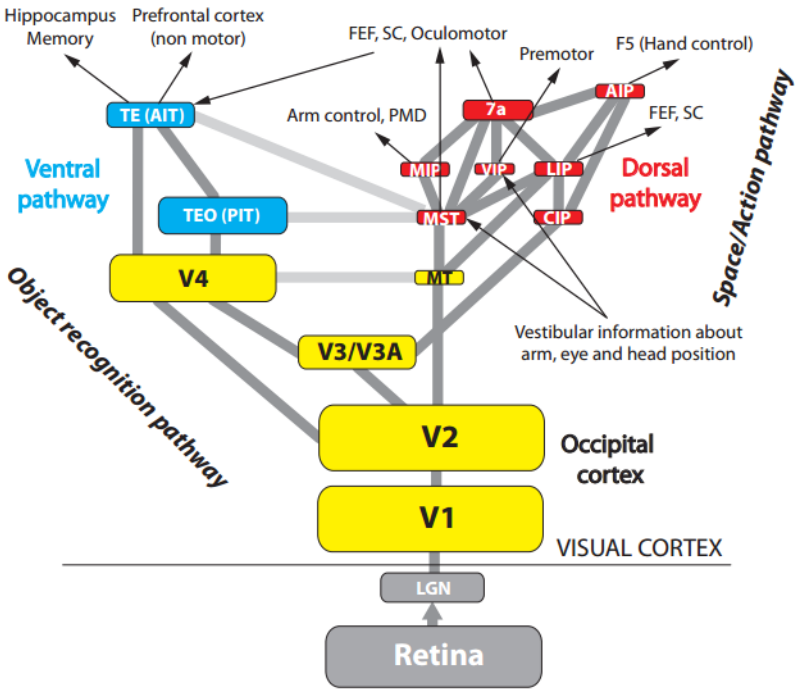
Deep Learning

- Introduzione
  - Perché deep?
  - Livelli e complessità
  - Tipologie di DNN
  - Ingredienti necessari
  - Relu, Dropout
  - ML Frameworks e Hardware
- Convolutional Neural Networks (CNN)
  - Da MLP a CNN
  - Architettura
  - Volumi e Convoluzione 3D
  - Esempi di reti
  - Training e Transfer Learning
- Recurrent Neural Networks (RNN)
- Natural Language Processing e LLM
- Reinforcement Learning

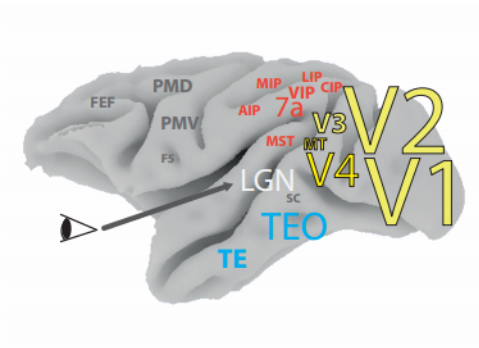
Perchè deep?

Con il termine DNN (Deep Neural Network) si denotano reti «profonde» composte da molti livelli (almeno 2 hidden) organizzati gerarchicamente.

- Le implicazioni di universal approximation theorem e la difficoltà di addestrare reti con molti livelli hanno portato per lungo tempo a focalizzarsi su reti con un solo livello hidden.
  - L'esistenza di soluzioni non implica efficienza: esistono funzioni computabili con complessità polinomiale operando su k livelli, che richiedono una complessità esponenziale se si opera su $k-1$ livelli (Hastad, 1986).
- L'organizzazione gerarchica consente di condividere e riusare informazioni (un po' come la programmazione strutturata). Lungo la gerarchia è possibile selezionare feature specifiche e scartare dettagli inutili (al fine di massimizzare l'invarianza).
- Il nostro sistema visivo opera su una gerarchia di livelli (deep):

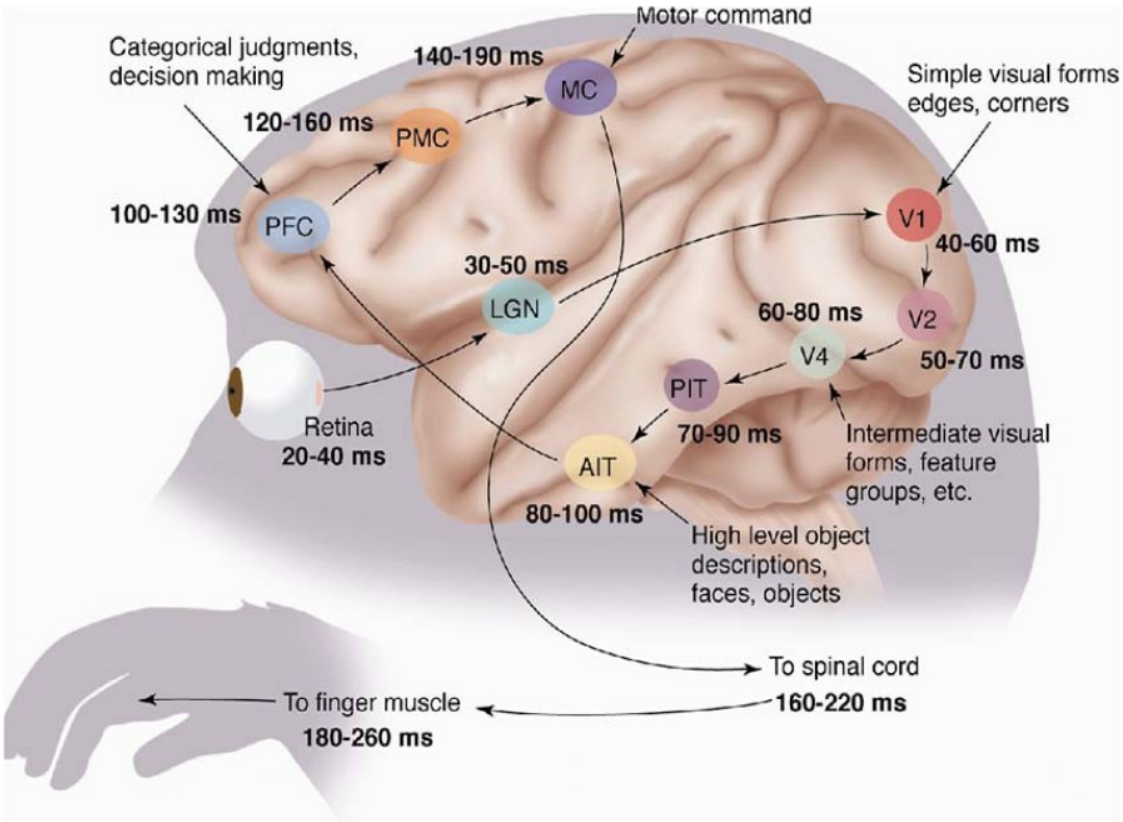


[Kruger et al. 2013]



ma quanto deep?

