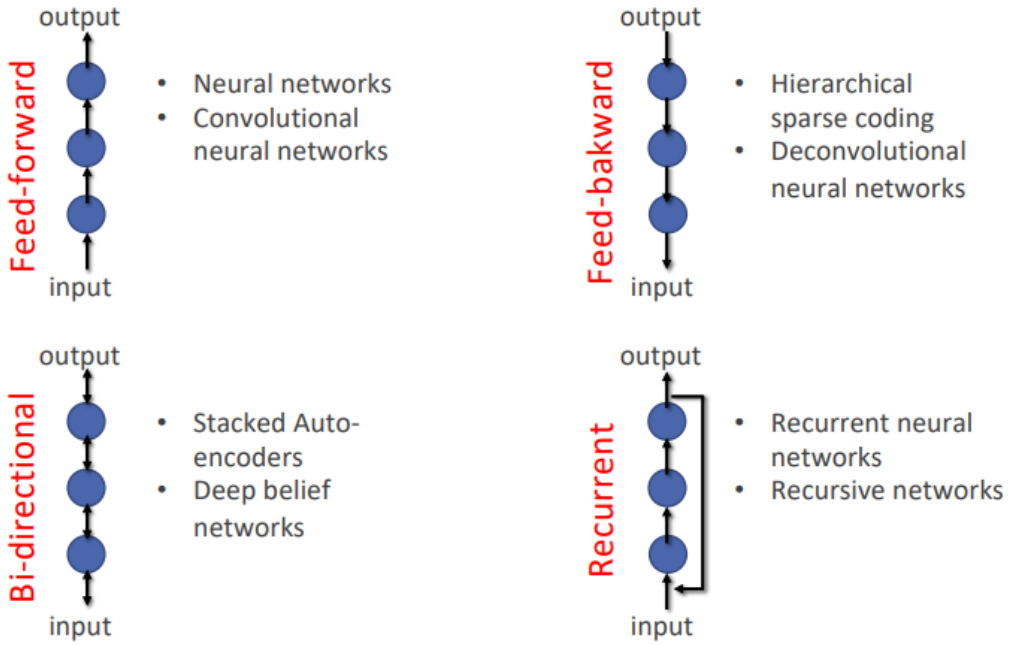
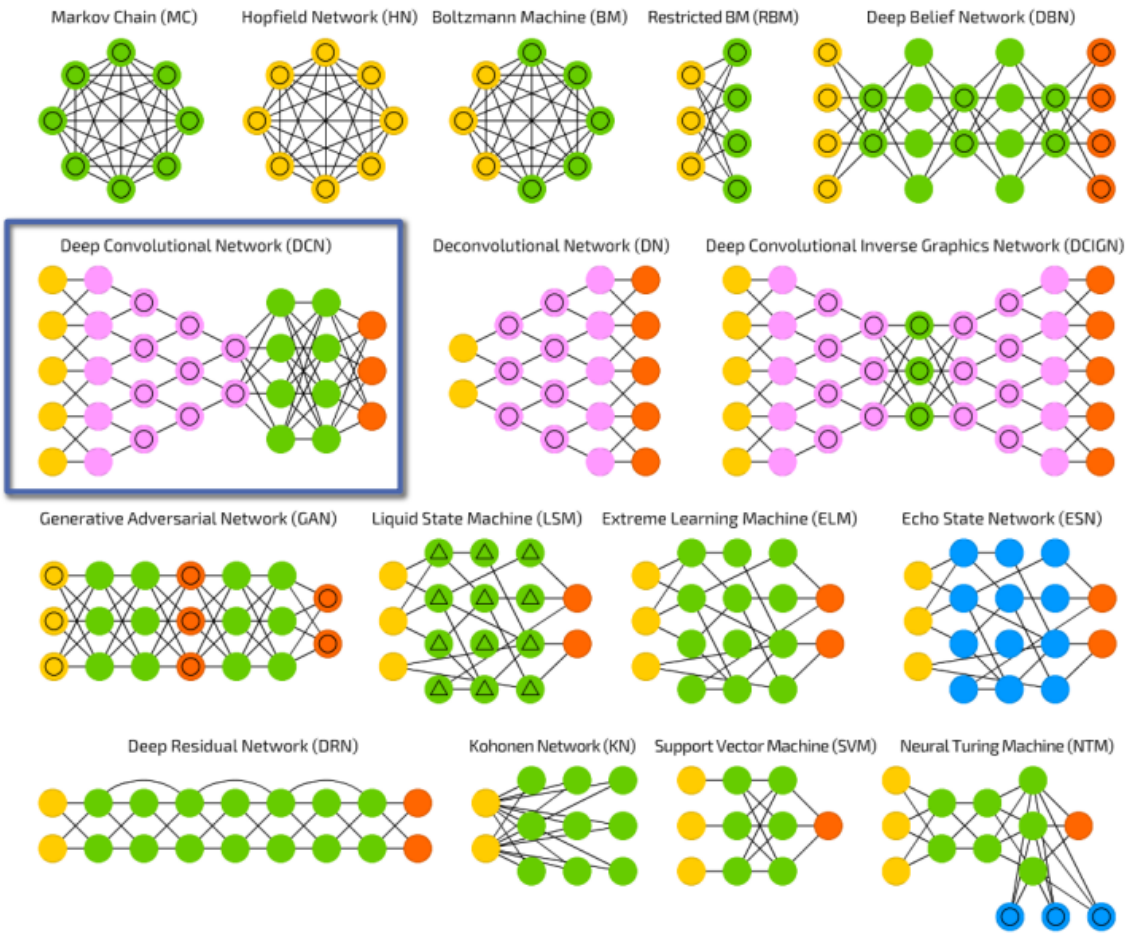
Lezione 7 - CNN - 15/11/2024

Diversi tipi di NN:



Il tipo **recurrent** è fatto per dati sequenziali (la vedremo verso fine corso). Parte dell'informazione processata è ridata in input al prossimo elemento della sequenza. Il nuovo input potrebbe essere per esempio il prossimo sample di un segnale audio.

