Intro alle CNN

domenica 5 maggio 2024 11:28

Sono state inventate inspirandosi dall'organizzazione della corteccia visiva animale ed anche umana. Molte idee di invenzioni nelle reti neurali prendono spunto dalla natura.

La loro principale peculiarità è il **Pattern Recognition.**

Le CNN sono brave a riconoscere pattern ricorrenti all'interno dei dati.

Ad esempio se la CNN si addestra su immagini di volti umani che sorridono, tutte queste immagini avranno dei tratti in comune (ad es. inclinazione delle labbra). **Questo è un esempio di pattern ricorrente che le CNN sono brave ad estrarre**.

Dunque le CNN sono molto brave nel **Riconoscimento di Pattern ricorrenti nei segnali che gli si danno input**, siano questi segnali audio, serie temporali, segnali video, segnali statici come le immagini, segnali testuali ecc.

Pattern = schema, configurazione.

Convoluzione

domenica 5 maggio 2024 12:25

La Convoluzione ci permette di fondere 2 funzioni dando vita ad una terza funzione che racchiude un po' l'interazione tra le due funzioni di input.

Dal punto di vista matematico una immagine RGB può essere vista come una segnale discreto, cioè un segnale rappresentato da un valore finito di valori (numeri).

Dunque, nel caso delle immagini RGB ogni pixel è rappresentato da valori compresi tra 0 e 255 per ciascuno dei 3 canali RGB. Ciò significa che una immagine RGB può essere rappresentata come un segnale discreto tridimensionale (una dimensione per ogni canale RGB).

Nei task di image processing uno dei due input della convoluzione sarà appunto l'immagine e l'output della convoluzione sarà un'altra immagine.

Un Kernel (es. 3x3, 5x5) viene sovrapposto ad ogni gruppo di pixel di pari dimensione dell'immagine originale, facendolo scorrere lungo tutta l'immagine di input.

Durante queste sovrapposizioni si fanno delle operazioni di prodotto e somme algebriche i cui risultati costituiscono i pixel di una nuova immagine.

A seconda del Kernel che viene utilizzato è possibile estrarre features (caratteristiche) diverse da un'immagine.

Ad esempio l'effetto Blur (sfocatura) di una immagine è il risultato di operazione di convoluzione su un'immagine.

Un altro esempio è l'estrazione dei bordi di una immagine è il risultato di un'operazione di convoluzione su una immagine.

Ciò che cambia da un esempio all'altro (sfocatura e bordi) è l'utilizzo di kernel (filtro) diversi che scorrono sull'immagine di input.

Diversi Kernel possono estrarre diverse feautures (caratteristiche) dall'immagine di input.

Esempio di convoluzione

domenica 5 maggio 2024 15:57

vedi sito: [Image Kernels explained visually (setosa.io)]

Nell'immagine data in input è una immagine a scala di grigi quindi i pixel sono rappresentati da valori tra 0 e 255 (monodimensione) dove 0 è nero e 255 è bianco ed i valori intermedi rappresentano una toanlità di grigio più o meno intensa.

Ci sono casi in cui il risultato della convoluzione tra kernel e porzione di pixel dell'immagine diano in output valori che non sono compresi tra 0 e 255, alcuni sono negativi (ad es. -45) altri troppo grandi (460), nelle prossime sezioni vedremo come comportarci in tali casi.

Padding

domenica 5 maggio 2024 16:26

Ipotesi: Ogni pixel di un'immagine di dimensioni WxH viene convoluta con un kernel di dimensioni Wk x Hk.

Risultato: L'immagine finale ha dimensioni (W - Wk + 1) x (H - Hk + 1)

Esempio: immagine da 6x6, kernel (filtro) da 3x3, immagine di output: (6-3+1) x (6-3+1) = 4x4

Ciò accade perché i pixel della cornice non danno contributi in una convoluzione.

Se eseguiamo una ulteriore convoluzione tra il risultato della prima convoluzione ed il kernel otteniamo una immagine ancora più piccola. Se continuiamo ad eseguire convoluzioni dopo convoluzioni otterremmo immagini impossibili con dimensione negativa. Quì entra in gioco il **padding**.

Il **padding** è l'allargamento dell'immagine di input per far si che dopo la convoluzione si produca un'immagine di dimensione pari alla dimensione della immagine di input senza il padding.

Ad es. padding = 1 aggiungiamo una cornice di zeri alla immagine di input di dimensione 1.
Ciò serve anche per far si che anche i pixel ai confini diano un contributo alla convoluzione cosa che senza il padding non fanno.

Abbiamo 2 tipi di Padding:

Same: dove il padding ci serve per preservare la dimensione dell'immagine di input dopo la convoluzione (padding= (dimensione del filtro dispari -1) / 2)

Valid: padding =0 la convoluzione fa il downsampling dell'immagine di input.

Stride

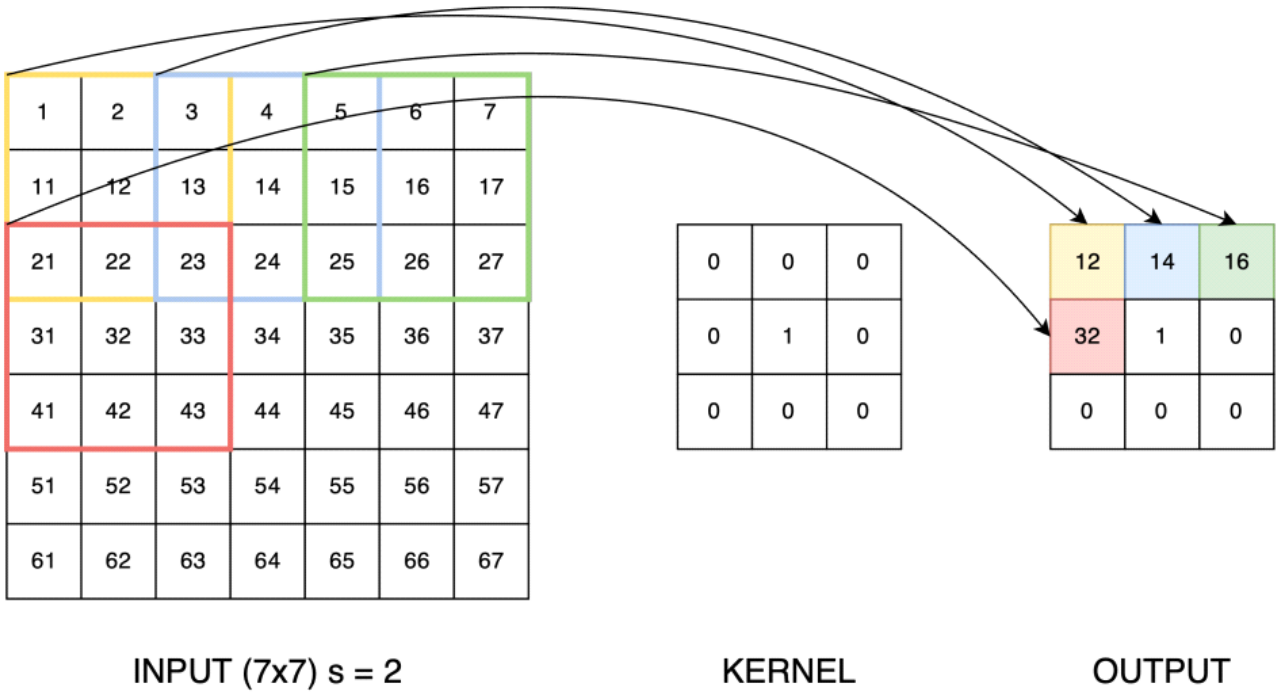
domenica 5 maggio 2024 17:16

È un parametro che ci indica di quanto si sposta il kernel mentre scorre le immagini.

Con s = 1 significa che il kernel viene traslato un pixel per volta.

Con s = 2 significa che il kernel "salta" un pixel (si sposta di 2 pixel per volta)

NB: il kernel si sposta di una quantità pari a S anche in verticale.



Ovviamente anche lo STRIDE come il PADDING influenza la dimensione dell'immagine di output della convoluzione.

Convoluzione volumetrica

domenica 5 maggio 2024 18:23

Nel caso in cui abbiamo più canali (dimensioni) come con le immagini RGB, in tal caso oltre all'altezza ed alla lunghezza abbiamo anche la profondità. Abbiamo il canale Red, il canale Green ed il canale Blue.

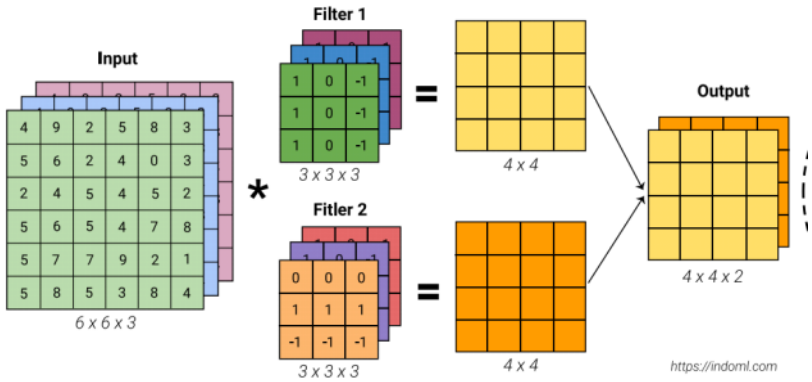
In tal caso associamo lo stesso kernel per ogni canale dell'immagine.

L'immagine RGB è il risultato della sovrapposizione dei canali Rosso, Verde e Blu.
Ad es. nel canale Rosso ciascun pixel ha un valore che va da 0 a 255 che indica l'intensità di rosso per quel pixel. Analogamente per i canali Verde e Blu.

Il punto è abbiamo tanti kernel tanti quanti sono i canali che vengono sovrapposti e fatti scorrere su ciascun rispettivo canale isolato. L'operazione di convoluzione sui singoli canali darà in output i risultati per ogni convoluzione sui rispettivi canali. Ad es. 3 canali -> 3 kernel -> 3 output da combinare con una operazione algebrica di somma -> 1 risultato che va costituire il valore del pixel dell'immagine 2D di output. Infatti il risultato di una convoluzione volumetrica è FLAT immagine 2D con un valore per ciascun pixel.

Filtro = insieme di kernel uguale al numero di canali di input, che lavorano indipendentemente sui rispettivi canali.

Per ottenere un output volumetrico, cioè non soltanto un'immagine 2D ma ad esempio 2 immagini 2d da sovrapporre, utilizziamo più filtri (insieme di kernel), così otteniamo un output multicanale.



Ciascun filtro serve per poter estrarre diverse features dalla immagine di input.

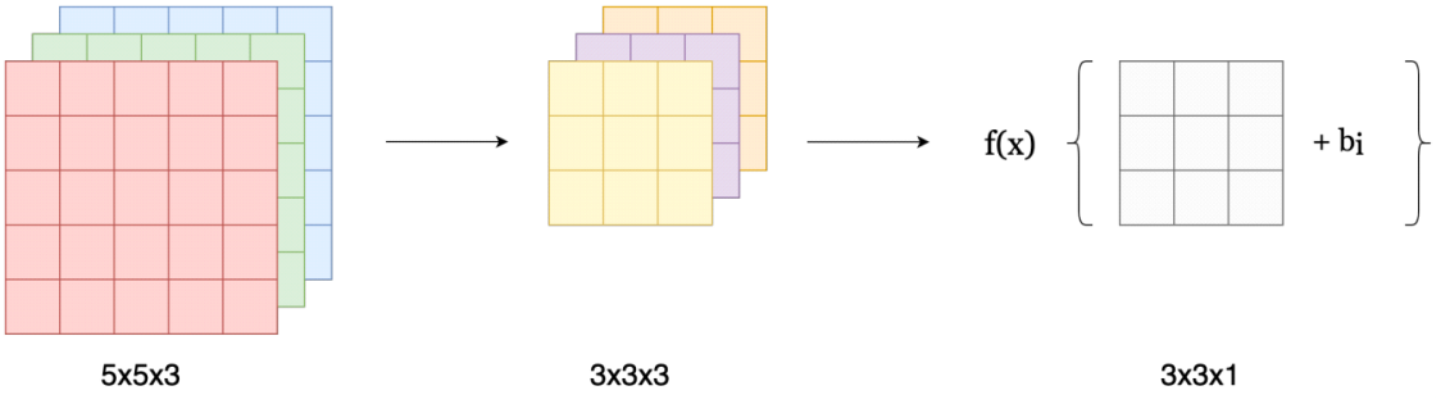
Filtri, kernel e parametri

domenica 5 maggio 2024 19:07

Livelli convolutivi: la convoluzione non è l'unica operazione che viene fatta nelle reti neurali convolutive CNN.

Al risultato di una convoluzione bisogna aggiungere un bias ed una funzione di attivazione. Il risultato di queste tre parti (convoluzione + bias + funzione di attivazione) **è un livello convolutivo**.

Quindi, al risultato FLAT di una convoluzione volumetrica fatta utilizzando un filtro (insieme di kernel), ci sommiamo il BIAS e tutto questo va dato in input ad una funzione di attivazione.



Ciò sarà il risultato di un livello convolutivo!!

I parametri per i kernel (dimensione, volume, padding e stride) sono parametri del livello.

Per ogni filtro (insieme di kernel) viene aggiunto un BIAS ed il risultato della convoluzione + bias attraversano la funzione di attivazione del neurone (ReLU, sigmoide, tanh, ecc.) per aggiungere non linearità al livello, esattamente come nelle ANN.

IMPORTANTE: L'addestramento nelle CNN avviene sui parametri dei singoli kernel. Appunto come sappiamo modificando i valori dei kernel si posson estrarre features diverse dalla immagine, quindi sono questi i valori da addestrare.

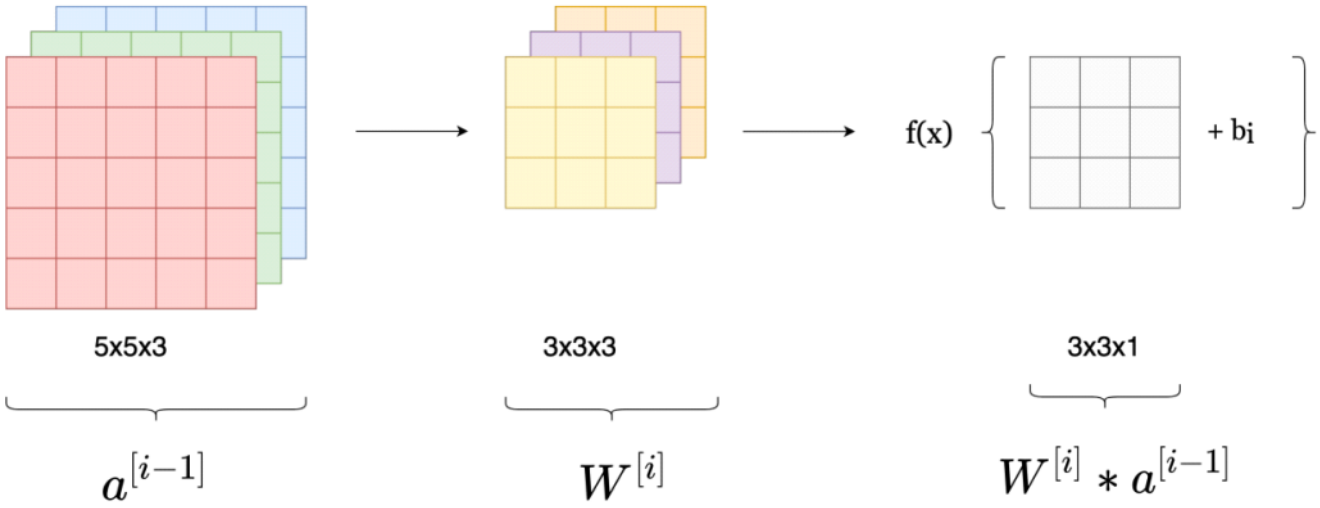
Ricordiamo che in una Rete Neurale:

$$z^{[i]} = W^{[i]} * a^{[i-1]} + b^{[i]}$$

E che l'**output** di un livello in una rete neurale
è dato dall'applicazione della funzione di attivazione

$$a^{[i]} = g(z^{[i]})$$

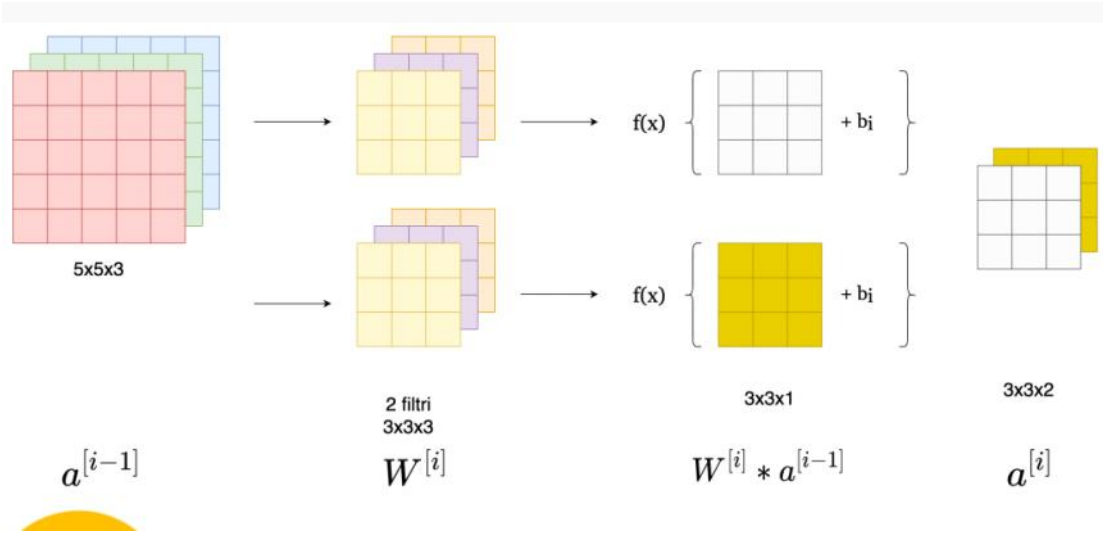
Applicato alle reti neurali convolutive otteniamo:



Parametri: (Dim. Filtro:3x3x3=27 + 1 bias = 28)*1 filtro = 28 parametri addestrabili in questo layer convolutivo

Il numero di filtri usati influenza il volume dell'output di una layer convolutivo.

