Deep Learning

Università Roma Tre
Dipartimento di Ingegneria

Anno Accademico 2022 - 2023

Convolutional Neural Networks (CNN)
3a parte

Sommario

- Calcolo del numero dei parametri
- LeNet-5 e calcolo dei parametri
- Architettura AlexNet
- 1x1 convolution

Calcolo del numero di parametri di una rete neurale

- Il calcolo del numero di parametri è **fondamentale per comprendere la complessità** della rete e apportare miglioramenti all'architettura (es. introducendo pooling layer per ridurre i parametri).
- Il calcolo dipende dal tipo di layer che stiamo considerando e dai valori ricevuti dal layer precedente.
- Consideriamo il calcolo del numero di parametri per le seguenti configurazioni:
  - Un **Convolutional layer** seguito da un **FC layer** (**CONV→FC**)
  - Un **Input layer** seguito da un **Convolutional layer** (**I→FC**)
  - Un **FC layer** seguito da un **FC layer** (**FC→FC**)

Calcolo del numero dei parametri: **I→FC**

- I **parametri** consistono nell'insieme dei **pesi e bias** nel layer convoluzionale, che produrranno i valori delle attivazioni nelle feature maps.
  - Il layer di input non ha pesi associati.
- Se indichiamo con:
  - $\#\mathbf{W_c}$ e $\#\mathbf{B_c}$ il numero di pesi e bias del layer convoluzionale
  - $\mathbf{f}$ la dimensione del LRF
  - $\mathbf{N_c}$ numero dei filtri nel convolutional layer
  - $\mathbf{C}$ profondità delle istanze in input (es. 3 per immagine a colori RGB)
- allora si ha:

$$\#\mathbf{W_c} = \mathbf{f}^2 \times \mathbf{C} \times \mathbf{N_c} \quad \text{e} \quad \#\mathbf{B_c} = \mathbf{N_c}$$

- Lo stesso risultato si ottiene per configurazioni **CONV→CONV**, considerando come profondità $\mathbf{C}$ la profondità delle feature maps nel layer precedente.
- Si nota come il numero di parametri è indipendente dalla dimensione X,Y dell'input.

Calcolo del numero dei parametri: **CONV→FC**

- I **parametri** consistono nell'insieme dei **pesi e bias** del layer FC connesso alle feature maps prodotte dal convolutional layer precedente.
- Se indichiamo con:
  - $\#\mathbf{W_{cf}}$ e $\#\mathbf{B_{cf}}$ il numero di pesi e bias del layer FC
  - **O** dimensione delle feature maps nel convolutional layer, supponendo larghezza e altezza coincidenti.
  - $\mathbf{N_c}$ numero dei filtri nel convolutional layer
  - **F** numero dei nodi nel layer FC
- allora si ha:

$$\#\mathbf{W_{cf}} = \mathbf{O^2} \times \mathbf{N_c} \times \mathbf{F} \quad \text{e} \quad \#\mathbf{B_{cf}} = \mathbf{F}$$

- Spesso si opera una "**linearizzazione**" dell'output del convolutional layer. Se abbiamo $\mathbf{N_c}$ filtri e una dimensione delle feature maps pari a OxO, introduciamo una rappresentazione 1-dimensionale con un vettore di $\mathbf{O^2} \bullet \mathbf{N_c}$ elementi, passato in input al layer fully-connected.

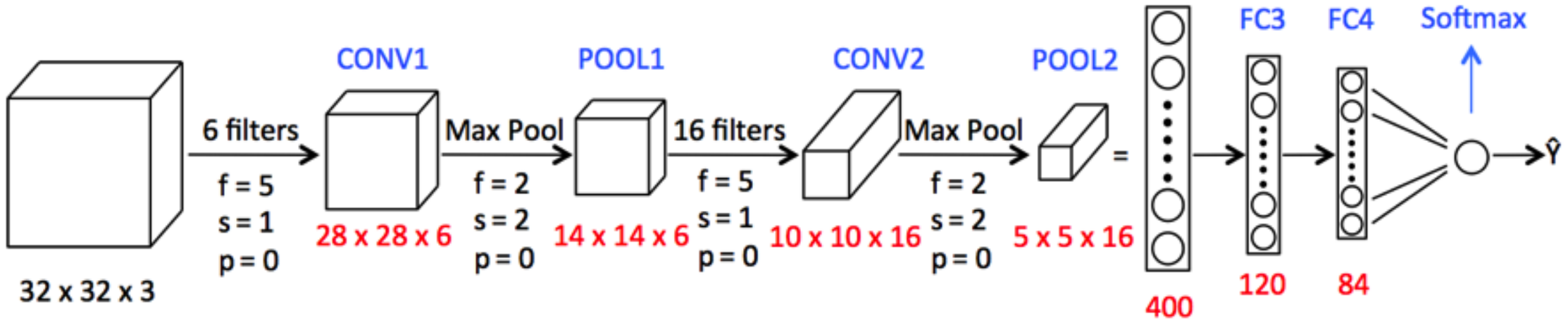
Calcolo del numero dei parametri: **FC→FC**