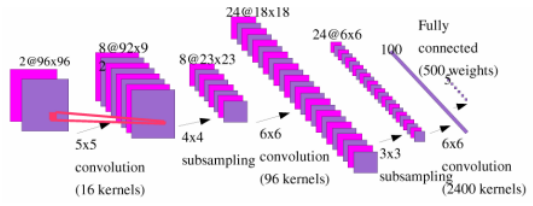
In realtà ci sono molti altri tipi:



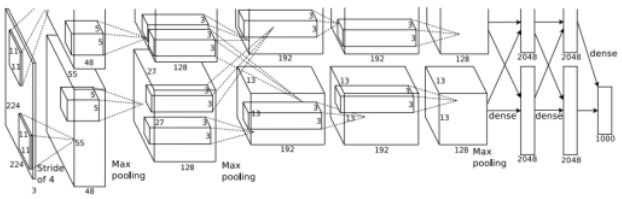
Oggi vedremo le Deep Convolutional Neural Networks. Qui abbiamo parti separate (diversi colori nell'immagine), oltre all'input e all'output, ci sono layers di tipo diverso.

Convolutional Neural Networks

Sono quelle più usate nella computer vision. Impilano diversi stadi, dove quelli più alti (più lontani dall'input) computano features più generali.

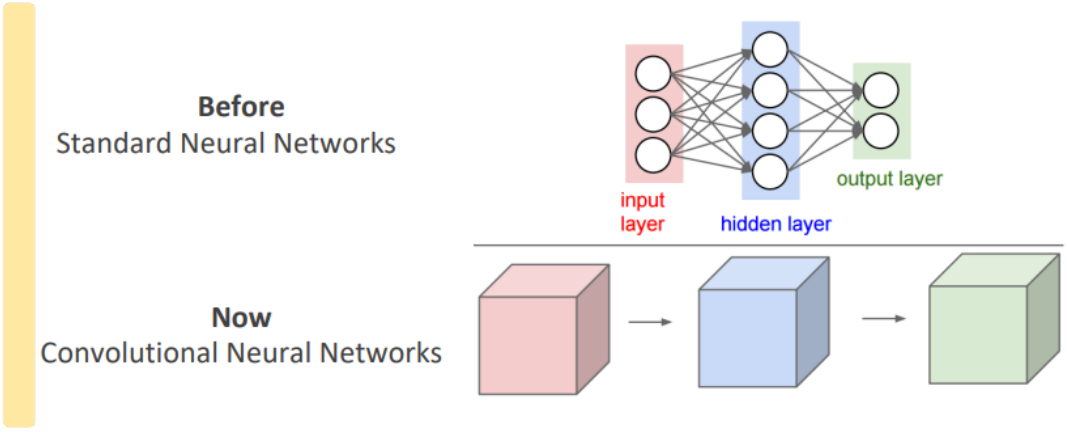


LeNet [1]



AlexNet [2]

Nelle NN tradizionali abbiamo l'input, l'output e il layer nascosto.



Ora invece i pesi sono organizzati in volumi.

Supponiamo che stiamo lavorando con una NN tradizionale, con un'immagine 32×32×3 (molto piccola).

Se usiamo un percettrone (un neurone), questo viene collegato da 3072 pesi, tutti su un singolo neurone.

Il nuovo tipo di neurone non sarà collegato all'intera immagine, ma solo ad una piccola porzione dell'immagine. Per esempio lo collegheremo ad un sub-patch 5×5×3, che avrà 75 pesi. Quindi abbiamo **connettività locale**. Questa connettività è **locale nello spazio** (5×5 nell'immagine 32×32) ma **completa nella profondità** (considera tutti i canali dell'immagine).

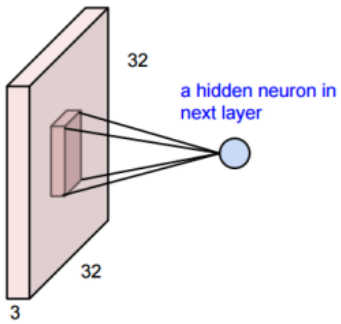


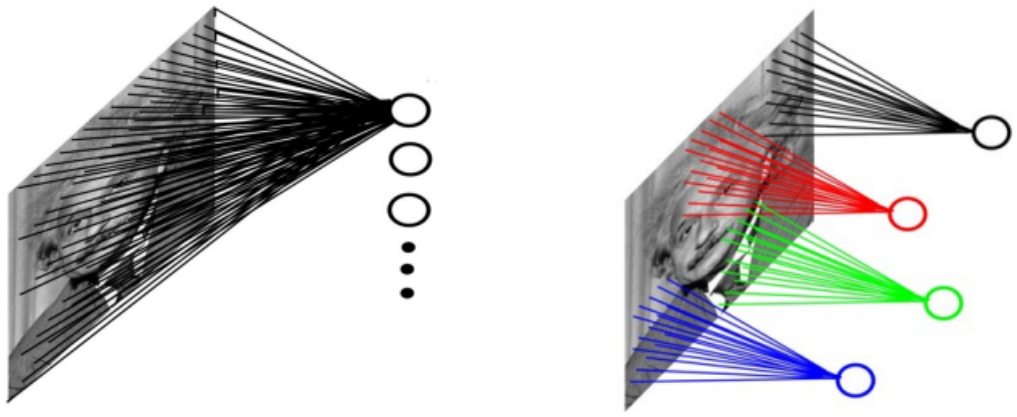
Image: 32x32x3 volume

Before: full connectivity 32x32x3 = 3072 weights

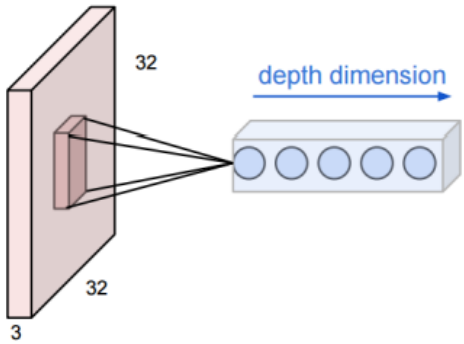
Now: one neuron will connect to, e.g. 5x5x3 patch and only have 5x5x3 = 75 weights

