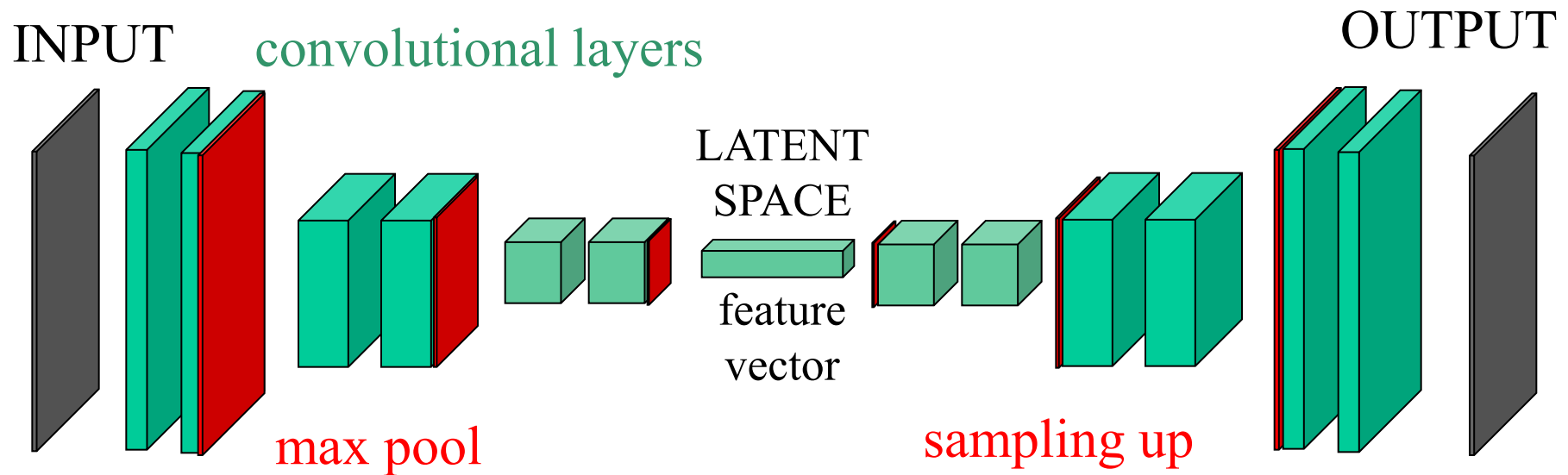
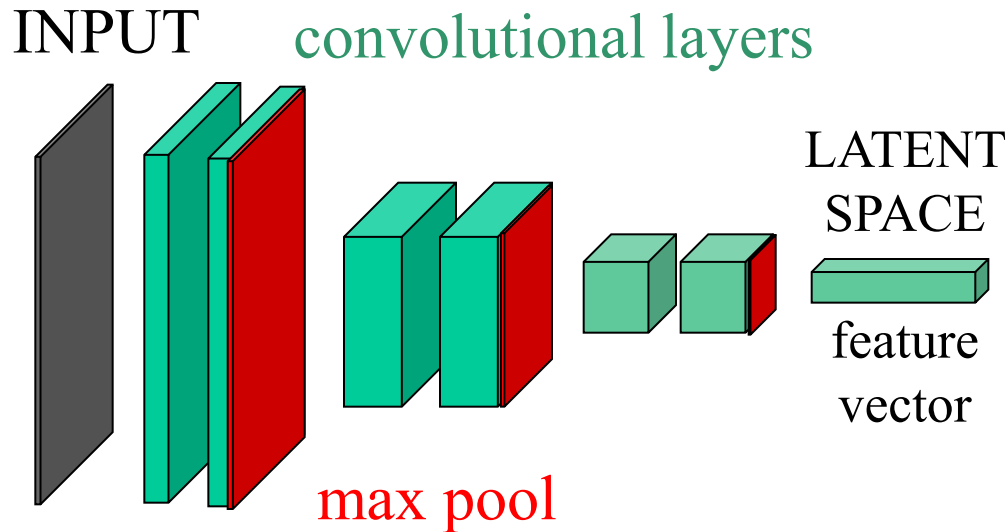


