

Tecniche di Deep Learning per il riconoscimento di oggetti in robotica

Sistemi Intelligenti Robotici

Giacomo Bartoli
mat. 795161

14 Giugno 2018

1 Introduzione

Negli ultimi dieci anni le tecniche di Deep Learning si sono dimostrate in grado di superare lo stato dell'arte in molti problemi di classificazione tipici dell'Intelligenza Artificiale. Alla base di queste tecniche vi sono reti neurali artificiali composte da multipli livelli e con migliaia di neuroni per ogni singolo layer:

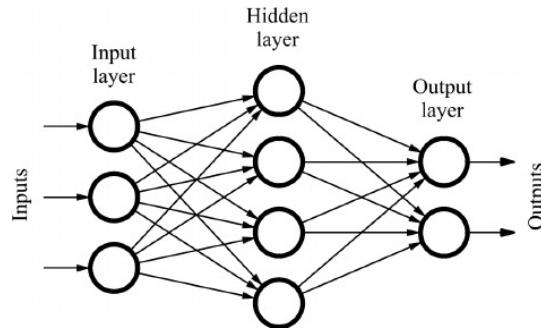


Figure 1: Esempio di rete neurale artificiale

Una rete neurale artificiale è composta da una serie di neuroni, in figura rappresentati come cerchi, che prendono in ingresso i pesi degli archi provenienti dai neuroni del livello precedente, ne fanno la somma pesata, applicano una funzione matematica al risultato e propagano l'output al livello successivo. Una rete con queste caratteristiche viene detta *feedforward* perché l'output va sempre verso il livello successivo. Altra caratteristica della rete mostrata in figura è quella di essere *fully connected*: ogni neurone propaga il risultato della sua computazione verso ogni neurone del livello successivo. Una rete che è sia feed-forward che fully connected prende il nome di *Multilayer Perceptron* (MLP).

Il cuore degli algoritmi di Deep Learning consiste proprio nel trovare i giusti pesi agli archi della rete affinché la somma degli errori dei neuroni di output sia minimizzata. Per raggiungere questo obiettivo è necessario capire:

- quale funzione utilizzare in ogni neurone (funzione d'attivazione).
- quale funzione utilizzare per calcolare la somma degli errori (loss function).
- quale algoritmo utilizzare per trovare i giusti pesi della rete.

In realtà, non esiste una sola soluzione ma, come ogni problema di ottimizzazione, la soluzione cambia al variare del dominio applicativo.

Tuttavia, in generale si può affermare che:

- la funzione di attivazione che si utilizza più spesso è la *Rectified Linear Unit* (ReLU), una approssimazione della funzione sigmoide o tangente iperbolica. La ReLU è continua in R, costante fino a zero e poi assume andamento lineare:

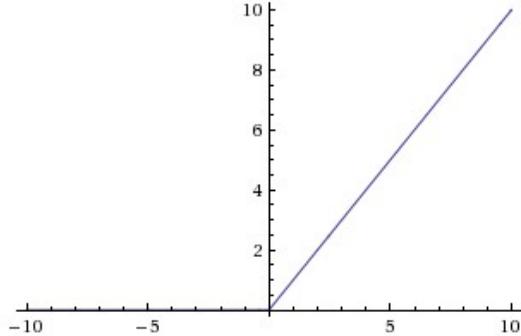


Figure 2: Rectified Linear Unit

La funzione ReLU risulta quindi di facile implementazione ed anche di basso costo computazionale, fattore da non trascurare a fronte di una rete con multipli strati e migliaia di neuroni per ogni strato.

- E' possibile calcolare l'errore della rete semplicemente confrontando i valori di output con i valori desiderati. Infatti l'apprendimento è supervisionato.
- Come tutti gli algoritmi di Machine Learning, il dataset viene suddiviso in training, test e validation set. Il training set viene utilizzato per trovare i parametri della rete, ovvero i pesi sugli archi che collegano neuroni di diversi livelli. Il test set viene utilizzato per valutare le prestazioni della rete. Se la rete ha ottime performance sul training set, ma non sul test set significa che la rete generalizza troppo (*overfitting*). Infine, il validation set viene utilizzato per tarare eventuali iperparametri, ovvero tutti quei