- I **parametri** consistono nell'insieme dei **pesi e bias** del layer FC connesso al FC precedente.
- Se indichiamo con:
  - $\#\mathbf{W_{ff}}$ e $\#\mathbf{B_{ff}}$ il numero pesi e bias del layer FC
  - **F** il numero di nodi nel layer FC
  - $\mathbf{F_{-1}}$ il numero di nodi nel layer FC precedente
- allora si ha:

$$\#\mathbf{W_{ff}} = \mathbf{F_{-1}} \times \mathbf{F} \quad \text{e} \quad \#\mathbf{B_{ff}} = \mathbf{F}$$

LeNet-5: numero di parametri

- Indichiamo con **f**, **s** e **p** rispettivamente la dimensione del filtro, stride e pooling (dove 0 corrisponde al pooling VALID).

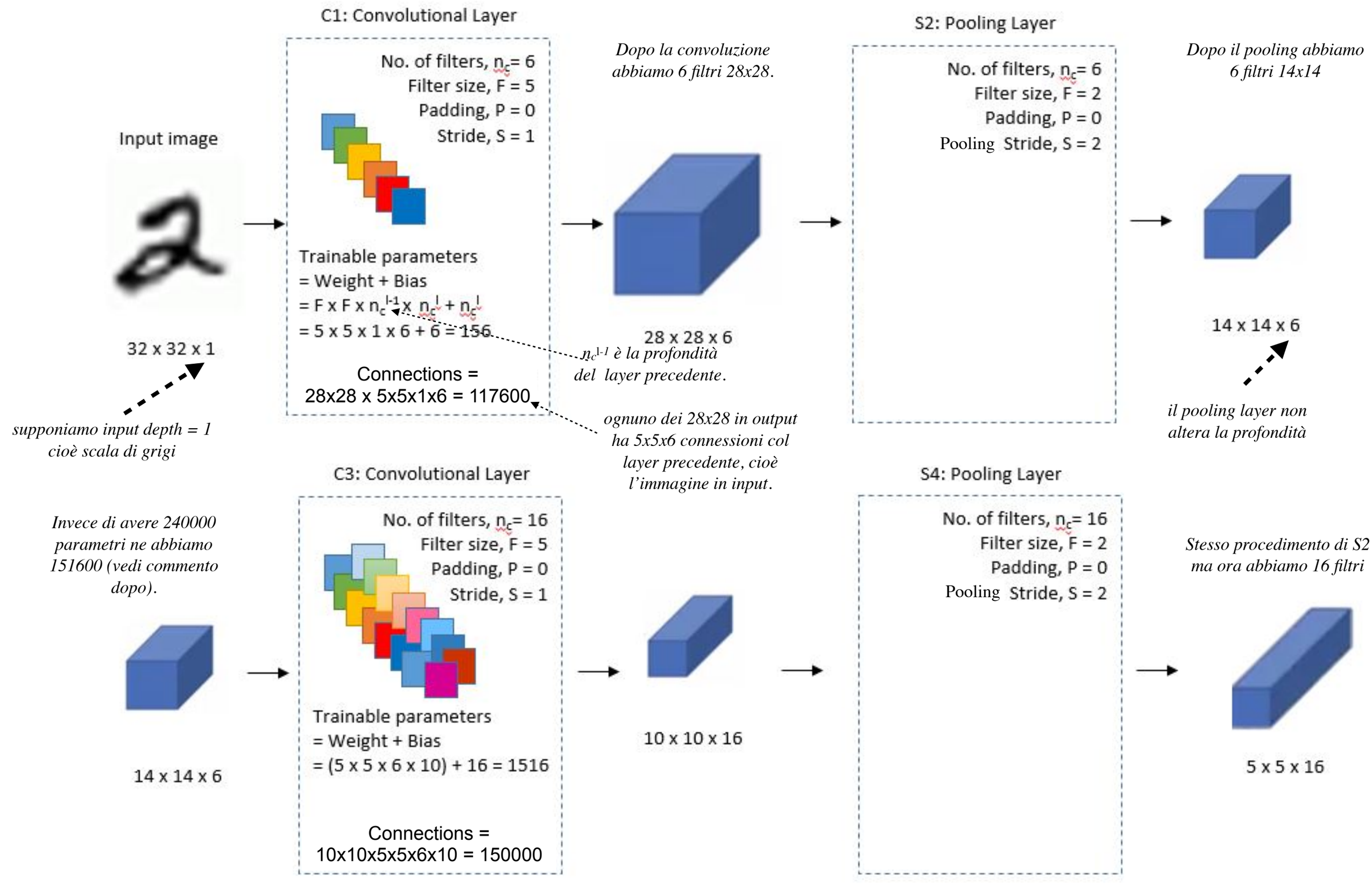


28x28x6
feature maps di 6 filtri
di dimensione 28x28 l'uno

14x14x6
*un pooling layer con **f** e **s** pari*
a 2 dimezza le dimensioni,
ma mantiene uguale la depth
della feature maps.

