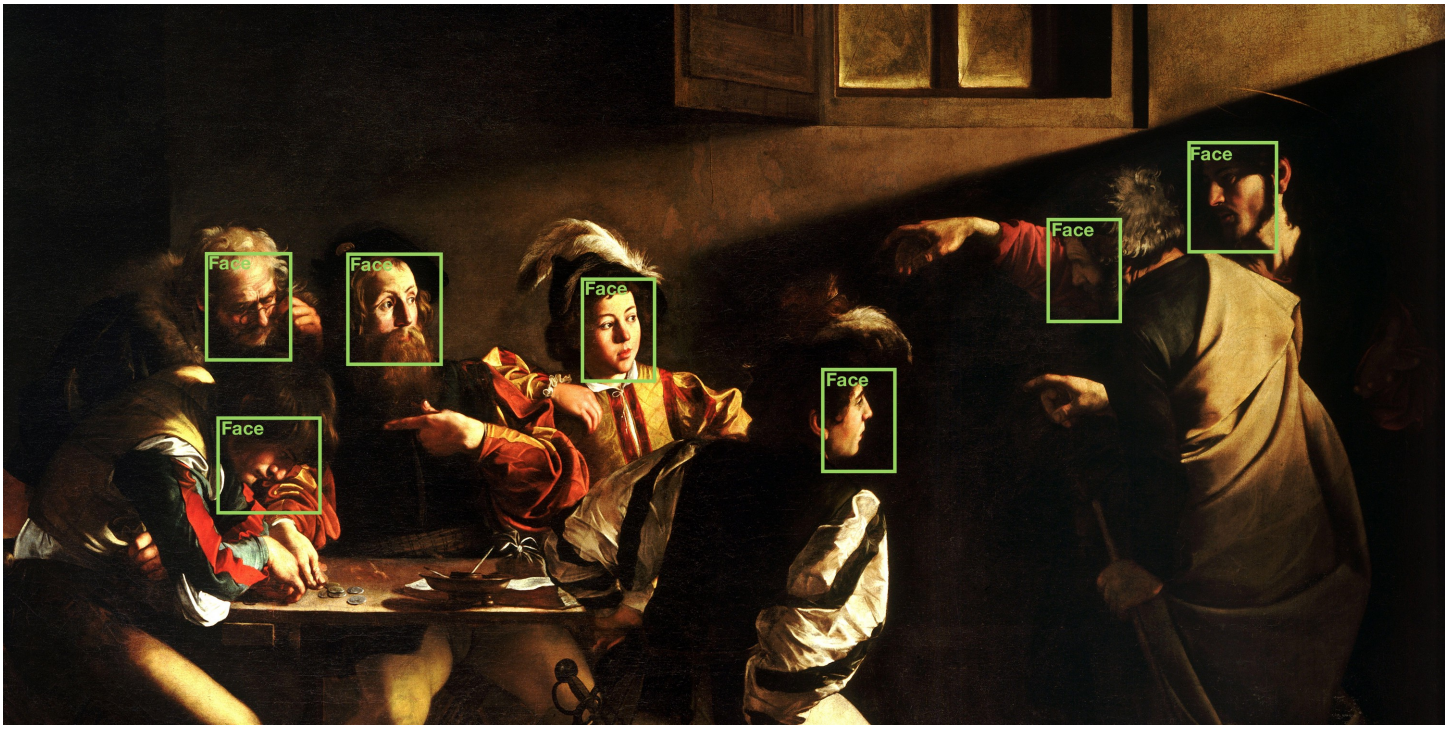


The Calling of St Matthew di Caravaggio, nel caso in cui ti piaccia l'arte :)

RetinaNet: come la perdita focale corregge il rilevamento a colpo singolo

Il rilevamento di oggetti è un campo estremamente importante nella visione artificiale necessaria per la guida autonoma, la videosorveglianza, le applicazioni mediche e molti altri campi. Hai sentito parole d'ordine come YOLO o SSD? In questo articolo, ti spiegherò alcuni importanti progressi recentemente compiuti nel rilevamento di oggetti all'avanguardia usando l'apprendimento profondo e ti guiderò verso grandi risorse che dimostrano come implementarli. Dopo aver letto questo articolo avrai una buona panoramica del campo, non sono richieste conoscenze preliminari nel rilevamento degli oggetti.

Fabio M. Graetz

25 nov 2018 · 17 minuti di lettura

Questo articolo è organizzato come segue:

Innanzitutto, ti spiego un modello semplice che è in grado di rilevare e classificare *un singolo* oggetto in un'immagine. Quindi, riassumo diversi metodi che sono stati utilizzati per rilevare e classificare *diversi* oggetti in un'immagine concentrandosi sul confronto tra i metodi *proposti per regione* e quelli *a scatto singolo* . Successivamente, spiego *in dettaglio* come funzionano i rilevatori come *SSD* o *YOLO* , perché sono più veloci ma anche meno precisi dei metodi di *proposta* di *regione* e infine come *RetinaNet,* recentemente sviluppato, risolve questi problemi.

. . .

Rilevamento di singoli oggetti

Le reti neurali possono essere utilizzate per risolvere *problemi di classificazione* (predire classi) e *problemi di regressione* (prevedere valori continui). Oggi faremo entrambi allo stesso tempo. Iniziamo con un'attività semplificata: rilevare e classificare *un* singolo oggetto in un'immagine anziché diversi oggetti.

Dati

Come appare un set di dati per il rilevamento di oggetti? Bene, gli input per il nostro modello sono ovviamente immagini e le etichette sono in genere quattro valori che descrivono un riquadro di delimitazione della verità di base più una categoria di cui l'oggetto in questo riquadro appartiene due.

Questi quattro valori potrebbero, per esempio, descrivono la x ed y coordinate dell'angolo inferiore sinistro e superiore destro del riquadro di delimitazione. Oppure le coordinate di un angolo specifico e l'altezza e la larghezza del rettangolo di selezione.

Architettura

La nostra rete neurale che prende l'immagine come input deve prevedere *quattro valori* che rappresentano le coordinate del riquadro di limitazione *previste* e che confronteremo con le quattro coordinate del riquadro di limitazione della *verità* di base (etichetta) .

Inoltre, la rete neurale deve prevedere una probabilità di classe per ciascuna delle n categorie che vogliamo classificare. Questo ci dà un totale di $4 + n$ valori che la rete deve prevedere

deve prevedere.

Per costruire il nostro *rivelatore a singolo oggetto* potremmo prendere qualsiasi rete di classificazione delle immagini convoluzionale come VGG-16 o ResNet (idealmente pre-addestrata) e rimuovere tutti i livelli di classificazione nella parte superiore della rete, appiattire l'output della rete di base troncata e aggiungere uno o due strati lineari che alla fine producono $4 + n$ valori. Dovresti anche rimuovere tutti i livelli di pool (adattivi) prima dei livelli di classificazione perché distruggono le informazioni spaziali di cui abbiamo bisogno per regredire le coordinate dei bordi dei riquadri di delimitazione.

Perdita

Quindi come potremmo formare una rete di *rilevamento di oggetti così singoli* che produca valori $4 + n$? Prevedere la classe dell'oggetto (probabilità della classe n) è un problema di *classificazione* . Prevedere le quattro coordinate per il rettangolo di selezione è un problema di *regressione* . Abbiamo bisogno di una funzione di perdita che combini questi due problemi.