Note that connectivity is:
- Local in space (5x5 inside 32x32)
- Full in depth (all 3 depth channels)

Quindi ciascun neurone esplora una parte diversa dell'input.



Supponiamo ora di avere multipli neuroni che guardano alla stessa parte dell'immagine. Ciascun neurone ci darà delle informazioni diverse, ma tutti guardano alla stessa parte. Ciascun neurone è indipendente, non prendono informazioni dagli altri, solo da quella porzione dell'immagine.

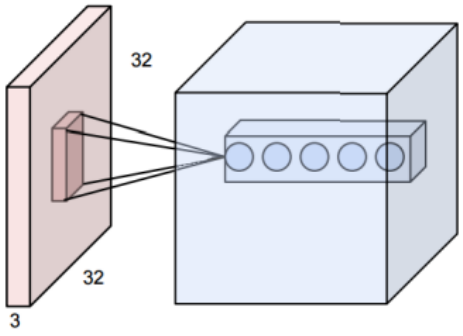


Before: hidden layer of 200 neurons

Now: output volume of depth 200

Multiple neurons all looking at the same region of the input volume, stacked along depth

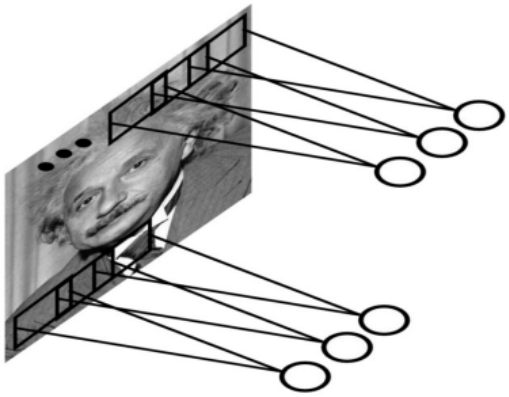
Questi neuroni quindi formano un volume, perchè ci saranno altre file di neuroni che guardano alle altre parti dell'immagine.



These form a single [1 x 1 x depth] «depth column» in the output feature map volume

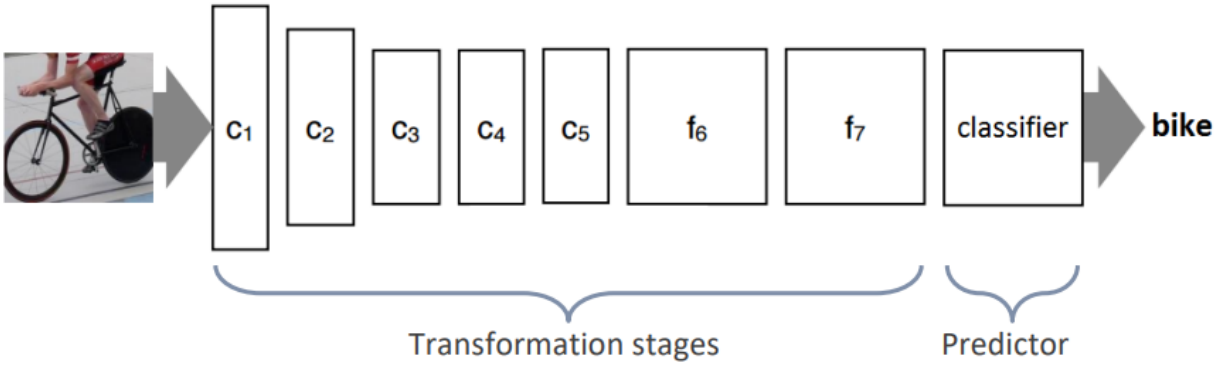
Per ridurre il numero di parametri che abbiamo bisogno, introduciamo l'idea di **weight sharing**. I pesi sono condivisi nella posizione spaziale. Quindi non avremo 5 neuroni che guardano ogni posizione, ma avremo gli stessi 5 neuroni che guardano a parti diverse dell'immagine.

- Local connectivity
- Weights sharing
  - Weights in the layer are shared across spatial positions



Questa è l'architettura di una convolutional NN: abbiamo degli stadi che contengono il tipo di strato che abbiamo appena visto.

Ad un certo punto avremo una conversione (reshape) a vettore monodimensionale, e da lì in poi avremo una rete neurale tradizionale.