Se ho usato 2 filtri l'output del livello convolutivo avrà volume pari a 2 e quindi nel successivo livello dovrà essere impiegato almeno un filtro di volume pari 2



Per calcolare il numero di parametri addestrabili in un livello si fa: (dimensione(altezza*larghezza) del filtro corrente * dimensione output livello precedente (profondità da usare per il filtro corrente) + 1 bias = parametri in un filtro) * numero di filtri che uso nel livello corrente.

Local Receptive fields

lunedì 6 maggio 2024 17:25

Poiché la convoluzione non viene applicata solo nel campo di image processing, l'output di una convoluzione non necessariamente è un'immagine(matrice con valori per i pixel).

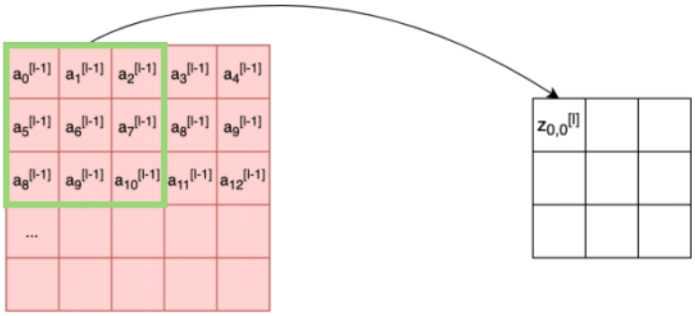
Feature Map: è il risultato di operazione di convoluzione + funzione di attivazione + bias.

Neurone: è un singolo elemento di una feature map.

Receptive Field: è la porzione dell'input di un livello convolutivo "collegato" ad un certo neurone di uscita.

In una CNN un neurone è connesso solo ad una parte dell'input:
- Ogni neurone è "responsabile" solo per una definita porzione dei dati di input.
- Questa porzione prende il nome di **Local Receptive Field.**

Nella seguente immagine vediamo il neurone della feature map (z0, 0) che è collegato all'area ben definita dell'input di dati che entrano nel livello convolutivo, ovvero il suo Local Receptive Field.



Dunque un neurone è collegato solo ed esclusivamente ad un Local Receptive Field, ciò significa che tale neurone è influenzato solo dai cambiamenti in tale area dell'input.

Questo influisce in maniera determinante all'invarianza.

INVARIANZA

È la capacità della CNN di riconoscere e classificare oggetti o pattern all'interno dei dati di input in maniera totalmente indipendente alle variazioni o trasformazioni che vengono applicate agli oggetti od ai pattern.

L'invarianza è particolarmente rilevante nelle attività di visione artificiale od image processing in cui gli stessi oggetti possono apparire in diverse posizioni, in diversi orientamenti, in scale diverse o in varie condizioni di illuminazione.

Le variazioni a cui possono essere soggetti gli oggetti ed i pattern nei dati di input sono trasformazioni.

Le trasformazioni alle quali una buona CNN dovrebbe essere invariante sono:
- **Traslazione**: una buona CNN deve essere in grado di riconoscere pattern ed oggetti indipendentemente dalla traslazione (spostamento, in posizioni diverse) di essi all'interno dell'input. Ciò si ottiene grazie ai **weight sharing** (condivisione dei pesi) nei livelli convolutivi. Condividendo gli stessi pesi (i filtri) nell'intero spazio di input, le CNN possono rilevare le stesse features indipendentemente della loro posizione.
- **Rotazione**: con la tecnica di **Data Augmentation,** aumentiamo i dati di input con versioni ruotate dei dati di input. Così la rete è esposta alle varie versioni ruotate dei dati di input imparando ad essere invariante alle rotazioni.
- **Scala**: grazie ai livelli di **Pooling** che ricampionano le feature map rendendo le CNN capaci di riconoscere oggetti su scale diverse.
- **Deformazione e distorsione**: **attraverso l'apprendimento gerarchico** le CNN possono catturare caratteristiche di

un livello superiore, più astratte, che sono meno influenzate da deformazione o distorsione locale
- **Rumore**: grazie sempre all'apprendimento gerarchico ed alle funzioni non lineari che consentono alla CNN di concentrarsi sulle caratteristiche più salienti scartando le info irrilevanti giudicate come rumore. Dunque le CNN sono invarianti al rumore.

Il grado di invarianza di una CNN dipende da molti fattori: qualità del data set di addestramento, tecnica di data augmentation, architettura di rete, dal task specifico che deve essere svolto dalla rete.

Non tutte le invarianze possono essere possibili, dobbiamo trovare un compromesso tra le invarianze della rete e la sensibilità della rete a rilevare pattern desiderati in base ai task specifici in cui la rete viene impiegata.

Weights Sharing

lunedì 6 maggio 2024 18:38

È una delle tecniche principali per avere l'invarianza nelle CNN.

Concetto: un filtro capace di estrarre una feature (es. un bordo verticale) in un particolare LOCAL RECEPTIVE FIELD potrebbe essere utile anche in **un'altra parte dell'immagine**.

Un filtro scorre sull'input generando una FEATURE MAP. Lo stesso filtro viene usato in differenti parti dell'immagine e i suoi parametri sono sempre gli stessi.

I filtri che generano le stesse feature map in un determinato livello sono **condivisi**.

Questo abbatte il numero di parametri che la rete deve imparare.

Il numero di parametri in una CNN non dipende dalla dimensione dell'input.

Il numero di parametri dipende esclusivamente :
- Dalla dimensione dei filtri
- Dal numero di filtri

Questo genera meno connessioni e meno calcoli e garantisce una maggiore velocità di addestramento delle CNN.

I parametri appresi son sfruttati in parti diverse dell'immagine di input o delle feature map di input quando parliamo di livelli deep della CNN.

ESEMPI DI CALCOLO DEI PARAMETRI E DIMESIONE OUTPUT VEDI SLIDES, MA SONO SEMPLICI.

$$f^{[l]} = dim.\ filtro$$
$$p^{[l]} = dim.\ padding$$
$$s^{[l]} = dim.\ stride$$
$$n^{[l-1]} = dim.\ input$$
$$n^{[l]} = dim.\ output$$
$$n_c^{[l]} = num.\ filtri$$

Ogni filtro ha dimensioni:

$$f^{[l]} * f^{[l]} * n_c^{[l-1]}$$

Dimensioni output livello convolutivo:

$$n^{[l]} = \left\lfloor \frac{n^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

Numero di parametri:

$$(f_w^{[l]} * f_h^{[l]} * n_c^{[l-1]} + 1) * n_c^{[l]}$$

N.B. larghezza e altezza del filtro potrebbero essere diversi fra loro

Downsampling e Upsampling

mercoledì 8 maggio 2024 12:59

La convoluzione non è l'unica operazione che può essere fatta all'interno di un layer convolutivo.

Ma vi sono anche operazioni di Upsampling e Downsampling.
Tali operazioni servono a modificare le dimensioni delle Feature Map in un CNN ed aiutano a creare una rappresentazione gerarchica delle feature estratte da un input.

Downsampling: catturare informazioni di alto livello per portarli ai livelli più DEEP della CNN.

Upsampling: ripristinare dettagli ed informazioni locali.

I layer che sono delegati a fare il Downsampling sono i così detti **livelli di Pooling.**

Il pooling aiuta infatti aiuta a rendere le Feature Map ad essere invarianti a piccole traslazioni spaziali nell'input.

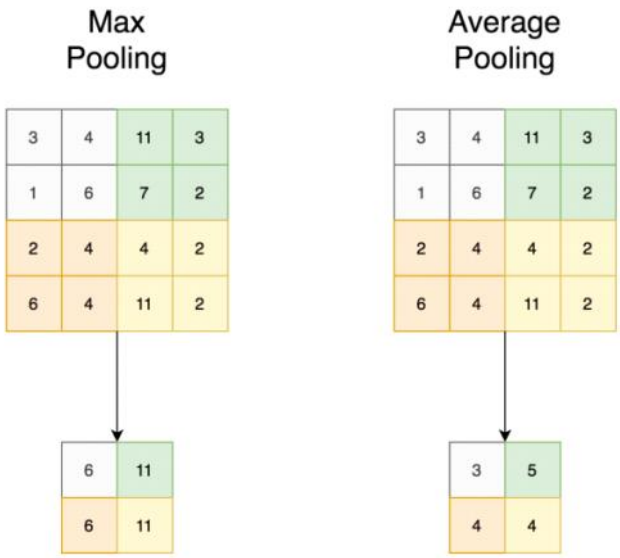
Se una determinata feature (un volto, sorriso ecc.) si sposta nell'input di una quantità non particolarmente alta in pixel, il valore dell'output corrispondente a quella regione non cambierà, e quindi avremo una invarianza del modello a queste traslazioni dovute a questi livelli di pooling.

L'invarianza del modello a questi spostamenti piò essere molto utile se siamo interessati alla presenza di una feature più che alla sua posizione.

L'operazione che viene fatta nei livelli di pooling è una operazione molto semplice ad esempio media, min, max ecc. al fine di associare 1 pixel di output ad un gruppo di pixel.

Il risultato è una diminuzione del volume di uscita al livello (delle dimensioni della feature map), e una conseguente **diminuzione dei parametri da apprendere**, con conseguente velocizzazione dei parametri da apprendere.

Max Pooling / Average Pooling



Quindi nei livelli di pooling downsampling consistono nel diminuire la dimensione delle Feature Map con queste operazioni matematiche.

UPSAMPLING

Consente di avere un volume di output maggiore del volume di input per quanto riguarda alle due prime dimensioni (altezza e larghezza non numero di feature map).

Avremo quindi delle feature map più grandi.

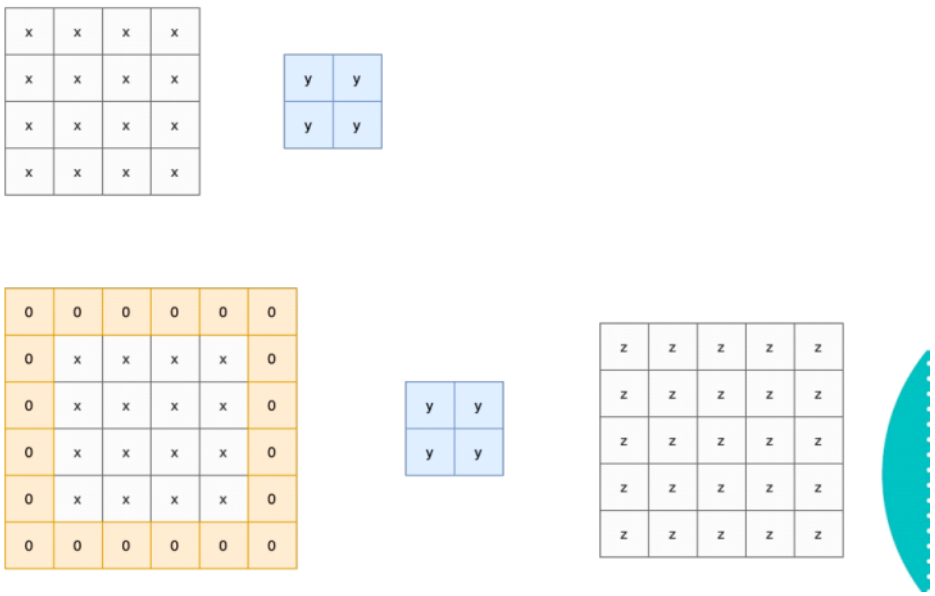
Lo si può ottenere con la convoluzione del layer trasposto, cioè una convoluzione che viene effettuata su dei dati di input lievemente modificati.

È un tipo di convoluzione che produce degli output di volume maggiore.

I livelli che fanno ciò vengono chiamati **Transposed Convolutional Layer.**

- Partendo da valori definiti di stride e padding:
  - Si "**allargano**" fra loro i pixel dell'immagine di una quantità pari ad (s − 1) (i pixel nuovi si settano a 0)
  - Si aggiunge una cornice di zeri di profondità pari a (f − p − 1)
  - Si fa la convoluzione con stride s' pari a 1 (si considera ogni pixel)
  - Dim. Output $(n_c - 1) * s + f - 2p$

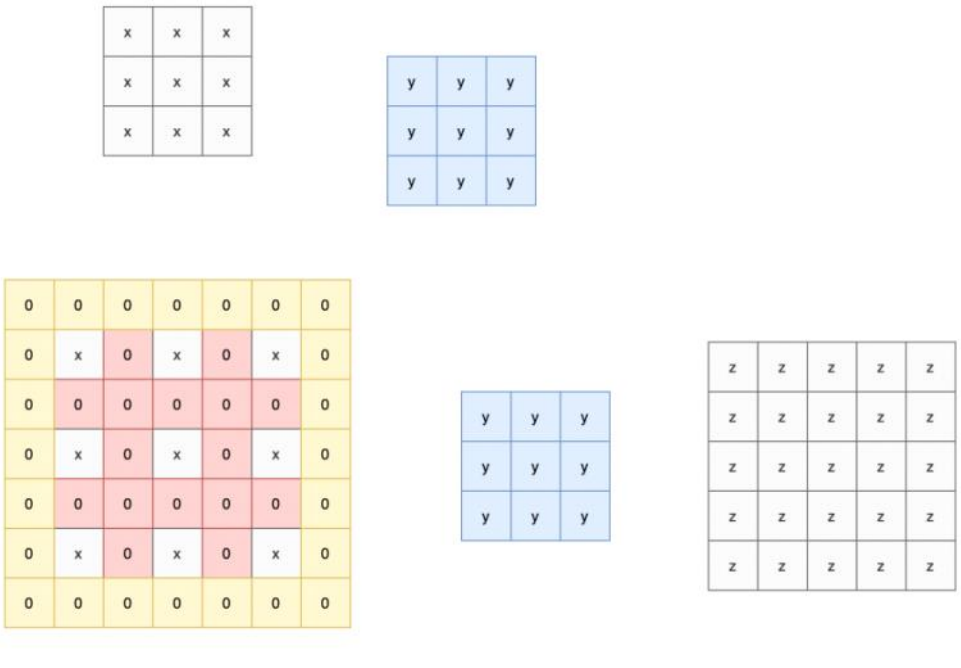
- Dim. Immagine: 4x4
- Dim. kernel: 2x2
- Padding: 0
- Stride: 1
- Dim. Output : 5x5



Una volta fatto l'upsampling andiamo ad eseguire la convoluzione sul nuovo input con lo stesso kernel e con stride pari a 1 (cioè si considerano tutti i pixel).

Nell'esempio abbiamo che l'output è di (4-1)*1 + 2 - 2*0 = 3 +2 = 5

- Dim. Immagine: 3x3
- Dim. kernel: 3x3
- Padding: 1
- Stride: 2
- Dim. Output : 5x5



In questo esempio invece otteniamo s-1 =1 quindi intervalliamo i pixel dell'input con 1 pixel di valore a 0. Aggiungiamo una cornice di zeri pari di profondità pari a 3-1-1 = 1 e poi possiamo fare la convoluzione con stride pari a 1 sul nuovo input ed otterremo un output di dimensione pari a (3-1)*2 + 3 -2*1 = 4 +1 = 5 che è più grande dell'input originale che era di dim 3.

FULLY CONNECTED

In una CNN possiamo trovare anche livelli fully connected, come abbiamo visto nelle reti neurali tradizionali e si possono trovare di solito nella parte finale di una CNN.

L'input di un livello FC va appiattito (flattened). Quindi quando ci arriva un input quadrato o cubico (ad es feature map volumentrica) va flattenizzato in un'unica dimensione.

Questi livelli FC sono quelli che si occupano di fare la classificazione finale e di dirci cosa effettivamente è stato trovato nell'immagine o nel segnale di input.

Addestramento di una CNN

mercoledì 8 maggio 2024 19:07

In una CNN i pesi appresi durante l'addestramento sono i parametri che catturano i modelli ed i pattern e le relazioni nei dati presenti nell'input.

Questi pesi svolgono un ruolo cruciale per le capacità della rete di fare previsioni accurate o di eseguire il task desiderato.

I pesi che vengono appresi durante l'addestramento sono proprio i parametri dei filtri/kernel, i pesi dei livelli FC, i bias ed i parametri di Normalizzazione e di Regolarizzazione che vedremo più in là.

Questi parametri vengono appresi durante l'addestramento tramite le solite tecniche di discesa del gradiente o varianti di essa con il solito obiettivo di ridurre al minimo una funzione di costo che quantifica quanto si discostano l'output previsto dalla rete e le etichette presenti all'interno dei dati.

Tutti questi parametri vanno addestrati iterativamente e che poi vanno a formare il modello finale che sarà in grado di identificare caratteristiche (feature) all'interno di un particolare input per poi espletare una probabilità od una classe di uscita.

Vediamo le fasi di come apprende una CNN:

- Algoritmo è sempre la **discesa del gradiente** (in pratica però vengono usate delle varianti)
  - **Step 1:** Setup architettura (es. 1 livello convolutivo con 1 filtro con kernel 3x3)
  - **Step 2**: Inizializzazione (es. 9 pesi del kernel casuali, 1 bias pari a 0)
  - **Step 3**: Scelta loss function (es. cross entropy)
  - **Step 4**: Forward pass (es. applicazione del filtro, supponiamo di avere feature map 30x30)
  - **Step 5**: Calcolo dell'errore
  - **Step 6:** Backpropagation (calcolo dei gradienti per tutti i 10 parametri)
  - **Step 7: Aggiornamento dei parametri**
  - Ripetere passaggi 4-7 fino a convergenza
  - **Step 8:** Valutazione del modello su un test set mai visto dalla rete

Esercitazione CNN e Keras

mercoledì 8 maggio 2024 19:44

Keras è un sottomodulo di TensorFlow e ci permette, tramite le sue API di alto livello, di creare delle reti neurali e quindi anche delle CNN.

Per questo primo esercizio utilizziamo ancora il dataset MNIST che contiene migliaia di immagini di cifre scritte a mano con le rispettive etichette (labels).

Il MNIST contiene 60000 immagini 28x28 ciascuna, quindi non hanno i canali RGB.
Quindi non sono degli input volumetrici ma bidimensionali.

Nelle reti di Keras dobbiamo definire anche la terza dimensione, quindi dovremo fare un reshape delle immagini, sia nel dataset di train che in quello di teste dire che sono (28x28x1).