parametri che non siano i pesi degli archi. Il metodo più utilizzato per trovare i giusti pesi è il l'*algoritmo di retropropagazione*, che consiste nel ripercorrere a ritroso la rete (dal livello di output fino a quello di input) e risettare i pesi sugli archi, che in principio sono inizializzati in maniera casuale. Ad ogni passo i nuovi pesi vengono calcolati sfruttando la *discesa del gradiente*, nota tecnica di ottimizzazione che consente di trovare minimi locali di funzioni a più variabili, e quindi minimizzare l'errore, a fronte di un ulteriore parametro chiamato *Learning Rate*. Quest'ultimo parametro determina la velocità di convergenza del gradiente: se il learning rate è troppo alto l'algoritmo converge velocemente ma rischia di fare overshooting, ovvero di non trovare il minimo locale e continuare ad oscillare tra più possibili soluzioni. Viceversa, se il learning rate è troppo piccolo, l'algoritmo di discesa del gradiente rischia di convergere fin troppo lentamente. L'apprendimento è quindi un problema di ottimizzazione molto complesso perché il numero di pesi in gioco è troppo alto. Allo stato dell'arte sono milioni di parametri.

2 Reti Neurali Convoluzionali

Le reti neurali artificiali, in particolare il modello MLP, sono estremamente potenti perché secondo il *Universal Approximation Theorem* sono in grado di approssimare qualsiasi funzione. Tuttavia, non si applicano bene alle immagini. Infatti, le immagini sono array 2D strutturati a griglia. A fronte di qualche milione di pixel i parametri esplodono. In più le immagini hanno dei pattern che si ripetono trasversalmente. E' possibile sfruttare la presenza dei pattern per dare gli stessi pesi e rendere il problema più trattabile. Nel cervello succede la stessa cosa: i dati che arrivano attraverso la retina alla corteccia visiva primaria (v1) passano attraverso strati neuronali che creano queste features gerarchiche. L'intuizione nasce da un esperimento di D.Wiesel e T.Hubel, che risale al 1962, in cui i due ricercatori si accorsero che i gatti, e in maniera analoga gli umani, sono dotati di due tipi di cellule: le cellule semplici e quelle complesse [3]. Le cellule semplici si eccitano al riconoscimento di piccoli e semplici pattern, mentre quelle complesse aggregano l'informazione proveniente dalle cellule semplici per riconoscere pattern più generici. C'è una gerarchia tra semplici e complesse che si ripete. Tramite questa ripetizione più si va in là nella gerarchia più il pattern a cui le cellule si eccitano è sempre più complesso.

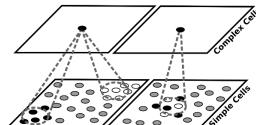


Figure 3: Gerarchie tra cellule semplici e complesse

Le reti neurali convoluzionali (CNN) [2] si basano esattamente sulla stessa

rappresentazione gerarchica. Tuttavia, al posto di utilizzare una rete fully connected si utilizza una *convoluzione locale con filtro* per ogni posizione dell'immagine. L'operazione di convoluzione si traduce in un filtro che viene passato in ogni zona della immagine di input della nostra CNN. Questa operazione rappresenta esattamente la cellula semplice di Wiesel e Hubel.

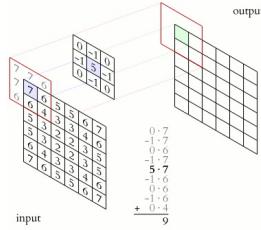


Figure 4: Convoluzione con filtro

La cellula complessa è rappresentata invece dalla operazione di *pooling*: prende un insieme di cellule semplici ed applica una operazione che tipicamente è il massimo o la media per aggregare informazioni. L'idea è quella di presentare una sorta di 'riassunto' al livello successivo, per questo si usa il massimo o la media.