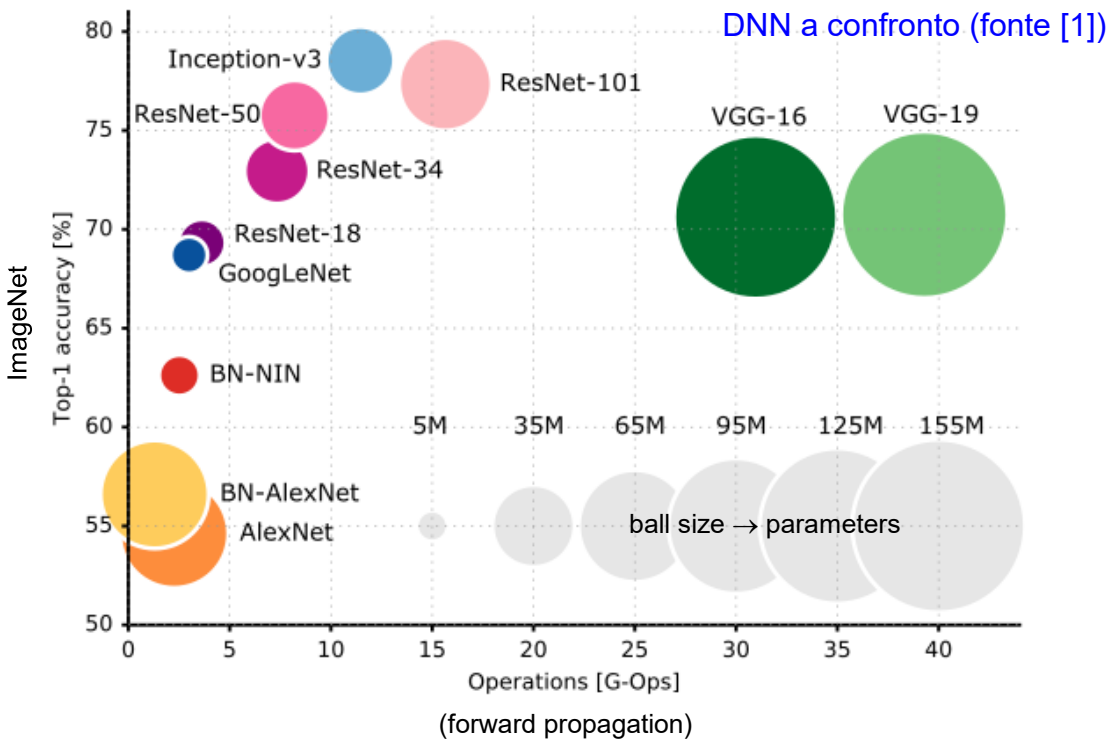
- Le DNN oggi maggiormente utilizzate consistono di un numero di livelli compreso tra 7 e 100.
  - Reti più profonde hanno dimostrato di poter garantire prestazioni leggermente migliori, a discapito però dell'efficienza.
  - «solo» una decina di livelli tra la retina e i muscoli attuatori (altrimenti saremmo troppo lenti a reagire agli stimoli).



[Simon Thorpe 1996]

Livelli e Complessità

- La profondità (numero di livelli) è solo uno dei fattori di complessità. Numero di neuroni, di connessioni e di pesi caratterizzano altresì la complessità di una DNN.
- Maggiore è il numero di pesi (ovvero di parametri da apprendere) maggiore è la complessità del training. Al tempo stesso un elevato numero di neuroni (e connessioni) rende forward e back propagation più costosi, poiché aumenta il numero (G-Ops) di operazioni necessarie.
  - AlexNet: 8 livelli, 650K neuroni e 60M parametri
  - VGG-16: 16 livelli, 15M neuroni e 140M parametri
  - GPT-3: 96 livelli, 175B parametri
  - Corteccia umana: 10^{11} neuroni e 10^{14} sinapsi (GPT-4?)



[1] Canziani et al. 2016, *An Analysis of Deep Neural Network Models for Practical Applications*

Principali tipologie di DNN

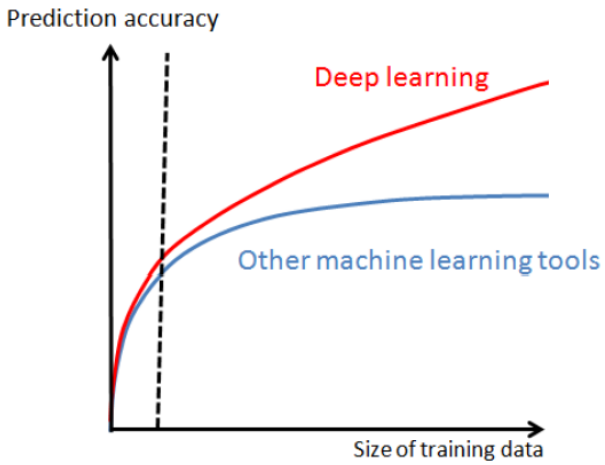
- Modelli feedforward «discriminativi» per la classificazione (o regressione) con training prevalentemente supervisionato:
  - FC DNN - Fully Connected DNN (MLP con almeno due livelli hidden)
  - CNN - Convolutional Neural Network (o ConvNet)
- Modelli ricorrenti con memoria e attenzione (utilizzati per sequenze, speech recognition, natural language processing,…):
  - RNN - Recurrent Neural Network
  - LSTM - Long Short-Term Memory
  - Transformers
- Addestramento non superivisionato: modelli addestrati a ricostruire l'input e utilizzati per denoising, anomaly detection, ...
  - Autoencoders (stacked, denoising)
- Modelli «generativi» per generare dataset sintetici (i.e., data augmentation), style transfer, art applications.
  - GAN – Generative Adversarial Networks
  - VAE – Variational Autoencoders
- Reinforcement learning (per apprendere comportamenti):
  - Deep Q-Learning

Ingredienti necessari

CNN ottengono già nel 1998 buone prestazioni in problemi di piccole dimensioni (es. riconoscimento caratteri, riconoscimento oggetti a bassa risoluzione), ma bisogna attendere il 2012 (AlexNet) per un radicale cambio di passo. AlexNet non introduce rilevanti innovazioni rispetto alle CNN di LeCun del 1998, ma alcune condizione al contorno sono nel frattempo cambiate:

- BigData: disponibilità di dataset etichettati di grandi dimensioni (es. ImageNet: milioni di immagini, decine di migliaia di classi).

La superiorità delle tecniche di deep learning rispetto ad altri approcci si manifesta quando sono disponibili grandi quantità di dati di training.



- GPU computing: il training di modelli complessi (profondi e con molti pesi e connessioni) richiede elevate potenze computazionali. La disponibilità di GPU con migliaia di core e GB di memoria interna ha consentito di ridurre drasticamente i tempi di training: da mesi a giorni.

- Vanishing (or exploding) gradient: la retro propagazione del gradiente (fondamentale per backpropagation) è problematica su reti profonde se si utilizza la sigmoide come funzione di attivazione. Il problema può essere gestito con attivazione Relu (descritta in seguito) e migliore inizializzazione dei pesi (esempio: Xavier, Hu inizialization).

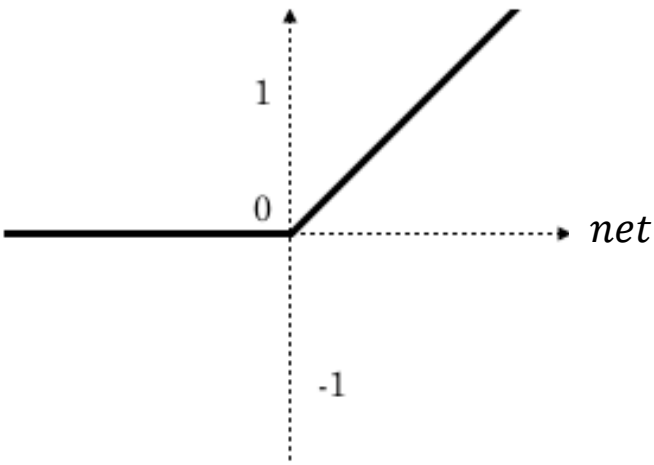
Funzione di attivazione: Relu

- Nelle reti MLP la funzione di attivazione (storicamente) più utilizzata è la sigmoide. Nelle reti profonde, l'utilizzo della sigmoide è problematico per la retro propagazione del gradiente (problema del vanishing gradient):
  - La derivata della sigmoide è tipicamente minore di 1 e l'applicazione della regola di derivazione a catena porta a moltiplicare molti termini minori di 1 con la conseguenza di ridurre parecchio i valori del gradiente nei livelli lontani dall'output. Per approfondimenti ed esempi:

    http://neuralnetworksanddeeplearning.com/chap5.html
  - La sigmoide ha un comportamento saturante (allontanandosi dallo 0). Nelle regioni di saturazione la derivata vale 0 e pertanto il gradiente si annulla.