Poi normalizziamo i valori dei pixel che vanno da 0 a 255 per passarli da 0 a 1.

Poi grazie alla funzione to_categorical() di keras.utils passiamo alla codifica one-hot li etichette, questo perché keras si aspetta una codifica di questo tipo per gli output della rete.

Fatto ciò possiamo iniziare a creare la rete convolutiva.

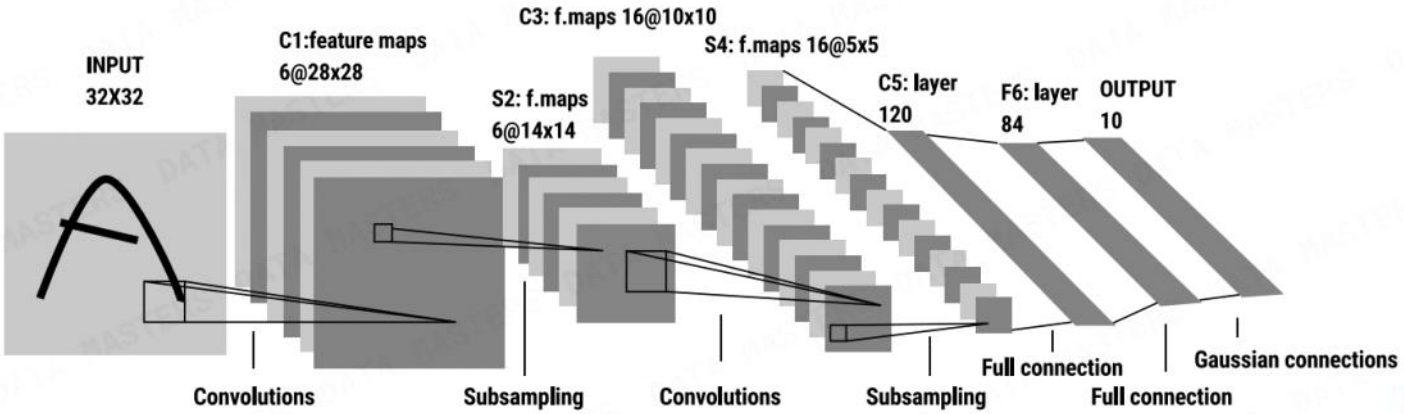
Una volta costruita, addestrata e valutata la nostra rete convolutiva possiamo utilizzarla per fare delle vere previsioni.

Esercitazione LeNet-5 con Keras

venerdì 10 maggio 2024 17:25

Rete ideata nel 1998 da LeCan.

Pietra miliare nelle CNN poiché ci consente di studiare le caratteristiche delle reti convolutive moderne, con tutti i layer che si utilizzano nelle CNN di oggi.



In inputa abbiamo immagini bidimensionali, quindi non volumetriche 32x32.

Nel primo layer convolutivo usiamo 6 filtri con un solo kernel, che danno in output 6 features map di dim 28x28.

Poi c'è un livello di Subsampling dove viene fatto il pooling che riduce le dimensioni degli output del primo livello, ottenendo 6 features map di dim 14x14, ed abbiamo sempre 6 filtri.

Poi applichiamo un livello convolutivo utilizzando 16 filtri ottenendo quindi 16 features map di dimensione 10x10.

Poi facciamo ancora un livello di pooling che come risultato diminuiscono la dim delle 16 features map a dim 5x5.

Successivamente applichiamo il flattening dove otteniamo un'array di 400 valori che contenevano le 16 features map.

Questo array viene dato in input a 120 neuroni in maniera fully connected.

Gli output di questo 120 neuroni passano ad un altro layer fully connected di 84 neuroni.

Gli output di questi 84 neuroni passano ad un ultimo layer di 10 neuroni in maniera fully connected.

Questo livello poi genera 10 valori di uscita.

Passiamo alla creazione di una rete LeNet con keras e la addestriamo sul dataset MINST.

Una volta creata la rete la addestriamo e valutiamo usando anche dei grafici per vedere l'andamento dell'accuratezza e della loss function lungo le epoche.

Dopodiché possiamo fare delle previsioni usando la nostra LeNet-5 e vedere quante ne azzecca.

Esercitazione Spam Detection

venerdì 10 maggio 2024 18:56

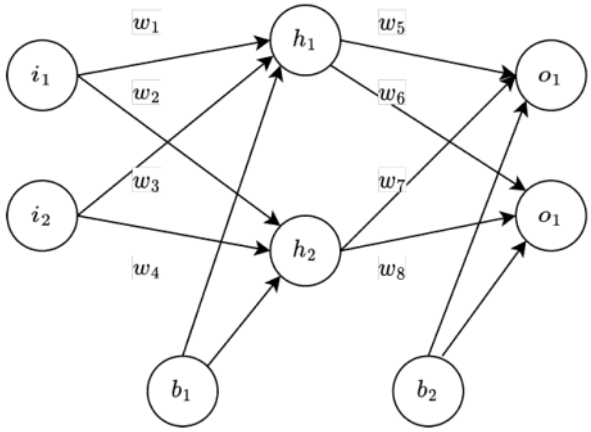
Vedi il notebook fatto ad hoc da te

Residual Networks

sabato 11 maggio 2024 22:03

Nella fase di backpropagation si aggiornano i parametri della rete neurale per diminuire l'errore.

Supponiamo di eseguire la backpropagation nelle seguente rete:



I parametri addestrabili sono [w1, w2, w3, w4, w5, w6, w7, w8] compresi anche i bias [b1, b2].

PROBLEMA NELLA BACKPROPAGATION:

- Il contributo dei gradienti (l'entità di aggiornamento dei pesi) è numericamente più alto nei livelli più vicini all'uscita
- Man mano che ci avviciniamo all'input della rete neurale i gradienti diminuiscono sempre di più
- La conseguenza di ciò è che l'aggiornamento dei parametri nei livelli più vicini all'input avviene più lentamente

Questo problema diventa rilevante in reti molto profonde (con tanti layers) dove nei livelli vicino agli input avremo aggiornamenti sempre più piccoli, fino a diventare trascurabili.

Ciò porta ad avere la convergenza molto lentamente fino ad addirittura non avere un processo di apprendimento in tali livelli vicino all'input (poiché i parametri si aggiornano ma di una entità trascurabile).

Questo fenomeno viene chiamato SVANIMENTO DEL GRADIENTE (VANISHING GRADIENT).

VANISHING GRADIENT:

Può succedere che i gradienti diventino sempre più piccoli fino a diventare pari a zero:

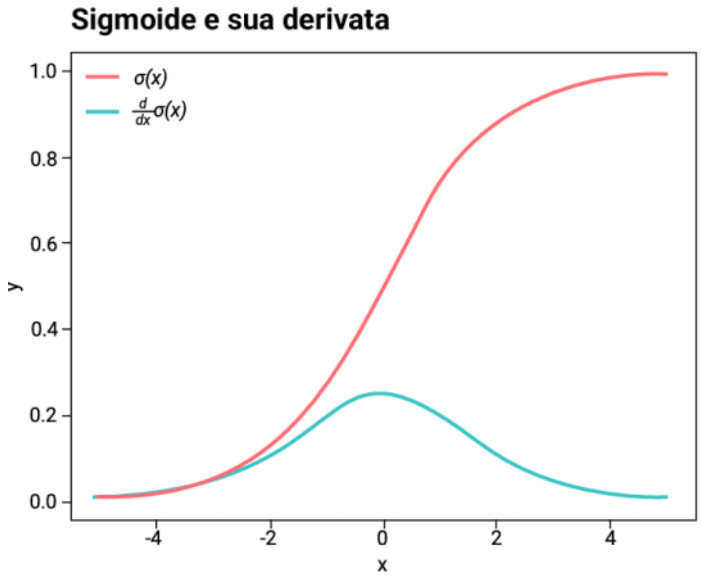
- I pesi non si aggiornano più o lo fanno molto lentamente

EXPLODING GRADIENT

Di contro può accadere che i gradienti diventino sempre più grandi fino a generare variazioni dei pesi molto elevate:

- Le prestazioni del modello non convergono più

Il problema del Vanishig Gradient è spesso associato a **funzioni di attivazione** come la sigmoide.



Vediamo che appunto la sigmoide prende in input i valori di x che spaziano tra [-inf, +inf] e li schiaccia tra [0, 1], inoltre la derivata della sigmoide ha un valore massimo pari a 0.25 il che significa che in backpropagation anche nei migliori dei casi avremo un abbassamento pari al 25% del valore originario che poi viene retropropagato. Il che significa che la funzione sigmoide abbassa del 75% la quantità di informazioni che viene retropropagata.

Dunque con funzioni di attivazione come la sigmoide abbiamo un ulteriore abbassamento della quantità di gradiente che viene retropropagato con un VANISH GRADIENT che viene manifestato in maniera molto più rapida.

GOING DEEPER

Aggiungere livelli ad una rete neurale, dal punto di vista concettuale, dovrebbe rendere migliori le prestazioni di una CNN.

Dal punto di vista pratico ed empirico, a causa del Vanishing Gradient o del Exploding Gradient, si nota che in reti più DEEP, dopo una certa fase di discesa, l'accuracy tenderà a scendere man mano che aggiungiamo livelli alla nostra rete.

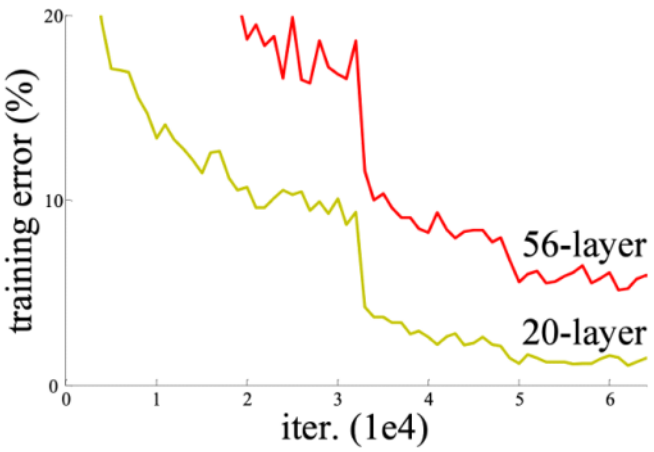
Cioè la performance della rete degrada rapidamente man mano che aumentiamo il numero di livelli.

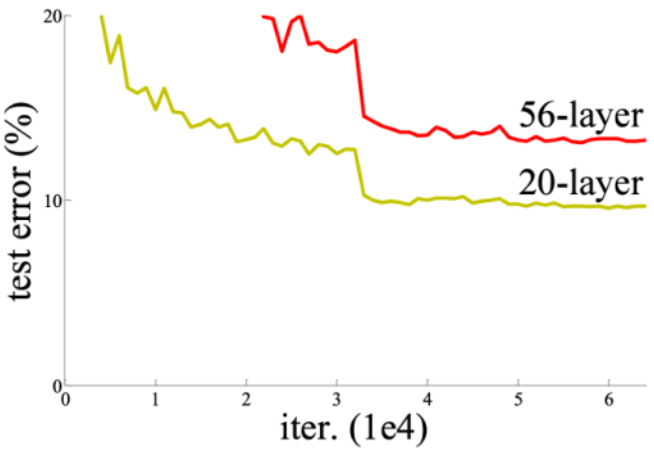
I ricercatori che per primi hanno rilevato questo problema lo chiamano "DEGRADATION PROBLEM".

DEGRADATION PROBLEM:

Supponiamo di avere una rete di 20 layer ed una rete di 56 layer che si addestrano sullo stesso task.
Intuitivamente dovremmo pensare che la rete con 56 layer dovrebbe performare meglio.

In realtà ciò che viene verificato dai ricercatori è che l'errore (loss function dopo le varie epoche) è più basso nella rete con soli 20 layers, sia in fase di training sia in fase di test.





Quello su cui si basa il framework ideato dai ricercatori è che: abbiamo una rete shallow (20 layers) ed una rete deep (56 layers), quando aggiungiamo più livelli ad una rete shallow in teoria possiamo fare in modo che i livelli extra non facciano nulla se non copiare l'output della rete shallow nei livelli più deep, questa tecnica si chiama I**dentity Mapping.**

In un caso analogo la rete più profonda dovrebbe funzionare bene come quella shallow.

Quindi la rete deep se abbiamo un Identity Mapping (cioè se portiamo i risultati della rete shallow nei livelli più deep della rete deep) dovremmo avere un funzionamento della rete deep analogo a quello della rete shallow.

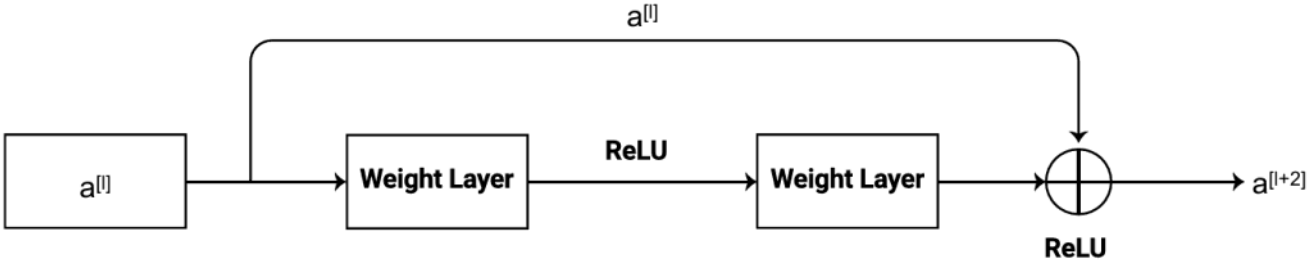
In pratica è molto più difficile trovare delle buone soluzioni per le reti deep, cioè soluzioni che dopo la fase di addestramento della rete deep
raggiungono questo identity mapping.

In altre parole, con gli attuali metodi di addestramento delle reti neurali è difficile far imparare a quel gruppo di livelli più deep il così detto Identity Mapping per avere una performance almeno uguale a quello della rete Shallow.

RESIDUAL BLOCK:

I ricercatori che per primi hanno notato il degradation problem propongono un framework per ovviare tale problema.
Propongono di applicare il BLOCCO RESIDUALE.

Tale blocco consiste nella propagazione in avanti, facendo saltare un livello o più livelli, ad un certo input (output) di un determinato livello in maniera che arrivi direttamente (tramite quella che viene chiamata skip connection) ai livelli più deep della rete per poi effettuare una concatenazione del segnale con il percorso normale all'interno di una funzione di attivazione (ad es. ReLu).



Quindi i fase di applicazione del livello di attivazione abbiamo due contributi, il primo proveniente dal percorso feedforward normale della rete ed il secondo proveniente dalla skip connection del livello precedente L.

L'idea è quella di passare ai livelli più deep l'informazione dei livelli più shallow.

L'uscita del livello L+2 sarà:

(Pesi del livello l+2) * (l'uscita del livello (l+1)) + (bias del livello l+2) + l'uscita del livello L

$$a^{[l+2]} = g(W^{[l+2]}a^{[l+1]} + b^{[l+2]} + a^{[l]})$$

L'introduzione di un blocco residuale ad una rete con architettura deep consente di apprendere una **mappatura residua** invece di apprendere la **mappatura diretta**.

Mapping diretto:
In una rete neurale feedforward standard, cioè con un percorso unico diretto, ogni livello trasforma in sequenza i dati di input per estrarre rappresentazioni gerarchiche dei dati. In una attività come la classificazione delle immagini ogni livello acquisisce delle caratteristiche diverse delle immagini (dai semplici bordi a feature complesse come forme e oggetti), la rete complessiva mira ad apprendere proprio la mappatura. Dall'input andiamo a cercare di fare imparare alla rete la mappatura diretta cioè quella che mi consente di mappare direttamente il risultato finale.

Da x la rete impara H(x)

Mapping Residuale:
Parliamo di mapping residuale quando la rete al posto di imparare direttamente il mapping diretto va ad imparare la mappatura residua, la mappatura residua è data dalla funzione ipotesi meno l'input. La rete in questo caso impara quanto differisce l'output dall'input, cioè il residuo.
Il residuo è quello che viene effettivamente calcolato tramite l'implementazione del blocco residuale.

Infine, l'output finale è calcolato come: input + il mapping residuale.

- Mapping diretto

$$x \implies H(x)$$

- Mapping **residuale**

$$F(x) = H(x) - x$$

- Successivamente l'output è calcolato come:

$$H(x) = F(x) + x$$

Le SKIP CONNECTIONS:

Le skip connection servono proprio a fare in modo che il ramo del blocco residuale si addestri imparando il residuo.

In fase di backPropagation abbiamo una suddivisione dei gradienti che una volta arrivati alla funzione di attivazione, che unisce l'out dal ramo feedforward ed il ramo della skip connection, giunti a tale punto anche i gradienti prendono strade diverse. Consentendo di retropropagare l'informazione saltando dei livelli da un lato, ed invece dall'altro lato consentendo al ramo normale di imparare/di addestrarsi sul residuo.

Dunque quello che avviene non è più l'addestramento normale col fine di imparare il mapping diretto H(x) ma l'addestramento avviene su F(x) dove F(x) è il residuo H(x) - x.

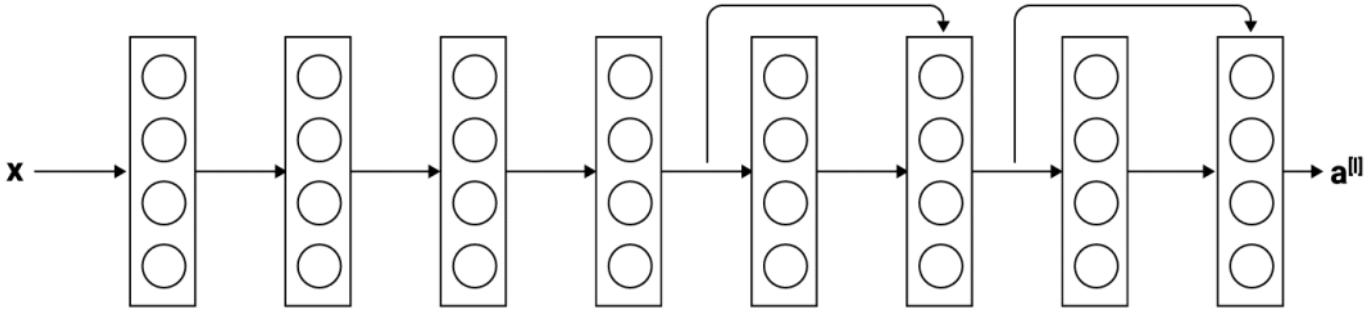
Questo rappresenta anche una soluzione per il Vanishing Gradient dal momento che i gradienti vengono propagati all'indietro in maniera efficiente in modo che conservino di più il contenuto informativo anche nelle reti più deep.

