | 13 | 96 | 218 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 157 | 153 | 174 | 168 | 150 | 152 | 129 | 151 | 172 | 161 | 155 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 105 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 96 | 50 | 2 | 109 | 249 | 215 |
| 187 | 196 | 235 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 195 | 206 | 123 | 207 | 177 | 121 | 123 | 200 | 175 | 13 | 96 | 218 |

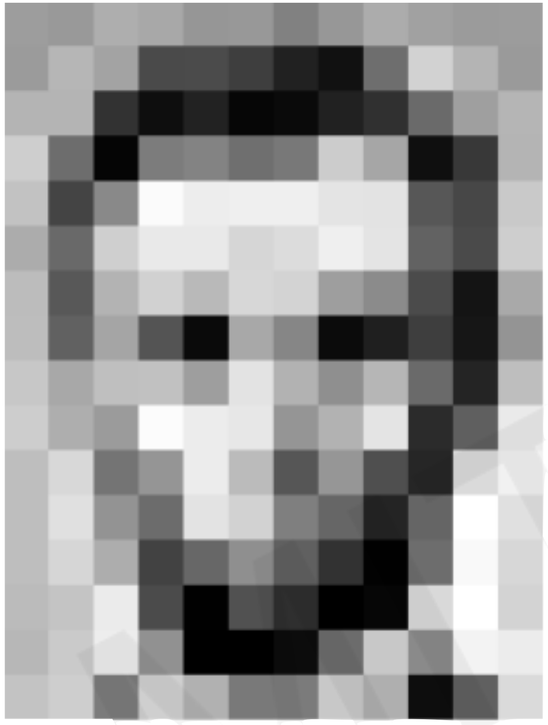
immagine a scala di grigi a 8bit: 12x16x1 intensità $\in [0,256]$

immagine a colori a n-bit: $m_1 x m_2 x 3$  ogni intensità RGB $\in [0,2^n]$

*da MIT AI course*

# COME UNA ANN “VEDE” UN’IMMAGINE

Task di classificazione:



| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 157 | 153 | 174 | 168 | 150 | 152 | 129 | 151 | 172 | 161 | 155 | 156 |
| 155 | 182 | 163 | 74 | 75 | 62 | 33 | 17 | 110 | 210 | 180 | 154 |
| 180 | 180 | 50 | 14 | 34 | 6 | 10 | 33 | 48 | 106 | 159 | 181 |
| 206 | 109 | 5 | 124 | 131 | 111 | 120 | 204 | 166 | 15 | 56 | 180 |
| 194 | 68 | 137 | 251 | 237 | 239 | 239 | 228 | 227 | 87 | 71 | 201 |
| 172 | 105 | 207 | 233 | 233 | 214 | 220 | 239 | 228 | 98 | 74 | 206 |
| 188 | 88 | 179 | 209 | 185 | 215 | 211 | 158 | 139 | 75 | 20 | 169 |
| 189 | 97 | 165 | 84 | 10 | 168 | 134 | 11 | 31 | 62 | 22 | 148 |
| 199 | 168 | 191 | 193 | 158 | 227 | 178 | 143 | 182 | 106 | 36 | 190 |
| 205 | 174 | 155 | 252 | 236 | 231 | 149 | 178 | 228 | 43 | 95 | 234 |
| 190 | 216 | 116 | 149 | 236 | 187 | 86 | 150 | 79 | 38 | 218 | 241 |
| 190 | 224 | 147 | 108 | 227 | 210 | 127 | 102 | 36 | 101 | 255 | 224 |
| 190 | 214 | 173 | 66 | 103 | 143 | 96 | 50 | 2 | 109 | 249 | 215 |
| 187 | 196 | 235 | 75 | 1 | 81 | 47 | 0 | 6 | 217 | 255 | 211 |
| 183 | 202 | 237 | 145 | 0 | 0 | 12 | 108 | 200 | 138 | 243 | 236 |
| 196 | 206 | 123 | 207 | 177 | 121 | 123 | 200 | 175 | 13 | 96 | 218 |

| Label | Probabilità |
|---|---|
| Lincoln | 0.8 |
| Washington | 0.1 |
| Jefferson | 0.05 |
| Obama | 0.05 |

*da MIT AI course*

# COME UNA ANN “VEDE” UN’IMMAGINE

La classificazione avviene identificando quante più caratteristiche presenti nell’immagine specifiche degli oggetti che vogliamo riconoscere …