- L'utilizzo di Relu (Rectified Linear) come funzione di attivazione risolve il problema:

$$f(net) = max(0, net)$$



La derivata vale 0 per valori negativi o nulli di net e 1 per valori positivi

Per valori positivi nessuna saturazione

Porta ad attivazioni sparse (parte dei neuroni sono spenti) conferendo maggiore robustezza alla rete

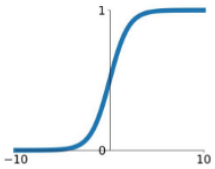
Relu: varianti

- Sebbene Relu sia la spesso funzione di attivazione di default, alcune sue varianti (leggermente più complesse) possono essere più efficaci (vedi link).
  - Leaky Relu, evita che nella parte negativa il gradiente sia 0, e quindi permette a neuroni «spenti» durante il training di potersi «risvegliare».
  - ELU (Exponential Linear Unit), introdotta nel 2015, oltre a non avere gradiente nullo nella parte negativa, è più smoothed vicino allo 0 (no discontinuità per $\alpha = 1$).

Activation Functions

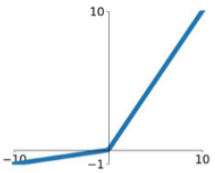
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



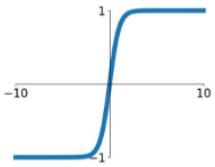
Leaky ReLU

$\max(0.1x, x)$



tanh

$\tanh(x)$

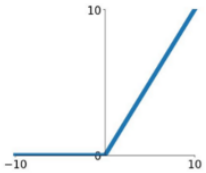


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

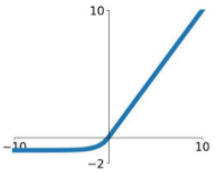
ReLU

$\max(0, x)$



ELU

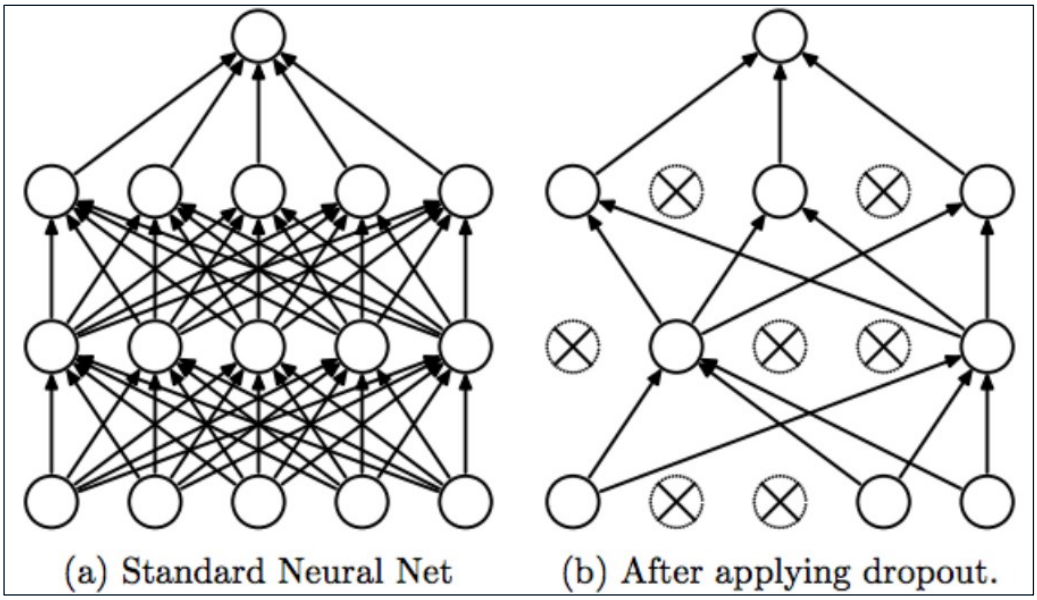
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Dropout

- È una tecnica di regolarizzazione utile per controllare overfitting:
  - specialmente nel caso di modelli molto grandi addestrati su dataset relativamente piccoli
  - rallenta la convergenza, ma può migliorare la generalizzazione.
  - usato per addestrare AlexNet su ImageNet.

- Durante il training alcuni neuroni sono «spenti» (con una data probabilità, es. 50%), per far sì che l'output non dipenda da specifici neuroni, ma sia determinato in modo più robusto. In inference tutti i neuroni sono attivi.



- In una CNN è solitamente applicato solo ai livelli fully connected nella parte finale della rete (ma non all'output).

ML frameworks

- L'implementazione di CNN (da zero) è certamente possibile. Il passo forward non è nemmeno complesso da codificare. D'altro canto la progettazione/sviluppo di software che consente:
  - il training/inference di architetture diverse a partire da una loro descrizione di alto livello (non embedded nel codice)
  - di effettuare il training con backpropagation del gradiente (rendendo disponibili le numerose varianti, parametrizzazioni e tricks disponibili)
  - ottimizzare la computazione su GPU (anche più di una)

  richiede molto tempo/risorse per lo sviluppo/debug.

- Fortunatamente sono disponibili numerosi framework (spesso open-source) che consentono di operare su DNN. Tra i più utilizzati:
  - TensorFlow (Google) – *lo useremo in laboratorio*
  - PyTorch (Facebook)
  - Caffe (Berkley) – oramai abbandonato

Seminario (Lorenzo Pellegrini)

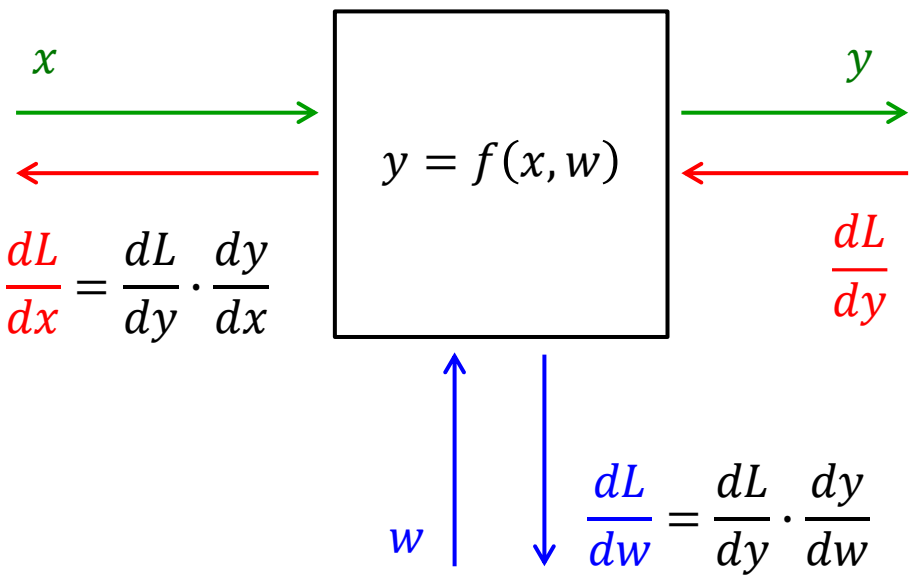
Open-Source Frameworks for Deep Learning: an Overview

Link (slide e codice) sul sito del corso.