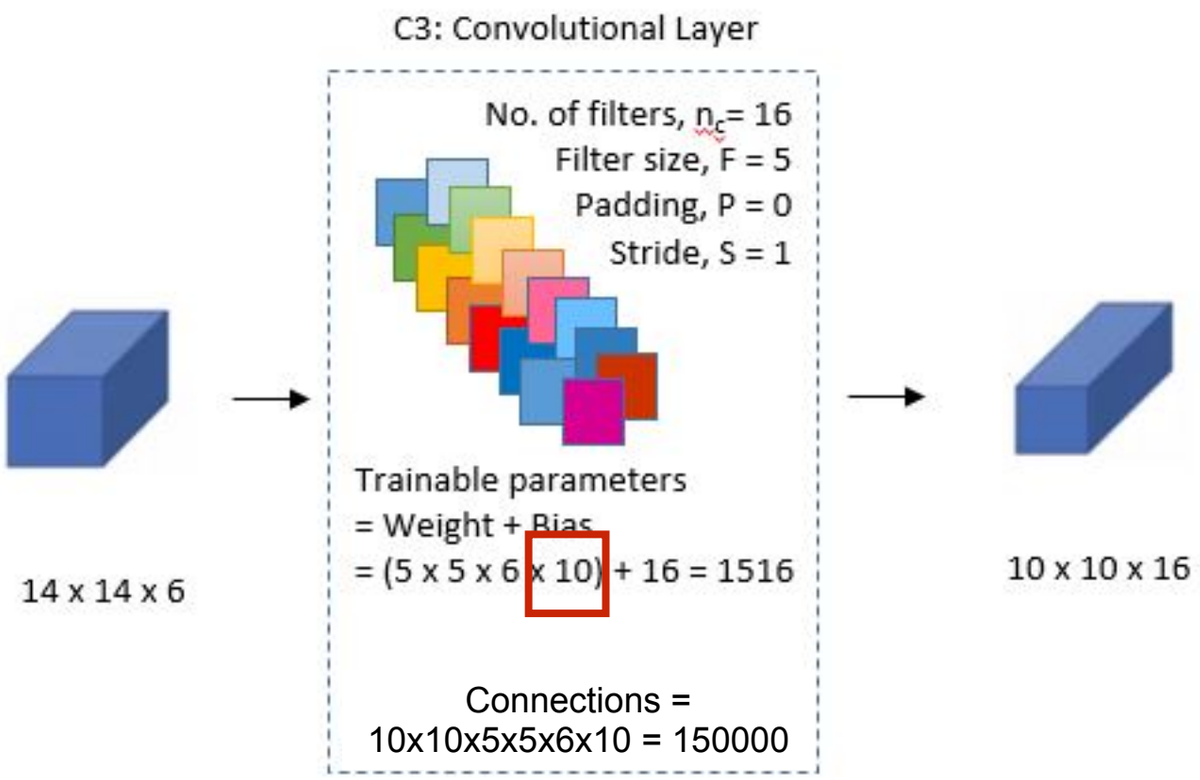
Non è un vero layer,
ma una linearizzazione dei
dati: rendiamo flat la
rappresentazione
5x5x16 -> 400

LeNet-5: numero di parametri (2)



LeNet-5: peculiarità

- Nel #3 hidden layer, con lo scopo di ridurre potenziali simmetrie e il numero di connessioni, gli autori hanno deciso che **solo 10 delle 16 features maps sono connesse con le 6 features maps del layer precedente**.
  - La tecnica **dropout** introdotta solo successivamente ha automatizzato questo step, perciò non si riscontra in architetture più recenti.



Schema di interconnessione tra feature maps impiegato.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I
EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

LeNet-5: numero di parametri (3)

Rendiamo "flat" l'output precedente. Abbiamo 400 (5x5x16) nodi dal layer S4. Il primo strato fully connected layer ha 120 nodi. Ogni nodo del layer è connesso con i 400 nodi dello strato precedente.