NASO,
BOCCA,
OCCHI,
…

FARI,
RUOTE,
TARGA,
…

PORTA,
SCALE,
FINESTRE,
…

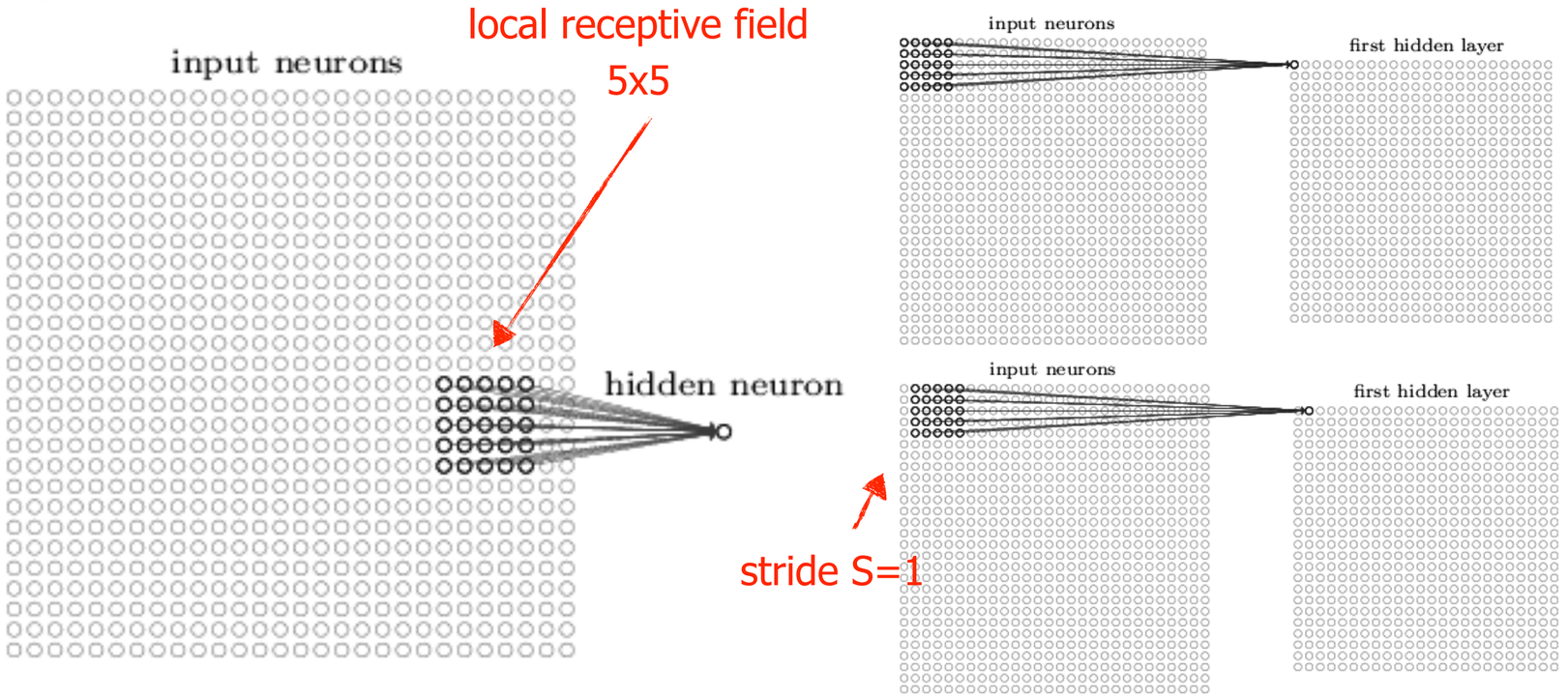
*da MIT AI course*

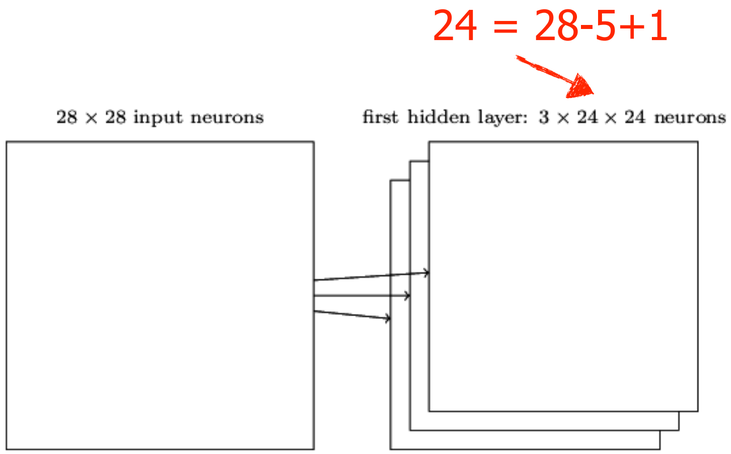
# CONVOLUTIONAL FEATURE EXTRACTION LAYER

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

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

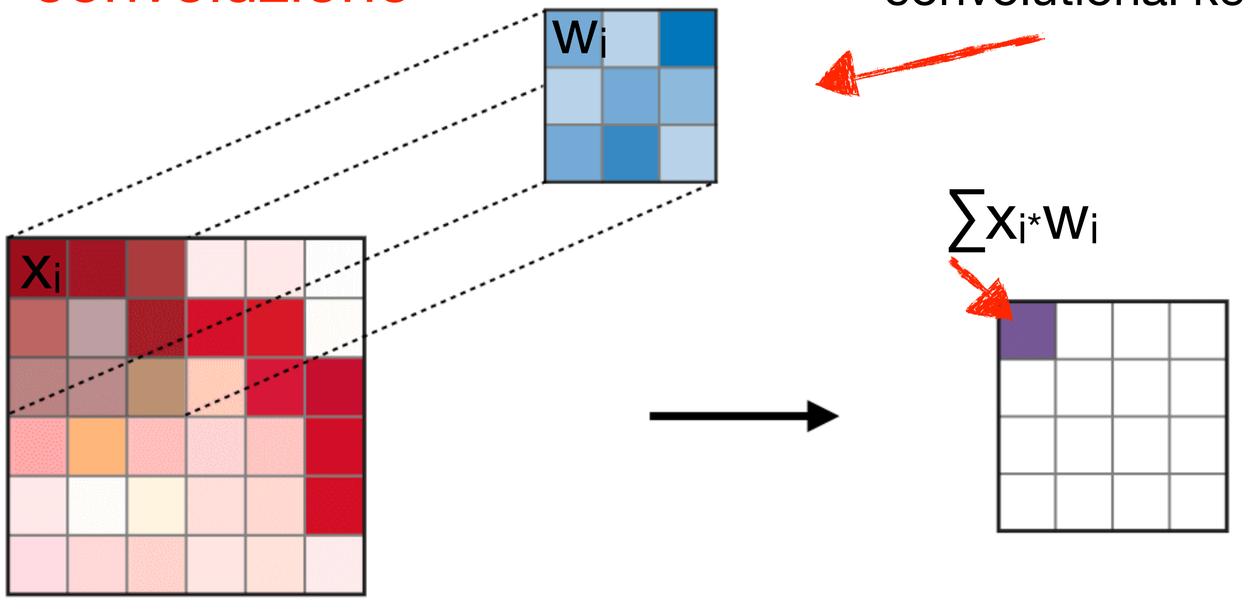
- i neuroni di input (uno per ognuno degli NxN pixel dell'immagine) NON sono fully connected con tutti i neuroni del layer nascosto. Le connessioni esistono solo per una piccola regione spaziale dell'immagine (local receptive field)
- il local receptive field viene fatto scorrere attraverso l'intera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

