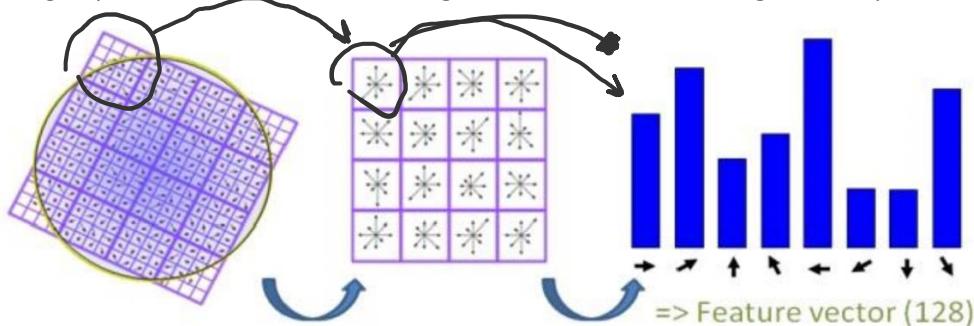


Pacchetto 9:

I description (feature) dovrebbero essere robusti alle variazioni di luminosità e alla rotazione.
I descriptor SIFT e SURF si basano su **gradienti orientati**.

SIFT (Scale-invariant feature transform):

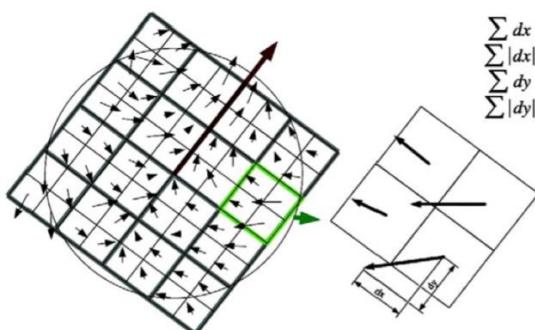
- Il primo step è quello di trovare la dominant orientation per ruotare la regione. Per ogni pixel viene calcolato un vettore che rappresenta quanto quel pixel differisce dai pixel vicini. Viene creato un grafico con tutte le direzioni dei pixel e il picco del grafico viene scelto come **dominant orientation**. Solitamente viene applicato un kernel gaussiano che serve per filtrare le aree non interessanti.
- Per descrivere la regione che abbiamo preso in considerazione si deve prendere una window 16x16 pixels, pesando i gradienti tramite una funzione gaussiana (dando più importanza a quelli centrali), per ogni quadrante 4x4 viene fatto un **gradient orientation histogram** composto da 8 bins.



- Gaussiana per eliminare rumore à Giro la regione tramite la dominant orientation à Calcolo i gradienti per ogni pixel della finestra à Applico la Gaussian fall off function à Per ogni quadrante 4x4 calcolo un istogramma di vettori. Lo faccio per ogni quadrante quindi in totale avrò 16 istogrammi ognuno da 8 colonne.
- Il picco che serve per calcolare la dominant orientation è robusto alle variazioni di luminosità, dato che ruoto la regione tramite la dominant orientation allora SIFT è rotation invariant.

SURF (Speed Up Robust Features)

- Considera le stesse informazioni di STIF ma viene costruito in maniera molto più efficiente per via del calcolo della somma dei pixel fatto con l'integral images.
- Lavora sempre con la stessa immagine e non viene ruotata ogni volta, nemmeno per la detection. Vengono usate le immagini integrate ad ogni step. Viene creato un vicinato circolare di raggio $6s$ dal centro della regione, dove s è la distanza al quale è stato trovato un punto di interesse.
- Viene valutata la **Haar wavelet** rispetto a x e y pesando la risposta con un kernel gaussiano dove $\sigma = 2s$ per ottenere i vettori. Il vettore più lungo è preso come **orientation** mentre la finestra è larga 60 gradi in totale.
- Viene creato un quadrato 20sx20s (nell'esempio 8sx8s) raggruppato in sottoregioni grandi 4x4 e per ogni sottoregione vengono valutate le haar wavelets.



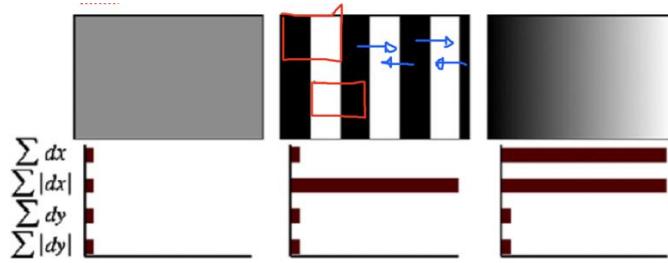


Fig. 13. The descriptor entries of a sub-region represent the nature of the underlying intensity pattern. Left: In case of a homogeneous region, all values are relatively low. Middle: In presence of frequencies in x direction, the value of $\sum |dx|$ is high, but all others remain low. If the intensity is gradually increasing in x direction, both values $\sum dx$ and $\sum |dx|$ are high.

- Vengono calcolate le somme delle risposte delle haar wavelet che ci danno informazioni sulla local region. Viene fatto sia il valore assoluto che no per distinguere, come nell'esempio, dallo zebraato allo sfumato.

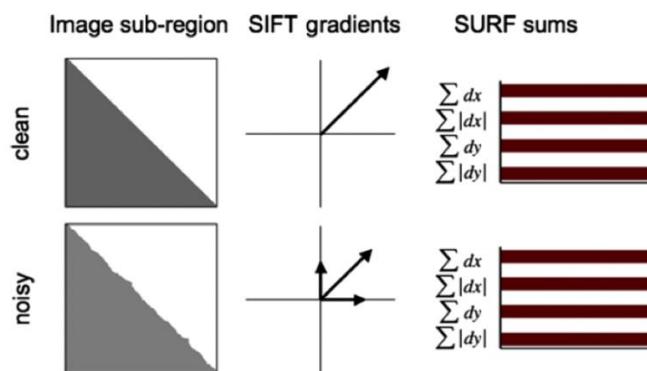


Fig. 14. Due to the global integration of SURF's descriptor, it stays more robust to various image perturbations than the more locally operating SIFT descriptor.

- SURF è più robusto al rumore rispetto a SIFT. Nel mondo reale SIFT è meglio mentre SURF è più efficiente.

BINARY DESCRIPTORS

- Sono costruiti da set di coppie di punti di cui si compara l'intensità, ogni bit del descrittore è il risultato di una sola comparazione. Viene usata la Hamming Distance come misura di similarità.
- Il pattern di sampling è prefissato e vari binary descriptors si differenziano in base a questo:
 - BRIEF
 - ORB
 - BRISK

BRIEF (Binary Robust Independent Elementary Features)

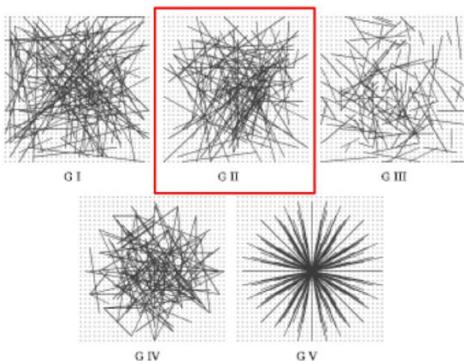
Binary test

$$\tau(p; x, y) := \begin{cases} 1 & \text{if } p(x) < p(y) \\ 0 & \text{otherwise} \end{cases}$$

descriptor

$$f_{n_d}(p) := \sum_{1 \leq i \leq n_d} 2^{i-1} \tau(p; x_i, y_j)$$

- Binary test rappresenta un bit, il descriptor è la concatenazione di tutti i binary test. Per ogni regione andiamo a fare smoothing e poi i binary test.
- Posso fare un sampling di 128, 256 o 512 comparazioni (eg. bits) usando una distribuzione gaussiana isotropica (quella in rosso).



ORB (Oriented FAST and Rotated BRIEF)

- La detection è basata su FAST, la misura degli angoli è basata su Harris.
 - I descrittori di BRIEF sono orientati secondo i keypoint.
 - Vengono fatte 256 coppie di comparazione di intensità, esse sono state scelte tramite machine learning da un set di 205590 coppie, massimizzando la varianza del descrittore e minimizzando la correlazione.

BRISK (Binary Robust Invariant Scalable Keypoints)

- La detection (scale-space) dei keypoint è simile a quello di SIFT, il descrittore è simile a quello di BRIEF.
 - Each sample point representing a Gaussian blurring of its surrounding pixels (?)
 - Il gradiente dominante è ottenuto facendo comparazioni a lunga distanza.
 - 512 comparazioni a corta distanza sono usate per la descrizione.
 - E' scale and rotation invariant.

11. Local Features Matching

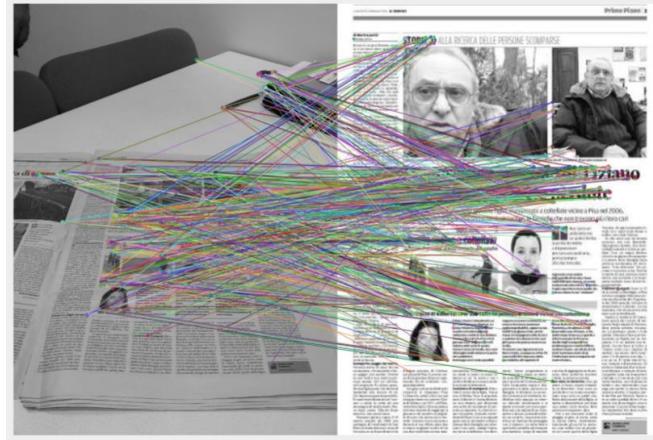
Dopo aver trovato la feature attraverso i detectors e dopo aver descritto il contenuto della feature attraverso i descrittori (SIFT, SURF o binary descriptors) possiamo andare a confrontare local features appartenenti a immagini differenti.

Features matching

Viene usata la distanza per valutare la *somiglianza fra due local features*, se abbiamo usato SIFT o SURF è necessario utilizzare la **distanza euclidea**, se abbiamo fatto uso di descrittori binari allora possiamo adottare la **hamming distance**. Questa distanza consiste nel contare il numero di bit differenti fra due local features.

Distanza euclidea $d(X, Y) = \sqrt{\sum_{i=1}^k (x_i - y_i)^2}$

Con l' **1-NN** ogni feature della prima immagine viene associata con la più simile della seconda immagine, quindi con la più vicina. Nell'immagine sottostante si può notare che vengono creati tantissimi match fra le features, è necessario trovare un metodo per filtrare i match e scegliere solo quelli necessari.



Matches Filtering

Il nostro scopo è quello di selezionare solo i match che hanno alta probabilità di essere buoni. Per fare questo i **binary descriptors** usano una **distance threshold**.

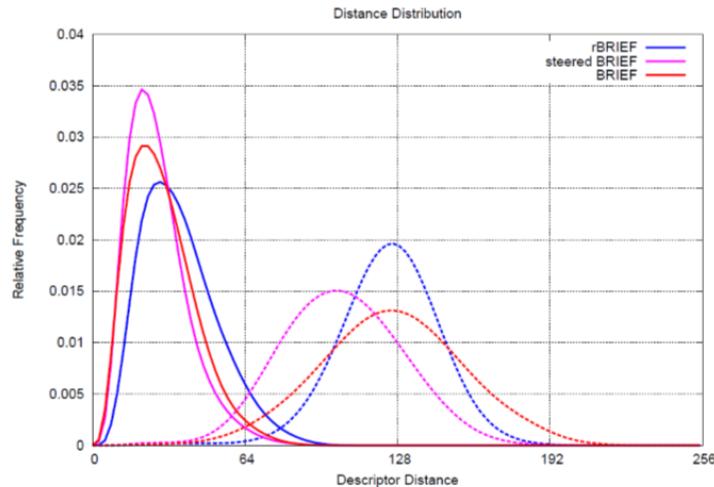


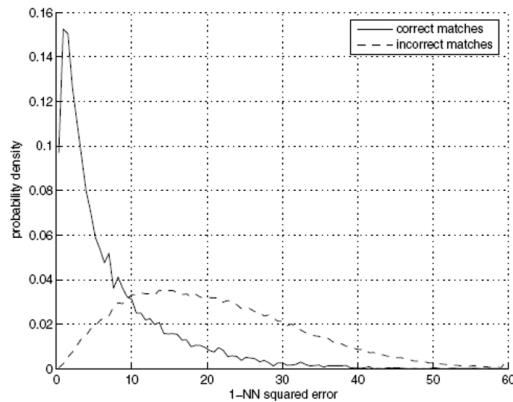
Figure 5. The dotted lines show the distances of a keypoint to outliers, while the solid lines denote the distances only between inlier matches for three feature vectors: BRIEF, steered BRIEF (Section 4.1), and rBRIEF (Section 4.3).

In figura si può notare che mettendo una soglia di circa 50/60 vado a escludere tutti i match sbagliati (quelli tratteggiati), nell'immagine successiva viene posta una soglia pari a 45 e si utilizza ORB sulla stessa immagine del giornale.



È facile notare che le feature che fanno match sono molte meno e sono molto più precise rispetto ai match ottenuti tramite 1-NN senza soglia.

SIFT utilizza una *threshold sul rateo fra la distanza del primo NN e il secondo NN*. È necessario quindi calcolare il secondo vicino.



Si può notare che il SIFT con solo l'1-NN non riesce a eliminare del tutti i bad match anche utilizzando una soglia piccola, infatti anche prima di 10 abbiamo un accavallamento delle due curve.

Nell'immagine seguente si può notare che il rateo permettere di eliminare quasi completamente i bad match prima di una certa soglia.

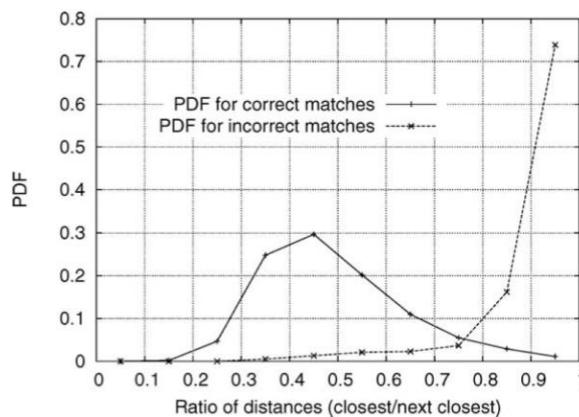


Figure 11. The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 keypoints, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

Quando voglio fare il match fra un logo e un oggetto contenete quel logo, come mostrato in figura, possono tranquillamente applicare i metodi visti.



Dobbiamo fare attenzione all'ordine quando nella scena (query) sono presenti più istanze dell'oggetto (train). Se utilizzassi come query il logo, **NON SI DEVE FARE**, andrei a trovare solo il match corrispondente all'1-NN, perdendo quindi tutti gli altri. Se come query utilizzassi, invece, l'immagine dello scaffale, per ogni logo sulla bottiglia andrei a individuare il suo 1-NN nel logo. Per fare i match si parte dalla scena (query) e si cerca nell'oggetto (train).



Geometric Transformation Estimation

Dobbiamo considerare le informazioni riguardanti le regioni descritte in modo da poter comprendere la relazione geometrica fra le due immagini.

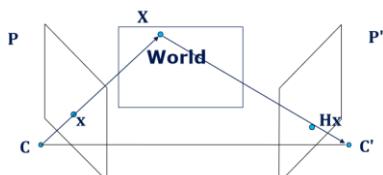
Le regioni sono descritte da:

- ☒ Un **centro** (p_x, p_y)
- ☒ Una **grandezza** o una **scala** tramite la quale la regione è stata selezionata
- ☒ Un **orientamento**, dato che le local features sono rotation invariant dobbiamo salvare anche l'orientamento originario
- ☒ La **local feature** perché a questo punto abbiamo già delle coppie che fanno match

Si vanno a cercare tre tipi di trasformazioni:

- ☒ Rotazione, scala e traslazione
- ☒ Trasformazioni Affini
- ☒ Omografie

Parlando dell'omografia, questa trasformazione trasforma l'immagine come si vedesse da un altro punto di vista.



Facciamo un esempio per chiarire le idee. La prima immagine del duomo di Pisa è l'oggetto, la seconda immagine è quella che vogliamo far combaciare con la prima.



Si possono applicare **rotazioni, scale e traslazioni** alla seconda immagine producendo il seguente risultato



Si può applicare una **trasformazione affine** che produce il seguente risultato



Entrambi i casi non producono risultati soddisfacenti in quanto le due immagini di partenza sono state scattate con due angolazioni differenti, applicando quindi l'**omografia** si ottiene il risultato migliore



L'omografia si può anche combinare a rotazioni, traslazioni e inclinazioni; tuttavia, non è sempre la scelta migliore da fare. Prendiamo ad esempio due immagini con vista dall'alto.



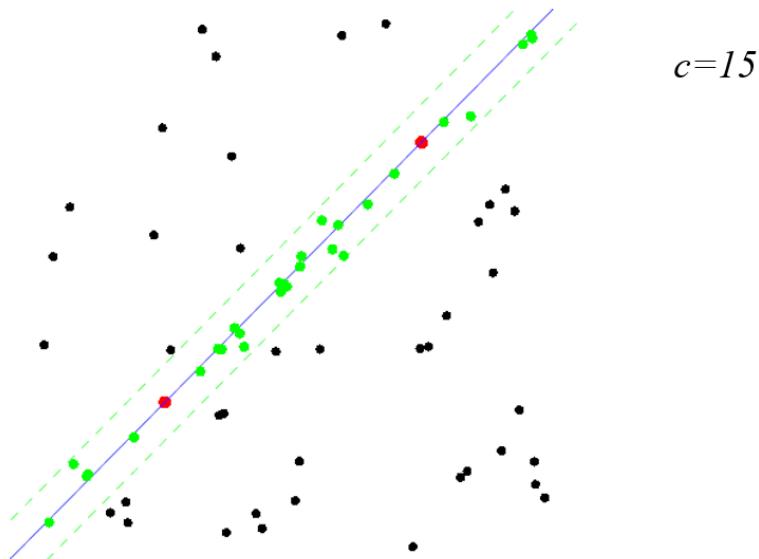


Come possiamo vedere per far coincidere le due immagini è necessaria solamente applicare rotazione, traslazione e scala.

RANSAC

RANSAC sta per **Random Sample Consensus**, dato un set di dati osservati, si stima un modello che possa spiegare o essere modellato ai dati. Iterativamente si fanno le seguenti azioni:

- ② Si seleziona un sottoinsieme dei dati originali in maniera randomica
- ② Il modello viene creato tramite i sample
- ② Vengono testati tutti gli altri dati tramite il modello appena creato, tutti i punti che producono un buon risultato attraverso il modello vengono considerati parte del **consensus set**
- ② Il modello creato è considerato attendibile se abbastanza punti sono stati inseriti nel consensus set



Abbiamo visto come l'omografia sia spesso la scelta migliore, il problema è però *trovare i 4 valori della matrice per applicare l'omografia*. Tramite RANSAC si possono stimare questi valori nel seguente modo:

- ② Si selezionano 4 coppie di features in maniera casuale (come nel giornal)
- ② Si calcola l'omografia
- ② Si calcolano gli **inliers** che sono i dati che sono consistenti con la trasformazione
- ② Si salva il set di inlier più grande trovato fino a questo momento (e si torna al punto 1)
- ② Si ricalcolano i mini quadrati di H sul set di inliers più grande

È possibile vedere nella seguente immagine i punti che fanno match riconosciuti tramite RANSAC



Measuring the confidence

Posso usare due metodi per calcolare la confidence della comparazione di due immagini.

- ② **#Inliers:** non è affatto dall'occlusion o dalla grandezza relativa fra oggetto e scena, come contro questo tipo di misura preferisce oggetti complessi.
- ② **#inliers/#ObjectLFs:** è relativo a quanto dell'oggetto è presente nella scena, come contro c'è il fatto che è sensibile all'occlusion dell'oggetto
- ② **#inliers/#SceleLFs:** è relativo a quanto della scena è occupato dall'oggetto, come contro c'è il problema di rilevare piccoli oggetti in scene grandi.

È importante sottolineare che la differenza in pixel nella scala dell'oggetto nelle due immagini può far variare significativamente le misure appena elencate.

Se non si è soddisfatti dei punti trovati nell'immagine precedente, si può applicare prima l'omografia e successivamente identificare i punti. Avremo sicuramente più match come mostrato nell'immagine seguente.



12/13. Aggregating Local Descriptors

Prima di tutto un breve riassunto riguardante le features.

Global

- Describe **overall** visual appearance (colors, textures, edges, etc...)
- Compared by using a **distance** or (dis)similarity function

Local

- Visual appearance of regions (of interest)
- **Distance** is computed between features
- Whole images are compared analyzing **matching points**



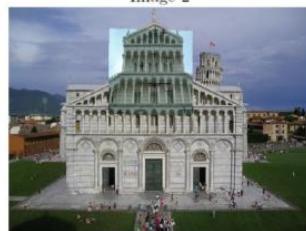
Image 1



Image 2



Rotation, Scale and Translation



Affine



Homography

Bag-Of-Words

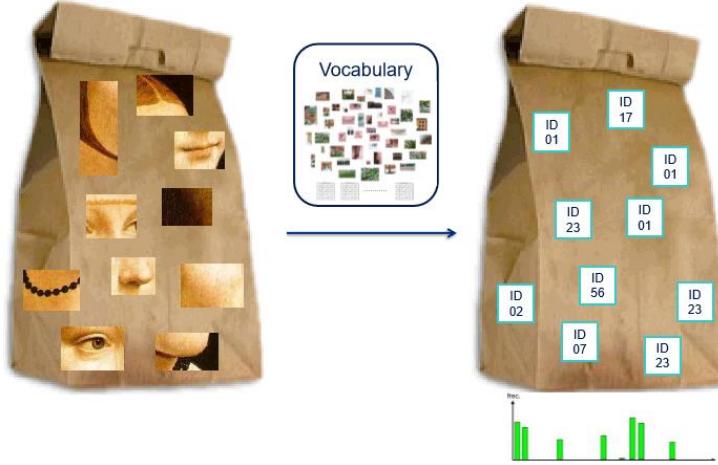
Facciamo prima un esempio visivo per capire meglio, nella seguente immagine viene mostrato il **bag-of-words**, chiamato anche Bag-of-Features, applicato ad un discorso del presidente degli stati uniti. Vengono riportare tutte le parole presenti nel discorso e, più una parola appare nel discorso e più la parola è grande. Il bag-of-words funziona come **un istogramma**; infatti, fa capire a colpo d'occhio quali sono le parole più usate e, di conseguenza, di cosa si stia parlando nel discorso.

Viene usato anche per **comparare due testi** infatti guardando il BoW di essi si può capire se trattano di argomenti simili.



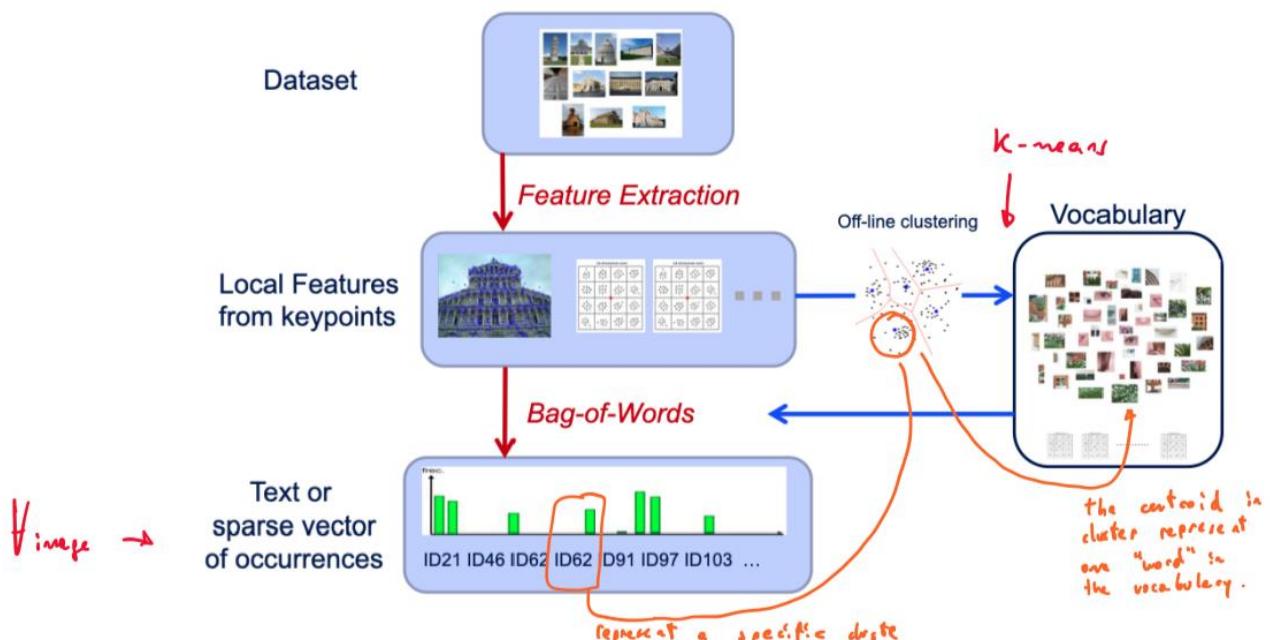
Le parole che appaiono sia in singolare che in plurale vengono unite in un'unica occorrenza.

Il bag-of-words funziona attraverso un **dizionario**, applicarlo a dei testi risulta semplice in quanto un dizionario contenente tutte le parole esiste per ogni lingua. Se vogliamo applicare il bag-of-words a delle immagini descritte tramite features incontriamo un problema, non disponiamo di un dizionario che ci permetta la classificazione di ogni feature.



È facile risolvere questo problema, per **creare il dizionario**:

- ② Si parte da un **dataset** contenente un elevato numero di immagini
- ② Si esegue una **feature extraction**
- ② Viene fatto un **clustering off-line** con le local feature appena estratte
- ② Dopo aver clusterato ogni local feature di ogni immagine del dataset di partenza si individua un **centroide** per ogni cluster, ad esempio utilizzando il *k-mean*, che rappresenta una "parola" all'interno del dizionario
- ② Per ogni immagine a cui vogliamo applicare il bag-of-words si valuta un **istogramma**. Per ogni local feature di un'immagine si guarda a che cluster appartiene e si aumenta di uno in numero di occorrenze relative al centroide di quel cluster. Nell'immagine seguente è illustrato il metodo in maniera visiva.

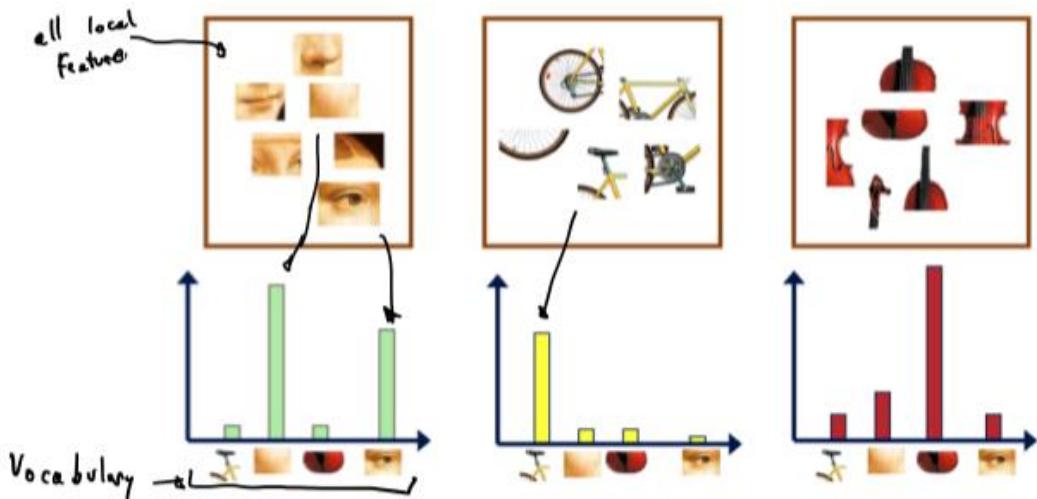


Di seguito un esempio di ordine di grandezza del processo appena descritto:

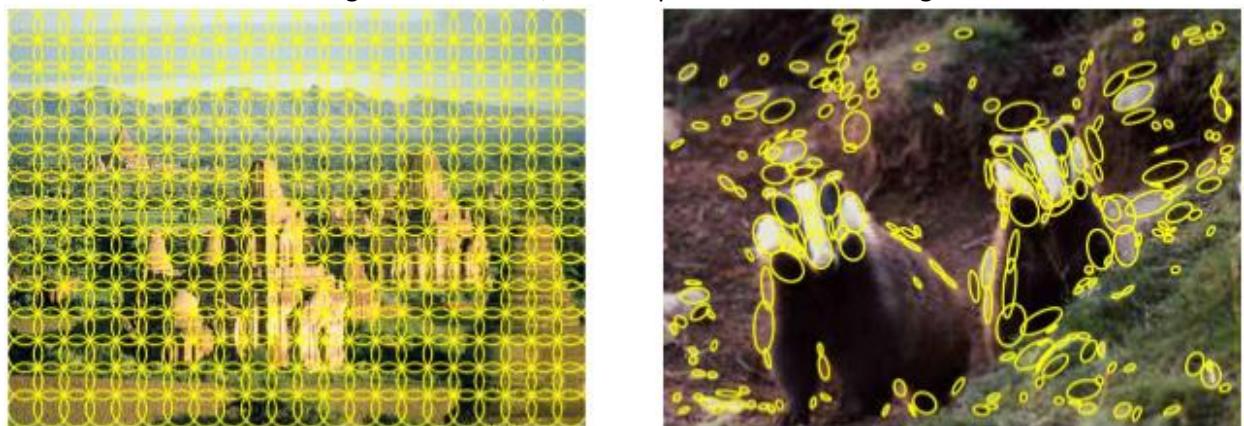
- ② Dataset: migliaia
- ② Local features per ogni immagine: 1000
- ② Vocabolario: 10'000

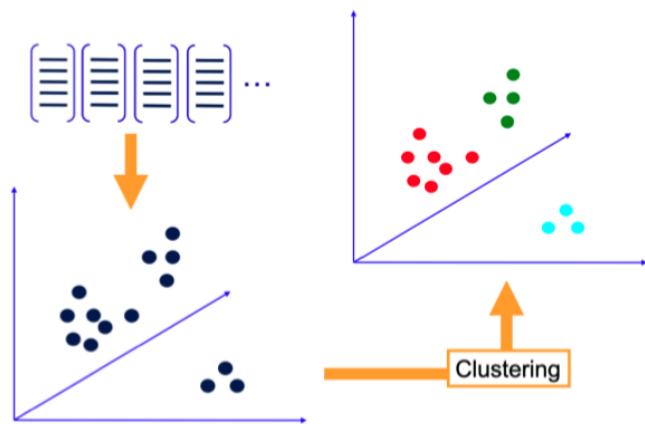
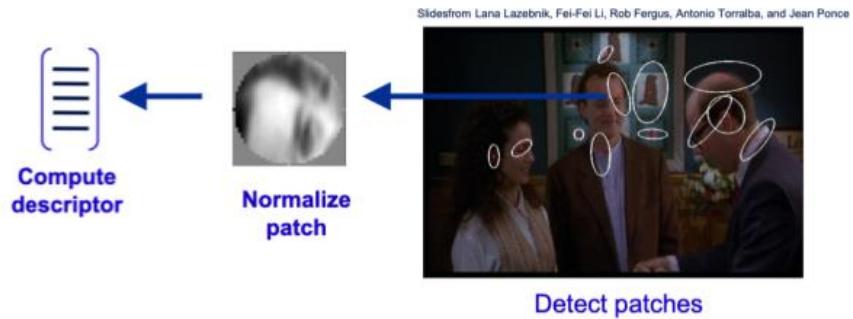
Notiamo come l'istogramma rappresentante un'immagine è **sparso**, infatti di 10000 "visual word" del dizionario solo un migliaio ne vengono riconosciute.