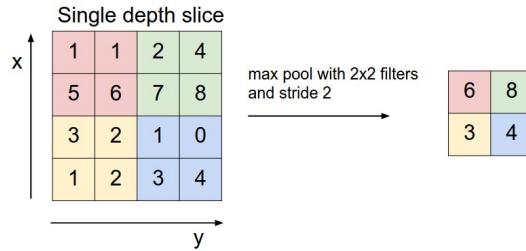


Figure 5: Pooling con operazione di massimo

Le CNN applicano in serie uno strato convoluzionale ed uno strato di pooling. La funzione di attivazione utilizzata tipicamente è la ReLU. Alla fine c'è sempre uno strato fully connected che è il classificatore e quindi classifica le feature apprese. Il risultato è una rete neurale dove i pesi sono condivisi e le connessioni al livello successivo sono locali. Di conseguenza, il numero di pesi da trovare è decisamente minore rispetto ad una MLP pertanto il problema risulta facilmente trattabile. Ad esempio, data una rete a due livelli con 6 neuroni sul primo e 3 sul secondo:

- MPL: 18 pesi da calcolare
- CNN: 3 pesi da calcolare

Alternando convoluzione e pooling le CNN si dimostrano ottime per problemi di classificazione su immagini.

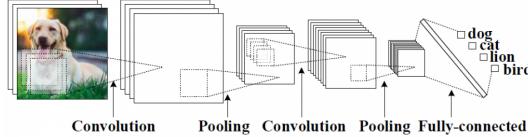


Figure 6: Reti neurali convoluzionali

Fino al 2012 i metodi di Machine Learning con features handcrafted rappresentavano lo stato dell’arte. Nello stesso anno Yann LeCun fece esplodere il Deep Learning tramite benchmark su ImageNet dopo essere stato escluso dalla comunità scientifica nel 1998 per essere stato battuto da SVM di Vapnik. Da lì in poi i successivi sviluppi hanno ridotto in maniera considerevole gli errori per i dataset di riferimento fino a battere le performance umane nei task di classificazione e detection. Per molti task il Deep Learning batte l’uomo, per questo oggi suscita molto interesse. Al momento le principali applicazioni riguardano la Computer Vision e Speech Recognition. E’ facile intuire quanto l’abilità di classificare correttamente un oggetto sia un task fondamentale per un robot. Ciò nonostante la complessità del problema aumenta secondo le seguenti casistiche:

- L’immagine potrebbe contenere più di un soggetto da classificare.
- Non mi basta classificare più soggetti, ma devo anche rilevare in quale zona dell’immagine si trovino (detection).
- In scenari particolarmente critici il tutto potrebbe accadere in real-time, quindi i tempi di detection e classificazione assumono ancora più rilevanza.

3 Tassonomia dei problemi

Prima ancora di proporre modelli di reti neurali più evoluti per assolvere i problemi elencati nel paragrafo precedente è necessario specificare la terminologia utilizzata in seguito:

- *Classificazione*: data una immagine in input identificare il pattern ricorrente.
- *Localization*: identificare la regione in cui si trova un oggetto. In particolare fare localization significa inscrivere il soggetto in un quadrilatero di cui vogliamo conoscere le coordinate. Pertanto il task si riduce ad un problema di regressione, dato un input vogliamo in output una tupla di numeri $[x_0, y_0, \text{width}, \text{height}]$.
- *Detection*: classificazione e localization assolvono la funzione di detection.

4 R-CNN

Lo scopo delle R-CNN (Region CNN) [5] è proprio quello di identificare gli oggetti principali data una immagine in input. Dunque, la prima cosa che viene fatta è quella di identificare le regioni dove potenzialmente potrebbero trovarsi gli oggetti da identificare e successivamente dare in pasto alla CNN per la fase di identificazione. Questa fase, detta *Regions Proposal Phase*, viene computata attraverso un algoritmo chiamato *Selective Search* [4]. Tale algoritmo considera l'immagine attraverso finestre di dimensioni diverse, e per ogni dimensione cerca di raggruppare i pixel adiacenti per trama, colore o intensità:

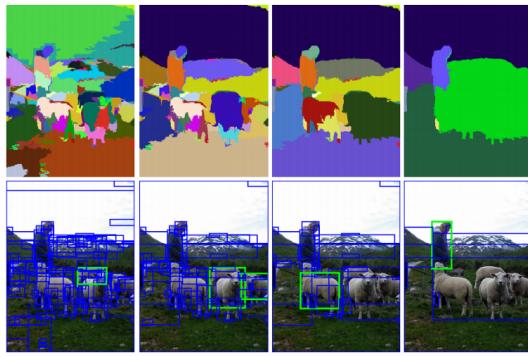


Figure 7: Selective Search

Il risultato di questa fase sono delle *Regions of Interest*, ovvero aree della immagine di partenza che contengono i soggetti su cui fare detection. Ogni Region of Interest viene aggiustata per fare in modo che l'input della Region of Interest corrisponda con quello della CNN (fase di *warping*).