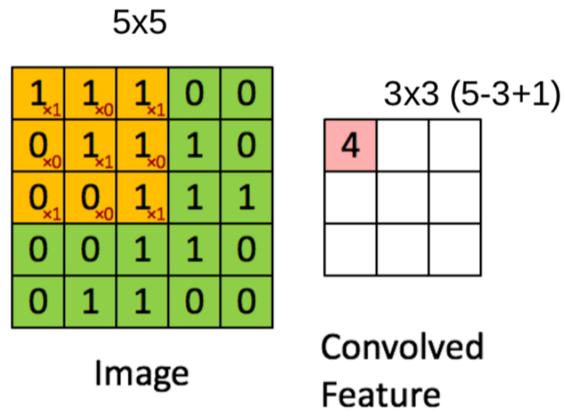


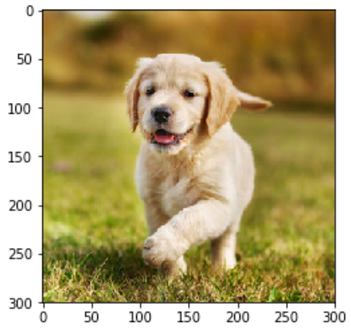



# l'operazione di convoluzione

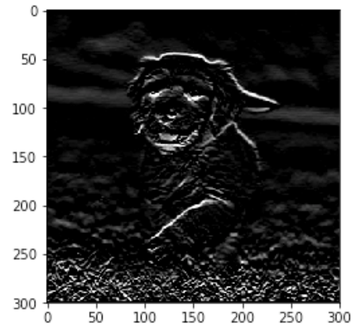
local receptive field o convolutional kernel



```
# 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 …

# FEATURE MAP PRODOTTE VS PESI DEL KERNEL

ORGINALE


SHARPEN


EDGE DETECT

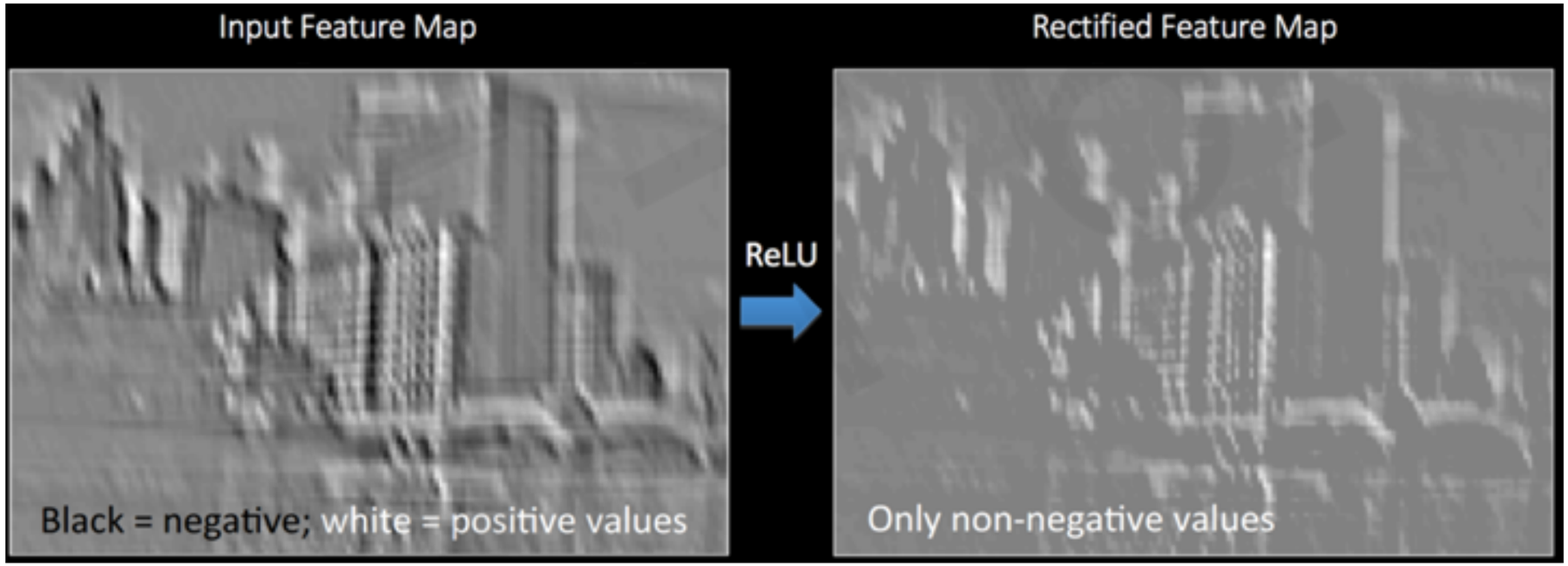

STRONG EDGE DETECT

differenti valori dei pesi enfatizzano caratteristiche diverse dell’immagine

# 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)

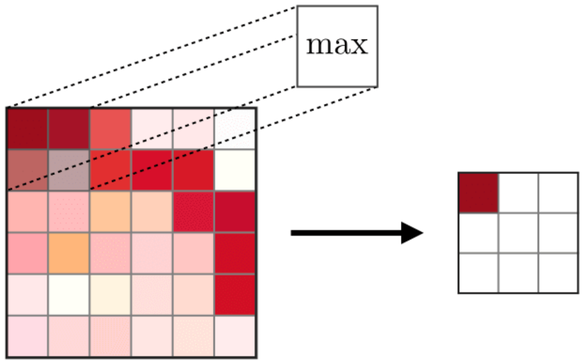
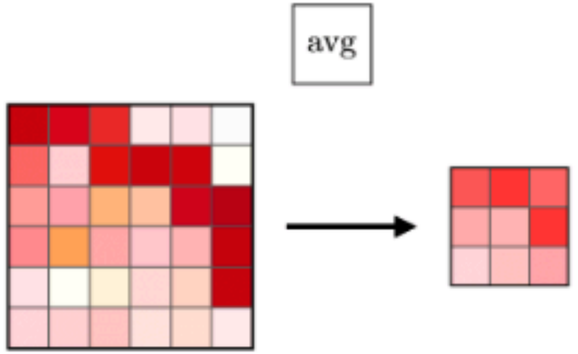
- enfatizzate solo alcune delle caratteristiche dominanti delle sub-feature selezionate dal filtro

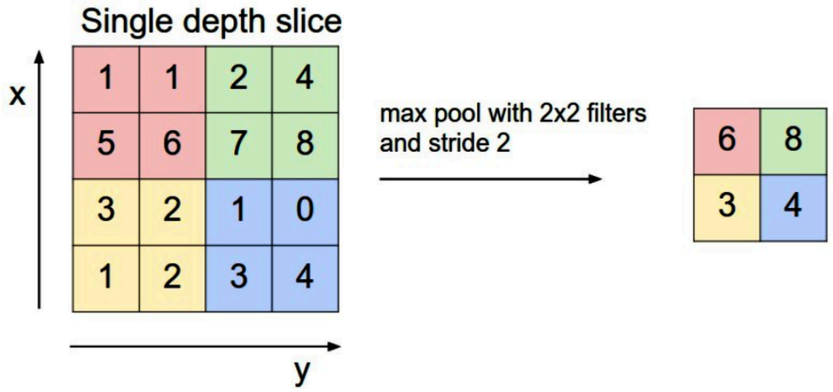



before ReLU

after ReLU

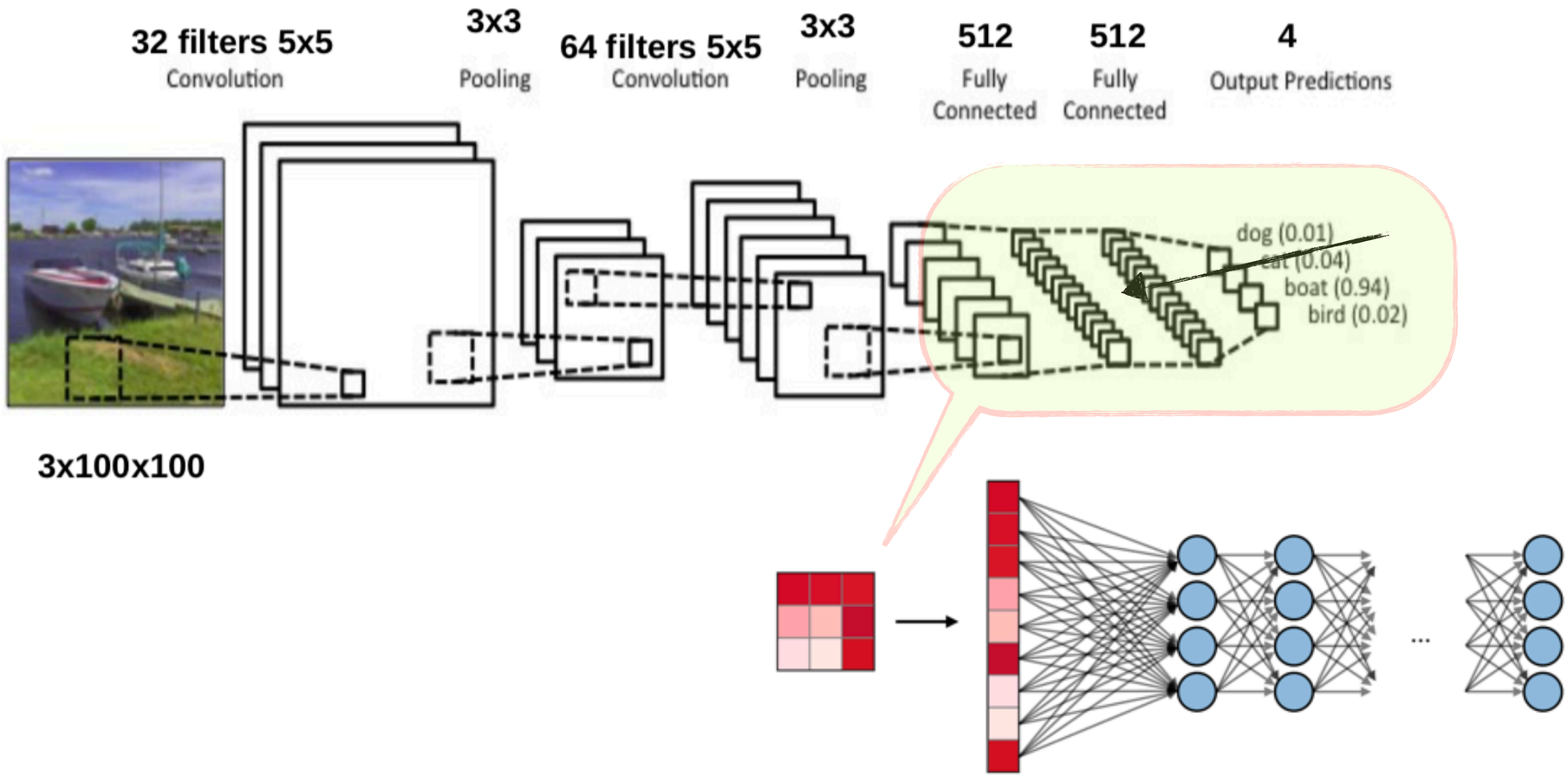
- pooling layers:
  - in aggiunta ai layer convolutionali una CNN è anche equipaggiata con layer detti di pooling, tipicamente applicati dopo ogni gruppo di layer convoluzionali. Lo scopo è quello di applicare un'operazione di downsampling: i.e. semplificare l'informazione in output dal layer convoluzionale 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 |   |   |
| Comments | • Preserves detected features<br>• Most commonly used | • Downsamples feature map<br>• Used in LeNet |



# CNN COMPLETA

- generalmente l'output dei layer convoluzionali viene connesso, via un layer di flattening, che trasforma il risultato dell'ultima convoluzione in un vettore 1D che sarà l'input di una MLP utilizzata per ottimizzare la funzione di loss rispetto alla task specifica della rete (classificazione, regressione, …) etc..