Usare l'istogramma significa **quantizzare le feature** usando un vocabolario, di seguito è illustrato un esempio pratico dove il vocabolario è già stato creato.



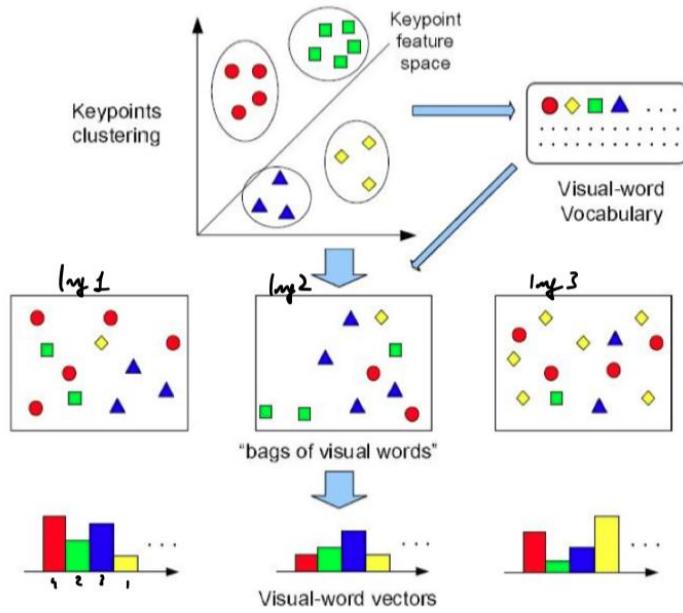
La feature extraction può essere fatta tramite una griglia regolare che copre l'intera immagine oppure si può fare analizzando solo le regioni d'interesse, un esempio è mostrato nella figura sottostante.





Visual vocabulary

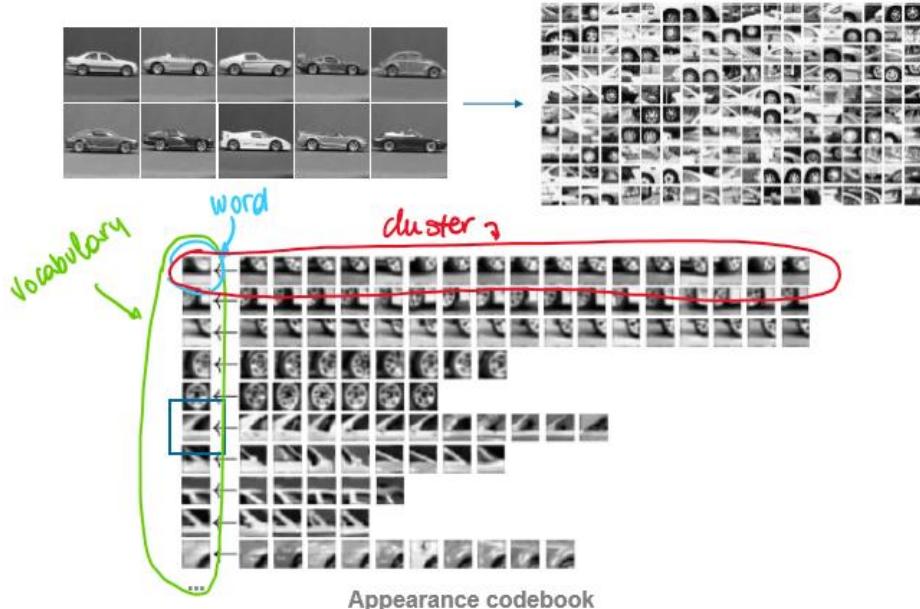
center of cluster "face"



È importante sottolineare che il vocabolario, anche chiamato **codebook**, può essere calcolato tramite un processo di unsupervised learning su un test set differente, inoltre se il test set è abbastanza rappresentativo allora il codebook può essere considerato universale.

Il codebook viene usato per quantizzare le features, un vettore di feature viene mappato all'indice del **codevector** (visual word) più vicino all'interno del codebook.

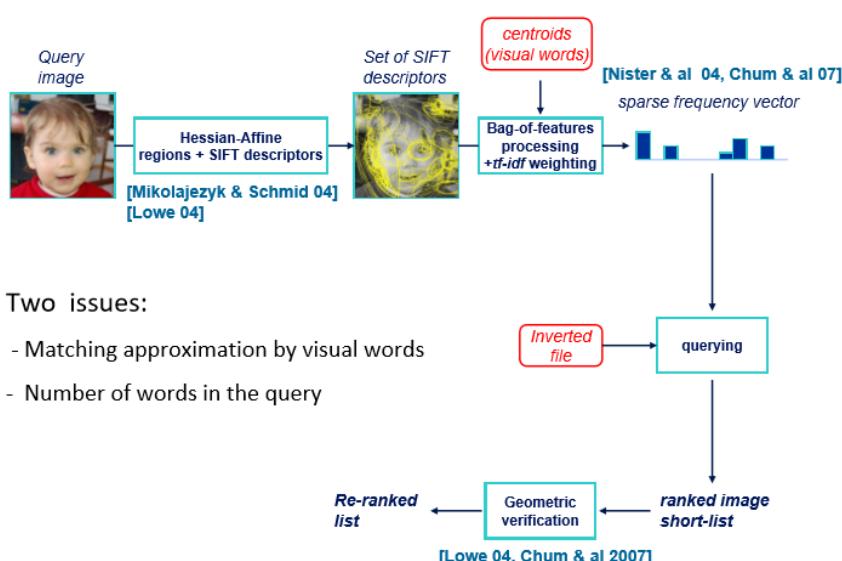
Ecco un esempio di un codebook creato per il riconoscimento di automobili.



Sorge un problema quando si deve scegliere la **grandezza** del vocabolario:

- ② Se si sceglie il vocabolario *troppo piccolo* allora si ottengono delle visual words che non rappresentano tutte le patch, si ottiene quindi un vocabolario troppo generico
- ② Se si sceglie il vocabolario *tropo grosso* ottengo un vocabolario troppo specifico che porta a **overfitting**. Le local feature che rappresentano la stessa cosa potrebbero appartenere a cluster differenti.

Bag-of-words può anche essere utilizzato per cercare immagini simili semplicemente cercando bag-of-words simili. Ad esempio, se in una scena voglio trovare un gran numero di loghi posso, prima di tutto, utilizzare il bag-of-words per cercare i loghi più simili alla scena. In questo modo prima riduco i loghi che hanno possibilità di essere nella scena e solo dopo applico le feature matching. Il processo è mostrato nell'immagine seguente.



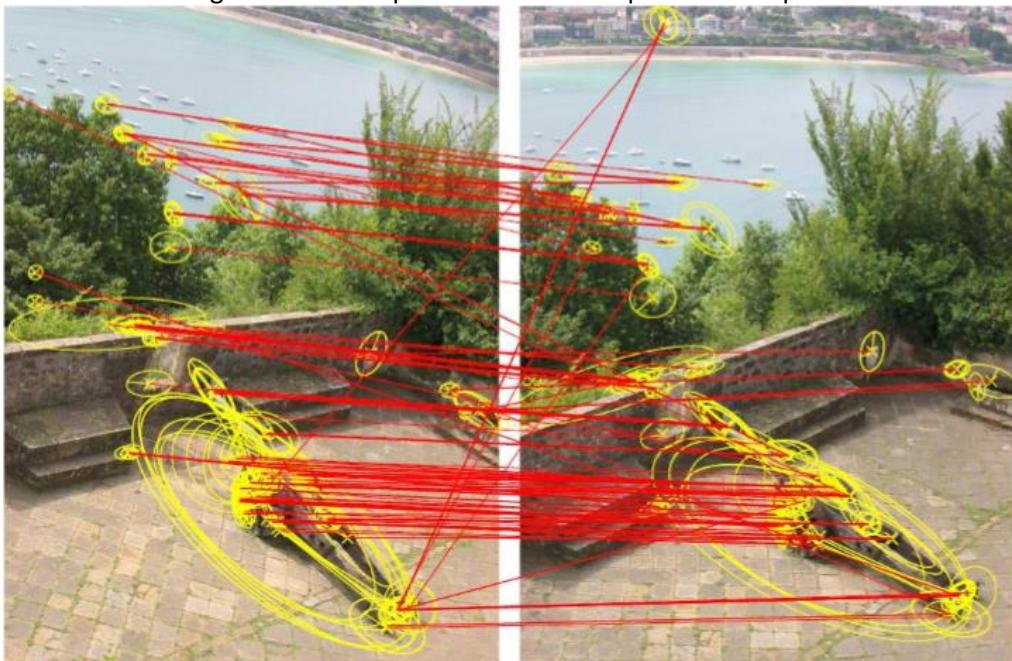
Bag-of-words può anche essere usato per fare **local descriptors matching**. Per farlo è molto semplice, il *same-word matching* sostituisce il local feature matching e filtering. Questo metodo porta a delle differenze:

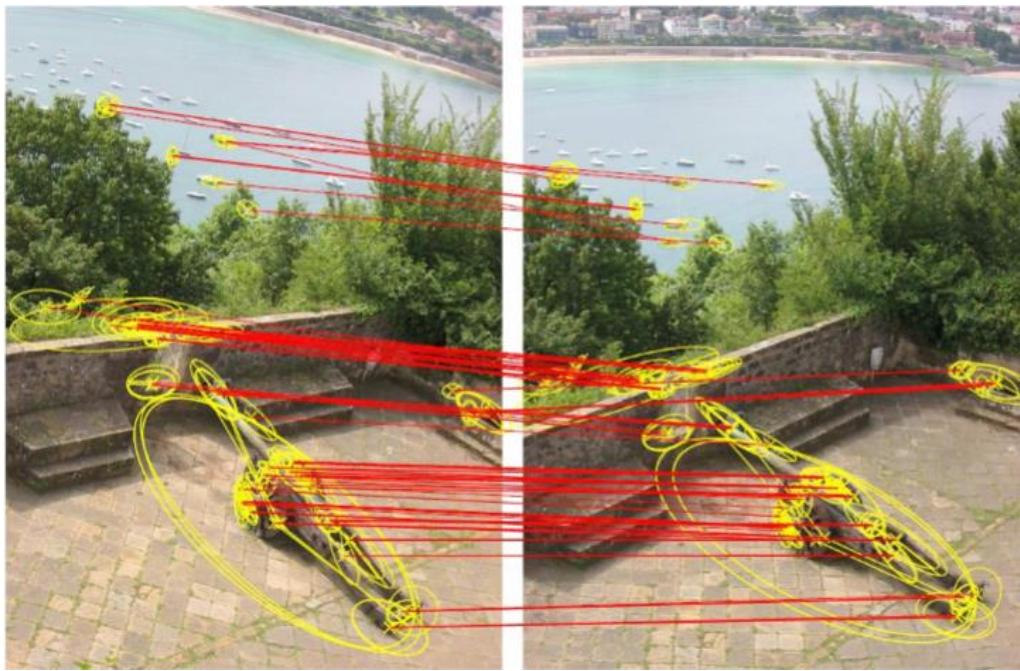
- ② Una regione nell'immagine di query (scena) può essere associata con 0....n regioni nell'oggetto.
- ② Non si deve più calcolare l'1-NN e il 2-NN
- ② I match possono essere definiti tramite indici invertiti

Il processo consiste adesso nei seguenti **4 step**:

1. I descrittori sono le word, si deve trovare l'1-NN fra le word nel vocabolario per tutte le regioni della query (scena)
2. Si cercano i match per ogni regione query. Data una regione query e la sua word associata, ogni regione nel training (oggetto) associata alla stessa word risulta in un match
3. Geometric transformation estimation
4. Measure the confidence

Nelle due immagini successive vediamo come le local feature vengono collegate fra di loro. Nella prima si usa un vocabolario composto da 20K parole, possiamo notare come siano presenti sia match buoni che match sbagliati; questo è dovuto al fatto che non ci sono abbastanza parole all'interno del vocabolario. Aumentando le parole a 200K vediamo il risultato nella seconda immagine, possiamo notare come la maggior parte dei match sbagliati non sono presenti lasciando spazio solo a quelli corretti.





VLAD

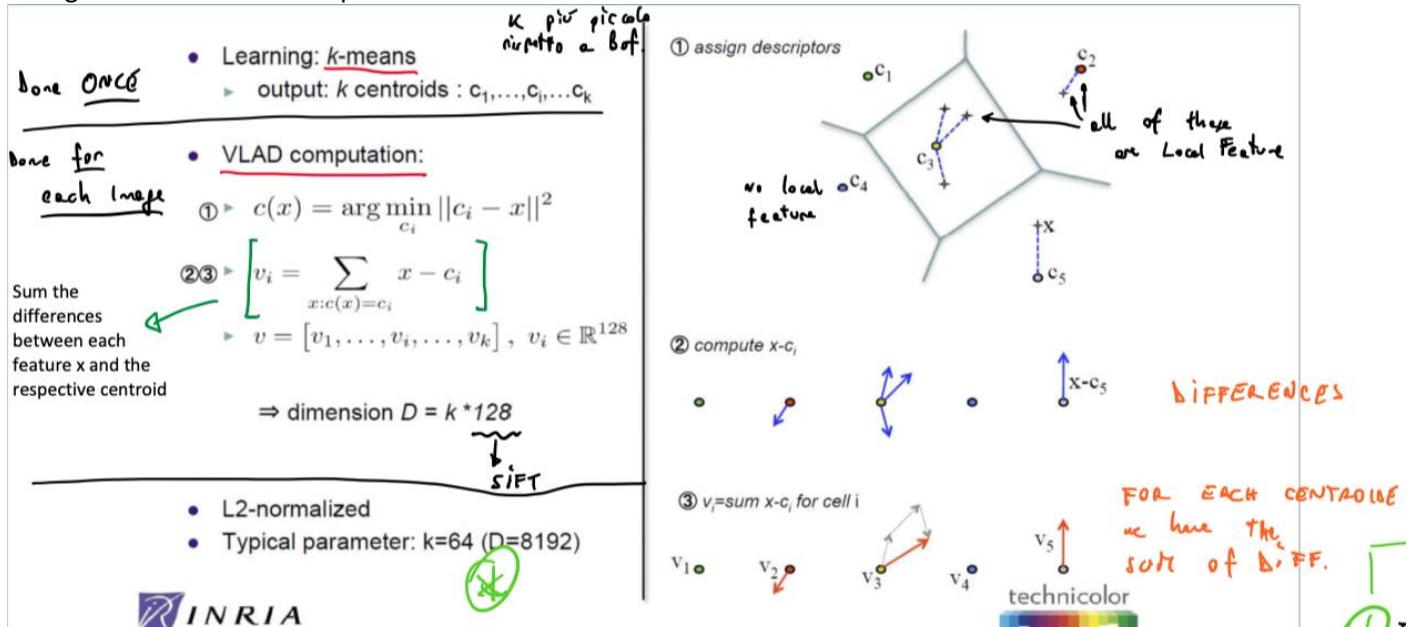
Siamo nell'ambito large scale del visual information retrieval.

L'aggregazione di descrittori locali viene fatta con il bag of words, come visto prima, tramite il VLAD oppure con i fisher vector.

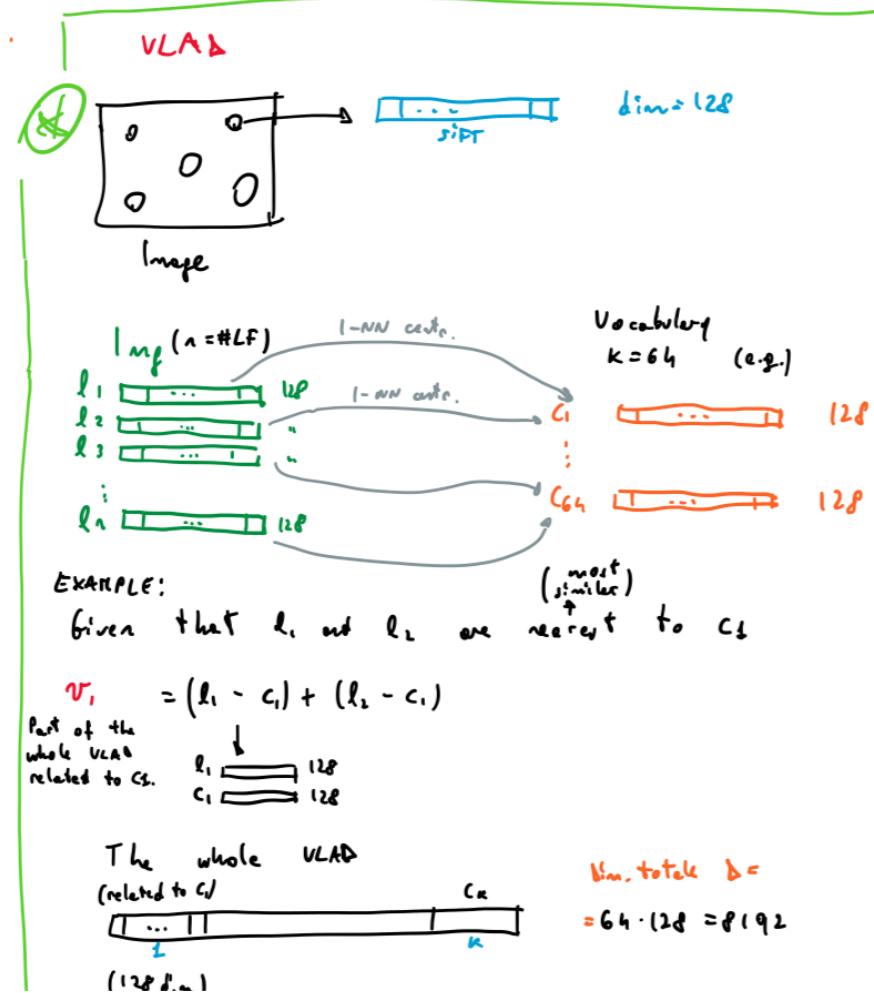
Abbiamo più vettori (senza metriche dello spazio) dello stesso tipo (eg SIFT) che sono aggregato in un singolo vettore di dimensione fissa. Vengono scelti dai 64 ai 512 centroidi, notiamo che sono molti meno rispetto a quelli del BoF. Dato che la dimensione è comunque grande (eg. 64 centroids * 128 features dimension = 8192) possiamo applicare il PCA sia su feature che nell' aggregazione.

VLAD ha una relazione con BoF e può essere visto come un'infinite soft-assignment (come Fisher Vector).

Di seguito viene mostrato il procedimento

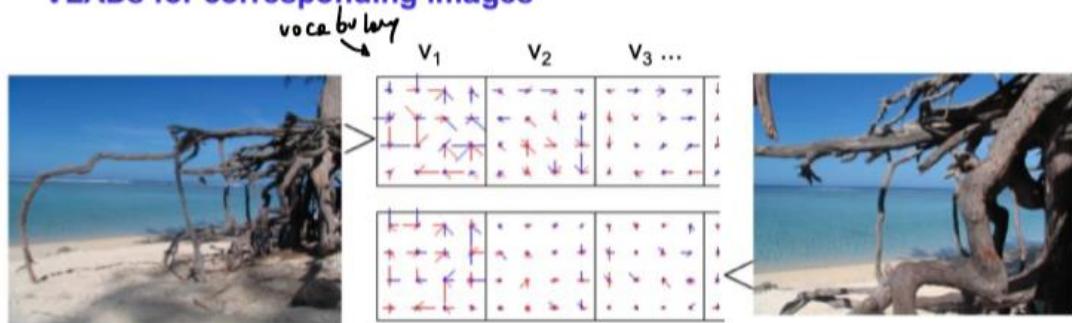


La dimensione $D=8192$ da cosa è data? Qualsiasi sia il numero delle feature globali rimane sempre 8192, perché questa è data dalla dimensione del vocabolario.



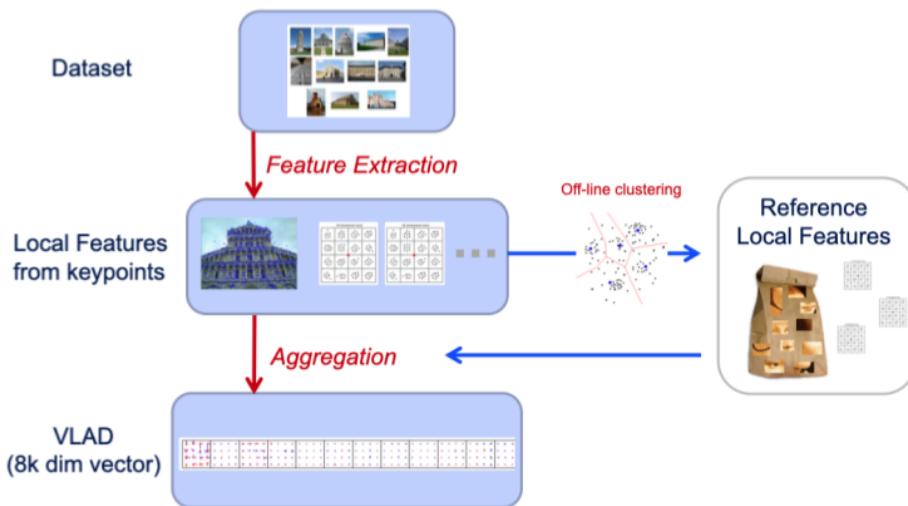
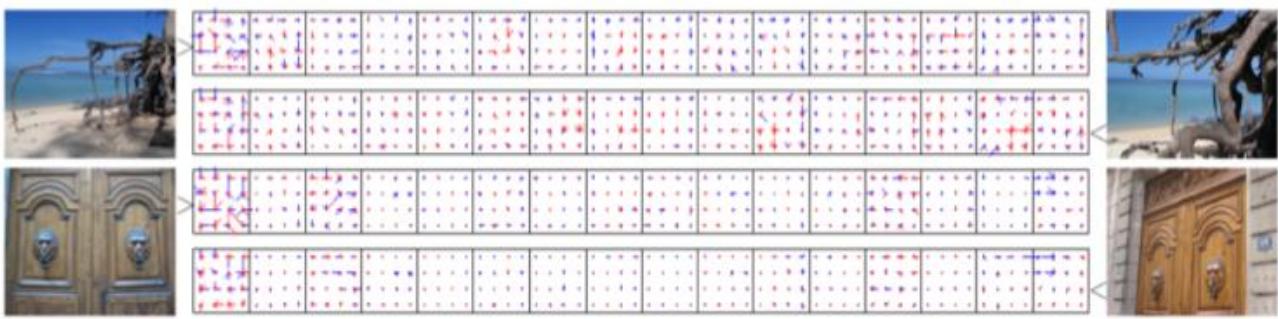
Per ogni centroidi viene mostrata una rappresentazione come SIFT, tramite questo si possono facilmente confrontare immagini.

VLADs for corresponding images



SIFT-like representation per centroid (+ components: blue, - components: red)

- good coincidence of energy & orientations



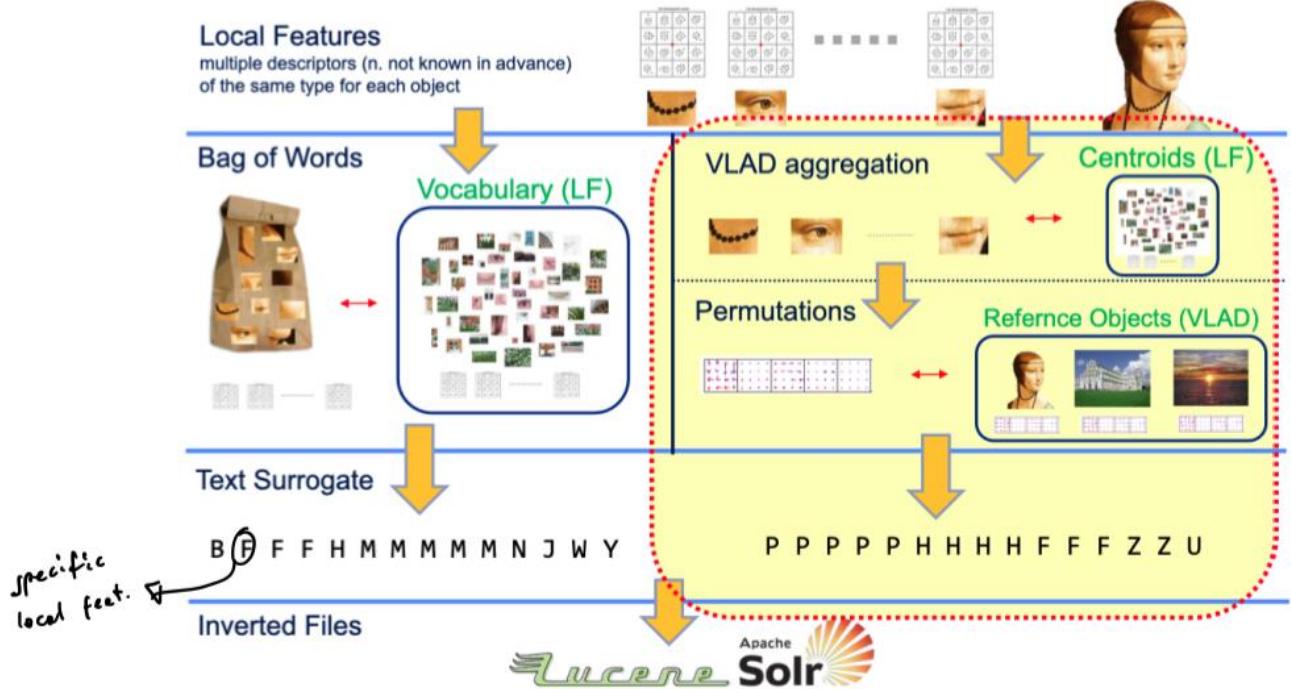
Comparison are performed using Scalar Product
Solo l'ultimo step è differente da Bof.

Compariamo VLAD con BoF (Su INRIA dataset), la dimensione è ridotta da D a D' con PCA.
 K è il vocabolario. D è la dimensione descrittore, in BoF per ogni "parola" del vocabolario abbiamo un numero che rappresenta il numero di occorrenze.
Osserviamo che:

- VLAD è meglio di BoF per via della dimensione del descrittore
- Scegli un D piccolo se la dimensione di output D' è piccola

Aggregator	k	D	D=D (no reduction)	D'=128	D'=64
BoF	1,000	1,000	41.4	44.4	43.4
BoF	20,000	20,000	44.6	45.2	44.5
BoF	200,000	200,000	54.9	43.2	41.6
VLAD	16	2,048	49.6	49.5	49.4
VLAD	64	8,192	52.6	51.0	47.7
VLAD	256	32,768	57.5	50.8	47.6

We can use a text representation for BoF and this is possible also for VLAD.



Fisher Vector

Sample (local features) are generated by a Gaussian Mixture Model (GMM)

A single Gaussian

$$u_k(x) = \frac{1}{(2\pi)^{D/2} |\Sigma_k|^{1/2}} \exp \left\{ -\frac{1}{2} (x - \mu_k)' \Sigma_k^{-1} (x - \mu_k) \right\}$$

The Mixture

$$u_\lambda = \sum_{i=1}^K w_i u_i(x)$$

The parameter

$$\lambda = \{w_i, \mu_i, \Sigma_i, i = 1 \dots K\}$$

Number of visual word

mixture weight, mean vector and covariance matrix

Mixture significa che non abbiamo solo un concetto visuale ma ne abbiamo k.

Per ogni Gaussiana abbiamo 3 parametri,

Fisher Vector misura la differenza tra due immagini misurando come i parametri che descrivono Gaussiana dovrebbero essere adattati per l'immagine specifica.

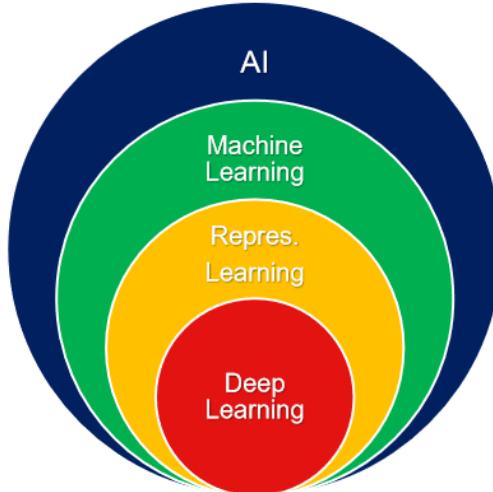
La frequenza di una parola nel BoW viene rappresentata, nel mixture model, come i pesi della singola gaussiana.

VLAD non valuta i pesi ma cerca di valutare lo spostamento della media della gaussiana per l'immagine specifica.

DA COMPLETARE NON BEN CAPITO

14. Deep learning

Il deep learning è un sottoinsieme dell'AI.



Parlando di machine learning si intende il campo di studi che permette ad un computer di imparare senza che sia esplicitamente programmato, grazie a questo un computer può imparare ad esempio a giocare una partita di scacchi. In questo campo si deve rispondere alla domanda: "come si costruisce un programma che migliora automaticamente con l'esperienza?". In altre parole un programma **impara** dall'esperienza **E** rispetto ad una task **T** e rispetto ad una misura **P** se le performance del programma sulla task **T** misurata da **P** **migliora** con l'esperienza **E**.

I metodi di deep learning sono metodi di **representation learning** che permettono ad una macchina che ha come input i raw data di trovare automaticamente le descrizioni necessarie per la detection o per la classification. Deep learning sono metodi di representation learning con **livelli multipli** di rappresentazione ottenuti componendo moduli **non lineari** ognuno dei quali trasforma la rappresentazione da un livello ad una rappresentazione ad un livello superiore.