La rete durante l'addestramento (backpropagation) ottimizza i pesi del blocco residuale per ridurre al minimo la differenza tra l'0utput desiderato e l'output effettivo. In questo modo imparando il residual mapping la rete può concentrarsi sul perfezionamento dell'input iniziale piuttosto che imparare a mappare direttamente, l'input sull'output desiderato.

Questa tecnica dell'introduzione del blocco residuale si è dimostrata molto efficace anche per l'addestramento di reti molto profonde.

RESIDUAL NETWORK

Una rete residuale è una rete che presenta uno o più blocchi residuali che in fase di backPropagation consentono di addestrare il ramo feedforward sul mapping residuale e in più consentono di portare all'indietro in maniera più efficiente le uscite di livelli più deep in maniera da mitigare al massimo il problema del Vanishing Gradient.

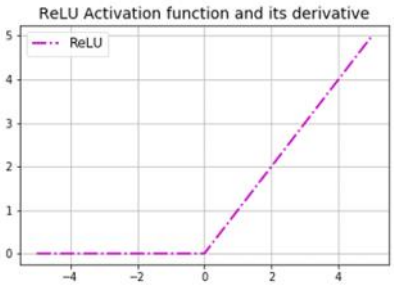


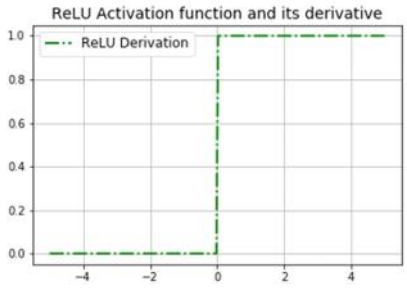
Funzione di attivazione ReLu:

Perché viene utilizzata la funzione di attivazione ReLu?

Perché come abbiamo detto prima la funzione sigmoide taglia il gradiente del 75%, ciò significa che i gradienti saranno tagliati al massimo di 0.25.

Invece utilizzando una ReLu in fase di backpropagation abbiamo una quantità di gradiente che viene retropropagata pari a 1, quindi in fase di backpropagation la ReLu si comporta meglio per il problema di Vanishing Gradient.

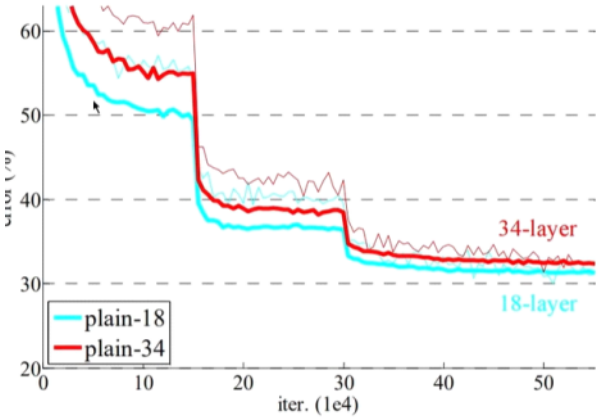


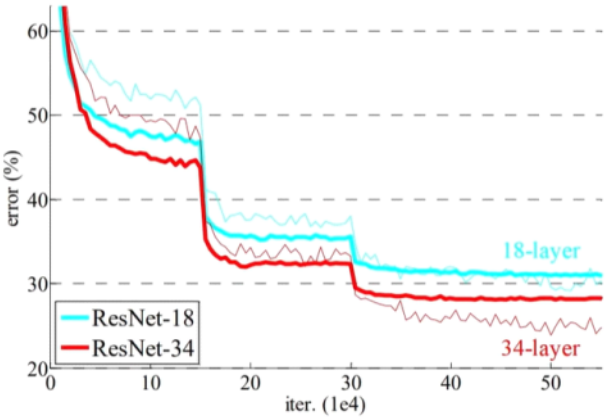


Conclusione:

Grazie all'introduzione delle skip connection (blocchi residuali) i livelli più deep avranno almeno il livello L da cui partire evitando il degradation problem (calo delle performance).
E quindi le reti neurali più deep hanno performance almeno pari al loro corrispettivo shallow.

Nel caso migliore invece i livelli più deep miglioreranno le performance del livello L rispetto alla loro controparte shallow andando a modificare i propri parametri per imparare il mapping residuale e quindi migliorare le performance della rete neurale più deep rispetto alla controparte shallow.





Di fatto l'applicazione del blocco residuale porta a miglioramenti delle reti più deep rispetto alle loro controparti shallow.

Esercitazione Residual Networks

lunedì 20 maggio 2024 22:21

Vedi jupyter notebook

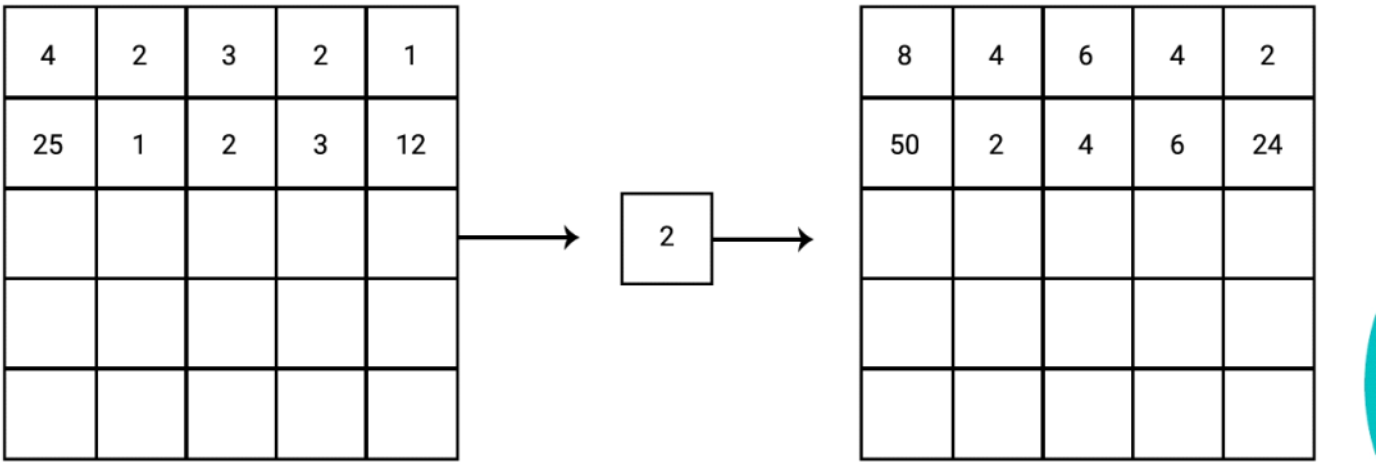
1x1-Conv-Inception

martedì 21 maggio 2024 22:46

1x1 CONVOLUTION

La dimensione del kernel è 1x1 che va a scorrere lungo la finestra di input.

Pertanto, questo tipo di convoluzione corrisponde alla moltiplicazione per uno scalare dei dati di input (prodotto scalare).



Quindi il risultato di questa convoluzione, proprio come le altre, serve per estrarre delle features dalle dai dati e come risultato otteniamo un feature map che poi entrerà come input in una funzione di attivazione con il bias.

Questo tipo di convoluzione non considera le relazioni di tipo spaziale, il kernel più piccolo che ce le ha è il kernel 3x3 che ha nozioni di sopra, sotto, destra e sinistra rispetto al pixel centrale.

USI COMUNI DELLA CONVOLUZIONE 1X1:
- Riduzione della dimensionalità: Utilizzando una convoluzione 1x1 possiamo ridurre il numero di canali (il volume) delle feature map in uscita dal layer convolutivo 1x1. Ciò è utile quando dobbiamo ridurre il numero di parametri all'interno della nostra rete.
  - Aggiungere non linearità: dal momento che possiamo applicare trasformazioni non lineari all'input direttamente tramite la convoluzione 1x1 e questo può aiutare a catturare schemi (pattern) molto complessi.
  - Feature combination: se usate in combinazione con convoluzioni più grandi ad es 3x3, le convoluzioni 1x1 possono essere utilizzate per combinare features di diversi canali (ad es RGB)

L'utilizzo più comune delle convoluzioni 1x1 applicate con tutte e 3 queste idee, è nella progettazione dei moduli **Inception** che sono i componenti fondamentali della architettura innovativa di GoogleNet, dove si utilizzano massicciamente le convoluzioni 1x1 per andare a ridurre la dimensione delle feature map prima di applicare convoluzioni più grandi per cercare di estrarre le feature più complesse dai dati di input.

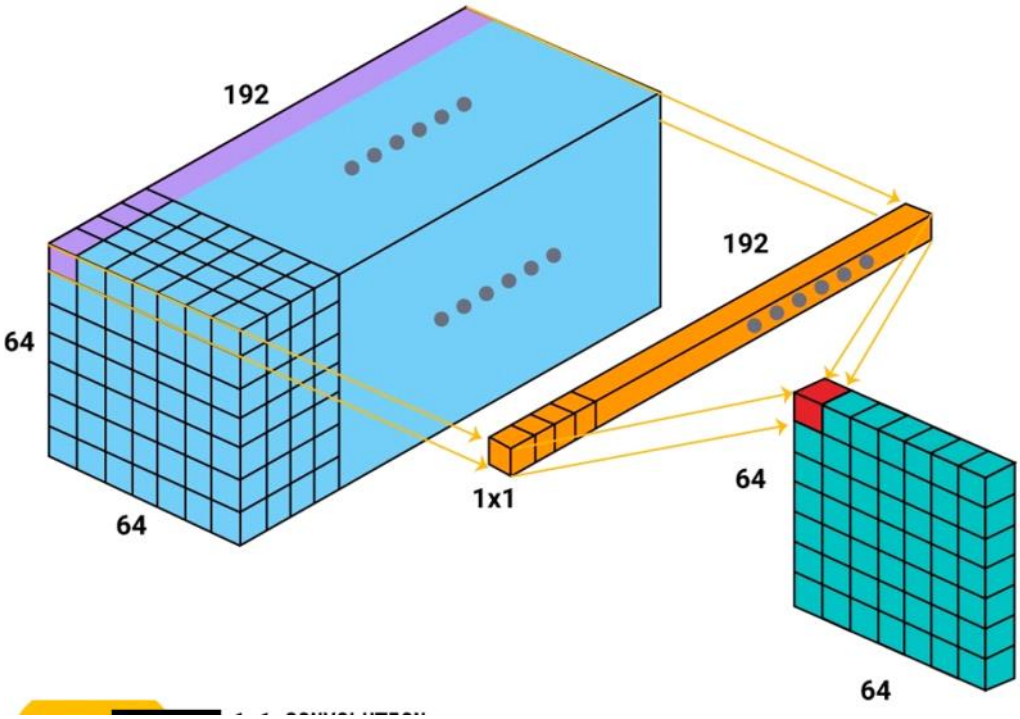
DIMENSIONALITY REDUCTION

Il numero di features map che ottengo dopo una convoluzione dipende unicamente da quanti filtri ho usato nello strato convolutivo. Ad es se uso 3 filtri di dim 3x3x3 su una immagine RGB ciò che ottengo saranno 3 features map, che poi possono essere impilate e fatte scorrere da filtri di volume 3.

Le convoluzioni 1x1 possono essere utilizzate strategicamente per ridurre il volume delle features map.

Ad esempio abbiamo una feature map con un volume pari a 192, quindi abbiamo 192 features map ciascuna delle quali con dimensione 64x64 che vanno in pasto alla convoluzione 1x1, che avrà anche essa un volume pari a 192.

Nel layer convolutivo 1x1 si va in profondità vedi il quadratino in alto a sinistra del tensore nell'immagine.



Il fatto che andiamo ad operare tale convoluzione con un solo filtro 1x1x192 ci restituisce una sola feature map di dim 64x64 .

EFFETTI DELLA CONVOLUZIONE 1X1:

Dunque le convoluzioni 1x1 vengono usate per modificare il numero di canali di profondità in uscita.

Gli effetti dell'applicazione delle convoluzioni 1x1 sono:

- Riduzione della complessità computazionale: le architetture che hanno un numero elevato di filtri in ogni livello possono diventare molto difficoltose da addestrare e da eseguire. Andando ad eseguire le convoluzioni 1x1 per ridurre il numero di canali nelle features map intermedie andiamo anche a ridurre la complessità computazionale di tutta la rete, portando a tempi di addestramento molto più rapidi ed a requisiti di memoria che sono ridotti.
- Efficienza dei parametri: il numero di parametri che si possono apprendere in un livello convolutivo è direttamente proporzionato al numero di canali di input e di output. L'utilizzo di convoluzioni 1x1 comporta un numero inferiore di parametri nei livelli successivi, quindi abbiamo una maggiore efficienza dei parametri in termini di numero di parametri che possono essere addestrati. Questo può essere molto utile per la riduzione della capacità computazionale quando abbiamo risorse limitate di memoria e capacità computazionale.
- Abbassamento del rischio di overfit: man mano che abbiamo troppi parametri all'interno della rete, soprattutto nelle reti deep, il rischio di overfitting aumenta con il numero di parametri che vengono addestrati all'interno della rete. Andando a ridurre la dimensionalità delle feature map le convoluzioni 1x1 aiutano anche a controllare la capacità del modello di generalizzare rendendolo meno incline all'overfit.
- Livelli "bottleneck": in alcune architetture si possono incontrare questi livelli di bottleneck che di solito sono seguite da operazioni molto costose con filtri più grandi, perciò è dove le convoluzioni 1x1 vengono applicate proprio per ridurre significativamente il numero di canali e rendendo così più efficiente l'architettura.
- Feature extraction: la riduzione della dimensionalità delle feature map operata dalle convoluzioni 1x1 aiuta a creare feature map che conservano solo informazioni essenziali. Andando a ridurre il numero di canali le caratteristiche ridondanti o quelle meno importanti sono di solito scartate e quindi la rete può concentrarsi su caratteristiche più importanti. Questo downsampling lo si fa anche con i livelli di pooling.

CONVOLUZIONE 1X1 VS. POOLING

	1x1	Pooling
Scopo	Feature combination e dimensionality reduction	Downsampling e quindi dim. reduction
Operazione	Convoluzione con il più piccolo receptive field	Max o media su receptive field più ampi
Output	Potenzialmente possono conservare più informazioni	«Riassumono» le info potenzialmente con perdita
Impatto sui parametri	Numero minimo di parametri	Non hanno parametri

Possiamo dire che le convoluzioni 1x1 principalmente si concentrano sulla riduzione del numero di canali preservando le informazioni al dettaglio, mentre le operazioni di Pooling riducono la dimensionalità sulle prime due dimensioni ottenendo si maggiore una efficienza computazionale però a scapito di qualche dettaglio informativo identificato dalla rete fino a quel momento.

NON LINEARITà

- Le 1x1 quindi vengono usate per **aggiungere non linearità ai filtri**
- Le conv 1x1 aggiungono non linearità perché sono **dei livelli convolutivi veri e propri**
  - Conv 1x1 di per sé è lineare
  - Dopo la conv 1x1 si applica la **funzione di attivazione** che aggiunge non linearità
- La non linearità in generale:
  - Aumenta il potere di rappresentazione di una rete
  - Aiuta contro l'overfit
  - Aiuta con la cattura di informazioni complesse

Sono le funzioni di attivazione ad aggiungere non linearità non la convoluzione di per sé poiché sono operazioni lineari (somma di prodotti). Le funzioni di attivazione sono una caratteristica peculiare delle reti neurali e non della operazione di convoluzione.

Ad esempio la funzione di attivazione ReLu introduce **non linearità** disattivando i valori negativi e facendo passare al layer successivo solo i valori positivi o pari a zero.

Le funzioni non lineari nelle reti neurali aiutano a catturate caratteristiche complesse tipiche del mondo reale che non sono relazioni lineari tra i dati di input ed i labels. Questo aumenta la capacità di rappresentazione delle reti neurali.

Inoltre nell'Image processing (uno dei principali utilizzi delle reti convolutive) i confini decisionali che separano classi diverse sono molto complessi ed altamente non lineari e grazie alle funzioni di attivazione che introducono questa non linearità consentono alla rete di aumentare il proprio potere

di rappresentazione portando a migliori prestazioni in termini di classificazione.

Grazie alle funzioni di attivazione andiamo a ridurre un certo livello di rumore nel modello che rendono il modello più robusto e meno incline all'overfitting.

Grazie alla non linearità il modello è in grado di catturare feautures (caratteristiche) molto complesse nei dati.

FEATURE COMBINATION

- Una conv 1x1 può essere vista come un **aggregatore** di canali
  - È sostanzialmente una **somma ponderata** degli output di un canale
- La feature combination è l'unione o fusione di informazioni provenienti da diverse feature map in un'unica feature map
- Di norma le feature composite sono più ricche e più informative delle feature di partenza

- Tutte queste caratteristiche sono state utilizzate nell'**Inception Block**

Quando parliamo di Feature Combination parliamo di fusione/unione di informazioni da diverse feature map, che formano il volume, in una rappresentazione unificata.

Può succedere che alcune di queste feature map (nel tensore) possano essere in grado di rappresentare aspetti diversi delle stesse informazioni sottostanti (ad es da una immagine) per esempio una feature map potrebbe evidenziare i bordi di un oggetto, mentre un'altra si concentra sulla sua texture e così via. La combinazione di queste caratteristiche complementari può portare ad una rappresentazione più completa ed informativa in generale dei dati di input.

In sintesi la Feature Combination, la Dimensionality Reduction e la Non Linearità introdotte dalle convoluzioni 1x1 migliora le capacità delle reti convolutive consentendo loro di acquisire informazioni diverse ed interconnesse tra loro a partire dai dati di input.

I primi ad aver utilizzato in maniera massiva questo tipo di convoluzione sono stati gli ingegneri di Google quando hanno introdotto la GoogleNet che fa utilizzo dei moduli Inception che a loro volta utilizzano le convoluzioni 1x1.