Ogni immagine viene processata dalla CNN, tipicamente AlexNet, e sull'ultimo layer, quello fully connected, si applica SVM per il task di classificazione e Bounding Box Regression per il task di localization. Le regioni iniziali proposte dall'algoritmo di Selective Search potrebbero non essere perfettamente centrate rispetto al soggetto che contengono. Proprio per questo, l'ultimo step di Bounding Box Regression è un ulteriore raffinamento per fare in modo che il soggetto sia centrato rispetto alle coordinate del quadrilatero in cui è inscritto. Pertanto possiamo considerare le R-CNN delle CNN dove l'input viene suggerito dall'algoritmo di Selective Search e l'ultimo layer assolve il task di classificazione usando SVM. E' evidente che le R-CNN soffrono di un problema critico: ogni Region of Interest diventa l'input di una CNN, quindi il numero delle CNN utilizzate è pari al numero di Region of Interest. Ciò rende il problema intrattabile per applicazioni real-time, dove il tempo di detection deve essere quasi istantaneo.

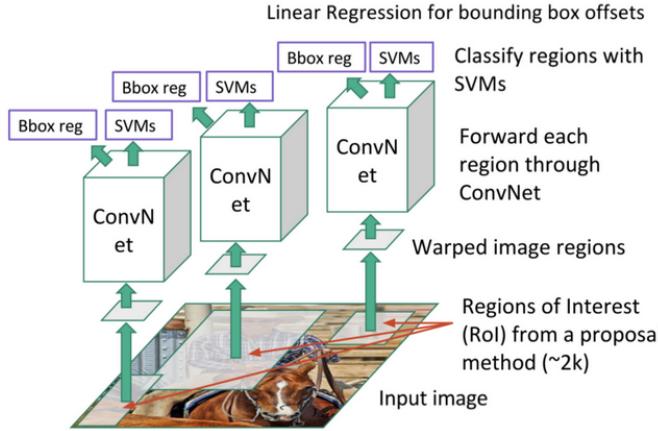


Figure 8: Architettura delle R-CNN

5 Fast R-CNN

Le Fast R-CNN [7] sono un ulteriore miglioramento delle R-CNN. La differenza sostanziale consiste nell'anteporre un *RoI Pooling layer* tra le CNN e l'ultimo livello fully connected. Mentre con le R-CNN i componenti su cui fare il training erano rispettivamente CNN, SVM e Bounding Box, le Fast R-CNN risultano migliori perché viene utilizzato un solo componente (il RoI Pooling layer) per fare il training. Il RoI Pooling layer riceve in input sia le Region Proposals, ottenute ancora dall'algoritmo di Selective Search, sia l'ultimo layer della CNN. L'output risultante sono vettori che hanno la stessa dimensione che vengono processati dal fully connected layer. In maniera analoga alle R-CNN classiche, si fa classificazione e regressione sulla base di quest'ultimo livello, con la differenza che il task di classificazione è affidato non più ad SVM ma all'algoritmo *Soft-Max*, che restituisce una confidenza. La parte di regressione rimane invariata, utilizzando Bounding Box Regressor [6].