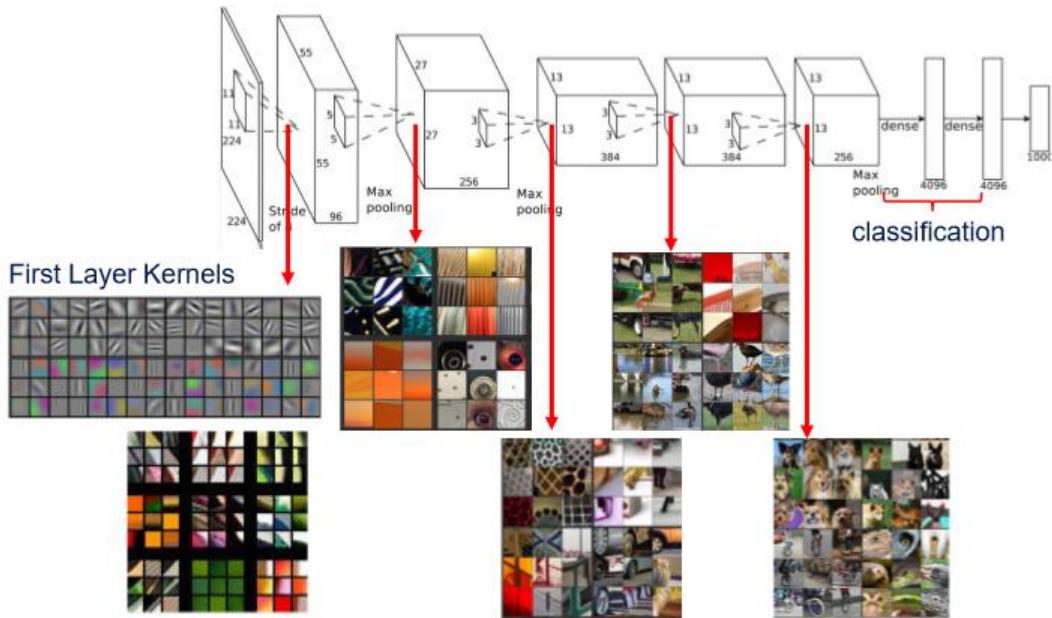
Le architetture di deep learning sono composte da molteplici livelli di operazioni non lineari come nelle reti neurali. Il sistema punta a imparare la **gerarchia delle feature** dove le feature dei livelli alti sono formate dalle feature dei livelli più bassi.

Prendiamo in esempio una rete di deep learning che ha in ingresso un'immagine, vediamo ad esempio cosa potrebbero fare i primi 3 layer:

- Il primo layer solitamente si occupa di trovare la presenza o meno di edge con particolari orientamenti e in particolari locazioni dell'immagine
- Il secondo layer solitamente trova i **motifs** cercando particolari assetti degli edge indipendentemente da piccole variazioni nella posizione di essi
- Il terzo layer solitamente raggruppa i motifs in combinazioni più ampie che corrispondono a parti di oggetti familiari. I layer successivi combineranno questi gruppi per riconoscere gli oggetti interi.

L'aspetto fondamentale del deep learning è che i layer di feature non sono programmati da umani, ma sono generati dai dati usando una procedura di learning **general-purpose**.

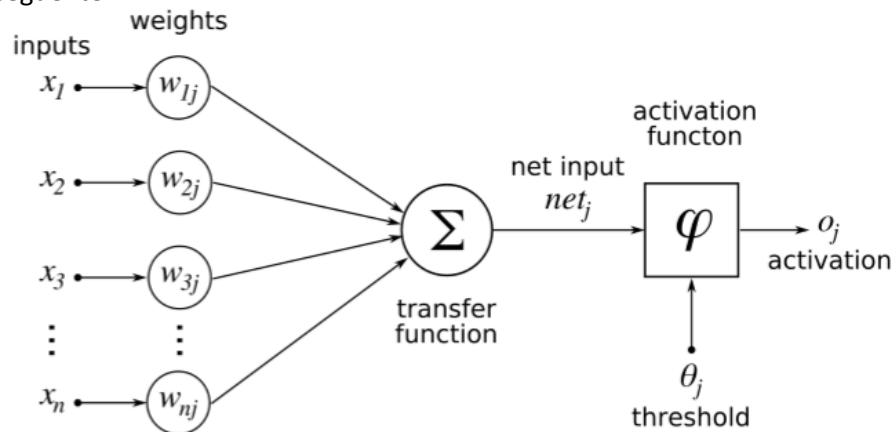
AlexNet, 2012, Trained on a Classification task of 1,000 classes.



Nell'immagine è mostrato un sistema di deep learning che ha come input un'immagine "spessa" 3 che rappresenta i tre livelli di RGB.

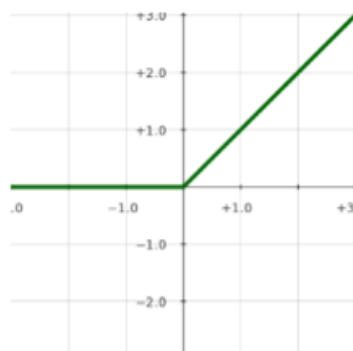
Building blocks

Il componente principale di una rete di deep learning è il neurone. Matematicamente è definito come nell'immagine seguente.



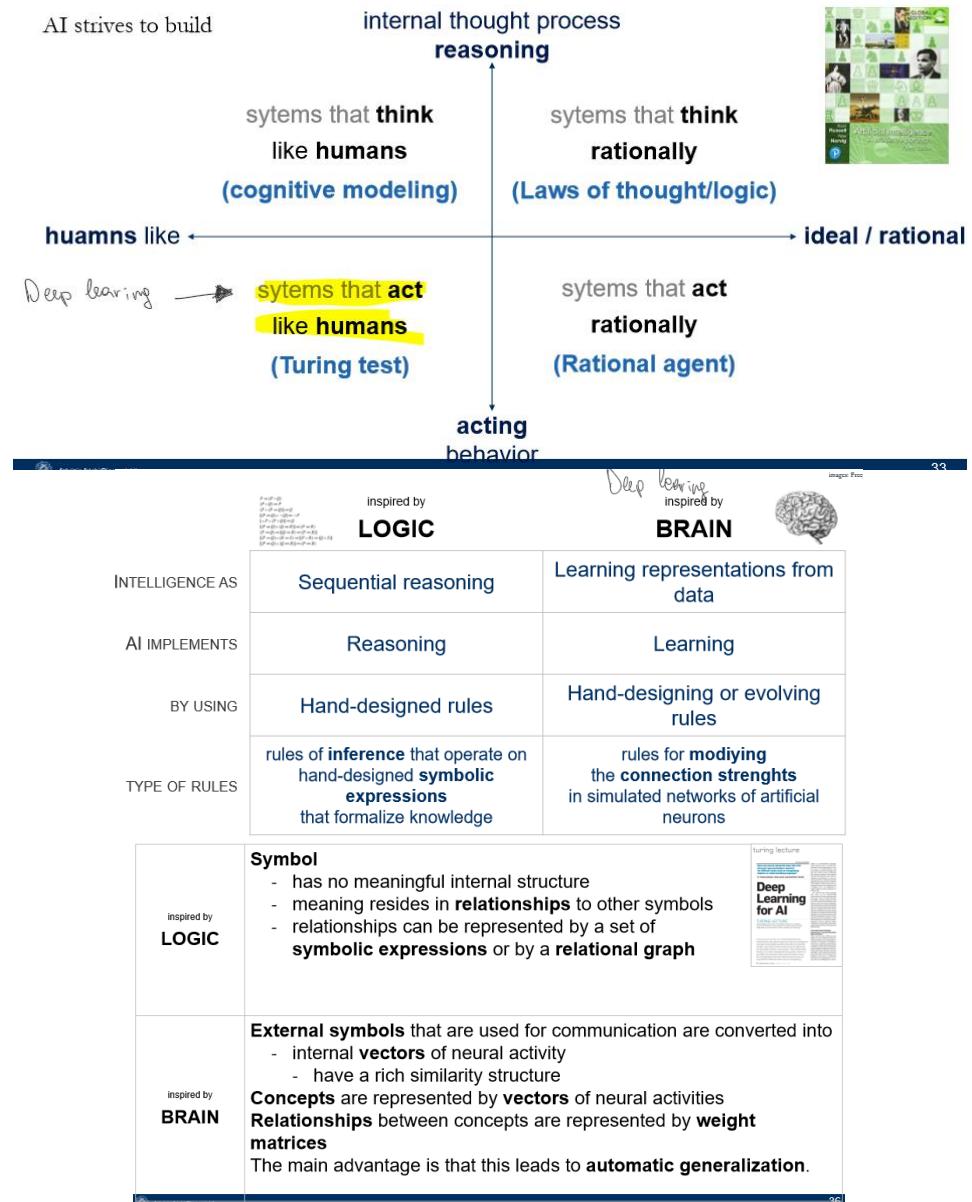
Esistono diverse funzioni di attivazione che vengono utilizzate ma la più comune è la seguente:

$$f(x) = \max(0, x)$$



Questa funzione è la più usata perché rende il train molto veloce. Se la somma degli input risulta negativa allora la mando in output, se la somma degli input è negativa allora non avrà nulla in output.

Il deep learning può trovare strutture intricate in grandi dataset usando l'algoritmo di **backpropagation**, esso indica di quanto la macchina dovrebbe variare i propri parametri interni per ottenere la risposta voluta, questa tipologia è chiamata **supervised learning**. (riguarda)



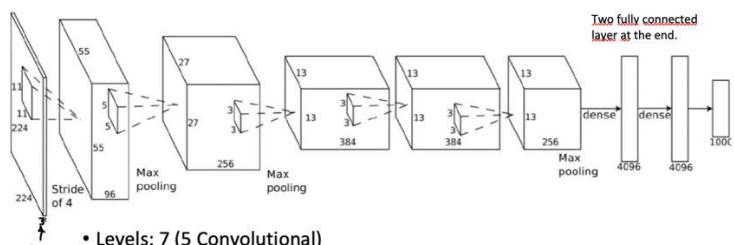
15. Convolutional Neural Networks

Le **convolutional network** sono semplicemente reti neurali che usano la convoluzione, invece di una generica moltiplicazione di matrici, in almeno uno dei suoi layer.

Usando la convoluzione andiamo a migliorare il sistema di machine learning, questo perché portano tre importanti idee:

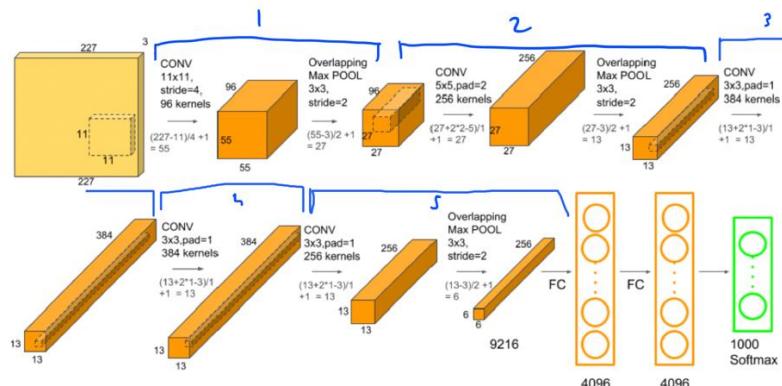
- **Sparse interactions:** Il kernel della convoluzione è più piccolo dell'input, non abbiamo quindi reti neurali fully connected ma sono sparse.
- **Parameter sharing:** vengono usati stessi parametri per più di una funzione all'interno di un modello, cioè per alcuni neuroni i pesi devono essere per forza gli stessi.
- **Equivariant representation:** il livello di convoluzione è equivalente alla traslazione. Ciò significa che se cambio la posizione di una certa regione all'interno dell'immagine la rete neurale lo riconosce senza alcun problema, si dice appunto *Translational Equivariance*.

Esempio di convolutional neural network: **AlexNet** (2012)

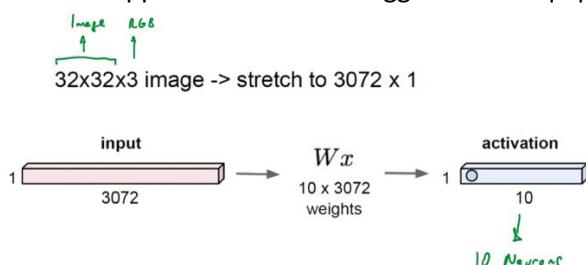


- Levels: 7 (5 Convolutional)
- Convolutional filters (kernels) per layer:
96, 256, 384, 384, 256
- Neurons: 650,000
- Parameters (to be learned): 60,000,000

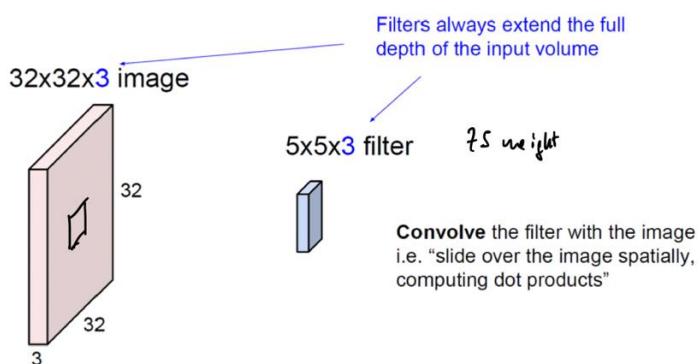
Un'altra prospettiva di AlexNet, notare le 5 convoluzioni segnate.

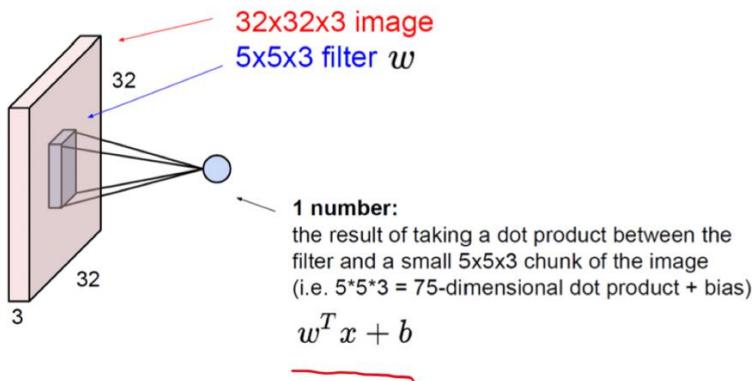


Esempio di **fully connected layer**: l'immagine viene messa su un vettore di profondità 1 ed ogni input viene collegato ad ogni neurone. Quindi un neurone ha 3072 input e tira fuori un output. L'output da 10 viene chiamato rappresentazione dell'oggetto ed è equiparabile ad una feature.

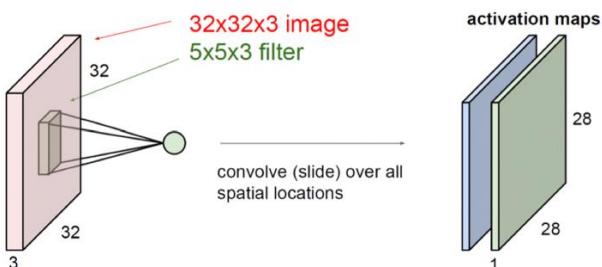
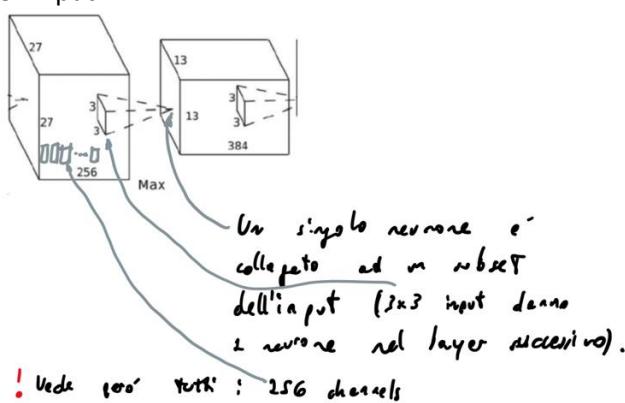


Convolutional layer





Nell'esempio sopra riportato abbiamo un'immagine 32x32 espressa in RGB (appunto profondità 3). I neuroni sono connessi con un subset di input (finestra 5x5) ma sono connessi a tutti i channel (RGB) dell'input.



Qua sopra evidenziamo che **possiamo avere differenti neuroni connessi alla stessa window!!** (Per ogni finestra ci sarà un neurone dell'activation map blu e uno per quella verde).

Cambiando finestra abbiamo altri neuroni ma con gli stessi pesi rispetto alle altre finestre.

L'input viene preso a finestre, ad ognuna di esse vengono applicati filtri differenti (comunque ogni filtro ha una dimensione uguale al numero di channels dell'input) tramite neuroni differenti, per andare a creare "pezzi" di activation map differenti. Quando la finestra viene spostata i filtri applicati sono gli stessi. Ovviamente neuroni diversi hanno pesi diversi, altrimenti l'output (activation map) sarà la stessa.

Zero Padding

Applicando lo zero padding andiamo ad aggiungere righe e/o colonne all'input, in tal modo possiamo come nell'esempio sotto ottenere un output grande quando l'input: senza zero padding abbiamo un output 5x5 mentre con diventa 7x7 (usando un filtro 3x3).

0	0	0	0	0	0		
0							
0							
0							
0							

e.g. input 7x7
 3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

Altri modi per vedere la convoluzione di cui stiamo parlando:

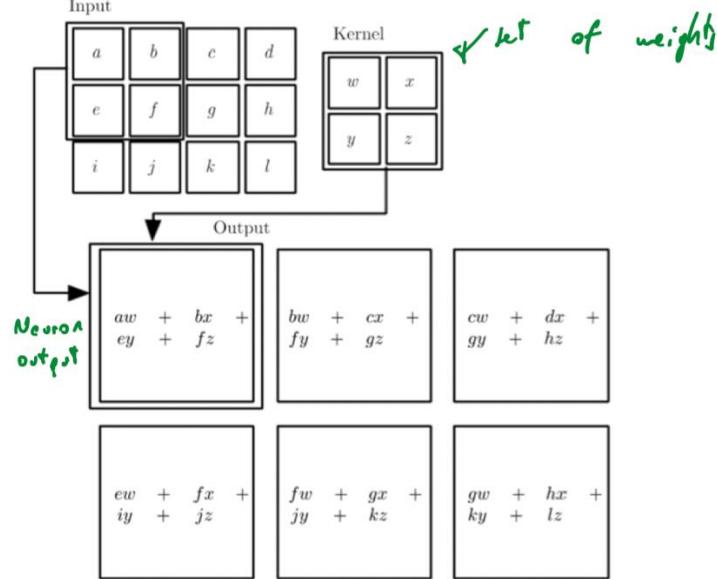
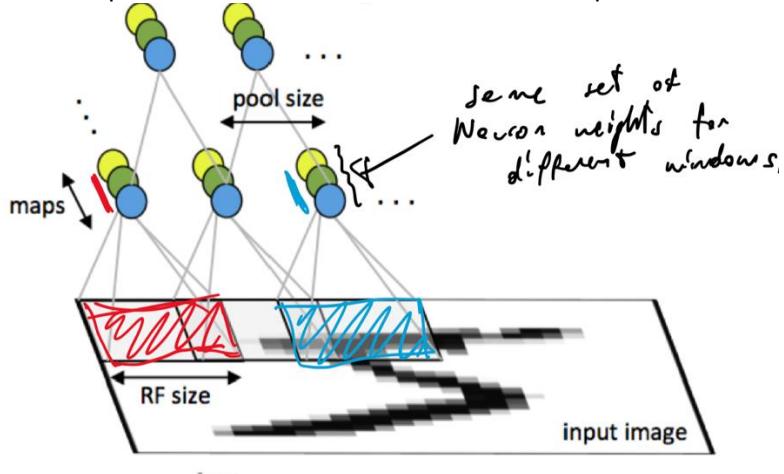
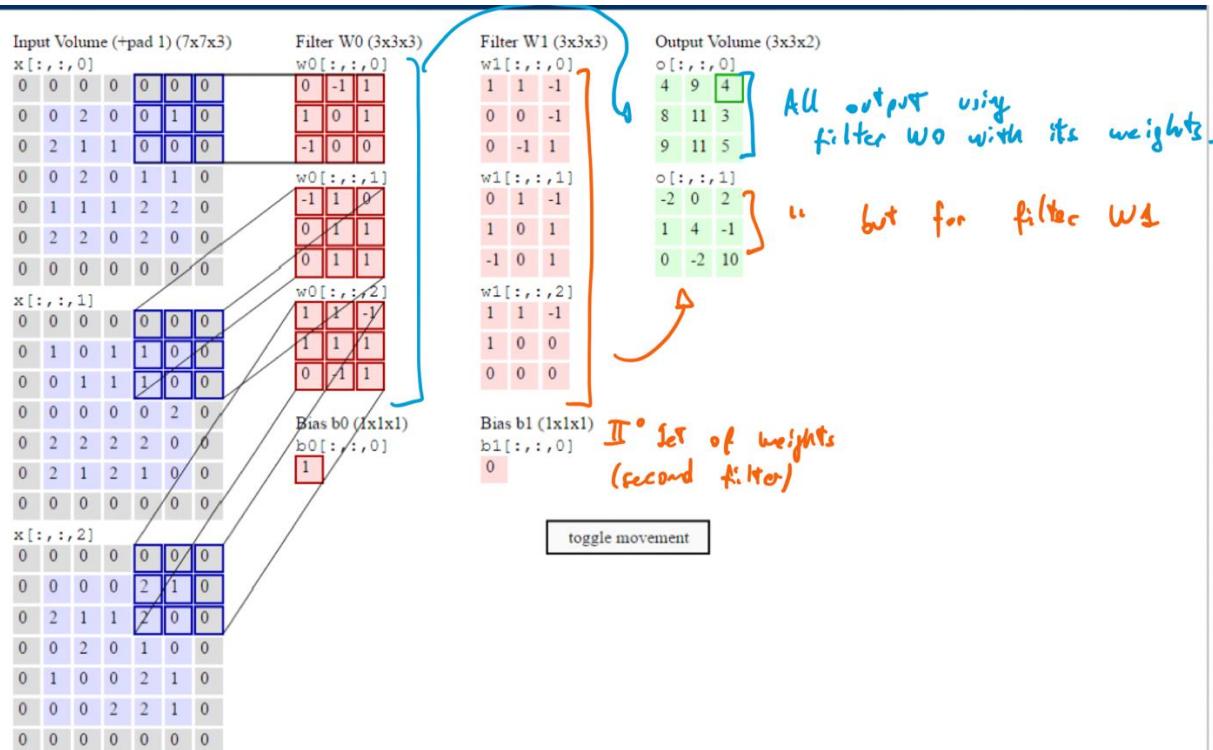
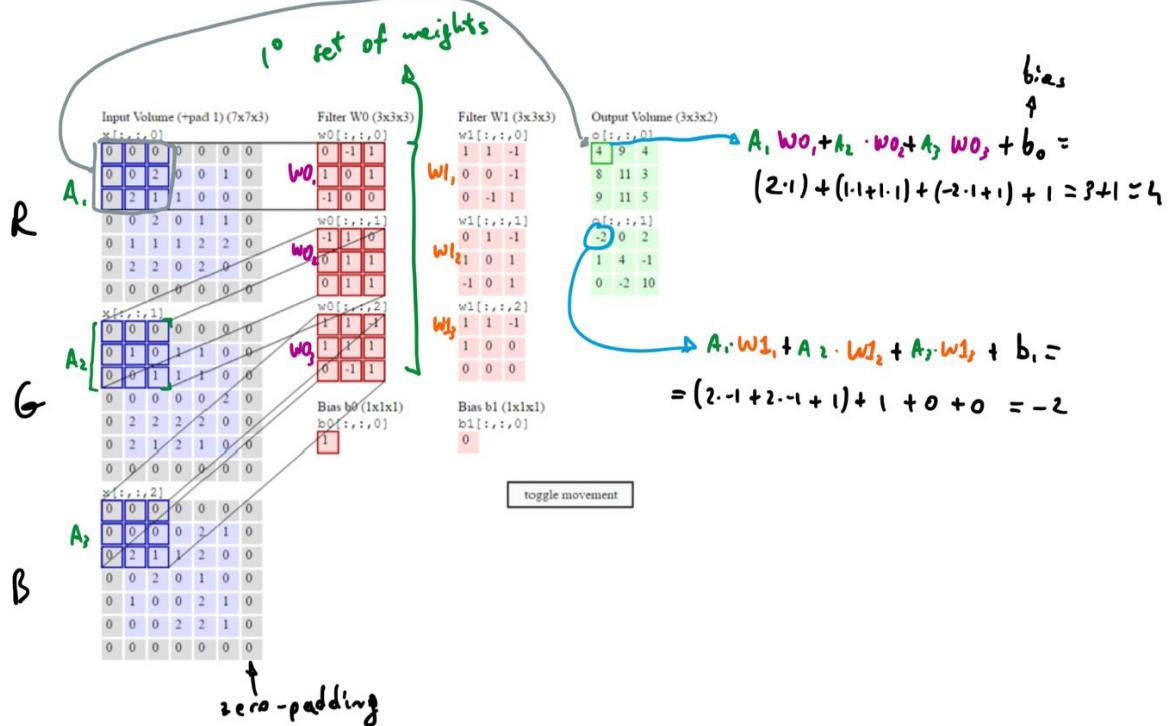
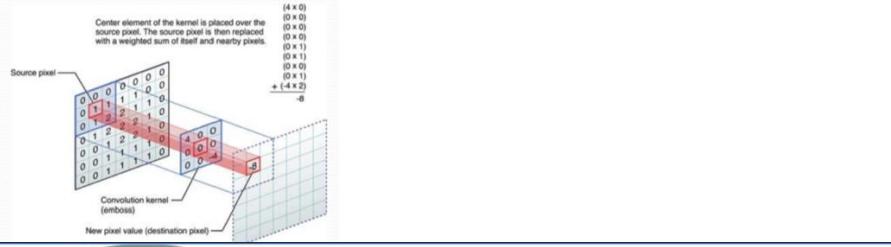


Figure 9.1: An example of 2-D convolution without kernel-flipping. In this case we restrict the output to only positions where the kernel lies entirely within the image, called “valid” convolution in some contexts. We draw boxes with arrows to indicate how the upper-left element of the output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor.

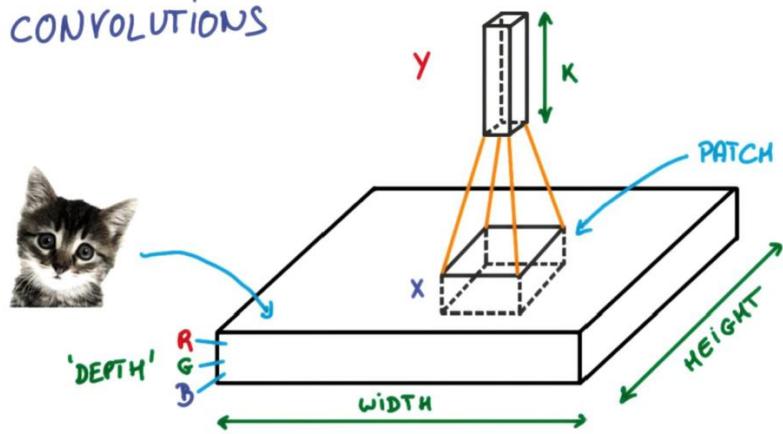
- Discrete: $f \otimes g[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$

- Continuous: $f \otimes g(n) = \int_{-\infty}^{\infty} f(m)g(n-m)dm$

- Image processing:



CONVOLUTIONS



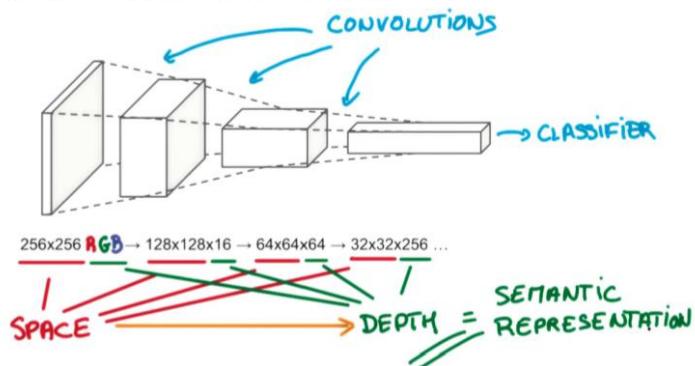
There are K neurons!!!

k = output depth

patch or kernel

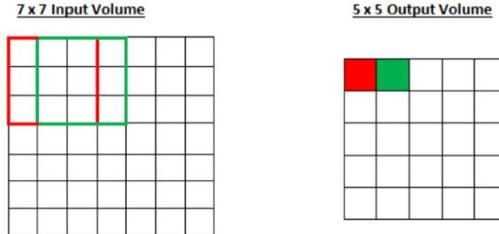
the input depth represent the feature maps

- From the input to the output
 - Space is squeezed
 - Depth increase \rightarrow Semantic Representation is richer

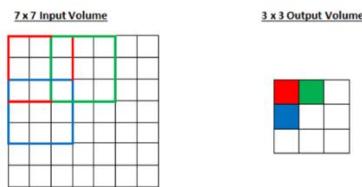


Stride

- Stride controls how the filter convolves around the input volume.
- In the example we had in part 1, the filter convolves around the input volume by shifting one unit at a time. The amount by which the filter shifts is the stride. In that case, the stride was implicitly set at 1. Stride is normally set in a way so that the output volume is an integer and not a fraction. Let's look at an example. Let's imagine a 7×7 input volume, a 3×3 filter (Disregard the 3rd dimension for simplicity), and a stride of 1. This is the case that we're accustomed to

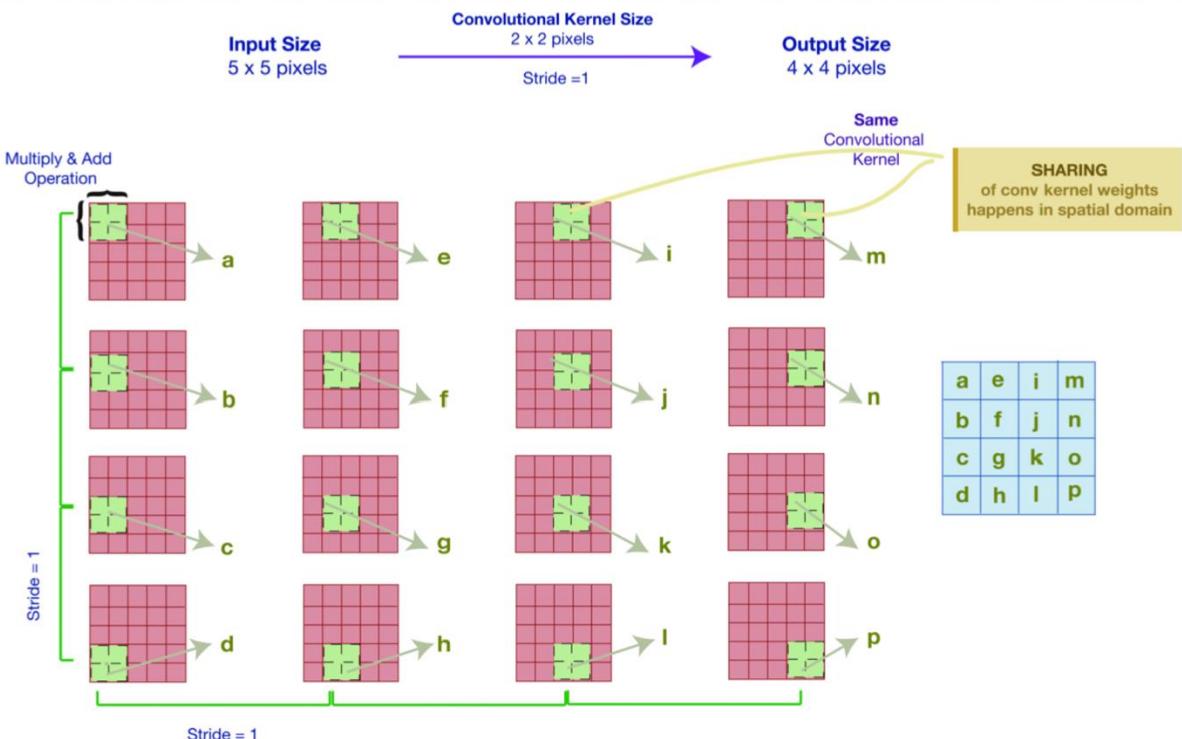


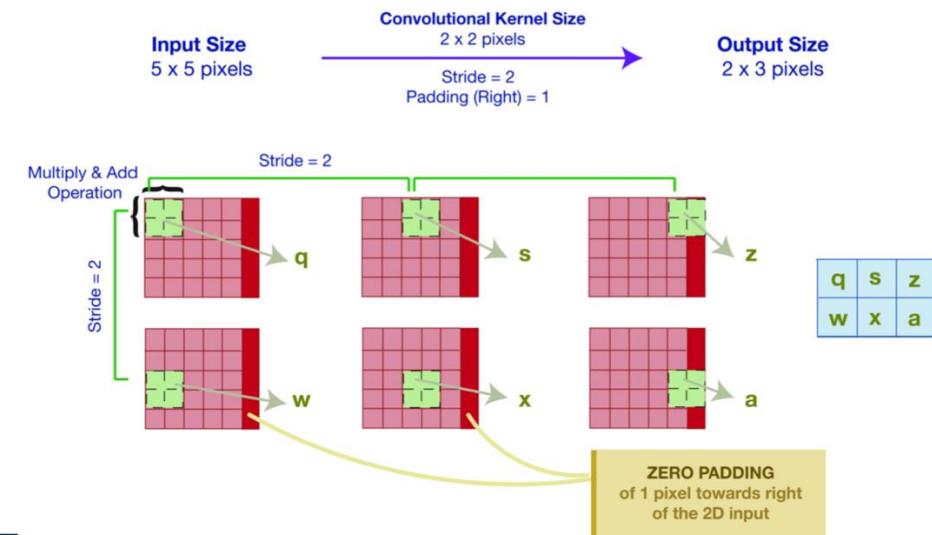
- Same old, same old, right? See if you can try to guess what will happen to the output volume as the stride increases to 2.



