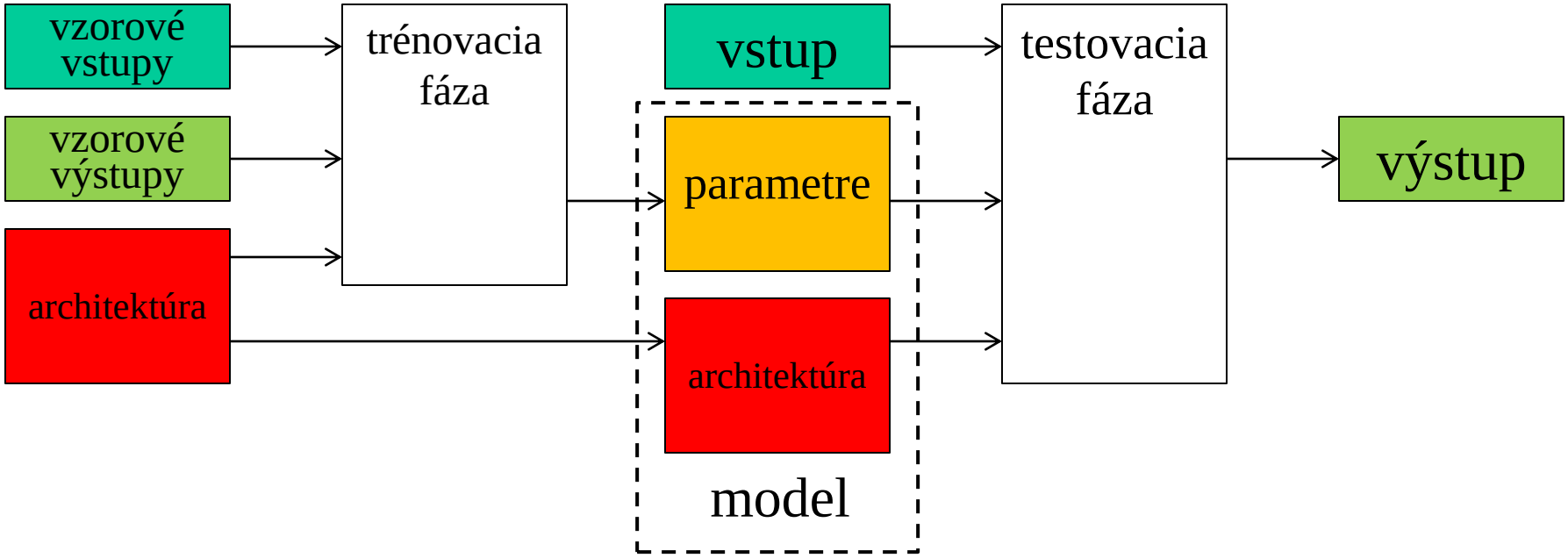
Ako bolo vymyslené hlboké učenie

Andrej Lúčny
Katedra aplikovanej informatiky, FMFI (Matfyz), Univerzita Komenského

Strojové učenie

- Empirický prístup k programovaniu
- Zo vzorových vstupov a výstupov skonštruujeme model
- Pomocou modelu transformujeme ďalšie vstupy



Vzorový výstup určuje kategóriu

na obrázku sú hodinky

Klasifikácia

Regresia

Detektor

Vzorový výstup určuje hodnotu nejakej funkcie vstupu

hodinky sú na obrázku vľavo dole

Vstupné dáta

Input data

Anotácia dátových vzoriek

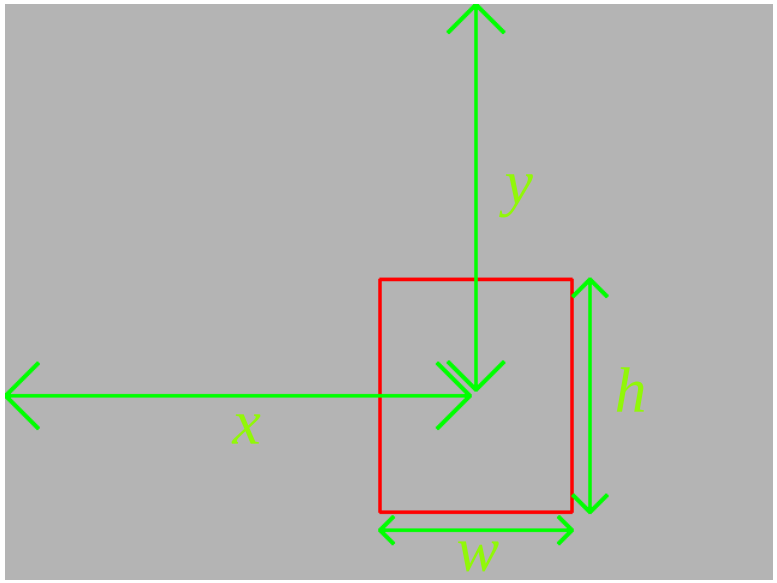
Data annotation

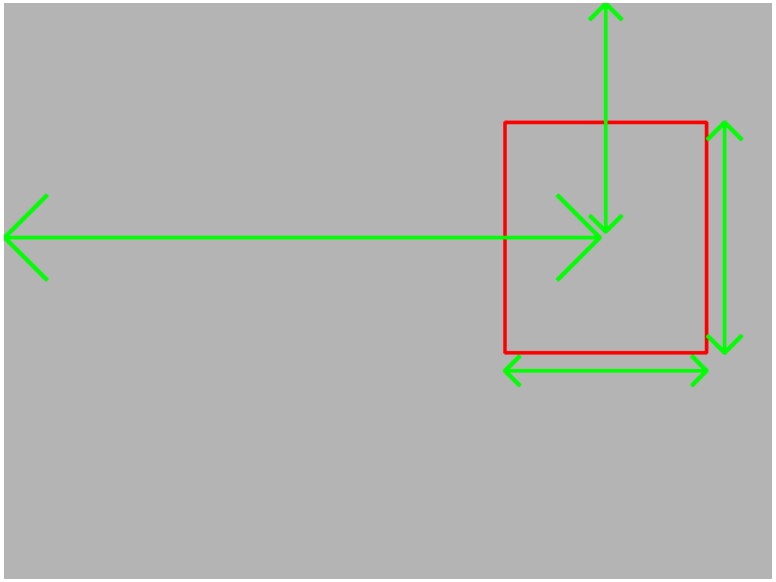
Sada dátových vzoriek

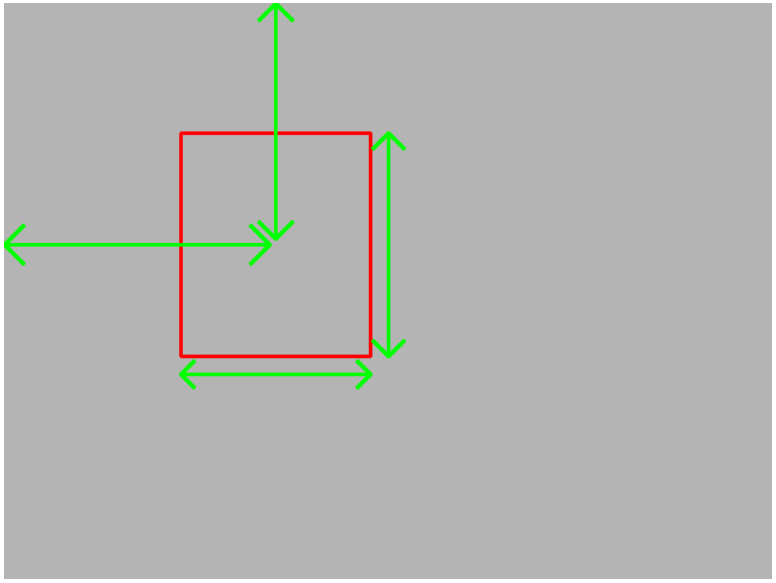
Dataset

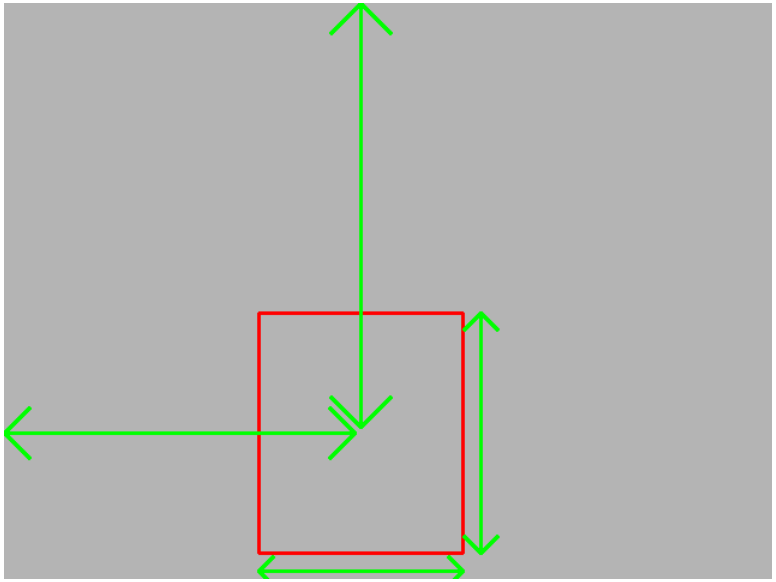
Vzorový vstup:
farebný obraz v RGB
480 x 640 x 3 čísel 0 až 255
premenený na 519168 =
416 x 416 x 3 čísel 0.0 až 1.0

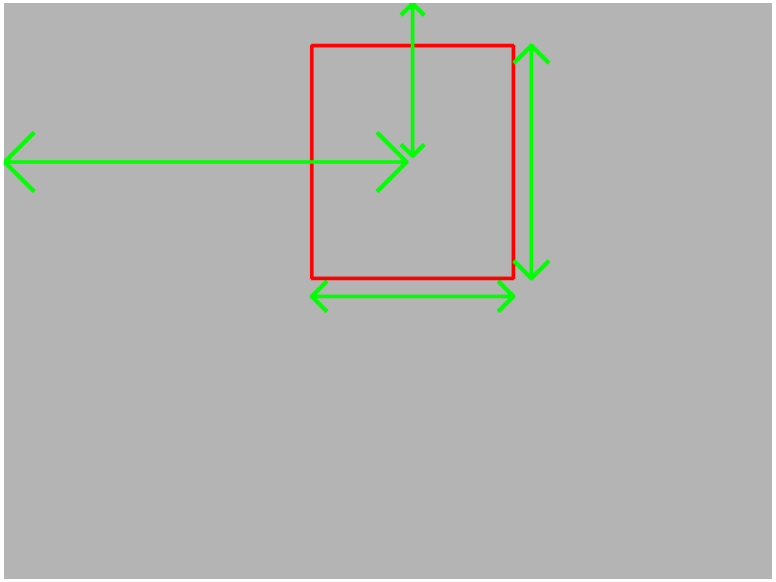
Vzorový výstup:
číslo kategórie: 0 = hodinky
4 čísla 0.0 až 1.0
x, *y* ... pozícia stredu objektu
w, *h* ... šírka a výška objektu