Le CNNS sono:

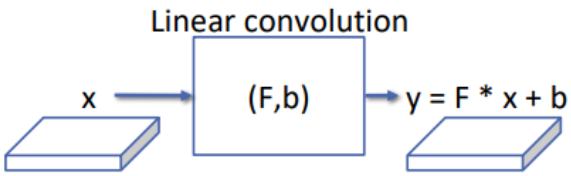
- Feed forward
- Usano supervised training con back propagation
- Hanno layers speciali:

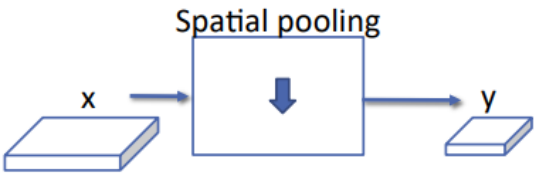
- Spatial pooling
- Local response normalization

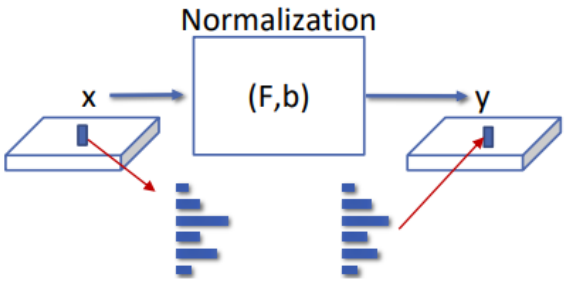
Sono utili per:

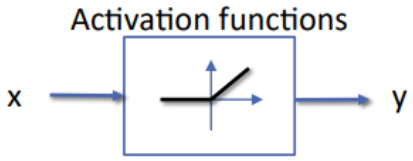
- Reduce computa9onal burden
- Increase invariance
- Ease the op9miza9on

Overview dei componenti:









Linear convolution

Partendo dall'input, abbiamo un filtro con un bias, calcoliamo la convoluzione del filtro con l'input (F*x) e poi calcoliamo il bias (+b). Questa è un'operazione lineare.

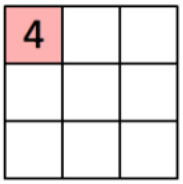
- Linear
- Local
- Translation invariant
- Filter bank to form a richer representation of the data

Se prima avevamo 10 neurone in un hidden layer, ora abbiamo un **filter bank** di 10 filtri.

- Input $x = H \times W \times K$ array
- Filter bank $\mathrm{F} = H' \times W' \times K \times Q$ array
- Output $y = (H - H' + 1) \times (W - W' + 1) \times Q$ array

$$y_{ijq} = y_q + \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} \sum_{k=1}^{K} x_{u+i,v+j,k} F_{u,v,k,q}$$

Image



Convolved Feature

Abbiamo un **input** x, e un numero di canali K.

Il **filter bank** ha una dimensione, di solito più piccola (H'<H etc), questo è full in depth (ha la stessa profondità dell'input (K), e abbiamo un numero di filtri che usiamo (Q).

L'**output** avrà un diverso numero di righe e colonne, in base alla dimensione del filtro che usiamo, avrà un numero di canali di profondità uguale al numero di filtri della filter bank (Q).

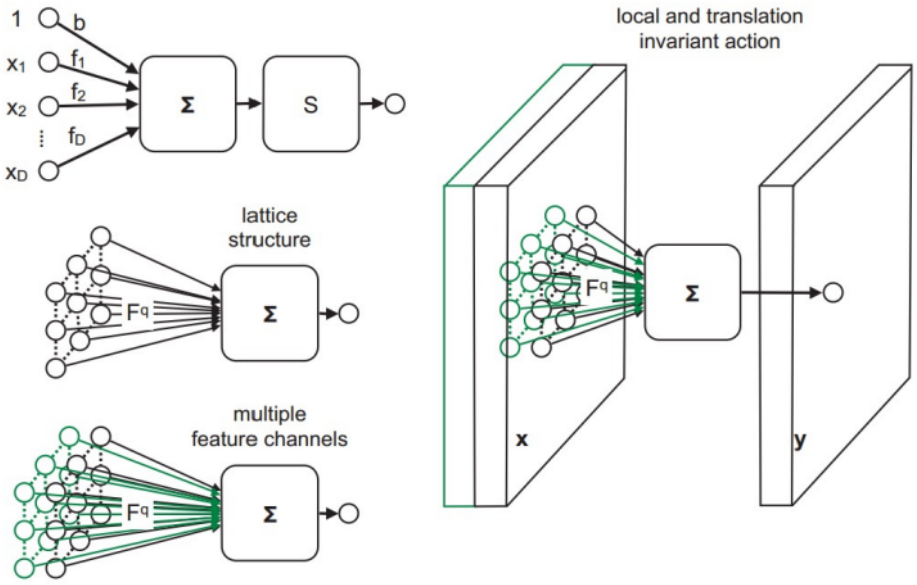
La formula prende una posizione nell'immagine (x), moltiplichiamo elemento per elemento con il filtro (F), e poi sommiamo su tutte le posizioni spaziali e tutti i canali di profondità questa moltiplicazione. A questo sommiamo il bias y_q. Raggiungiamo il risultato di quella posizione (il 4 è il risultato del filtro applicato alla prima posizione).

Poi ci muoviamo alla prossima posizione, e facciamo lo stesso procedimento (in questo caso shifto il filtro 1 pixel a destra). Il filtro DEVE essere completamente contenuto nell'immagine.

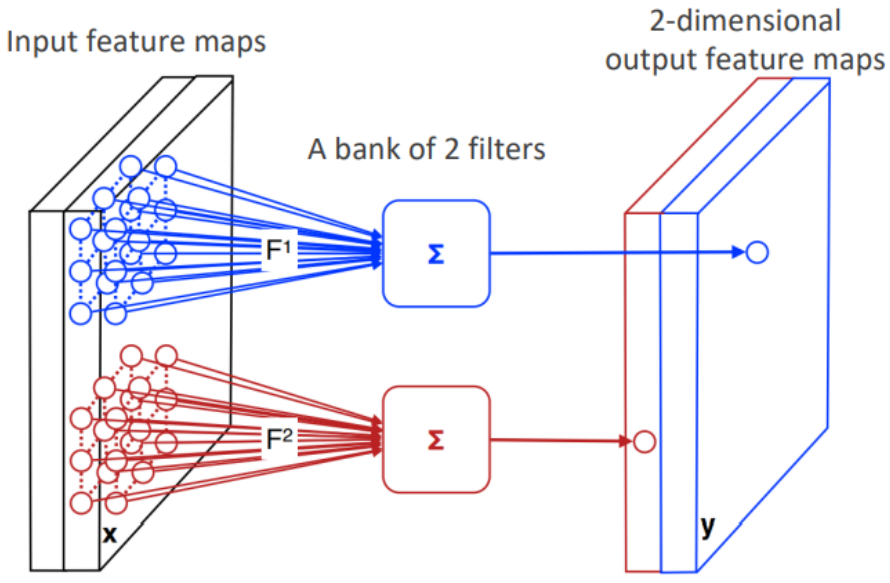
In questo esempio ci spostiamo di 1 posizione, ma questo può essere scelto. Questo valore è chiamato **stride**.