Cominciamo con i riquadri di delimitazione: la *perdita di localizzazione* potrebbe essere la perdita di *L1* delle coordinate previste del riquadro di delimitazione *x_pred* e le coordinate del riquadro di delimitazione della verità di terra *x_label* :

$$L_{\text{loc}} = \sum_{i=1}^{4} \left| x_{\text{pred},i} - x_{\text{label},i} \right|$$

Questa è la somma di tutte le *differenze assolute* tra le quattro coordinate del riquadro di delimitazione della verità prevista e quattro. Più la scatola prevista differisce dalla scatola di delimitazione della verità di terra, maggiore sarà questa perdita. È possibile utilizzare anche la perdita di *L2* , tuttavia, il modello sarà più sensibile agli outlier e si adatterà per ridurre al minimo i singoli casi anomali a scapito degli esempi normali che presentano un errore molto più piccolo.

Cosa facciamo con i valori n che la rete emette per le probabilità di classe? Innanzitutto, per convertire questi n valori in probabilità, applichiamo loro la funzione di *attivazione* del *softmax* . Quindi usiamo la *funzione di perdita di entropia incrociata* per confrontarli con l'etichetta . Chiamiamo questa *perdita di fiducia.*

La funzione di perdita combinata è semplicemente la somma ponderata della *perdita* di *localizzazione* (caselle di delimitazione) e *perdita di confidenza* (classi):

$$L = L_{\text{loc}} + a \cdot L_{\text{conf}}$$

Esempi dal set di dati Pascal VOC 2007 con un singolo oggetto rilevato

Che cos'è *un* ? In linea di principio, la perdita di localizzazione potrebbe essere molto maggiore della perdita di fiducia (o viceversa). In tal caso, la rete si concentrerebbe esclusivamente sull'imparare a prevedere i limiti, ignorando completamente l'attività di classificazione. Pertanto, è necessario esaminare i valori di entrambe le perdite e moltiplicarne una per un fattore *a* che le rende approssimativamente dello stesso ordine di grandezza.

Ora sappiamo come potrebbe apparire un set di dati per il rilevamento di oggetti, come potremmo impostare una *singola* architettura di *rilevamento di oggetti* e come formulare la funzione di perdita. Abbiamo tutto il necessario per addestrare la rete. Se desideri addestrare un modello del genere, ti suggerisco di iniziare con la codifica alla lezione 8 di fastai :)

. . .

Ok, abbiamo discusso i passaggi minimi necessari per rilevare un singolo oggetto in un'immagine. Ma questo non è abbastanza, giusto? In uno scenario del mondo reale probabilmente dovremo rilevare molti oggetti (di un numero sconosciuto) in una singola immagine. Come possiamo farlo?

La visione computerizzata classica utilizzava spesso un approccio chiamato *finestra scorrevole* . Un rilevatore scorre sopra un'immagine e una volta che rileva un oggetto, viene disegnato un riquadro di delimitazione nell'area che il rilevatore sta attualmente

guardando.

Con l'avvento del Deep Learning i cosiddetti metodi a *due fasi* o *proposti* per *regione hanno* iniziato a dominare: il primo stadio prevede un insieme di *posizioni di oggetti candidati* che dovrebbero contenere tutti gli oggetti ma filtrare la maggior parte dello sfondo. La seconda fase, una *convnet* , quindi classifica gli oggetti in quelle posizioni candidate come una delle categorie ricercate o come sfondo. Google per R-CNN se sei interessato a questa tecnica.

I metodi di proposta per regione producono risultati all'avanguardia ma sono spesso troppo computazionalmente costosi per il rilevamento di oggetti in tempo reale, specialmente su sistemi embedded.

YOLO-Sembri solo una volta (Redmon et al. 2015) e il *rivelatore MultiBox SSD-Single Shot* (Liu et al. 2015) sono architetture che mirano a risolvere questo problema predicendo le coordinate del riquadro di limitazione e le probabilità per le diverse categorie in un *unico passaggio in avanti* attraverso la rete. Sono ottimizzati per la velocità a scapito della precisione.

Lin et al. (2017) ha recentemente pubblicato un bellissimo documento: hanno spiegato perché metodi come SSD sono meno precisi dei metodi a due fasi e hanno proposto di affrontare il problema riscalando la funzione di perdita. Con questo miglioramento, i metodi a colpo singolo non sono solo più veloci dei metodi a due stadi, ma ora sono altrettanto accurati consentendo nuove fantastiche applicazioni del mondo reale di rilevamento di oggetti in tempo reale.

Ora ti spiegherò in dettaglio come funziona SSD, perché i metodi single-shot sono meno precisi dei metodi a due stadi e come RetinaNet e la perdita focale risolvono questo problema.

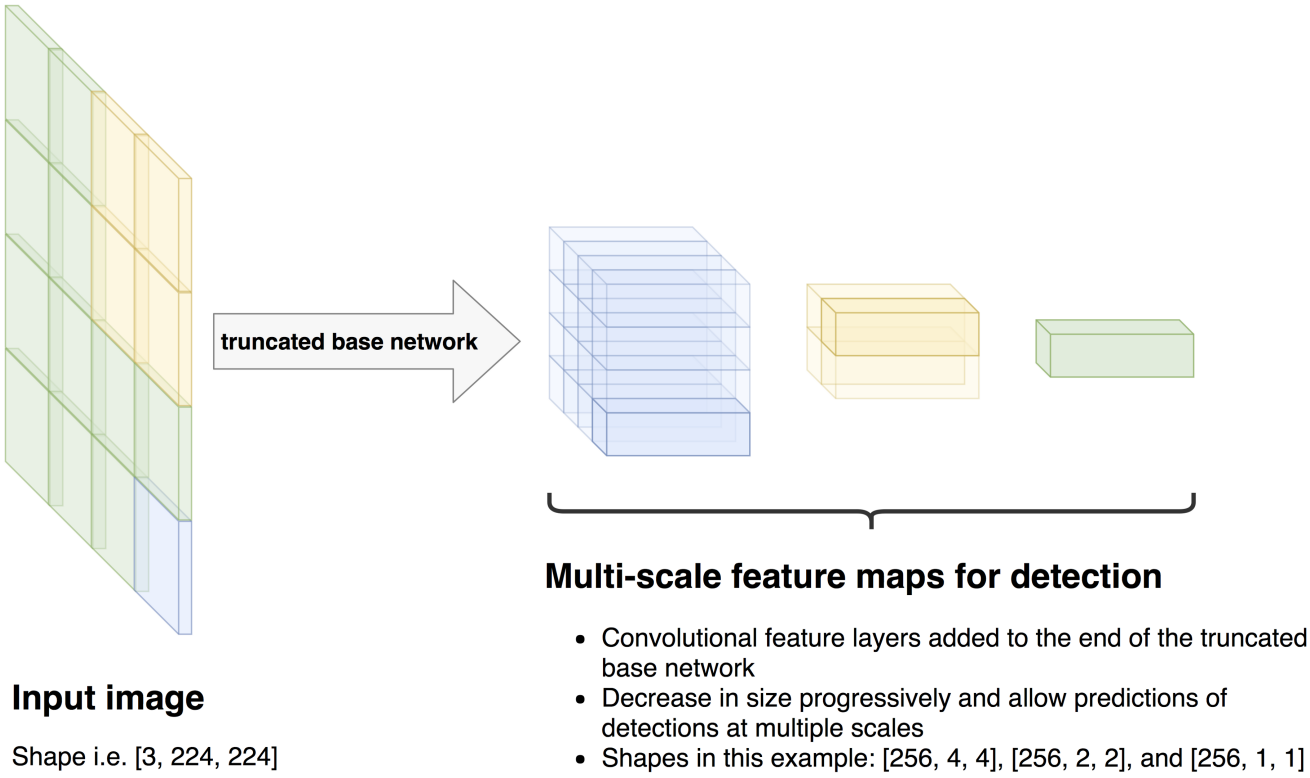
. . .