Queste convoluzioni 1x1 vengono utilizzate col fine di migliorare le prestazioni delle reti convolutive ed il flussi con cui le informazioni viaggiano attraverso la rete stessa.

Inception Block

mercoledì 22 maggio 2024 19:23

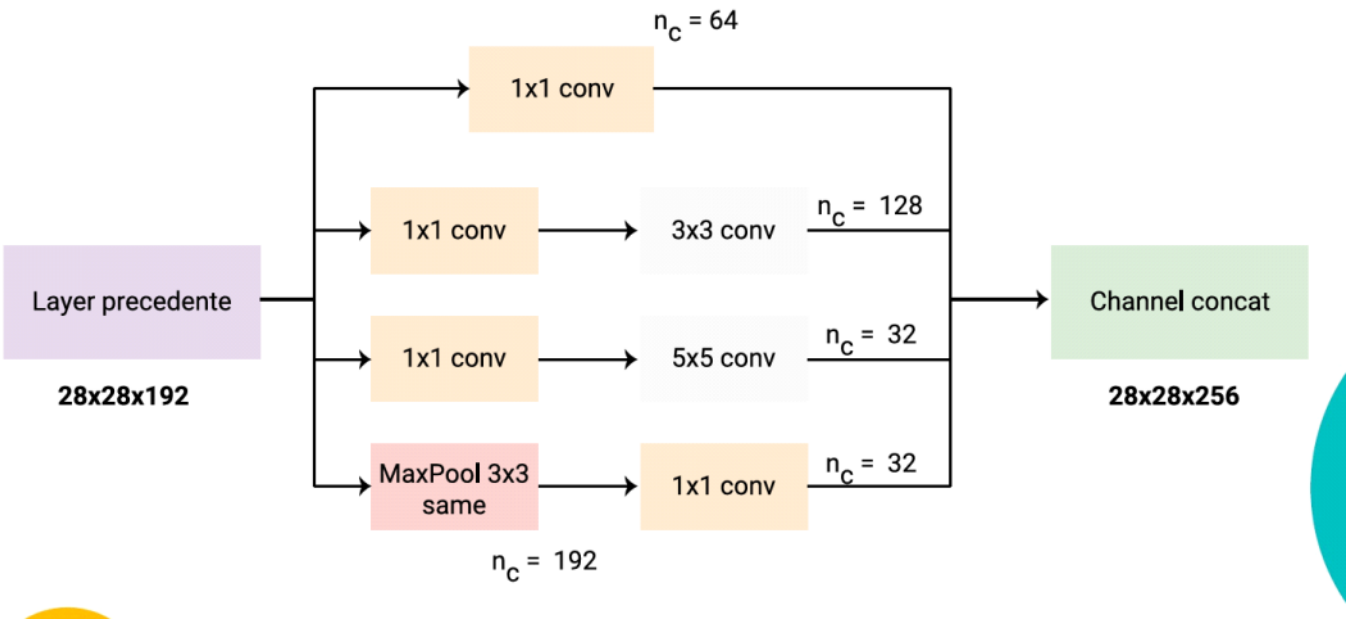
Idea dell'Inception Block: Comprendere features LOCALI contemporaneamente a feature GLOBALI, proprio grazie all'utilizzo delle convoluzioni 1x1.

L'idea di base dell'Inception Block è quello di parallelizzare delle operazioni che poi verranno concatenate in un unico leyer di uscita.

Useremo:
- Convoluzioni 1x1 con 64 filtri in uscita
- Convoluzioni same 3x3 con 128 filtri in uscita
- Convoluzione same con 32 filtri 5x5 in uscita
- MaxPooling

L'uscita di questo Inception Block è 28x28x256.

L'idea è appunto quella di PARALELLIZZARE queste convoluzioni diverse al posto di usare un'unica convoluzione, che poi saranno concatenate.



Vediamo che il primo percorso entra in una convolutiva 1x1 con un numero di filtri 1x1 pari a 64.

Poi abbiamo 2 rami con una sola convoluzione 1x1 per introdurre Feature Combined, Non Linearità e Dimensionality Reduction per poi passare la feature map ottenuta a delle convoluzioni same, 3x3 con 128 filtri nel secondo ramo e 5x5 con 32 filtri nel terzo ramo.

Infine nell'ultimo ramo facciamo il MaxPooling con un filtro 3x3 same, logicamente fatti sulle 192 dimensioni del tensore di input, quindi con 192 filtri e poi nello stesso ramo si fanno convoluzioni 1x1 con 32 filtri 1x1.

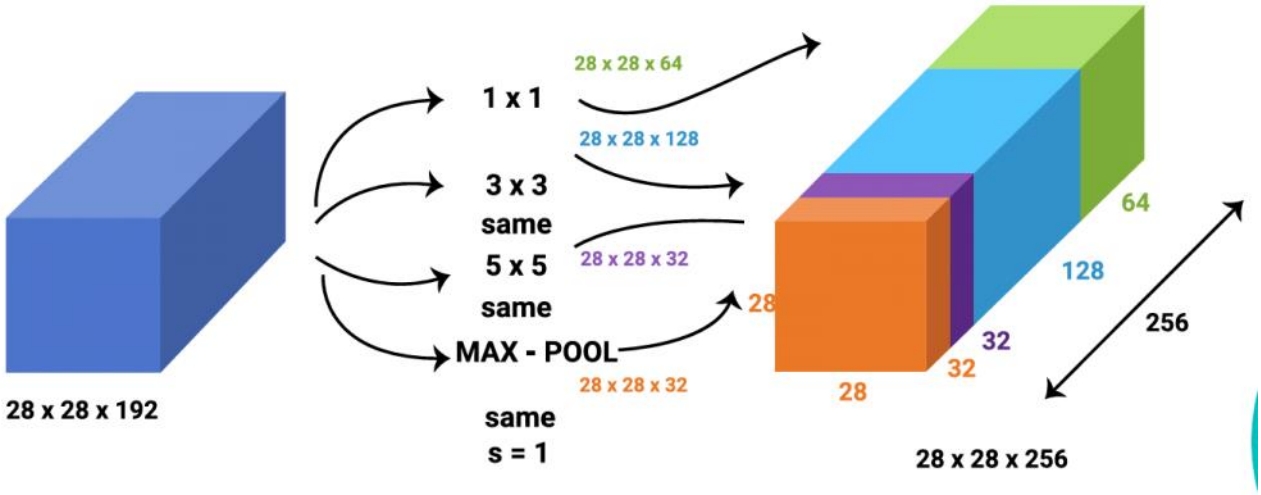
Per ultimo CONCATENIAMO tutte le feature maps ottenute dai 4 branch ottenendo un volume pari a 256 (64+128+32+32=256), che è il volume di uscita dell'Inception Block.

L'Inception Block consente alla rete di acquisire features in scale diverse (small scale e large scale). I percorsi paralleli con diverse dimensioni di filtri consentono al modello di selezionare le caratteristiche più rilevanti in maniera adattiva a varie risoluzioni spaziali.

Le convoluzioni 1x1 utilizzate all'inizio dei branch aiuta no anche a ridurre la complessità computazionale.

Andando ad impilare insieme più blocchi Inception viene realizzato il modello GoogLeNet, creando quindi una rete molto deep ed efficiente, andando ad apprendere delle rappresentazioni dei dati input molto ricche, essendo molto efficiente per compiti come: classificazione delle immagini ed l'object recognition.

Vediamo L'inception Block in un altro grafico:



COSTO COMPUTAZIONALE DI UNA SOLA CONVOLUZIONE 5X5 SAME (SENZA CONVOLUZIONE 1X1 PRECEDENTE):

Formule per il calcolo della dimensione dell'output e del numero di parametri:

$f^{[l]} = dim.\ filtro$
$p^{[l]} = dim.\ padding$
$s^{[l]} = dim.\ stride$
$n^{[l-1]} = dim.\ input$
$n^{[l]} = dim.\ output$
$n_c^{[l]} = num.\ filtri$

Ogni filtro ha dimensioni:
$f^{[l]} * f^{[l]} * n_c^{[l-1]}$

Dimensioni output livello convolutivo:

$$n^{[l]} = \left\lfloor \frac{n^{[l-1]} + 2p^{[l]} - f^{[l]}}{s^{[l]}} + 1 \right\rfloor$$

Numero di parametri:

$$(f_w^{[l]} * f_h^{[l]} * n_c^{[l-1]} + 1) * n_c^{[l]}$$

N.B. larghezza e altezza del filtro potrebbero essere diversi fra loro

- Calcoliamo il **numero di parametri** in un livello convolutivo di una CNN
  - s = 4
  - p = 0
  - input = 227x227x3
  - n. filtri = 96
  - dim. filtri = 11x11x3

- Calcoliamo innanzitutto la dimensione dell'output:
  - (227 – 2*0 – 11)/4 + 1 = **55**
  - Avremo un output di **55x55x96**

Se utilizziamo 32 filtri di dim 5x5x192 abbiamo (5*5*192*32=153.600+1= 153.601 parametri addestrabili)

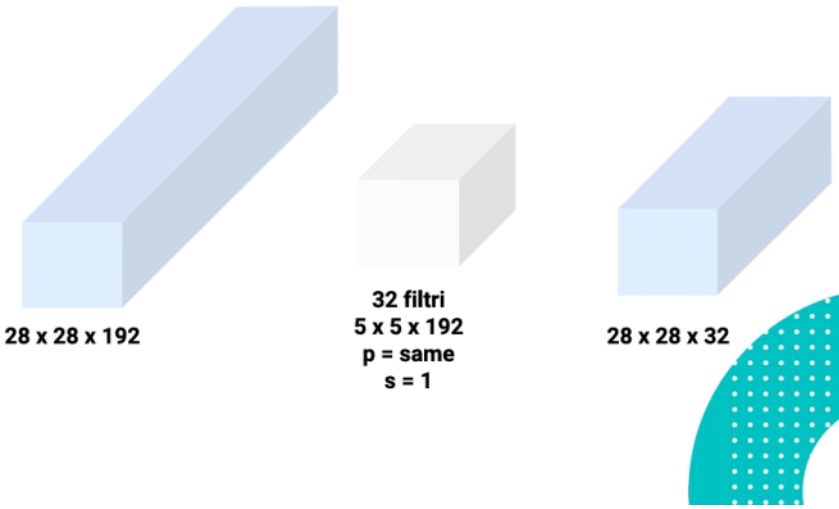
28-0-5/1 = 23 +1 = 24 avremo un output pari a 24x24x32

Tuttavia nell'esempio che vediamo il pdding è uguale a SAME: "same" results in padding the input such that the output has the same length as the original input.

Quindi l'output sarà di dim uguale a 28x28x32!

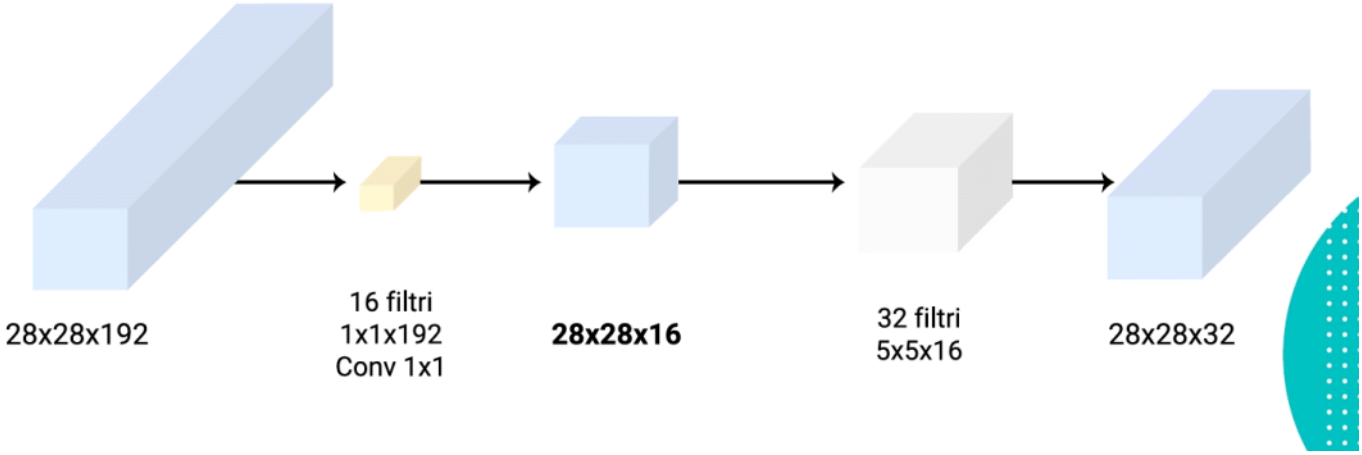
Il numero di moltiplicazioni da effettuare sono: 28*28*32*5*5*192=120.422.400

- Costo computazionale del **solo** blocco convolutivo 5x5 same
  - Input: 28x28x192
  - 32 filtri 5x5x192
  - Output: 28x28x32
- Num. Totale di moltiplicazioni da effettuare
  - 28x28x32x**5x5x192 =~120M**



ORA VEDIAMO IL **COSTO COMPUTAZIONALE** CON LA CONVOLUZIONE 5X5 SAME MA PRECEDUTA DALLA CONVOLUZIONE 1X1:

- Usando una convoluzione 1x1 (bottleneck):



Usando 16 filtri 1x1 e con padding same otteniamo 16 feature maps 28X28 sulle quali poi andiamo ad applicare la convoluzione 5x5 usando 32 filtri e con padding=same ottenendo infine 32 feature maps di dim 28x28.

Il numero di moltiplicazioni in questo caso sono:

1*1*192*16*28*28=2.408.448 di moltiplicazioni nella convoluzione 1x1

5*5*16*32*28*28=10.035.200 di moltiplicazioni nella convoluzione 5x5

Totale: 2.408.448+10.035.200=12.443.648 moltiplicazioni che sono di gran lunga inferiori a 120M

12M << 120M

Quindi introducendo un livello di bottleneck (convoluzione 1x1) riduciamo la complessità computazionale come già avevamo detto.

Inoltre anche il numero di parametri da addestrare diminuisce:

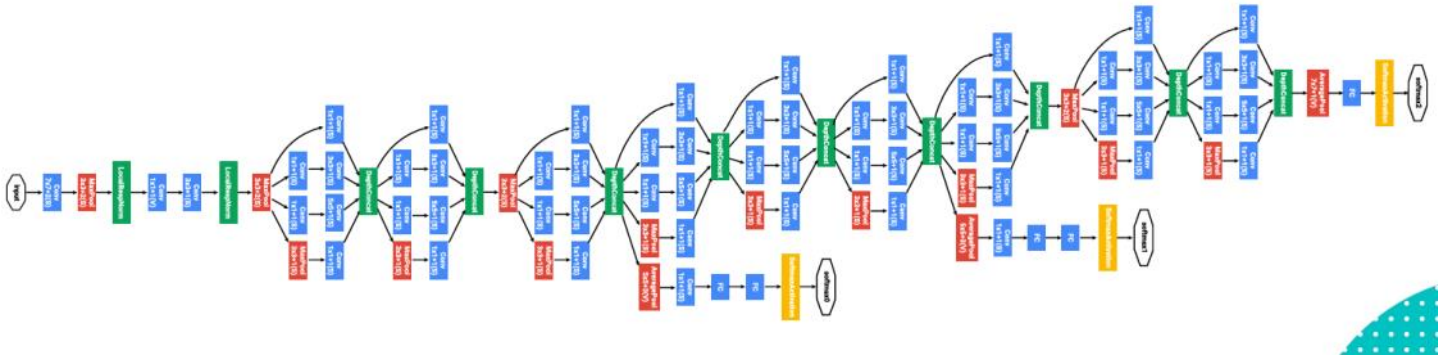
1*1*192*16 = 3.072 + 1=3.073 , 5*5*16*32=12.800 + 1=12.8001

Totale: 3.073+12.801=15.874 che sono molto inferiori ai 153.601 precedenti.

<u>Dunque riduciamo sia le operazioni da svolgere sia il numero di parametri da addestrare.</u>

Senza la convoluzione 1x1 si ha una esplosione di numero di operazioni e di numero di parametri da addestrare, quindi una esplosione computazionale e le convoluzioni 1x1 servono appunto a ridurla.

INCEPTION NETWORK (GOOGLE-NET)



Questa rete introduce i blocchi Inception che oprano la riduzione di parametri e di operazioni riducendo la complessità computazionale.

Inoltre abbiamo anche la presenza di **<u>"side branch"</u>** che sono classificatori ausiliari che formano dei layer di output dove vengono calcolati i

gradienti iniziali e che aiutano a mitigare la scomparsa del gradiente durante l'addestramento poiché dalla loro uscita (essendo più vicini all'input) e quindi il gradiente ha meno spazio (livelli) da attraversare durante la retropropagazione e quindi aiutano ad un addestramento delle rete più stabile evitando il problema del Vanishing Gradient tipico delle reti deep.

Abbiamo anche l'utilizzo di layer di Pooling finali anziché layer fully connected, questo per ottenere una lunghezza fissa delle feature maps oltre che a fare la dimensionality reduction e quindi anche qui ridurre il numero di parametri e di operazioni.

Tale rete è stata una delle prime se non la prima ad introdurre 22 livelli che all'epoca nel 2014 era molto deep. Grazie a questa architettura GoogLeNet è stata in grado di avere prestazioni molto elevate ma con molti meno parametri rispetto ad altre architetture più tradizionali.

Esercitazione Inception

giovedì 23 maggio 2024 23:09

Vedi jupyter notebook

U-Net

giovedì 23 maggio 2024 19:13

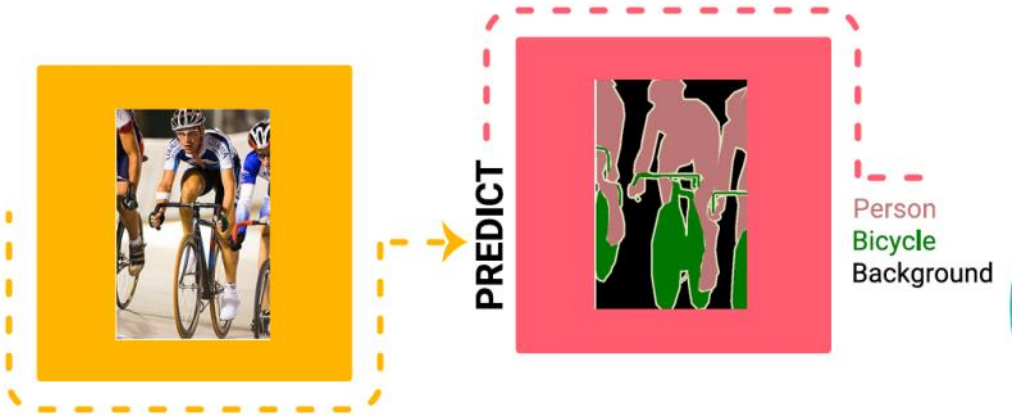
La U-Net è un'architettura per le reti convolutive inizialmente progettata per la segmentazione semantica in ambito biomedicale.