Autodiff

- In passato ogni volta che veniva proposto un nuovo modello, era necessario derivare manualmente il gradiente e codificare le corrispondenti equazioni nel software.
- Oggi fortunatamente la maggior parte dei framework di deep learning rende disponibili meccanismi di composizione e propagazione automatica del gradiente attraverso reverse-mode autodiff:
  - ancora una volta si sfrutta la regola di derivazione a catena che permette di isolare il contributo di ogni layer (operatore) per poi ricomporre automaticamente i pezzi.
  - se si definisce un nuovo layer (od operatore) f non esistente per un framework è richiesto di definire solo i gradienti locali:
    - $\frac{dy}{dx}$ dell'output y rispetto all'input x
    - $\frac{dy}{dw}$ dell'output y rispetto ai parametri locali w

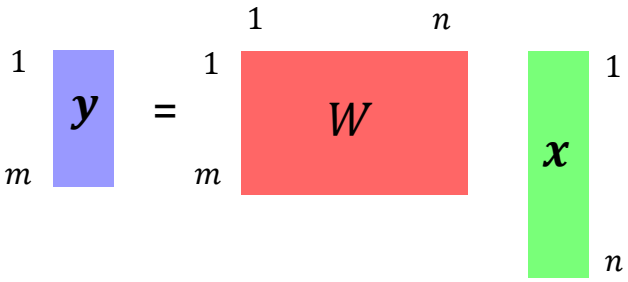
    La propagazione e combinazione con il resto dei gradienti è eseguita automaticamente.



Autodiff (2)

- In pratica input, output e pesi di un livello sono organizzati in tensori (multidimensionali) e il calcolo dei gradienti richiede derivate parziali (organizzate in matrici Jacobiane).
  - La chain rule può essere applicata moltiplicando tra loro matrici Jacobiane (con le necessarie trasposizioni): vedi link.
  - Le GPU risultano particolarmente adatte ad eseguire gemm (general matrix-matrix) multiplications.

- Esempio: livello fully connected (senza bias): $\boldsymbol{y} = W\boldsymbol{x}$



- $\boldsymbol{x} \in \mathbb{R}^n$ (n neuroni di input)
- $\boldsymbol{y} \in \mathbb{R}^m$ (m neuroni di output)
- $W \in \mathbb{R}^{m \times n}$ (matrice dei pesi locali)

$$\frac{d\boldsymbol{y}}{d\boldsymbol{x}} = \left[\frac{dy_i}{dx_j}\right] = [W_{ij}] = W$$

$$\frac{d\boldsymbol{y}}{dW_i} = \boldsymbol{x}, \qquad i = 1..m$$

riga i-esima matrice W

Hardware per il training

- Il training di modelli complessi (profondi e con molti pesi e connessioni) su dataset di grandi dimensioni (es. ImageNet) richiede elevate potenze computazionali.
- La disponibilità di GPU con migliaia di core e GB di memoria interna è di fatto necessaria per contenere i tempi di training: da mesi a giorni/ore.

Nvidia Titan RTX (2.7K€)

- 4608 Core
- 24 GB RAM
- 16.3 TFLOPS (fp32)

  CPU normalmente < 1 TFLOPS

- L'addestramento può essere parallelizzato suddividendo il carico tra diverse GPU (i principali framework lo supportano in modo trasparente):
  - Workstation (es: NVIDIA DXG Station) con 4 GPU collegate tra loro (Nvlink) per massimizzare il trasferimento dei dati senza passare dal bus PCI-express.
  - GPU Cloud (es. Amazon, Google).
  - Supercomputer (es. Leonardo del Cineca) con 3456 nodi booster, ciascuno dotato di 4 GPU Nvidia A100 (240 PFLOPS)

Inference on the Edge

- Per la fase di inference (a partire da modelli pre-addestrati, ottimizzati e congelati) è possibile usare «normali» CPU. Per prestazioni real-time o quando la CPU non è sufficientemente potente si possono utilizzare chip custom.

- Smartphone: nei modelli di fascia alta presenti NPU (Neural Processing Units), ovvero acceleratori hardware per reti neurali deep e relativi SDK.

  - Apple A15 Bionic
  - Google Neural Networks API (NNAPI)

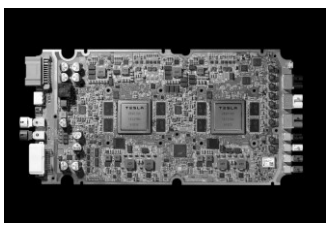
- IoT, Robot, Droni:

  - NVIDIA Jetson: schede a basso consumo con GPU.

  - Smart camera (es. Google Vision Kit): hardware economico per maker e sviluppatori con custom chip (Intel Movidius) per accelerare l'inferenza in modelli di reti neurali profonde

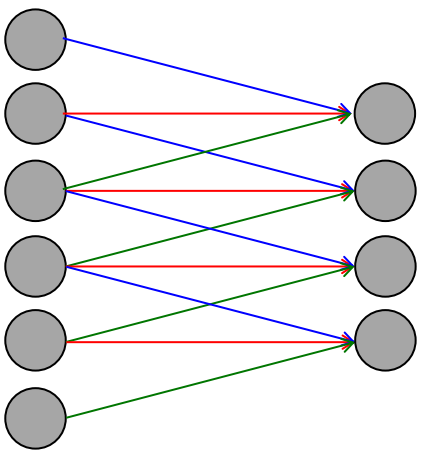
Seminario (Giacomo Bartoli 2018)
slide sul sito del corso

- Self driving chip (Tesla, 2019).



Da MLP a CNN

- Hubel & Wiesel (1962) scoprono la presenza, nella corteccia visiva del gatto, di due tipologie di neuroni:
  - Simple cells: agiscono come feature detector locali (fornendo selettività)
  - Complex cells: fondono (pooling) gli output di simple cell in un intorno (garantendo invarianza).
- Neocognitron (Fukushima, 1980) è una delle prime reti neurali che cerca di modellare questo comportamento.
- Convolutional Neural Networks (CNN) introdotte da LeCun et al., a partire dal 1998. Le principali differenze rispetto a MLP:
  - processing locale: i neuroni sono connessi solo localmente ai neuroni del livello precedente. Ogni neurone esegue quindi un'elaborazione locale. Forte riduzione numero di connessioni.
  - pesi condivisi: i pesi sono condivisi a gruppi. Neuroni diversi dello stesso livello eseguono lo stesso tipo di elaborazione su porzioni diverse dell'input. Forte riduzione numero di pesi.



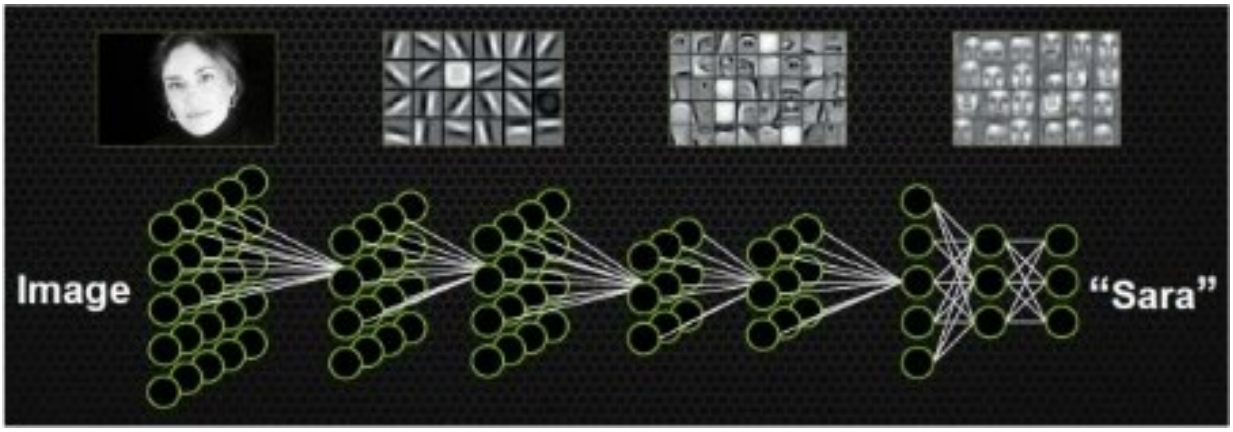
Esempio: ciascuno dei 4 neuroni a destra è connesso solo a 3 neuroni del livello precedente. I pesi sono condivisi (stesso colore stesso peso). In totale 12 connessioni e 3 pesi contro le 24 connessioni + 24 pesi di una equivalente porzione di MLP.