5 x 5 x 16

C5: Fully Connected Layer

120

Trainable parameters = Weight + Bias
= (400 x 120) + 120 = 48120

Fully connected layer con 84 neuroni.

F6: Fully Connected Layer

84

Trainable parameters = Weight + Bias
= (120 x 84) + 84 = 10164

softmax

Output

0
1
2
3
4
5
6
7
8
9

CNN - Esercizio

Se le tue GPU non hanno memoria sufficiente per una rete CNN, quali sono le 5 cose che puoi fare per risolvere il problema?

CNN - Esercizio

Se le tue GPU non hanno memoria sufficiente per una rete CNN, quali sono le 5 cose che puoi fare per risolvere il problema?

1. Ridurre la **dimensione del mini-batch**.

CNN - Esercizio

Se le tue GPU non hanno memoria sufficiente per una rete CNN, quali sono le 5 cose che puoi fare per risolvere il problema?

1. Ridurre la **dimensione del mini-batch**.

2. Ridurre la **dimensionalità nella rete**, ad esempio con stride > 1 in uno o più layers, o con pooling layers.

   - Anche un convolutional layer riduce la dimensione del layer precedente ma, a differena del pooling layer, introduce ulteriori parametri da apprendere.

CNN - Esercizio

Se le tue GPU non hanno memoria sufficiente per una rete CNN, quali sono le 5 cose che puoi fare per risolvere il problema?

1. Ridurre la **dimensione del mini-batch**.

2. Ridurre la **dimensionalità nella rete**, ad esempio con stride > 1 in uno o più layers, o con pooling layers.

   - Anche un convolutional layer riduce la dimensione del layer precedente ma, a differena del pooling layer, introduce ulteriori parametri da apprendere.

3. Cambiare l'architettura **rimuovendo un layer**.

CNN - Esercizio

Se le tue GPU non hanno memoria sufficiente per una rete CNN, quali sono le 5 cose che puoi fare per risolvere il problema?

1. Ridurre la **dimensione del mini-batch**.
2. Ridurre la **dimensionalità nella rete**, ad esempio con stride > 1 in uno o più layers, o con pooling layers.
   - Anche un convolutional layer riduce la dimensione del layer precedente ma, a differena del pooling layer, introduce ulteriori parametri da apprendere.
3. Cambiare l'architettura **rimuovendo un layer**.
4. Usare rappresentazioni **float a 16** bit invece che 32.

CNN - Esercizio

Se le tue GPU non hanno memoria sufficiente per una rete CNN, quali sono le 5 cose che puoi fare per risolvere il problema?

1. Ridurre la **dimensione del mini-batch**.
2. Ridurre la **dimensionalità nella rete**, ad esempio con stride > 1 in uno o più layers, o con pooling layers.
   - Anche un convolutional layer riduce la dimensione del layer precedente ma, a differena del pooling layer, introduce ulteriori parametri da apprendere.
3. Cambiare l'architettura **rimuovendo un layer**.
4. Usare rappresentazioni **float a 16** bit invece che 32.
5. **Distribuire la computazione** su più elaboratori.