Qui dovrebbe caricare poi un esempio di computazione

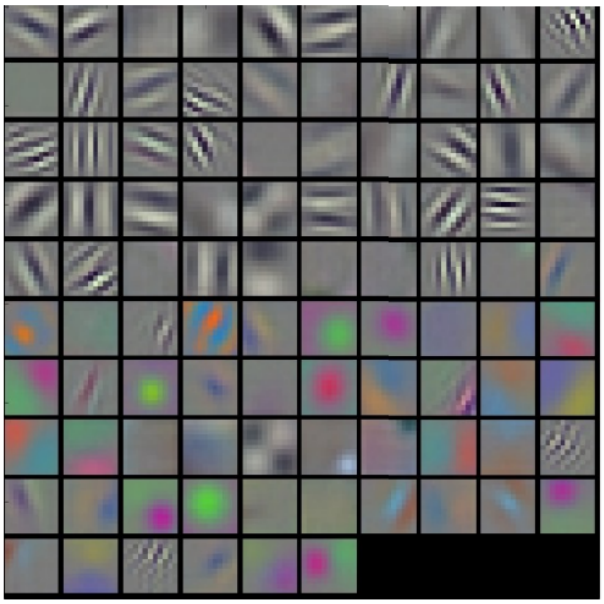
Questo è visualmente come è generato l'output:



Se invece abbiamo un bank filter con 2 filtri:



Questi sono i filtri del primo layer convoluzionale di alexnet:



A bank of 96 filters of the first convolutional layer of AlexNet trained on ILSVRC2012 dataset

- Each one is a 11×11 pixels 3D filter (it applies to a RGB image)

Quindi abbiamo 96 neuroni nel primo layer.

Abbiamo dei pattern bianco/nero a diverse orientazioni. Questi estraggono dei bordi delle immagini, a diverse orientazioni.

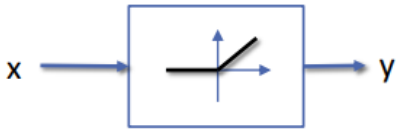
Poi abbiamo quelli che cercano il contrasto dei colori.

Poi nei layers successivi ci sarà un'astrazione aumentata, verranno combinati gli edges, verranno formati oggetti, poi scene... etc etc.

Activation functions

Fin ora abbiamo visto un'operazione lineare. Però vogliamo creare un classificatore non lineare. Dobbiamo aggiungere una non linearità.

Dopo aver computato un neurone, viene applicata la non linearità della funzione di attivazione.



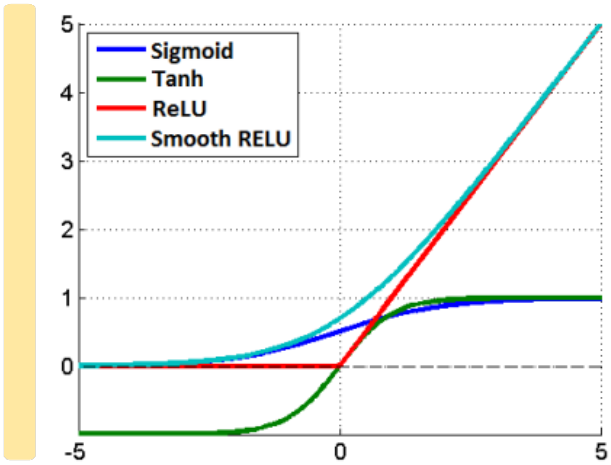
- Scalar non-linearity

$y = \frac{1}{1+e^{-x}}$ Sigmoid

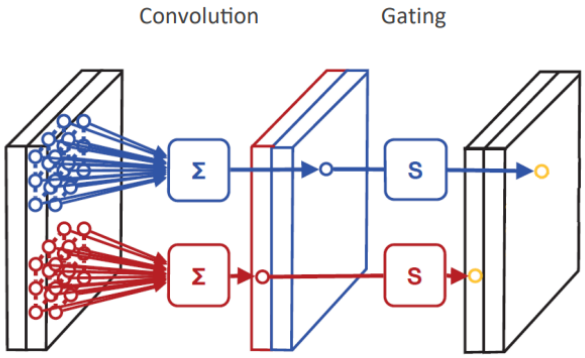
$y = \tanh x$ Hyperbolic tangent

$y = \max\{0, x\}$ Rectified Linear Unit (ReLU)

$y = \log\ (1 + e^{x})$ Smooth ReLU



L'output della funzione di attivazione avrà la stessa dimensione.

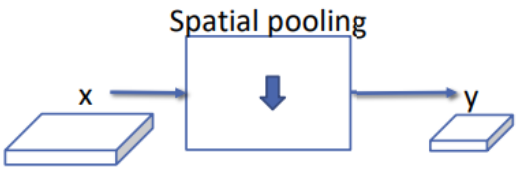


Filters are followed by non-linear operators (e.g. gating function)

Spatial pooling

Questo è un secondo tipo di layer.