# (Convolutional) autoencoder

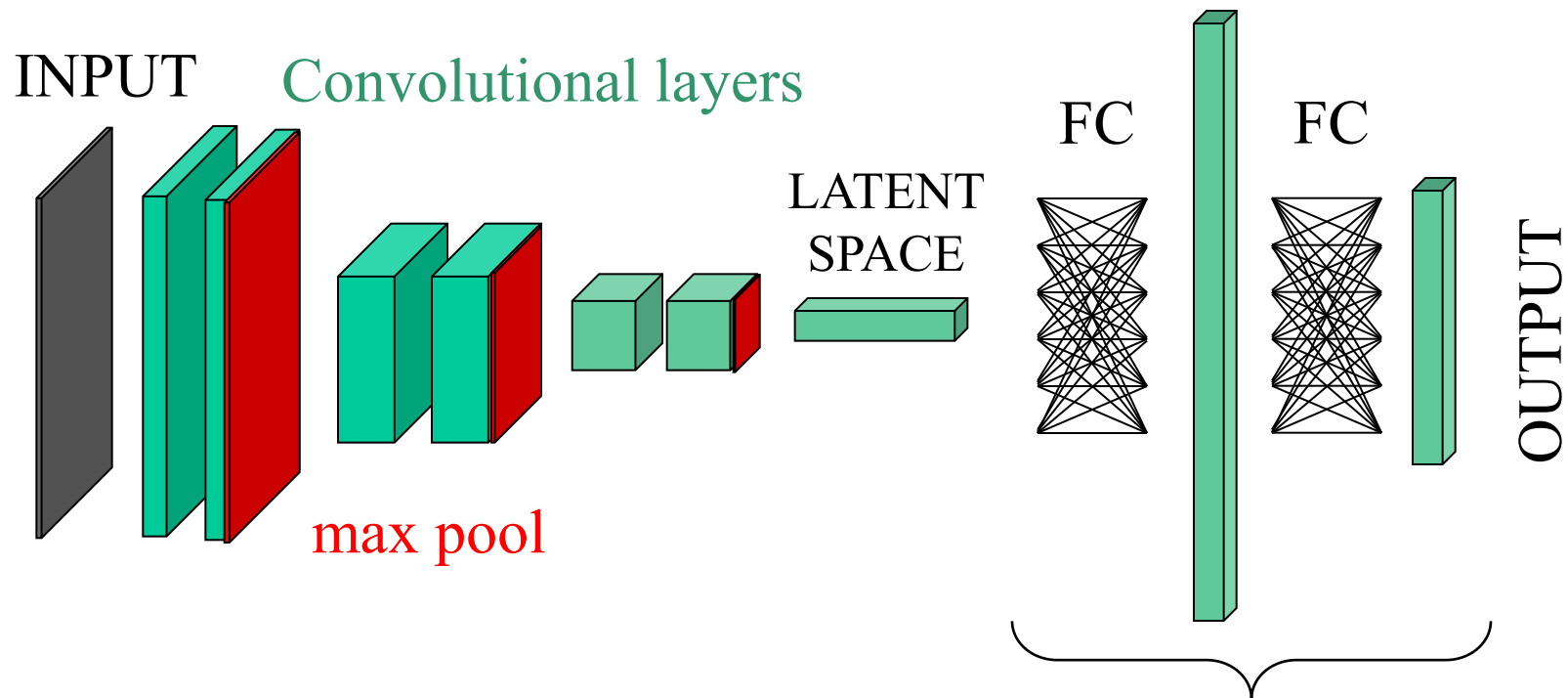


# Encoder



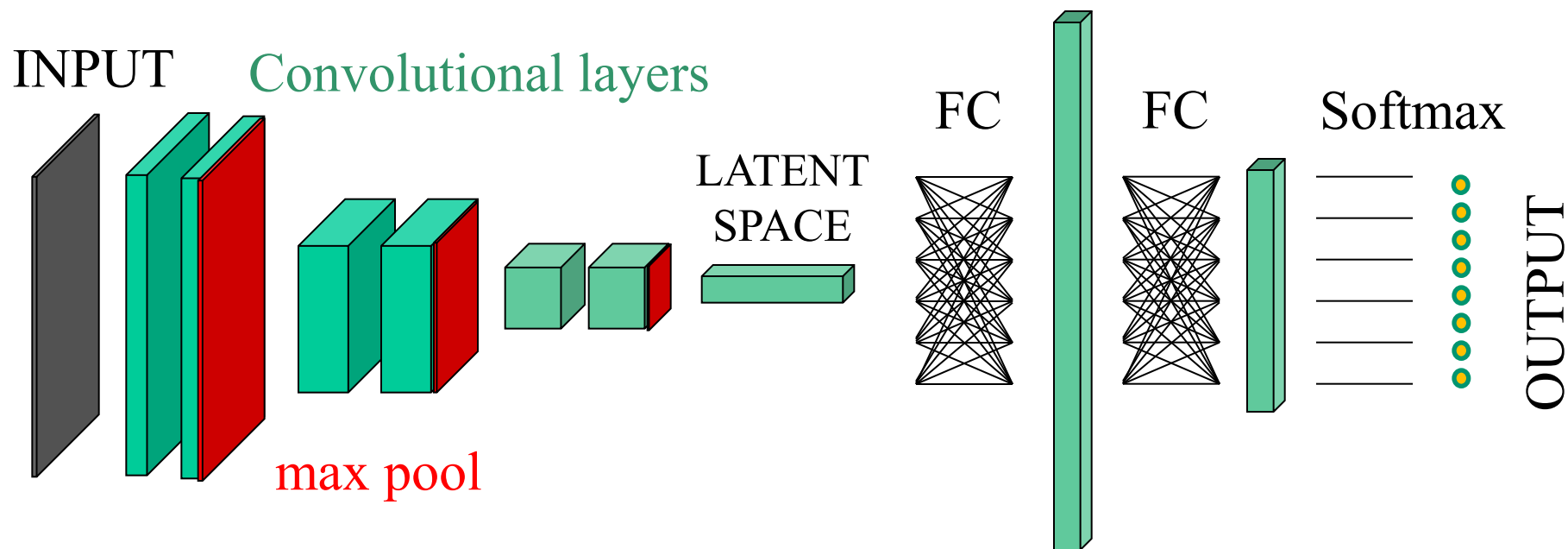
Encoder (the first half of autoencoder) transforms image to features (point in the latent space) ...

# Regressor



... and for features perceptron works  
also in practice

# Classifier



pridaním Softmaxu alebo Sigmoidy vieme  
regressor premeniť na klasifikátor

logit  
Softmax mení čokoľvek na pravdepodobnosti

$$\text{softmax}_i(x) = \frac{e^{x_i}}{\sum_k e^{x_k}}$$

```
def softmax(x):  
    e_x = np.exp(x - np.max(x))  
    return e_x / e_x.sum(axis=0)
```

Softmax má zmysel len keď sa kategórie navzájom vylučujú.  
Pokiaľ máme napr. koleso aj bicykel, používame Sigmoid

# Základná vlastnosť DNN

- Keď si vieme zdôvodniť ako nejaký systém urobiť postupne a po častiach (spravíme autoencoder, odrežeme dekóder, pripojíme perceptron, ...), môžeme ho urobiť priamo v koncovej architektúre (End-to-End system)

# Trénovanie neurónových sietí

- Pri komplikovanejších sieťach trénovanie ľahko uviazne v lokálnom minime
- Do siete preto pridávame stavebné prvky, ktoré sa tomu snažia brániť
- Pozornosť tiež venujeme inicializácii váh

# Dropout

- Počas trénovania sa náhodne pokazí prenos signálu medzi dvomi vrstvami (pošle sa nula)
- Je to prevencia uviaznutia v lokálnom minime a vedie to k lepšiemu rozloženiu zodpovednosti medzi neuróny v rámci jednej vrstvy