- So, as you can see, the receptive field is shifting by 2 units now and the output volume shrinks as well. Notice that if we tried to set our stride to 3, then we'd have issues with spacing and making sure the receptive fields fit on the input volume. Normally, programmers will increase the stride if they want receptive fields to overlap less and if they want smaller spatial dimensions.

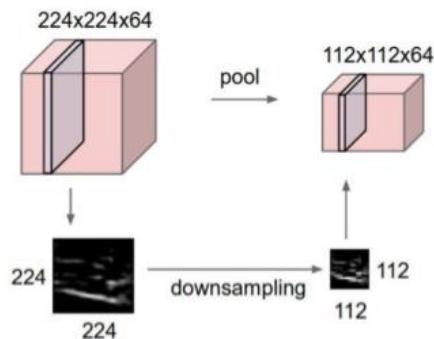
If we jump with more than 1 square we can significantly reduce the volume of output.





Pooling

Tramite pooling andiamo a ridurre la dimensione della matrice, mantenendo però invariato il numero di channel, il pooling opera su ogni activation map indipendentemente dalle altre.



Da un input di dimensione $W_1 \times H_1 \times D_1$, e due hyperparameters:

- Spatial extent F (solitamente 2 o 3), che sarebbe la grandezza della finestra
- Stride S

E produce un output di dimensione $W_2 \times H_2 \times D_2$ dove:

$$W_2 = (W_1 - F)/S + 1$$

$$H_2 = (H_1 - F)/S + 1$$

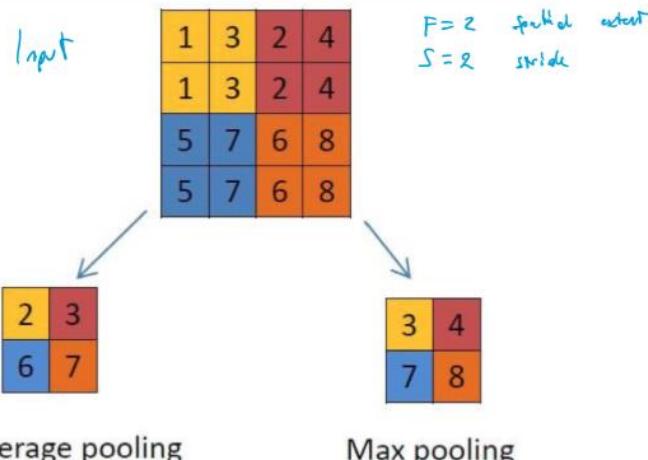
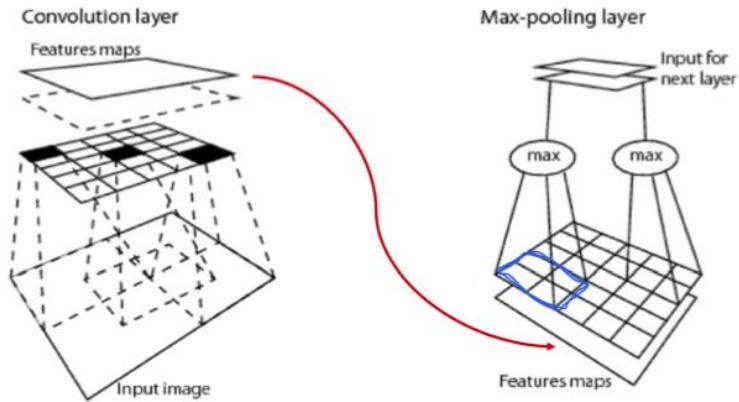
$$D_2 = D_1$$

Non sono necessari parametri dato che viene calcolata una funzione prefissata dell'input, solitamente non viene usato lo zero padding per i pooling layers.

Il pooling è più preciso rispetto ad una convoluzione perché usando quest'ultima perdo dei dati significativi (nota ad esempio che per ottenere uno stesso output il pooling usa stride=1 e la conv stride=2, ma è ovviamente anche più dispendioso perché vengono fatte più convoluzioni).

Una funzione di pooling **rimpiazza** l'output della rete ad un certo livello, sostituendolo con **una somma statistica degli output vicini**.

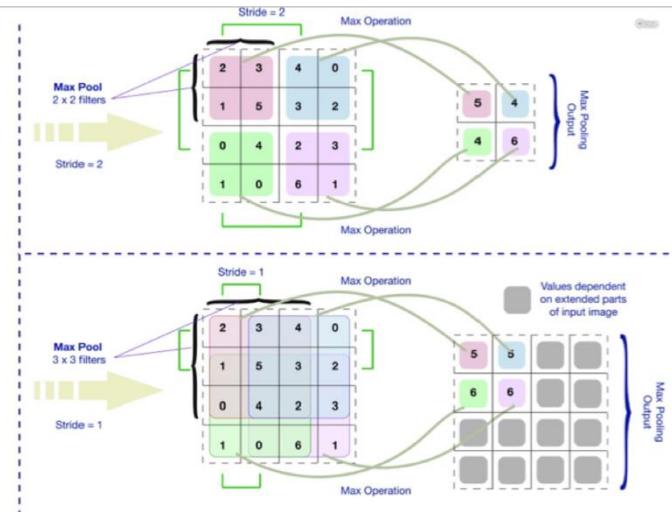
Il più utilizzato è il **max pooling** dove andiamo semplicemente ad inserire il massimo tra gli output vicini (all'interno della finestra). L'**average pooling** inserisce invece la media dei valori all'interno della finestra. E' importante ricordare che il pooling viene fatto **per ogni channel dell'input**.



Complete Pooling Examples

Please note:

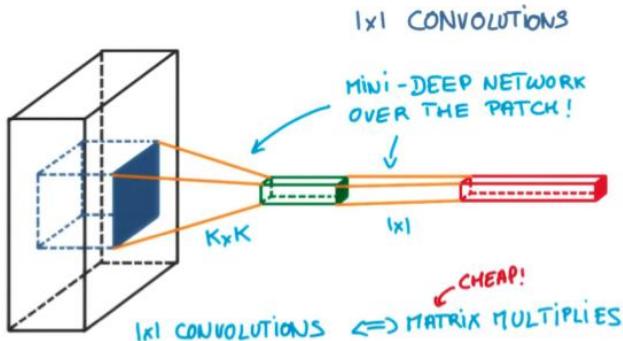
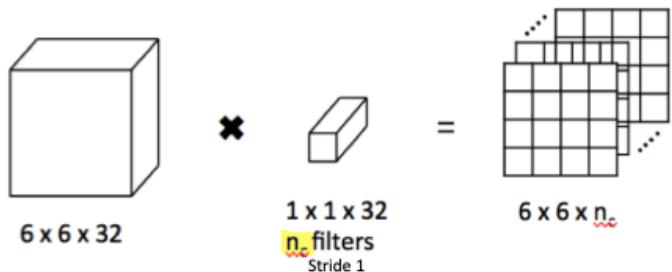
- the number in the input matrix are typically the result of a convolution
- the stride of the pooling is not the convolution stride



1x1 Convolution (Network-in-Network)

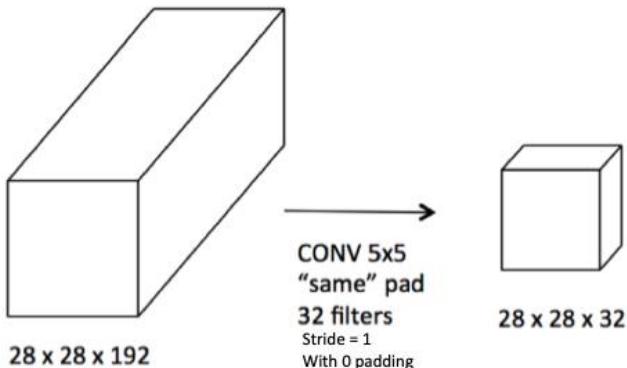
Le convoluzioni 1x1 servono per **cambiare la profondità**, unendo channel, lasciando invariate le dimensioni della matrice di input (H e W).

Dall'immagine sotto si può capire il funzionamento, abbiamo un input di $6 \times 6 \times 32$ e n_c filtri di dimensione $1 \times 1 \times 32$ che produrranno un output $6 \times 6 \times n_c$ dove n_c è il numero di channel. In parole povere, ogni filtro viene applicato su tutto l'input per creare un channel in uscita, ovviamente i pesi saranno i medesimi e saranno diversi dai pesi degli altri $n_c - 1$ filtri.

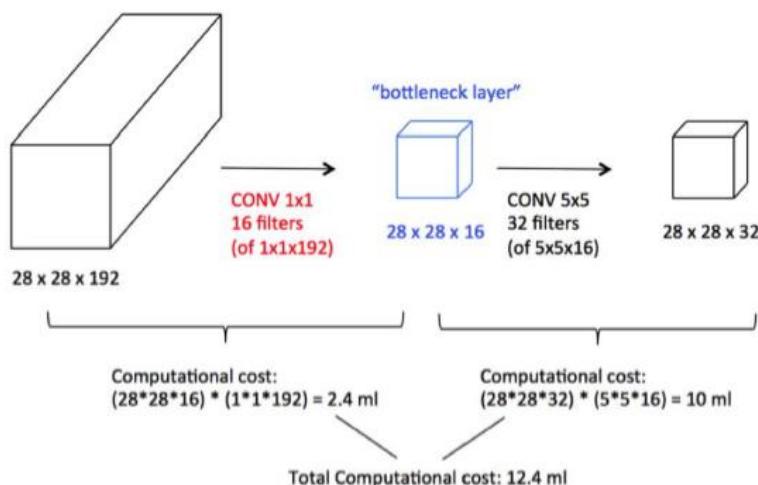


Da sotto vediamo come è **conveniente**, in termini di numero di moltiplicazioni, usare una convoluzione 1x1 quando si vuole cambiare profondità dell'input.

$$(28 \times 28 \times 32) * (5 \times 5 \times 192) = 120 \text{ million multiplications}$$



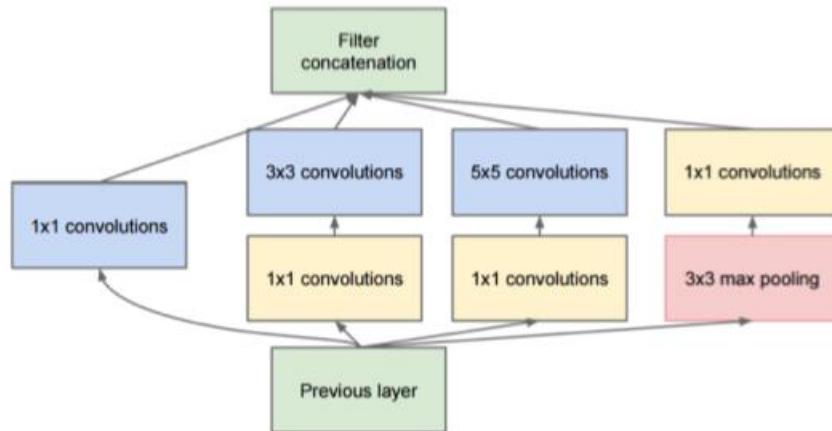
$$(28 \times 28 \times 16) * (1 \times 1 \times 192) + (28 \times 28 \times 32) * (5 \times 5 \times 16) = 12.4 \text{ million multiplications}$$



Domanda: quanti pesi sono necessari per la conv 5x5 del primo esempio? $5 \times 5 \times 192 * 32$
 Quanti nella conv 1x1 del secondo esempio? $1 \times 1 \times 192 * 16$

Inception

Idea alla base: quando si costruisce un layer di una ConvNet è necessario scegliere il numero di filtri e il tipo di layer (pooling, conv), tramite **l'inception** possiamo lasciar decidere cosa usare durante il learning. Possiamo ridurre la complessità di questo processo riducendo la dimensionalità utilizzando 1x1 con e il Bottleneck Layer.



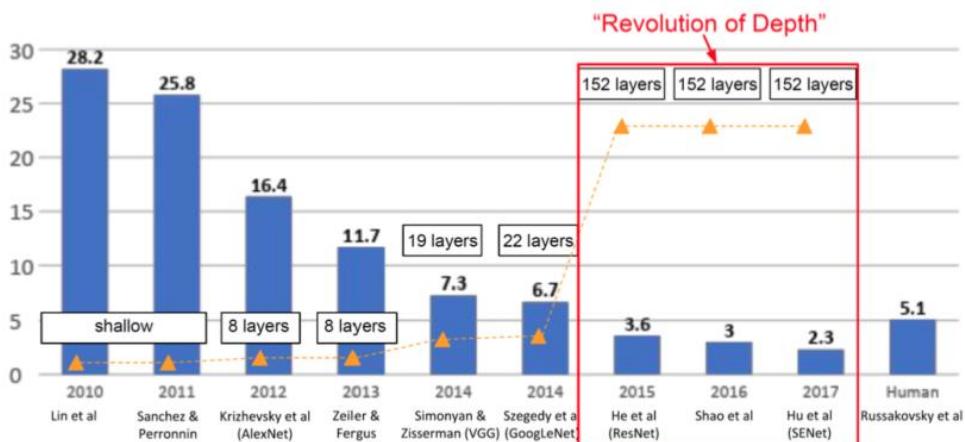
(b) Inception module with dimensionality reduction

L'inception module permette alla rete di far cambiare i pesi dei kernel in base alle performance della rete, vengono applicate le 1x1 conv per ridurre il costo computazionale.

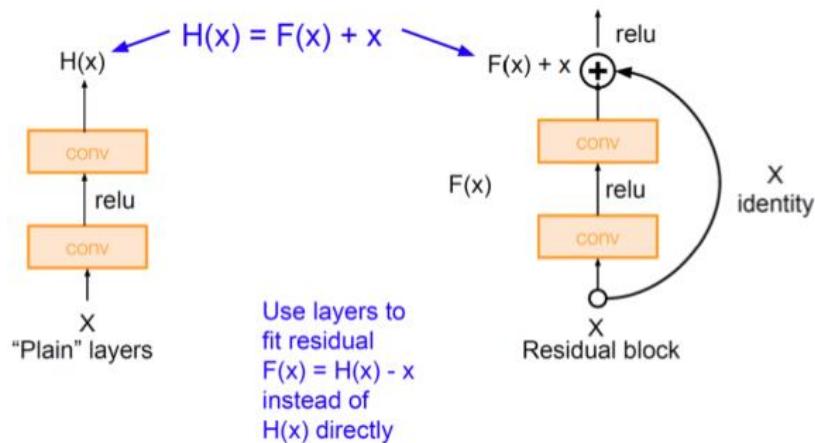
Invece di fare una rete con un'unica convoluzione, in questo modo facciamo diverse reti e poi concatenando il risultato raggiungiamo l'output, questo tipo di rete ci permette anche di andare a cambiare dinamicamente i valori dei pesi a seconda delle performance.

Residual Networks

In termini di numero di layer, una rete più profonda non è sempre meglio di una meno profonda infatti tramite i **residual network** la rete può adattarsi mandando a zero i pesi di alcuni layer e facendo passare direttamente l'input. In tal modo possiamo creare reti con tantissimi layer ma che poi, adattandosi, collasseranno in pochi a seconda dei pesi assegnati.

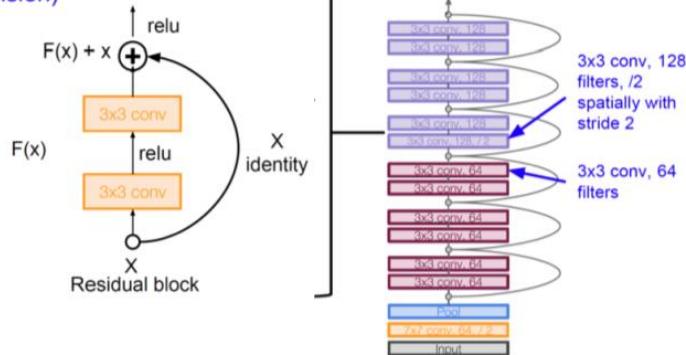


Nell'immagine sopra vediamo come il numero di layer negli ultimi anni è incrementato di molto diminuendo l'error rate (in blu), migliorando quindi la qualità generale delle reti.



Full ResNet architecture:

- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (/2 in each dimension)



on & Serena Yeung [slides](#) and [video](#)

Manca un pezzetto a fine pacchetto anche se dice poco o nulla.

17. Some deep learning related topics

Data Processing and Batch Normalization

Regularization outline:

- L2 \&L1
- Dropout
- Data Augmentation

Representations and Learning

- Face Recognition
- Triplet Loss
- Siamese Learning

Preprocessing

Per le reti neurali vorremmo avere i dati come nel terzo grafico, questo si può fare ad esempio normalizzando la varianza. (Viene messa la media nel punto 0,0)

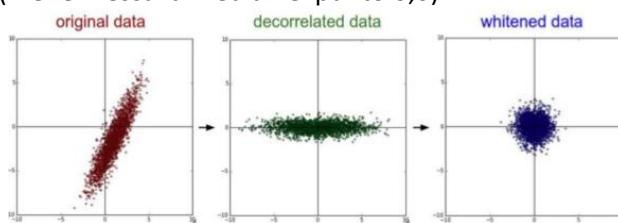


Image preprocessing

Subtract per-channel mean (VGGNet): calcoliamo la media per canale e poi per ogni pixel ce la sottraiamo

Subtract the mean image (Alex net): calcoliamo la media sull'immagine intera e la sottraggio ad ogni pixel.

Non è comune per le immagini normalizzare la varianza, applicare il PCA e fare whitening.

Batch normalization (Normalizzazione per reti neurali)

Dato normalizziamo l'input del primo layer, vorremmo anche normalizzare l'output dei layer successivi, questo viene fatto con la batch normalization, nella quale possiamo normalizzare per mini-batch mean, variance, etc..

Essenzialmente BM fa whitening nei layer intermedi della rete.

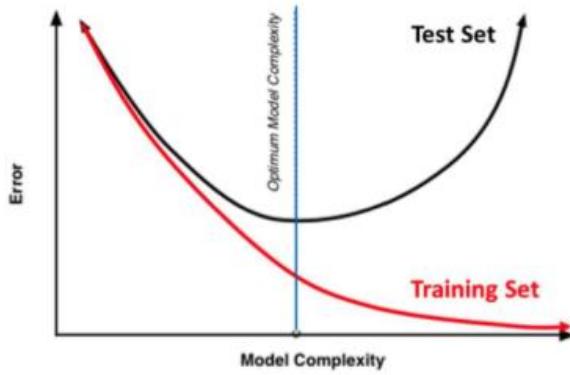
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;	
Parameters to be learned: γ, β	
Output: $\{y_i = BN_{\gamma, \beta}(x_i)\}$	
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$ // mini-batch mean	
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$ // mini-batch variance	
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$ // normalize	
$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i)$ // scale and shift	

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

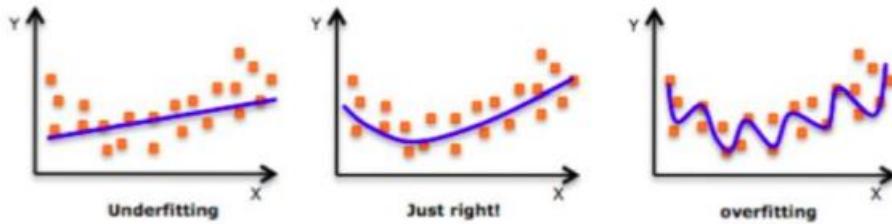
Regularization

All'aumentare della complessità del modello, andiamo ad aumentare l'errore, questo è dovuto al fenomeno dell'**overfitting**, nel quale andiamo ad essere troppo specifici portando a conseguenti errori.

Training Vs. Test Set Error



Tramite la tecnica di **regularization** si fanno delle modifiche all'algoritmo di learning, ottenendo un modello che generalizza al punto giusto come fatto sotto. In tal modo andiamo a migliorare le performance del modello anche su dati che ancora non abbiamo visto.



Regularization: L2 & L1

Le due tecniche più conosciute nel Deep Learning per la regularization sono L1 e L2, entrambe vanno ad aggiungere una penalità al costo in base alla complessità del modello, quindi, invece di calcolare il costo solo con una loss function andiamo ad aggiungere un altro termine chiamato **regularization term** che **penalizza appunto la complessità del modello**.

Per **loss function** si intende la misura dell'errore rispetto al risultato atteso (y).

Quindi pesi grandi portano ad aumentare la complessità del modello perché un errore su di essi porta ad un grande cambiamento dell'output.

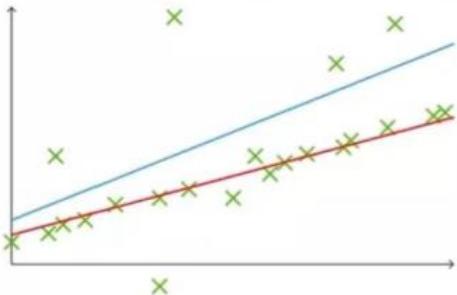
$$\begin{aligned} \text{L1 Regularization} \\ \text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j| \\ \text{L2 Regularization} \\ \text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M W_j^2 \end{aligned}$$

Loss function Regularization
 Term
 Or Penalization term

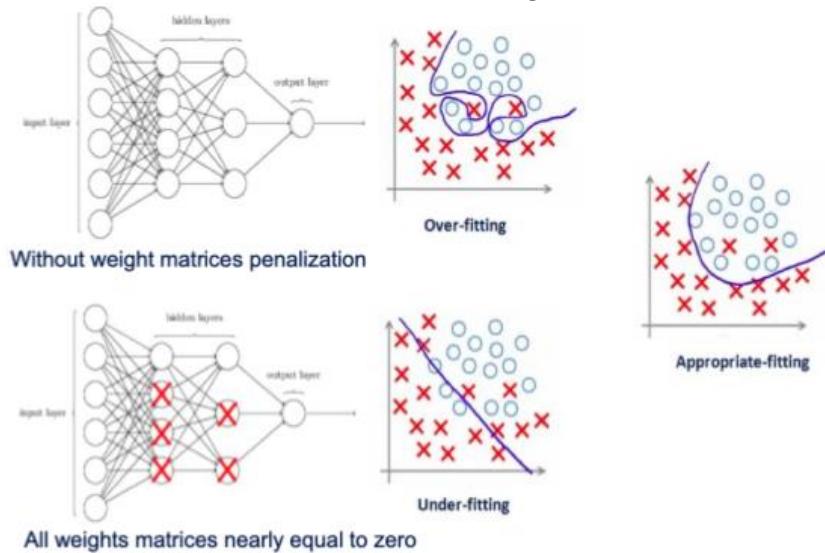
Dal punto di vista teorico: L2 enfatizza l'errore dovuto ai pesi, infatti, W viene messo al quadrato, e cerca di minimizzare gli errori equamente quindi la linea sarà un po' spostata dal trend perché grandi errori influenzano di più di piccoli.

D'altra parte, per L1 ogni errore ha la stessa importanza, quindi, minimizzerà un sacco di errori andando molto vicino al trend principale, anche in presenza di outlier.

Given : A set of points in 2-dimension
 Goal : Find a line to fit those points
 Output : ℓ_2 minimizer line, ℓ_1 minimizer line



Vediamo appunto che se non facciamo penalizzazione sui pesi andiamo in over-fitting, mentre, se posiamo vicino a 0 tutti i pesi delle matrici allora andiamo in under-fitting.



Regularization: Dropout

Reti neurali profonde e composte da un grande numero di parametri sono degli ottimi strumenti per il machine learning, tuttavia **l'overfitting** è un importante problema, inoltre abbiamo anche lo svantaggio che, essendo lente, è difficile ovviare all'overfitting combinando le predizioni di tante e grandi reti neurali.

Il **dropout** è una tecnica utilizzata per risolvere questo problema, l'idea è quella di **eliminare randomicamente nodi (e i suoi collegamenti)** durante il training.

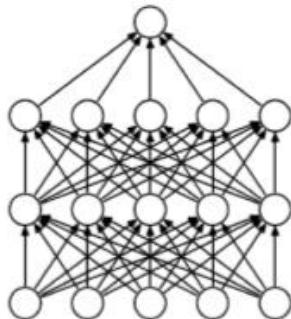
Facendo questo la rete impara a dare il risultato corretto anche con vari sottoinsiemi di reti più piccole, questo porta a rendere la rete più **robusta** perché, essendo presenti differenti e ridondanti rappresentazioni, il risultato della rete non si baserà su collegamenti specifici ma sarà dato da più parti della rete (come se dovesse raggiungere un **consenso** per dare il giusto output). Inoltre, ricordiamo che è importantissimo per **ridurre l'overfitting**, per via dei motivi appena spiegati.



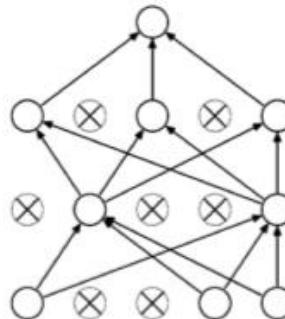
REDUNDANT REPRESENTATIONS



 → 'IT'S A CAT'
 → 'I AGREE'
 → 'ME TOO!'
 → 'CLEARLY NOT A DOG...'



(a) Standard Neural Net



(b) After applying dropout.

: Dropout Neural Net Model. Left: A standard neural net with 2 hidden layers. Right: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Come detto prima andiamo a prendere il consenso dalle varie "sottoreti", il consenso è la media E tra i vari output y_t delle reti alle quali è stato applicato il dropout. Dato che andiamo ad ottenere una parte degli activation, è bene usare un moltiplicatore (ad esempio $x2$ se il dropout avviene su metà degli activation) per i risultati delle sottoreti, altrimenti il consenso sarebbe troppo basso.

Ricordiamo anche che il dropout viene fatto **durante il training!** E non nell'evaluation.

TRAINING

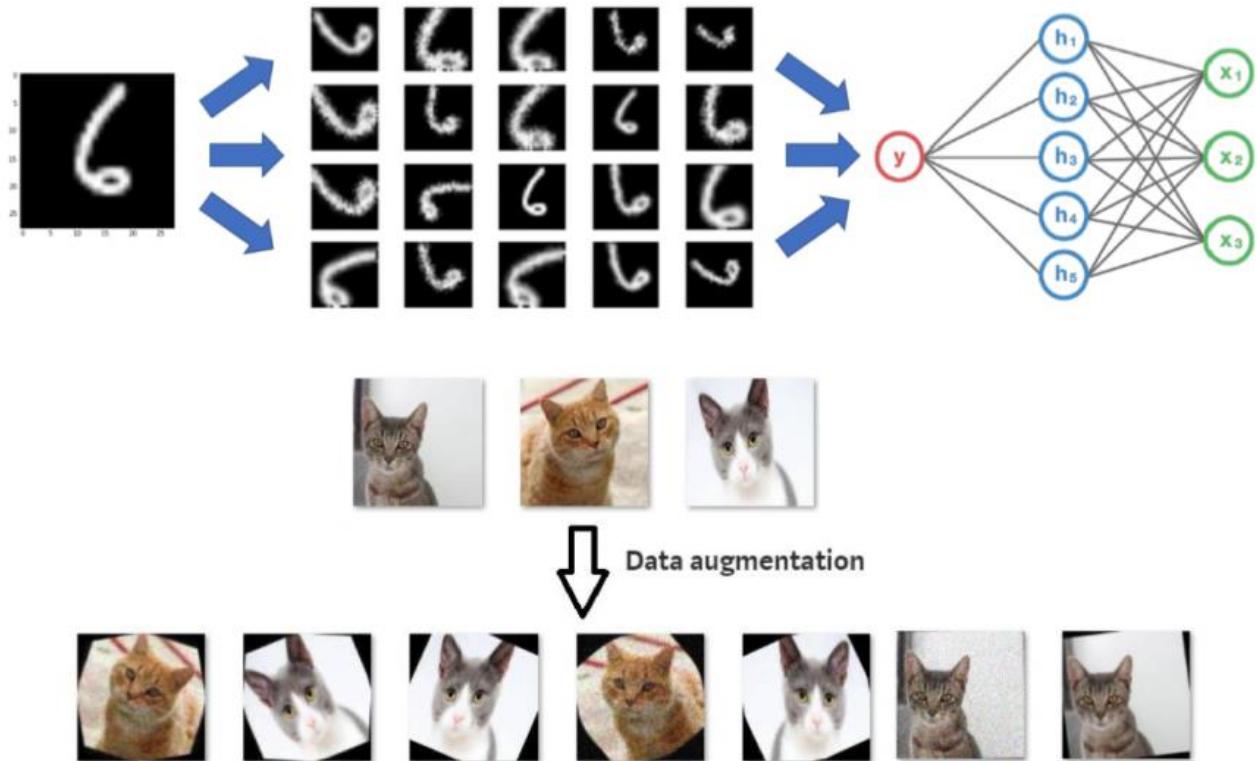
$$\begin{array}{rcl}
 1.0 & \xrightarrow{\quad X \quad} & 0.0 \\
 0.2 & \xrightarrow{\quad} & 0.2 \times 2 \\
 -0.3 & \xrightarrow{\quad X \quad} & 0.0 \\
 0.5 & \xrightarrow{\quad} & 0.5 \times 2
 \end{array}
 \left[Y_t \right]$$

EVALUATION

$$\begin{array}{rcl}
 1.0 & \xrightarrow{\quad} & 1.0 \\
 0.2 & \xrightarrow{\quad} & 0.2 \\
 -0.3 & \xrightarrow{\quad} & -0.3 \\
 0.5 & \xrightarrow{\quad} & 0.5
 \end{array}
 \left[Y_e \sim E(Y_t) \right]$$

Regularization: Data Augmentation

Prendiamo l'esempio sotto, se durante il training diamo solo la prima immagine con il 6, la rete sarà pronta a riconoscere solo un'immagine molto simile. Tramite la **data augmentation** andiamo a ruotare e scalare l'immagine creando un training set più grande e quindi la rete sarà in grado di riconoscere correttamente più input.