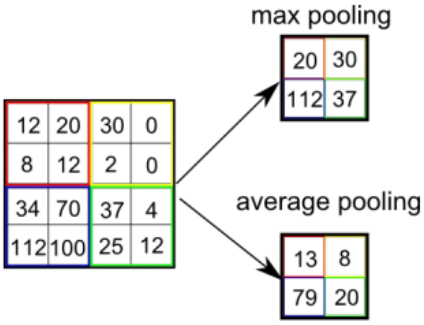
Questi funzionano canale per canale, prendono come input x, riducono la dimensione spaziale, e producono l'output y. Non cambia il numero di canali.



- Pooling and sub-sampling
- Encodes translation invariance
- Pooling computes the average/max of the features in a neighbourhood
- It is applied channel-by-channel

$$y_{ijk} = \max_{pq \in \Omega_{ij}} x_{pqk}$$ Max pooling

$$y_{ijk} = \operatorname{avg}_{pq \in \Omega_{ij}} x_{pqk}$$ Average pooling



Abbiamo 2 tipi:

- max pooling
- average pooling

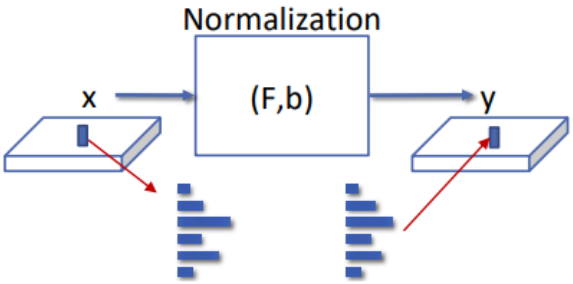
Funzionano nello stesso modo ma fanno un'operazione diversa. Dopo che abbiamo definito la grandezza della cella (tipo 2×2), nel caso di max pooling prendiamo il valore massimo di quest'area, poi ci muoviamo di uno stride uguale alla grandezza della cella. Otteniamo quindi un output 2×2 in questo caso, da un input 4×4. Quando usiamo average pooling prendiamo invece il valore medio.

Questo è usato per ridurre la dimensione spaziale dell'input di modo che il prossimo layer abbia un input più piccolo, che quindi farà meno computazioni.

Local response normalization (LRN)

Questo è un altro tipo di layer.

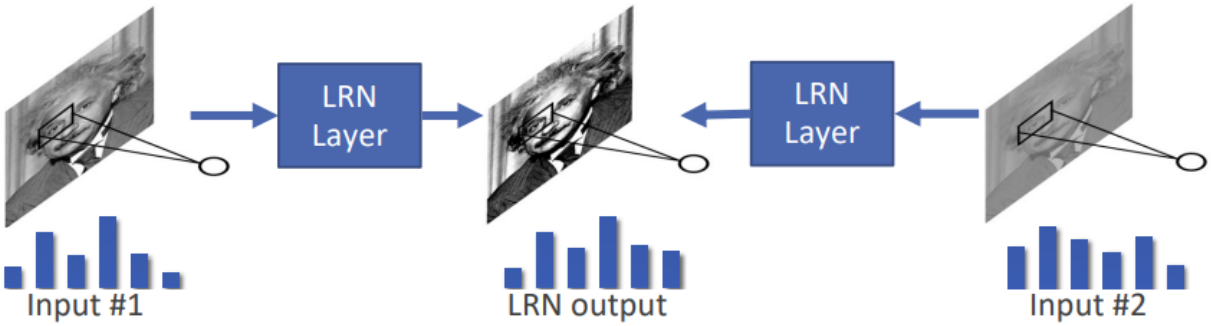
Prende un input x, lo processa, dandoci un output y che ha la stessa grandezza dell'input. Questo layer normalizza il valore dell'input. L'idea è quella di fare una sorta di contrast-normalization per migliorare la varianza.



Contrast normalization

Effects:

- Improves invariance
- Improves optimization
- Improves sparsity



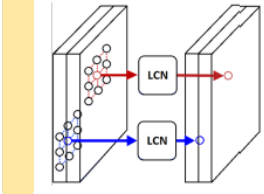
We want the same response

Per esempio vorremmo che le immagini nell'esempio abbiamo lo stesso output, quindi a prescindere dal contrasto dell'immagine.

Ci sono due modi diversi in cui questo può funzionare:

WITHIN CHANNEL

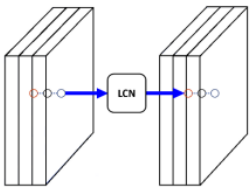
- Operates independently on different feature channels
- Rescales each input feature basing on a local neighborhood



$$y_{ijk} = x_{ijk}\left(\kappa + \alpha \sum_{(u,v)\in\mathcal{N}(i,j)} x_{uvk}^2\right)^{-\beta}$$