  - alternanza livelli di feature extraction e pooling.

CNN: Architettura

Esplicitamente progettate per processare immagini, per le quali elaborazione locale, pesi condivisi, e pooling non solo semplificano il modello, ma lo rendono più efficace rispetto a modelli fully connected. Possono essere utilizzate anche per altri tipi di pattern (es. speech).

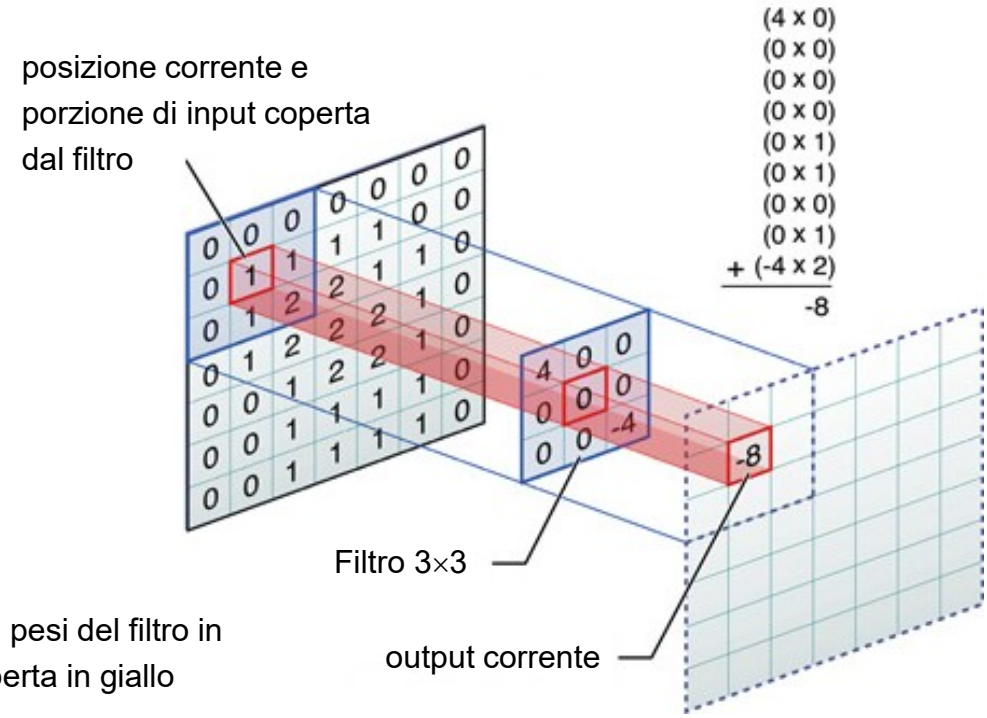
- Architettura: una CNN è composta da una gerarchia di livelli. Il livello di input è direttamente collegato ai pixel dell'immagine, gli ultimi livelli sono generalmente fully-connected e operano come un classificatore MLP, mentre nei livelli intermedi si utilizzano connessioni locali e pesi condivisi.



- Il campo visivo (receptive field) dei neuroni aumenta muovendosi verso l'alto nella gerarchia.
- Le connessioni locali e condivise fanno sì che i neuroni processino nello stesso modo porzioni diverse dell'immagine. Si tratta di un comportamento desiderato, in quando regioni diverse del campo visivo contengono lo stesso tipo di informazioni (bordi, spigoli, porzioni di oggetti, ecc.).
- Visualizzazione 3D di una CNN: *http://www.cs.cmu.edu/~aharley/vis/*

Convoluzione

- La convoluzione è una delle più importanti operazioni di image processing attraverso la quale si applicano filtri digitali.
- Un filtro digitale (una piccola maschera 2D di pesi) è fatto scorrere sulle diverse posizioni di input; per ogni posizione viene generato un valore di output, eseguendo il prodotto scalare tra la maschera e la porzione dell'input coperta (entrambi trattati come vettori).



Esempio (animato): pesi del filtro in rosso, porzione coperta in giallo

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

4		

Esempi applicazione filtri a Immagini

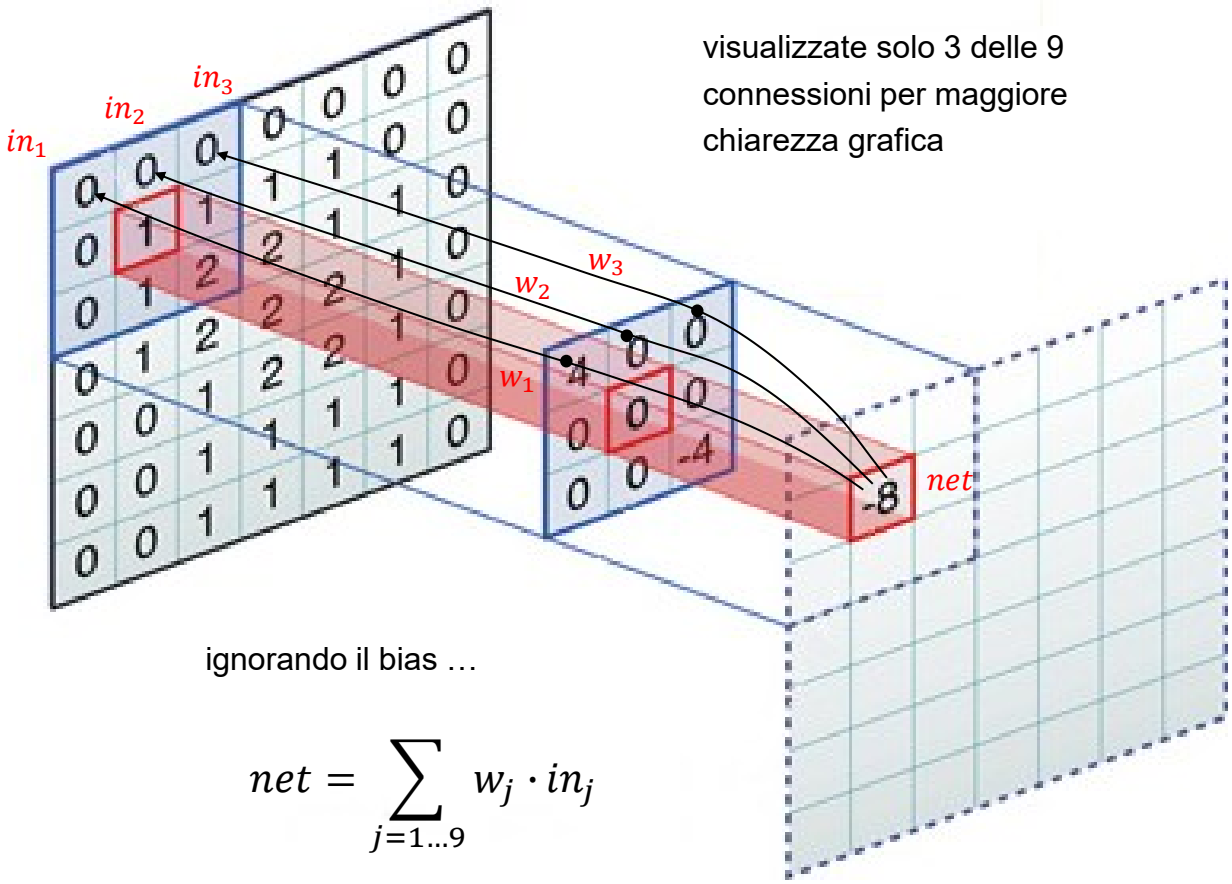
Immagine input | Filtro | Immagine output

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Neuroni come convolutori

- Consideriamo i pixel come neuroni e le due immagini di input e di output come livelli successivi di una rete. Dato un filtro 3×3, se colleghiamo un neurone ai 9 neuroni che esso «copre» nel livello precedente, e utilizziamo i pesi del filtro come pesi delle connessioni w, notiamo che un classico neurone (di una MLP) esegue di fatto una convoluzione.