Nella segmentazione semantica l'obiettivo è quello di classificare ogni pixel dell'immagine di input in differenti classi o categorie di output. Quindi ogni pixel avrà un suo valore di uscita che ci dirà se quel pixel appartiene ad una delle categorie identificate.

In generale diciamo che la segmentazione semantica è un task di computer vision che divide una immagine di input in segmenti multipli, dove ciascun segmento corrisponde ad una classe.

L'obiettivo è quello di assegnare ad una label (classi) ogni singolo pixel dell'immagine.

Ciò che otteniamo è una mappa pixel-wise (in termini di pixel) di classificazione.



In altre parole la segmentazione semantica va oltre la object detection tradizionale che tipicamente fornisce dei boundig-box (rettangoli) attorno agli oggetti. Al contrario di questo tipo di segmentazione la segmentazione semantica etichetta in maniera precisa ogni pixel associandolo ad un'oggetto o classe di appartenenza, andando ad evidenziare bordi e confini andando anche a catturare dettagli più fini rispetto al bounding-box normale.

In output otteniamo un'immagine colorata dove ciascun pixel (in base al suo valore) è colorato in base all'oggetto (classe) di appartenenza.
Ad es se un pixel è stabilito che appartiene alla classe 'persona' sarà colorato di rosso.

Caratteristiche chiave della segmentazione semantica:

- Pixel-wise: ogni pixel dell'immagine viene classificato ciò produce una mappa di segmentazione ad alta

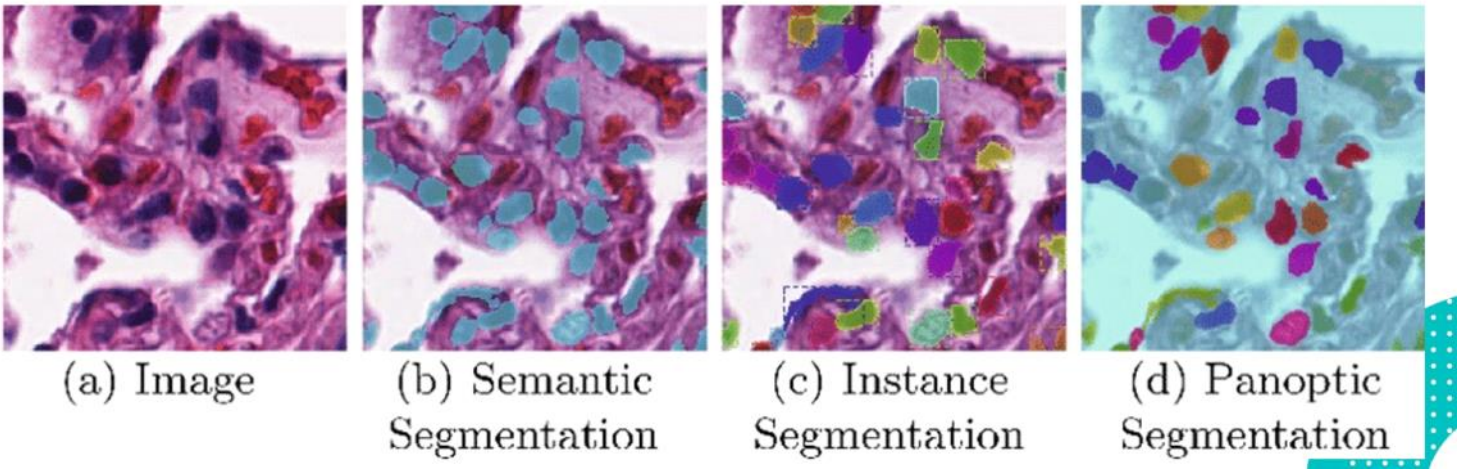
risoluzione.
- Boundary Preservation: la segmentazione semantica deve evidenziare i bordi degli oggetti per assicurare che i pixel siano correttamente classificati.
- Class-Agnostic: un-algoritmo di segmentazione semantica non ha conoscenza a priori del numero e dei tipi di oggetto presenti all'interno di un'immagine. Tale algoritmo deve classificare gli oggetti a prescindere che le classi siano tante o poche.

DIVERSI TIPI DI SEGMENTAZIONE:

Ci sono altre tecniche di segmentazione di immagine in base allo scopo.

INSTANCE SEGMENTATION: non solo classifica ogni pixel dell'immagine ma va anche a differenziare le istanze individuali (gli oggetti della stessa classe). Abbiamo una label unica che appartiene ad istanze distinte di una particolare classe. Molto utilizzata per compiti dove abbiamo bisogno di riconoscere una locazione molto specifica degli oggetti e dove dobbiamo contare gli oggetti. Ad es riconoscimento delle persone

PANOPTIC SEGMENTATION: Combina la segmentazione semantica e la segmentazione ad istanza con l'obiettivo di fornire una rappresentazione unificata che includa le regioni non object (come lo sfondo) e classi che ci diano informazioni sulle istanze degli oggetti.
Quindi tutti i pixel sono classificati ed in più i singoli oggetti sono ulteriormente differenziati.



U-NET OPERAZIONI:

- Convoluzioni 3x3 con funzioni di attivazione ReLu
- Skip Connections (copy and crop)
- Downsampling con l'uso di MaxPooling 2x2
- Upsampling con i Transposed Convolutional Layer
- Convoluzioni 1x1

FASI

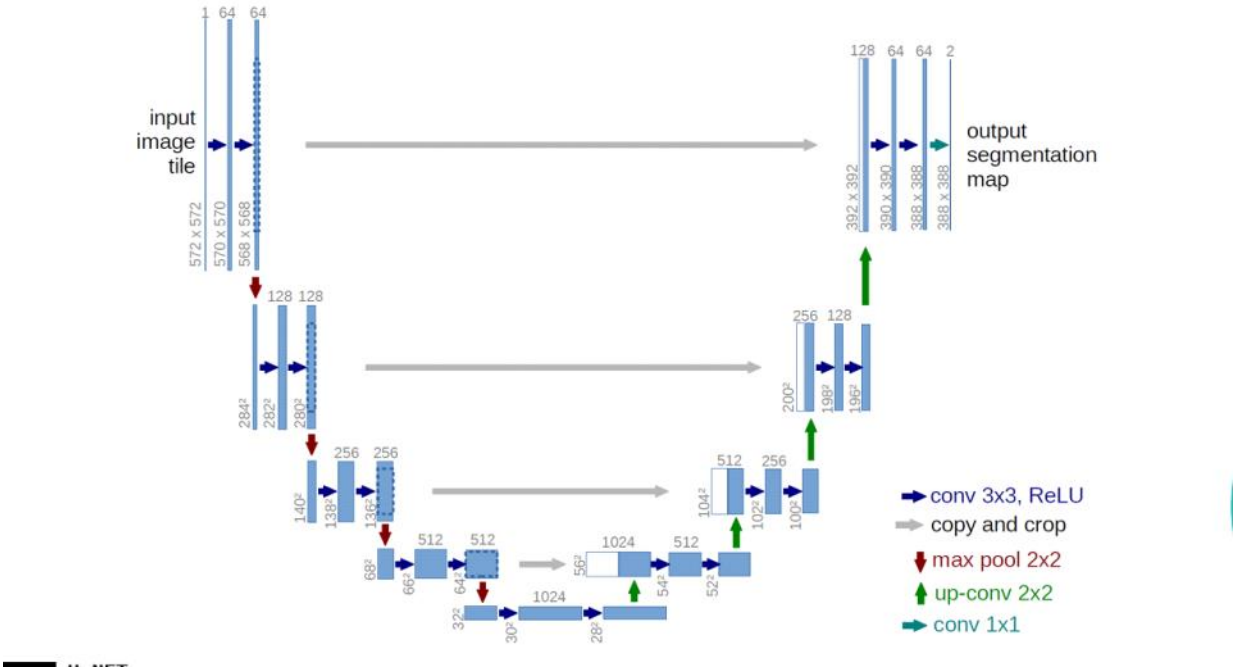
Nella U-Net abbiamo due fasi: la fase di contrazione e la fase di espansione.

FASE DI CONTRAZIONE:

Dove si susseguono convolutional layers tradizionali e livelli di MaxPooling che vanno a fare downsampling.
Si fa per catturare il contesto ma in maniera grossolana cioè sappiamo **cosa** c'è ma non sappiamo con precisione **dove** si trovano tali feature che abbiamo individuato.

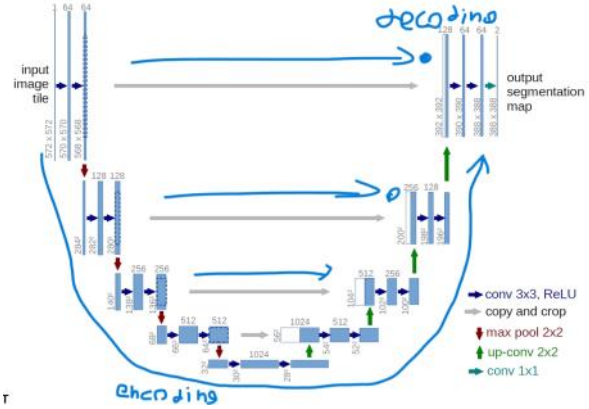
FASE DI ESPANSIONE:

Le informazioni sul **dove** si trovano le feature individuate nella fase di contrazione avviene in questa fase grazie alle skip connection.
Fase dove viene creta la segmentation map vera e propia.
Lavoriamo su feature map ripulite dalla fase di contrazione.



SKIP CONNECTIONS

Queste connessioni che implementano un blocco residuale permettono al decoder (dopo lo skip) di accedere a feature map ad alta risoluzione fornite dall'encoder (percorso normale) in corrispondenza di determinate dimensioni delle feature map e quindi in corrispondenza di determinate feature identificate all'interno del processo di encoding.



Le feature map dell'encoder sono concatenate con le feature map Upsample.

Andando a fare il merge di queste feature map anche a scale diverse, il decoder può accedere a dettagli localizzati ed ad anche informazioni più grossolane andando quindi ad effettuare una segmentazione molto più precisa rispetto a quella che farebbe utilizzando semplicemente le feature map ottenute durante il processo di encoding.

Quindi le Skip Connection aiutano alla U-Net a recuperare informazione spaziale più dettagliata che è essenziale per la creazione di segmentation map pixel-wise accurate.

Combinando il percorso di encoding e di decoding la U-Net ottiene informazioni molto efficaci per segmentare le immagini, utilizzando sia informazioni locali che globali.

LIVELLO DI ISCITA FINALE:

Tipicamente si usa una funzione di softmax per offrire un vettore di probabilità di appartenenza alle classi in maniera pixel-wise.

In generale l'output finale è la mappa di segmentazione che asseganno ai pixel una label di classe indicando quindi a quale categoria tale pixel appartiene.

In origine si utilizzava una convoluzione 1x1 che restituiva una segmentation map con 2 canali uno per il background e l'altro gli oggetti foreground.

Esercitazione U-Net

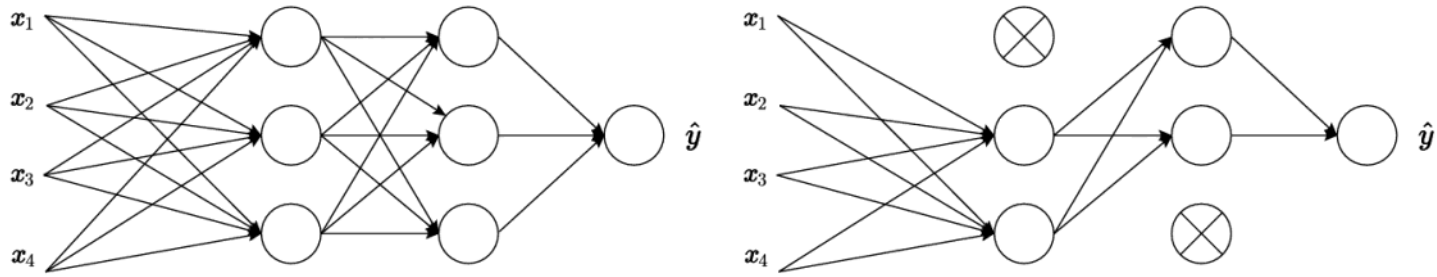
giovedì 23 maggio 2024 23:09

Vedi file jupyter notebook

Dropout

martedì 28 maggio 2024 10:22

È una tecnica di regolarizzazione quindi serve a ridurre l'overfitting della rete neurale.



Il dropout consoste nell'eliminazione (temporanea) in maniera randomica di alcuni neuroni della rete neurale.

Quindi l'addestramento verrà fatta sulla rete con alcuni neuroni mancanti.

I neuroni eliminati vengono poi riinseriti nella rete una volta terminato l'addestramento lungo tutti i batch ed una epoca.

Quando si inizia una nuova epoca si eliminano in maniera del tutto casuale altri neuroni e lungo i batch e la epoca la rete viene addestrata con questi nuovi neuroni mancanti che poi una volta terminata la epoca di addestramento vengono riinseriti nella rete.

Ciò significa che non effettuiamo più l'addestramento di un'unica rete neurale ma di molte architetture di rete che hanno lo stesso obiettivo.

Il dropout può essere applicato a tutti i livelli hidden della rete.

È valido sia per la fase di foreward che per la fase di backpropagation.

Lo scopo del dropout è quello di fare in modo che non ci sia interdipendenza tra neuroni di livelli diversi (un neurone droppato non passa il suo contributo ai neuroni del layer sccessivo, ma neanche riceve gli input del livello precedente). Questo fa si che il contributo del neurone droppato sia nullo, causando una maggiore distribuzione negli altri neuroni dei valori dei pesi (regolarizzazione).

Pensa al flusso d'acqua con tre sbocchi e poi con due.

FASE DI TRAINING:

Per ogni livello hidden, per ogni sample, per ogni iterazione, si ignora il nodo con probabilità P. Ciò significa che i nodi hanno una certa probabilità di essere droppati.

FASE DI TESTING:

Tutte le funzioni di attivazione vengono ridotte di un fattore P.

In questa fase non si utilizza il dropout poiché durante le predizioni in fase di testing perché otterremo dei risultati non desiderabili introducendo fattori casuali ottenendo delle predizioni con un rumore più accentuato.

VANTAGGI:

Avere dei parametri che sono meno interdipendenti dagli altri.

Si utilizzano delle subnet sempre diverse per l'addestramento che simulano modelli di rete diversi fra loro ma con lo stesso obiettivo.

Se H è il numero di unità nascoste, abbiamo 2^H modelli possibili in fase di training.

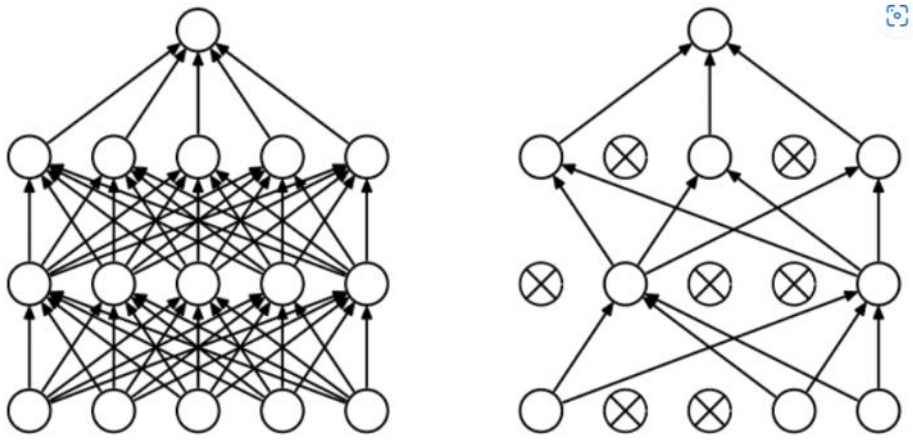
In fase di test invece sono considerate tutte le unità ma ridotte di un fattore P (che è la probabilità che una neurone non venga considerato in fase di training).

SVANTAGGIO:

Il numero di epoche necessarie per raggiungere dei risultati accettabili raddoppia.
Tuttavia la durata dell'addestramento della singola epoca diminuisce poiché ho meno parametri da addestrare.

SPIEGATO MEGLIO DA FONTE: <https://ichi.pro/it/dropout-un-modo-semplice-per-impedire-l-overfitting-delle-reti-neurali-265419173669522>

Le reti neurali profonde sono sistemi di apprendimento automatico molto potenti, ma sono soggetti a overfitting. Le reti neurali di grandi dimensioni addestrate su set di dati relativamente piccoli possono sovradattare i dati di addestramento. Questo perché il modello apprende il rumore statistico nei dati di addestramento, il che si traduce in scarse prestazioni quando il modello viene valutato su un set di dati di test. Il dropout è una tecnica per affrontare questo problema. L'idea chiave è rilasciare casualmente i nodi (insieme alle loro connessioni) dalla rete neurale durante l'addestramento. Ciò impedisce ai nodi di coadattarsi troppo.



(a) Standard Neural Net (b) After applying dropout.

Model Ensemble

Un'altra tecnica utilizzata per ridurre l'errore di generalizzazione consiste nel combinare diversi modelli diversi, che viene spesso chiamato come insieme di modelli. Questo ha senso perché, in un modello, possiamo avere errori in una parte dei dati del test, mentre un altro modello ha errori in un'altra parte dei dati del test.