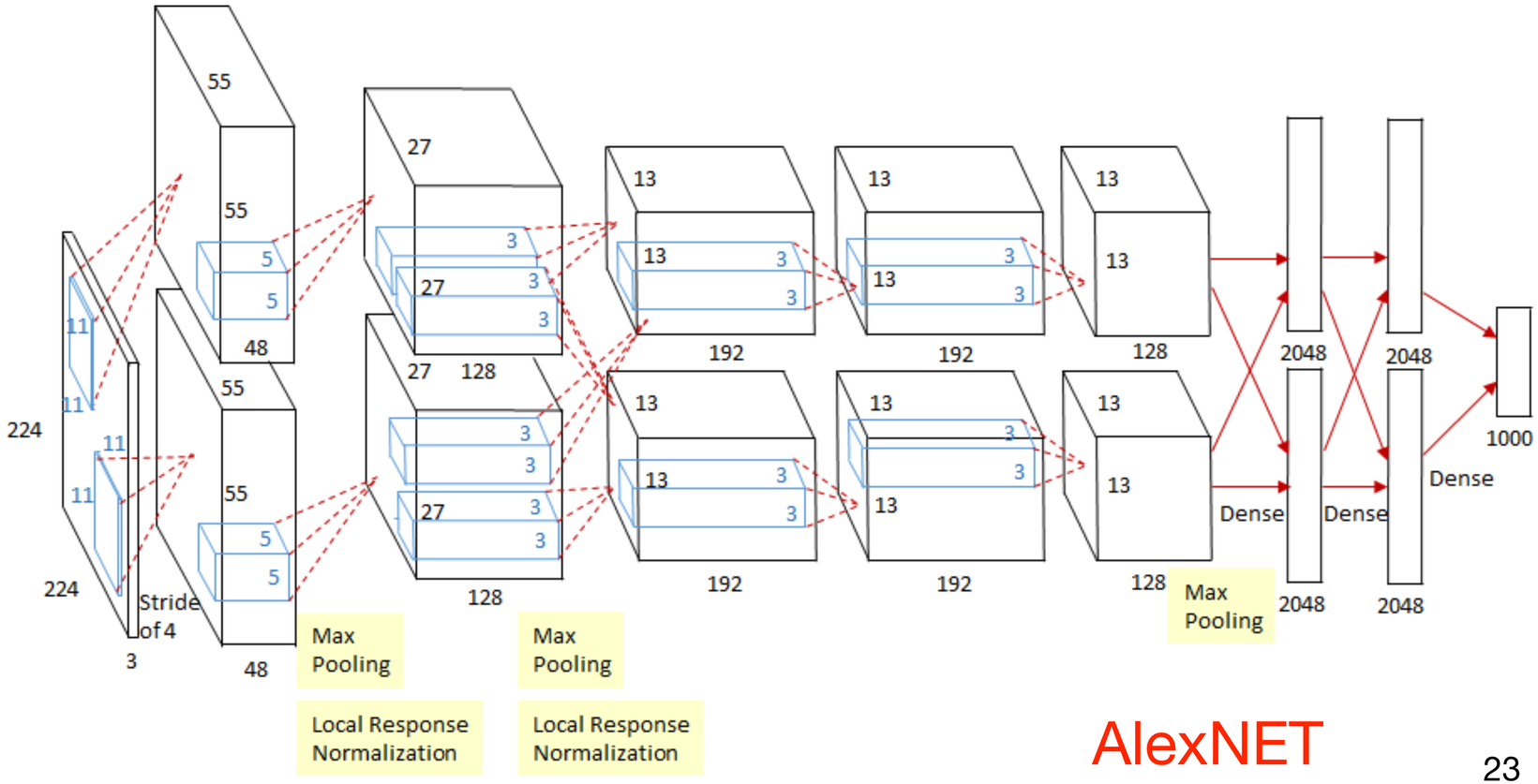


# LA PRIMA CNN AD ALTE PRESTAZIONI: ALEXNET

- Basata sull'architettura di Krizhevsky et Al. vincitrice del contest Imagnet 2012
- ambiente: Caffe framework  (Berkeley Vision Deep Learning framework: http://caffe.berkeleyvision.org)

stesso approccio top-down di LeNet + miglioramenti:

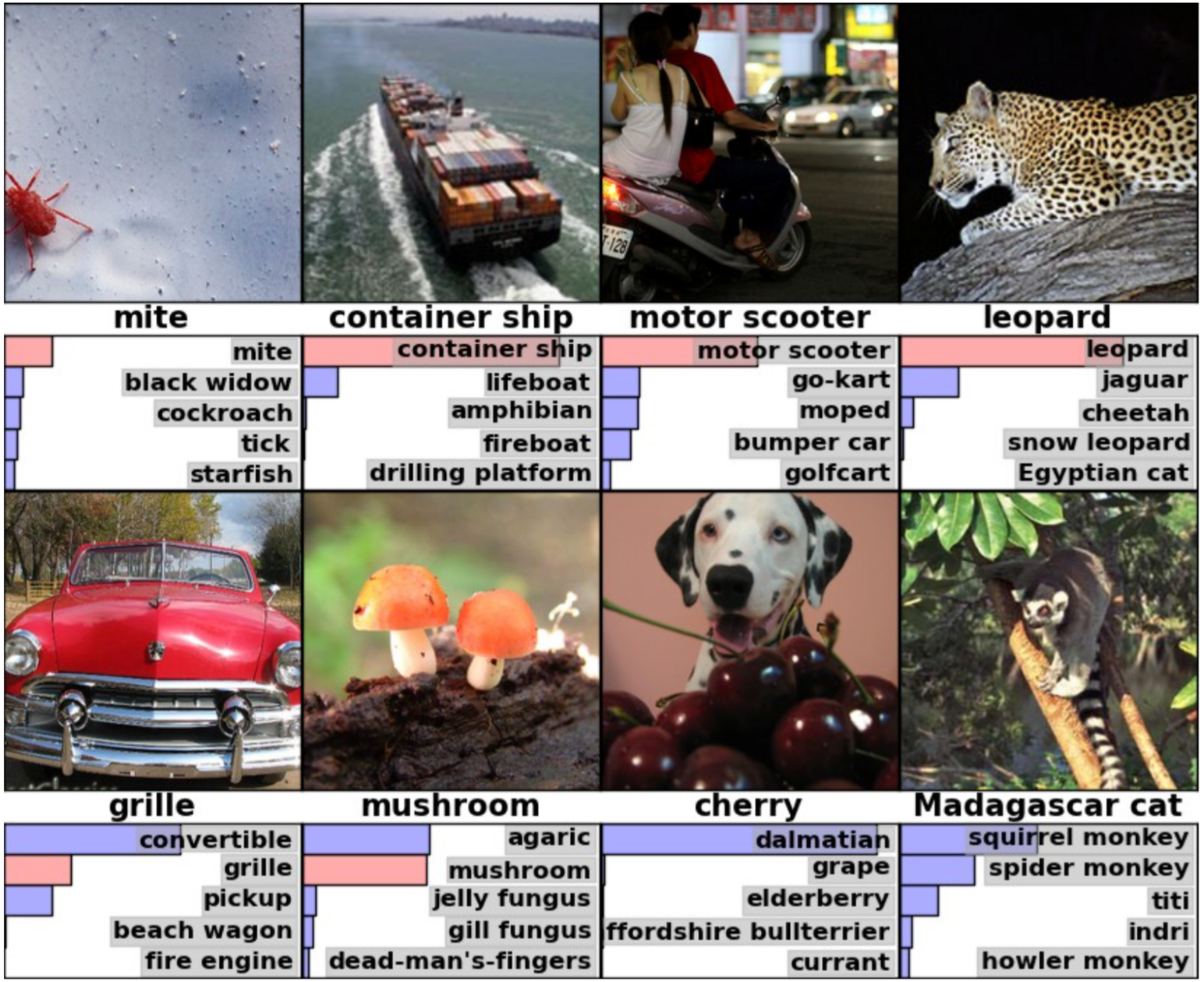
1. migliore back-propagation via ReLU
2. dropout regularisation
3. batch normalisation
4. data augmentation: random translation, rotation, crop
5. deeper architecture: più convolutional layers (7), i.e. identifica un maggior numero di feature elementari




AlexNET

# Validation classification

- feature initialised with white gaussian noise
- fully supervised training
- training on GPU NVIDIA for ~1 week
- 650K neurons
- 60M parameters
- 630M connections



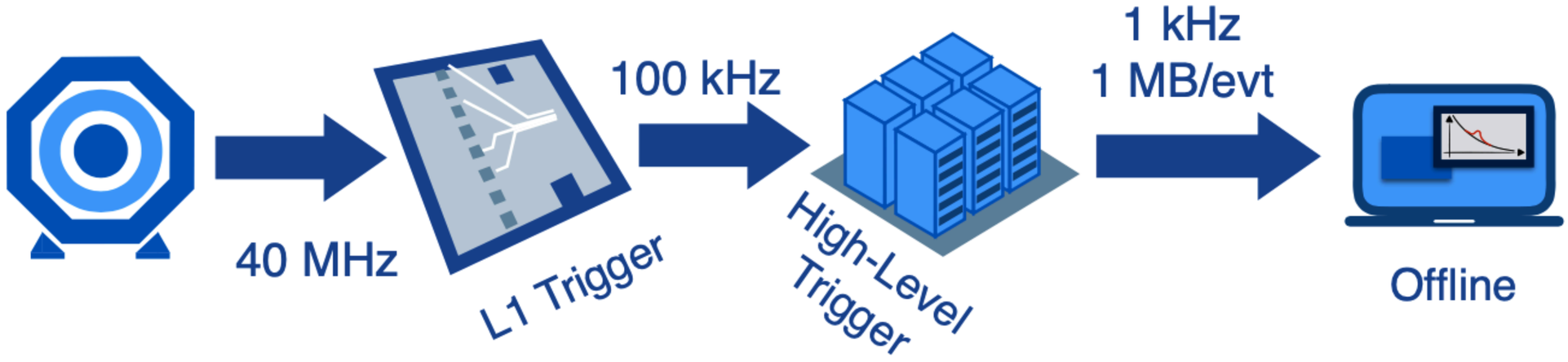
# DNN SU SISTEMI REAL-TIME
# &
# MODEL OPTMIZATION/COMPRESSION

# DNN NEI TRIGGER DI ESPERIMENTI DI FISICA DELLE ALTE ENERGIE

- grazie alle loro prestazioni le reti neurali artificiali vengono applicate già da diversi anni nella fisica delle alte energie, espandendo il potenziale di scoperta e di misura dei grandi esperimenti ai collider di particelle e permettendo di sviluppare nuove tecniche di rivelazione
- 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
    - l'uso di DNN consente di sviluppare algoritmi molto più flessibili rispetto a quelli usati tradizionalmente