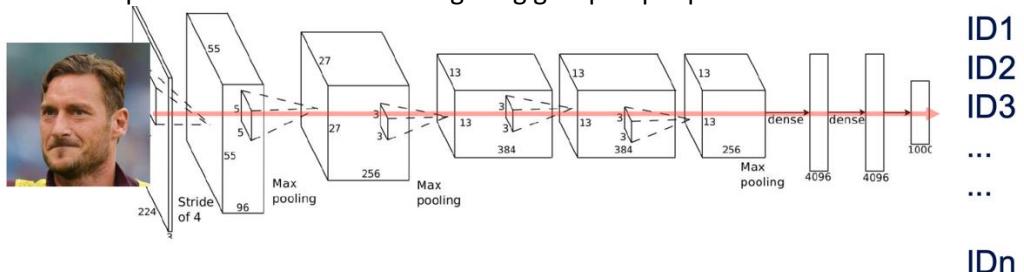


Representations and Learning

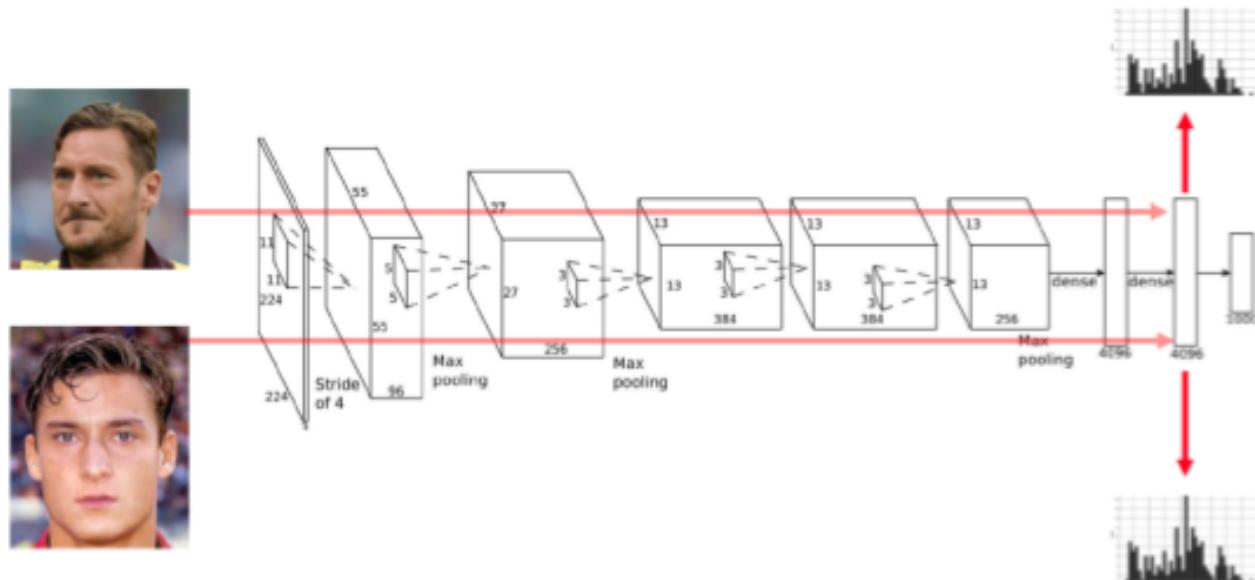
Per fare **face recognition** non possiamo trainare una rete neurale per riconoscere tutte le persone nel mondo, ma nemmeno un gruppo più grande di un migliaio di persone. Ciò che invece possiamo fare per face recognition è comparare facce sconosciute con facce conosciute, questo processo si chiama **face verification**.

Per comparare due facce usando il deep learning andiamo a comparare la loro rappresentazione negli **hidden layers**.

Face identification: impossible if we're considering a big group of people!



Abbiamo bisogno quindi di *face verification*: Andiamo a confrontare due immagini, e confrontando le rappresentazioni (descritte da features) diciamo se nelle immagini è presente la stessa persona oppure no.



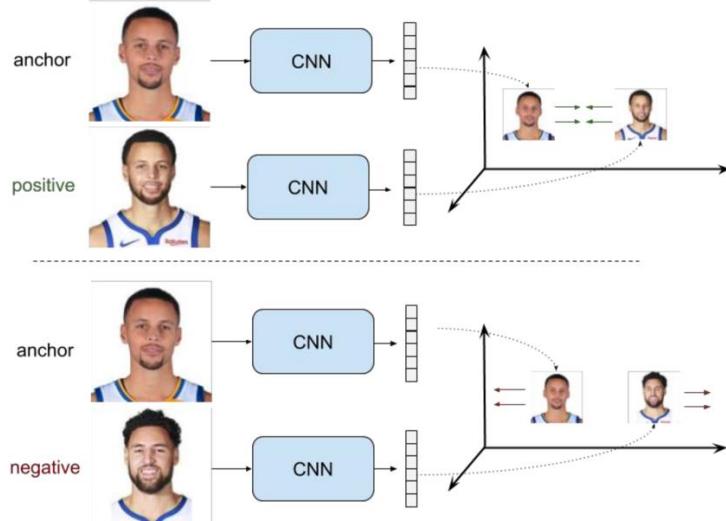
Questo possiamo farlo con il **Ranking loss** oppure con il **siamese learning**.

Ranking Loss

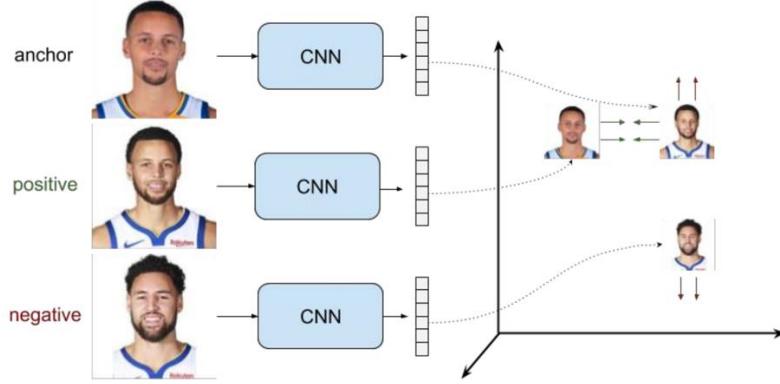
Sono presenti due tipi di Ranking Loss:

- Usando **pairs of training data points**
- Usando **triplets of training data points**

Pairs of training data points: Nello spazio tridimensionale Andiamo a mettere le rappresentazioni delle immagini. Data un anchor, se viene data un img che rappresenta la stessa persona allora forziamo le rappresentazioni ad essere simili, altrimenti ad essere più differenti.



Triplet ranking loss



Il ranking loss viene attuato con una rete in **backpropagation**: diamo in pasto a tre reti neurali con parametri condivisi le tre immagini, tiriamo fuori le rappresentazioni e andiamo a fare backpropagation per allontanare le rappresentazioni di anchor e negatives e avvicinare anchor e positives.

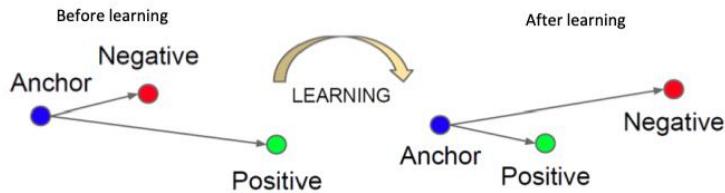
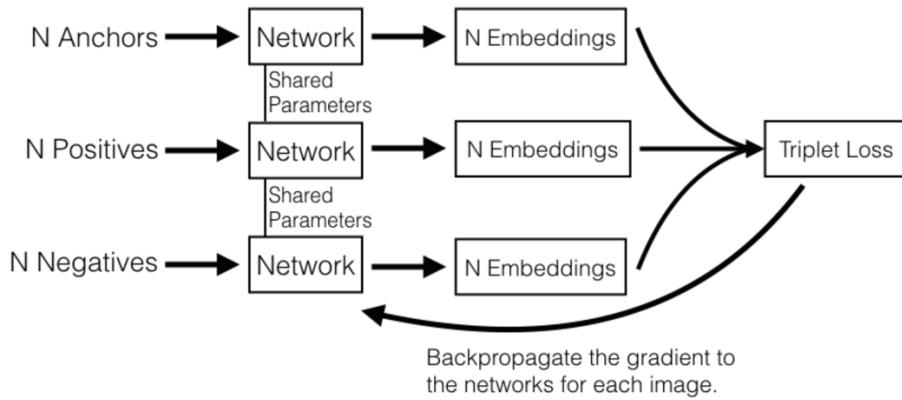


Figure 3. The **Triplet Loss** minimizes the distance between an *anchor* and a *positive*, both of which have the same identity, and maximizes the distance between the *anchor* and a *negative* of a different identity.

The loss that is being minimized is then $L =$

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

anchor positive anchor negative

$f(x_i)$ è la rappresentazione di anchor, positive o negative.

Vogliamo la prima **differenza euclidea** che sia piccola perché le rappresentazioni devono essere vicine, mentre la seconda differenza deve essere grande perché devono stare lontane.

La differenza tra le due appena descritte deve superare anche un margine alpha.

Let's analyze 3 situations of this loss:

Easy Triplets:

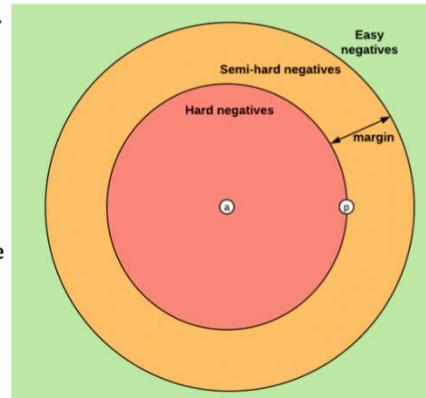
The negative sample is already sufficiently distant to the anchor sample respect to the positive sample in the embedding space. The loss is 0 and the net parameters are not updated.

Hard Triplets:

The negative sample is closer to the anchor than the positive. The loss is positive (and greater than mm).

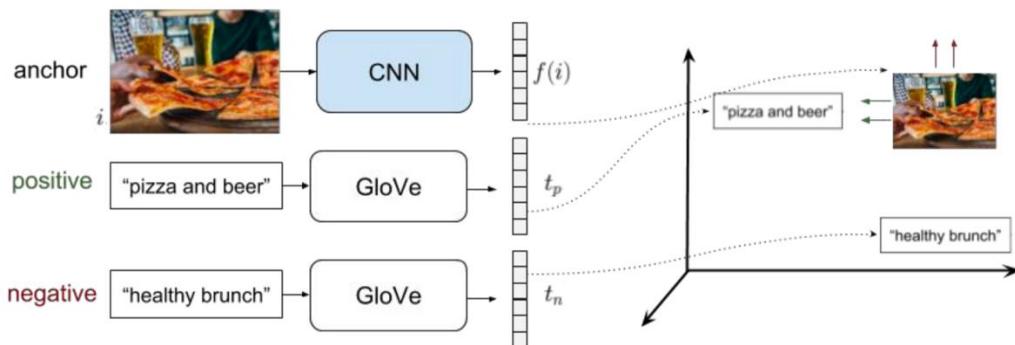
Semi-Hard Triplets:

The negative sample is more distant to the anchor than the positive, but the distance is not greater than the margin, so the loss is still positive (and smaller than mm).



Ranking loss for multi-modal retrieval

Rappresentazione Learning può essere anche **cross media** (si può comparare immagini e testi ad esempio), basta che le rappresentazioni siano comparabili, cioè devono essere della stessa dimensione.

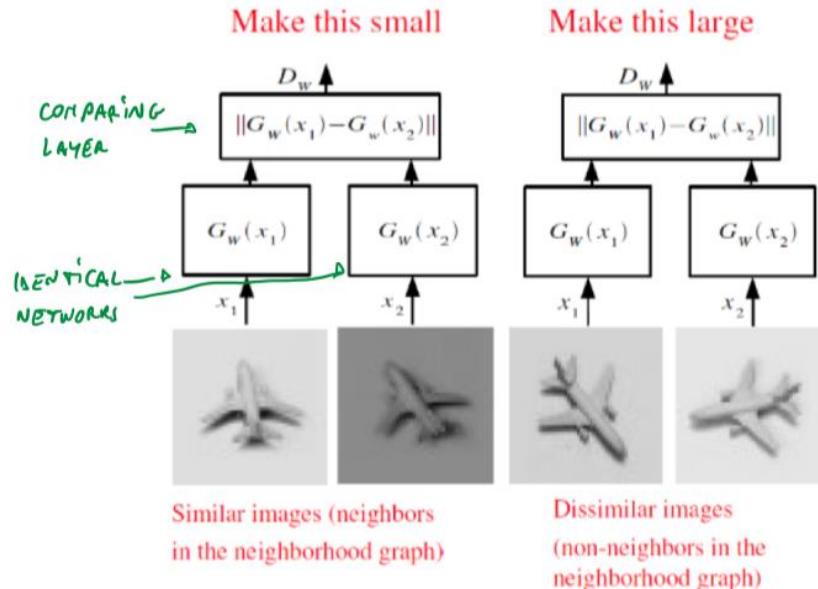


Siamese Learning

Come detto serve per verificare se due immagini contengono lo stesso oggetto oppure no.

Le siamese networks sono composte da **two-stream networks che hanno pesi identici**, ad esse vengono date due immagini da confrontare come input. Gli activations sono confrontati in un layer interno alla rete e l'**output della rete sarà un singolo numero** o metrica, che rappresenta la distanza tra le due immagini.

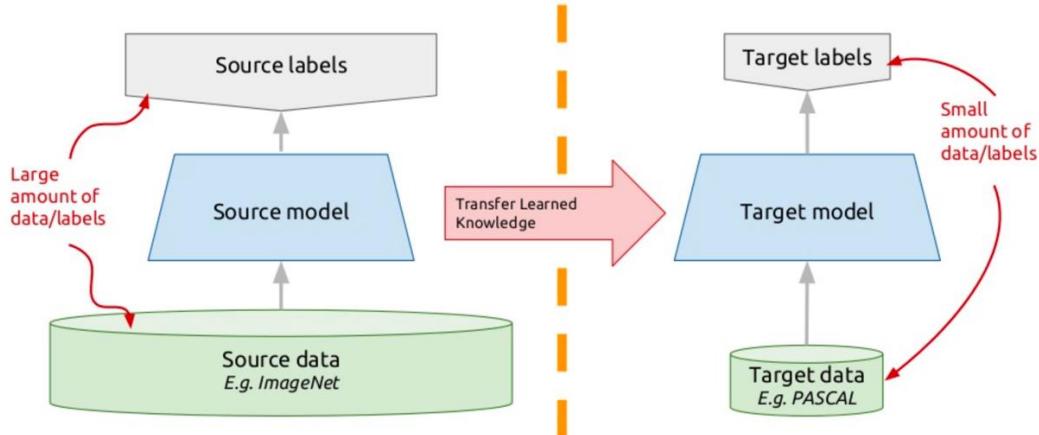
Training: immagini considerati simili hanno un output basso mentre quelle differenti lo avranno alto.



18. Transfer Learning and Domain Adaptation

Il **Transfer learning** è l'abilità di applicare il knowledge (i pesi) imparato da task precedenti e di applicarlo a nuovi task. Si basa sull'apprendimento umano, le persone spesso applicano le conoscenze imparate precedentemente a nuove situazioni mai viste.

Solitamente il knowledge viene imparato da una grande mole di dati, in tal modo la rete sarà in grado di riconoscere e adattarsi a più situazioni/oggetti.



Al giorno d'oggi nessuno traina una rete da zero (con pesi randomici) dato che è rarissimo avere un dataset di una dimensione sufficiente, quel che viene fatto è fare un pretrain della ConvNet su un dataset molto grande (eg. ImageNet, contiene 1.2 milioni di immagini con 1000 categorie) per poi poter usare la rete come punto di partenza oppure per fare una fixed feature extractor per il task di interesse.

Vediamo ora due approcci di transfer learning :

- Deep Features: sottorete già trainata, fixed feature extractor, non si cambiano mai i pesi
- Fine Tuning

Deep Features

Una deep CNN è trainata in modo completamente supervisionato, questa rete la useremo come base di partenza.

Deep feature: l'attivazione dei neuroni è data dai pesi già imparati, l'attivazione dei neuroni viene usata come feature.

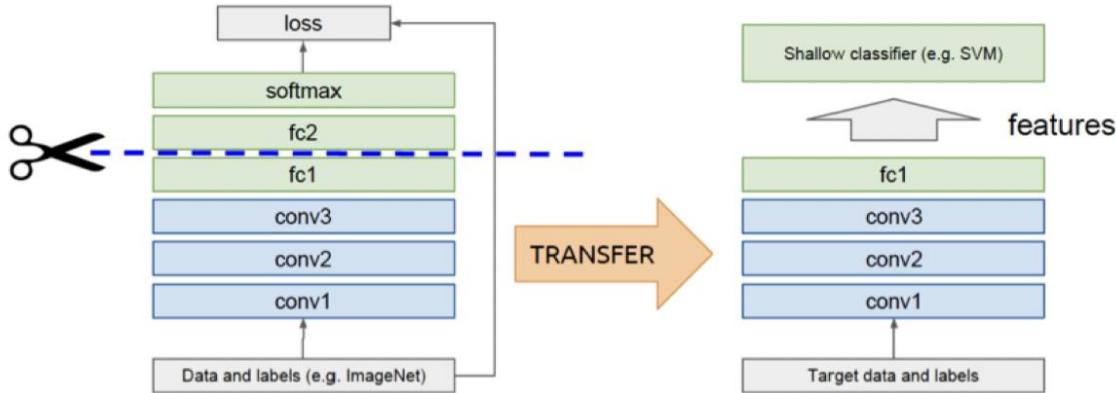
Le deep feature possono essere usate per:

- la **classificazione** (transfer learning) usando un semplici classificatori
- la **ricerca per similarità** (eg CBIR)

Presi una rete intera, andiamo a tagliarla ad un certo punto, la parte sottostante andiamo a riutilizzarla senza bisogno di fare training su quella parte, infatti, i pesi rimarranno fissi.

Vediamo gli **step**:

1. Prendiamo una ConvNet pretrainata su ImageNet
 - a. Rimuoviamo l'ultimo fully-connected layer (questo layer ha in output lo score per 1000 classi)
2. Trattiamo il resto della ConvNet come un feature extractor per il nuovo dataset
 - a. Chiamiamo questa parte di rete feature CNN codes oppure Deep Feature
3. Una volta estratte le feature per tutte le immagini
 - a. Trainiamo un classificatore lineare (eg. Linear SVM or SoftMax classifier) per il nuovo dataset



Fine-Tuning

L'idea generale è quella che in output avrà tutte le label presenti nel dataset con associata una probabilità, ai pesi della parte sottostante della rete viene applicato il fine-tuning utilizzando una continua a backpropagation.

1. Rimpiazza e retraina il classificatore in cima alla ConvNet con il nuovo dataset
2. Fine-tuning sui pesi della rete pre-trainata facendo una continua backpropagation

È possibile ottimizzare tutti i layer di una ConvNet, oppure è possibile mantenere fissi alcuni dei layer precedenti (a causa di problemi di overfitting) e ottimizzare solo alcune parti della rete di livello superiore. Questo è motivato dall'osservazione che i primi layer di una rete sono quelli più generali (ad esempio che riconoscono edge e colori) che possono essere utili per molte attività, i layer successivi sono invece più specifici in base al contenuto del dataset.

Osserviamo che il retrain sulla parte "alta" della rete è banalmente necessario perché vengono cambiate le classi del dataset.

Ad esempio, nel caso di ImageNet, che riconosce tante razze canine, i layer in cima alla rete saranno molto specifici e si occuperanno di riconoscere le razze (nella parte sottostante della rete ci saranno layer per riconoscere che si tratta effettivamente di un cane, etc..).

Vediamo delle situazioni che possono accadere:

- **Nuovo dataset piccolo e simile all'originale**
 - Dato che abbiamo pochi dati, non è una buona idea fare fine-tuning alla ConvNet per i problemi di overfitting. Dato che i dati sono simili ci aspettiamo un alto livello di feature rilevante anche per il nuovo dataset. Quindi la migliore idea è quella di trainare un classificatore lineare sulle CNN (applico le **deep features**)
- **Nuovo dataset grande e simile all'originale**
 - Dato che abbiamo tanti dati e simili al dataset originale non abbiamo problemi di overfitting, posso quindi fare **fine-tuning su tutta la rete** per aggiustare i pesi al nuovo dataset.
- **Nuovo dataset piccolo e molto differente dall'originale**
 - Questo è l'esempio più difficile, dato che abbiamo pochi dati è meglio fare solo **training di un classificatore lineare**. Dato che i dati sono molto differenti, potrebbe non essere una buona idea trainare il classificatore in cima alla rete (che contiene feature più specifiche), si potrebbe però fare su layer precedenti che riguardano caratteristiche più generali. Ovviamente non possiamo aspettarci grandi risultati dato i dati che abbiamo.
- **Nuovo dataset grande e molto differente dall'originale**
 - Dato che il dataset è molto grande e differente dall'originale potremmo creare una **CNN da zero**, tuttavia, conviene prendere una rete pretrainata e **fare fine-tuning su tutta la rete** per adattare i pesi.

Dataset (w.r.t. training data)	SMALL	LARGE
SIMILAR	deep features from higher layers + linear classifier top hidden layer	fine-tune
DIFFERENT	deep features from lower layers + linear classifier earlier hidden layers Worst case	train from scratch or fine-tune  Most different

In small and similar: Takes features from top hidden layer.

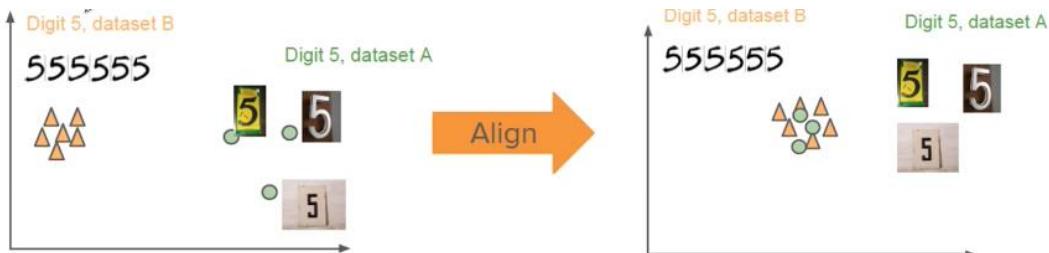
In small and different: Takes features from earlier hidden layers.

Domain Adaptation

Partiamo con un esempio per capire, consideriamo una telecamera di sorveglianza su una strada ed una su un'auto, entrambe devono riconoscere le persone ma i punti di vista sono differenti. Dato che il task e l'output sono gli stessi si può usare *la stessa NN*, questo è possibile tramite il **Domain Adaptation** che permette di **allineare le rappresentazioni** delle persone (features) dai due punti di vista differenti, in questo modo la rete non distinguerà la provenienza delle immagini ma sarà in grado di dire che rappresentano la stessa cosa.

E' presente uno spostamento di dominio quando:

- La dimensione dello spostamento è spesso misurata come distanza tra i sottospazi di origine e target
- Un approccio tipico è quello di imparare una trasformazione dello spazio delle feature per allineare le rappresentazioni di source e target (riducendo la divergenza del dominio)

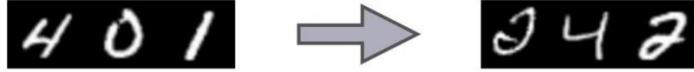


Ricordiamo che una buona rappresentazione deve essere:

- **Discriminativa** per la classificazione di label
- **Indiscriminativa** per la classificazione del dominio

Vediamo che se il punto di vista è lo stesso parliamo di generalizzazione, mentre si parla di domain adaptation nel secondo caso, nel quale, sono differenti.

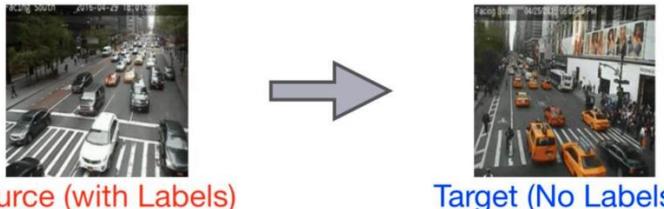
Generalization: **Source (Train)** = **Target (Test)**



Source

Target

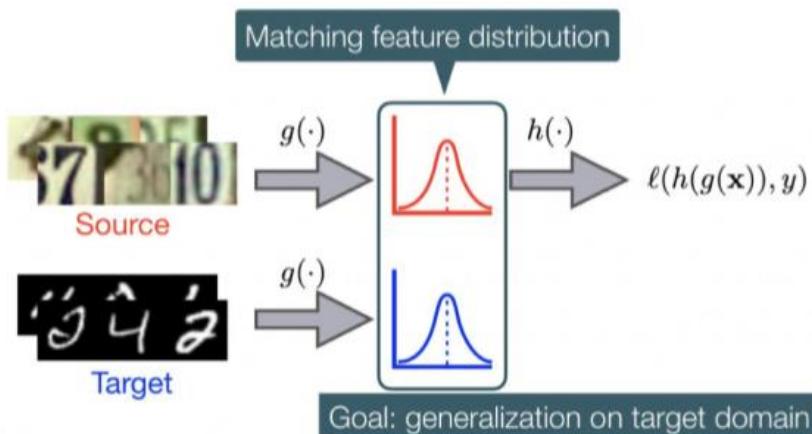
Domain adaptation: **Source (Train)** \neq **Target (Test)**



Source (with Labels)

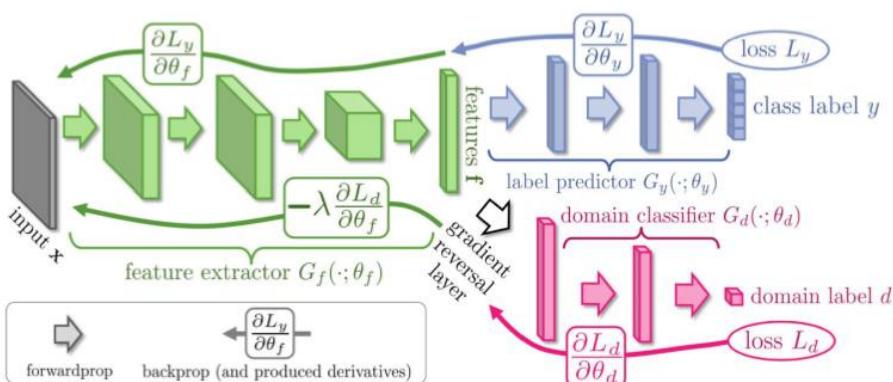
Target (No Labels)

Possiamo fare domain adaptation sia se source e target hanno entrambe le label sia se ce l'ha solo la source, vediamo appunto nell'immagine come possiamo andare a fare matching sulla distribuzione delle feature, applichiamo semplicemente una funzione h il quale output sarà tale al grafico blu.



Una volta estratte le feature dell'immagine (alla fine della verde) le usiamo nella blu per classificare gli oggetti dell'immagine mentre la rossa serve per capire il dominio di appartenenza.

La rete rossa serve rendere indistinguibili i due punti di vista, porterà la rete a non distinguere se lo stesso oggetto proviene da un dominio diverso.



19/20. Object Detection and Segmentation

Semantic segmentation

Label for the whole image.

Classification

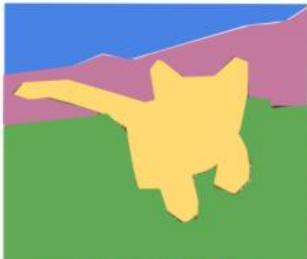


CAT

No spatial extent

Each pixel in the image is labelled. Don't recognize instances. We don't know if there are two cats or not.

Semantic Segmentation



**GRASS, CAT,
TREE, SKY**

No objects, just pixels

We are trying to detect object

Object Detection



DOG, DOG, CAT

Multiple Object

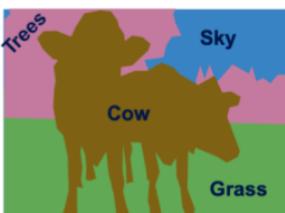
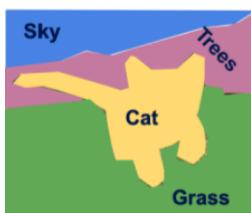
We're able to detect the dog and then we can classify each pixels of dogs and cats.

Instance Segmentation

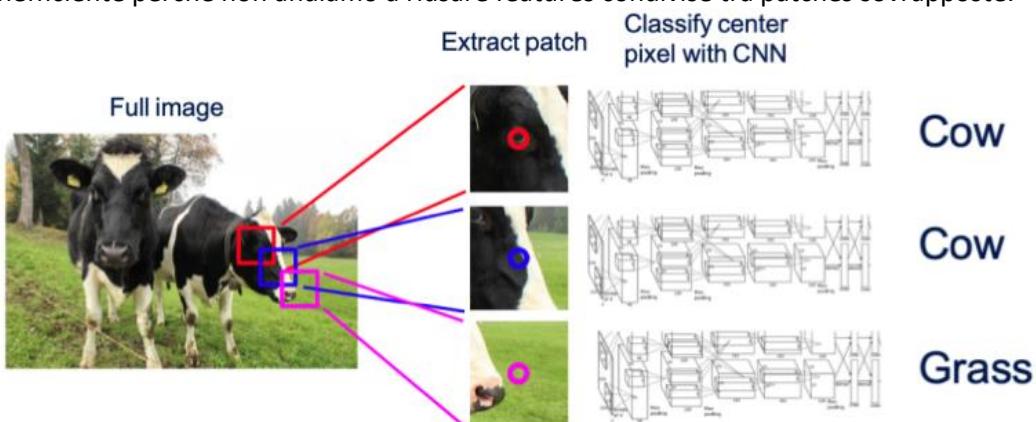


DOG, DOG, CAT

This image is CC0 public domain



Usiamo una sliding window tramite la quale classifichiamo ogni pixel guardando i vicini. Questo è molto inefficiente perché non andiamo a riusare features condivise tra patches sovrapposte.