Architettura CNN più recenti

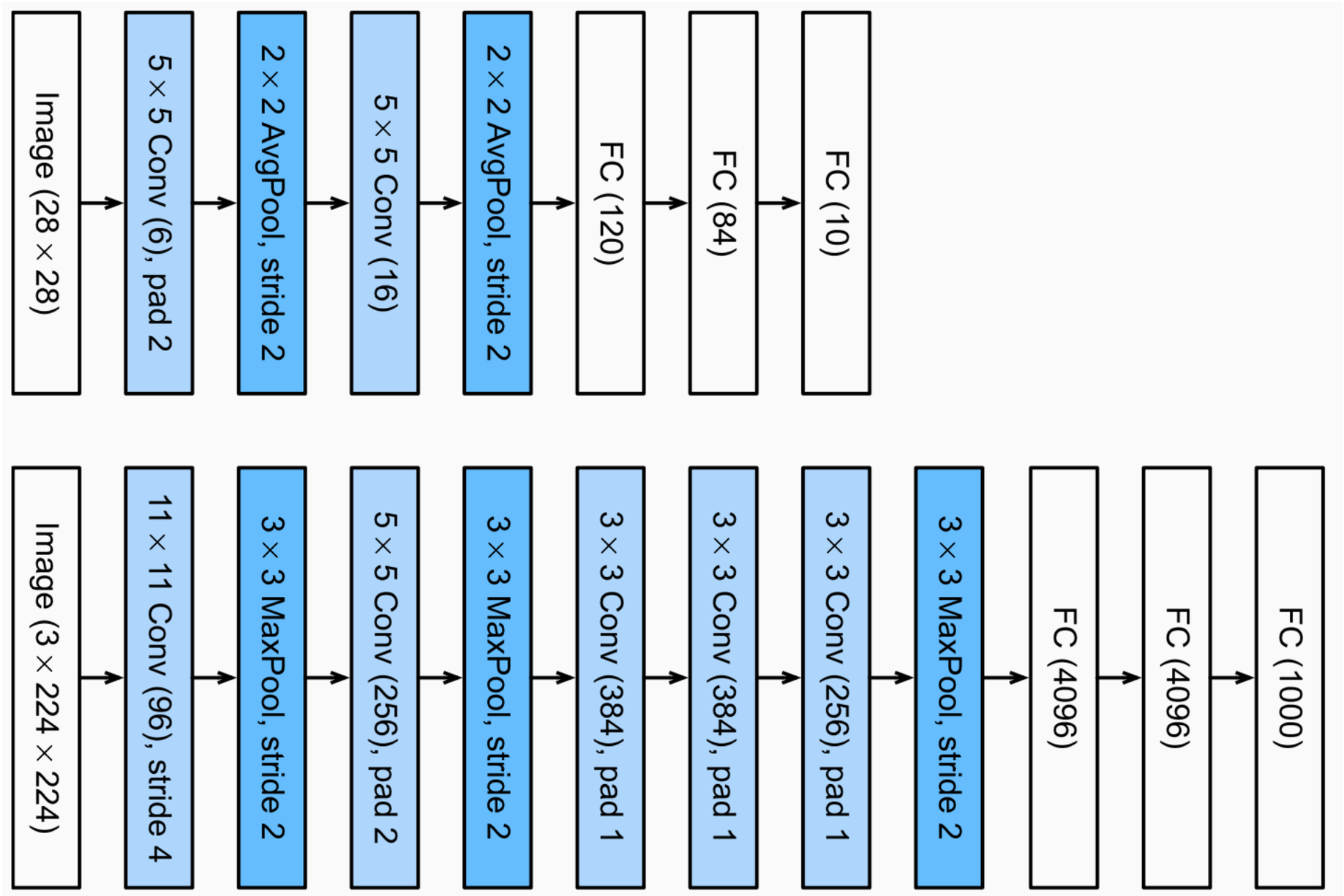
- Ne sono state proposte molte. Anche se sviluppate in un particolare task di computer vision, esse sono state impiegate in modo proficuo in altri domini, es. tracking, segmentation, object detection, style transformation.
- La challenge ImageNet (dal 2010) è favorito lo sviluppo di molte architetture.
- Le CNN sono relativamente semplici, ma creare una architettura efficiente richiede intuizione, una base algebrica, e molti tentativi.
  - Speso nuove architetture sfruttano elementi di architetture precedenti.

Architettura CNN più recenti #1

- Sebbene LeNet sia efficiente per il problema OCR, non si adatta facilmente a dataset più grandi ed eterogenei. Effettivamente dal 1995 (LeNet) al 2012 (AlexNet) sono state proposte tecniche ML alternative (es. kernels, ensemble, structured estimation) efficienti in molti tasks.
- Perché abbiamo atteso così a lungo per avere una rete più versatile e capace di competere con le altre architetture ML?
- Nel anni '90 una scheda GPU come la NVIDIA GeForce 256 era capace di 480 MFLOP, senza la disponibilità di framework software per semplificare la programmazione. Oggi la NVIDIA Ampere A100 raggiunge i 300 TFLOPS . Un dataset di cifre a bassa risoluzione (28x28) era considerato molto arduo da trattare.
  - In pratica, era molto complesso testare architetture GPU-based anche su dataset semplici.
- I **dati disponibili** adatti all'addestramento sono aumentati sensibilmente, e questo ha garantito la sperimentazione di un numero maggiore di architetture.
  - ImageNet è stato costruito mediante Google Image e per mezzo di Amazon Mechanical Turk per la classificazione manuale.