SSD: rivelatore MultiBox a colpo singolo

Diamo un'occhiata all'architettura di SSD (simile all'implementazione in fastai) e discutiamo il concetto del *campo ricettivo di un'attivazione* lungo la strada. YOLO funziona in modo simile a SSD con la differenza che utilizza layer completamente connessi anziché layer convoluzionali nella parte superiore della rete. Ci concentreremo sull'SSD superiore.

L'immagine viene alimentata con un'architettura standard per una classificazione delle immagini di alta qualità. Eventuali livelli di classificazione alla fine della rete vengono nuovamente *troncati* . Il documento SSD utilizza una rete *VGG-16* , ma funzionano anche altre reti come R *esNets* .

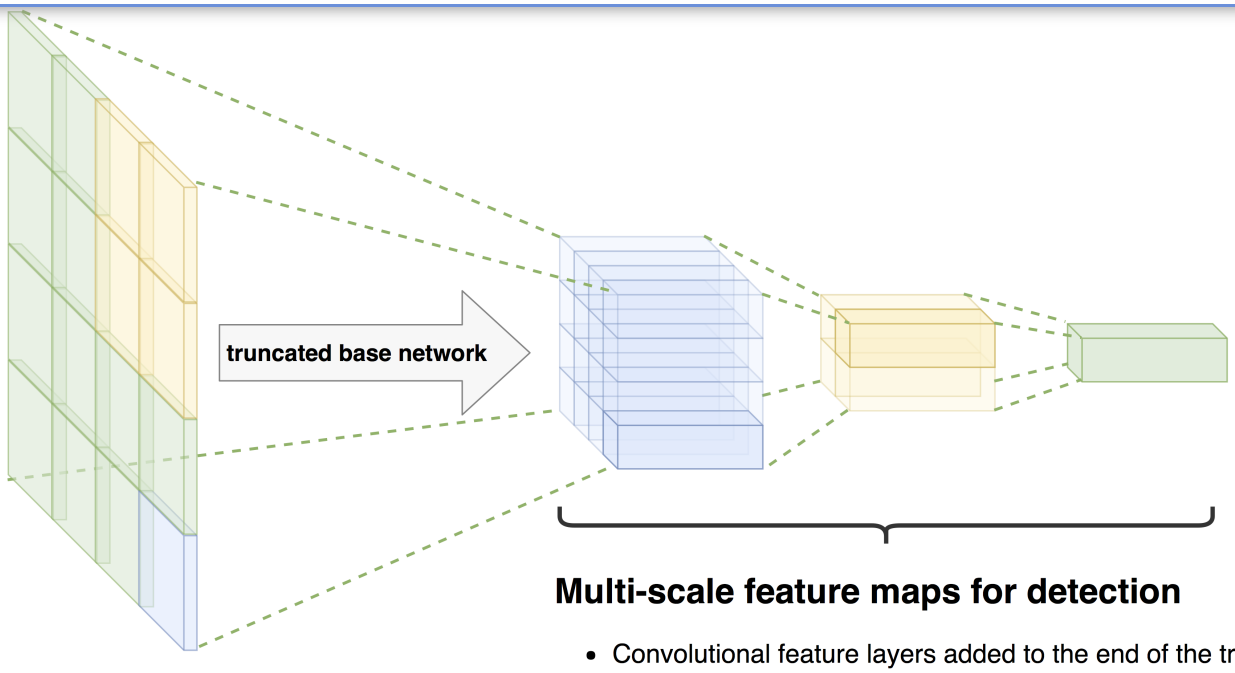
Diciamo ad esempio che l'immagine di input ha la forma [3, 224, 224] e che l'output della rete troncata ha la forma [256, 7, 7]. Attraverso altre tre convoluzioni creiamo tre *mappe caratteristiche* nella parte superiore della rete con le forme [256, 4, 4] (blu), [256, 2, 2] (giallo) e infine [256, 1, 1] (verde):



L'architettura nel documento SSD ha più mappe di funzionalità nella parte superiore della rete ma a scopo dimostrativo, ci atteniamo a tre livelli. Il principio è esattamente lo stesso.

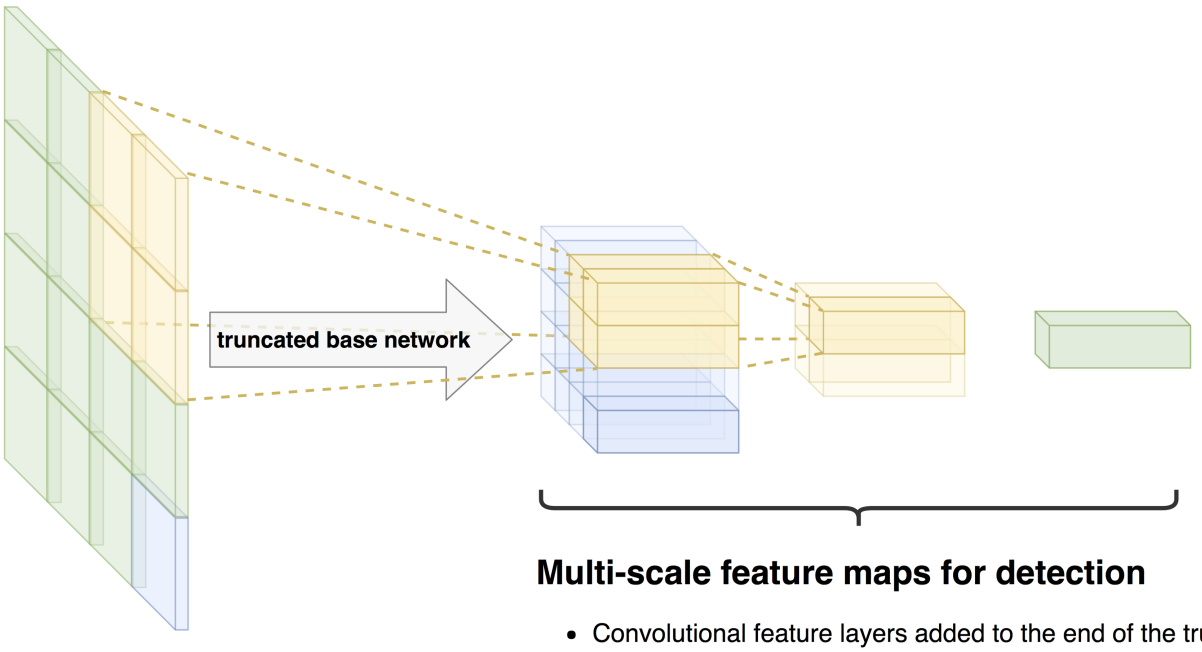
Diamo un'occhiata al *campo ricettivo* delle attivazioni in quei diversi strati:

Iniziamo con il livello finale (verde). Le attivazioni nel livello verde finale hanno dipendenze da tutte le attivazioni nei livelli precedenti e il campo ricettivo è quindi l'intera immagine di input:



- Convolutional feature layers added to the end of the truncated base network
- Decrease in size progressively and allow predictions of detections at multiple scales
- Shapes in this example: [256, 4, 4], [256, 2, 2], and [256, 1, 1]

Diamo ora un'occhiata al penultimo strato (giallo). Si noti che il campo ricettivo di un'attivazione nel livello giallo è solo un quarto dell'immagine di input:



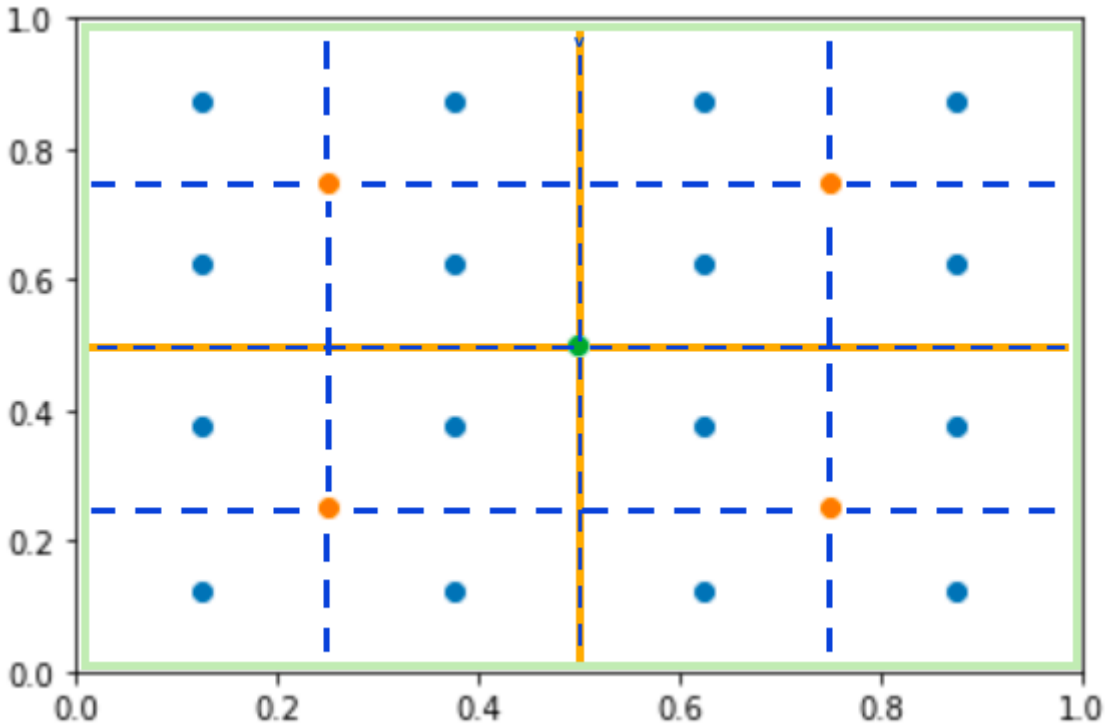
- Convolutional feature layers added to the end of the truncated base network
- Decrease in size progressively and allow predictions of detections at multiple scales
- Shapes in this example: [256, 4, 4], [256, 2, 2], and [256, 1, 1]

Il campo ricettivo di un'attivazione nel livello blu è un 16 ° dell'immagine di input.

Si noti che le illustrazioni che ho realizzato sono semplificate a scopo dimostrativo e, ad esempio, non tengono conto del fatto che un'attivazione ha più dipendenze provenienti dal centro del suo campo ricettivo rispetto alle regioni circostanti. Guarda qui per un'ottima spiegazione.

L'idea che voglio far capire, tuttavia, è che ha senso che un oggetto che riempie quasi l'intera immagine dovrebbe essere meglio rilevato dalle attivazioni nell'ultimo (livello verde) e un oggetto che riempie approssimativamente il quarto inferiore sinistro dell'input l'immagine dovrebbe essere rilevata dalle attivazioni nel livello giallo.

Pertanto, definiamo un numero di cosiddette *caselle predefinite* o *caselle di ancoraggio* :



I punti sono i cosiddetti *ancoraggi* e segnano i centri delle rispettive *caselle predefinite* o di *ancoraggio* . L'ancora verde al centro è il centro della casella verde attorno all'intera immagine che è la casella predefinita corrispondente all'ultimo livello convoluzionale (verde).

Le ancore giallo / arancione definiscono i centri delle quattro caselle giallo / arancione che appartengono al penultimo strato convoluzionale (giallo) e le 16 caselle predefinite blu appartengono al livello blu nell'architettura SSD illustrata.

Potresti chiederti ora come sarebbe simile a questo nel codice. Ad esempio, è possibile creare un array in cui ogni riga definisce una casella predefinita e che ha quattro colonne, ad esempio le coordinate xey degli angoli in basso a sinistra e in alto a destra o le coordinate di un angolo più altezza e larghezza.

Cosa facciamo dopo? Quando abbiamo esaminato il modo in cui possiamo rilevare un singolo oggetto in un'immagine, abbiamo previsto 4 valori per le coordinate del riquadro di delimitazione e una probabilità per ciascuna delle classi *n* dandoci un totale di 4 + *n* numeri restituiti dalla nostra rete neurale. Tieni presente che ora vogliamo anche classificare lo *sfondo,* quindi *n* dovrebbe essere il numero di categorie nel tuo set di dati + 1.

Questo è ciò di cui abbiamo bisogno ora *per ogni riquadro predefinito* : ogni riquadro predefinito richiede *n* valori che rappresentano le probabilità che una determinata classe sia stata rilevata in quel riquadro e 4 valori che ora non sono coordinate assolute del riquadro di delimitazione previsto ma piuttosto l'offset rispetto al rispettivo scatola predefinita.

Per le caselle predefinite blu 4x4, abbiamo bisogno di un totale di 16 volte 4 valori per calcolare le coordinate delle caselle di delimitazione previste e 16 volte *n* probabilità per *n* categorie che proviamo a classificare.

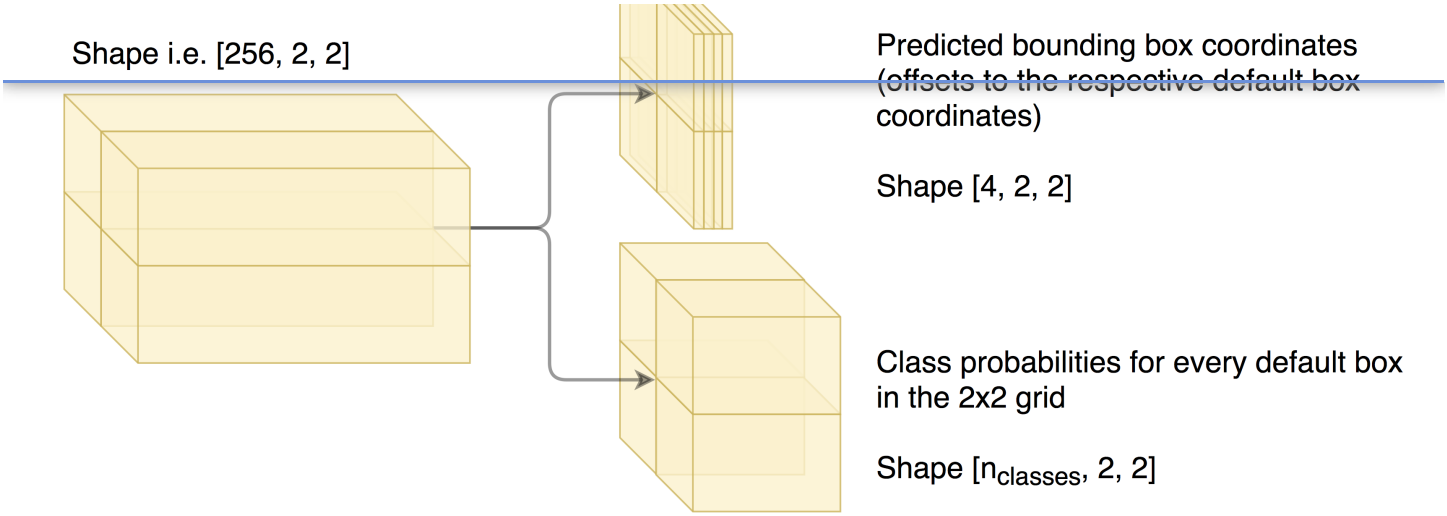
Per le caselle di default gialle 2x2, abbiamo bisogno di un totale di 4 volte 4 valori per le coordinate e 4 volte *n* probabilità per le *n* classi.