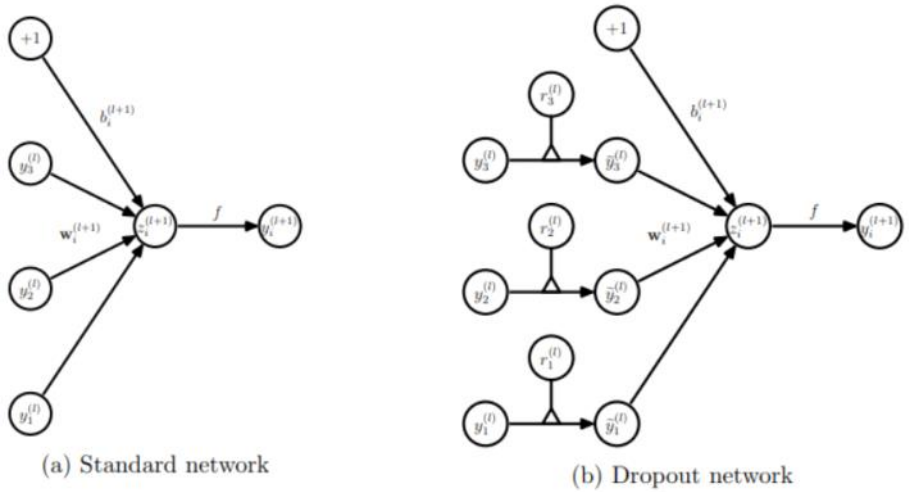
Pertanto, combinando diversi modelli possiamo ottenere un risultato più robusto poiché le parti che sono già corrette nella maggior parte dei modelli non cambieranno e l'errore sarà ridotto.

Tuttavia, l'addestramento di molti modelli diversi è difficile e l'addestramento di ciascuna rete di grandi dimensioni richiede molti calcoli. Inoltre, le reti di grandi dimensioni normalmente richiedono grandi quantità di dati di addestramento e potrebbero non essere disponibili dati sufficienti per addestrare reti diverse su diversi sottoinsiemi di dati.

Dropout incorpora entrambe queste tecniche. Previene l'overfitting e fornisce un modo per combinare approssimativamente in modo esponenziale molti diversi modelli di rete neurale in modo efficiente.

Fase di formazione

L'intuizione per la fase di drop out training è abbastanza semplice. Disattiviamo alcuni nodi della rete neurale al momento dell'addestramento per rendere l'architettura di rete diversa ad ogni iterazione (batch compresi) dell'addestramento. Il modo in cui disattiviamo i nodi può essere visualizzato nella figura seguente. Moltiplichiamo ogni input *y (l) per* un nodo *r (l)* che è una distribuzione a due punti che restituisce 0 o 1 con distribuzione di Bernoulli.



(a) Standard network (b) Dropout network

Qui, *r (l)* è un vettore di variabili casuali di Bernoulli, ciascuna delle quali ha una probabilità (tasso di abbandono) *p* di essere 1. Questo vettore viene campionato e moltiplicato per elemento con gli output di quello strato *y (l)* per creare il uscite assottigliate *trattino y (l).*

$$
\begin{aligned}
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \mathbf{y}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}),
\end{aligned}
$$

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p), \\
\tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \\
z_i^{(l+1)} &= \mathbf{w}_i^{(l+1)} \tilde{\mathbf{y}}^l + b_i^{(l+1)}, \\
y_i^{(l+1)} &= f(z_i^{(l+1)}).
\end{aligned}
$$

Qui nella seconda riga, possiamo vedere che aggiungiamo un neurone *r* che mantiene il nodo moltiplicando l'input per 1 con probabilità *p* oppure rilascia il nodo moltiplicando l'input per 0 con probabilità 1- *p* , quindi fai lo stesso passaggio in avanti come senza abbandono.

Viene introdotto un nuovo iperparametro *(tasso di abbandono) p* che specifica la probabilità con cui gli output del livello vengono esclusi o, inversamente, la probabilità con cui vengono mantenuti gli output del livello.

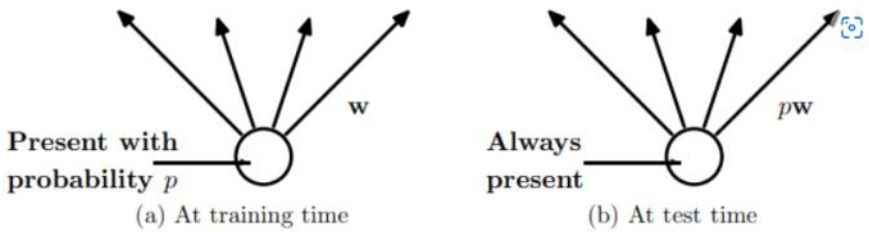
Analisi mia: se per esempio l'input è un valore di intensità di un pixel (su scala di grigi) ad esempio con valore 65 , questo verrà moltiplicato con 0 (con una probabilità di 1-p) oppure con 1 (con probabilità p), il risultato di tale moltiplicazione sarà moltiplicato con il rispettivo peso e poi dato ingresso alla funzione di attivazione assieme a tutti gli altri input (anche essi moltiplicati con 0 o 1 precedentemente e con il rispettivo peso poi) più il bias.

Fase di test

Ora, in fase di test, possiamo utilizzare il metodo della media come insieme di modelli, ovvero eseguire i dati di test in tutte le possibili reti neurali e fare la media dei risultati. Tuttavia, non è fattibile e computazionalmente costoso calcolare la media esplicita delle previsioni da molti modelli in modo esponenziale. **Quindi, il documento propone di ottenere la media implementando la rete neurale senza il dropout.**

Il dropout non viene utilizzato durante la previsione, in questo modo i pesi della rete aumentano del normale. Pertanto, prima di finalizzare la rete per il test, i pesi vengono prima ridimensionati in base al tasso di abbandono scelto. Questo metodo è chiamato ridimensionamento del peso.

Se un'unità viene mantenuta con probabilità p *durante l'addestramento, i pesi in uscita di quell'unità vengono moltiplicati per* p *al momento del test.*



(a) At training time (b) At test time

Attivazione del nodo basata sulla probabilità (p) al momento dell'addestramento e, ridimensionamento dei pesi della stessa probabilità quando il nodo è sempre presente al momento del test - Srivastava et al. (2014)

In pratica, il riscalaggio dei pesi può essere eseguito durante l'allenamento invece che durante il test. Questo è chiamato *dropout inverso* , in cui le uscite vengono ridimensionate dal tasso di dropout.

Non stiamo toccando tutti i risultati sperimentali mostrati nel documento, ma assicura che la tecnica del dropout migliora le prestazioni di generalizzazione su tutti i set di dati rispetto alle reti neurali che non hanno utilizzato il dropout.

Come vediamo nell'esercitazione sia con dropout che senza dropout abbiamo lo stesso numero di parametri addestrabili.

SENZA DROPOUT

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 512)               401920

 activation (Activation)     (None, 512)               0

 dense_1 (Dense)             (None, 512)               262656

 activation_1 (Activation)   (None, 512)               0

 dense_2 (Dense)             (None, 10)                5130

 activation_2 (Activation)   (None, 10)                0

=================================================================
Total params: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

CON DROPOUT

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_5 (Flatten)         (None, 784)               0

 dense_15 (Dense)            (None, 512)               401920

 dropout_8 (Dropout)         (None, 512)               0

 activation_15 (Activation)  (None, 512)               0

 dense_16 (Dense)            (None, 512)               262656

 dropout_9 (Dropout)         (None, 512)               0

 activation_16 (Activation)  (None, 512)               0

 dense_17 (Dense)            (None, 10)                5130

 activation_17 (Activation)  (None, 10)                0

=================================================================
Total params: 669706 (2.55 MB)
Trainable params: 669706 (2.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [14]:
```
# altre 10 epoche di addestramento per avere un numero
# di aggiornamenti dei parametri paragonabile al modello
# senza dropout. (il dropout impostato al 50% come in
# questo esempio non permette l'aggiornamento di metà
# delle celle ad ogni batch di ogni epoca)
history_dropout = model_dropout.fit(
    X_train,
    y_train,
    epochs=10,
    batch_size=32,
    validation_split=0.1,
    verbose = 1,
    shuffle=True
)
```

SPIEGAZIONE DAI PROF:

1. Nella test della prima epoca due pesi saranno effettivamente inizializzati con valori randomici e quindi la val_loss potrebbe essere più bassa. In casi come quello che hai citato prima però il numero di epoche viene aumentato tanto quanto risulta essere il dropout, in modo da far si che tutti i neuroni siano stati addestrati un certo numero di volte. Sempre nel caso che hai riportato, le epoche verrebbero raddoppiate (per fronteggiare il dropout del 50%)
   AinTziLLo — Oggi alle 22:59
   Ciao @AnOtherPeppe e @Felipe Riva tutto corretto, però le celle da disattivare con il dropout sono selezionate casualmente AD OGNI BATCH e non ad ogni epoca. Normalmente in un'epoca abbiamo decine, se non centinaia o migliaia o più ancora batch... quindi anche dopo solo un'epoca, un po' tutti i parametri del modello sono stati ottimizzati più e più volte

Esercitazione Dropout

martedì 28 maggio 2024 12:07

Vedi file jupyter

In [14]:
```
# altre 10 epoche di addestramento per avere un numero
# di aggiornamenti dei parametri paragonabile al modello
# senza dropout. (il dropout impostato al 50% come in
# questo esempio non permette l'aggiornamento di metà
# delle celle ad ogni batch di ogni epoca)
history_dropout = model_dropout.fit(
    X_train,
    y_train,
    epochs=10,
    batch_size=32,
    validation_split=0.1,
    verbose = 1,
    shuffle=True
)
```

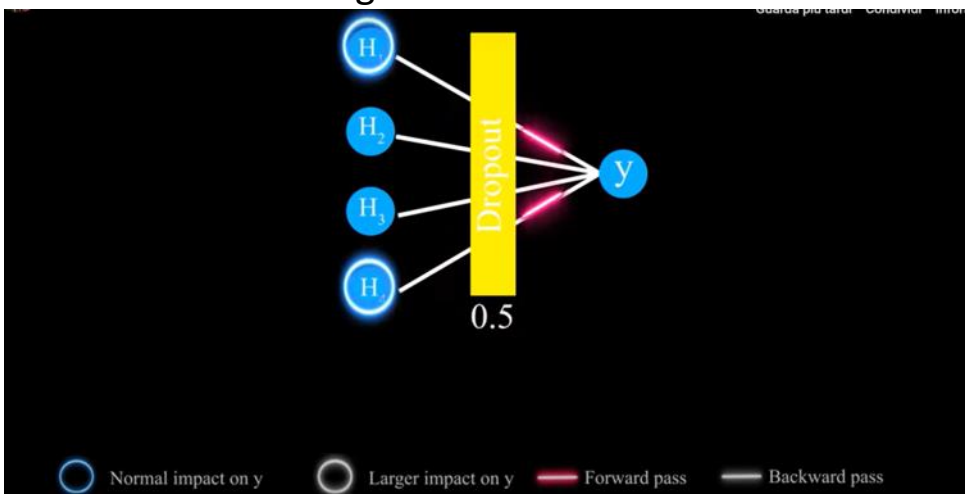
Richiesta spiegazione dropout

mercoledì 29 maggio 2024 20:49

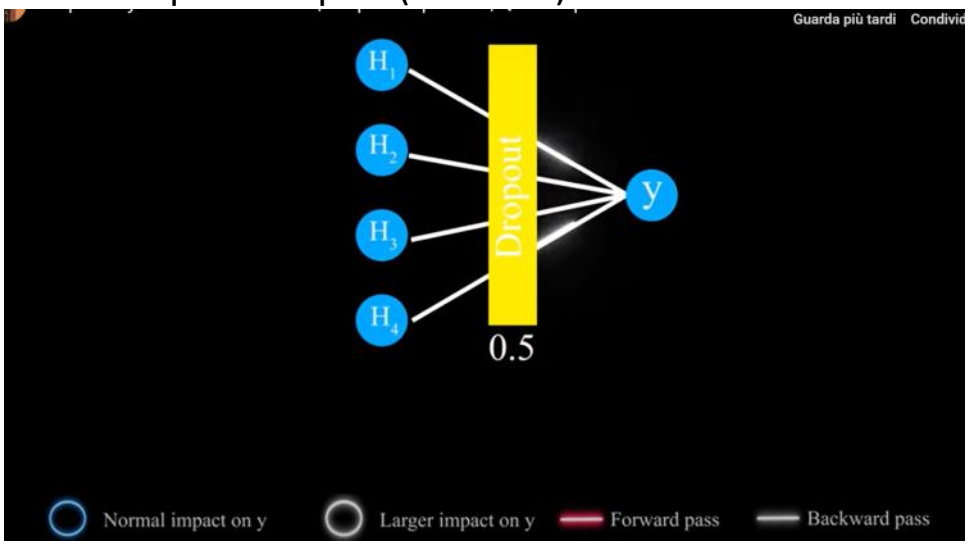
Ciao a tutti, volevo chiedervi aiuto sulla fase di test nell'utilizzo del Dropout.

Non ho capito bene come vengono considerati i pesi nella fase di test nella regolarizzazione di Dropout.

Nella fase di training vengono passati solo alcuni input moltiplicati per i rispettivi pesi (w1 e w4) poiché i due neuroni centrali sono stati disattivati come in figura:



Dopodiché nella fase di backpropagation vengono addestrati soltanto quei due pesi (w1 e w4):



In fase di test, tuttavia, vengono considerati tutti i neuroni e per condensare tutte le possibili architetture di rete possibili (con alcuni neuroni disattivati) si passano tutti i pesi ma moltiplicati per la

probabilità P che poi si moltiplicano a loro volta con i rispettivi valori di output da un neurone come in figura:

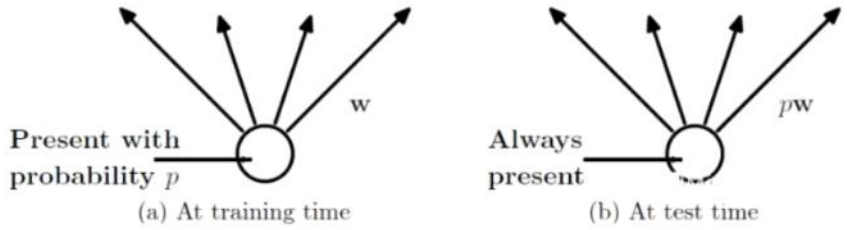


Figure 5: Dropout Operation during Model Prediction

Dropout formula during the testing phase:

$$w_{test}^{(l)} = pW^{(l)}$$

Ora, quello che non ho capito è: siccome nella fase di training sono stati addestrati solo due pesi (w1 e w4), gli altri due pesi (w2 e w3) che entità hanno? Sono rimasti con lo stesso valore di quando sono stati inizializzati cioè con valori random?
E poi cosa succede se vengono addestrati (w2 e w3) e non (w1 e w4) in un'altra architettura di rete con i neuroni H1 e H4 disattivati?

SPIEGAZIONE DAI PROF:

1. Nella test della prima epoca due pesi saranno effettivamente inizializzati con valori randomici e quindi la val_loss potrebbe essere più bassa. In casi come quello che hai citato prima però il numero di epoche viene aumentato tanto quanto risulta essere il dropout, in modo da far si che tutti i neuroni siano stati addestrati un certo numero di volte. Sempre nel caso che hai riportato, le epoche verrebbero raddoppiate (per fronteggiare il dropout del 50%)
   AinTziLLo — Oggi alle 22:59
   Ciao @AnOtherPeppe e @Felipe Riva tutto corretto, però le celle da disattivare con il dropout sono selezionate casualmente AD OGNI BATCH e non ad ogni epoca. Normalmente in un'epoca abbiamo decine, se non centinaia o migliaia o più ancora batch... quindi anche dopo solo un'epoca, un po' tutti i parametri del modello sono stati ottimizzati più e più volte

Batch Normalization

sabato 1 giugno 2024 20:33

È una tecnica molto utilizzata nelle reti neurali deep dimostratasi molto efficace nel migliorare la velocità dell'addestramento e nel stabilizzare l'addestramento nella capacità di generalizzazione delle reti.

L'idea principale è quella Normalizzare i risultati delle funzioni di attivazione di ogni livello in una rete neurale facendo appunto una normalizzazione su questi risultati delle funzioni di attivazione in modo che essi abbiano media 0 e varianza 1.

Questo processo di normalizzazione è fatto per ogni mini-batch durante l'addestramento.

Il processo di normalizzazione è inoltre applicato ad ogni layer in maniera indipendente rendendo poi un po' più semplice per i livelli successivi l'addestramento ed anche l'imparare ed assicurare l'ottimizzazione più dolce nei livelli successivi durante la fase di training.

La Batch Normalization è semplicemente un altro livello inserito tra due layer adiacenti, il lavoro della batch normalization è quello di prendere l'output di livello L normalizzarlo e passarlo al livello L+1.

In alcune implementazioni non andiamo a prendere l'output delle funzioni di attivazioni ma lo Z cioè quello che andrà poi in ingresso alla funzione di attivazione, ovvero l'applicazione dei pesi più il Bias.

$$z^{[i]} = W^{[i]} * a^{[i-1]} + b^{[i]}$$

Ricordiamo che z^i cioè z di un i-esimo livello è uguale alla moltiplicazione dei pesi per l'uscita del livello i-1 più il bias dell'i-esimo livello.

Inoltre, ricordiamo che l'output di un livello in una rete neurale è dato dall'applicazione della funzione di attivazione g a z.

$$a^{[i]} = g(z^{[i]})$$

Nella seguente immagine la Batch Normalization viene rappresentata in blu.

Tale parte in blu (la Batch Normalization) da in output i suoi risultati ovvero il batch di addestramento normalizzati al layer hidden successivo.

PASSI PER LA BATCH NORMALIZATION

Viene effettuata feature per feature sulle funzioni di attivazione in uscita da un determinato livello di una rete neurale.