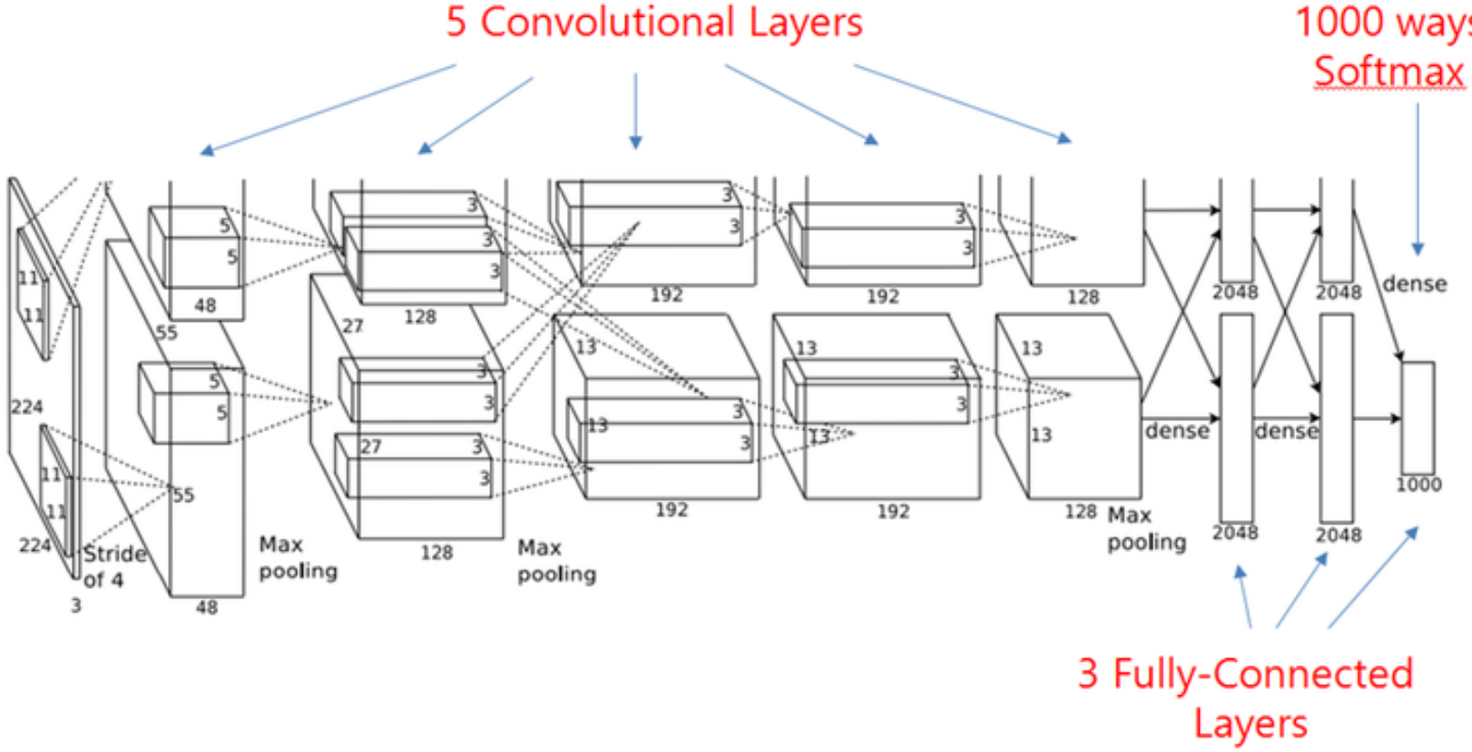
LeNet vs AlexNet

- AlexNet ha 8 layers: 5 convolutivi, 2 FC nascosti, 1 FC output.
- Usa la ReLU invece delle sigmoid o tanh.



Architettura AlexNet

- Architettura CNN vincitrice della challenge object detection ILSVRC 2012 con un top-5 error del 17% (il secondo ha ottenuto 26%) sviluppata da Alex Krizhevsky e Ilya Sutskever.
- Primo tentativo di sfruttare piattaforme hardware GPU-enabled per addestrare reti complesse.
- E' molto simile a **LeNet-5** ma con più profondità.



Dopo i 5 convolutional layers (11x11, 5x5 e 3x3) c'è il max pooling, e una rete FC da 3 layer. Impiega ReLI, SGD e momentum.

La doppia pipeline è dovuta all'hardware impiegato per l'addestramento (2 NVIDIA GTX 580s con 3Gb).

Architettura AlexNet (2)

- Le immagini di ImageNet sono 8x più grandi rispetto a MNIST.
- I LRF del primo strato sono 11x11, 5x5 nel secondo e 3x3 nel terzo.
- Dopo il primo, il secondo e 5o strato convolutivo, c'è un *max-pooling layers* con finestra 3x3 e uno stride pari a 2.
- AlexNet ha 10 volte i canali di LeNet.
- La rete FC multi-layer ha 1Gb di parametri. La doppia pipeline di elaborazione permetteva di suddividere l'occupazione.
  - Il numero elevato di parametri rende AlexNet poco efficiente rispetto ad architetture più recenti.
- La ReLU rende la computazione dei gradienti più rapida. Inoltre se l'inizializzazione dei parametri porta a valori di attivazione vicini ad 1 o 0 (estremi dell'intervallo) la derivata è vicina allo 0, e questo rallenta l'aggiornamento dei pesi. Il gradiente della ReLU è sempre 1 per valori positivi.