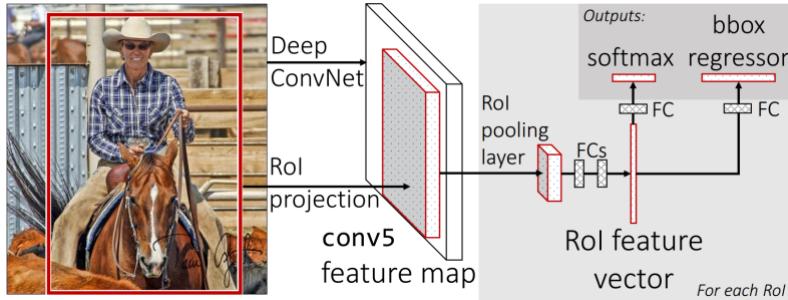


Figure 9: Architettura delle Fast R-CNN

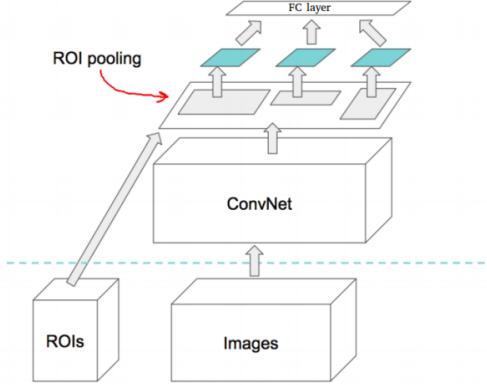


Figure 10: RoI Pooling Layer

Analizzando l’architettura di una rete Fast R-CNN risulta chiaro che tutto l’input e l’output necessario per i task di classificazione e localization proviene da una unica rete, che dunque si dimostra molto più performante di una classica R-CNN, dove per ogni oggetto corrisponde una CNN.

6 Faster R-CNN

Anche con tutti questi progressi c’è ancora una nota dolente che permane nelle (Fast) R-CNN: la fase di Region Proposal. Come abbiamo visto, il primo passo per scoprire le posizioni degli oggetti è generare un sacco di potenziali regioni di interesse da testare. L’algoritmo di Selective Search [4] è abbastanza lento e risulta essere il collo di bottiglia dell’intero processo. Le *Faster R-CNN* [8] si distinguono proprio perché sono in grado di fare classificazione e region proposal grazie ad una unica CNN. Si noti infatti che le RoI dipendono fortemente dalle features estratte dall’immagine, che sono calcolate anche dai primi livelli della CNN. Perché non sfruttare allora gli stessi risultati della CNN per le region proposal invece di eseguire la Selective Search in modo separato? In questo modo otteniamo le regioni di interesse in maniera quasi gratuita, l’unica accortezza è quella di addestrare la CNN. Le Faster R-CNN consistono in due moduli:

- RPN (Region Proposal Network): dato l’input del layer convoluzionale trova i rettangoli per la localization. I rettangoli vengono individuati solamente se all’interno vi è un soggetto rilevante.
- RoI pooling layer: classifica ogni proposal ed applica un fattore di correzione affinché il soggetto sia centrato rispetto al rettangolo nel quale è inscritto.

Il *Region Proposal Network* (RPN) è decisamente l’elemento che rende le Faster R-CNN più interessanti rispetto agli approcci visti fin’ora. Nello specifico l’RPN

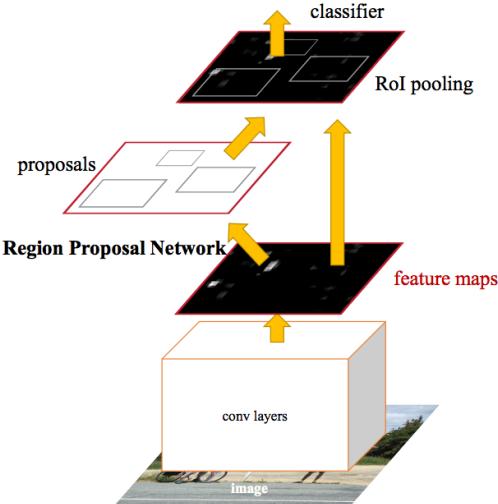


Figure 11: Architettura Faster R-CNN

utilizza una sliding window che viene fatta scorrere sulla feature map e classifica ciò che si trova al di sotto della sliding window come oggetto/non oggetto proponendo un bounding box nel primo caso. Tuttavia, sappiamo che questi bounding boxes avranno proporzioni diverse a seconda dell'oggetto che è stato identificato. Ad esempio, una persona sarà incorniciata in un rettangolo dove l'altezza sarà decisamente maggiore della larghezza. Viceversa, un monitor tv sarà identificato da un rettangolo con proporzione 4:3. L'RPN si occupa anche di trovare dei rettangoli con le adeguate proporzioni. Per fare questo vengono proposti una serie di *anchor boxes* con associati dei valori di confidenza. Gli anchor boxes il cui valore di confidenza cade sotto una certa soglia vengono scartati, mentre i restanti vengono passati al RoI Pooling layer per la fase di classificazione. Per ulteriori dettagli su come l'RPN calcola gli anchor boxes si rimanda al paper originale (vedi bibliografia).