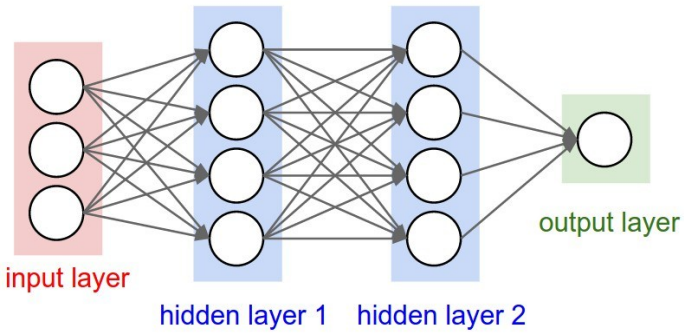
visualizzate solo 3 delle 9 connessioni per maggiore chiarezza grafica

ignorando il bias …

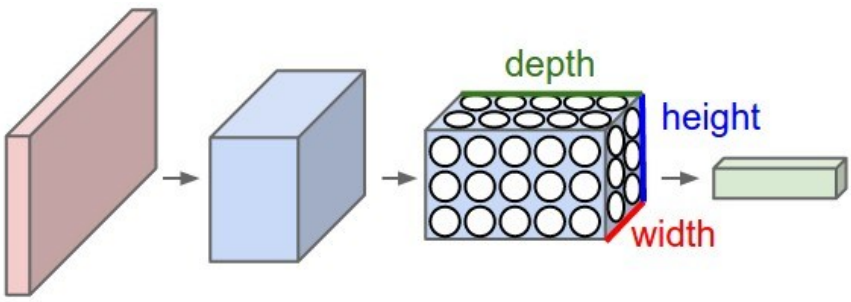
$$net = \sum_{j=1\ldots9} w_j \cdot in_j$$

CNN: Volumi

- Volumi: I neuroni di ciascun livello sono organizzati in griglie o volumi 3D (si tratta in realtà di una notazione grafica utile per la comprensione delle connessioni locali).



MLP: organizzazione lineare dei neuroni nei livelli

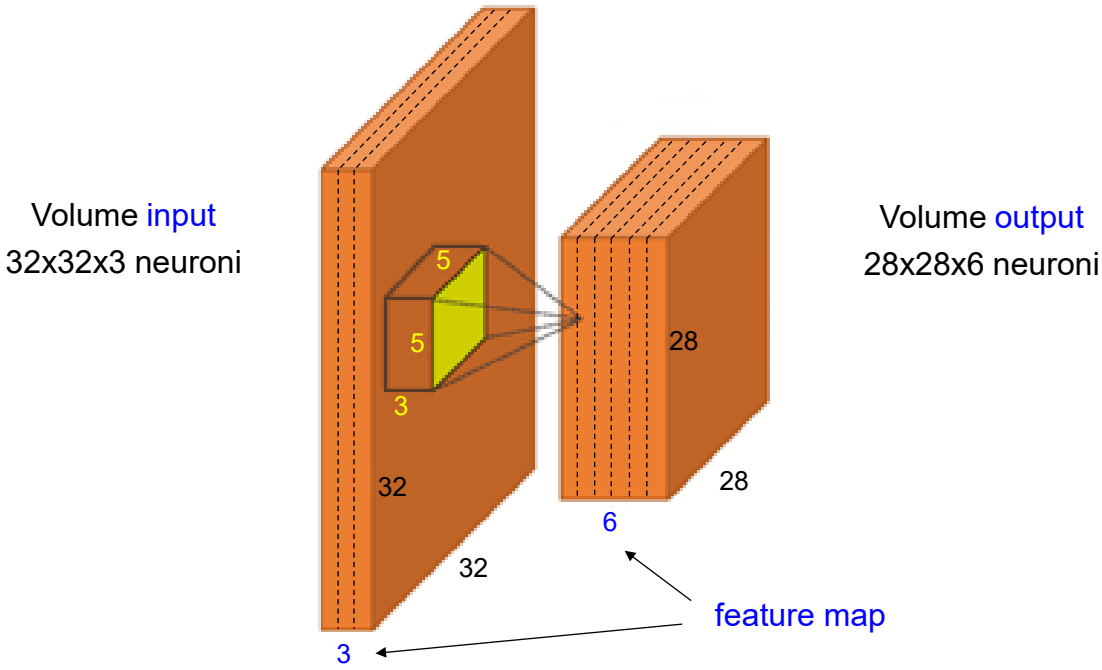


CNN: i livelli sono organizzati come griglie 3D di neuroni

- sui piani width - height si conserva l'organizzazione spaziale «retinotipica» dell'immagine di input.
- la terza dimensione depth (come vedremo) individua le diverse feature map.

CNN: Convoluzione 3D

- Il filtro opera su una porzione del volume di input. Nell'esempio ogni neurone del volume di output è connesso a 5x5x3=75 neuroni del livello precedente.



- Ciascuna «fetta» di neuroni (stessa depth) denota una feature map. Nell'esempio troviamo:
  - 3 feature map (dimensione 32x32) nel volume di input.
  - 6 feature map (dimensione 28x28) nel volume di output.

- I pesi sono condivisi a livello di feature map. I neuroni di una stessa feature map processano porzioni diverse del volume di input nello stesso modo. Ogni feature map può essere vista come il risultato di uno specifico filtraggio dell'input (filtro fisso).

- Nell'esempio il numero di connessioni tra i due livelli è (28x28x6) x (5x5x3) = 352800, ma il numero totale di pesi è 6 x (5x5x3 + 1) = 456. *In analoga porzione di MLP quanti pesi?*

bias

CNN: Convoluzione 3D (2)

- Quando un filtro 3D viene fatto scorrere sul volume di input, invece di spostarsi con passi unitari (di 1 neurone) si può utilizzare un passo (o $Stride$) maggiore. Questa operazione riduce la dimensione delle feature map nel volume di output e conseguentemente il numero di connessioni.
  - Sui livelli iniziali della rete per piccoli stride (es. 2, 4), è possibile ottenere un elevato guadagno in efficienza a discapito di una leggera penalizzazione in accuratezza.
- Ulteriore possibilità (per regolare la dimensione delle feature map) è quella di aggiungere un bordo (valori zero) al volume di input. Con il parametro $Padding$ si denota lo spessore (in pixel) del bordo.
- Sia W_{out} la dimensione (orizzontale) della feature map di output e W_{in} la corrispondente dimensione nell'input. Sia inoltre F la dimensione (orizzontale del filtro). Vale la seguente relazione:

$$W_{out} = \frac{(W_{in} - F + 2 \cdot Padding)}{Stride} + 1$$

Nell'esempio precedente: $Stride = 1$, $Padding = 0$, $W_{in} = 32$, $F = 5 \rightarrow W_{out} = 28$. Analoga relazione lega i parametri verticali.