    - possibilità di sfruttare l'informazione raw dei detector senza la tipica perdita di informazione e/o le difficoltà di implementazione legate al feature engineering
    - potrebbe 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

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




| | | | |
|---|---|---|---|
| latenza | O(100ns -1μs) | O(1ms -1s) | O(s) |
| computing device | ASIC<br>FPGA | CPU<br>GPU/FPGA | CPU/GPU/… |
| challenge | inferenza AI a bassissima latenza | inferenza AI a media latenza, alto throughput, flessibilità | |

# ARCHITETTURE DNN CONVENZIONALI E SISTEMI REAL-TIME

- 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 durante il training
    - 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:

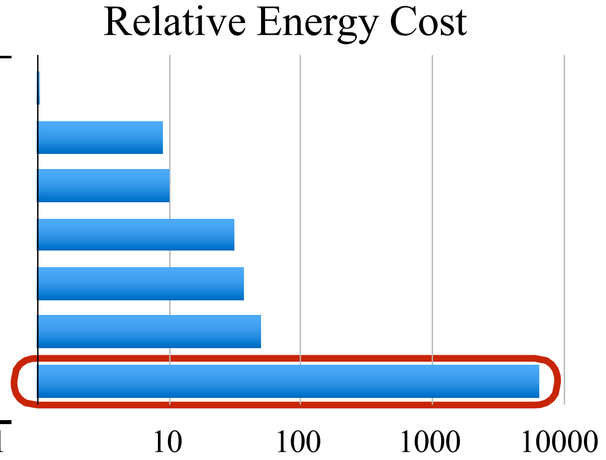
| | Error rate | Training time |
|---|---|---|
| ResNet18: | 10.76% | 2.5 days |
| ResNet50: | 7.02% | 5 days |
| ResNet101: | 6.21% | 1 week |
| ResNet152: | 6.16% | 1.5 weeks |

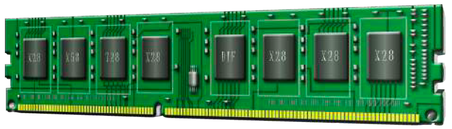
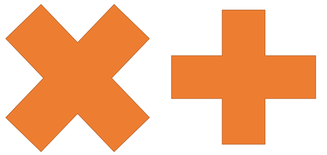
  - sia per l'inferenza: **non adatti per essere utilizzati in sistemi real-time a bassa latenza**

- **costo energetico:**
  - DeepMind GPT-3: 12M US$ per essere addestrato
  - già nel 2016 alphaGO richiedeva 3k US$ 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

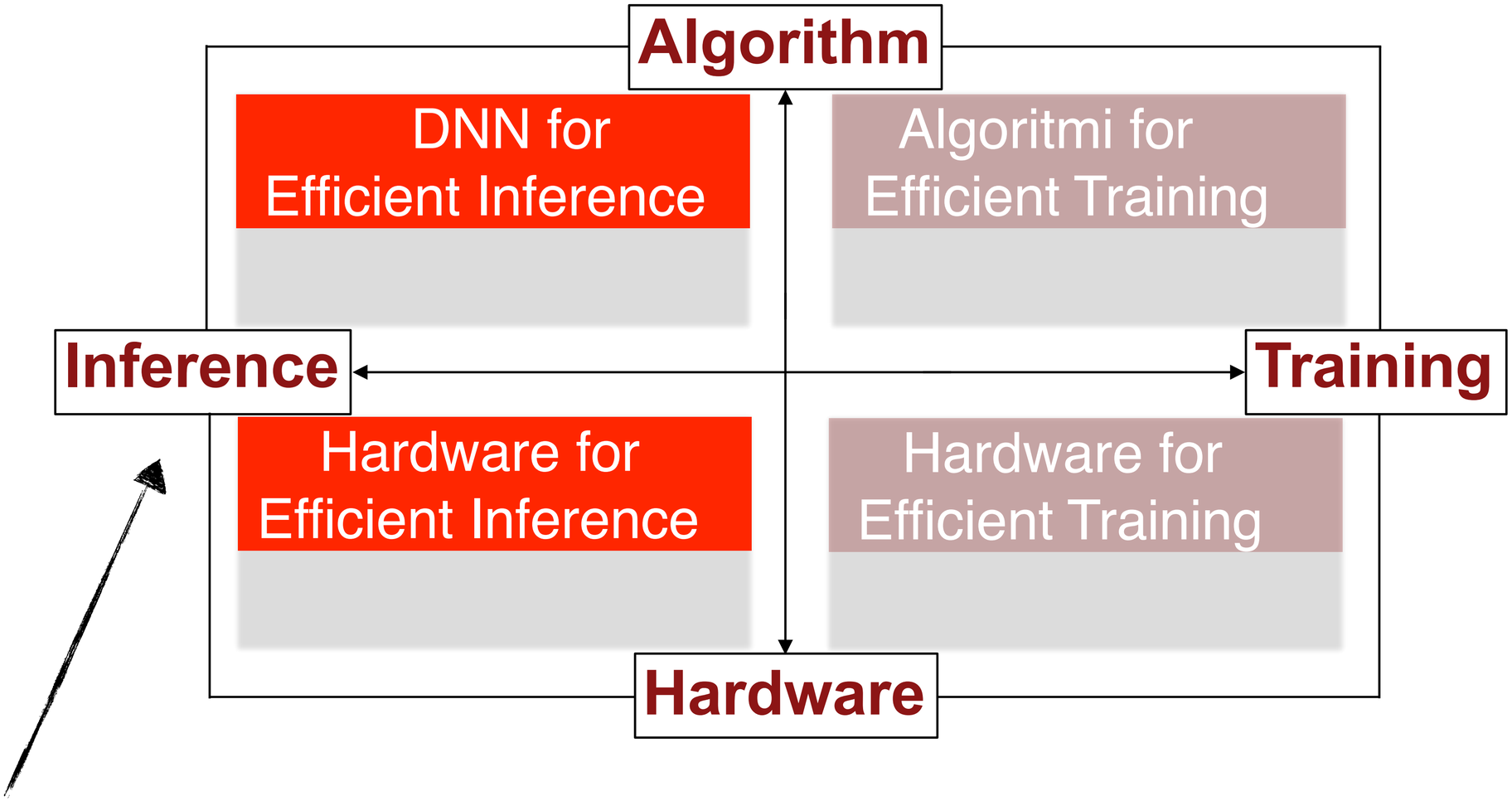
in sintesi: la dimensione della DNN è il problema principale sia per la latenza, sia per l'utilizzo delle risorse che per il consumo energetico

**possibile soluzione**

reti neurali compresse ed ottimizzate

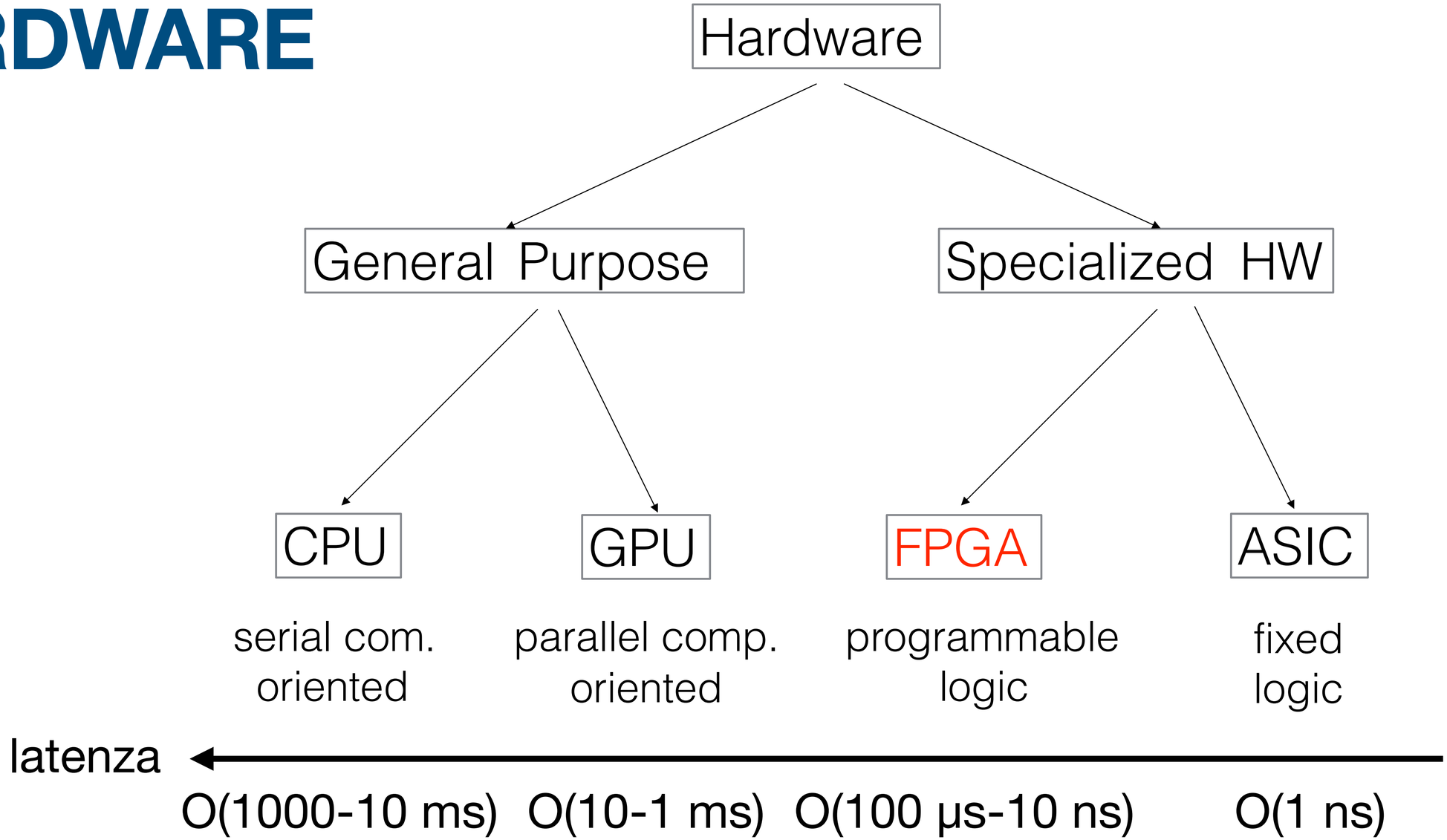
+