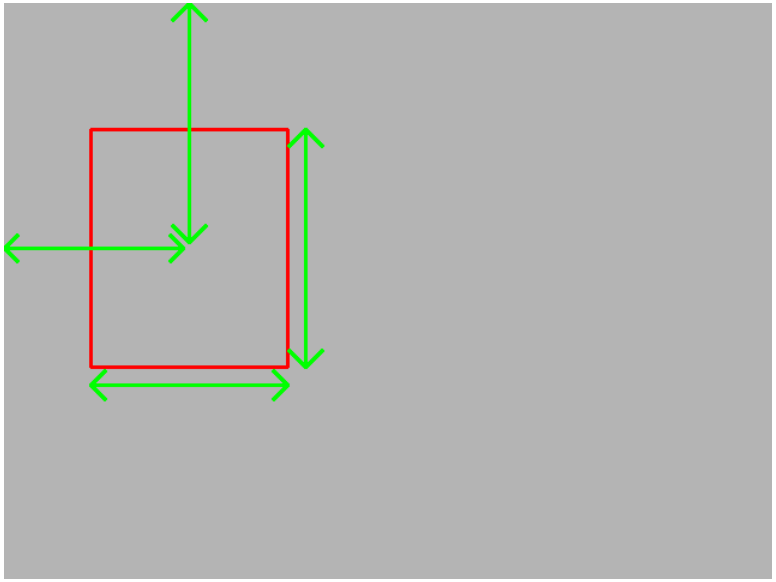


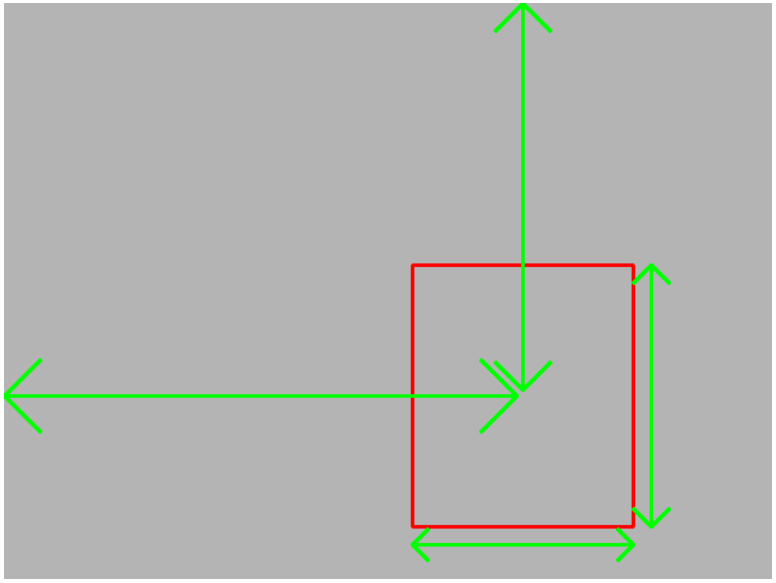


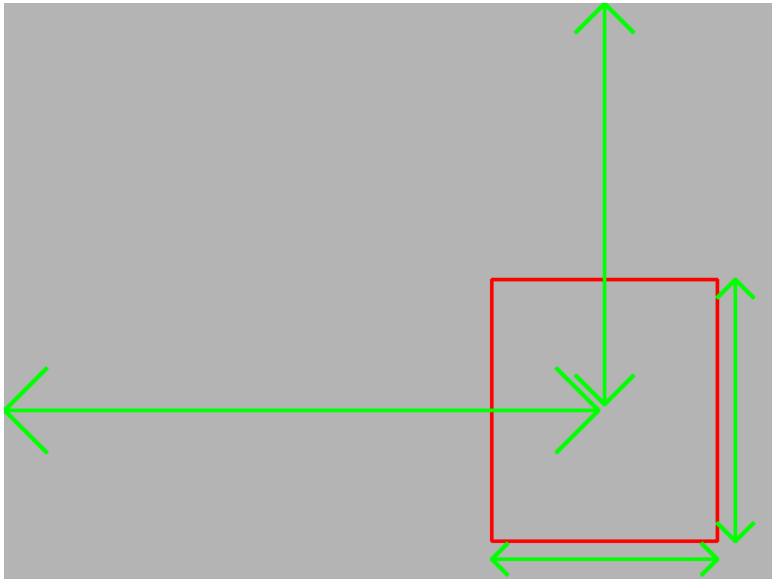


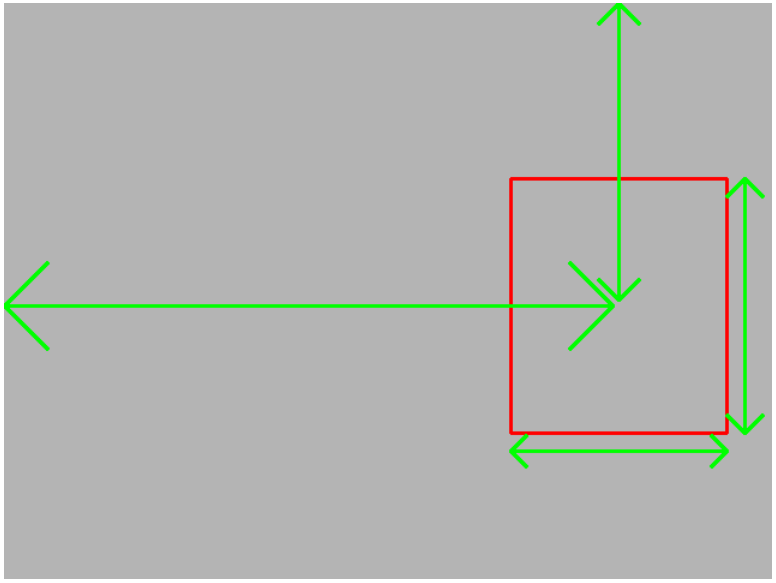


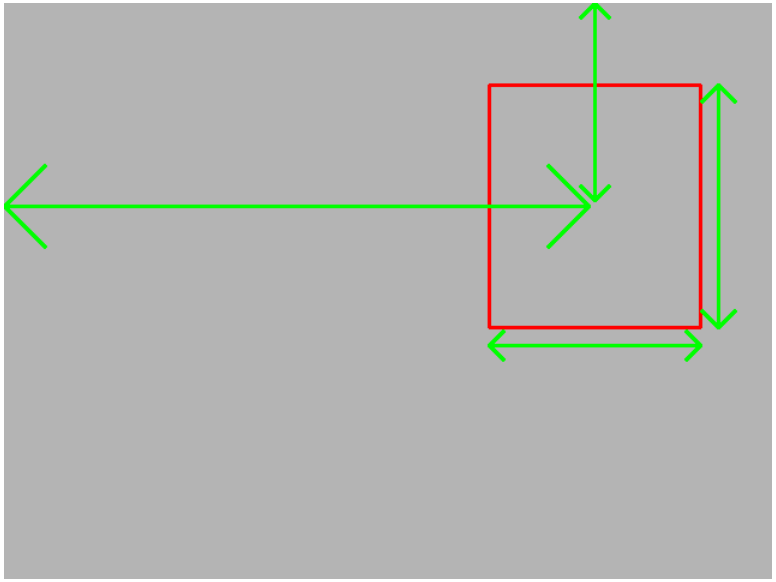


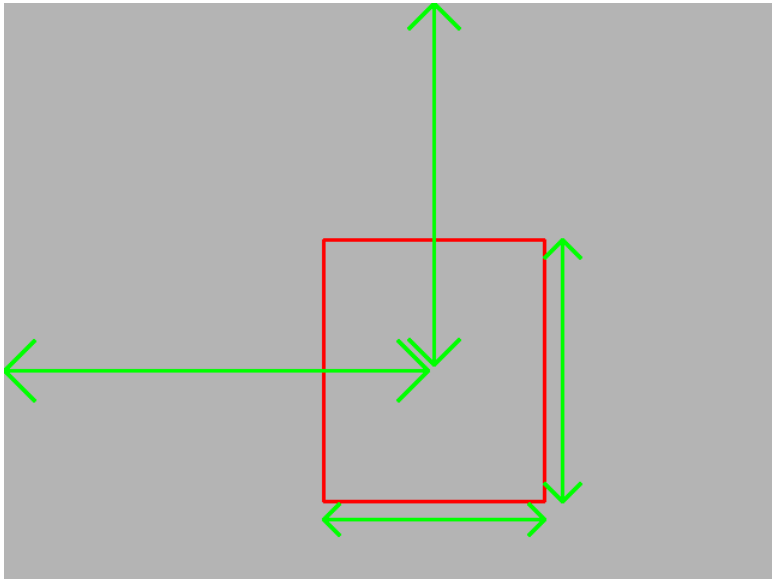


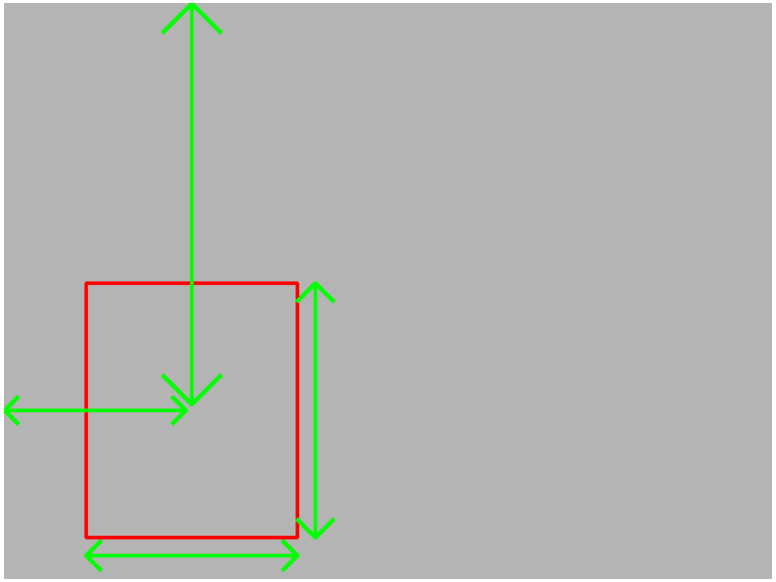






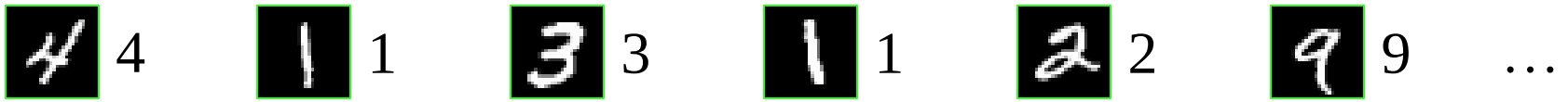






Týchto 12 párov obrázkov a anotácií je v našom príklade celý dataset (bude to stačiť)

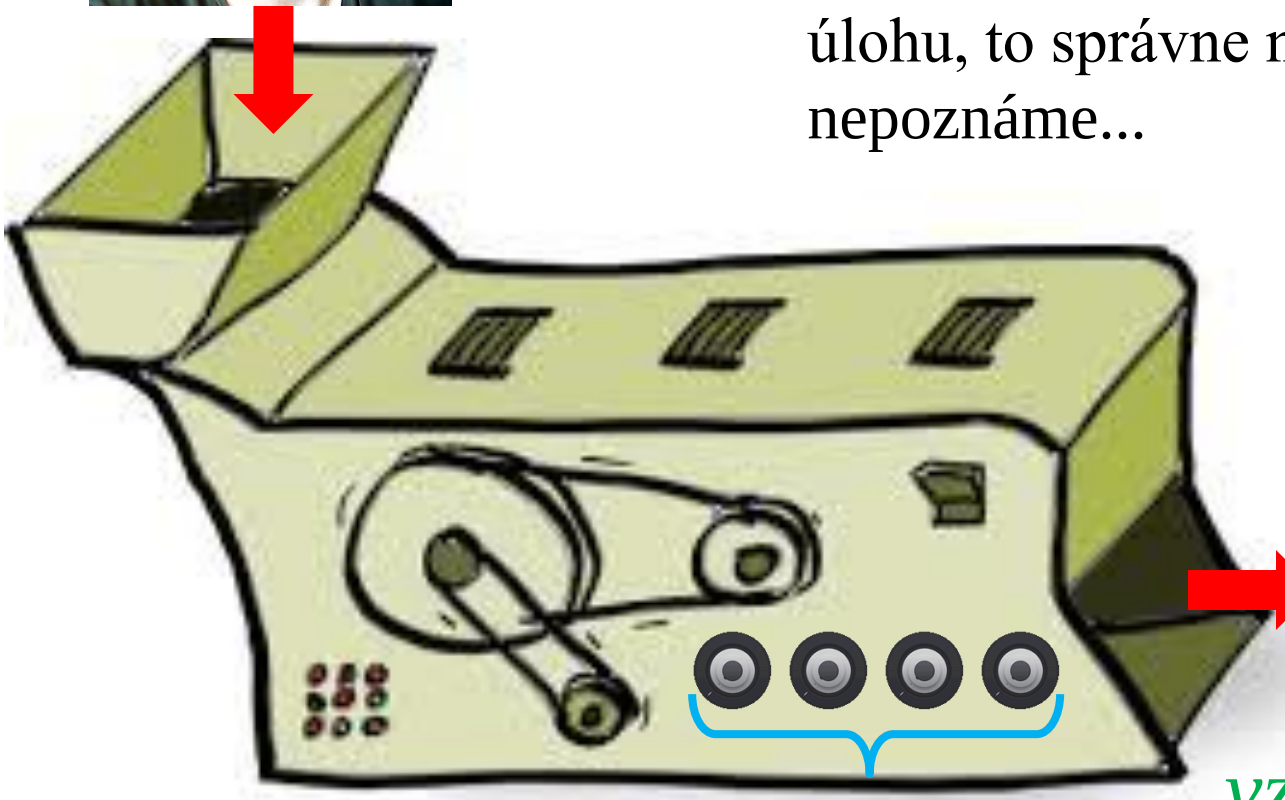
Skutočné datasety majú desaťtisíce príkladov vstupu a želaného výstupu. Napr. dataset MNIST má 60000 vzoriek 28 x 28 x 1:



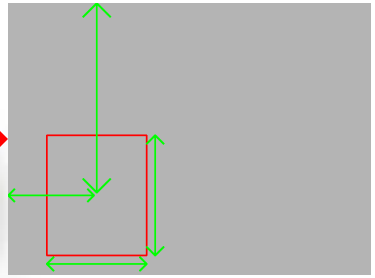
Univerzálny stroj

vzorový vstup

Predstavme si, že máme stroj, o ktorom vieme, že pri správnom nastavení kolečiek, realizuje našu úlohu, to správne nastavenie však nepoznáme...