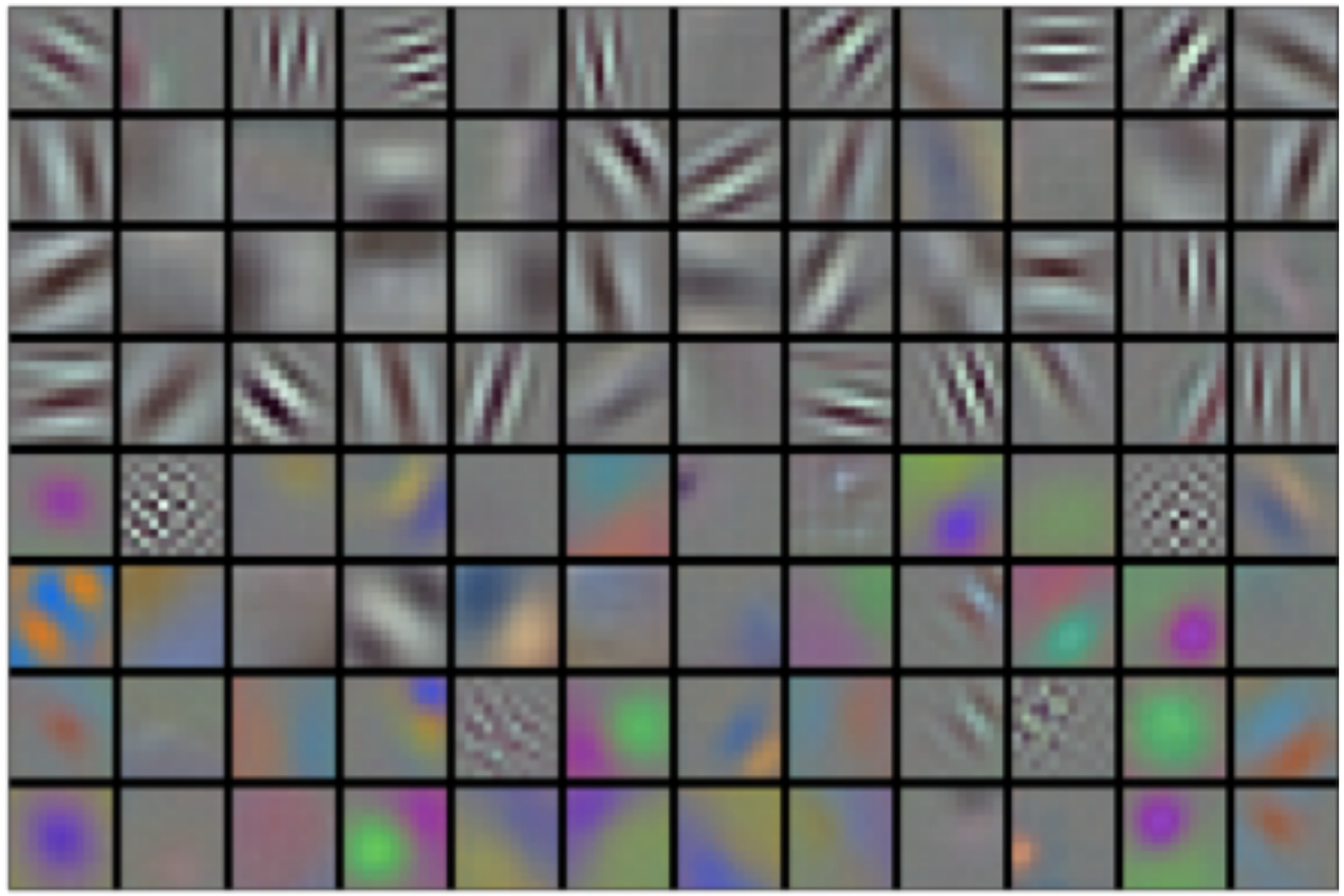
Architettura AlexNet (3)

- Impiega **dropout** sugli strati FC, e **data augumentation**. Nei layer C1 e C3 impiega la **Local response normalization:** se un nodo riceve una attivazione significativa, si inibiscono i nodi nella stessa posizione ma in altre feature maps.
  - Il dropout nei layer FC prende il posto del weight decay della LeNet. Questo garantisce una sorta di regolarizzazione dei parametri
- L'ipotesi è quella di favorire la competitività, specializzando ogni feature map su caratteristiche distinte.