hw ottimizzato per inferenza efficiente, veloce, e a basso consumo energetico (FPGA)




regione rilevante per le applicazioni
di DNN a sistemi di trigger

# HARDWARE


Hardware
General Purpose
Specialized HW
CPU
GPU
FPGA
ASIC
serial com. oriented
parallel comp. oriented
programmable logic
fixed logic
latenza
O(1000-10 ms)
O(10-1 ms)
O(100 µs-10 ns)
O(1 ns)
tipico trigger HLT
tipico trigger HW

# ALGORITMI DI MODEL COMPRESSION

- GOAL: realizzare modelli di reti neurali lightweight, che siano:
  - veloci
  - efficienti dal punto di vista dell'uso della memoria
  - efficienti dal punto di vista energetico
- mantenendo al contempo buone prestazioni rispetto alla task per le quali sono state addestrate

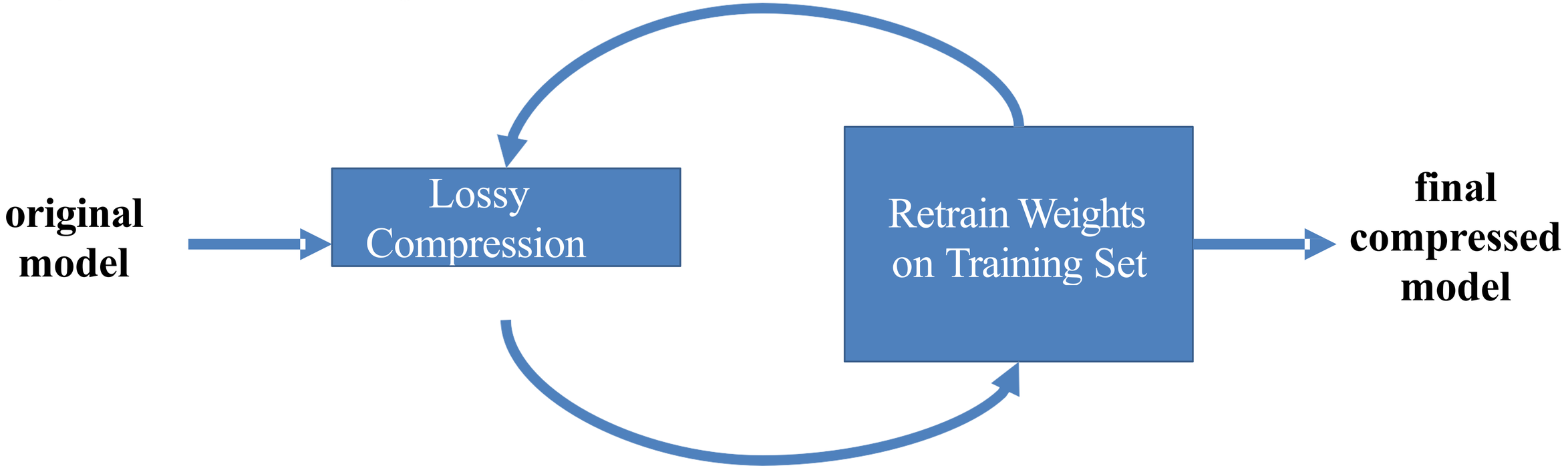
- Due tecniche di compressione principali e diversi paradigmi di addestramento:
  - post training compression: a partire da un modello complesso con prestazioni allo stato dell'arte si ottiene un modello lightweight eliminando parte delle informazioni nel modello originale
  - training aware compression: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

# TRAINING AWARE COMPRESSION 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 learning rate

spesso il fine tuning fornisce prestazioni migliori di un full training (specie con loss landscape complessi e dataset di taglia limitata)



# 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

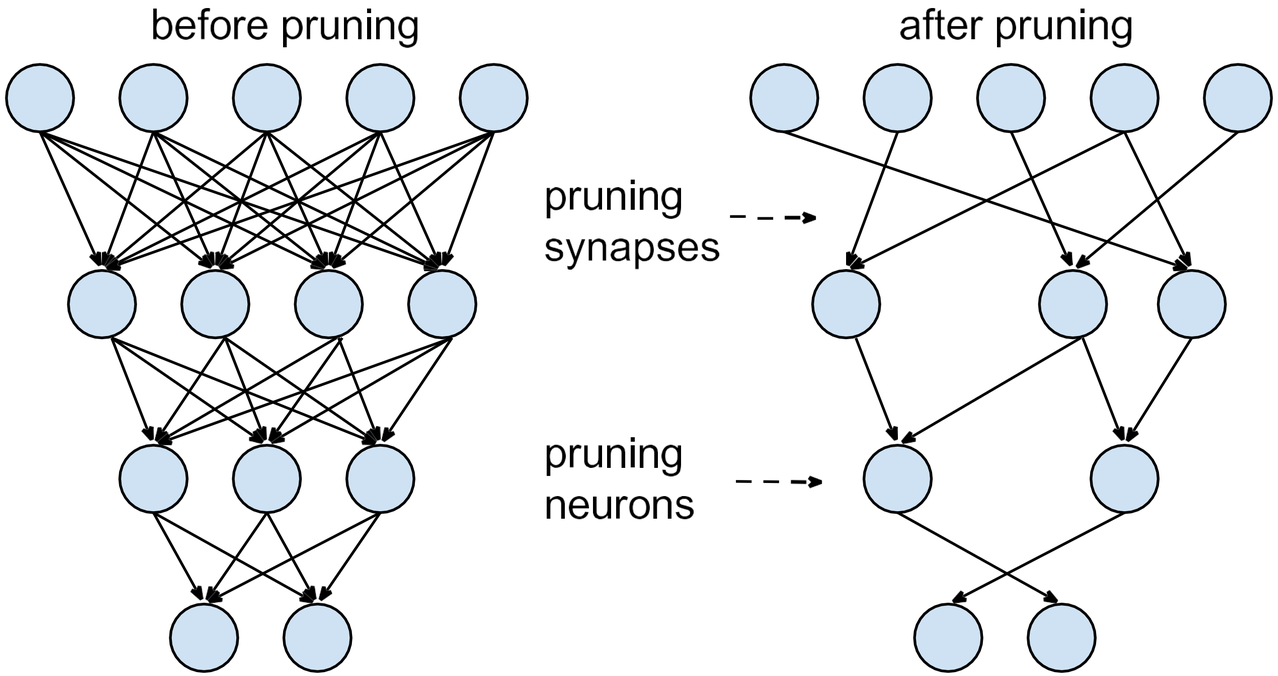
# PRUNING

- la tecnica di pruning (LeCun, NIPS ’89, Han, NIPS 2015) basata sulla grandezza dei pesi della rete, 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 fino a 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