- Il corso on-line di Andrej Karpathy (Stanford - OpenAI) introduce questi concetti in modo molto chiaro ed esaustivo.

  http://cs231n.github.io/convolutional-networks/

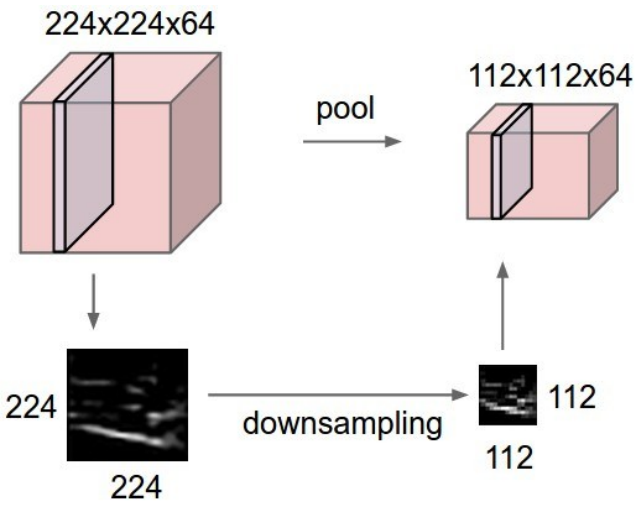
  In particolare vedi:

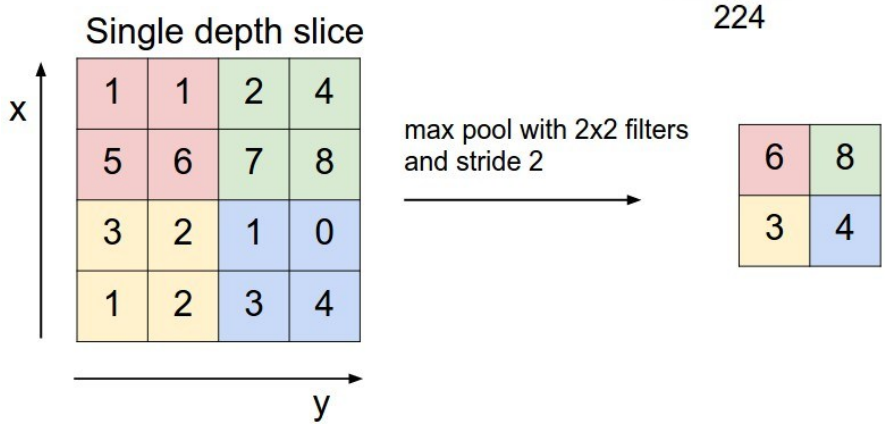
  - Esempio (animato) su convoluzione 3D
  - Esempi di volumi considerando Stride e Padding.

CNN: Pooling

- Un livello di pooling esegue un'aggregazione delle informazioni nel volume di input, generando feature map di dimensione inferiore. Obiettivo è conferire invarianza rispetto a semplici trasformazioni dell'input mantenendo al tempo stesso le informazioni significative ai fini della discriminazione dei pattern.
- L'aggregazione opera (generalmente) nell'ambito di ciascuna feature map, cosicché il numero di feature map nel volume di input e di output è lo stesso. Gli operatori di aggregazione più utilizzati sono la media (Avg) e il massimo (Max): entrambi «piuttosto» invarianti per piccole traslazioni. Questo tipo di aggregazione non ha parametri/pesi da apprendere.
- Nell'esempio un max-pooling con Filtri 2×2 e $Stride = 2$.

L'equazione precedente per il calcolo di W_{out} è valida anche per pooling (considerando $Padding$ = 0)





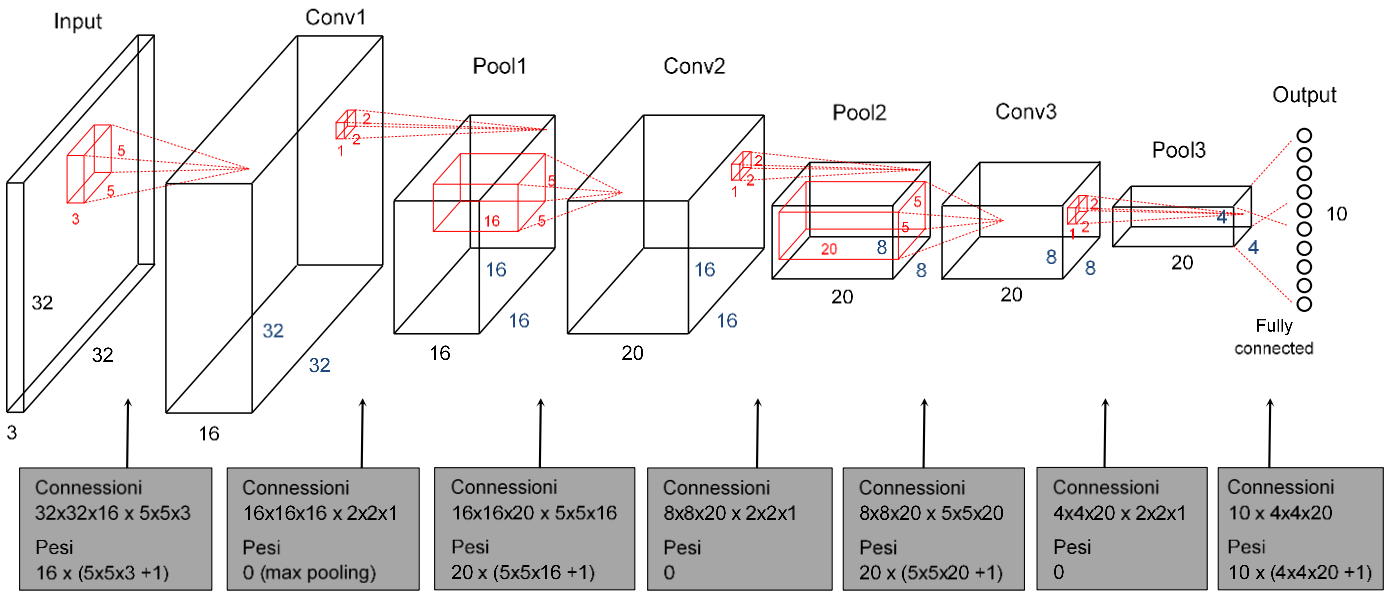
Ricomponiamo i pezzi

- Esempio 1: Cifar-10 (Javascript running in the browser).

  http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

  Architettura
  - Input: Immagini RGB 32x32x3;
  - Conv1: Filtri:5x5, FeatureMaps:16, stride:1, pad:2, attivazione: Relu
  - Pool1: Tipo: Max, Filtri 2x2, stride:2
  - Conv2: Filtri:5x5, FeatureMaps:20, stride:1, pad:2, attivazione: Relu
  - Pool2: Tipo: Max, Filtri 2x2, stride:2
  - Conv3: Filtri:5x5, FeatureMaps:20, stride:1, pad:2, attivazione: Relu
  - Pool3: Tipo: Max, Filtri 2x2, stride:2
  - Output: Softmax, NumClassi: 10

  Disegniamo la rete e calcoliamo neuroni sui livelli, connessioni e pesi



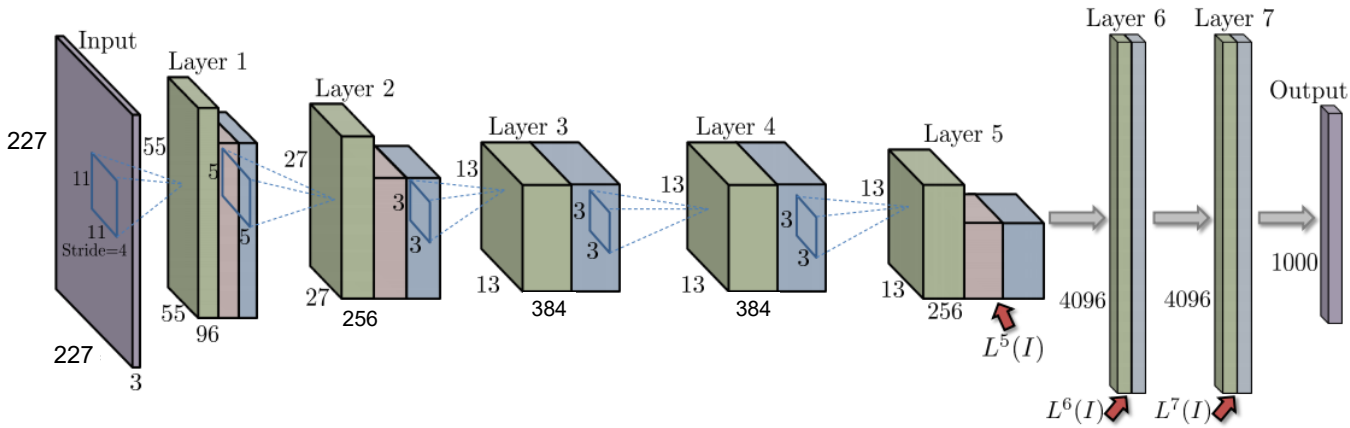
Neuroni totali: 31.562 (incluso livello input)

Connessioni totali: 3.942.784

Pesi totali: 22.466 (inclusi bias)

Ricomponiamo i pezzi (2)

- Esempio 2: CaffeNet (AlexNet porting in Caffe).