parametre



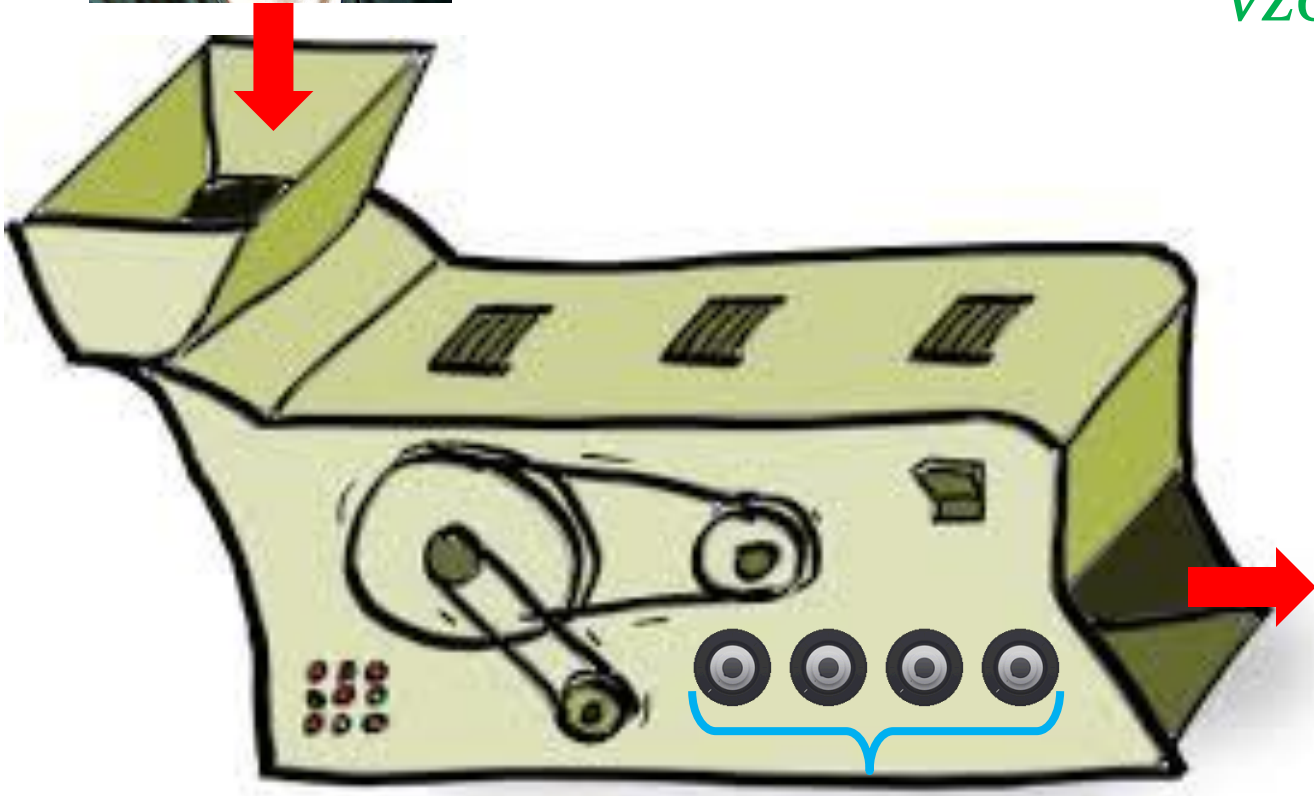
vzorový výstup

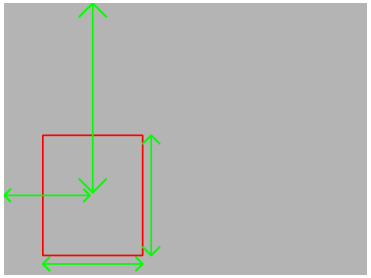
Univerzálny stroj

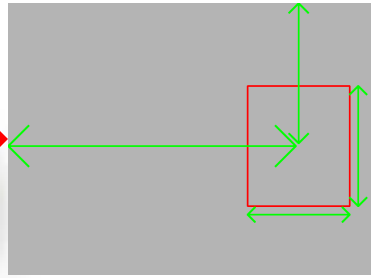
vzorový vstup

Niečo nastavíme, ale nedá nám to taký výstup aký chceme

vzorový výstup







výstup

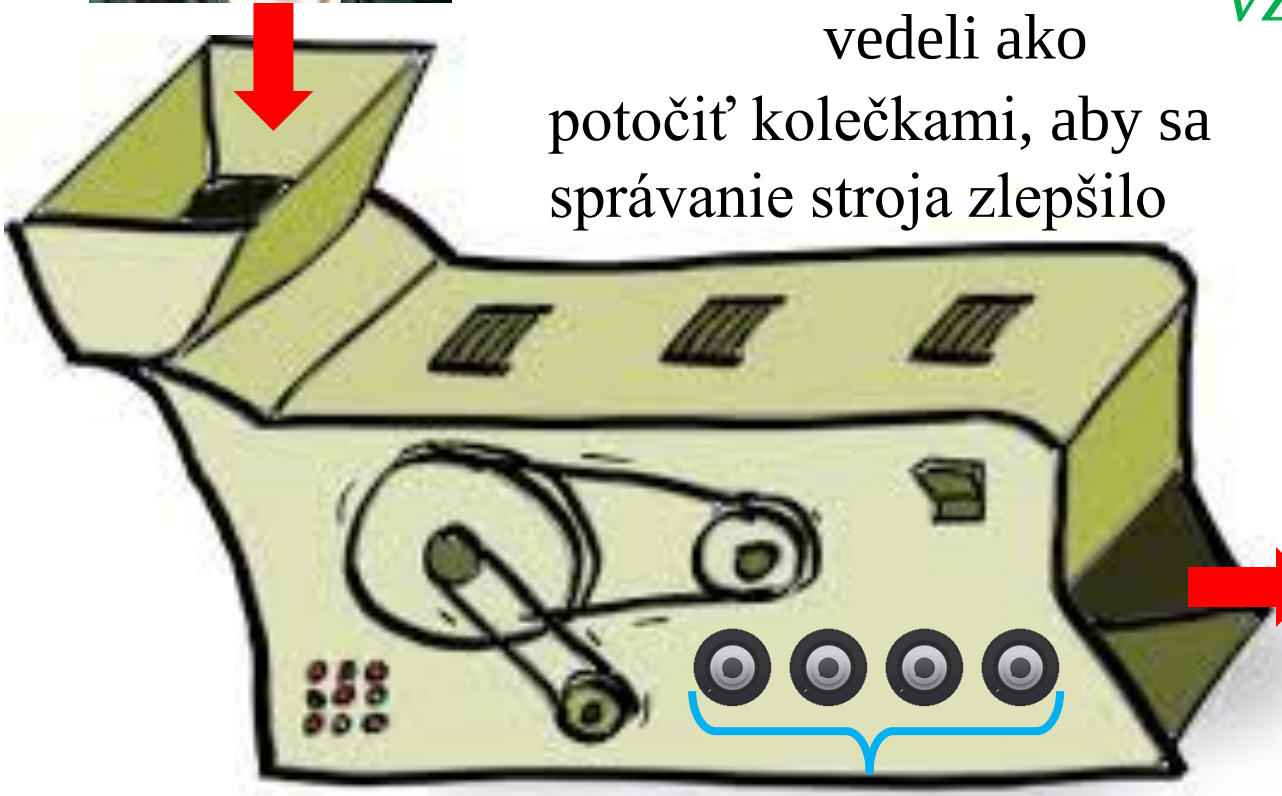
parametre

Univerzálny stroj

vzorový vstup

Určite by sme privítali, keby sme z porovnania vzorového výstupu a výstupu vedeli ako potočiť kolečkami, aby sa správanie stroja zlepšilo

vzorový výstup



výstup

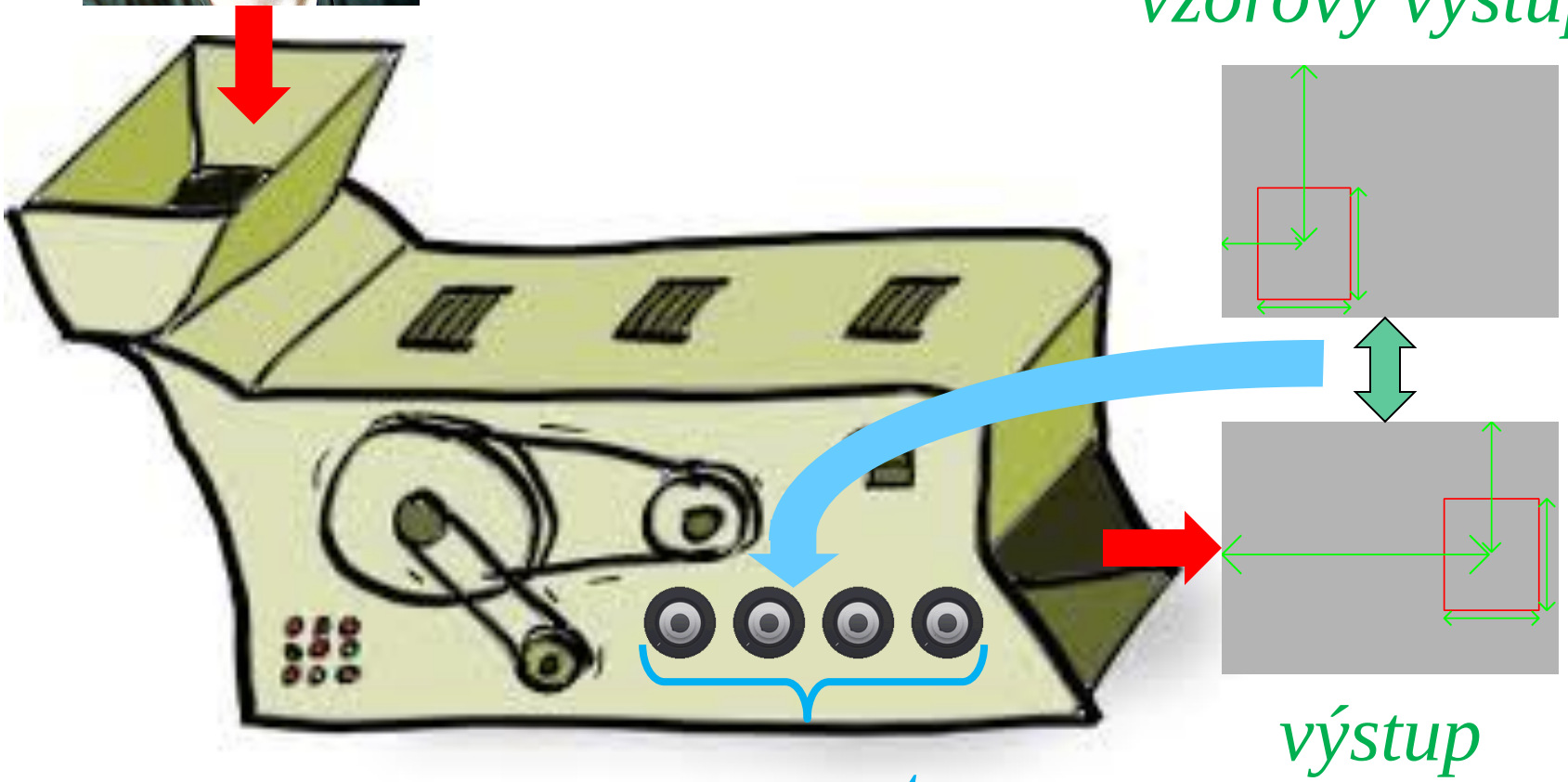
parametre

Neurónová sieť

vzorový vstup

Presne túto vlastnosť má neurónová sieť

vzorový výstup

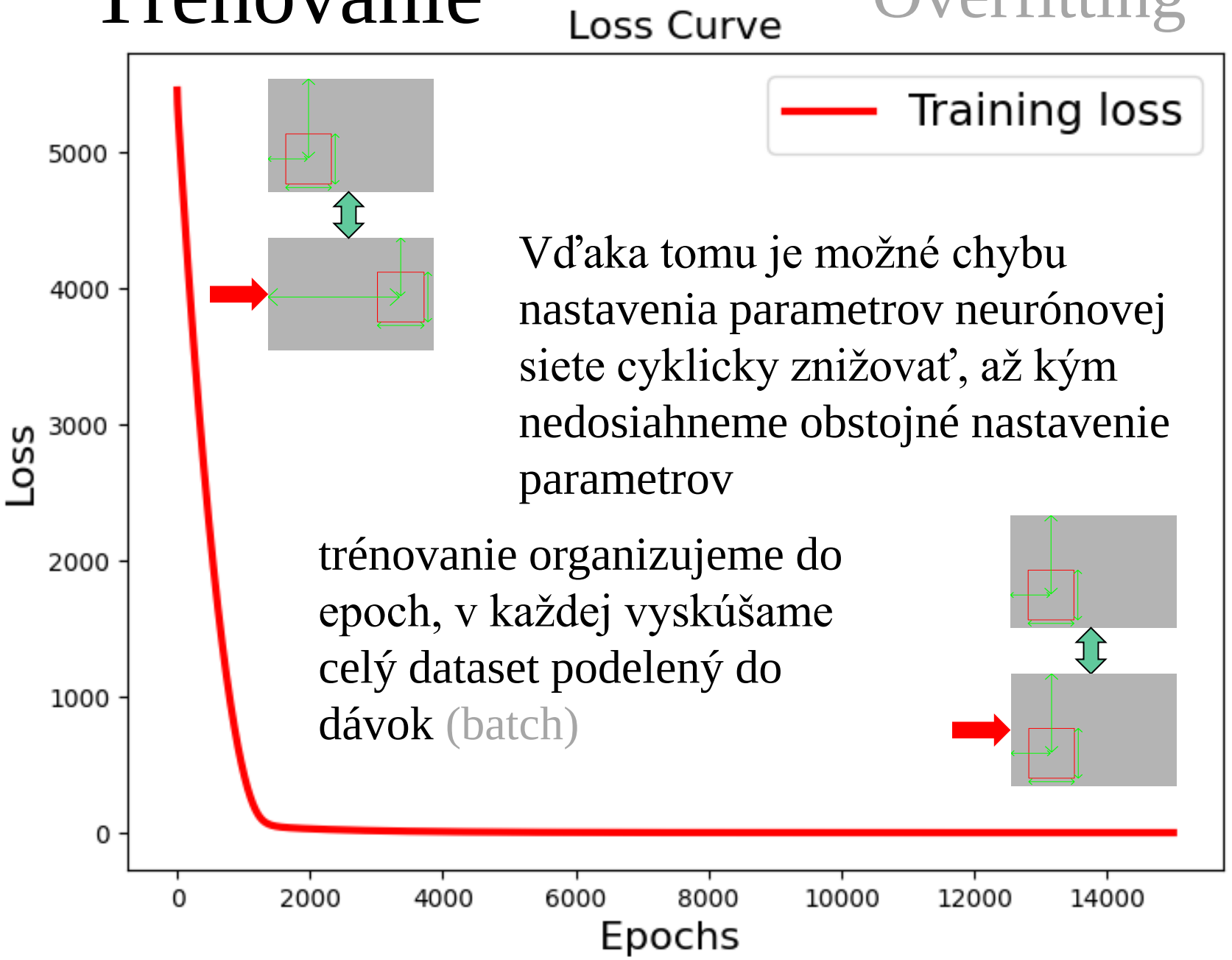


parametre

výstup

Trénovanie

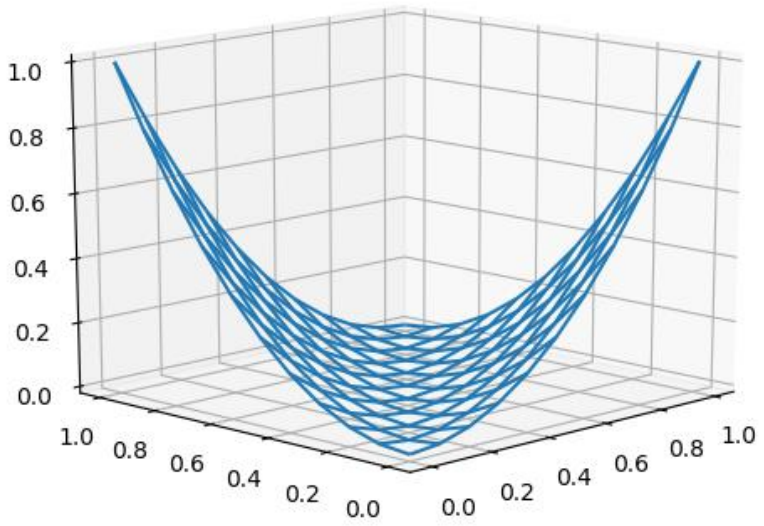
Overfitting

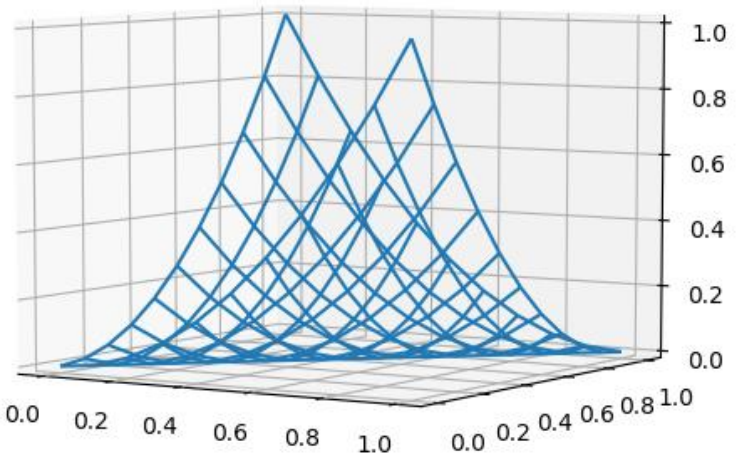


Vďaka tomu je možné chybu nastavenia parametrov neurónovej siete cyklicky znižovať, až kým nedosiahneme obstojné nastavenie parametrov

trénovanie organizujeme do epoch, v každej vyskúšame celý dataset podelený do dávok (batch)