Passi:

1. Calcolare media e varianza del mini-batch: durante l'addestramento per ogni mini-batch di dati vengono calcolati varianza e media dei risultati delle funzioni di attivazione. Tali statistiche sono calcolate in maniera indipendente per ciascuna feature considerando considerando comunque tutti gli esempi che fanno parte della mini-batch.
2. Applicare la Normalizzazione al Batch: Una volta che abbiamo calcolato media e varianza le attivazioni vengono normalizzate all'interno di tutto il mini-batch ed andiamo a sottrarre la media del batch andiamo a dividere per la radice quadrata della varianza per ogni feature.
3. Scale and Shift (apprendimento): I parametri che vengono introdotti sono due parametri per ciascuna feature (Scale and Shift introdotti per ogni feature) e dopo la normalizzazione le attivazioni sono scalate, grazie al parametro di Scale, e di shiftate di un parametrio Shift

BATCH NORMALIZATION DAL PUNTO DI VISTA MATEMATICO

- FASE DI TRAINING
    - Dato un mini-batch di risultati di funzione di attivazione per una feature particolare, in cui abbiamo n come dimensione della mini-batch andiamo a calcolare la media e la varianza del mini-batch come segue:

$$\mu = \frac{1}{n}\sum_i A^i \quad \sigma = \frac{1}{n}\sum_i (A^i - \mu)$$

media: si fa la somma di tutti i valori delle feature e poi si divide per la dimensione della mini-batch.
Varianza: per ogni featurte del nostro batch si calcola quanto differrisce dalla media e poi si sommano tutte queste differenze e si divide per la dimensione del mini-batch.

- Normalizziamo il vettore: dopo aver calcolato la media e la varianza del mini-batch si va a normalizzare il vettore dove si prendono i valori delle feature meno la media fratto la radice quadrata del quadrato della varianza meno epsilon dove epsilon è una piccola costante numerica che viene introdotta per evitare possibili divisioni per zero

$$A^{(i)}_{norm} = \frac{A^i - \mu}{\sqrt{\sigma^2 - \epsilon}}$$

- SCALE and SHIFT: durante la fase di Scale and Shift andiamo ad applicare i due parametri addestrabili gamma e beta applicando una trasformazione lineare di questi parametri. Questi parametri (addestrabili) sono aggiornabili durante la fase di back-propagation dove viene calcolato il gradiente e poi, tali parametri, vengono aggiornati iterazione per iterazione. L'idea è quella di permettere alla rete di scalare in maniera adattiva e shiftare i risultati delle funzioni di attivazione normalizzati basandosi sul task specifico e sui dati di addestramento specifici

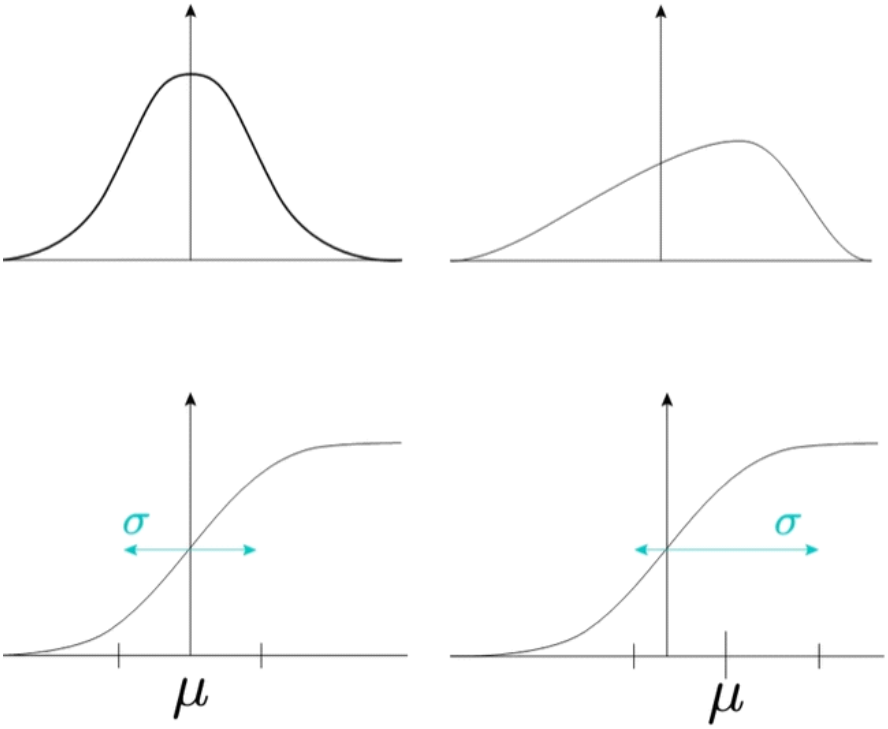
SCALE & SHIFT

Tramite l'operazione di Shift (apprendimento del valore beta) andiamo a spostare i valori in uscita di un layer SU un valore medio che è diverso da quello calcolato.

Tramite l'operazione di Scale (apprendimento del valore di gamma) andiamo a scalare ad un valore di varianza diverso rispetto al valore presente nel batch prima della normalizzazione.

Gamma e Beta non sono iperparametri ma sono parametri

addestrabili che vengono aggiornati durante la back-propagation.



FASE DI EVALUATION

Durante la fase di training vengono anche calcolate le medie mobili della media e della variazione standard che poi verranno utilizzate nella fase di evaluation cioè nella fase di test e nella fase di deploy per garantire stabilità alla rete durante queste fasi appunto.

FASE DI TEST

In fase di deploy non avremo dei batch ma dei singoli sample/ valori sui cui andremo ad effettuare le predizioni.

Vengono usati come media e deviazione standard quelle mobili calcolate in fase di addestramento.

Queste poi ci permettono poi di andare ad effettuare le operazioni di normalizzazione in fase di deploy, e poi lo scale and shift, sempre in fase di deploy, in maniera che le funzioni su cui andiamo a fare le normalizzazioni risentano anche della media mobile calcolata durante la fase di addestramento, e quindi in un certo modo ci diano un indice di come i valori su cui sta operando il modello in fase di deploy si discostano dalla fase di addestramento.

BATCH NORMALIZATION (INTUITION)

L'intuizione che sta alla base della batch normalization è quello di andare a mitigare fenomeni tipici della fase di addestramento delle reti neurali come:

- Vanishing ed Exploding gradient: ricordiamo che questo fenomeno si ha quando i gradienti sono troppo piccoli e quindi rallentano o addirittura bloccano il processo di addestramento. Con l'exploding gradient invece si ha quando abbiamo dei gradienti troppo grandi che vanno a rendere instabile l'addestramento delle reti neurali.
- Internal Covariate Shift: ovvero, durante l'addestramento delle reti neurali la distribuzione dell'input su ogni layer cambia non appena cambiano i parametri dei livelli precedenti. In altre parole, se cambiano i parametri dei livelli precedenti cambia la distribuzione degli input di un determinato livello.
  Questo fenomeno (Internal Covariate Shift) rallenta l'addestramento di una rete neurale dal momento che la rete ha bisogno di adattarsi continuamente ai cambi di distribuzione degli input, cerca di reagire a questi cambi di distribuzione degli input. Il fatto che andiamo a normalizzare l'input significa che andiamo a ridurre questo Internal Covariate Shift andando a normalizzare le attivazioni di ogni livello prima di darle in pasto ad un livello successivo. Questo garantisce che le attivazioni abbiano durante l'addestramento una distribuzione più consistente rispetto alle reti che non fanno uso di questo tipo di accorgimento.

Mantenendo un Batch Normalizzato e delle funzioni di attivazione normalizzate con media 0 e varianza 1 la Batch Normalization va a ridurre la probabilità che i gradienti siano o troppo piccoli o troppo grandi oltre che mitigare l'Internal Covariate Shift.

Effetti del Batch Normalization:

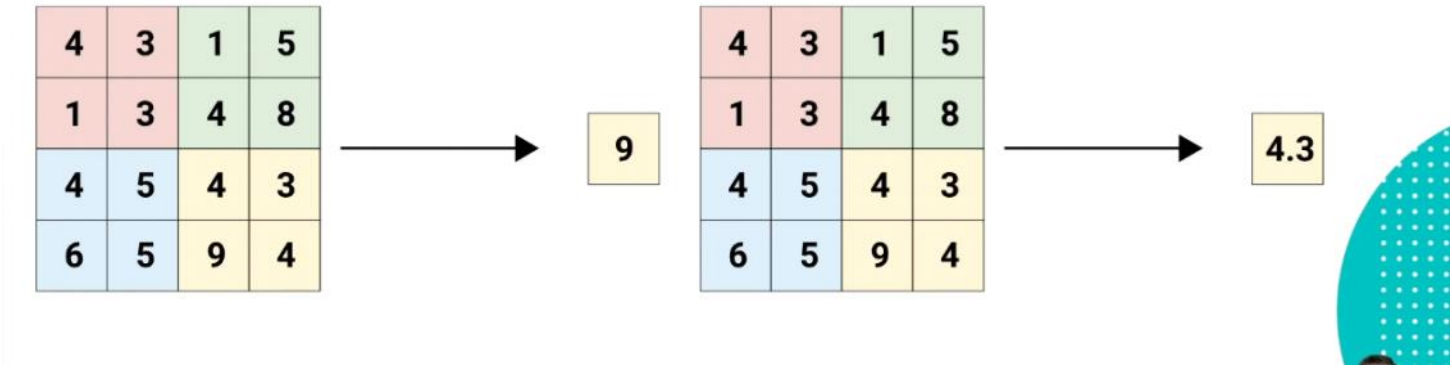
- Regolarizzazione: si introduce un po' di rumore dovuto alla variabilità tra le distribuzioni dei mini-batch otteniamo un'effetto di regolarizzazione che stabilizza l'addestramento delle reti.
- Accelerazione dell'addestramento

Con la batch normalization i livelli sono più indipendenti gli uni dagli altri

Global Pooling

mercoledì 23 ottobre 2024 17:51

I livelli di pooling Global fanno le stesse operazioni dei livelli di pooling già visti, solo che li effettuano su tutto l'input.

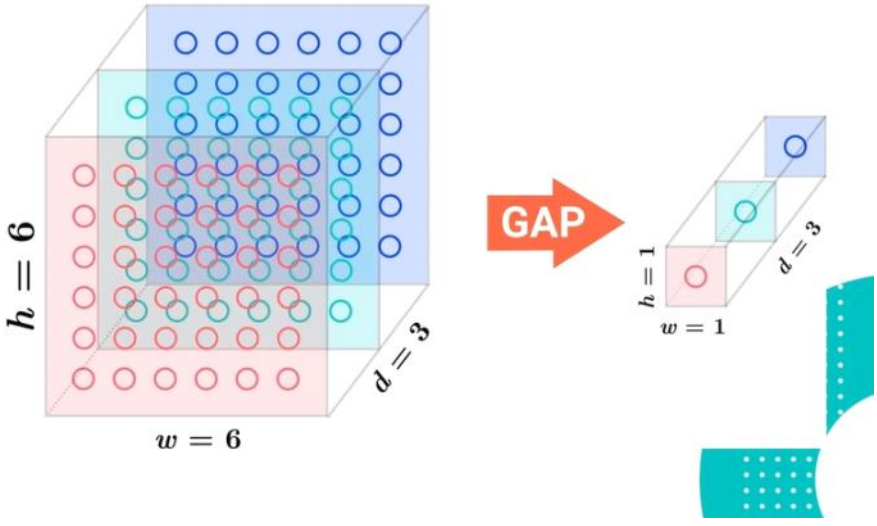


È una tecnica che viene utilizzata per ridurre le dimensioni spaziali delle feature map a una dimensione fissa.

Di solito i livelli di global pooling vengono utilizzati alla fine delle rete neurali presubilmente convolutive al posto dei livelli fully-connected.

Il global pooling fa corrispondere un valore numerico ad ogni feature map.

Si effettua un downsampling a 1x1xD (D=profondità dell'input)



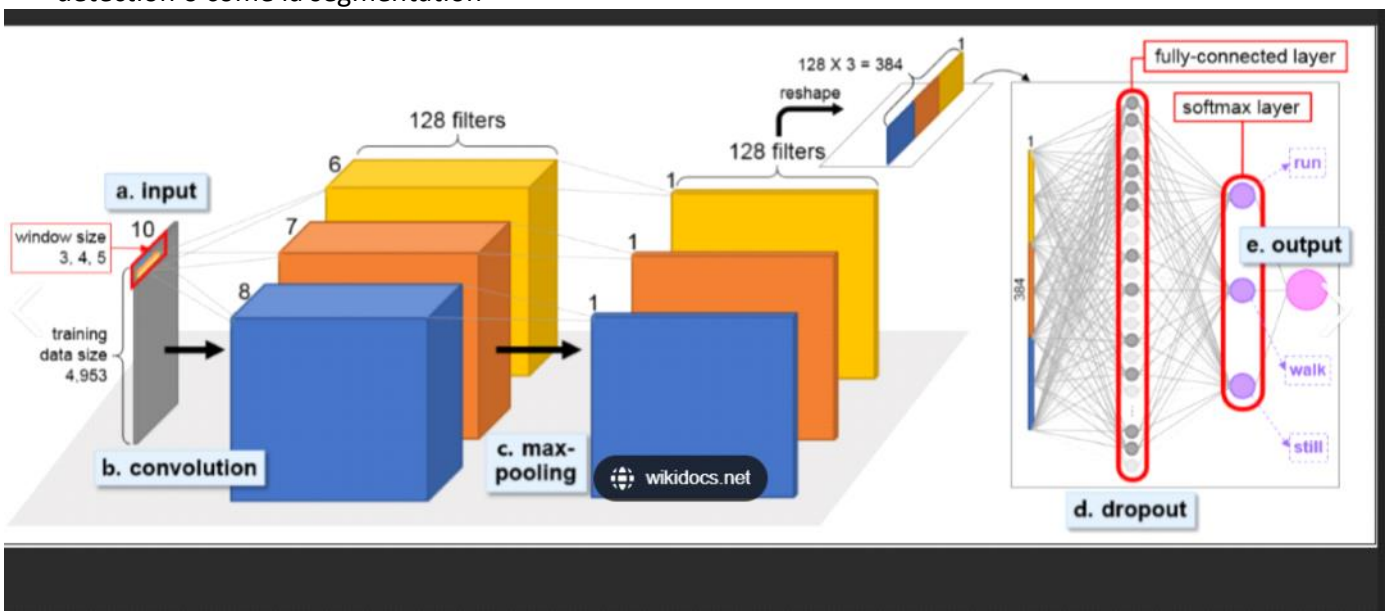
Uno dei classici utilizzi del global pooling è quello si sostituire i layer "dense" negli ultimi layer di una rete convolutiva in modo da rendere la mappa delle feature estratte come una mappa di confidenza delle categorie ricercate nell'input sottoposto alla rete.

Contrariamente a quello che accade nei layer delle reti tradizionali in cui abbiamo che dopo i vari livelli convolutivi e di pooling abbiamo una serie di livelli fully-connected che si occupano di fare la

classificazione vera e propria, che ci restituiscono il risultato finale.

Tuttavia questo approccio presenta alcune problematiche:

1. Alta dimensionalità: andare ad appiattire le feature map in un unico array di numeri molto grande per poi darlo in pasto ad uno o più livelly fully-connected comporta un costo abbastanza elevato in termini di complessità computazionale. È come se andassimo a creare una rete neurale tradizionale su un input che è già abbastanza grande di per se.
2. Perdità dell'informazione spaziale: I livelli fully connected non hanno nessun modo per mantenre le informazioni spaziali che invece potrebbero essere essenziali in task come l'object detection o come la segmentation

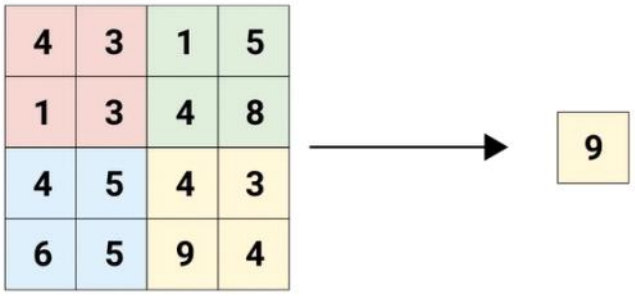


I layer di Global Pooling provano a risolvere questi problemi andando a performare un'operazione su tutta la dimensione spaziale delle feature map andando ad ottenere una rappresentazione globale di ogni canale presente all'interno del layer su cui applichiamo il global pooling.

I tipi più comuni di Pooling sono il Global Average Pooling che calcola il valore MEDIO su tutto il livello di input, ed il Global Max Pooling che calcola il valore MASSIMO su tutto il livello di input.

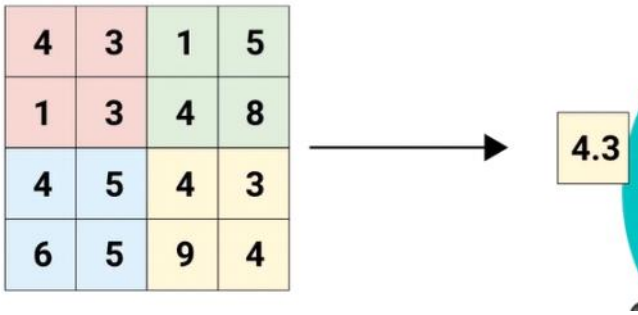
Global Max Pooling

Calcola il valore **massimo** su *tutto* il livello di input



Global Average Pooling

Calcola il valore **medio** su *tutto* il livello di input



Dunque si vanno a ridurre tutte le dimensioni spaziali ad una dimensione fissa e contemporaneamente andiamo a catturare le feature più salienti in ogni canale.

Un'altra caratteristica chiave del Global Pooling è la retention dell'informazione spaziale. Al contrario dell'operazione di Flattening e del dare in pasto il flattened array ai livelli fully-connected, con il Global Pooling andiamo a mantenere l'informazione spaziale all'interno di ogni canale, che ovviamente cambia a seconda del tipo di pooling globale che viene utilizzato, e questo è molto vantaggioso in task di Object Detection in cui abbiamo bisogno di una localizzazione molto precisa.

Un altro vantaggio del Global Pooling è L'INVARIANZA ALLA DIMENSIONE DELL'INPUT, dal momento che il global pooling produce sempre una rappresentazione fissa (1x1) questo semplifica il processo di sviluppo ma anche di deploy dei modelli e ci consente di utilizzare immagini o feature map in ingresso di varie dimensioni.

Il Global Pooling è diventato molto popolare in molte architetture di rete moderne come ad esempio nella Google Net o nelle reti Residuali che fanno un uso massiccio di questi livelli di Global Pooling.

È particolarmente utile in modelli che richiedono una rappresentazione globale ma allo stesso tempo informativa e compatta di tutto quello che viene trovato all'interno di una determinata immagine di Input mentre contemporaneamente si vuole ridurre il costo computazionale e contemporaneamente riducendo anche il rischio di Overfitting associati con i livelli fully-connected.

Esercitazione TensorFlow Datasets

domenica 19 gennaio 2025 17:32