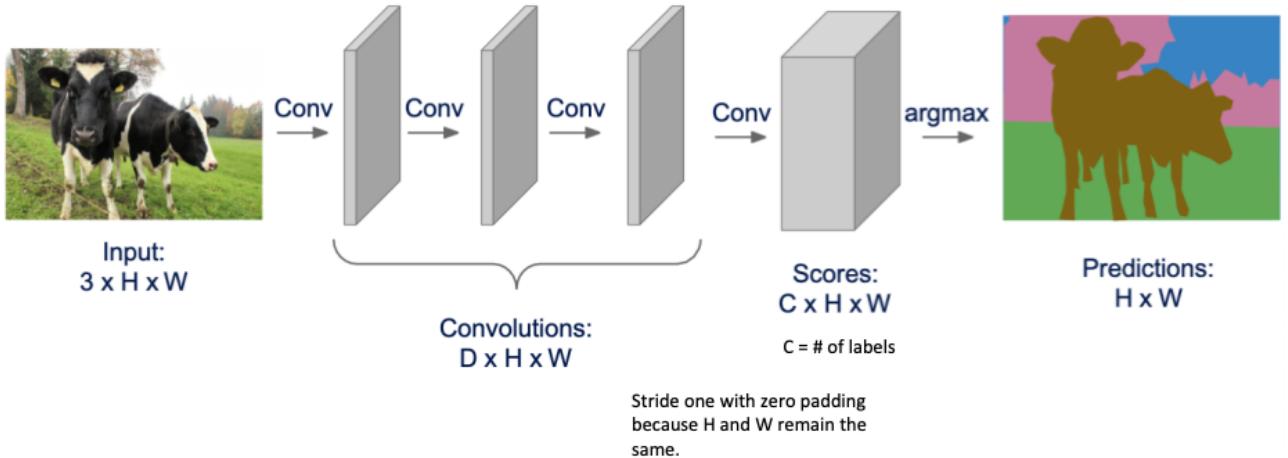


Andiamo a fare una rete con più convolutional layer per fare predizione sui pixel tutti in una volta sola, è importante notare che la grandezza (H e W) rimane invariata tra input, layers e predictions, adottiamo a tal motivo stride 1 con zero padding.

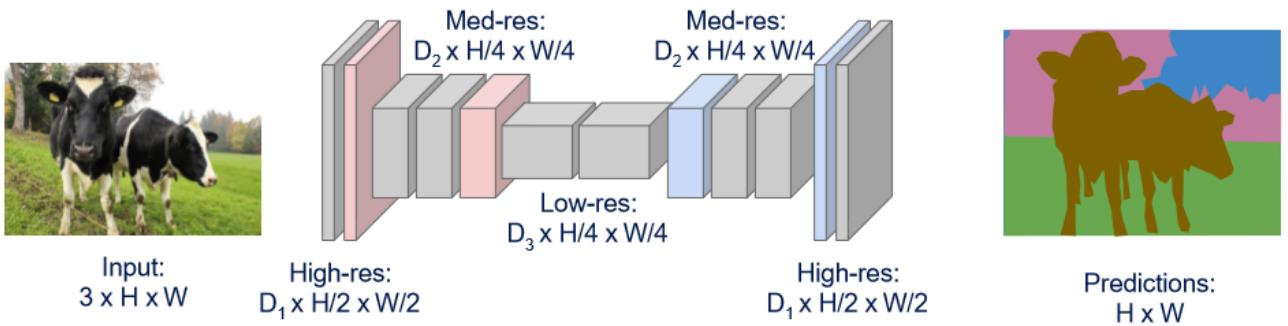
L'ultimo layer ha profondità C, che corrisponde al numero di label che la rete riconosce. L'argmax viene utilizzato per normalizzare l'output.

Per ogni pixel in output avrò tanti score quante sono le label da classificare, ognuno mi dirà la percentuale di che cosa sarà quel pixel (eg. 80% cow, 5% sky e quindi lo associerò a cow).

Design a network as a bunch of convolutional layers to make predictions for pixels all at once!

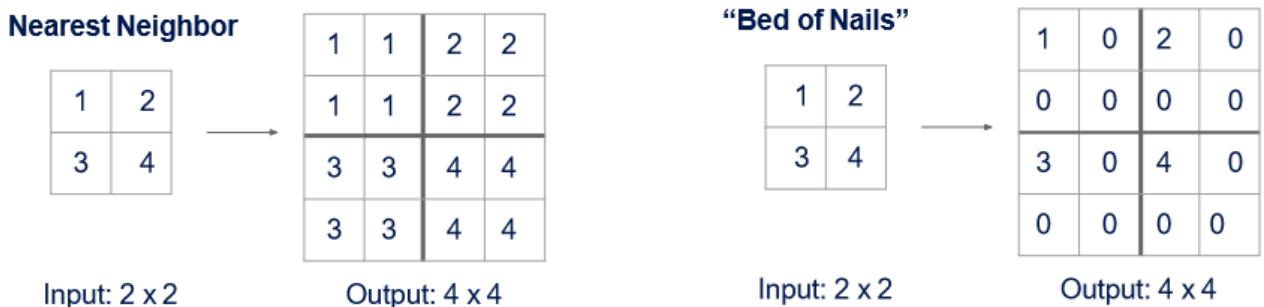


Un altro problema è che mantenere su ogni layer la dimensione dell'immagine originale porta la rete ad essere costosa dal punto di vista delle operazioni da svolgere e dei parametri da mantenere in memoria, per risolvere questo problema utilizziamo delle tecniche di **downsampling** e di **upsampling** per ottenere una rete simile a quella mostrata nella figura seguente.



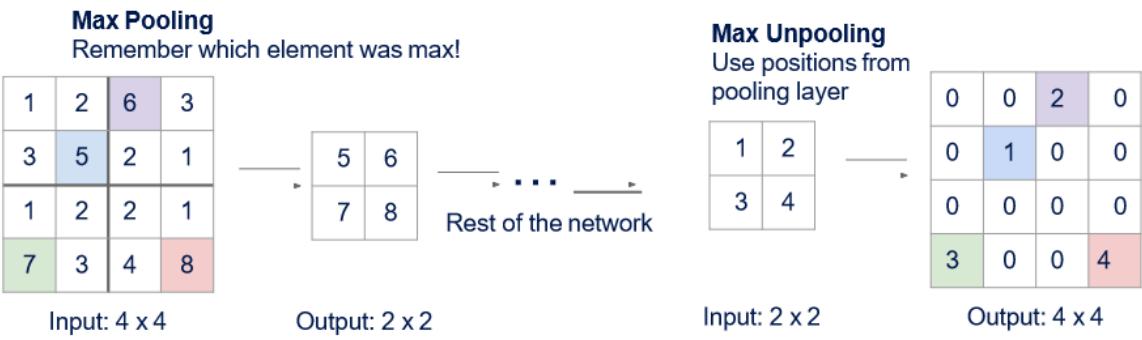
Per quanto riguarda il downsampling abbiamo già visto che può essere attuato tramite il **pooling** o tramite un layer di convoluzione con uno **stride** superiore a 1.

Per quanto riguarda l'upsampling esistono diverse metodologie.



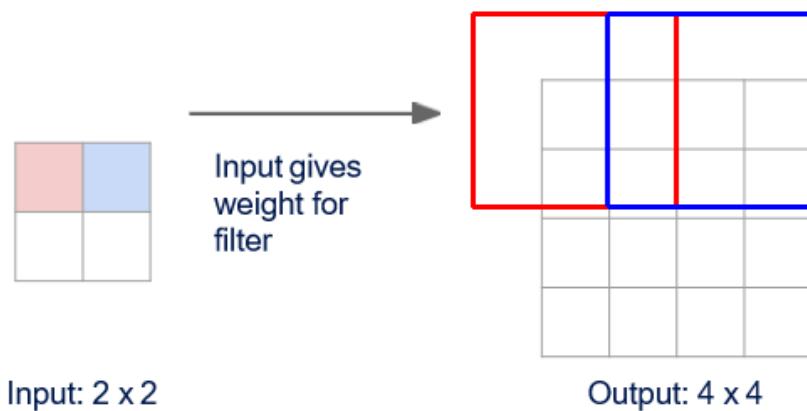
Con il nearest neighbor si inserisce lo stesso numero nella finestra 2×2 mentre con il bed of nails si inserisce il valore nella casella in alto a sinistra e il resto della finestra è riempita con 0.

Si può anche usare il **max unpooling**, come mostrato nella figura seguente, si salva la posizione del valore massimo prima del max pooling e successivamente il valore viene inserito nuovamente nella casella originaria, il resto sono riempiti con degli zeri.



Un altro approccio è quello della **transpose convolution**. Come abbiamo già detto, cambiando lo stride nella convoluzione è possibile ridurre la dimensione del layer in output, prendiamo questa idea per aumentare l'output in uscita dal layer. Vediamo un esempio tramite le figure sottostanti.

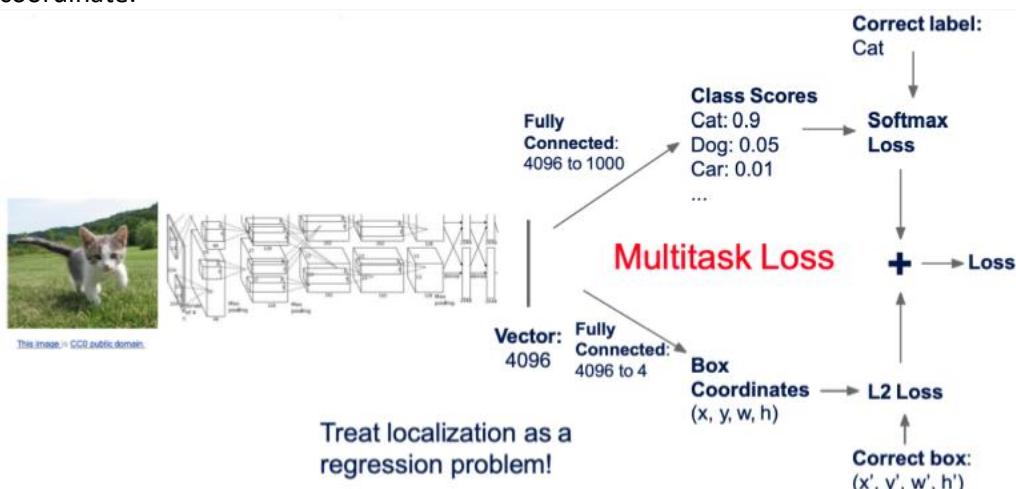
3 x 3 transpose convolution, stride 2 pad 1



Tramite lo stride muovo di due pixel la finestra nell'output per ogni casella dell'input. Si moltiplica l'input per i valori del filtro e, come mostrato nella figura, si sommano i valori nei punti dove le finestre si sovrappongono.

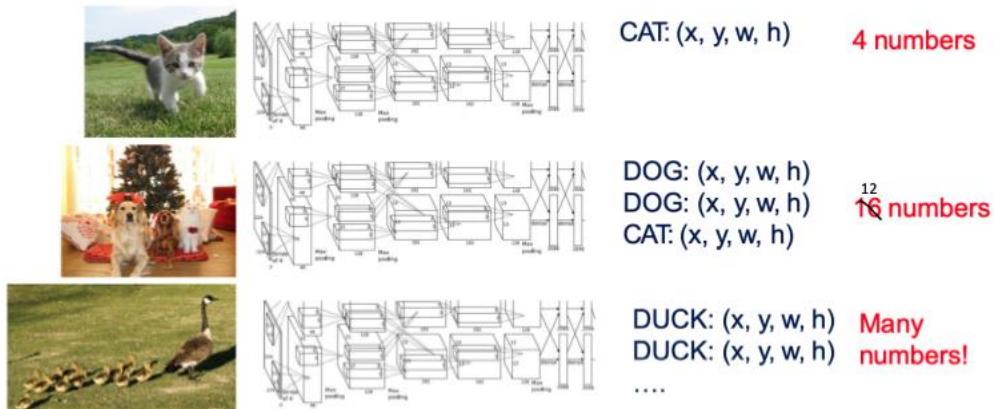
Object detection

L'object detection permette di riconoscere molteplici istanze dello stesso oggetto e applica anche un box intorno ad esso. È necessaria quindi una parte di CNN che si occupa di trovare anche le coordinate del box. Come mostrato dall'immagine seguente avremo una Loss function sia per modificare la label che le coordinate.

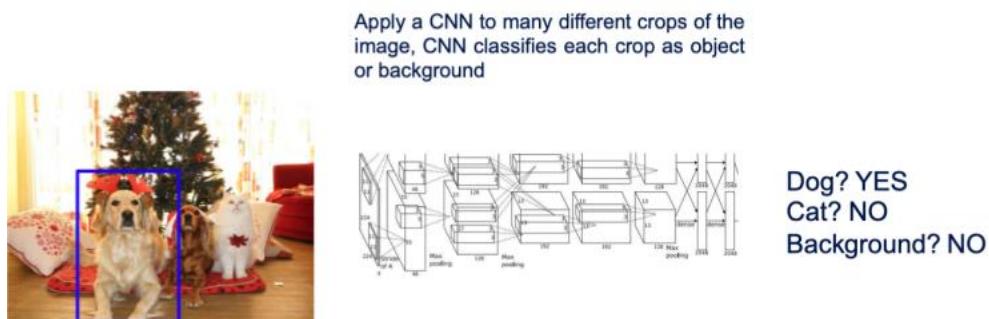


Chiaramente la CNN ha subito un pretrain tramite, ad esempio, ImageNet e poi adattata alla nostra task tramite il transfer learning. Dando l'intera immagine che input avrò in output 4 numeri per ogni istanza di

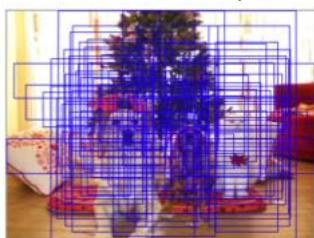
oggetto, questi numeri riguardano le coordinate del box, all'aumentare degli oggetti che devono essere riconosciuti aumentano di molto i valori che devo dare come output.



Una prima soluzione potrebbe essere quella di mandare in input alla CNN solo una parte dell'immagine alla volta, la CNN quindi la analizzerà la parte dando un risultato, come mostrato nella figura seguente.

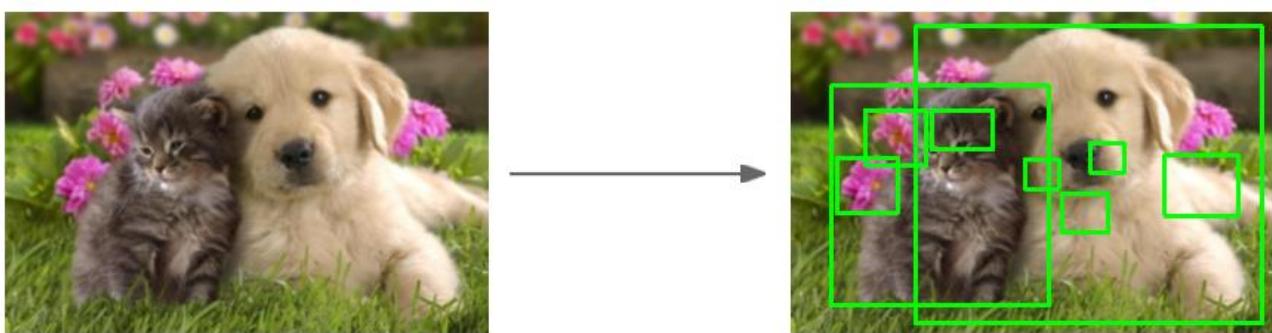


Il problema di questo approccio è che la CNN deve analizzare un grandissimo numero di sottoimmagini con scale e ratei diversi, questo porta a un enorme costo computazionale.

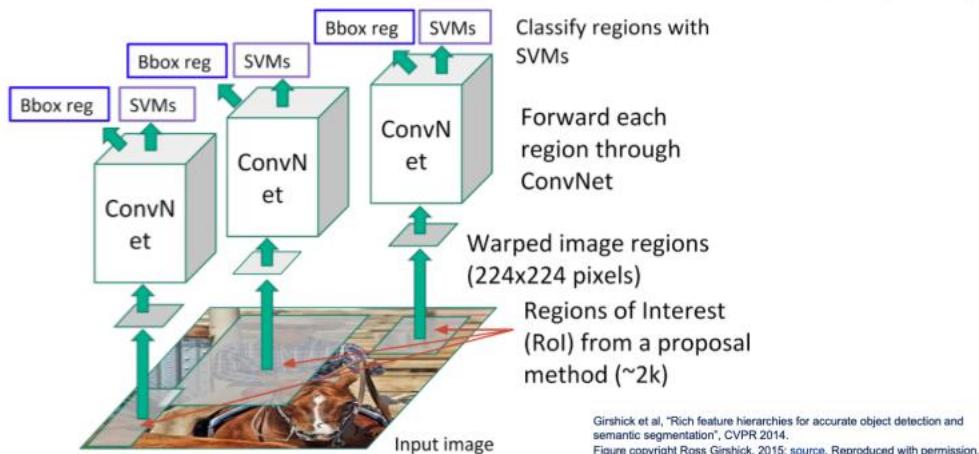


Region proposal

Questo approccio si basa non sull'analizzare tantissime sottoimmagini prese casualmente, ma di analizzare solo le **regioni di interesse** che sono quelle con più alta probabilità di contenere un oggetto. L'algoritmo deve essere veloce, possiamo usare ad esempio *Selective Search* che restituisce circa 2000 regioni di interesse in pochi secondi.



Vediamo nella seguente immagine che viene utilizzata una CNN insieme a questo algoritmo
Predict "corrections" to the Roi: 4 numbers: (dx, dy, dw, dh)

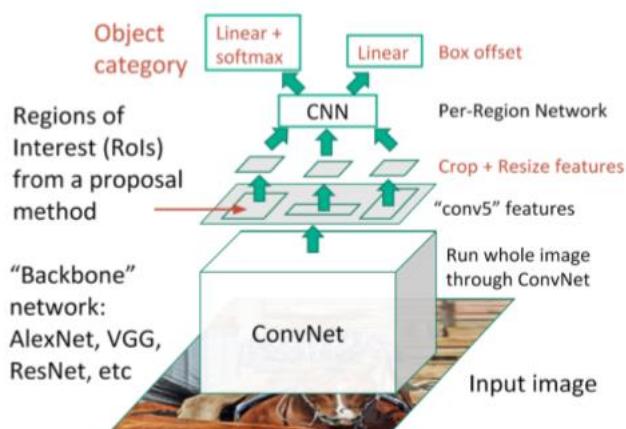


Partendo dal basso, analizziamo ogni regione di interesse dataci dall'algoritmo. Le immagini di interesse vengono ridimensionate in modo da essere quadrate tutte della stessa dimensione. Dopo questo step, si usa la **stessa** ConvNet per ognuna delle immagini, ogni ConvNet classifica la regione e trova le coordinate del box tramite gli ultimi due stadi. È necessario trovare le coordinate del box in quanto le regioni di interesse potrebbero non essere quelle ottime.

Questo approccio, tuttavia, è lento in quanto deve analizzare circa 2000 regioni di interesse e per ognuna applicare la ConvNet. Per questo motivo questo approccio viene chiamato **slow R-CNN**.

Per risolvere questo problema si può processare l'immagine prima di dividerla in sottoimmagini.

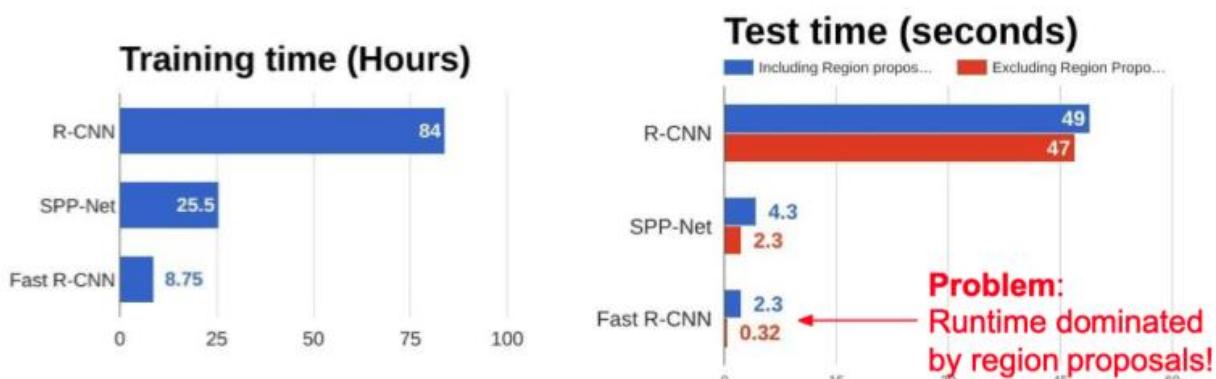
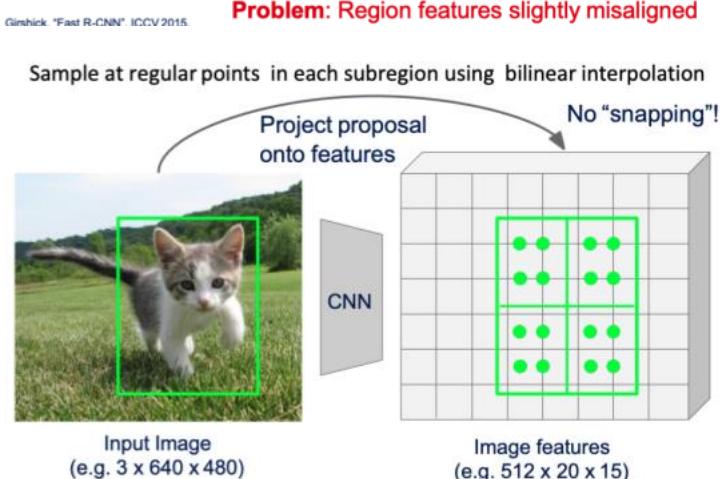
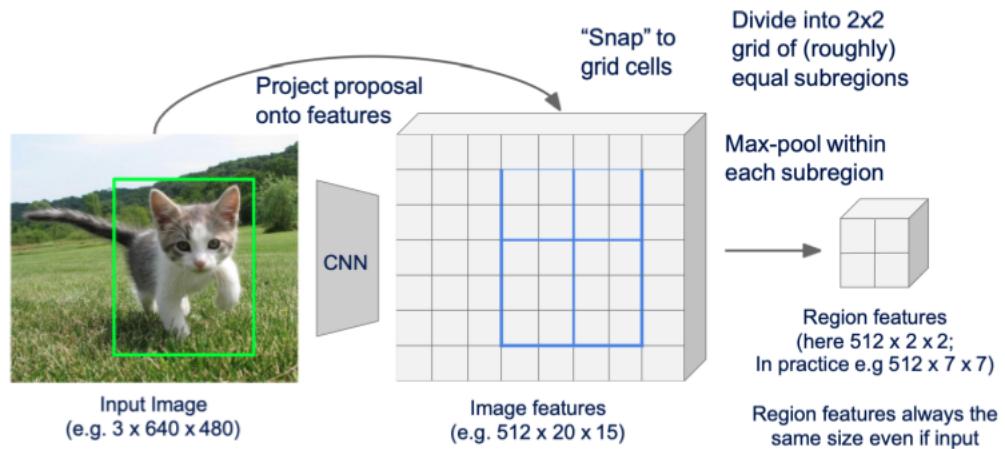
Questo nuovo approccio è chiamato **fast R-CNN** ed è mostrato nella seguente figura.



Come si vede, la ConvNet è applicata sull'intera immagine, la rete si ferma però ad un determinato layer contenente le features. Si trovano quindi le regioni di interesse sull'immagine e poi vengono trasportate sul layer contenente le features, si applica lo stesso ridimensionamento per tutte le regioni di interesse e successivamente posso classificare la label e trovare le coordinate del box tramite la stessa CNN.

Nelle immagini successive vengono mostrati i due metodi tramite i quali si possono ricavare le regioni di interesse sul layer con le fature.

Il primo consiste nel proiettare la regione dall'immagine al layer e poi allineare la proiezione alle griglie, questo porta ovviamente le due regioni ad essere leggermente disallineate. Per risolvere questo problema si può direttamente usare la proiezione così come l'abbiamo calcolata.



Si può notare dall'immagine precedente che l'approccio tramite l'algoritmo di region proposal, che lavora sull'immagine vera, proiettato sul layer con le feature risulta essere estremamente più veloce della sua controparte nella quale ogni regione viene analizzata separatamente.

Il region proposal impiega comunque troppo tempo (2.3 secondi) se intendiamo usare questo approccio, ad esempio, per la guida autonoma vorremmo un tempo di risposta più simile a quello senza region proposal (0.32 secondi).

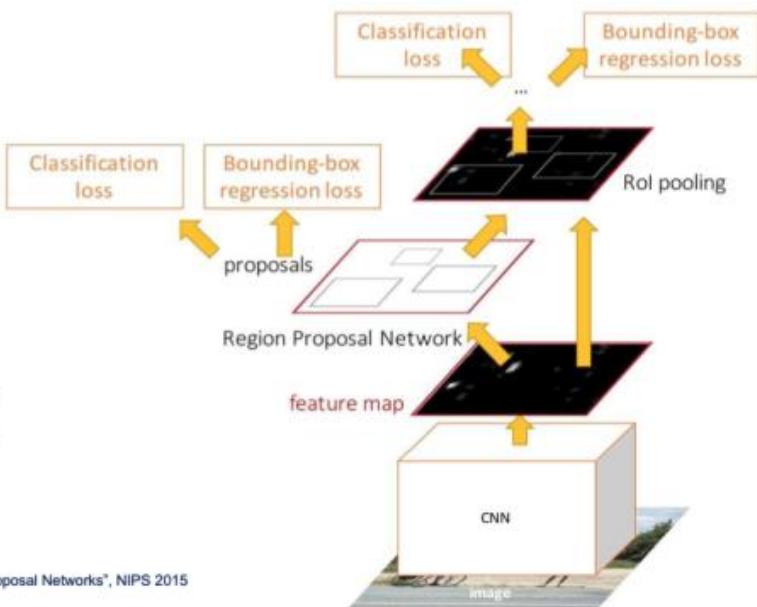
Per risolvere questo problema viene introdotto il **faster R-CNN**, esso sostituisce l'algoritmo per il region proposal con una CNN che ha lo stesso scopo, ma viene applicata direttamente sul layer con le feature. Un esempio è mostrato nella figura seguente.

In parole poche faster è differente per i seguenti motivi:

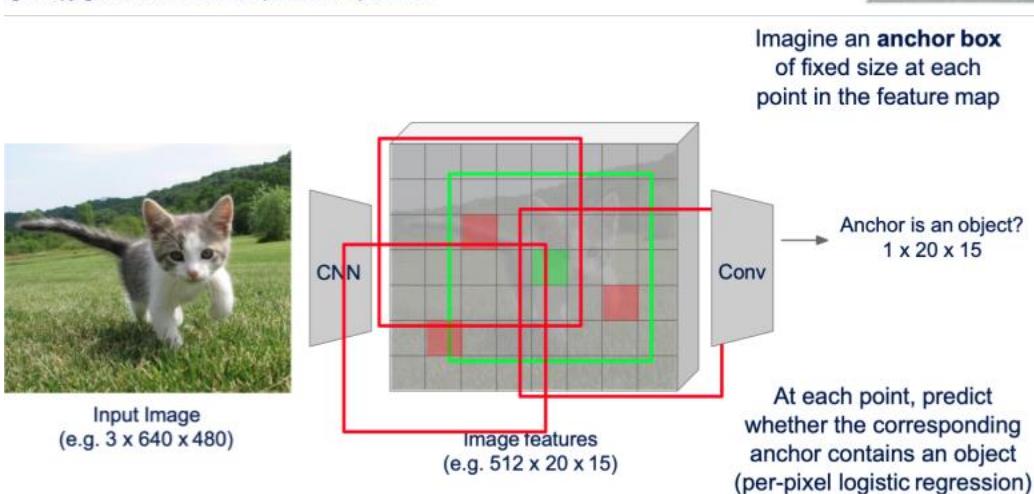
- Le regioni sono trovate sullo spazio delle feature e non più sull'immagine
- Usiamo un approccio di deep learning, infatti, usiamo una CNN per riconoscere le regioni di interesse

Insert Region Proposal Network (RPN) to predict proposals from features

Otherwise same as Fast R-CNN:
Crop features for each proposal,
classify each one

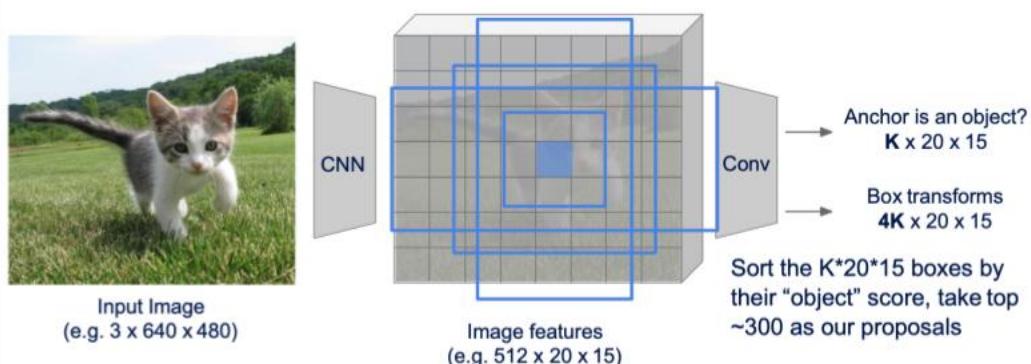


Ren et al., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission



Per trovare le regioni di interesse direttamente nel layer con le feature si procede nel seguente modo, si valuta una finestra di dimensione fissa per ogni cella del layer, chiamata **anchor**. Ogni anchor viene valutata per capire se contiene o meno un oggetto rilevante, se lo contiene la finestra originaria viene trasformata per renderla più vicina possibile alla box che contiene al meglio l'oggetto nell'immagine (ovviamente necessito di 4 valori).

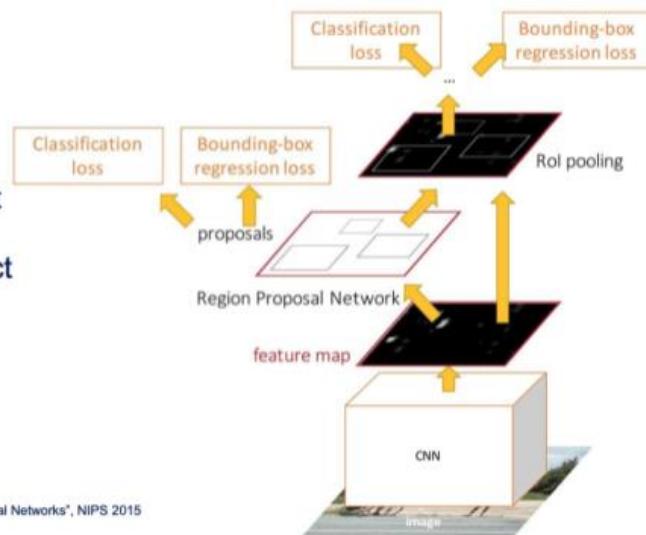
In practice use K different anchor boxes of different size / scale at each point



In realtà vengono valutate K finestre per ogni anchor e di conseguenza 4K valori (H,W,x,y) per identificare le relative box. Il passo successivo è far analizzare alla seconda CNN l'anchor per classificare l'oggetto che contiene. NOTA BENE: la CNN che trova le regioni di interesse non classifica l'oggetto ma riconosce solo che c'è un oggetto rilevante rispetto al background.