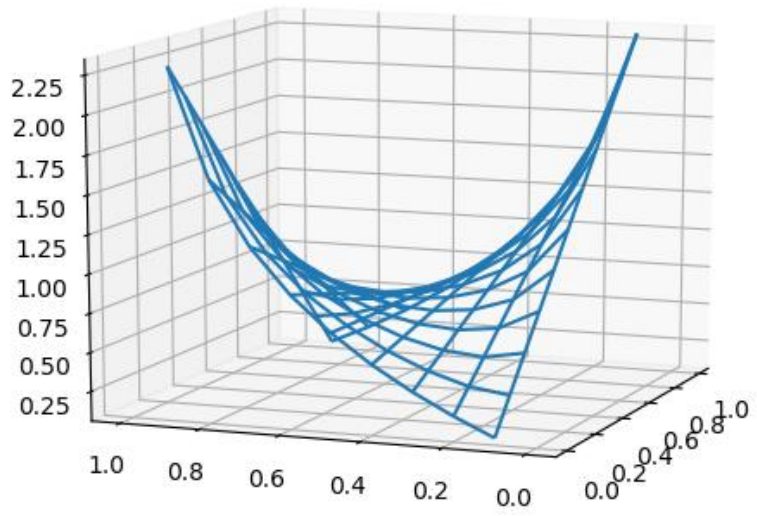
Chybové funkcie

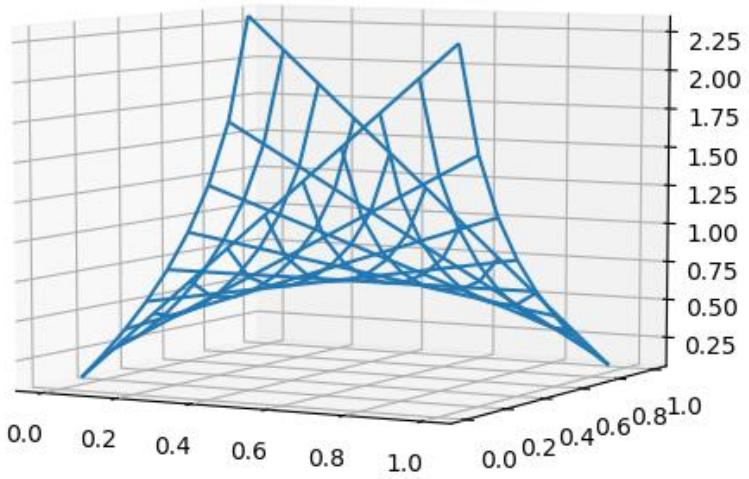




mse

Chybové funkcie



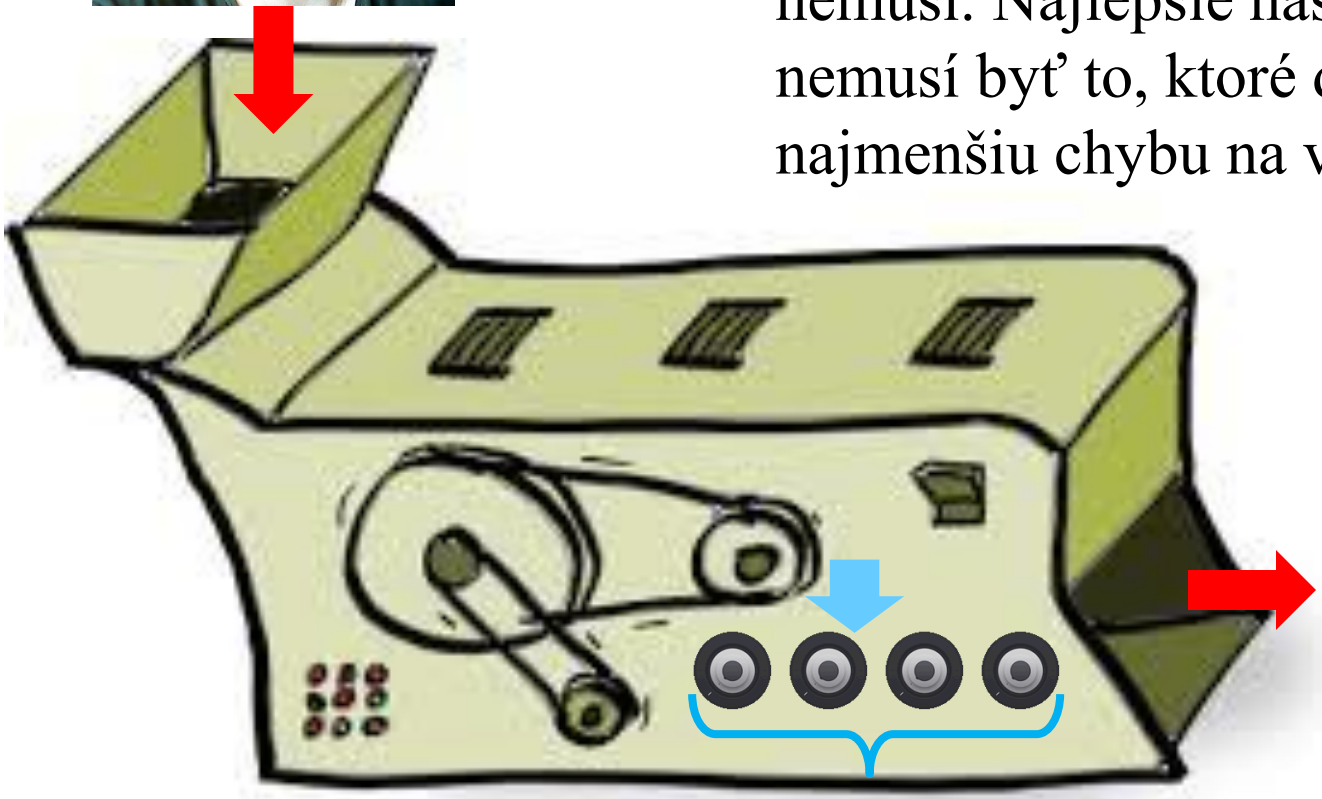


binary cross-entropy

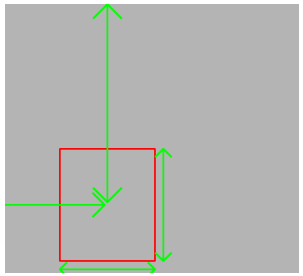
Preučenie

vstup

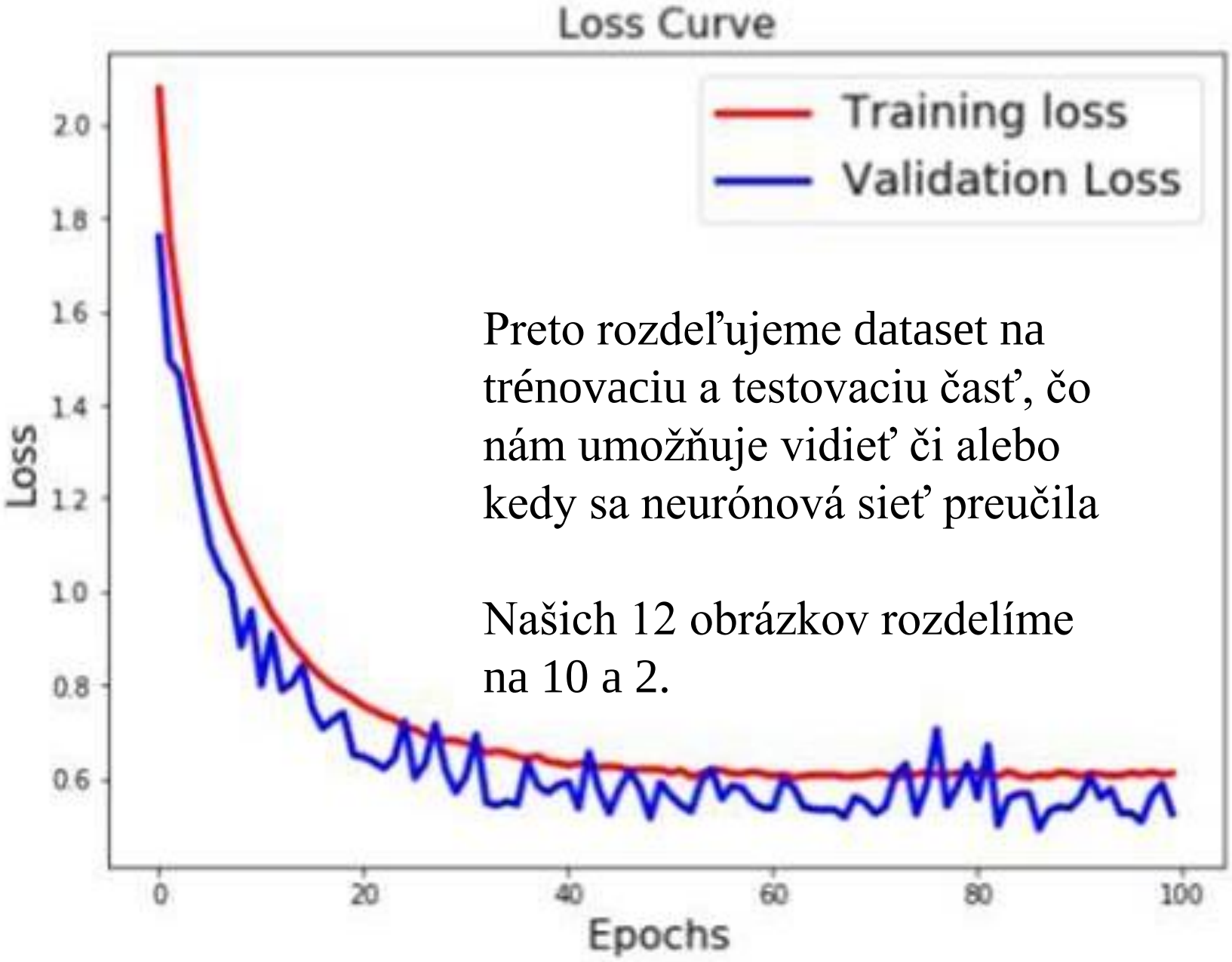
Dúfame potom, že toto nastavenie bude fungovať aj pre vstupy, ktoré nemáme v datasete. Fungovať to nemusí. Najlepšie nastavenie nemusí byť to, ktoré dáva najmenšiu chybu na vzorkách



parametre



výstup



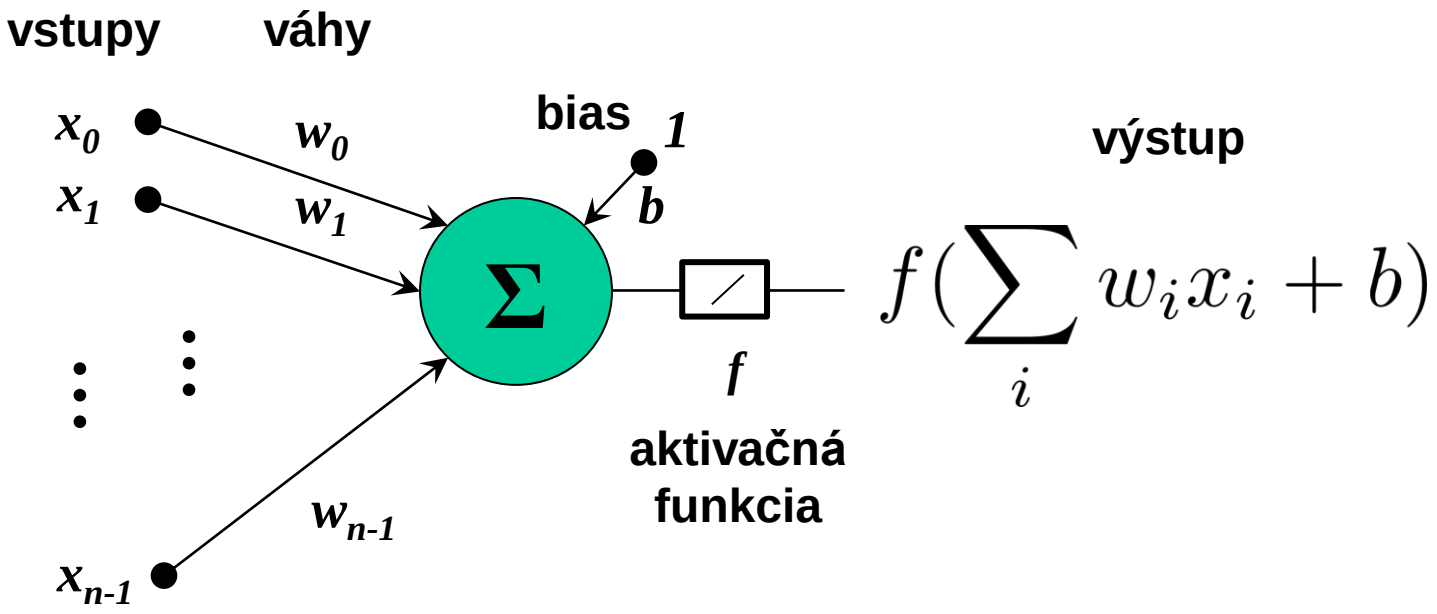
Preto rozdeľujeme dataset na trénovaciu a testovaciu časť, čo nám umožňuje vidieť či alebo kedy sa neurónová sieť preučila

Našich 12 obrázkov rozdelíme na 10 a 2.

Ako vyzerá taká neurónová sieť zvnútra?

- Hoci silnou motiváciou pri vymyslení vhodnej architektúry neurónovej siete bola snaha napodobniť ľudský mozog, podobá sa naň pramálo
- Kvôli tejto motivácii však nazývame základné stavebné prvky neurónmi
- Neuróny sú organizované do vrstiev
- Vrstvy sú organizované do blokov
- Každý blok spracúva jeden tenzor na druhý
- Tenzor je viacrozmerné pole dát, napríklad našich 10 obrázkov predstavuje vstupný tenzor s rozmermi 10 x 416 x 416 x 3

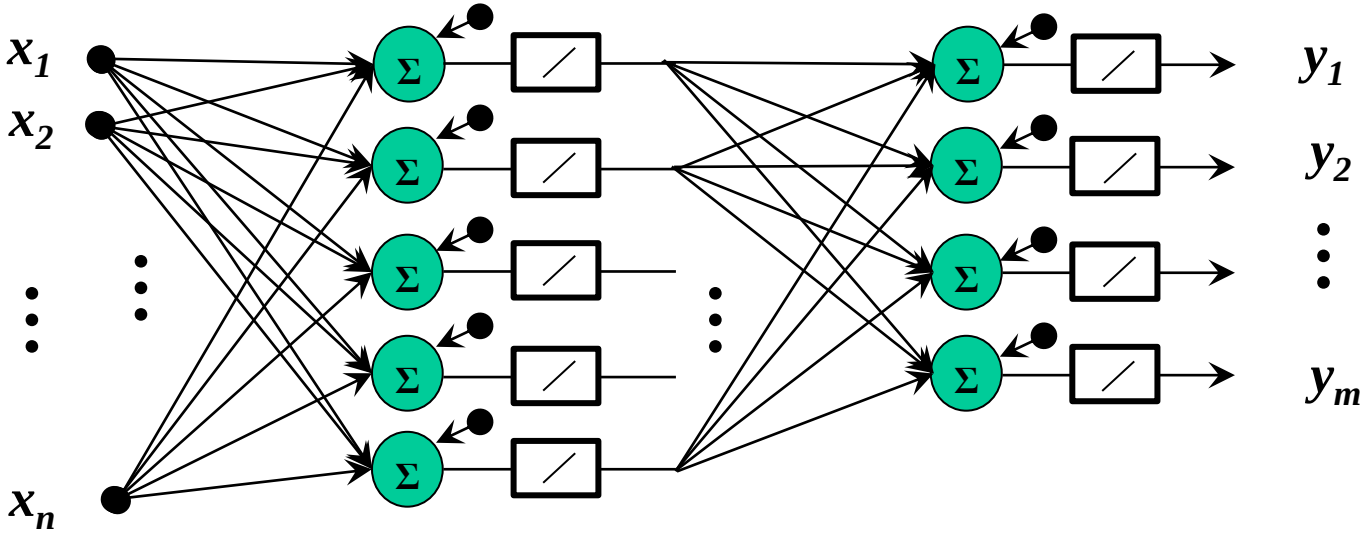
Neurón



- počíta vážený priemer vstupov, pripočíta bias a aplikuje aktivačnú funkciu
- ak má lineárnu aktiváciu, jeho trénovanie zodpovedá lineárnej regresii

[1943 McCulloch & Pitts]