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully Connected	–	1,000	–	–	–	Softmax
F9	Fully Connected	–	4,096	–	–	–	ReLU
F8	Fully Connected	–	4,096	–	–	–	ReLU
C7	Convolution	256	13 × 13	3 × 3	1	SAME	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	SAME	ReLU
S4	Max Pooling	256	13 × 13	3 × 3	2	VALID	–
C3	Convolution	256	27 × 27	5 × 5	1	SAME	ReLU
S2	Max Pooling	96	27 × 27	3 × 3	2	VALID	–
C1	Convolution	96	55 × 55	11 × 11	4	SAME	ReLU
In	Input	3 (RGB)	224 × 224	–	–	–	–

Esempio: Filtri di AlexNet

- Esempi di filtri dei primi layer di Alex Net dopo l'addestramento:



AlexNet e Keras

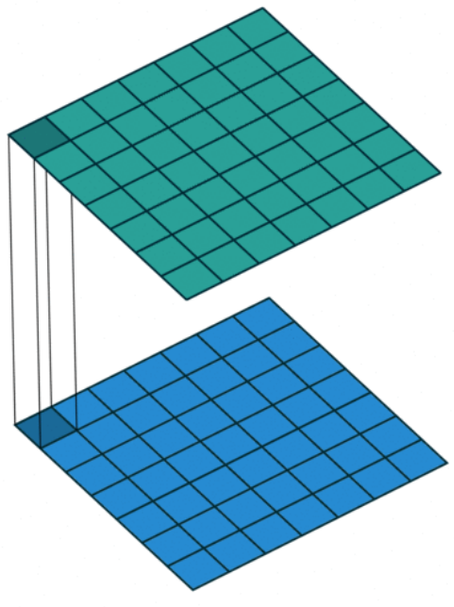
- 08-AlexNet.ipynb

CNN: 1×1 convolution

- La **1×1 convolution** è un filtro di dimensione **1×1×C** e (ovviamente) si applica a input con profondità **C**.
  - Può essere vista come una **rete neurale con un layer,** che prende in input un vettore di **C** elementi.
  - Per **C** pari a **1 non viene impiegato**
    - Un filtro 1×1×1 corrisponde ad una moltiplicazione per uno scalare, operazione inutile in una rete neurale.

- A cosa può servire?

*dimensione **LRF**:*
1×1×C

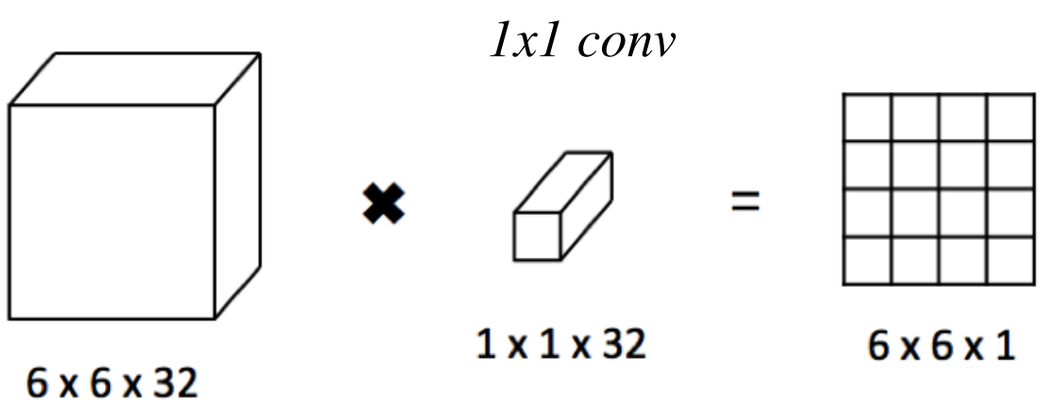


feature map
avrà la stessa
dimensione dell'input
ma profondità pari a 1

output layer precedente:
supponi una profondità C > 1

CNN: 1x1 convolution (2)

- Effettua un **feature pooling** cioè combina linearmente più features legate tra loro da un certa legame spaziale (es. i 3 valori dei canali RGB di un pixel).
  - Utile quando si hanno feature maps con grande profondità e si vuole ridurre il numero di paremetri nei layer successivi.
  - Mentre il **pooling** tradizionale aggrega più feature vicine all'interno della stessa feature map.
- Se in input abbiamo una feature maps con profondità **C**, ogni mappa rappresenterà l'importanza di una diversa feature in una certa posizione. La **1×1 convolution** raccoglie le informazioni di **C** features diverse valutate nella stessa posizione per determinare un singolo output.



La profondità è passata da 32 a 1.

Impiegano n filtri 1x1 conv, otteniamo una profondità n della feature maps in output.

CNN: 1x1 convolution (3)

- Oltre a ridurre il numero di parametri nei layer successivi in presenza di feature maps con grande profondità, la **1×1 convolution** viene usata anche per creare nuove **proiezioni lineari** a partire dalle feature map correnti.
  - Le proiezioni creazioni **nuove features** determinate dalla combinazioni di più feature maps nei layer precedenti.