# Xavier distribution

- Inicializácia váh má veľký vplyv na rýchlosť i výsledok trénovania
- Veľmi dobrou pre siete s ReLU je Xavierova (= Glorotova) inicializácia:

$$U[-x, x] \quad \text{kde} \quad x = \sqrt{\frac{6}{N+M}}$$

kde  $N$  je počet vstupov a  $M$  počet neurónov

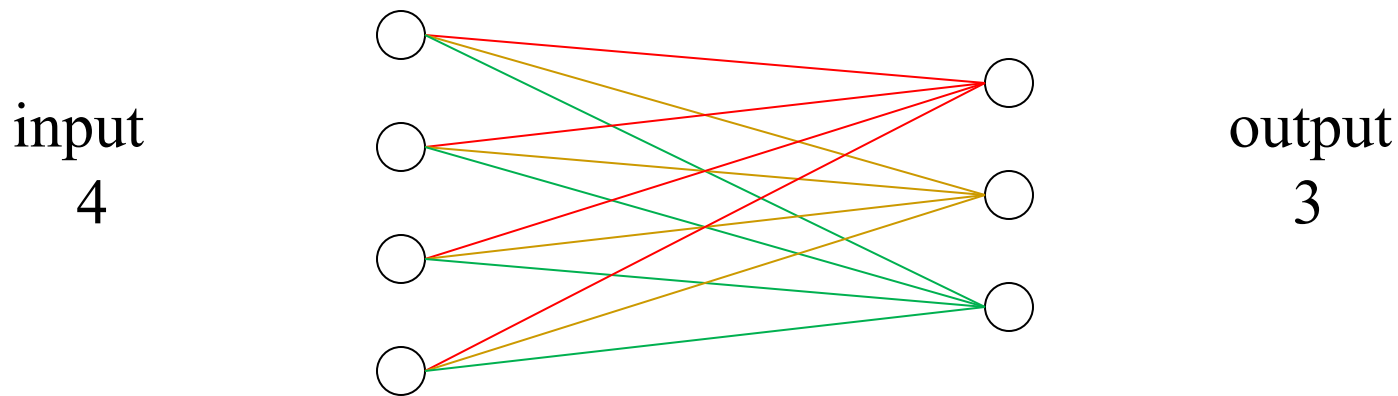
# Kaiming distribution

- Defaultnou inicializaciou v Pytorch je Kaimingova inicializácia:

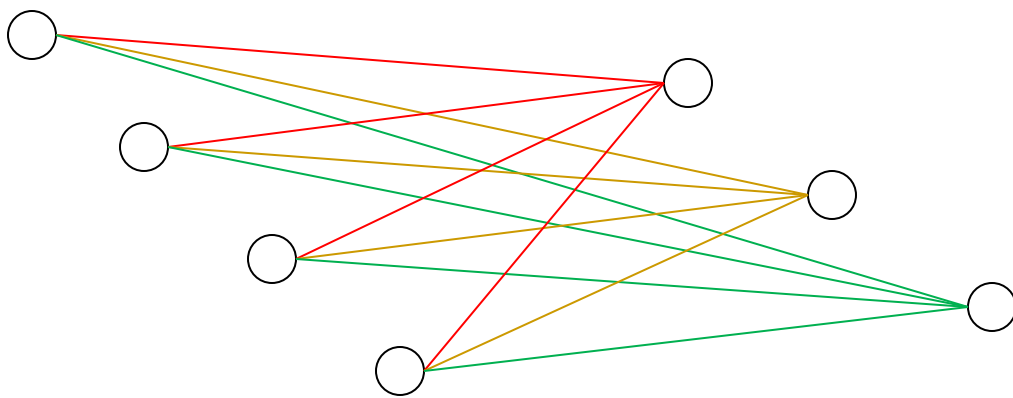
$$U[-x, x] \text{ kde } x = \sqrt{\frac{6}{N}}$$