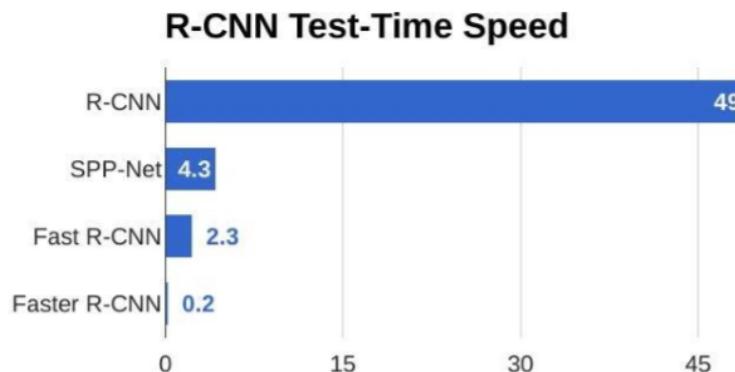
Jointly train with 4 losses:

1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates

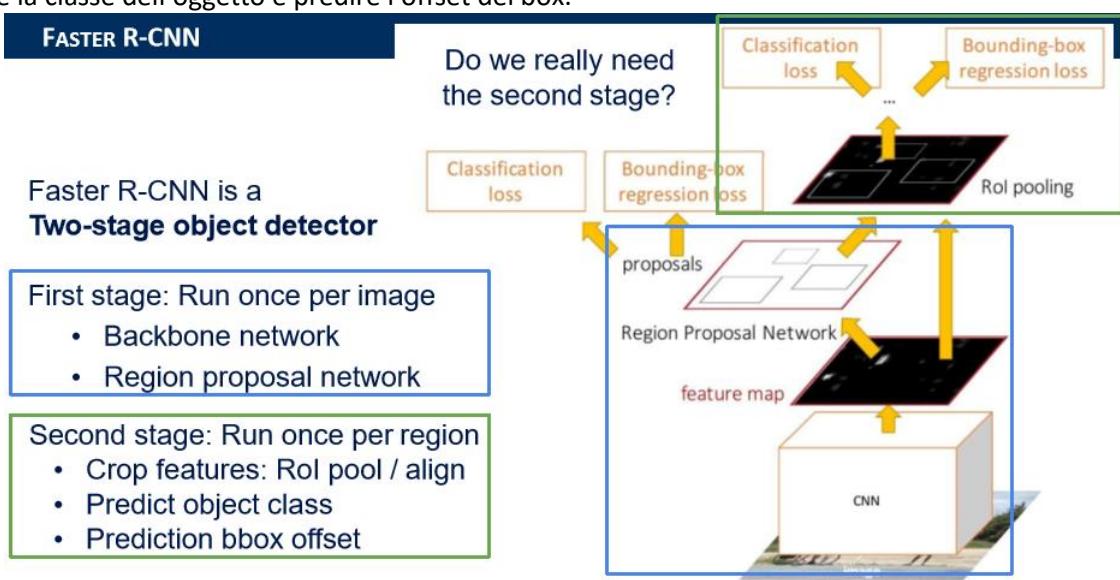


en et al., "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Vediamo nella seguente immagine il confronto tra le prestazioni

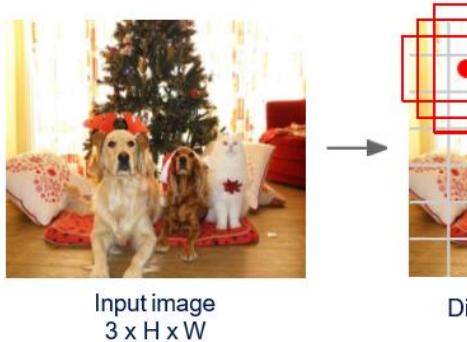


Faster R-CNN utilizza di due fasi per riconoscere l'oggetto, il primo stage gira una volta per immagine ed è composto dalla rete di **backbone**, cioè quella che preprocessa l'immagine, e dalla rete di **region proposal**. Il secondo stage gira una volta per regione trovata e consiste nel tagliare le regioni nel layer delle feature, predire la classe dell'oggetto e predire l'offset del box.

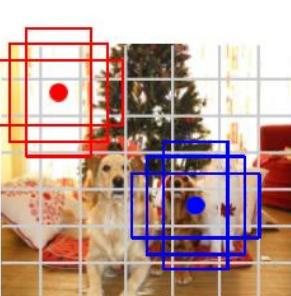


È intuibile che in realtà la seconda CNN è superflua in quanto anche il lavoro di classificazione può essere svolto dalla prima CNN. L'approccio utilizzato è chiamato **YOLO** (you only look once), essenzialmente si

elimina la CNN verde e, in quella blu, oltre a cercare se una regione contiene un oggetto rilevante si applicano anche tutte le label necessarie a classificarlo come mostrato nella figura seguente.
Altri approcci simili sono SSD e RetinaNet.



Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017



Divide image into grid
 7×7
Image a set of **base boxes**
centered at each grid cell
Here $B = 3$

Within each grid cell:

- Regress from each of the B base boxes to a final box with 5 numbers: $(dx, dy, dh, dw, confidence)$
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output:
 $7 \times 7 \times (5 * B + C)$

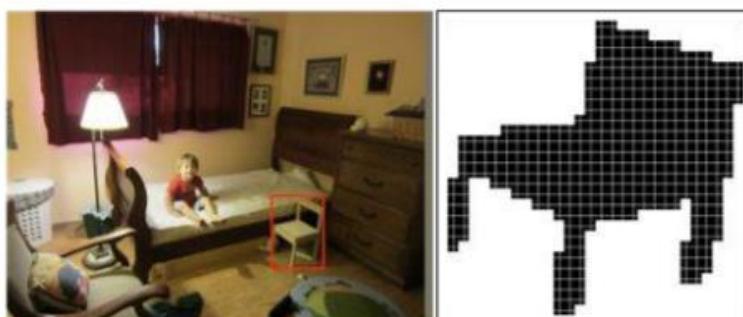
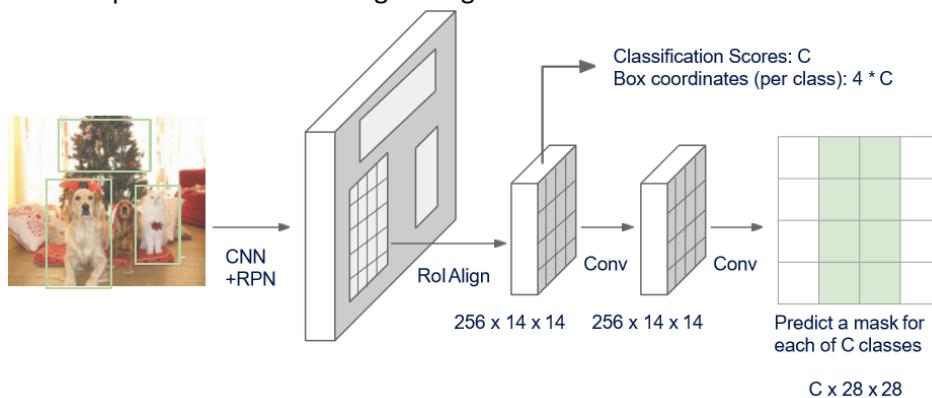
Come mostrato nella figura successiva, ci possono essere diverse variabili che modificano la struttura della rete generale.

Backbone Network	"Meta-Architecture"	Takeaways
VGG16	Two-stage: Faster R-CNN	Faster R-CNN is slower but more accurate
ResNet-101	Single-stage: YOLO / SSD	SSD is much faster but not as accurate
Inception V2	Hybrid: R-FCN	Bigger / Deeper backbones work better
Inception V3		
Inception ResNet		
MobileNet		
	Image Size # Region Proposals	
	...	

Instance segmentation

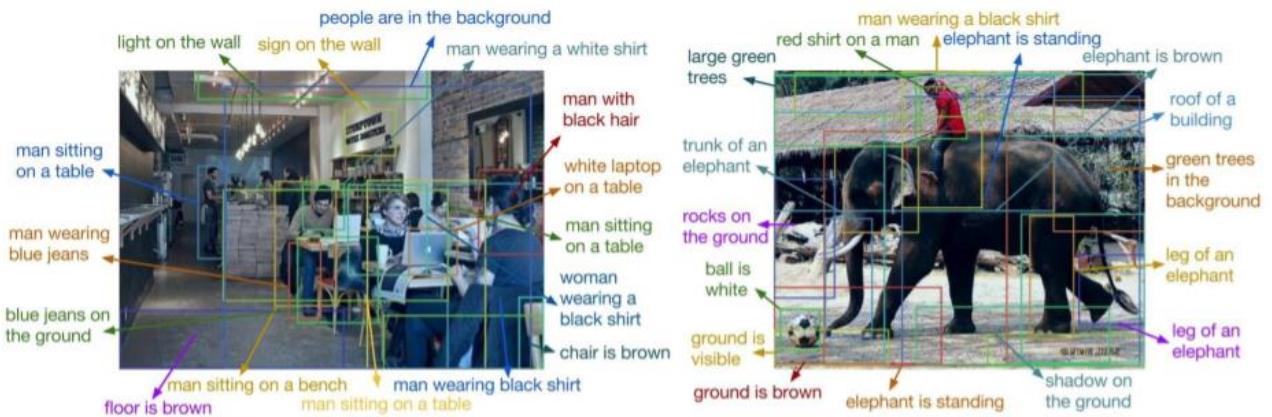
Per l'instance segmentation siamo interessati ad una maschera sopra l'oggetto e non più di un box che lo racchiude. Per fare questo possiamo semplicemente utilizzare la struttura del **faster R-CNN** aggiungendo una parte dedicata al **mask prediction**.

Un esempio è mostrato nelle figure seguenti.



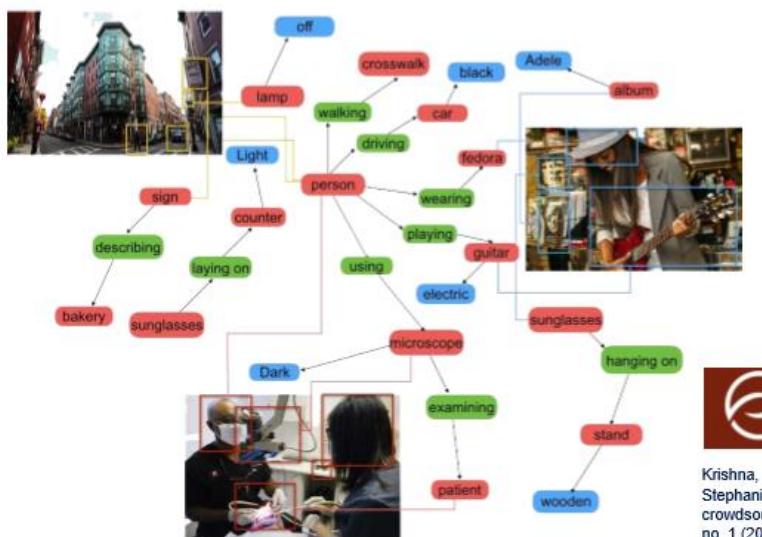
Beyond 2D object detection

È possibile anche assegnare una caption ad ogni regione dell'immagine invece di classificare l'oggetto intero come mostrato in figura.

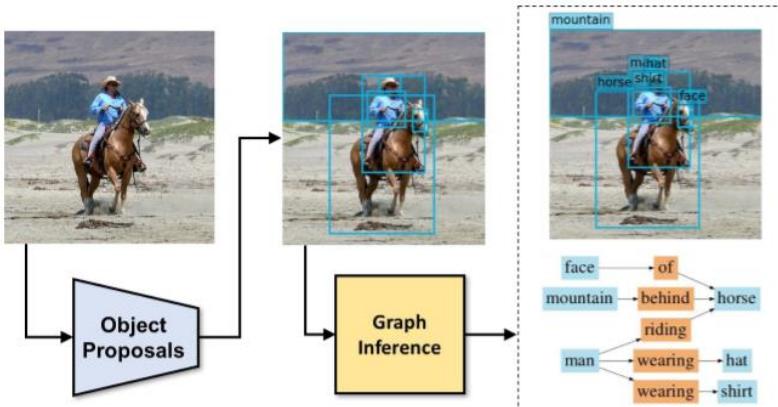


Di seguito è mostrata la rete in grado di svolgere questo compito, essenzialmente l'unica differenza è la parte chiamata LSTM che al posto delle classi genera del testo da affiancare alla regione.

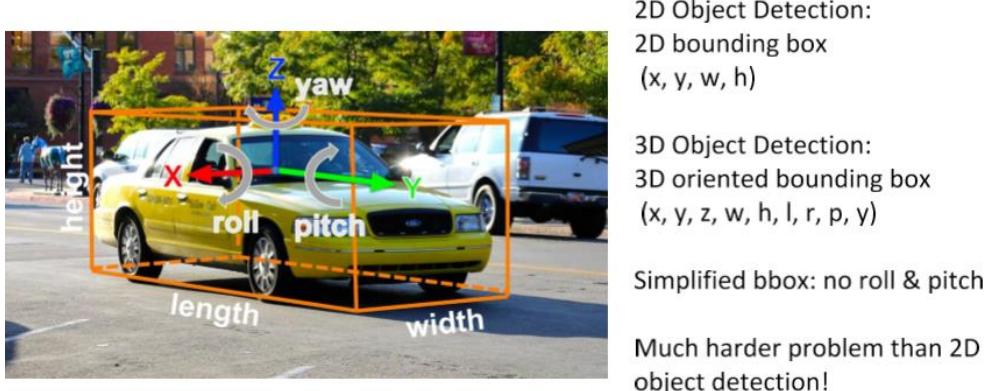
Si possono anche trovare delle relazioni fra delle immagini tramite **Scene Graph**, essenzialmente si crea un grafo che mette in relazione regioni di immagini che hanno caratteristiche simili come possiamo vedere dalla seguente immagine.



Alternativamente possiamo trovare le regioni di interesse su un'immagine, classificarle e poi chiedere al sistema di generale le relazioni. Otterremo una cosa simile a quella proposta nell'esempio seguente.



Infine, si possono riconoscere anche oggetti in 3D, la box intorno all'oggetto sarà quindi descritta tramite più valori come mostrato in figura. Chiaramente individuare un oggetto in 3D risulta essere molto più complicato di calcolare la box in due dimensioni attorno all'oggetto.



Si possono anche fare predizioni sull'oggetto in 3 dimensioni partendo dall'immagine nei tre seguenti modi

Voxel:

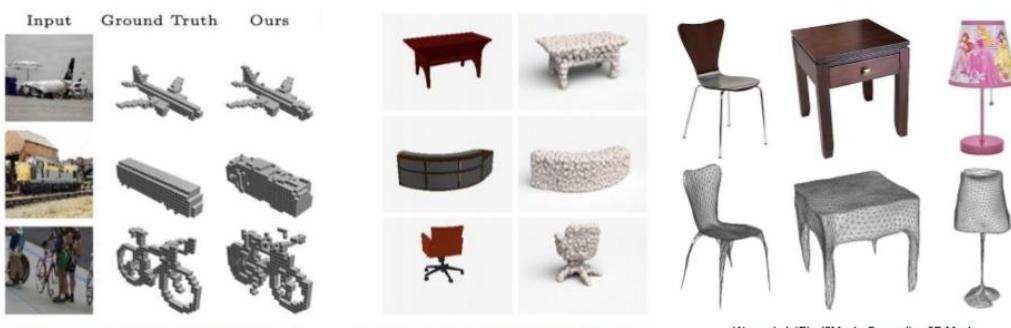
$D \times D \times D$ binary

Pointcloud:

$V \times 3$ float

Mesh:

$V \times 3$ float, $F \times 3$ int



21/22. Generative Models

I generative models sono l'abilità di generare immagini.

In questo capitolo vedremo la differenza tra supervised e unsupervised learning, e i seguenti generative models:

Approximate density

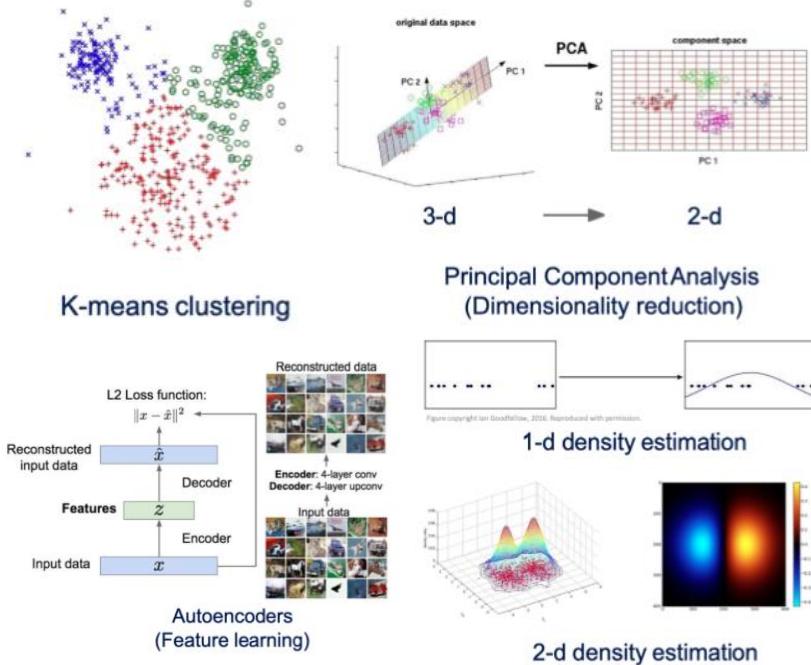
- PixelRNN PixelCNN (Recurrent NN as background), solo a scopo didattico in pratica non vengono usati
- Variational Autoencoder (Autoencoder as background)
- Generative Adversarial network

Supervised vs Unsupervised

Il **supervised learning** tratta dati (x, y) dove x è il dato e y la label, il nostro scopo è quello di imparare una **funzione** per mappare una label ad un'immagine. Gli esempi più comuni sono quelli di classificazione, regressione, object detection, semantic segmentation, image captioning (find image given text or the opposite), etc..

Nel **unsupervised learning** abbiamo solo i dati ma non abbiamo etichette! Lo scopo è quello di imparare alcune **strutture nascoste** tra i dati. In altre parole, lo scopo di questo approccio è quello di trovare features per distinguere i dati, anche se non abbiamo label. Esempi: clustering, dimensionality reduction, density estimation, feature learning, etc...

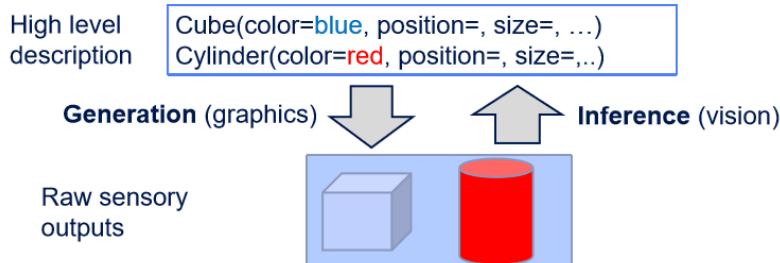
In questo approccio fare training sui dati è più economico rispetto al supervised, dal momento che non abbiamo label. Ricordiamo che risolvere l'unsupervised learning significa capire la struttura delle visual words, questo ci aiuterà a creare immagini, etc.



Introduction to Generative Models

Richard Feynman disse “What I cannot create, I do not understand”, per i generative model “What I understand, I can create”. Quindi dobbiamo prima capire i dati, e da questi possiamo quindi creare img. Nella figura sottostante vediamo la differenza tra la generazione di oggetti a partire dalla descrizione e viceversa.

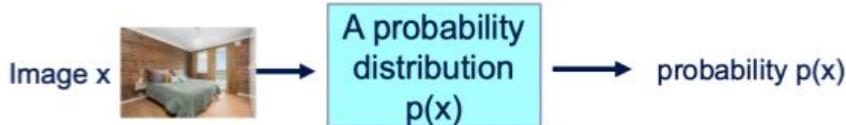
How to generate natural images with a computer?



Our models will have **similar structure (generation + inference)**

Un generative model statistico è la **distribuzione di probabilità $p(x)$** , i dati sono gli esempi (eg immagine di una camera da letto) e dobbiamo conoscere in anticipo alcuni parametri come la forma parametrica (eg Gauss), loss function (eg maximum likelihood), optimization algorithm, etc.

Quindi dalla probabilità $p(x)$ posso andare, facendo sampling, a creare una o più nuove img.

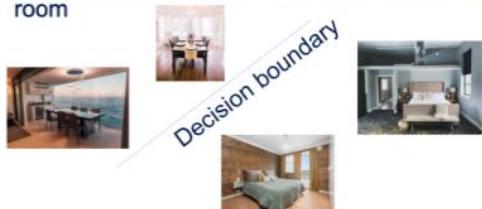


It is generative because **sampling from p(x) generates new images**



Sotto vediamo la distinzione tra discriminative e generative, nel primo andiamo a classificare avendo come input l'immagine X, usiamo una **probabilità condizionata (conditional distribution)** per ottenere un buon decision boundary. In generative invece non abbiamo l'immagine di input X, si richiede quindi il modello di **probabilità congiunta (joint distribution)**.

Discriminative: classify bedroom vs. dining room



The input image X is always given. Goal: a good decision boundary, via **conditional distribution**

$$P(Y = \text{Bedroom} | X = \text{[image]}) = 0.0001$$

Ex: logistic regression, convolutional net, etc.

Generative: generate X



The input X is not given.
Requires a model of the **joint distribution**

$$P(Y = \text{Bedroom}, X = \text{[image]}) = 0.0002$$

Here is not given

Joint and conditional are related via **Bayes Rule**:

$$P(Y = \text{Bedroom} | X = \text{[image]}) = \frac{P(Y = \text{Bedroom}, X = \text{[image]})}{P(X = \text{[image]})}$$

Discriminative: X is always given, does not need to model $P(X = \text{[image]})$

Therefore it cannot handle missing data $P(Y = \text{Bedroom} | X = \text{[image]})$

Class **conditional generative models** are also possible:

$$P(X = \text{[image]} | Y = \text{Bedroom})$$

It's often useful to condition on rich side information Y

$$P(X = \text{[image]} | \text{Caption} = \text{"A black table with 6 chairs"})$$

A discriminative model is a very simple conditional generative model of Y:

$$P(Y = \text{Bedroom} | X = \text{[image]})$$

I generative model possono essere anche condizionali dato che possiamo fornire una descrizione e vogliamo generare un'immagine conforme alla descrizione.

CHIEDI A QUALCUNO DI STA PROB CHE UN CHO CAPITO UN DICK

Questa pratica può essere applicata anche per aumentare la risoluzione dell'immagine partendo da una a bassa risoluzione, come si vede qua sotto.

Conditional generative model $P(\text{high res image} | \text{low res image})$



Oppure anche per sostituire o modificare immagini/video già esistenti, qua sotto vediamo come facciamo diventare il cavallo una zebra.

Conditional generative model $P(\text{zebra images} | \text{horse images})$



Abbiamo i training data con associata una probabilità, l'idea è quella di imparare una distribuzione di probabilità $p_{\text{model}}(x)$ simile a $p_{\text{data}}(x)$ in modo tale da poter generare dei samples.
La distribuzione di probabilità può essere esplicita, come nel caso appena descritto, oppure implicita dove andiamo ad imparare un modello che può fare sampling su $p_{\text{model}}(x)$ senza che la distr di prob sia conosciuta.



Training data $\sim p_{\text{data}}(x)$



Generated samples $\sim p_{\text{model}}(x)$

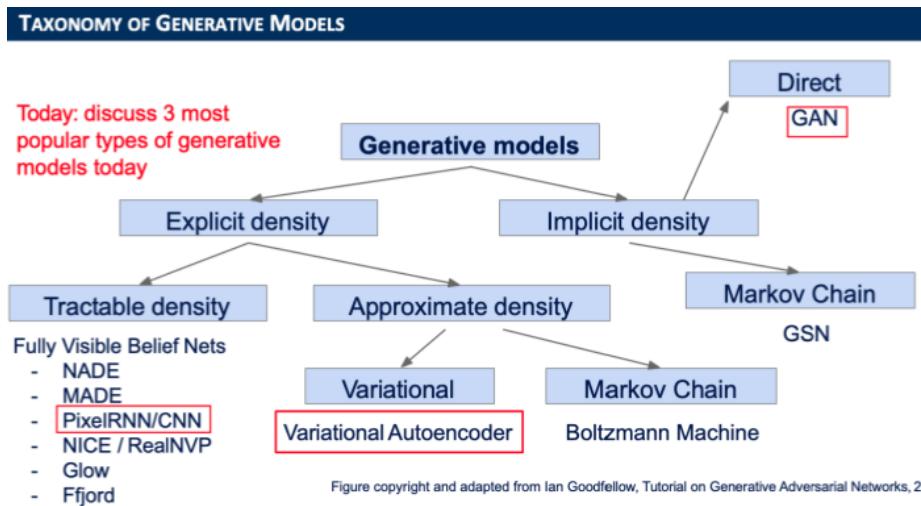
Want to learn $p_{\text{model}}(x)$ similar to $p_{\text{data}}(x)$

Addresses density estimation, a core problem in unsupervised learning

Several flavors:

- Explicit density estimation: explicitly define and solve for $p_{\text{model}}(x)$
- Implicit density estimation: learn model that can sample from $p_{\text{model}}(x)$ w/o explicitly defining it

Vediamo come possiamo suddividere i generative models



PixelRNN and PixelCNN

Questo approccio è **explicity density**.

Andiamo ad usare una catena di regole per decomporre la probabilità di un'immagine x in un prodotto di 1-d distribuzioni:

$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$

↑ ↑ ↑
 Likelihood of Probability of i'th pixel value Will need to define
 image x given all previous pixels ordering of "previous
 pixels"
 Complex distribution over pixel
 values => Express using a neural
 network!

Then maximize likelihood of training data

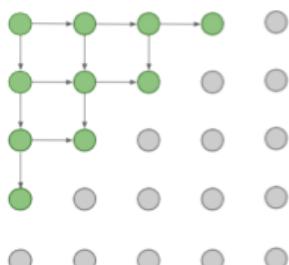
(likelihood means probability)

Quindi la prob dell'immagine sarà il prodotto della probabilità di un pixel considerando tutti i precedenti.

Vediamo appunto sotto l'uso ricorsivo di pixel già valutati, per creare mano a mano tutta l'img.

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

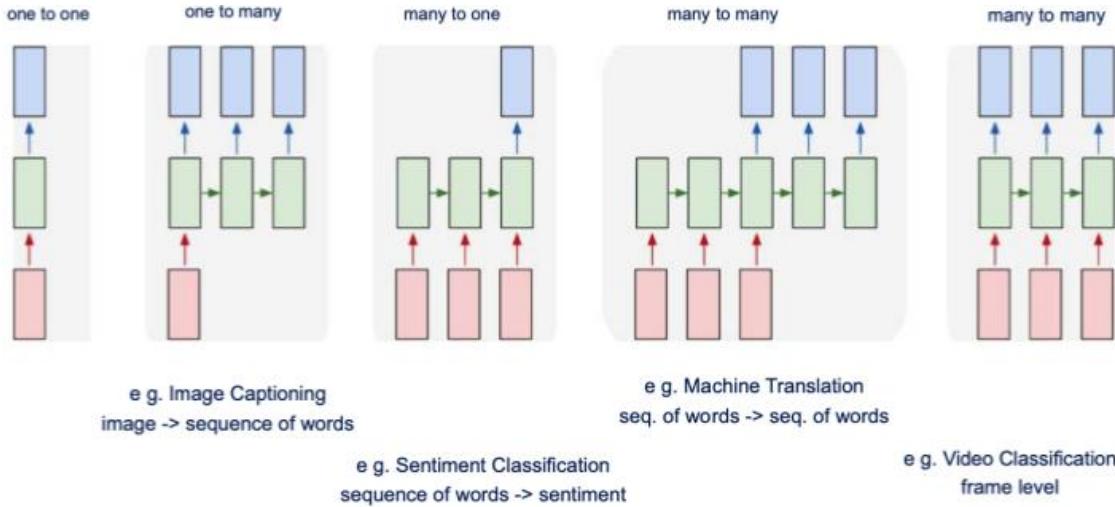


Drawback: sequential generation is slow!

La generazione dell'immagine è lenta dato che dobbiamo tenere di conto di tanti pixels.

RNN in brief

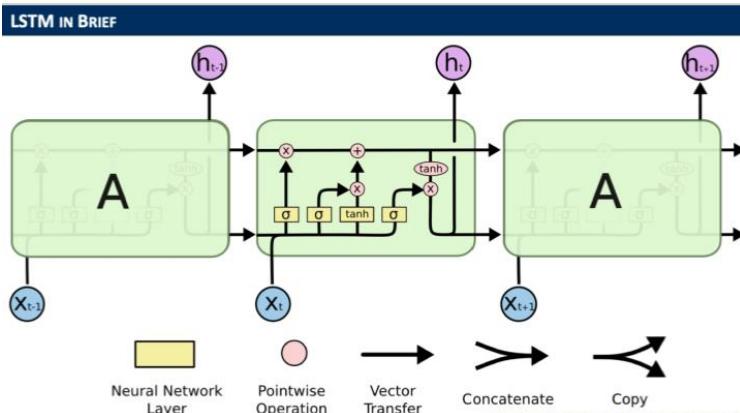
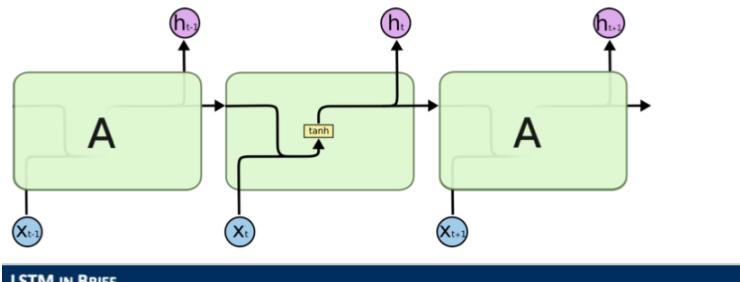
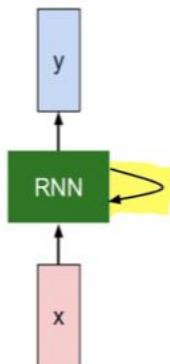
Vediamo degli esempi di Recursive Neural Network (la parte ricorsiva è evidenziata in verde).



We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

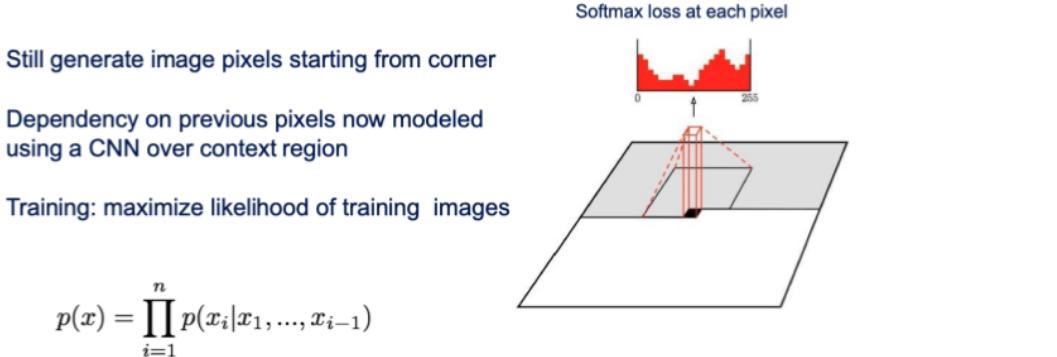
$$h_t = f_W(h_{t-1}, x_t)$$

new state old state input vector at some time step
 some function with parameters W



PixelCNN

Andiamo ancora a generare i pixel a partire da un angolo, ma per valutare i precedenti usiamo una **CNN**, sulla regione di interesse.