Come facciamo questo? Altri strati convoluzionali!

Le ultime tre mappe caratteristiche nell'architettura SSD tracciata - i livelli blu, giallo e verde - hanno le forme [256, 4, 4], [256, 2, 2] e [256, 1, 1]. Corrispondono alla griglia 4x4 di caselle predefinite blu, alla griglia 2x2 di caselle predefinite gialle e all'unica casella predefinita verde che copre l'intera immagine nella griglia mostrata sopra.

Ognuna delle tre mappe caratteristiche viene immessa in altri due livelli convoluzionali. So che diventa confuso ma abbi pazienza, ci siamo quasi :)

Feature map for the 2x2 grid of default boxes **Convolutional predictors for detection**



Diamo un'occhiata alla mappa caratteristica gialla (griglia 2x2). Abbiamo [256, 2, 2] attivazioni e abbiamo bisogno di un output di forma [4, 2, 2] e un secondo output di forma [*n* , 2, 2] (dove *n* è il numero di classi). Pertanto, la larghezza e l'altezza della mappa delle caratteristiche non devono essere modificate, ma i due livelli di output convoluzionali devono ridurre la mappa delle caratteristiche da 256 a 4 e *n* filtri, rispettivamente.

Per la mappa caratteristica blu (che corrisponde alla griglia 4x4), abbiamo bisogno di due livelli di output che riducono la mappa caratteristica [256, 4, 4] a un tensore [4, 4, 4] e [*n* , 4, 4] .

La casella predefinita verde che guarda l'intera immagine corrisponde alla mappa delle caratteristiche verde che ha la forma [256, 1, 1]. Abbiamo finalmente bisogno di altri due livelli di output convoluzionali che prendono come input la mappa verde delle caratteristiche e producono output a forma di [4, 1, 1] e [*n* , 1, 1].

So che è stato molto. L'idea importante è che facciamo per ogni casella predefinita nelle griglie di dimensioni diverse ciò che abbiamo fatto quando abbiamo previsto un singolo oggetto in un'immagine.

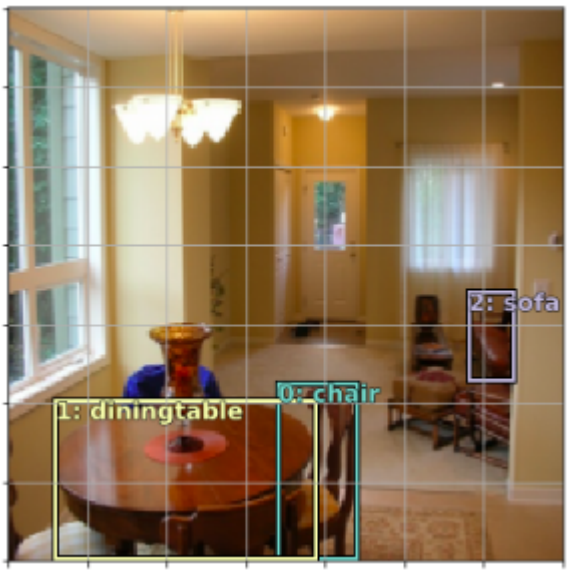
. . .

Ok, riassumiamo:

1. Abbiamo definito diverse griglie di caselle predefinite di dimensioni diverse che ci consentiranno di rilevare oggetti a diverse scale in *un unico passaggio in avanti* . Molte architetture precedenti hanno rilevato oggetti su scale diverse, ad esempio passando al rivelatore versioni di dimensioni diverse della stessa immagine, che

sono computazionalmente più costose.

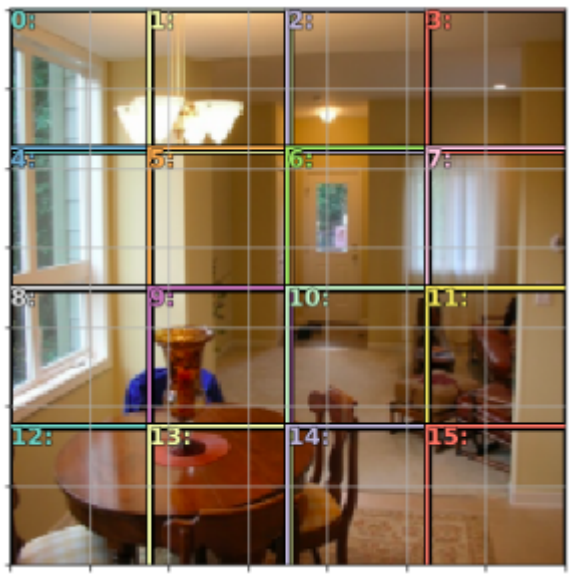
2. Per ogni riquadro predefinito in ogni griglia, la rete emette probabilità di classe *n* e 4 offset alle rispettive coordinate del riquadro predefinito che forniscono le coordinate previste del riquadro di delimitazione.



Scatole di delimitazione della verità di terra

Quindi, come possiamo addestrare la rete?

A sinistra, viene visualizzato un esempio del set di dati Pascal VOC 2007 per il rilevamento degli oggetti. Ci sono tre scatole di delimitazione della verità per un tavolo da pranzo e due sedie.