kde  $N$  je počet vstupov a  $M$  počet neurónov

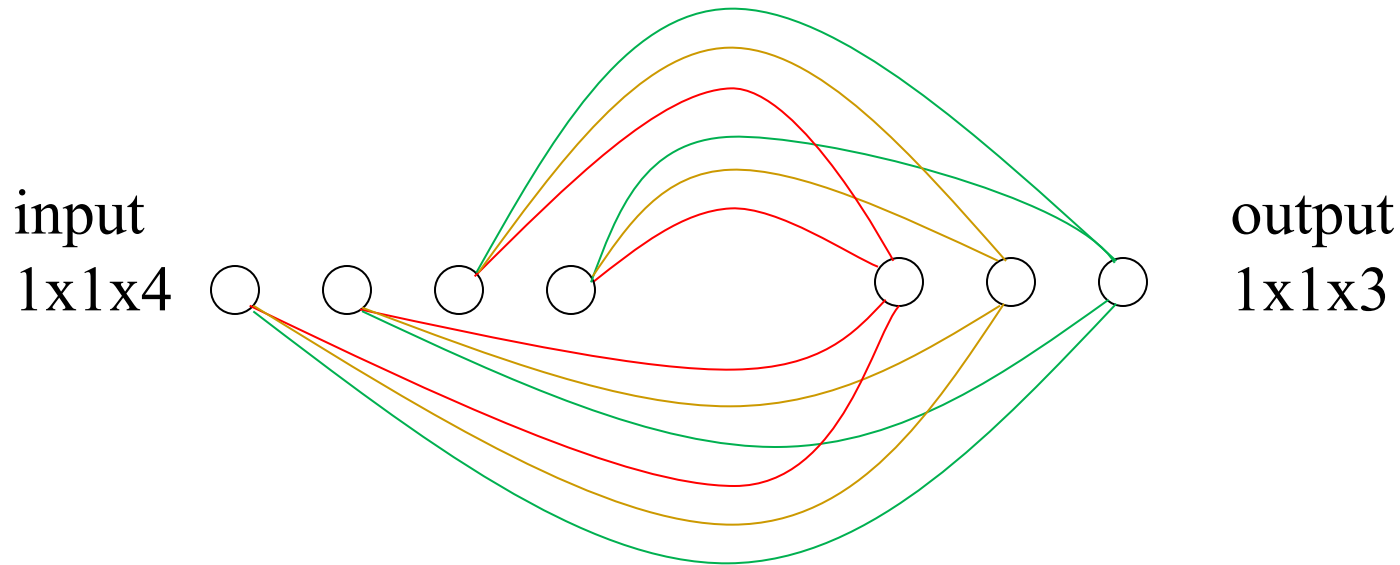
Fully – Connected layer (Linear) s  $N$  vstupmi a  $M$  neurónmi ( $N \times M + M$  parametrov) zodpovedá bloku  $M$  konvolučných vrstiev prepojených na vstup  $1 \times 1 \times N$  (rozlíšenie  $1 \times 1$  a  $N$  kanálov) s kernelom  $1 \times 1$



Fully – Connected layer (Linear) s  $N$  vstupmi a  $M$  neurónmi ( $N \times M + M$  parametrov) zodpovedá bloku  $M$  konvolučných vrstiev prepojených na vstup  $1 \times 1 \times N$  (rozlíšenie  $1 \times 1$  a  $N$  kanálov) s kernelom  $1 \times 1$