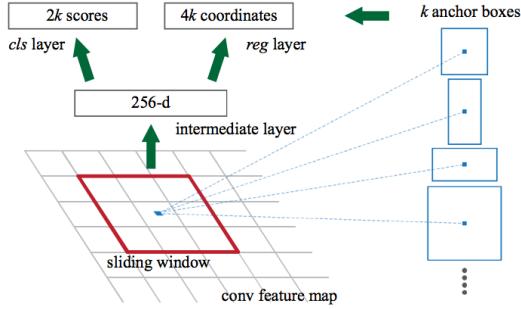


Figure 12: Region Proposal Network (RPN)

7 YOLO (You Only Look Once)

I sistemi di rilevamento precedenti riutilizzano classificatori o localizzatori per fare detection. Applicano il modello a un’immagine in più posizioni e scale. Le regioni con un punteggio elevato dell’immagine sono considerate rilevanti. YOLO [9] utilizza un approccio completamente diverso. Una singola rete neurale viene applicata all’intera immagine. Questa rete divide l’immagine in regioni e calcola bounding boxes e probabilità per ognuna di esse. Questo approccio comporta diversi vantaggi rispetto ai sistemi basati su classificatori. Infatti, effettua previsioni con una singola valutazione della rete a differenza di sistemi come R-CNN che ne richiedono migliaia per una singola immagine.

Come è possibile fare tutto con una singola rete neurale? Il primo step del processing che YOLO esegue è quello di dividere l’immagine in input in una griglia, dove ogni cella è responsabile per la predizione dei bounding boxes ed i rispettivi valori di confidenza. Se il box contiene un oggetto il valore di confidenza associato sarà molto alto, viceversa se non contiene alcun oggetto allora la confidenza sarà molto bassa. Dopo che ogni cella ha calcolato questi valori si ottiene una immagine dove i bordi dei rettangoli che identificano oggetti sono più spessi in accordo con i loro valori di confidenza:



Figure 13: YOLO: immagine divisa in bouding boxes

Giunti a questo punto si sanno esattamente quanti oggetti rilevanti ci sono nell'immagine, ma non quali sono. Pertanto, ogni cella si occupa anche di calcolare la probabilità che l'oggetto appartenga ad una certa classe (class probability):

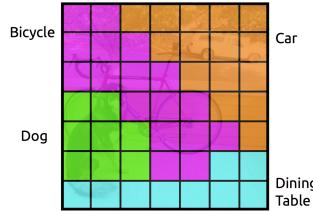


Figure 14: YOLO class probability

Si sta parlando di probabilità condizionata, ciò significa che il valore calcolato non rappresenta la probabilità di contenere quell'oggetto; significa che se quella cella contiene un oggetto allora la probabilità che quell'oggetto sia una bici, un cane o una macchina è pari al valore calcolato. Dopo di ché, la class probability viene moltiplicata per la probabilità condizionata per ottenere una griglia dove ogni bounding box considera sia la probabilità di contenere tale oggetto rapportata alla possibilità che effettivamente vi sia un oggetto all'interno:

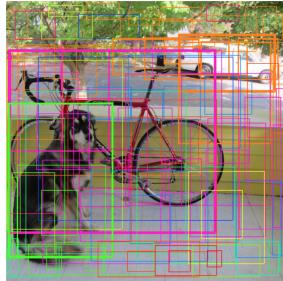


Figure 15: YOLO bounding box e probabilità condizionata

Si considerano poi solamente i rettangoli il cui valore di confidenza è superiore ad una certa soglia:

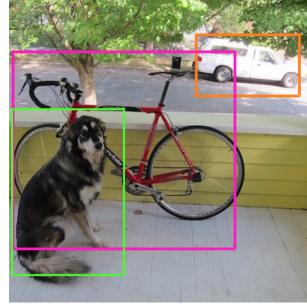


Figure 16: YOLO threshold detection

Nonostante alcuni errori di generalizzazione, YOLO rappresenta lo stato dell'arte negli algoritmi di object detection e la sua velocità nel rilevare oggetti all'interno di immagini rende la rete particolarmente duttile per risolvere task in tempo reale. L'architettura di YOLO è riassunta in figura 17:

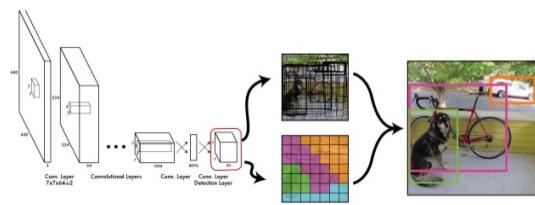


Figure 17: Architettura di YOLO

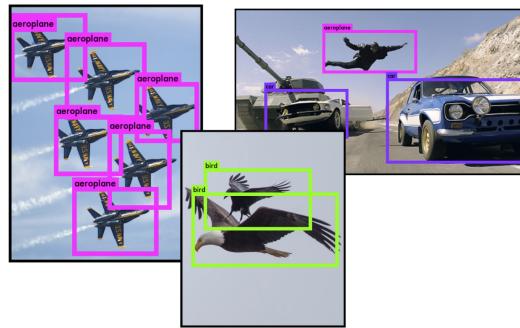


Figure 18: YOLO real-time detection

8 Benchmarks su dataset di riferimento

Testiamo le diverse architetture di Deep Learning appena introdotte su dataset Pascal 2007, che è il dataset di riferimento per i task di image classification ed object recognition considerato standard per la comunità scientifica. Consideriamo come criteri di qualità sia la precisione media (mAP - Mean Average Precision) che i tempi di detection:

	mAP	Speed
R-CNN	66	.05 FPS 20 s/img
Fast R-CNN	70	.5 FPS 2 s/img
Faster R-CNN	73	7 FPS 140 ms/img
YOLO	69	45 FPS 22 ms/img

Table 1: Benchmark su Pascal VOC 2007

Le reti Faster R-CNN si dimostrano le più precise in assoluto, ma processando solamente 7 frame al secondo e quindi ogni immagine impiega 140ms per essere identificata. YOLO risulta poco inferiore come prestazioni di precisione, ma considerando i tempi di detection si dimostra nettamente migliore. Allo stato dell'arte YOLO è il migliore sistema per fare object detection e la sua velocità lo rende particolarmente interessante per task real-time. Gli stessi autori di YOLO hanno poi introdotto nelle versioni successive [10] [11] la possibilità di parametrizzare il trade-off tra precisione e velocità, lasciando al progettista la libertà di scegliere il livello di precisione necessario a seconda del problema a cui si trova di fronte.

References

- [1] Materiale ed appunti dal corso di Sistemi Intelligenti Robotici, Università di Bologna, Prof. A.Roli
- [2] Yann LeCun, Yoshua Bengio, Geoffrey Hinton, *Deep Learning*, Nature, 2013
- [3] D. H. Hubel e T. N. Wiesel, *Receptive fields and functional architecture of monkey striate cortex*, The Journal of Physiology, vol 582 ed 1964
- [4] J.R.R. Uijlings, K.E.A. van de SandeT. GeversA. W.M. Smeulders, *Selective Search for Object Recognition*, International Journal of Computer Vision 2013
- [5] Girshick, Ross and Donahue, Jeff and Darrell, Trevor and Malik, Jitendra (UC Berkeley), *R-CNN for object detection*, 2014
- [6] Girshick, Ross and Donahue, Jeff and Darrell, Trevor and Malik, Jitendra, *Rich feature hierarchies for accurate object detection and semantic segmentation*, CVPR 2014
- [7] Ross Girshick, *Fast R-CNN*, International Conference on Computer Vision ICCV 2015
- [8] Shaoqing Ren and Kaiming He and Ross Girshick and Jian Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, Advances in Neural Information Processing Systems NIPS 2015
- [9] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, *You Only Look Once: Unified, Real-Time Object Detection*, CVPR 2016
- [10] Joseph Redmon, Ali Farhadi, *YOLO9000: Better, Faster, Stronger*, CVPR 2017
- [11] Redmon, Joseph and Farhadi, Ali, *YOLOv3: An Incremental Improvement*, arXiv, 2018