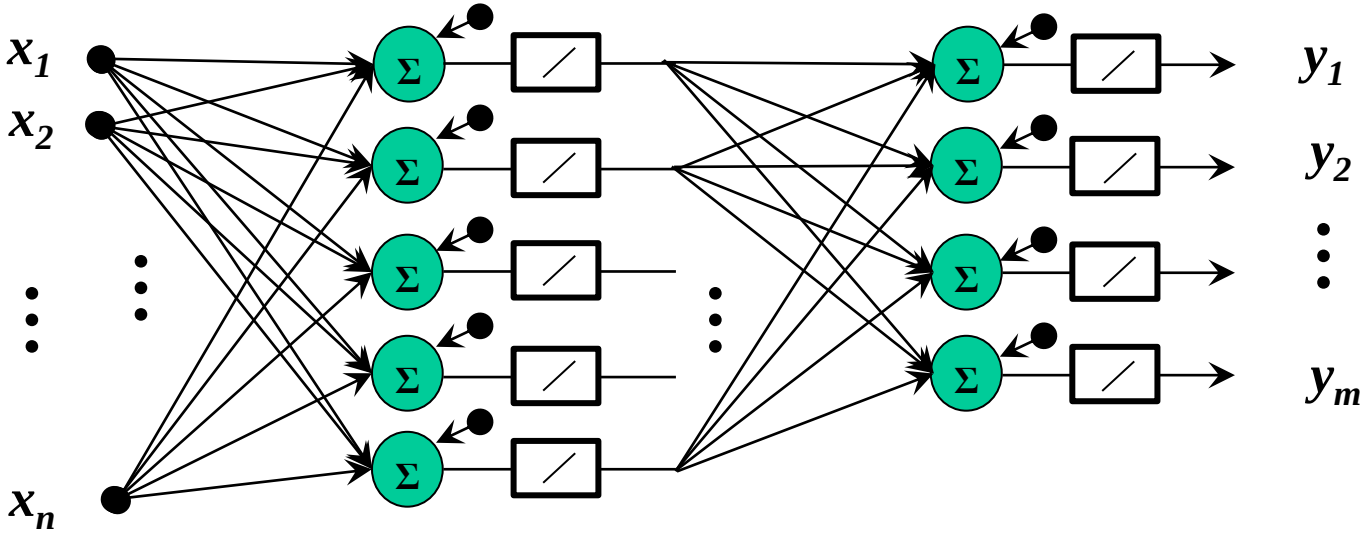
Perceptron



- jedna skrytá a jedna výstupná vrstva
- iba lineárna aktivácia
- možnosť trénovať len váhy výstupnej vrstvy

[1958 Rosenblatt]

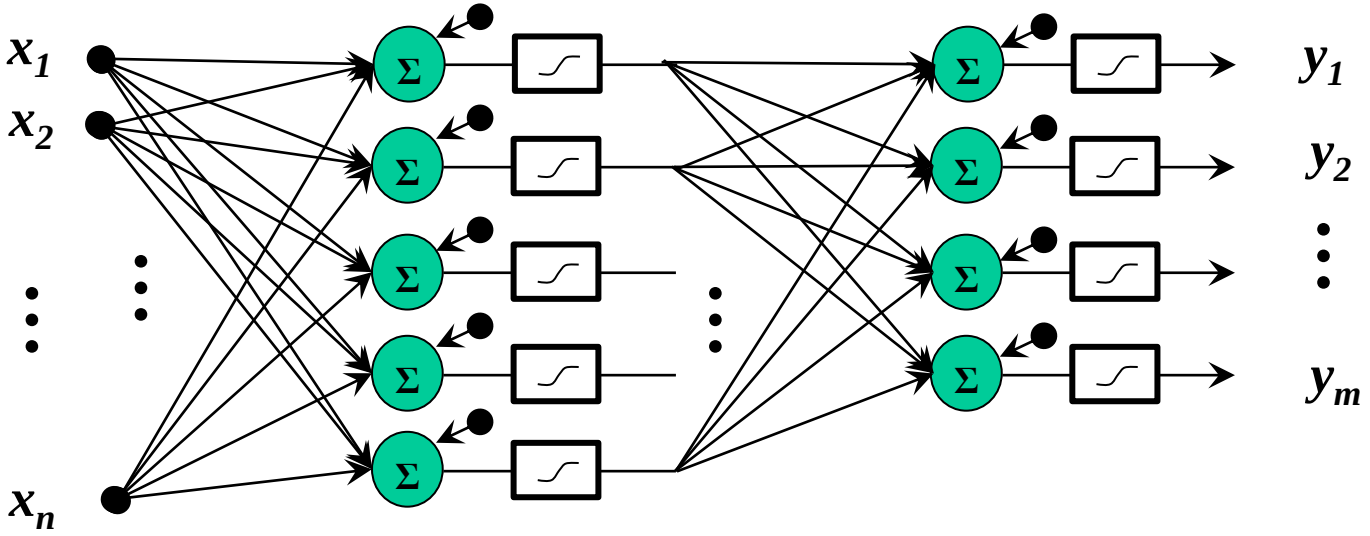
Perceptron



- analýza schopností perceptronu dokázala, že to ho vie dosť málo

[1968 Minsky]

Perceptron + nelinearita + BP

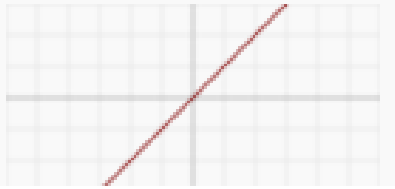


- ako aktivačnú funkciu začali používať sigmoidu a hyperbolický tanges
- Využili Leibnitzovo pravidlo derivácie zloženej funkcie z roku 1676 na trénovanie algoritmom spätnej propagácie (back propagation)

[1986 Rumelhart, Hinton & Williams]

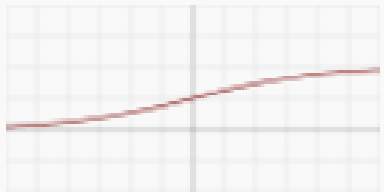
Aktivačné funkcie

linear



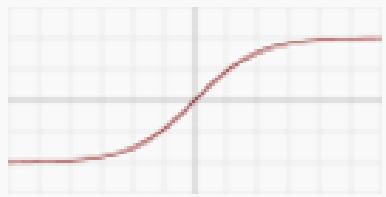
$$f(x) = x$$

sigmoid



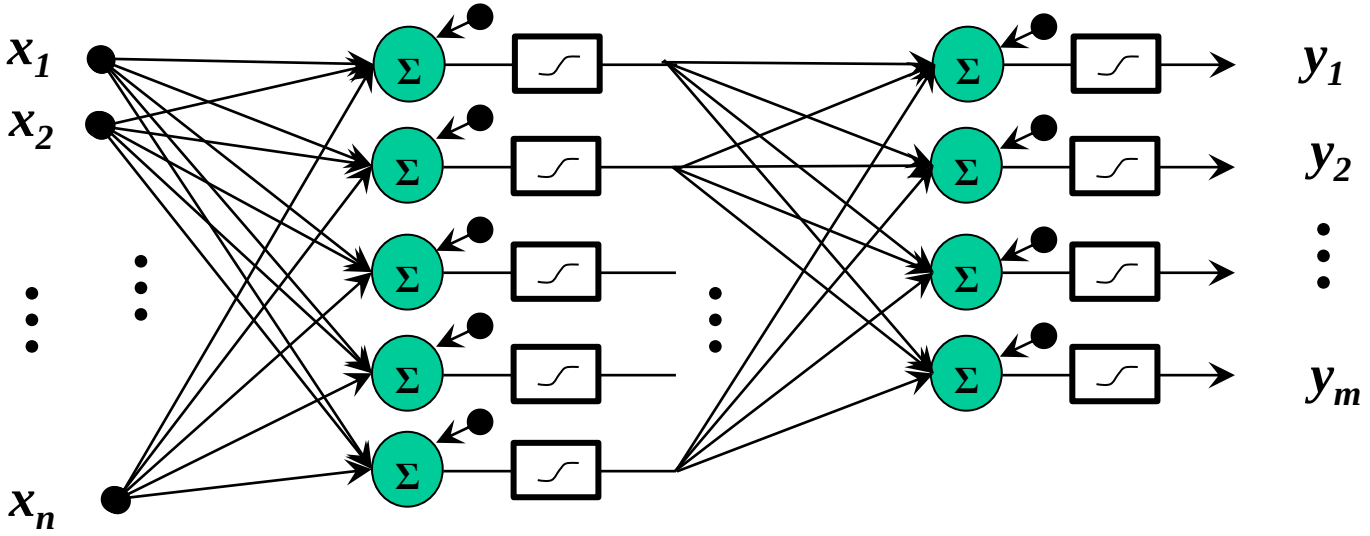
$$f(x) = \frac{1}{1 + e^{-x}}$$

tanh



$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

Univerzálny aproximátor



- matematicky bolo dokázané, že perceptron sa teoreticky dokáže s ľubovoľnou presnosťou naučiť akúkoľvek rovnomerne spojitú funkciu
- prakticky sa to však podarilo uskutočniť len pre vstupy pomerne malej dimenzie, pre naše dáta s dimenziou 519168 = 416 x 416 x 3 je to nepoužiteľné

[1989 Cybenko]

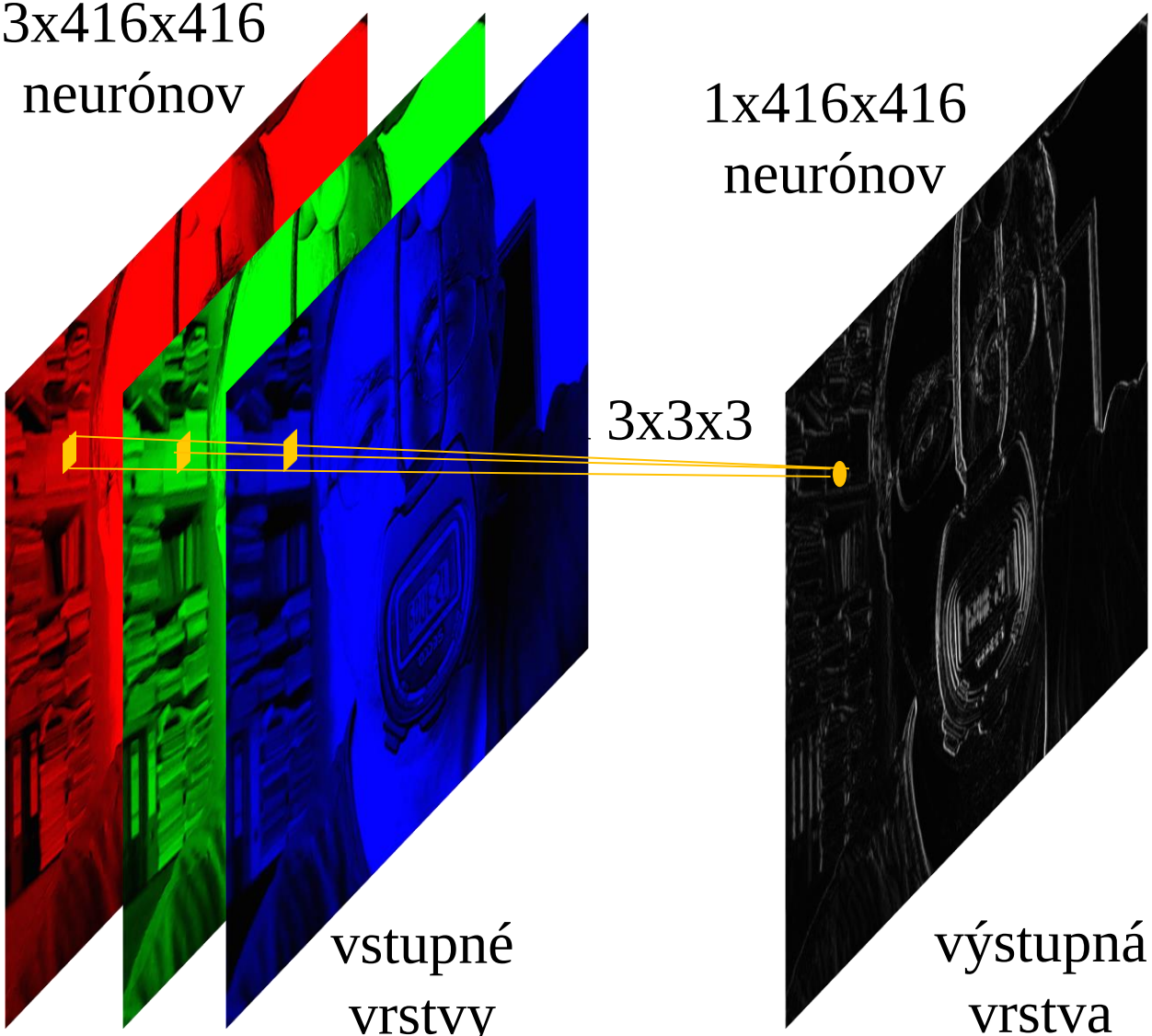
Spracovanie obrazu pomocou kernelov

3x3x3

-1/3	0	1/3
-2/3	0	2/3
-1/3	0	1/3

Napríklad: trojkanálový farebný obraz môžeme Sobelovým kernelom premeniť na zvislé hrany

Konvolučná vrstva

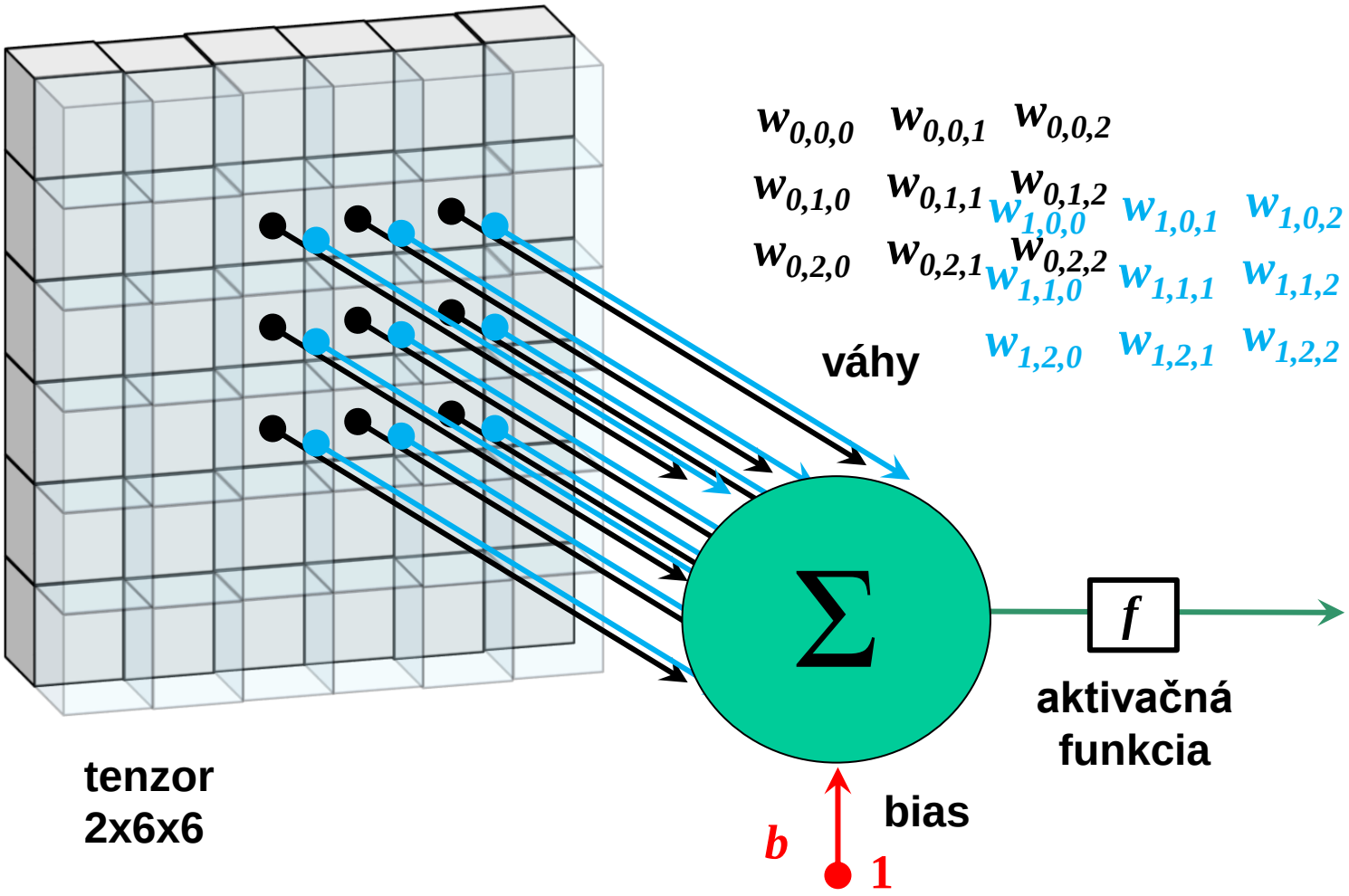


Každý neurón výstupnej vrstvy má spojenia na 3x3x3 = 27 neurónov vstupnej vrstvy

Váhy na spojeniach neuróny zdieľajú, t.j. celá vrstva má 173056 neurónov, ale len 27 parametrov, prípadne 28 s biasom