ACROSS CHANNELS

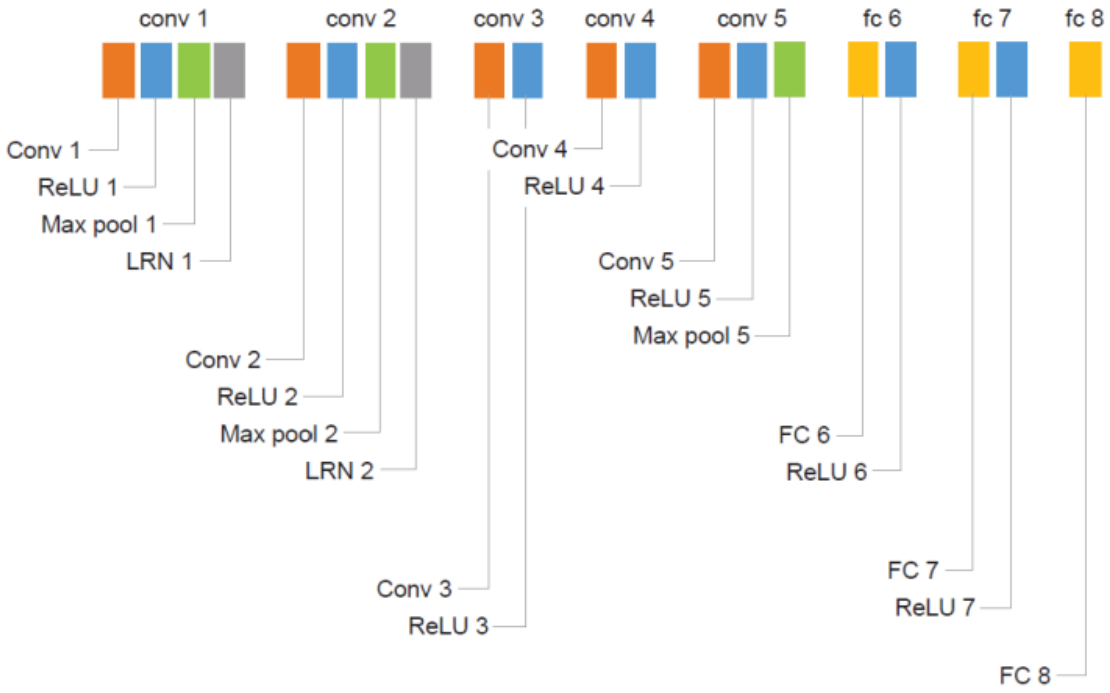
- Operates independently at each spatial location and groups of channels
- Normalizes groups G(k) of feature channels
- Groups are usually defined in a sliding window manner



$$y_{ijk} = x_{ijk}\left(\kappa + \alpha \sum_{q\in G(k)} x_{ijq}^2\right)^{-\beta}$$

Nel primo caso vogliamo che ciascun canale abbia una media di 0. Quindi ciascun canale è analizzato indipendentemente e ri-scalato. Nel secondo caso lavoriamo su tutti i canali contemporaneamente, vogliamo che la media tra tutti i canali sia 0.

Typical architecture



La divisione in blocchi è giusto per la visualizzazione.

Vediamo come la LRN è presente solo all'inizio della rete, perchè man mano che andiamo avanti stiamo aumentando il livello di astrazione. A livelli di astrazione alti, l'idea di normalizzare il contrasto non ha senso, non è utile per il training.

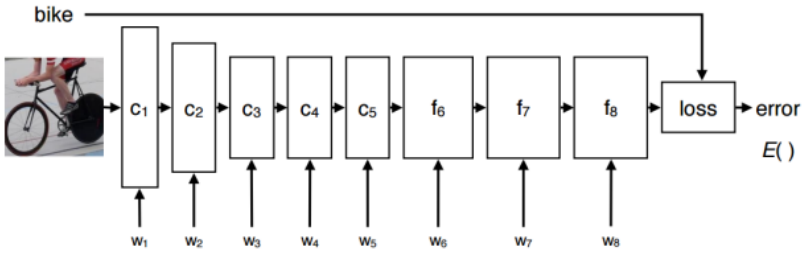
Dopo che abbiamo passato l'input per alcuni layer convoluzionali, facciamo un reshape come vettore monodimensionale e abbiamo dei layers fully connected classici.

Training

La domanda ora è come facciamo il training di un modello del genere?

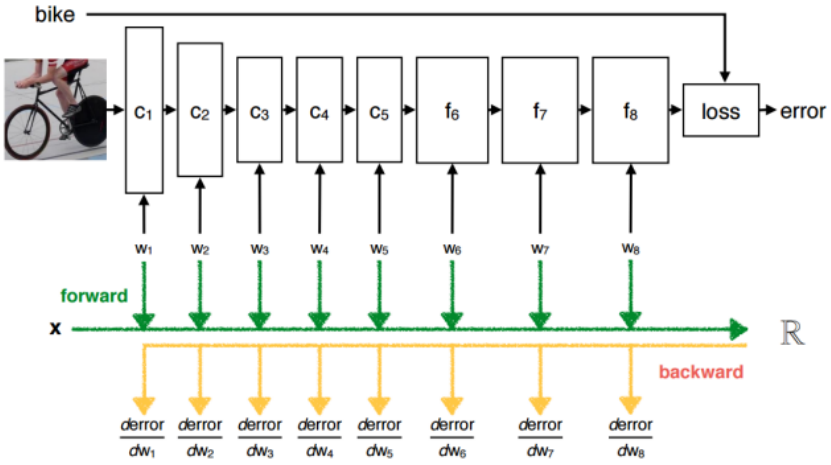
Il training avviene nello stesso modo delle reti neurali classiche. Usiamo la forward activation per poter computare la error function, vogliamo trovare i pesi di modo che l'errore sia minimizzato.

Qui ogni layer è una funzione, rispetto a quello che abbiamo fatto nel layer precedente. La computazione della derivata è meccanica. Il problema è trovare la soluzione per 0 della derivata, questo non lo possiamo fare quando l'equazione non è lineare. Quindi qui usiamo la backpropagation e la discesas del gradiente stocastica. Quindi usiamo i mini-batches, per fare updates più frequentemente durante il training, e perchè aggiungiamo una certa randomness (in base al mini-batch).



$$\text{argmin}\, E(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_8)$$

Questo meccanismo funziona allo stesso modo per i layer convoluzionali.