Codici colore

- Viola: Input (immagini 227x227x3) e Output (1000 classi di ImageNet)
- Verde: Convoluzione
- Rosa: Pooling (max)
- Blu: Relu activation

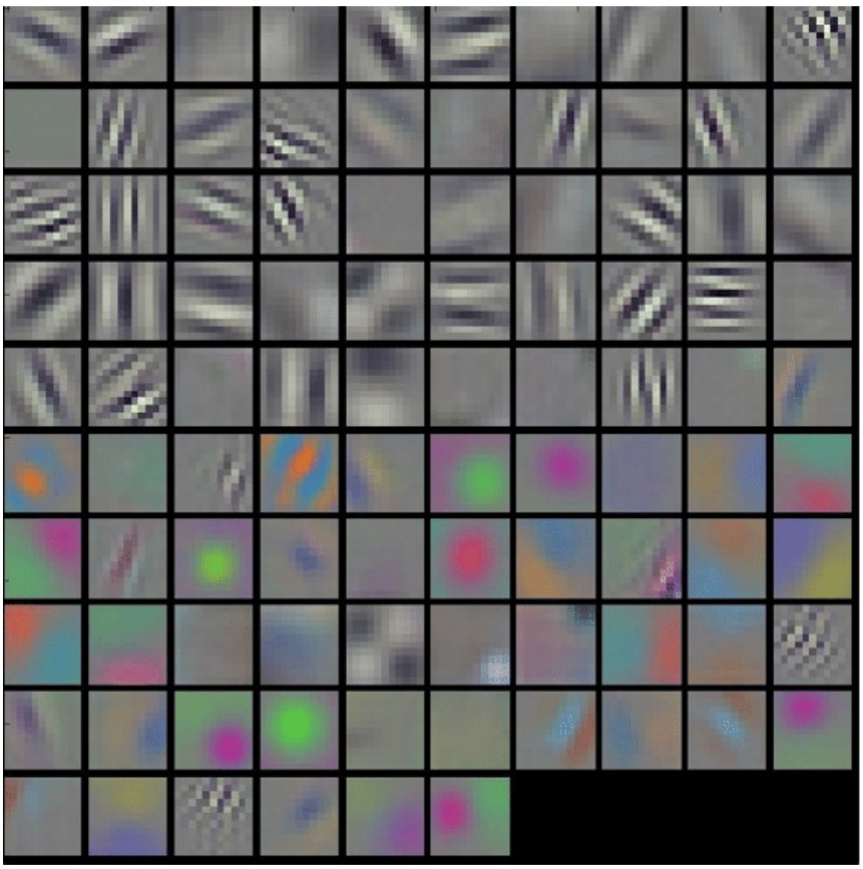
Note

- Layer 6, 7 e 8: Fully connected
- Layer 8: Softmax (1000 classi)
- Stride: 4 per il primo livello di convoluzione, poi sempre 1
- Filtri: Dimensioni a scalare: da 11x11 a 3x3
- Feature Map: Numero crescente muovendosi verso l'output
- L^5, L^6, L^7, denotano feature riutilizzabili per altri problemi (vedi transfer-learning e [1]).
- Numero totale di parametri: 60M circa

[1] Babenko et al., *Neural Codes for Image Retrieval*, 2014.

I filtri appresi da AlexNet

- La visualizzazione dei 96 filtri del 1 livello appresi da AlexNet al termine del training su ImageNet, fu piuttosto sorprendente.



- I primi 48 filtri processano le informazioni a livelli di grigio estraendo contorni a varie orientazioni e scale (in modo simile alle Simple Cells nella corteccia visiva).
- I successivi 48 processano informazioni collegate al colore e alle differenze di colore.
- La specializzazione dei due gruppi non è stata pre-programmata ma è la conseguenza di aver canalizzato i filtri in due gruppi diversi per poter meglio distribuire il carico su due GPU.

e dopo AlexNet ?

Le principali innovazioni che hanno portato a modelli deep sempre più performanti sono:

- Batch Normalization dopo ogni livello si normalizzano gli output affinché gli input al livello successivo non si spostino troppo nello spazio (covariate shift).
  - La normalizzazione avviene a livello di minibatch (e di singola feature map nelle CNN).
  - Training più stabile e veloce, con learning rate maggiore.
- Skip Connections (introdotte in ResNet). Si aggiungono dei collegamenti «laterali» ai layer che sommano l'input originario all'output del layer.
  - In questo modo il mapping da apprendere è in termini di scostamento (o residuale) rispetto all'identità (più semplice).
  - Reti più profonde, ridotto problema vanishing gradient.
- Depthwise (1x1) convolutions (introdotte nei moduli Inception di GoogleNet e rese popolari da MobileNet). Riduzione rilevante della complessità senza grossa perdita di prestazioni.

  Esempio: M convoluzioni 3D con kernel 3x3xC (costo $\propto$ Mx3x3xC) si suddividono in due step:
  - Per ciascuna delle C feature map di input si fa una convoluzione 2D (3x3) indipendente (costo $\propto$ Cx3x3).
  - A partire dai risultati step 1, per ciscuna delle M feature map di output si fa convoluzione con kernel 1x1xC (costo $\propto$ MxC).

Training e Transfer Learning

- Il training di CNN complesse (es. AlexNet) su dataset di grandi dimensioni (es. ImageNet) può richiedere giorni/settimane di tempo macchina anche se eseguito su GPU.
  - Fortunatamente, una volta che la rete è stata addestrata, il tempo richiesto per la classificazione di un nuovo pattern (propagazione forward) è in genere veloce (es. 10-100 ms).
- Inoltre il training di una CNN su un nuovo problema, richiede un training set etichettato di notevoli dimensioni (spesso non disponibile). In alternativa al training da zero, possiamo perseguire due strade (Transfer Learning):
  - Fine-Tuning: si parte con una rete pre-trained addestrata su un problema simile e:
    1. si rimpiazza il livello di output con un nuovo livello di output softmax (adeguando il numero di classi).
    2. come valori iniziali dei pesi si utilizzano quelli della rete pre-trained, tranne che per le connessioni tra il penultimo e ultimo livello i cui pesi sono inizializzati random.
    3. si eseguono nuove iterazioni di addestramento (SGD) per ottimizzare i pesi rispetto alle peculiarità del nuovo dataset (non è necessario che sia di grandi dimensioni).
  - Riutilizzo Features: si utilizza una rete esistente (pre-trained) senza ulteriore fine-tuning. Si estraggono (a livelli intermedi) le feature generate dalla rete durante il passo forward (vedi L^5, L^6, L^7 nell'esempio CaffeNet). Si utilizzano queste feature per addestrare un classificatore esterno (es. SVM) a classificare i pattern del nuovo dominio applicativo.

Per approfondimenti: http://cs231n.github.io/transfer-learning/