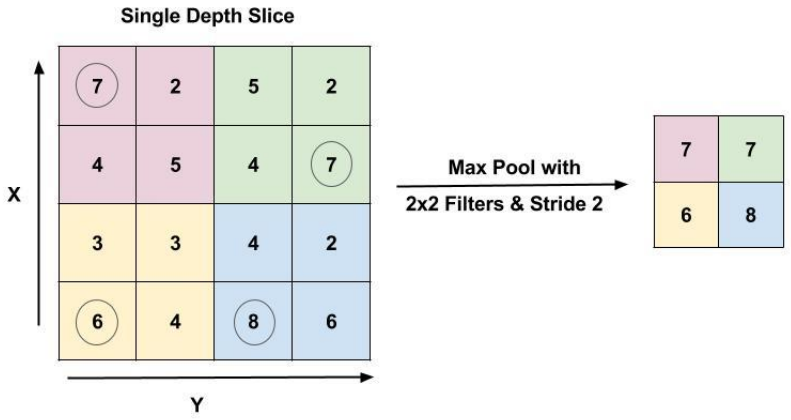
[1998 LeCun, Bottou, Bengio & Haffner]

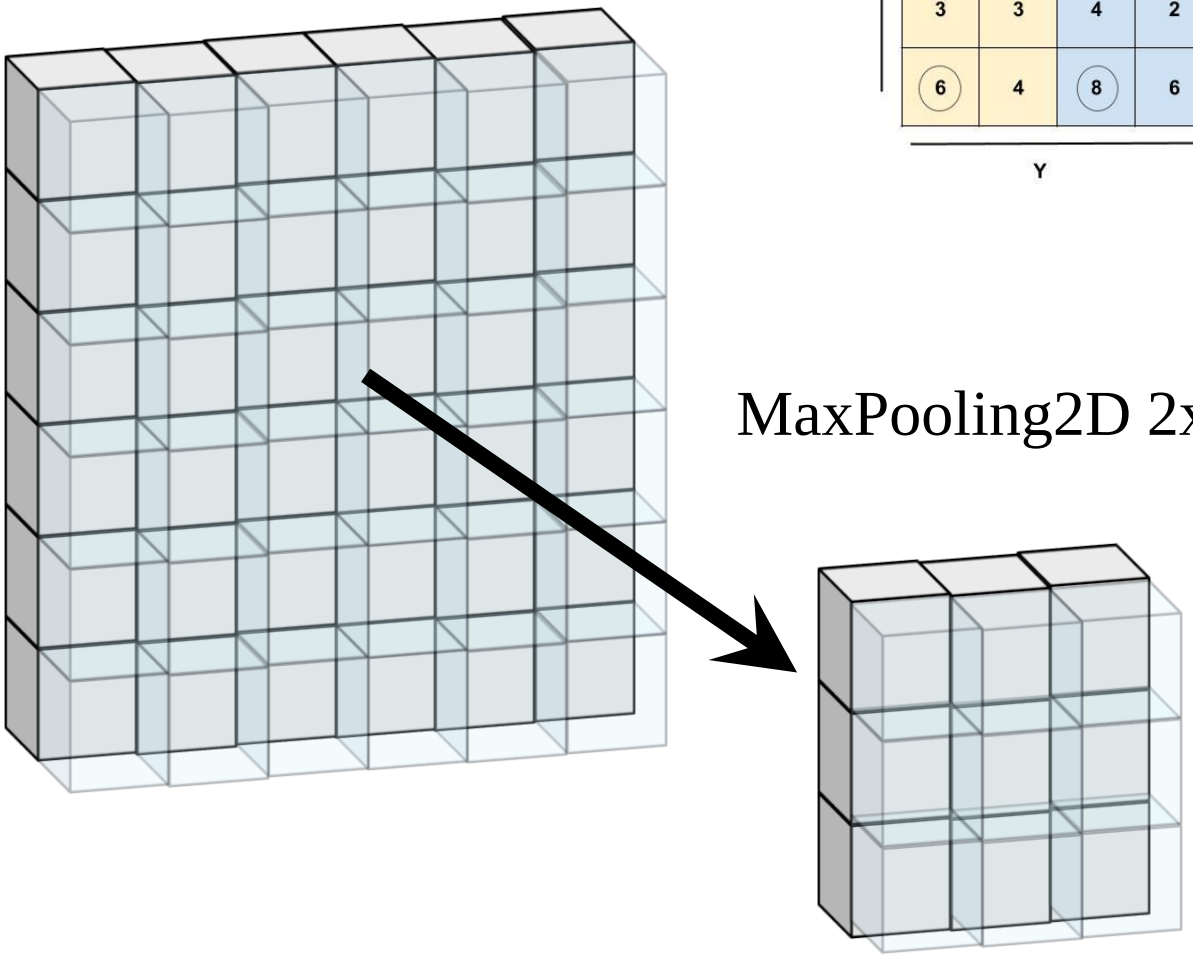
Kernel



Typické rozmery kernelov: 1x3x3, 3x3x3, 64x3x3, 32x5x5, 64x1x1

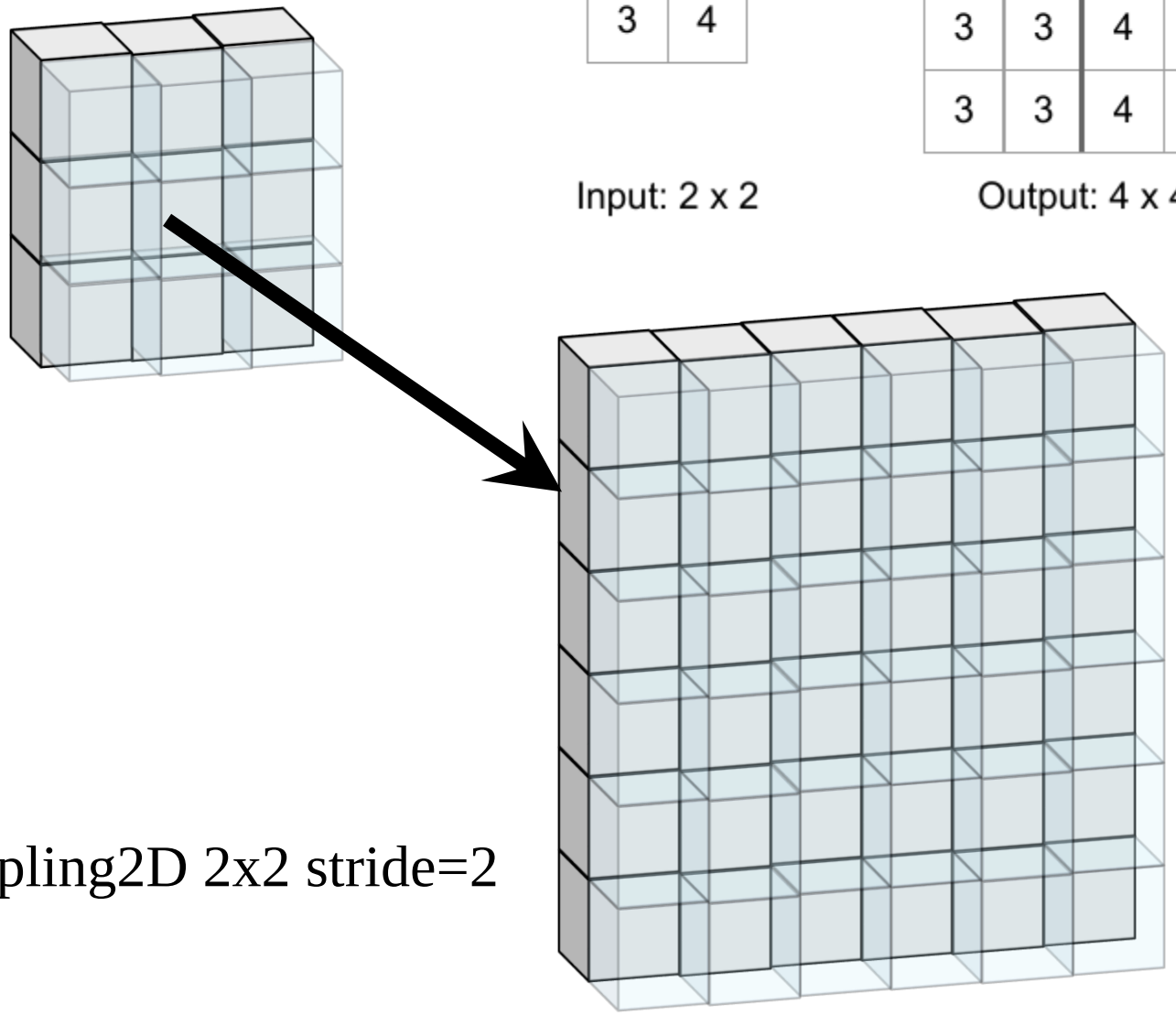
Max Pooling





MaxPooling2D 2x2 stride=2

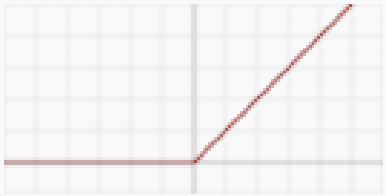
Up Sampling



UpSampling2D 2x2 stride=2

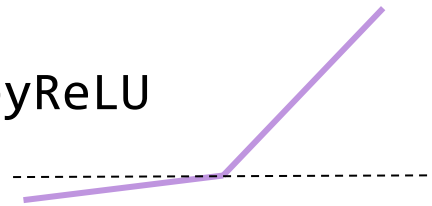
Aktivačné funkcie

ReLU

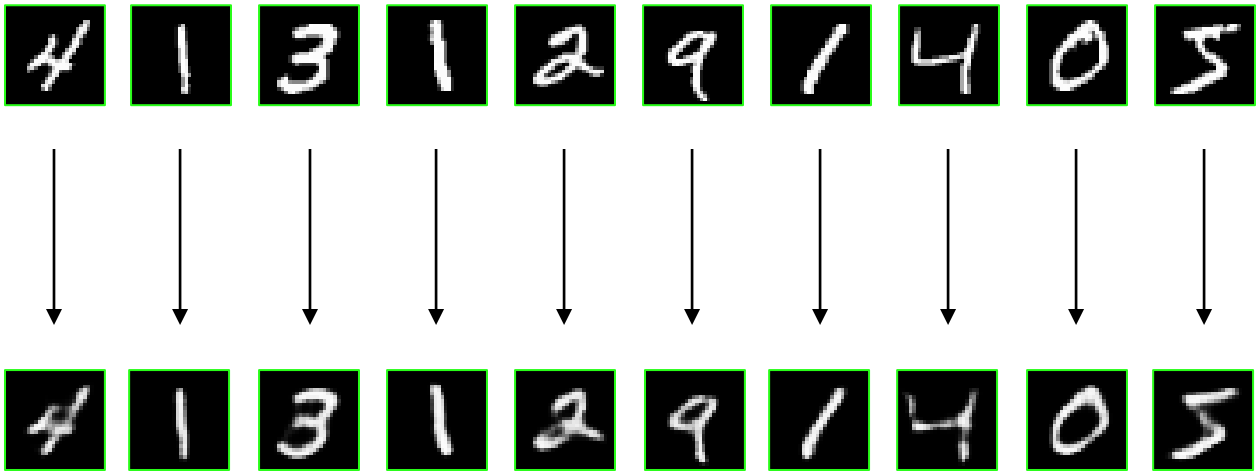


$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

LeakeyReLU



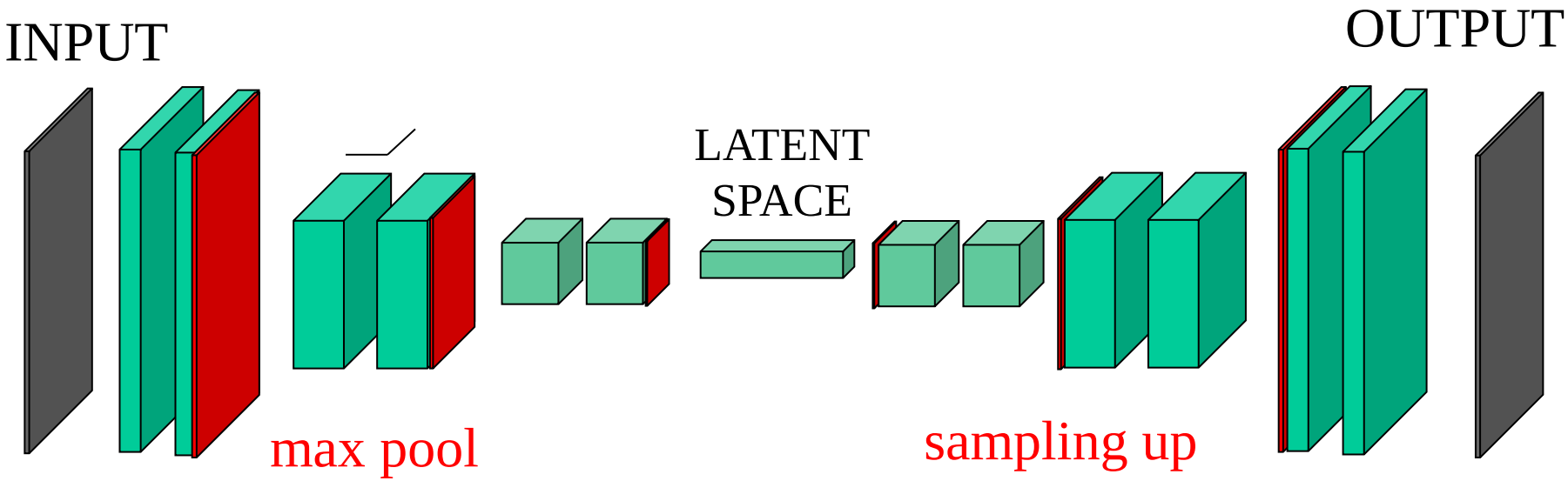
Autoencoder



Dataset MNIST

[1987 Ballard]

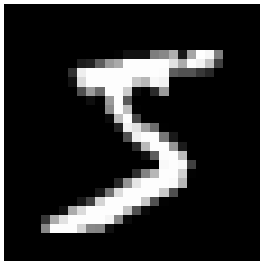
(Konvolučný) autoencoder



[2005 Morell] [2009 Bengio] [2011 Hinton Krizhevsky]