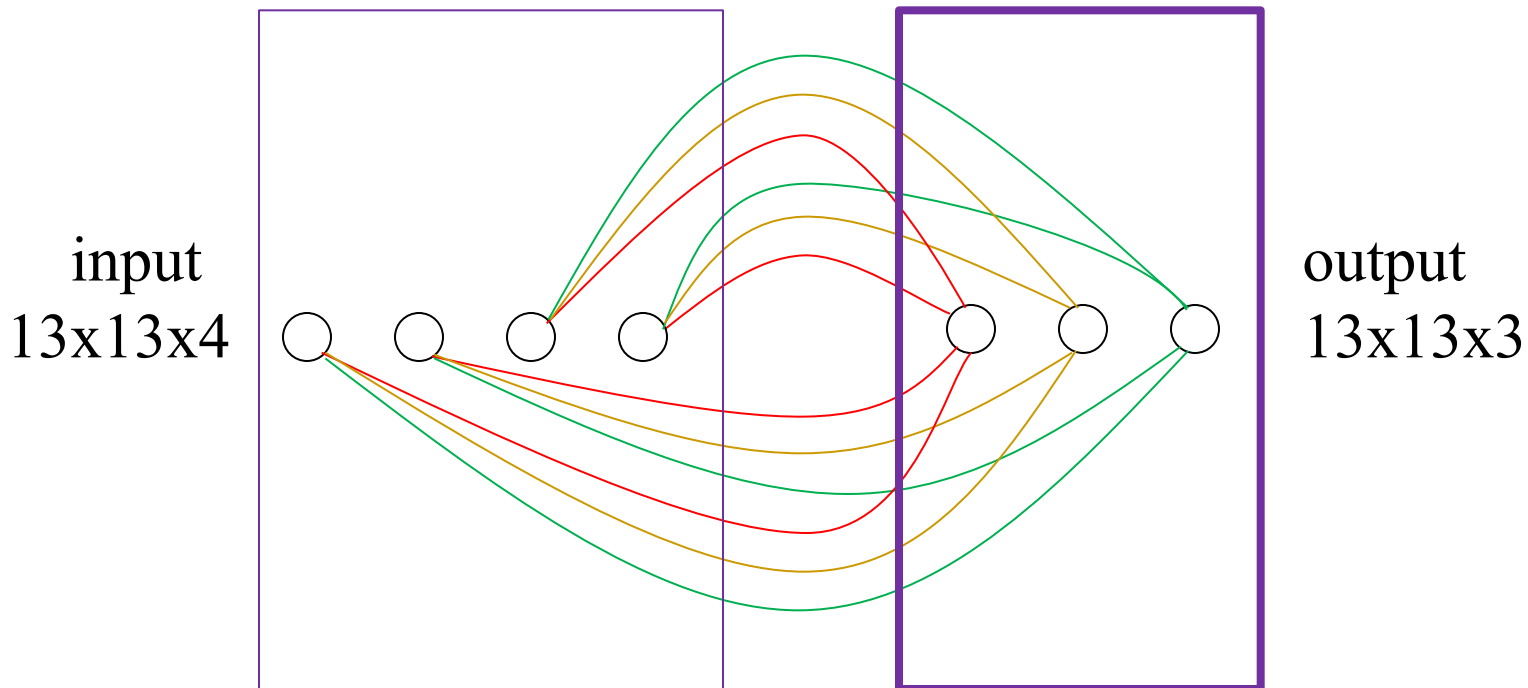


Fully – Connected layer (Linear) s  $N$  vstupmi a  $M$  neurónmi ( $N \times M + M$  parametrov) zodpovedá bloku  $M$  konvolučných vrstiev prepojených na vstup  $1 \times 1 \times N$  (rozlíšenie  $1 \times 1$  a  $N$  kanálov) s kernelom  $1 \times 1$



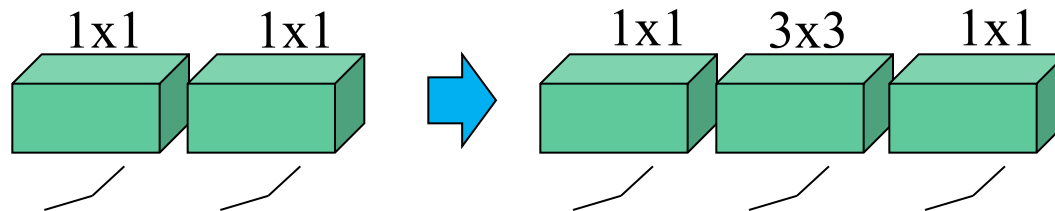
Ked' teraz ako vstup vložíme niečo s väčším resolution, napríklad  $13 \times 13 \times 4$ , spustíme 169 paralelne bežiacich perceptrónov, ktoré zdieľajú váhy a biásy

block of 3 convolutional  
layers with kernel  $1 \times 1$



# Čo je vlastne CNN?

- Čo ak teraz použijeme kernel väčší ako  $1 \times 1$ , napríklad  $3 \times 3$ ?
- Zabezpečí to výmenu informácií medzi susednými paralelne bežiacimi perceptrónmi.



# Čo je vlastne CNN?

- CNN je teda sieť, kde paralelne púšťame rovnaké spolupracujúce perceptrony nad rôznymi miestami obrazu



# DNN

- Povalením perceptronov sa značne zväčšuje počet vrstiev sietí
- Preto sa im hovorí hlboké - Deep NN

# Fully-convolutional Network

- DNN kde sú všetky Linear nahradené Conv2D sú fully-convolutional
- Tieto siete nemajú (na rozdiel od iných DNN) pevnú resolution vstupu