50 Trillion Synapses

1000 Trillion Synapses

500 Trillion Synapses

This image is in the public domain

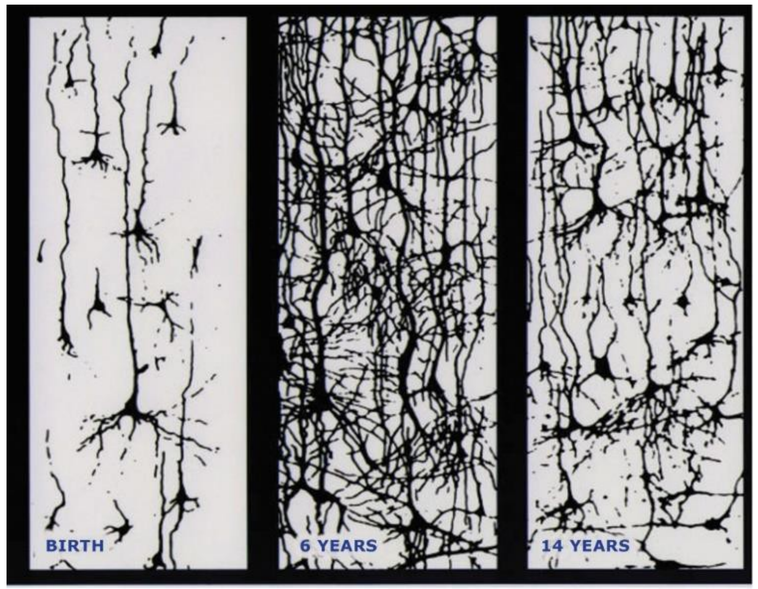
Newborn

This image is in the public domain

1 year old

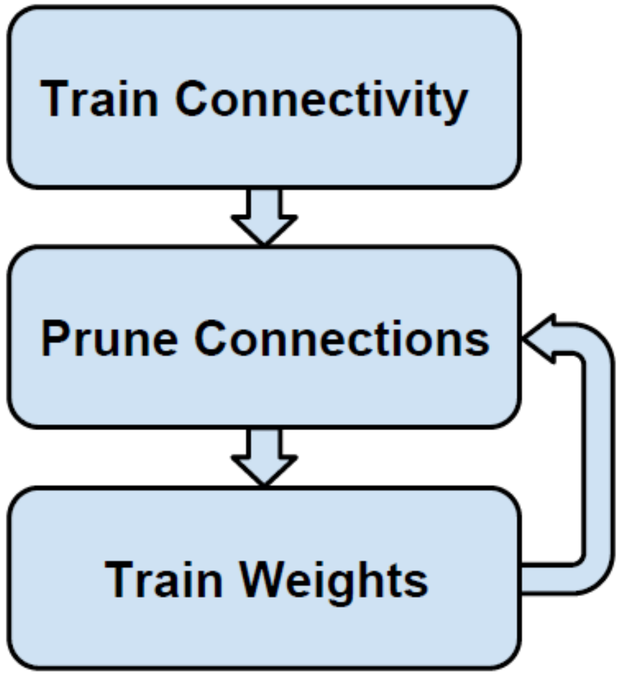
This image is in the public domain

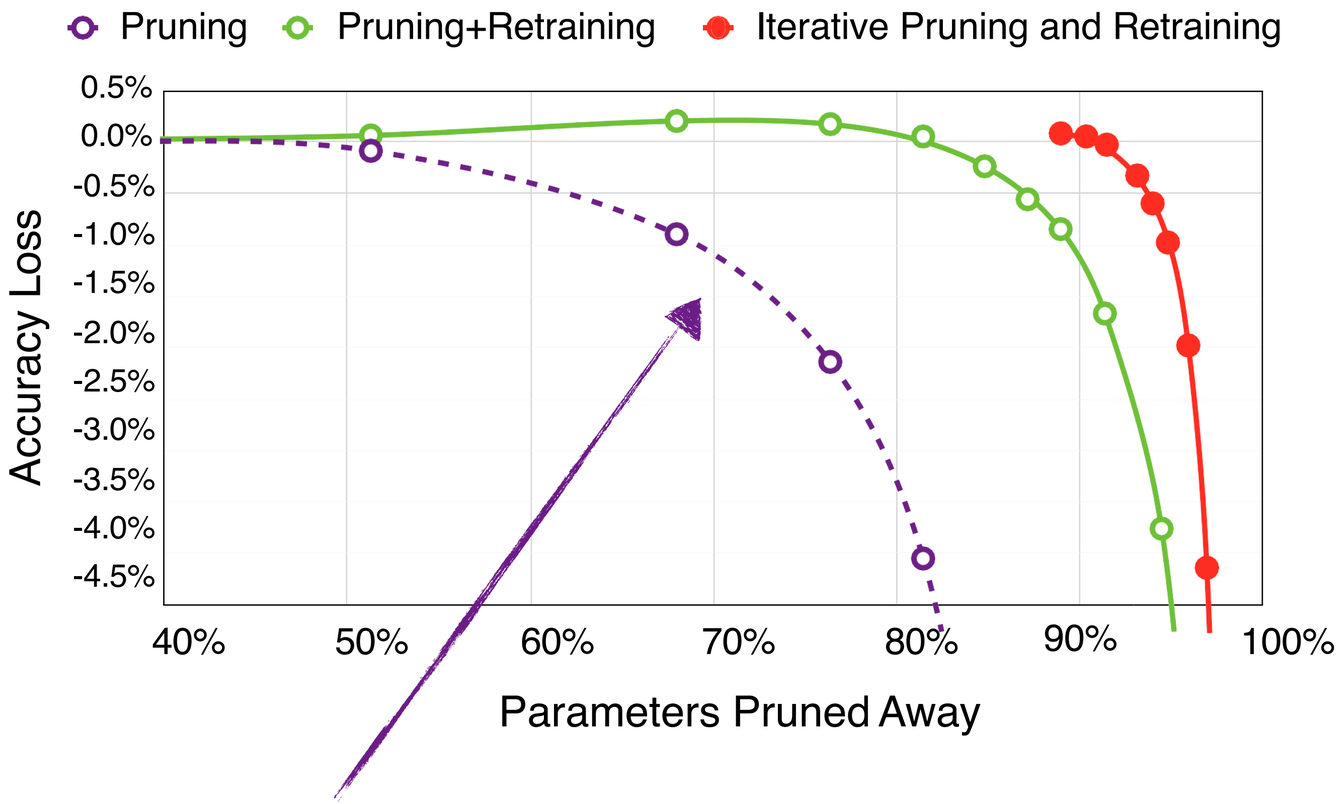
Adolescent




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

# PROCEDURA DI TRAINING AWARE PRUNING



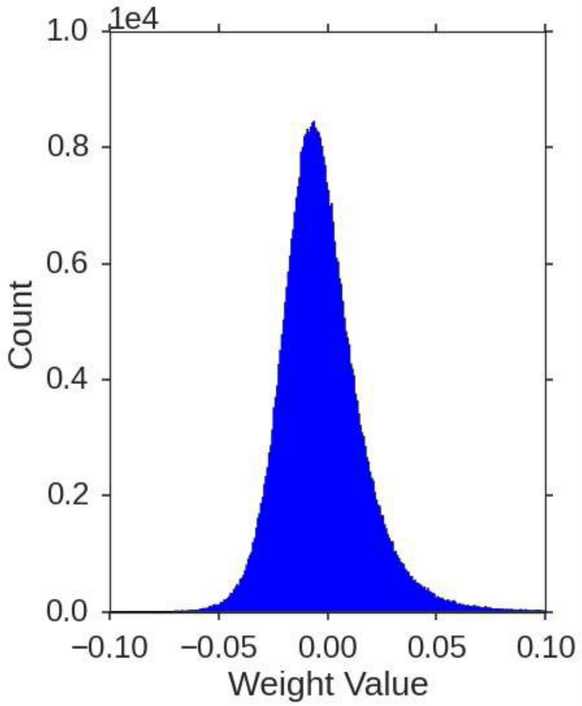




- remove weights for which $weight < threshold$
- retrain (fine tune) after pruning weights
- learn effective connections by iterative pruning

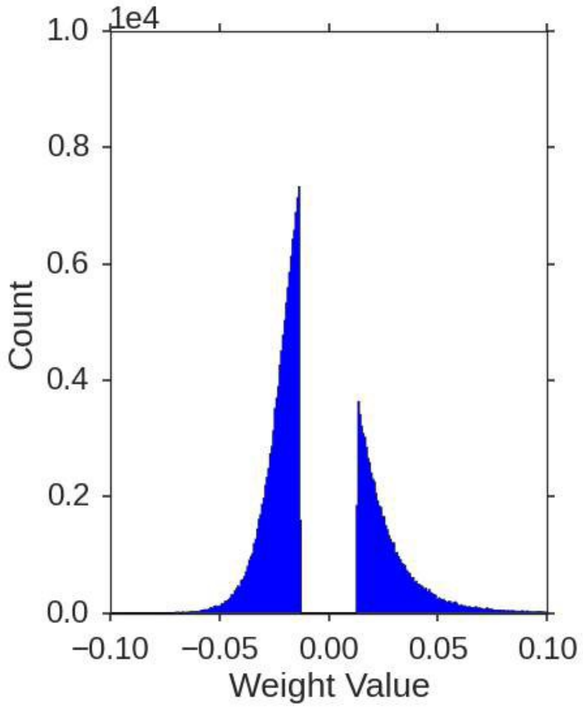
pruning after training (dropout) non è una tecnica molto efficace per forti compressioni

# DISTRIBUZIONE DEI PESI

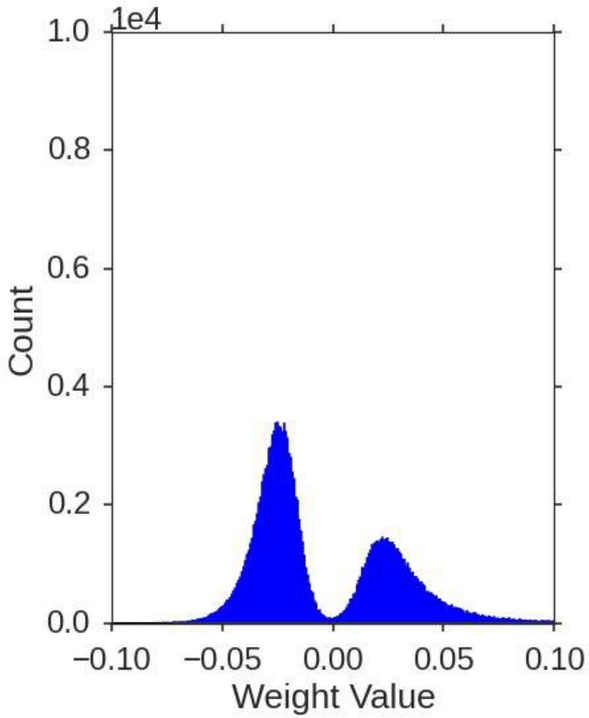
**Before Pruning**




**After Pruning**




**After Retraining**




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 quantization, più semplice ma spesso con perdita di prestazioni non recuperabile con fine tuning (è OK con bassi livelli di quantizzazione a 16/8 bit richiesti dalle DPU degli acceleratori commerciali per AI basati su FPGA)
  - TAQ: training aware quantization, richiede il ri-addestramento del modello quantizzato, ma porta spesso a prestazioni allo stesso livello (e 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 quantization)
- 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)): algoritmi “all-on-fpga”

# QUANTIZZAZIONE

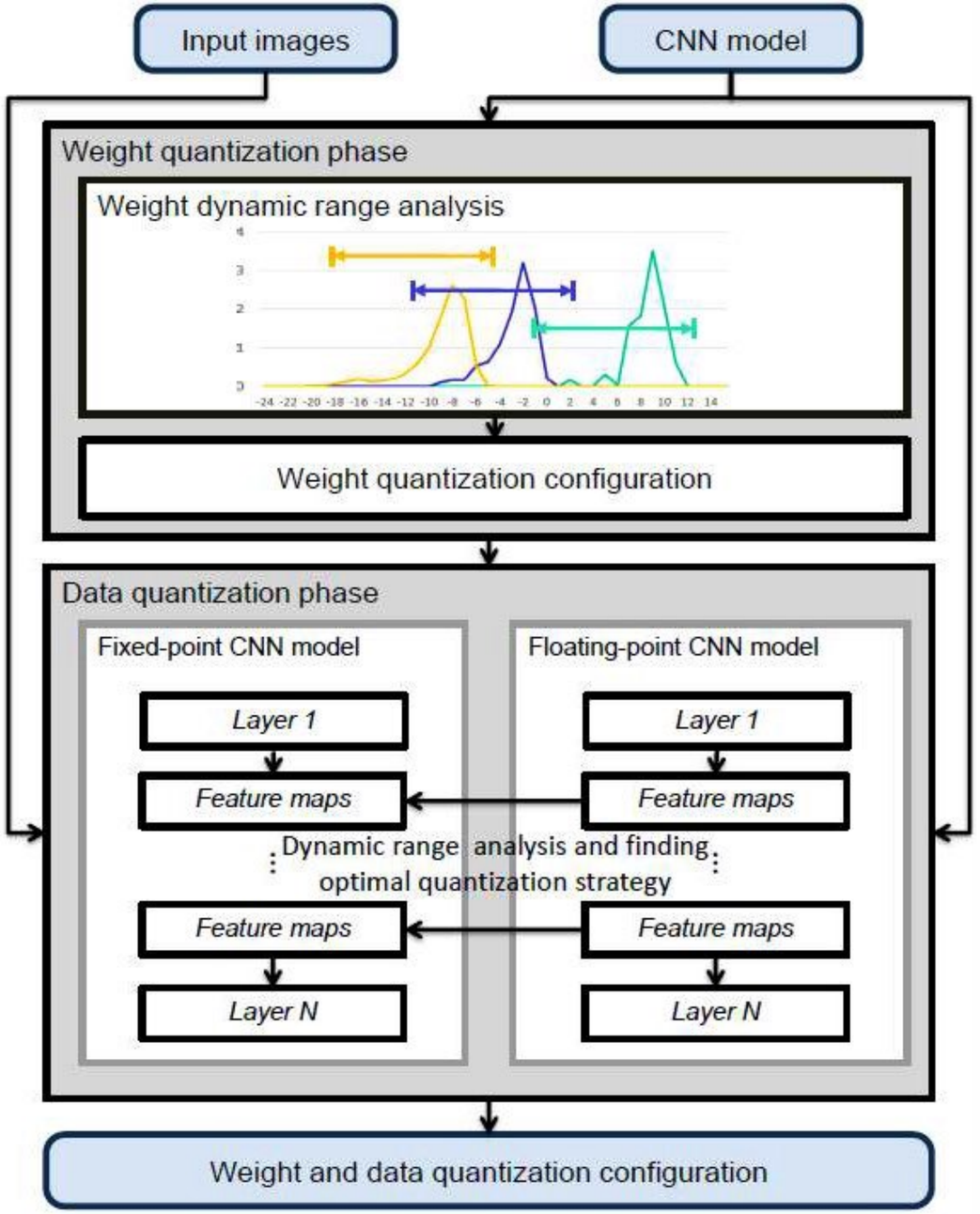