Autoencoder

VSTUP

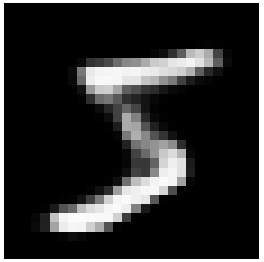


LATENTNÝ PRIESTOR

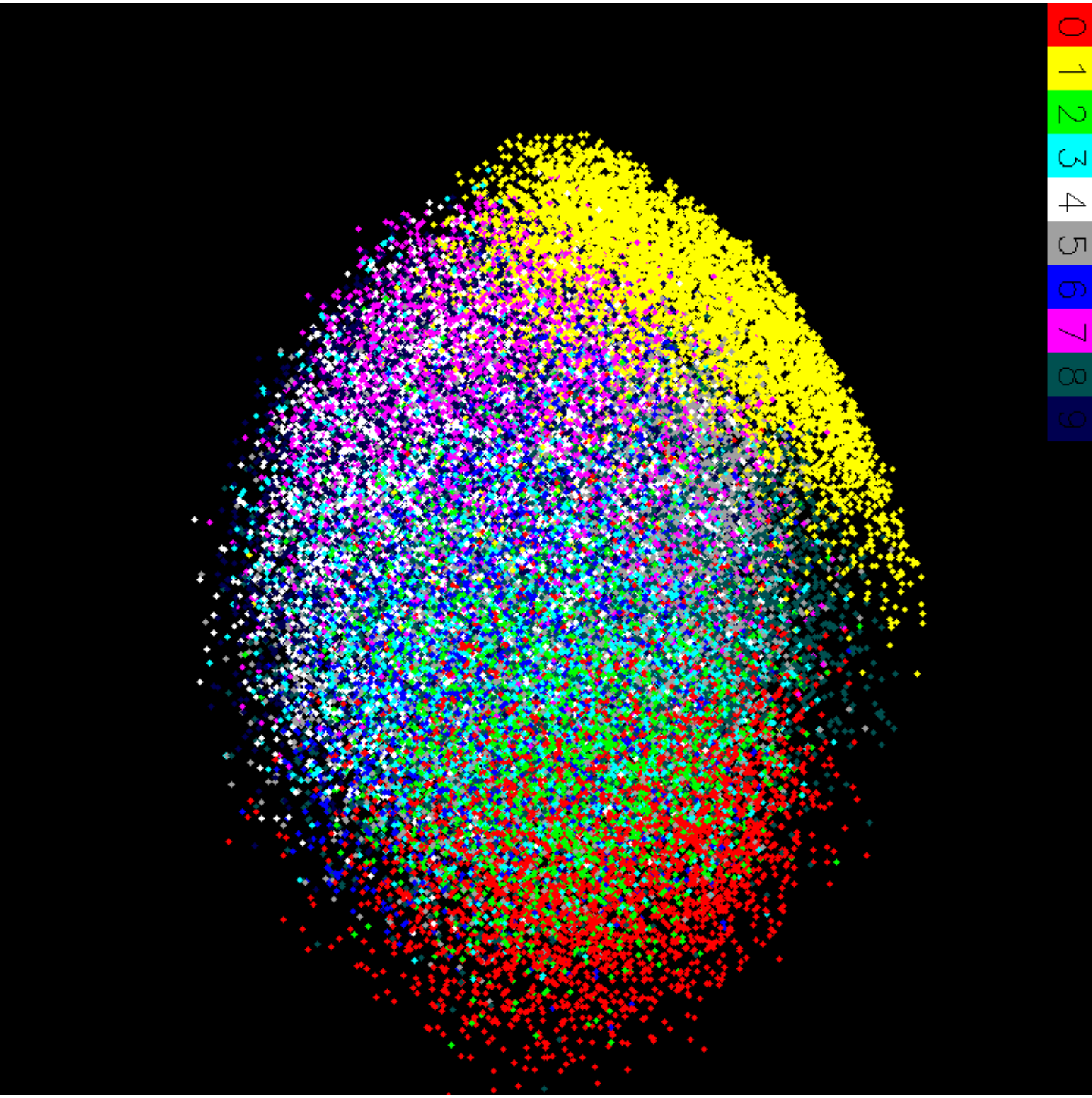
```
array([0.11400187, 0.47539777, 0.19979003, 0.03328802, 0.29702646,
       0.6671412 , 0.77884567, 0.98671937, 0.        , 0.7643514 ,
       0.37023163, 1.1198556 , 0.9364344 , 0.27932528, 1.3840352 ,
       1.1025171 , 0.        , 1.127322  , 0.6720804 , 1.0088537 ,
       0.91483283, 0.        , 1.3498058 , 0.8023532 , 0.13444054,
       1.1076059 , 0.805897  , 0.4363817 , 0.4396257 , 0.        ,
       0.5863744 , 0.41566792, 0.22631842, 0.20689899, 0.28988916,
       0.19635512, 0.92697734, 0.8330982 , 0.8810159 , 0.14809921,
       0.        , 0.20924592, 0.        , 2.7036035 , 1.7514778 ,
       0.84079874, 1.6247051 , 0.        , 0.25012553, 0.70244396,
       0.6414403 , 2.4568624 , 1.4004446 , 0.        , 1.3100656 ,
       0.        , 0.5075829 , 0.68791926, 0.65375787, 0.82646245,
       0.4355916 , 0.        , 0.19816275, 0.        , 0.        ,
       0.4575198 , 0.18170735, 0.12635085, 0.17334037, 0.4582858 ,
       1.0187354 , 0.75260663, 0.        , 0.4846586 , 0.        ,
       1.9696081 , 1.12253   , 0.8872602 , 1.3111267 , 0.        ,
       0.        , 1.1081457 , 0.5976082 , 2.1433632 , 1.2630261 ,
       0.        , 1.4435332 , 0.        , 0.43087044, 0.50078976,
       0.85700417, 0.23156954, 0.3238153 , 0.19322284, 0.23595949,
       0.        , 0.        , 0.11734977, 0.        , 0.8126336 ,
       1.2869604 , 0.65106845, 1.012244  , 0.        , 0.07893795,
       0.16735056, 0.15048887, 2.1369095 , 1.2226689 , 0.        ,
       1.0082622 , 0.        , 0.9999559 , 0.35816067, 0.4425221 ,
       1.7548463 , 0.36668733, 0.25854337, 0.35278222, 0.        ,
       0.7470093 , 0.42634767, 0.5120847 , 0.24160625, 0.23943251,
       0.61216664, 0.171287  , 0.35395604], dtype=float32)
```

vektor príznakov

VÝSTUP

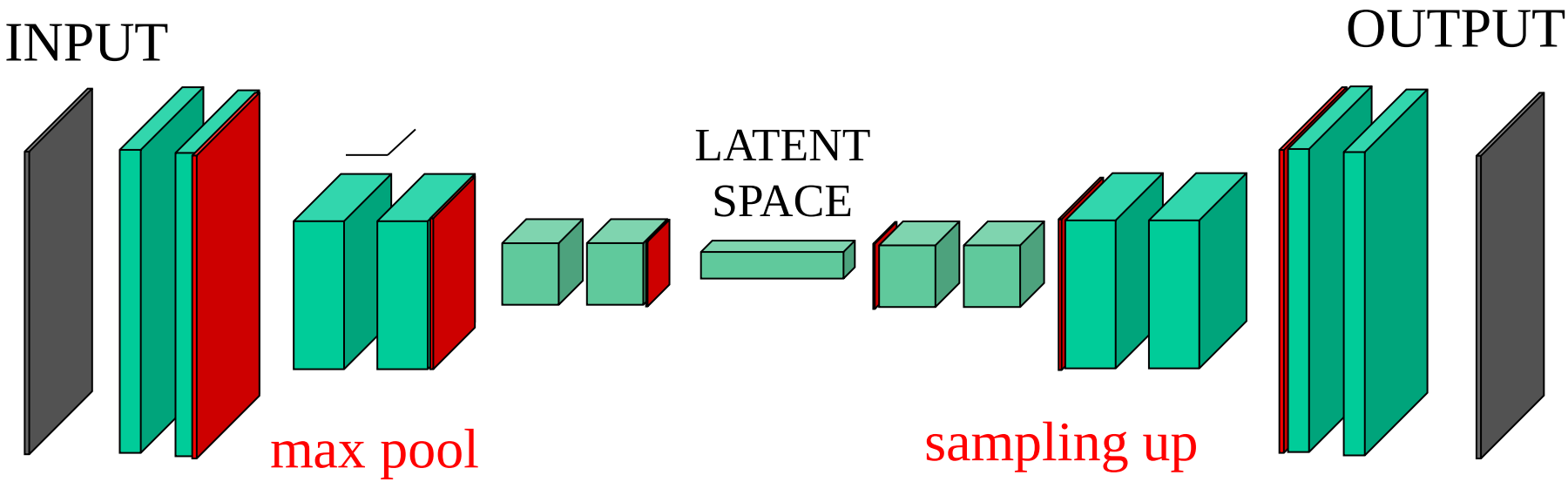


LATENTNÝ PRIESTOR



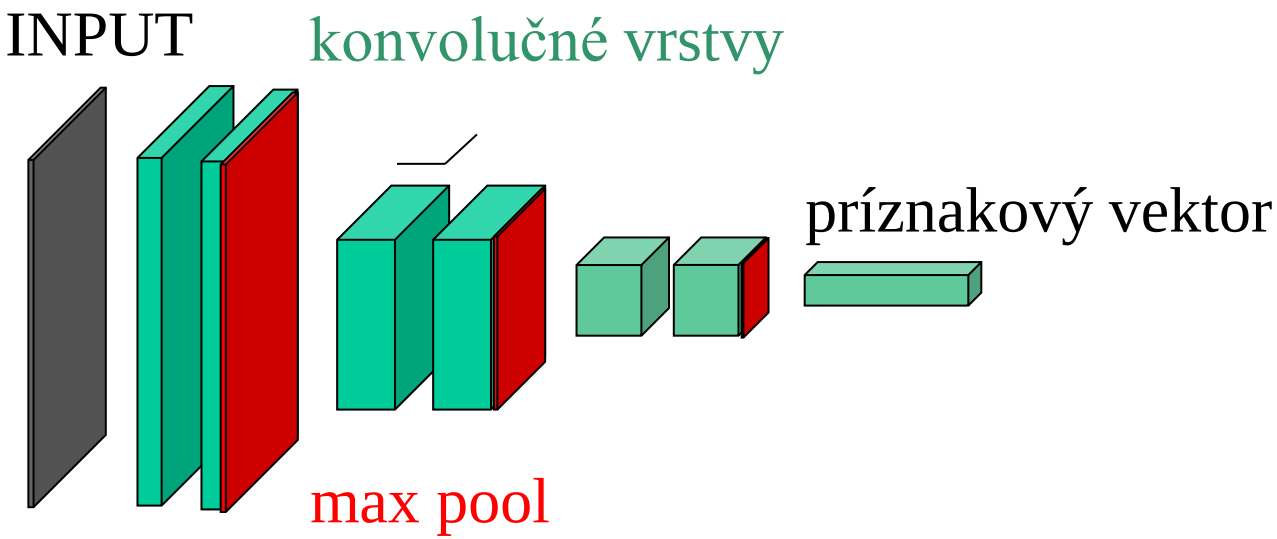
Dataset MNIST

Autoencoder

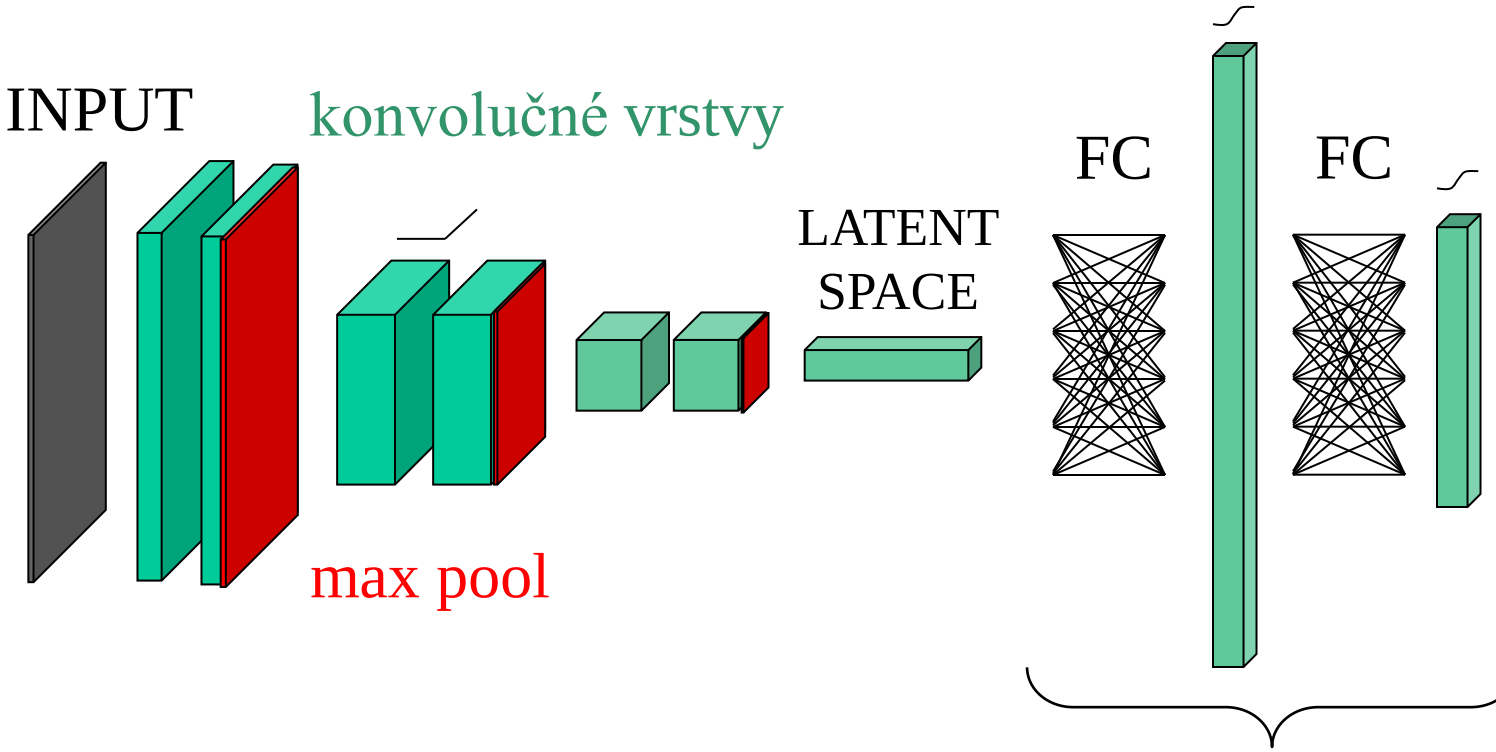


Po (úspešnom) natrénovaní možno autoencoder rozdeliť

Encoder



Encoder dokáže transformovať obraz do príznakov oveľa menšej dimenzie ...