Griglia 4x4 di caselle predefinite

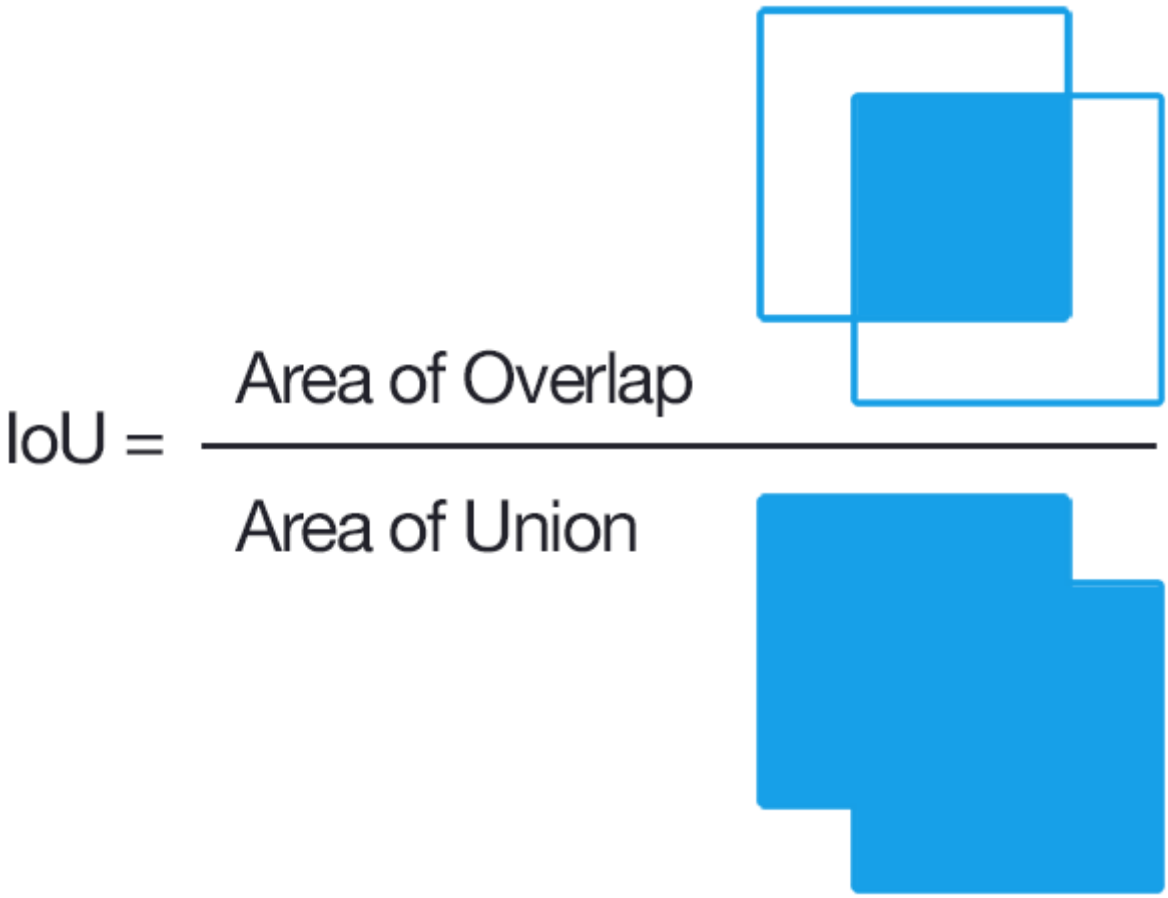
Nell'immagine sotto vedi la griglia 4x4 delle caselle predefinite (le griglie 2x2 e 1x1 non sono mostrate).

La rete prevede 4 coordinate (offset) e probabilità della classe *n* per ciascuna delle 16 caselle predefinite. Ma quale di queste previsioni dovremmo confrontare con le caselle predefinite di verità di base nella nostra funzione di perdita durante l'allenamento?

Dobbiamo abbinare ciascuna delle caselle di delimitazione della verità di base nel nostro esempio di addestramento a (almeno) una casella predefinita.

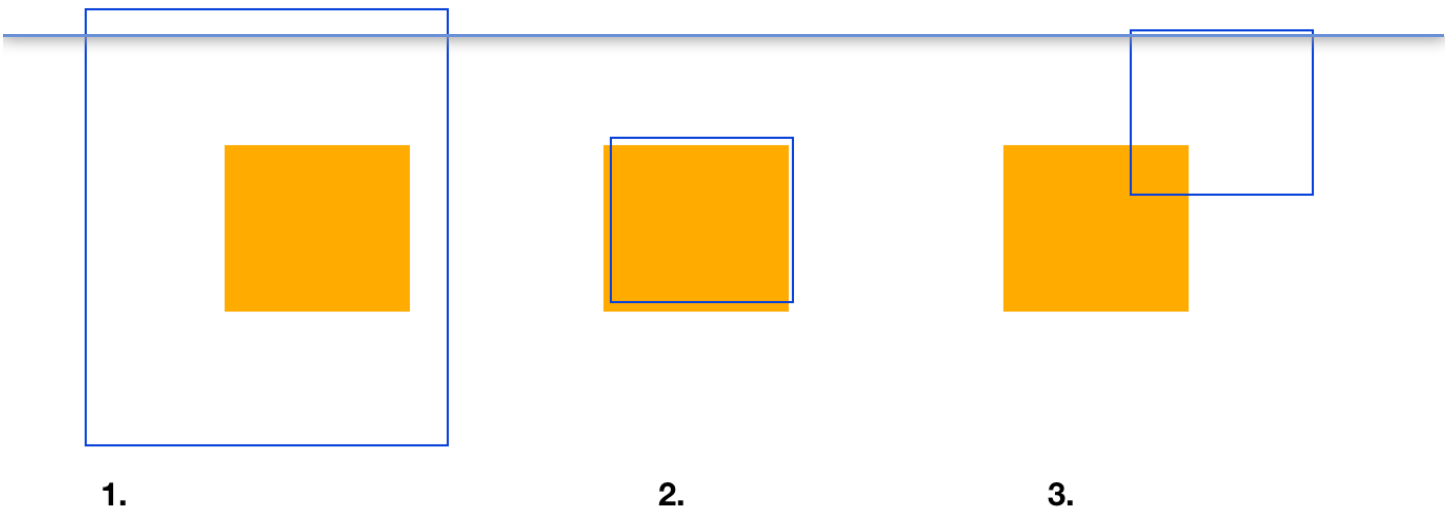
Problema di corrispondenza

Vogliamo abbinare il riquadro di delimitazione della verità di base a un riquadro predefinito che è "il più simile" possibile. Cosa intendo con simile? Due caselle sono simili quando si sovrappongono il più possibile pur avendo il minor spazio possibile che non si sovrappone. Questo è definito dall'indice *Jaccard* o dall'indice *IoU* :



fonte

Diamo un'occhiata ad alcuni esempi:



1. L'area di sovrapposizione (intersezione) è l'intera area arancione, quindi la sovrapposizione ha raggiunto il massimo. Ma l'unione (area blu) è molto più grande. Questa non è una buona corrispondenza che si traduce in un indice Jaccard basso.

2. Questa è la migliore corrispondenza: l'area di sovrapposizione e l'area dell'unione hanno quasi le stesse dimensioni. L'indice Jaccard dovrebbe essere vicino a 1.

3. Anche in questo caso, la sovrapposizione è molto più piccola dell'intera area delle due caselle (unione) che fornisce un indice Jaccard basso.

Calcoliamo l'indice Jaccard per *ogni* riquadro di delimitazione della verità di base nel nostro esempio di addestramento con *ogni* riquadro predefinito nelle nostre griglie 4x4, 2x2 e 1x1.

Un riquadro di delimitazione della verità di terra è abbinato al riquadro predefinito con cui condivide l'indice Jaccard più alto e a ogni altro riquadro predefinito con cui ha un indice Jaccard che è maggiore di una determinata soglia.

So che questi dettagli potrebbero sembrare confusi. L'idea importante è che vogliamo confrontare le scatole di delimitazione della verità di base nell'esempio di addestramento con le previsioni fatte dalle caselle di default che sono già molto simili alle scatole di delimitazione della verità di base.

Ora sappiamo ...

1. ... 4 coordinate previste (offset alle caselle predefinite) e *n* probabilità previste della classe per *ogni* casella predefinita e ...

2. ... quale riquadro di delimitazione della verità di base corrisponde a quale riquadro predefinito.

Ora facciamo la stessa cosa che abbiamo fatto quando volevamo rilevare un solo oggetto.

La perdita è di nuovo una somma ponderata della *perdita* di *localizzazione* (caselle di delimitazione) e *perdita di confidenza* (classi): calcoliamo quanto la casella di delimitazione prevista (coordinate della casella predefinita + offset previsti) differisce dalla casella di delimitazione della verità del terreno corrispondente (perdita L1) e quanto correttamente la casella predefinita ha previsto la classe (entropia incrociata binaria). Aggiungiamo questi due valori per ogni coppia di box di default e coppia di limiti di verità di base abbinati insieme e abbiamo una funzione di perdita che restituirà valori più bassi quando le scatole previste sono più vicine alle scatole di delimitazione di verità di terra e quando la rete ha fatto un lavoro migliore nel classificare oggetti.