- training con float
- quantizzazione dei pesi e delle attivazioni:
  - si accumula statistica per le distribuzioni
  - si sceglie l'intervallo dinamico più appropriato
- fine-tuning con float
- conversione a formato fixed-point

ap_fixed<width,integer>

0101.1011101010

integer | fractional

width




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

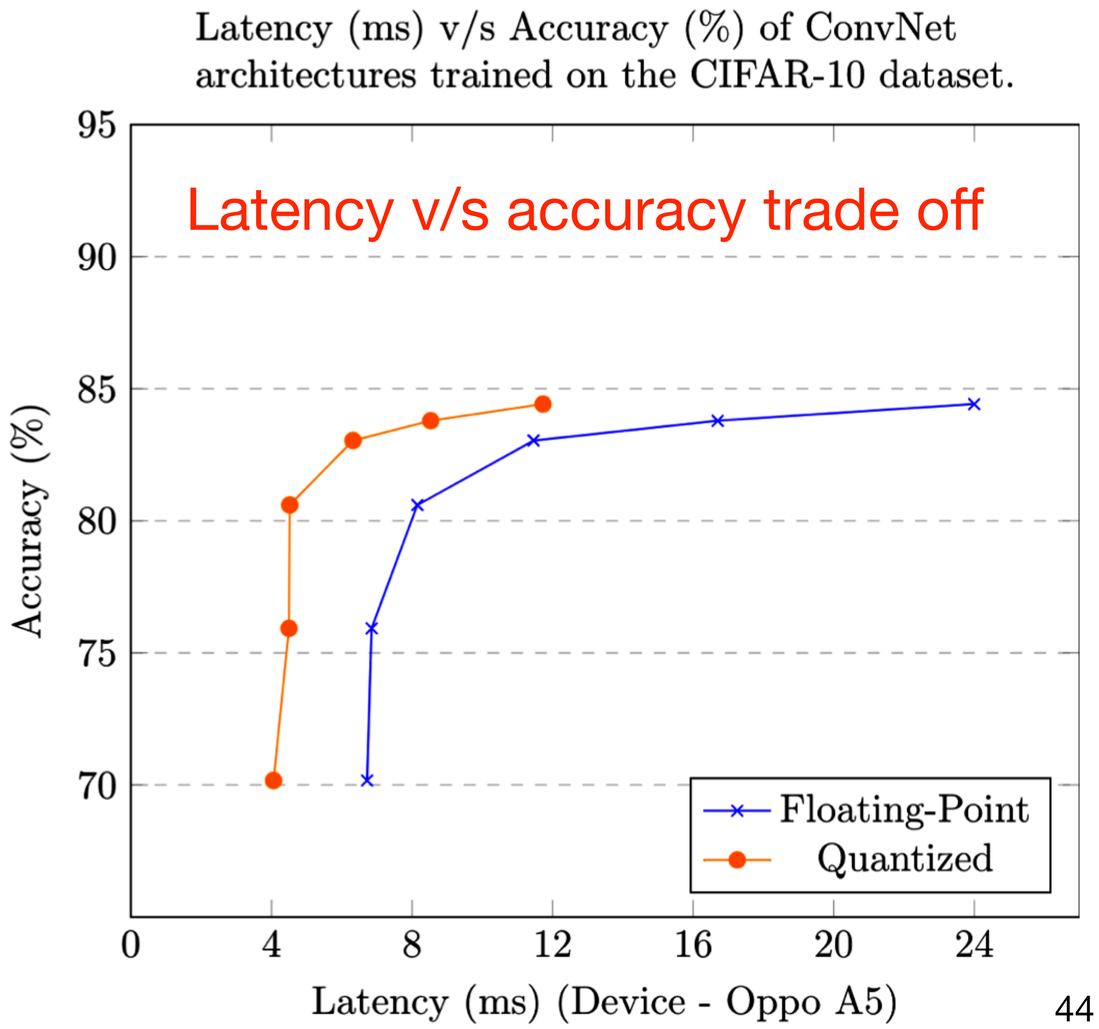
Qiu et al. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network, FPGA'16

# COMPRESSIONE VIA QUANTIZZAZIONE

- riduce la dimensione con cui i pesi e le attivazioni della rete vengono rappresentati
- tipicamente usate rappresentazioni fixed-point a 16, 8, 4 bit
- vantaggi:
  - dimensione del modello: riduzione di un fattore x32/n
  - latenza: l'operazione più costosa (a parte l'operazione di lettura/scrittura in memoria) nel calcolo di un layer di un NN è la moltiplicazione tra la matrice dei pesi e la matrice di feature

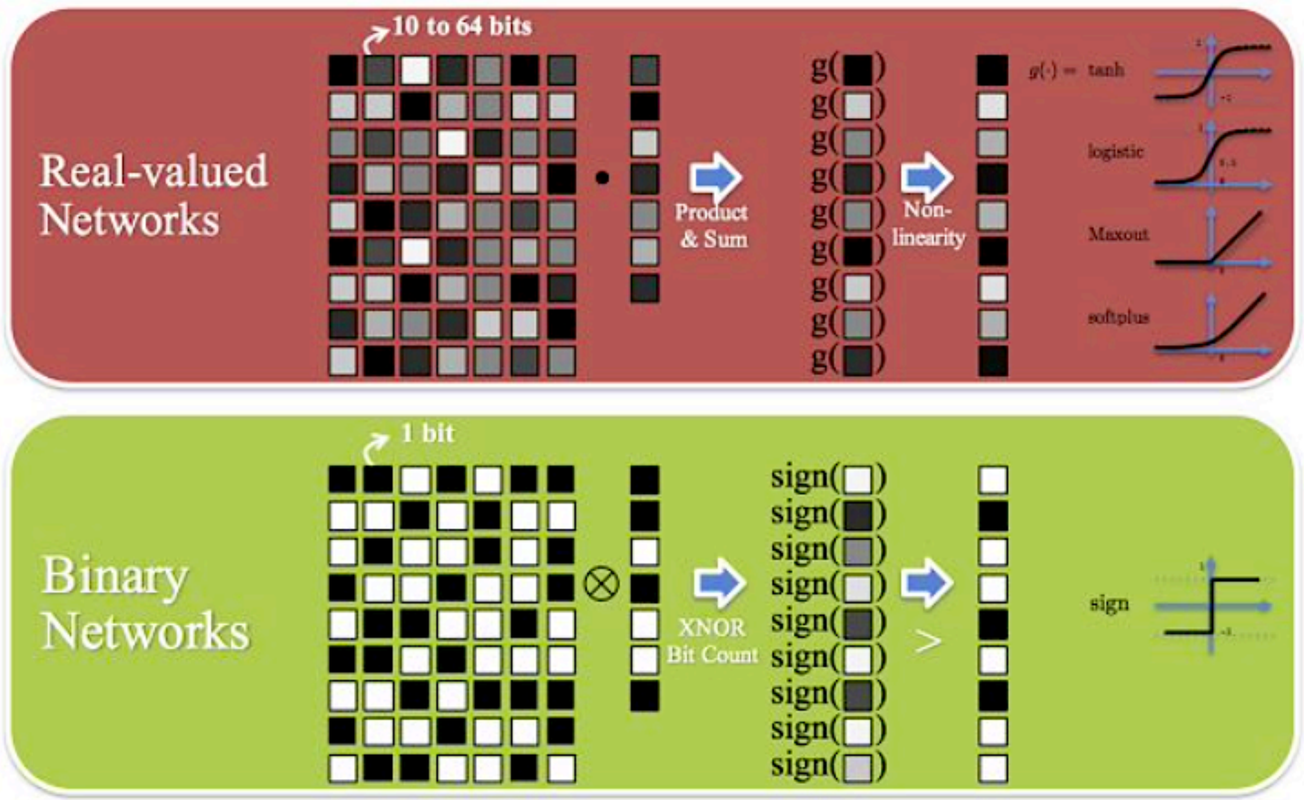
$$\phi(Wx + b)$$

il calcolo in fixed-point è più veloce che in floating point



# CASO ESTREMO: BINARY/TERNARY NETWORKS

- estremizzazione della quantizzazione:
  - rimpiazza pesi/attivazioni floating/fixed-point con aritmetica a 1 o 2 bit
    - binary net: 1 bit [0,1] (arxiv.:602.02830)
    - ternary net: 2 bit [-1,0,1] (arxiv:1605.04711)




- porta a una perdita delle prestazioni ma la compressione e il guadagno in latenza possono essere notevoli
- moltiplicazioni diventano operazioni di bit-flip
  - binary:

    *res = w == 0 ? -d : d;*
  - ternary:

    *res = w == 0 ? 0 : w == -1 ? -d : d;*