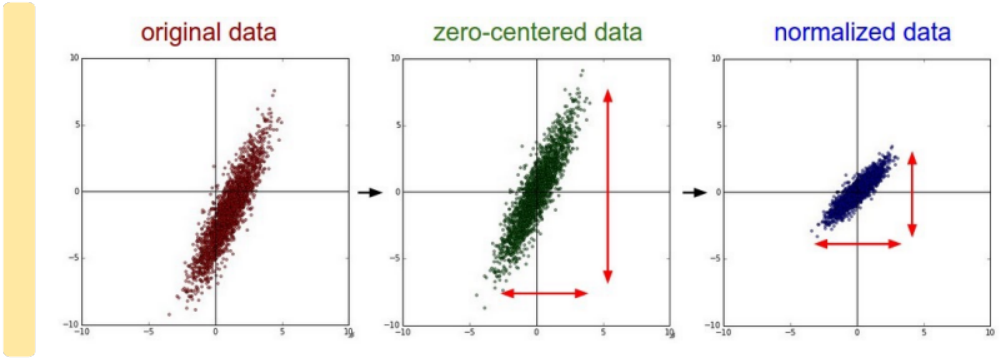


Il preprocessing può aiutare a velocizzare il training.

Potremmo normalizzare i dati di training. Questo è necessario perchè partiamo da una configurazione di pesi che è random (segue una certa distribuzione, di solito

centrata in 0), quindi se piazziamo i nostri dati nella stessa regione dove abbiamo i pesi, questo aiuta a velocizzare il training.

- **Local mean subtraction:** subtract the mean from original data
  - $X = X - \mu$
- **Normalization:** scale original data to a specific range
  - $X = \frac{X-\mu}{\sigma}$
- ...



Nello stesso modo in cui possiamo overfittare un problema in una rete normale, possiamo overfittare anche queste.

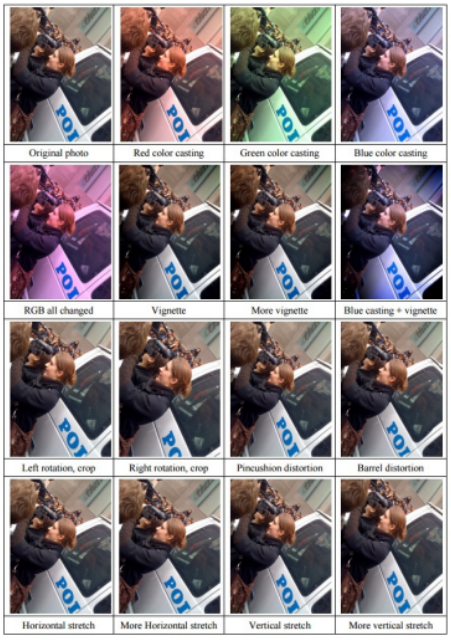
Possiamo quindi usare gli stessi metodi di regolarizzazione:

- Weight decay (penalizza i pesi grandi)
- Dropout (applicato solo in fully-connected layers, non quelli convoluzionali.
- Data augmentation

Augment the training set by “jittering” samples

Label preserving image transformations

- Horizontal flip
- Random crop
- Color casting
- Geometric distortion



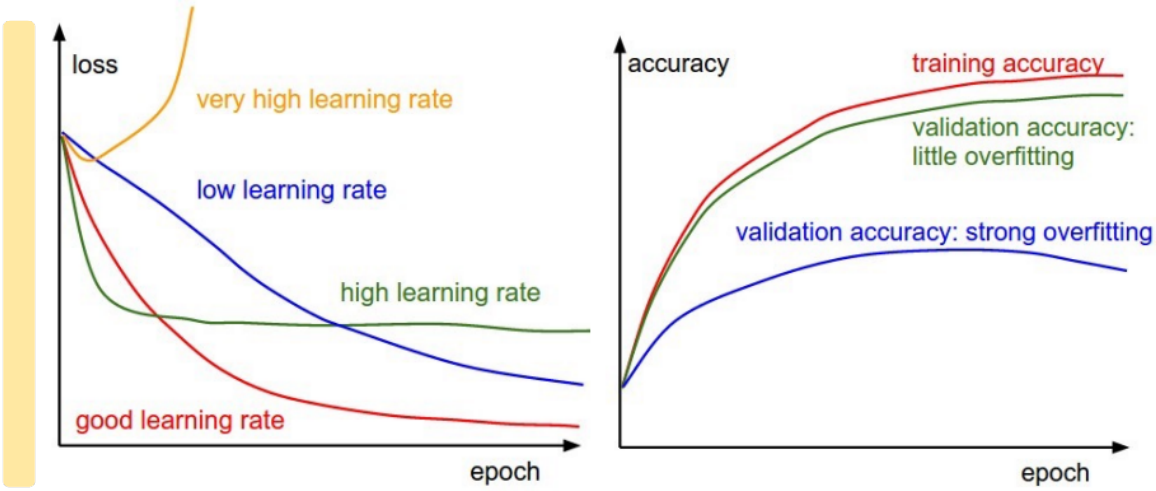
L’idea è quella che vogliamo delle trasformazioni delle immagini che mantengono i labels. Per esempio potremmo specchiare le immagini. Oppure posso fare un crop dell’immagine, possiamo fare delle colorazioni, delle distorsioni.

In base alla task, potremmo usare alcuni di questi metodi ma non altri.

Per esempio se stiamo lavorando sul dataset delle cifre, se faccio una rotazione di un 6 trovo un 9. Quindi le rotazioni non sono safe.

Per esempio se stiamo riconoscendo fiori, se il colore è una feature importante, allora non posso cambiare i colori delle immagini.

In ogni caso, avere dati reali darebbe più informazioni.



Vogliamo che la loss diminuisca durante il training. Potrebbe succedere che la loss inizia ad aumentare velocemente. Questo è il sintomo di un learning rate molto alto.