Quando nessuna probabilità di una singola classe supera una determinata soglia, la rete considererà l'oggetto in quella casella come sfondo. In realtà, scoprirai che per ogni oggetto reale nell'immagine la rete produrrà diversi rettangoli che si trovano quasi uno sopra l'altro. Una tecnica chiamata *soppressione non massima* viene utilizzata per ridurre tutte le caselle che prevedono la stessa classe e hanno un indice Jaccard che è maggiore di una determinata soglia a una singola casella prevista per oggetto rilevato.

. . .

C'è un piccolo dettaglio che ho omesso per motivi di comprensibilità. Per ogni ancoraggio, SSD definisce k caselle predefinite di diverse proporzioni e dimensioni invece di una sola come discusso in questo articolo. Ciò significa che abbiamo anche bisogno di 4 volte k offset previsti per le rispettive coordinate box predefinite anziché 4 e n volte k probabilità della classe anziché n . Più scatole di diverse dimensioni e proporzioni = migliore rilevamento degli oggetti. Questo dettaglio non è troppo importante per la comprensione generale, ma puoi vedere la Figura 1 in Liu et al. 2015 per l'illustrazione se sei interessato a questo dettaglio.

. . .

SSD e YOLO necessitano solo di un passaggio in avanti attraverso la rete per prevedere le scatole di delimitazione degli oggetti e le probabilità di classe. Rispetto alle *finestre scorrevoli* e ai metodi di *proposta delle regioni* , sono molto più veloci e quindi adatti al rilevamento di oggetti in *tempo reale* . SSD (che utilizza mappe di funzionalità convoluzionali su più scale nella parte superiore della rete anziché livelli completamente connessi come YOLO) è più veloce e più preciso di YOLO.

L'unico problema rimanente: i metodi di proposta regionali come R-CNN sono più *precisi* .

Perdita focale

Il vantaggio dei metodi a due stadi è che prima prevedono *alcune* posizioni degli oggetti candidati e quindi usano una rete neurale convoluzionale per classificare ciascuna di queste posizioni degli oggetti candidati come una delle classi o come sfondo. L'enfasi qui è su *alcune località candidate* .

Metodi come SSD o YOLO soffrono di uno *squilibrio di classe* estremo : i rilevatori valutano all'incirca tra le 10.000 e le 10000 posizioni candidate (molto più delle scatole predefinite 4x4, 2x2 + 1 nel nostro esempio qui) e, naturalmente, la maggior parte di queste scatole non contiene un oggetto. Anche se il rilevatore classifica facilmente questo grande numero di scatole come negativi / di fondo, c'è ancora un problema.

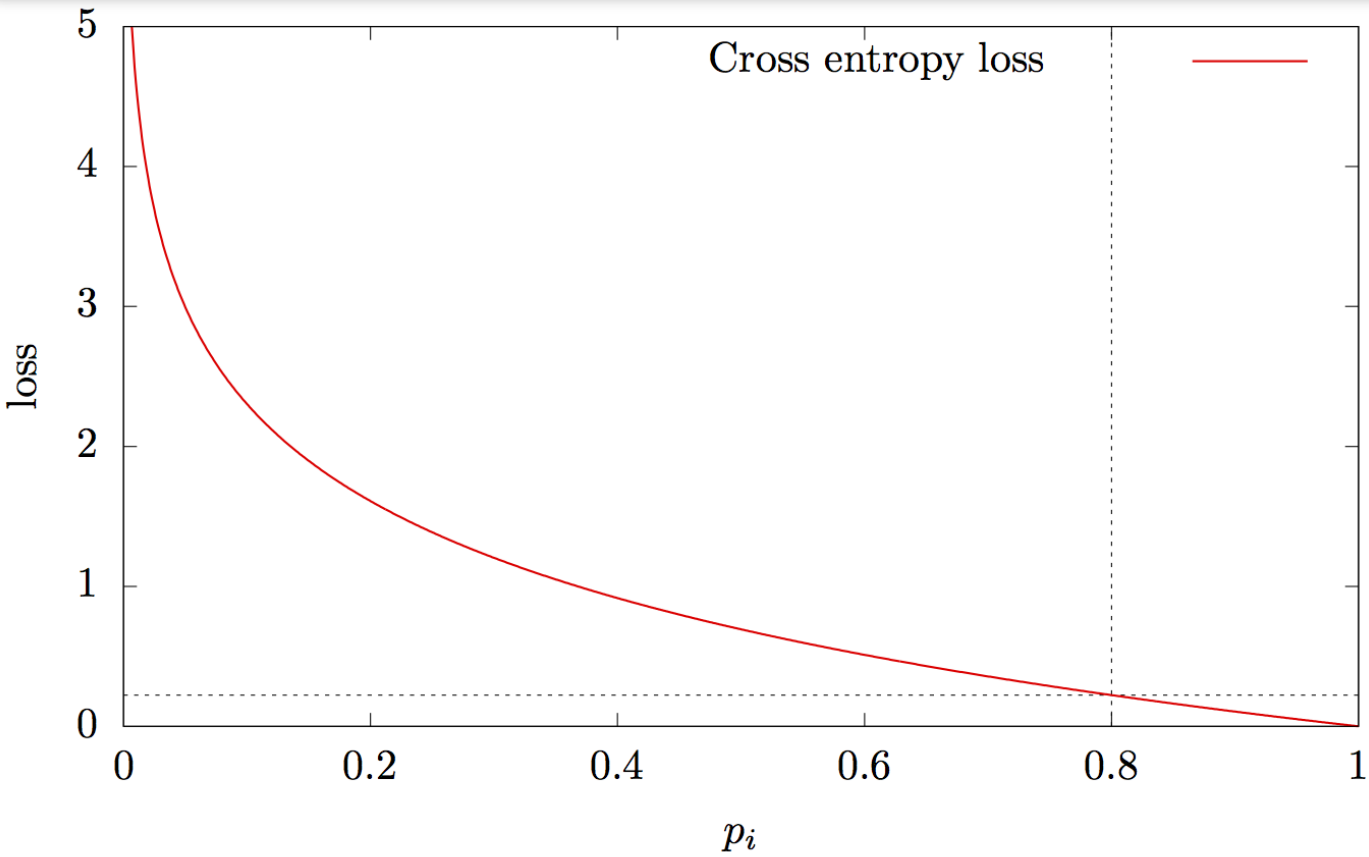
Ti spiego perché:

Questa è la funzione di perdita di entropia incrociata in cui *i* è l'indice della classe, *y_i* l'etichetta (1 se l'oggetto appartiene alla classe *i* , 0 altrimenti) e *p_i* è la probabilità prevista che l'oggetto appartenga alla classe *i* .

$$C(p, y) = -\sum_i y_i \ln p_i$$

Vedremo la trama, non preoccuparti troppo dell'equazione :)

Supponiamo che una casella contenga uno sfondo e che la rete sia sicura all'80% che in realtà è solo uno sfondo. In questo caso *y (sfondo) = 1* , tutti gli altri *y_i* sono 0 e *p (sfondo) = 0,8* .



Si può vedere che all'80% di certezza che la scatola contiene solo sfondo, la perdita è ancora ~ 0,22.

Diciamo ad esempio che nell'immagine sono presenti dieci oggetti effettivi e la rete non è davvero sicura a quale classe appartengano, quindi la loro perdita è pari a ~ 3. Questo ci darebbe circa 30 *(tutti i numeri sono assolutamente immaginari e scelti a scopo dimostrativo)* .