Il training è più veloce rispetto al PixelRNN perché in CNN si possono parallelizzare le convoluzioni dato che i valori del context region (i filtri ??) sono conosciuti dalle training images.
La generazione è comunque lenta perche deve procedere sequenzialmente.

Valutiamo pro e contro di PixelRNN e PixelCNN:

Pros:

- Can explicitly compute likelihood $p(x)$
 - Explicit likelihood of training data gives good evaluation metric
 - Good samples
- Improving PixelCNN performance
 - Gated convolutional layers
 - Short-cut connections
 - Discretized logistic loss
 - Multi-scale
 - Training tricks
 - Etc...
 - See

Con:

- Sequential generation => slow
 - Van der Oord et al. NIPS 2016
 - Salimans et al. 2017 (PixelCNN++)

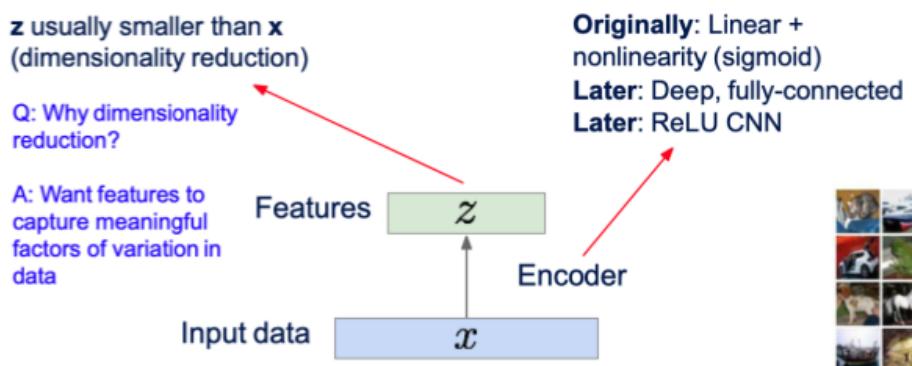
Autoencoder

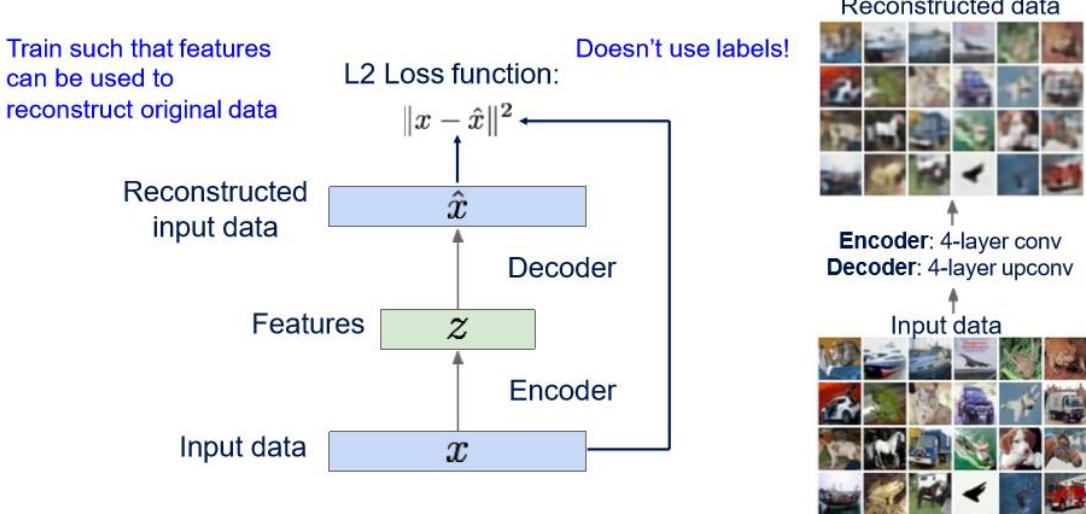
Un autoencoder neural network è un algoritmo di learning unsupervised, applica backpropagation e che mette i valori di output uguali a quelli di input.

La funzione di identità sembra particolarmente difficile da imparare, ma inserendo dei vincoli alla rete come limitare il numero di hidden units, possiamo trovare strutture di dati interessanti.

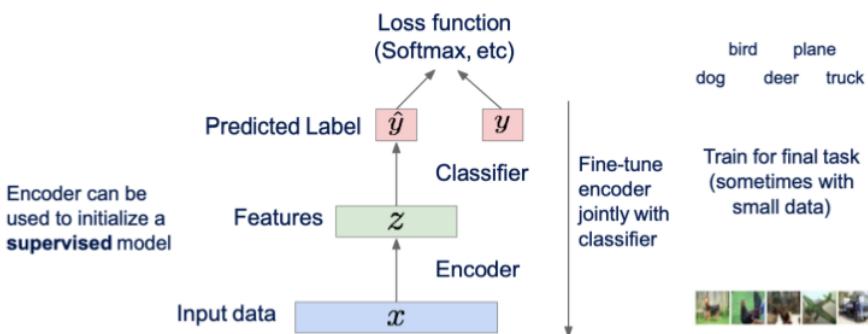
Vediamo qualche informazione sugli autoencoder prima di andare avanti.

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data





Possiamo usare una CNN per ricostruire le immagini di input, z contiene le feature estratte e con il decoder si ricostruiscono le immagini. Si usa come loss function la distanza euclidea a quadrato. Durante il training vogliamo che le immagini di output siano uguali a quelle di input. Dopo la fase di training possiamo eliminare il decoder e utilizzare la parte rimanente ad esempio per la classificazione. Gli autoencoder possono ricostruire i dati e fare da base per dei modelli supervised.

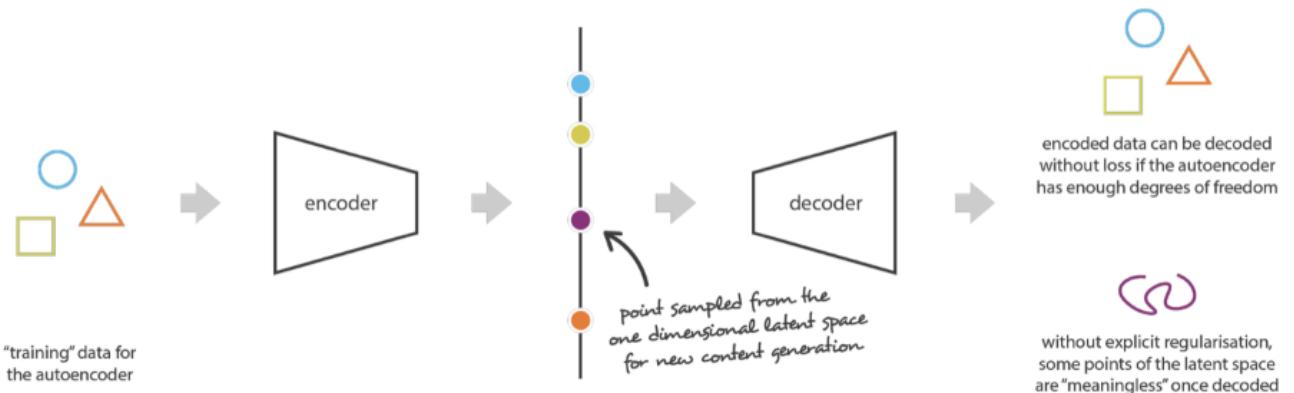


Gli autoencoder non sono utili se si vogliono generare delle nuove immagini a partire da una z randomica.

Variational autoencoder (VAE)

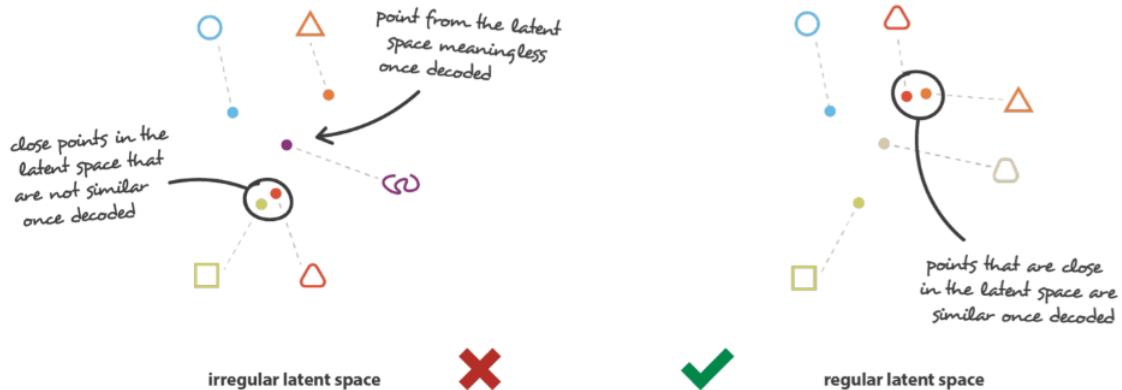
Possiamo generare delle immagini a partire da z come mostrato nella figura sottostante. Si possono generare cerchi, quadrati e triangoli senza problemi dato che erano presenti nel training set. Quando proviamo ad applicare il decoder al punto viola i risultati possono essere molto brutti in quanto nessuna immagine del training set è stata encodata in quel punto in z .

We can generate new data by decoding points that are randomly sampled from the latent space. The quality and relevance of generated data depend on the regularity of the latent space.



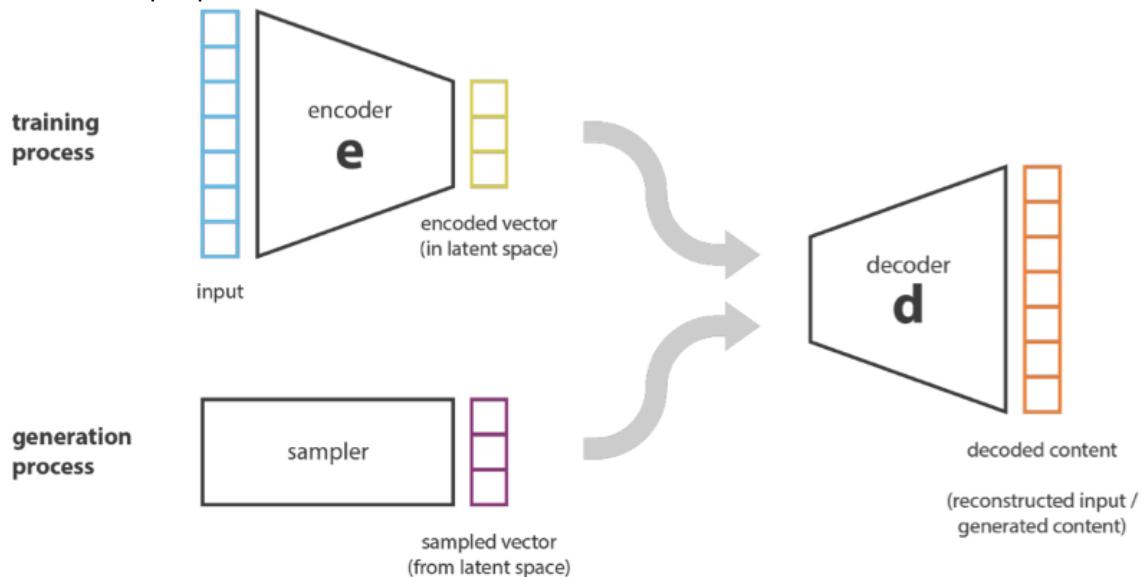
Questo è dovuto al fatto che il latent space spesso non è omogeneo in quanto ci sono punti che se tradotti non producono un output significativo e ci sono punti vicini che non rappresentano output simili.

Vorremmo un latent space più regolare come quello di destra nell'immagine seguente, per passare dal quadrato al triangolo abbiamo il punto grigio che rappresenta una via di mezzo fra i due.



L'autoencoder è stato trainato per avere in uscita un'immagine buona dato in input la stessa immagine, se invece noi prendiamo un sample randomicamente da z questo potrebbe non corrispondere a nessuna immagine che era presente nel training set, risultando quindi nel punto viola e in una conseguente traduzione senza significato.

L'idea è quella che durante in training set si usi un sampler che mantenga la struttura dei dati in modo da non avere output privi di senso.



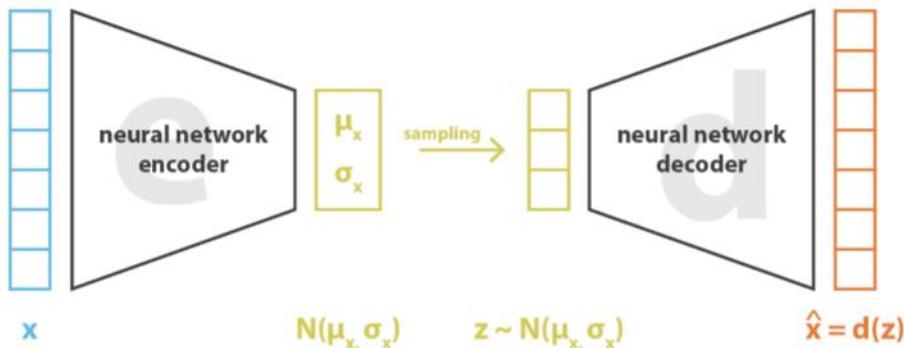
Vediamo matematicamente il perché gli autoencoder semplici non possono essere usati per la generazione di immagini.

Abbiamo già detto che negli autoencoder semplici dato un input riusciamo a trovare il punto nel latent space e successivamente riusciamo a tradurlo fornendo un output significativo. Questo non è garantito per ogni z , introduciamo quindi i variational autoencoder.

Con questo tipo di autoencoder il sample non viene preso direttamente da z , prima si costruisce la distribuzione del latent space e si prende il sample dalla distribuzione calcolata. Dall'esterno il comportamento sembra lo stesso dal momento che diamo un input e si ricostruisce l'output, la differenza sostanziale sta nel fatto che z non viene calcolata direttamente dall'input ma viene prese tramite la distribuzione del latent space. Nella figura seguente possiamo capire al meglio questo concetto.



La distribuzione di probabilità che si calcola è di tipo gaussiano e la struttura dei variational autoencoder è la seguente. Forziamo il latent space ad essere di tipo gaussiano, il modello deve imparare i parametri relativi alla distribuzione gaussiana.



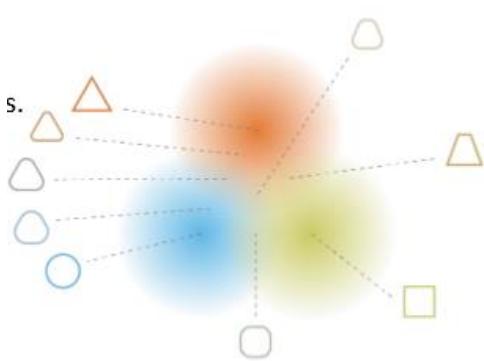
$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

Il secondo membro della loss function riguarda i parametri della distribuzione e $N(0, I)$ è il fattore di regolarizzazione.

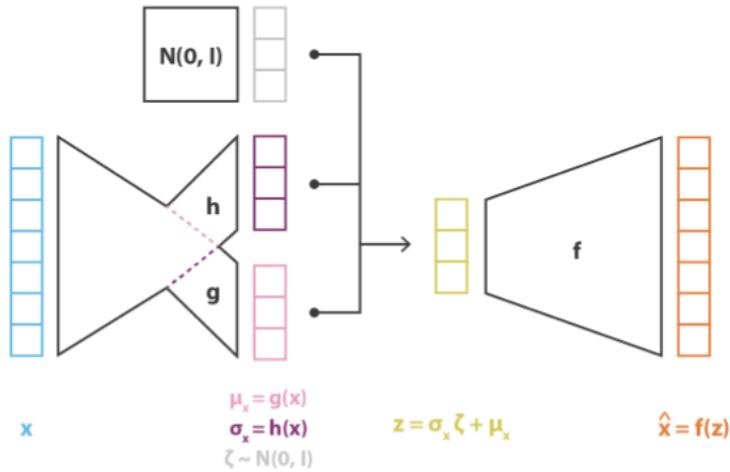
La regolarizzazione è fondamentale per ottenere un latent space con le giuste caratteristiche, senza otterremmo uno spazio come quello di sinistra dove tutte le distribuzioni gaussiane sono distanti fra di loro e ci sono sempre dei punti senza traduzione. Con la regolarizzazione lo spazio ottenuto diventa quello di destra.



Senza regolarizzazione formiamo un gradiente attorno all'informazione encodata che permette di avere vicini i punti di immagini simili, rimangono però punti vuoti nello spazio. Per evitare questo si devono cambiare ma **matrice di covarianza** e la **media** che sono i valori che caratterizzano la gaussiana. In questo modo forziamo le distribuzioni ad essere simili ad una distribuzione standard. La matrice di covarianza dovrà essere vicina alla matrice identica in modo da evitare punctual distribution e la media dovrà essere vicina a 0 in modo da evitare che le distribuzioni siano troppo lontane fra di loro. Otterremo quindi in risultato simile a quello mostrato in figura.



Nell' immagine seguente è mostrata la struttura di un autoencoder con la regolarizzazione.

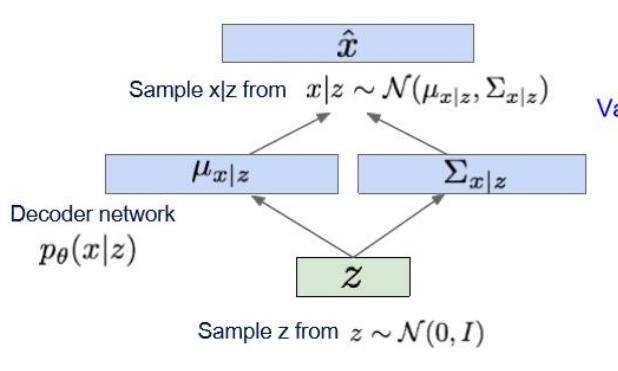


Per riassumere il concetto:

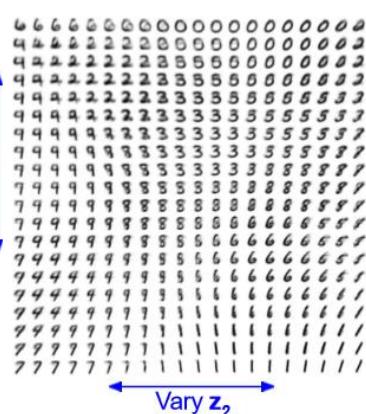
- Gli autoencoders semplici producono un singolo punto in z per ogni inpute questo porta a un latent space non adatto ad essere campionato randomicamente.
- Con i variational autoencoder abbiamo un autoencoder non forzisce direttamente z , ma fornisce i parametri per una distribuzione gaussiana. Successivamente durante il training si forzano le rappresentazioni ad essere più vicine possibile ad una gaussiana standard e a non essere distanti fra di loro.

Vediamo come funziona in pratica un variational autoencoder. In pratica l'encoder, dato l'inpute, crea un latent space che può essere campionato, nell'immagine seguente vediamo come l'encoder abbia creato lo spazio con due variabili.

Use decoder network. Now sample z from prior!



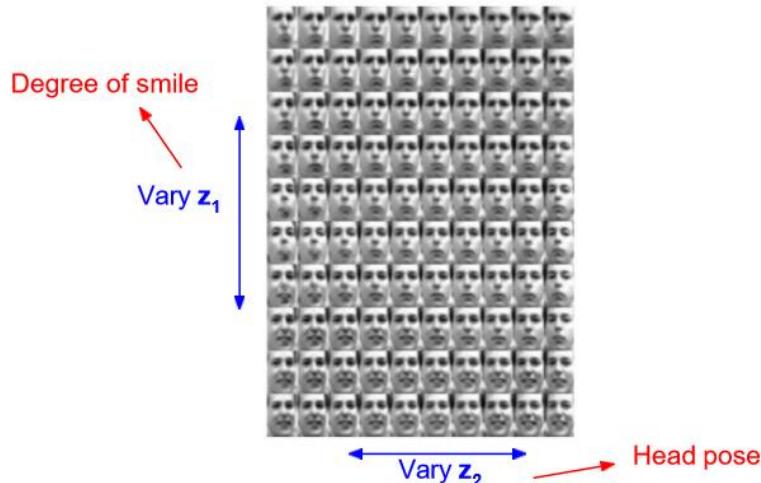
Data manifold for 2-d z



Il Sistema è forzato a creare un latent space nel quale ovunque mi sposti possa trovare delle immagini sensate, un altro esempio è il seguente. Le due variabili prese in considerazione sono il livello di sorriso e la posa della testa, possiamo vedere che tutte le immagini nello spazio sono significative e quindi possiamo effettuare un sample randomico senza preoccuparci che il risultato abbia senso.

Diagonal prior on \mathbf{z}
 \Rightarrow independent latent variables

Different dimensions of \mathbf{z} encode interpretable factors of variation



Vediamo i pro e i contro riguardo all'utilizzo dei variational autoencoder.

Probabilistic spin to traditional autoencoders \Rightarrow allows generating data

Defines an intractable density \Rightarrow derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Allows inference of $q(\mathbf{z}|\mathbf{x})$, can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

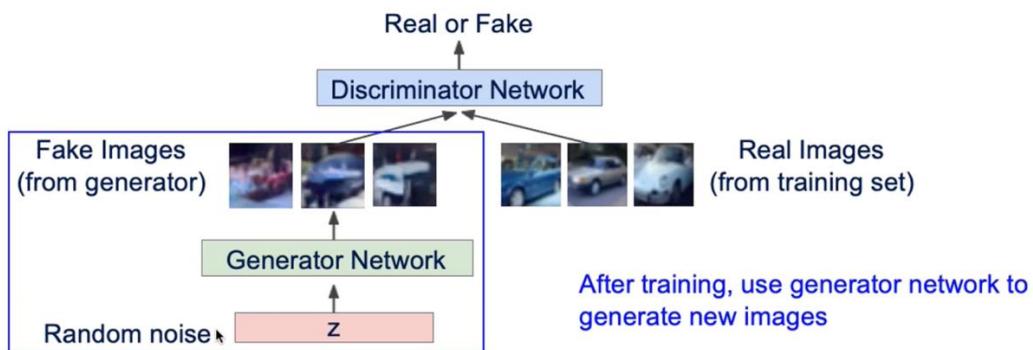
- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs)
- Incorporating structure in latent variables, e.g., Categorical Distributions

Generative Adversarial Networks (GAN)

Andiamo ad introdurre due tipi di reti:

- **Generator Network:** prova ad ingannare il discriminatore generando immagini che sembrano vere
- **Discriminator Network:** prova a distinguere un'immagine reale da una falsa.

Se facciamo backpropagation dall'uscita del discriminatore al generatore la rete si adatterà a creare immagini che sono sempre più simili a quelle reali in modo da ingannare il discriminatore. Ogni volta il generatore crea immagini più reali, il discriminatore poi se ne accorgerà e, tramite backpropagation, porterà la rete a creare img ancora migliori fino a che non saranno più distinguibili.



Fino ad ora abbiamo visto PixelCNN e VAE

PixelCNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i|x_1, \dots, x_{i-1})$$

VAEs define intractable density function with latent z :

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density,
and just want ability to sample?

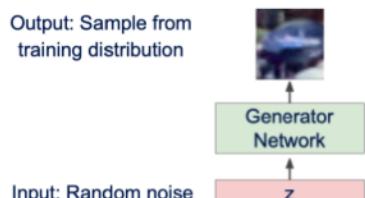
GANs: don't work with any explicit density function!

Instead, take game-theoretic approach:

learn to generate from training distribution through 2-player game

Problema: Vogliamo fare sampling da un complesso, high-dimensional distribuzione del training set. Non c'è un modo diretto per farlo!

Soluzione: Sampling da una semplice distribuzione, ad esempio il random noise. Impariamo la trasformazione dalla distribuzione del training.



Come possiamo rappresentare questa complessa trasformazione? Tramite una rete neurale.

Vediamo ora la loss function, abbiamo un **gioco a due** tra il generatore e il discriminatore

Train jointly in **minimax game**

Discriminator outputs likelihood in (0,1) of real image

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)}) \right]$$

- Il discriminatore (θ_d) vuole **massimizzare l'obiettivo** in modo che $D(x)$ sia vicino a 1 (reale) e $D(G(z))$ sia vicino a 0 (falso).
- Il generatore (θ_g) vuole **minimizzare l'obiettivo** in modo che $D(G(z))$ sia vicino a 1 (discriminatore viene ingannato pensando che il $G(z)$ generato sia reale).

Queste prox slides messe ma non dovrebbero essere chieste.

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. Gradient ascent on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

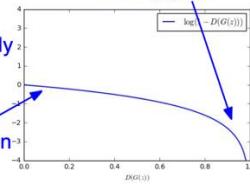
Gradient signal dominated by region where sample is already good

2. Gradient descent on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator. But gradient in this region is relatively flat!



Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Aside: Jointly training two networks is challenging, can be unstable. Choosing objectives with better loss landscapes helps training, is an active area of research.

Alternate between:

1. Gradient ascent on discriminator

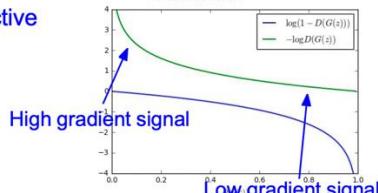
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. Instead: Gradient ascent on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Putting it together: GAN training algorithm

```

for number of training iterations do
    for k steps do
        • Sample minibatch of m noise samples { $z^{(1)}, \dots, z^{(m)}$ } from noise prior  $p_g(z)$ .
        • Sample minibatch of m examples { $x^{(1)}, \dots, x^{(m)}$ } from data generating distribution  $p_{data}(x)$ .
        • Update the discriminator by ascending its stochastic gradient:
            
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$


```

Some find $k=1$
more stable,
others use $k > 1$,
no best rule.

Recent work (e.g.
Wasserstein GAN)
alleviates this
problem, better
stability!

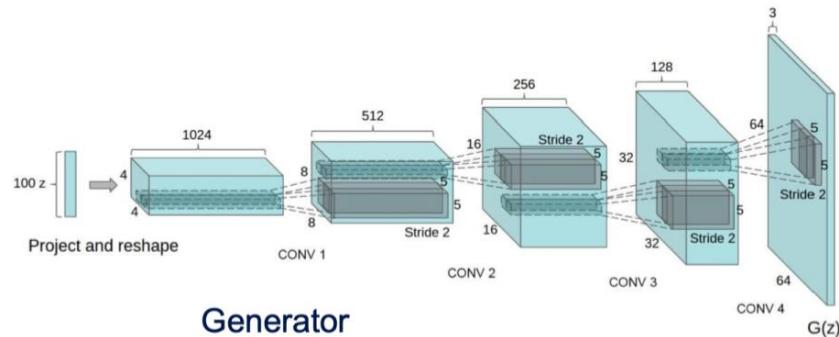
```

        • Sample minibatch of m noise samples { $z^{(1)}, \dots, z^{(m)}$ } from noise prior  $p_g(z)$ .
        • Update the generator by ascending its stochastic gradient (improved objective):
            
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

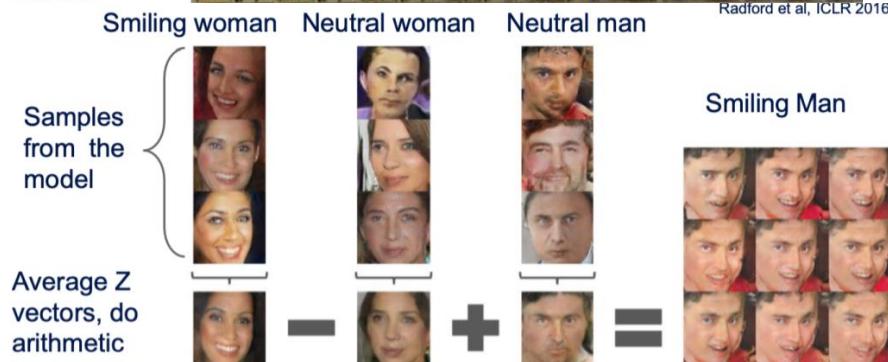
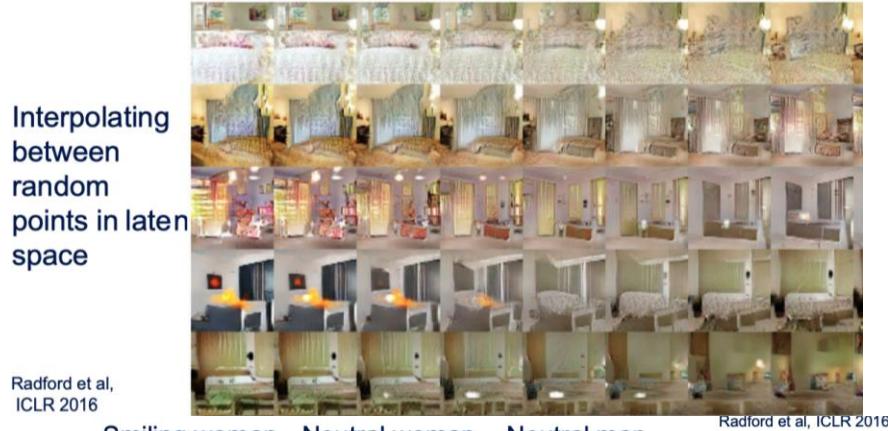

```

end for

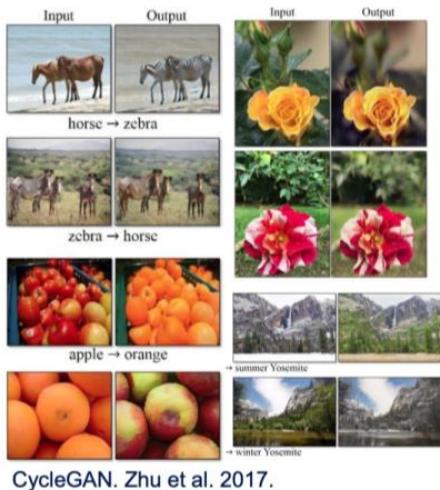
Dall'ultima slide, è importante che il discriminatore sia buono altrimenti, se è facile ingannarlo, il generatore non creerà immagini buone. Se invece abbiamo un buon discriminatore sarà più difficile ingannarlo e il generatore creerà immagini che assomiglano sempre di più a quelle reali.



Esempi



Source->Target domain transfer



Text -> Image Synthesis

this small bird has a pink breast and crown, and black primaries and secondaries. this magnificent fellow is almost all black with a red crest, and white cheek patch.



Many GAN applications



Pix2pix. Isola 2017. Many examples at <https://phillipi.github.io/pix2pix/>

Don't work with an explicit density function

Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:

- Beautiful, state-of-the-art samples!

Cons:

- Trickier / more unstable to train
- Can't solve inference queries such as $p(x)$, $p(z|x)$

Active areas of research:

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

Generative Models

- PixelRNN and PixelCNN Explicit density model, optimizes exact likelihood, good samples. But inefficient sequential generation.
- Variational Autoencoders (VAE) Optimize variational lower bound on likelihood. Useful latent representation, inference queries. But current sample quality not the best.
- Generative Adversarial Networks (GANs) Game-theoretic approach, best samples! But can be tricky and unstable to train, no inference queries.

NB: Una volta che abbiamo generato una fake image non possiamo tornare indietro ed ottenere z .