Tutte le altre ~ 10.000 caselle predefinite sono in background e la rete è sicura all'80% che sono solo in background. Questo ci dà una perdita di circa 10.000 * 0,22 = 2200.

Allora dimmi, cosa domina? I pochi oggetti veri che la rete ha effettivamente difficoltà a classificare? O tutte le caselle che la rete classifica facilmente come sfondo?

Bene, il gran numero di esempi facilmente classificati domina assolutamente la perdita e quindi i gradienti e quindi travolge i pochi casi interessanti con cui la rete ha ancora difficoltà e da cui dovrebbe imparare.

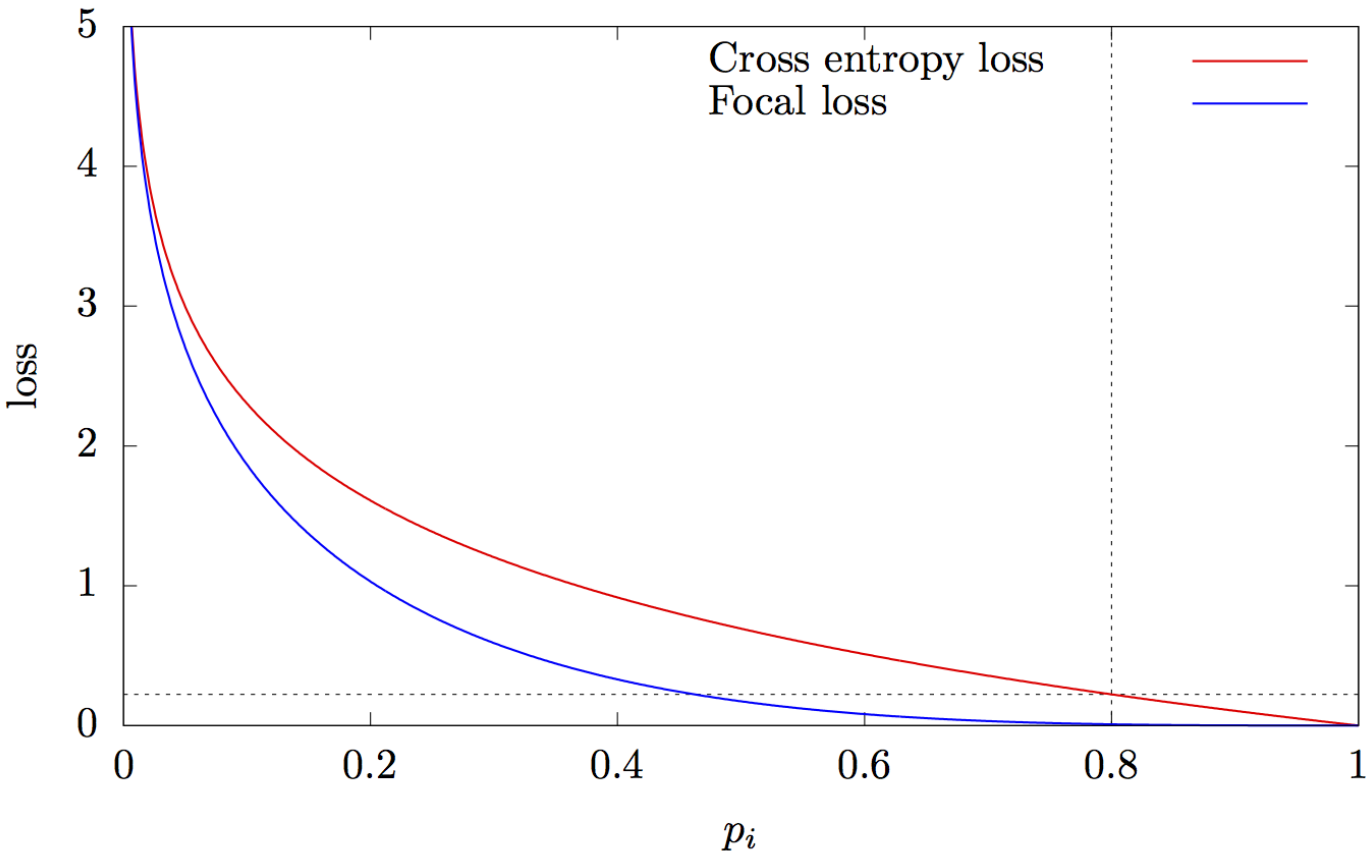
Quindi cosa facciamo? Lin et al. (2017) ha avuto la bella idea di ridimensionare la perdita di entropia incrociata in modo che tutti i semplici esempi su cui la rete sia già

perdita di entropia incrociata in modo che tutti i semplici esempi su cui la rete sia già

molto sicura contribuiscano meno alla perdita in modo che l'apprendimento possa *concentrarsi* su pochi casi interessanti. Gli autori hanno chiamato la loro funzione di perdita *Focal loss* e la loro architettura *RetinaNet* (si noti che RetinaNet include anche Feature Pyramid Networks (FPN) che è fondamentalmente un nuovo nome per *U-Net*).

$$C(p, y) = -\sum_i y_i (1 - p_i)^\gamma \ln p_i$$

Secondo la carta gamma = 2 funziona meglio. Ancora una volta, se non vuoi, non preoccuparti troppo dell'equazione, capirai la differenza che fa una volta che vedi la trama:



Si noti che quando la rete è abbastanza sicura di una previsione, la perdita è ora significativamente inferiore. Nel nostro precedente esempio dell'80% di certezza, la perdita di entropia crociata aveva un valore di ~ 0,22 e ora la perdita focale un valore di solo 0,009. Per le previsioni della rete non è così sicuro, la perdita è stata ridotta di un fattore molto più piccolo!

un fattore molto più piccolo!

Con questo riscalaggio, il gran numero di esempi facilmente classificabili (principalmente background) non domina più la perdita e l'apprendimento può concentrarsi sui pochi casi interessanti.

. . .

Con questo potente miglioramento, i rilevatori di oggetti che richiedono solo un singolo passaggio in avanti attraverso una rete sono improvvisamente in grado di competere con i metodi a due stadi per quanto riguarda la precisione, battendoli facilmente rispetto alla velocità. Ciò apre molte nuove possibilità per il rilevamento accurato di oggetti in tempo reale anche su sistemi embedded. Eccezionale!

. . .

Se sei affascinato da questa roba come me, ti consiglio vivamente di implementare tu stesso una rete di rilevamento di oggetti a colpo singolo. Ti suggerisco di programmare insieme alle lezioni 8 e 9 del grande corso *fastai - Cutting Edge Deep Learning for Coders* . Potete trovare il notebook corrispondente qui. In questo notebook, tutte le funzioni relative al problema di corrispondenza sono appena scritte in una cella e il loro output viene mostrato solo in seguito. Ciò riduce molto la comprensibilità del codice. Ho reimplementato il notebook, aggiunto altri commenti e portato tutto (specialmente il problema di abbinamento) in un ordine che si spera ti permetta di leggere il notebook dall'alto verso il basso mentre gradualmente costruisco una comprensione di come le scatole di delimitazione della verità di terra sono abbinate alle caselle predefinite prima procedere all'implementazione di funzioni a tale scopo. Sentiti libero di usare il mio notebook anche come riferimento aggiuntivo.

. . .

Spero che tu abbia imparato qualcosa di interessante da questo articolo :) Se qualche parte non è chiara o hai bisogno di ulteriori spiegazioni, per favore lascia un commento, mi piacerebbe aiutarti a capire.

Apprendimento automatico Apprendimento profondo Rilevamento oggetti Computer Vision

Verso la scienza dei dati

A proposito di AiutoLegale