... a s nimi perceptron dokáže fungovať aj z praktického hľadiska

GPU

NVIDIA

NVIDIA
CUDA

[2009 Ranja, Madhavan & Andrew Ng]

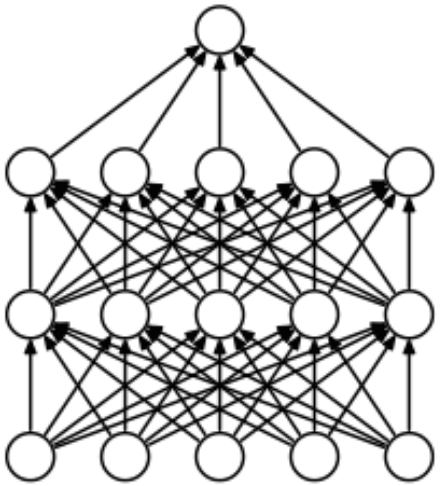
Softmax

softmax

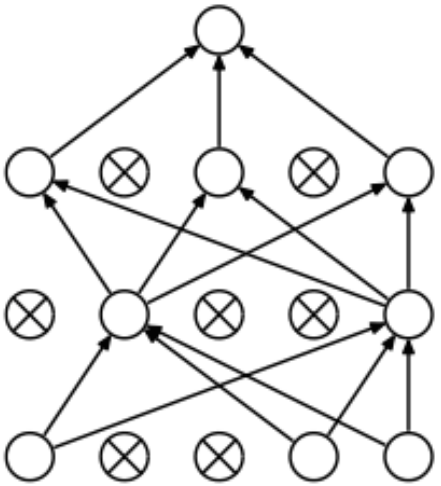
$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

- Podobne ako sigmoida zabezpečuje rozsah hodnôt 0-1
- Softmax navyše zabezpečuje aby ich súčet bol 1
- Dajú sa potom interpretovať ako pravdepodobnosti

Dropout



Standard Neural Net

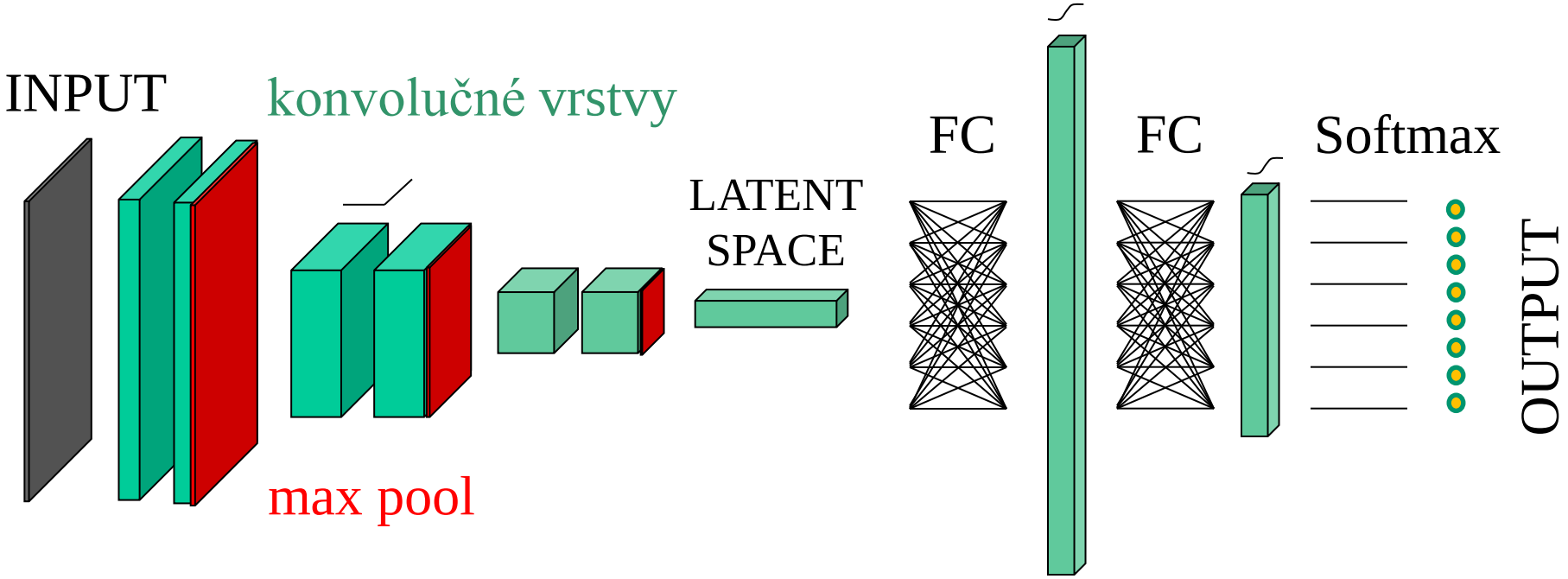


After applying dropout.

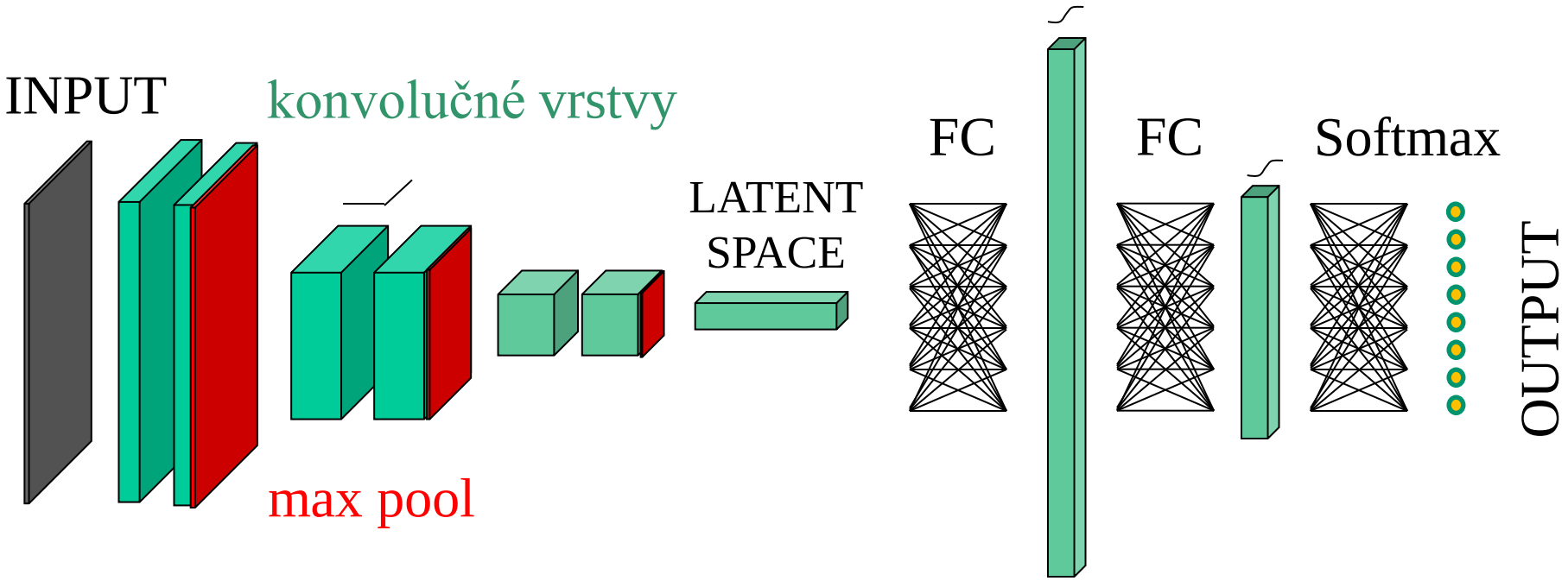
- Priveľká hĺbka siete hrozí problémom miznúceho gradientu (ak pri spätnej propagácii gradient klesne na nulu, úvodné vrstvy sa už nijako nezmodifikujú)
- Jedno z riešení je Dropout – povkladanie vrstiev, ktoré pri trénovaní náhodne miesto vstupu pustia na výstup neurónu nulu. V dôsledku toho sa medzi neuróny lepšie rozloží zodpovednosť za výsledok

[2012 Krizhevsky, Sutskever & Hinton]

Klasifikátor

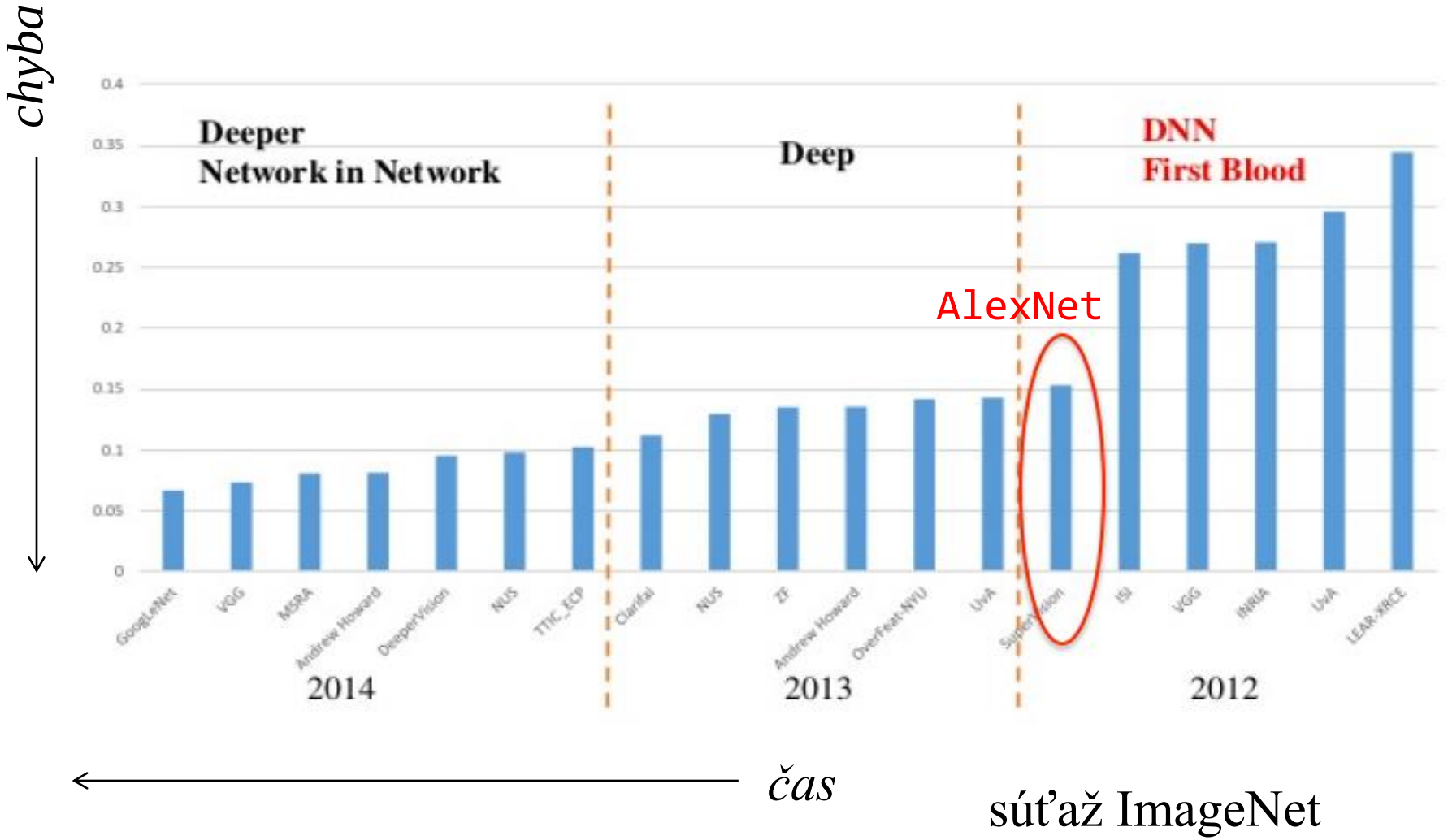


Klasifikátor



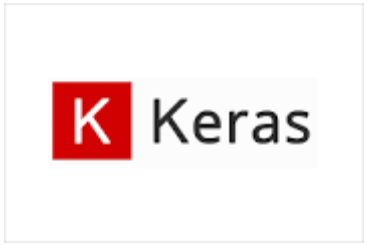
[2012 Krizhevsky, Sutskever & Hinton]

Deep Learning: Boom od roku 2012

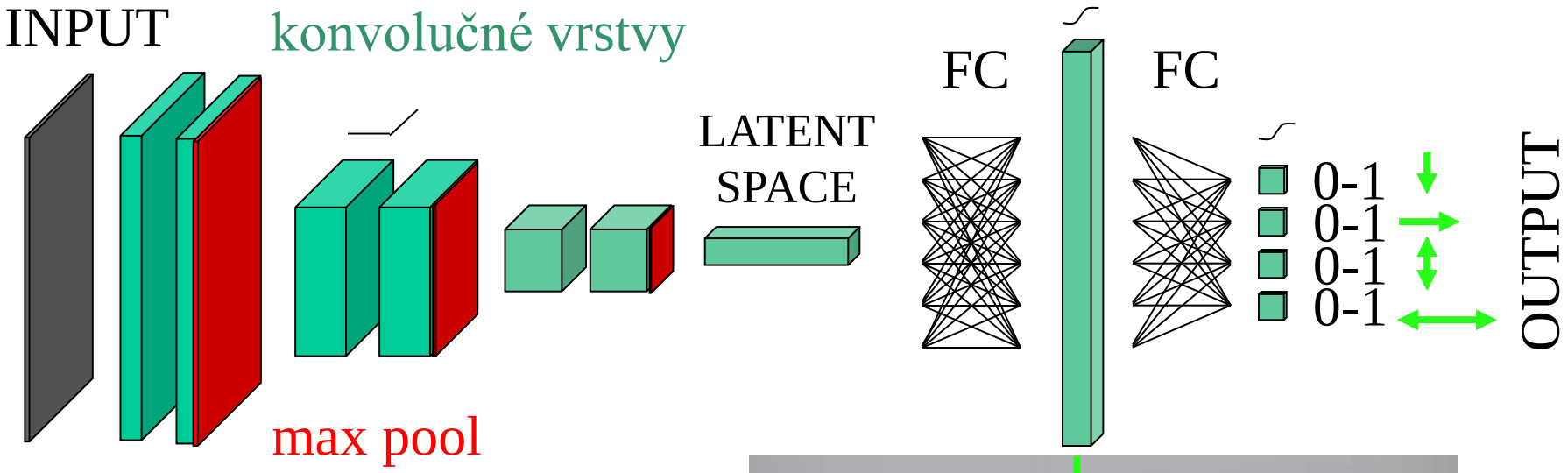


súťaž ImageNet

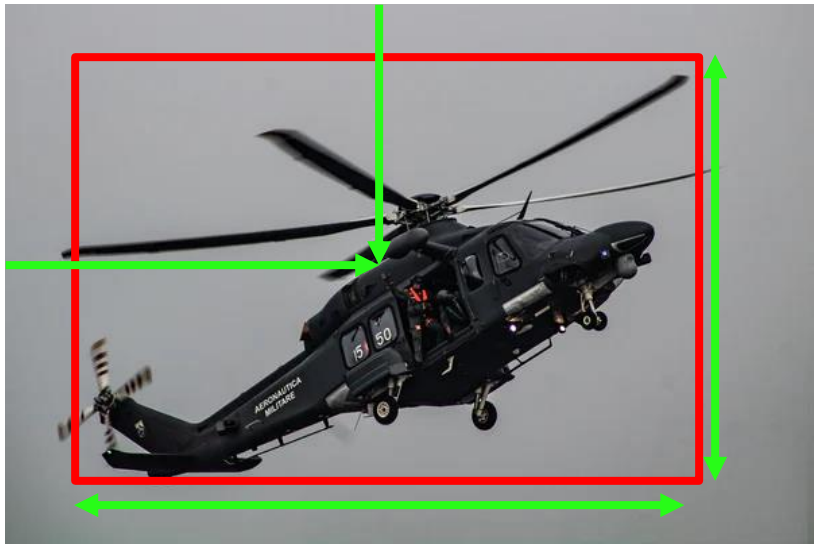
Softwarové nástroje DL



Nájdenie jedného objektu



Čo keď potrebujeme výskytov objektov viac?

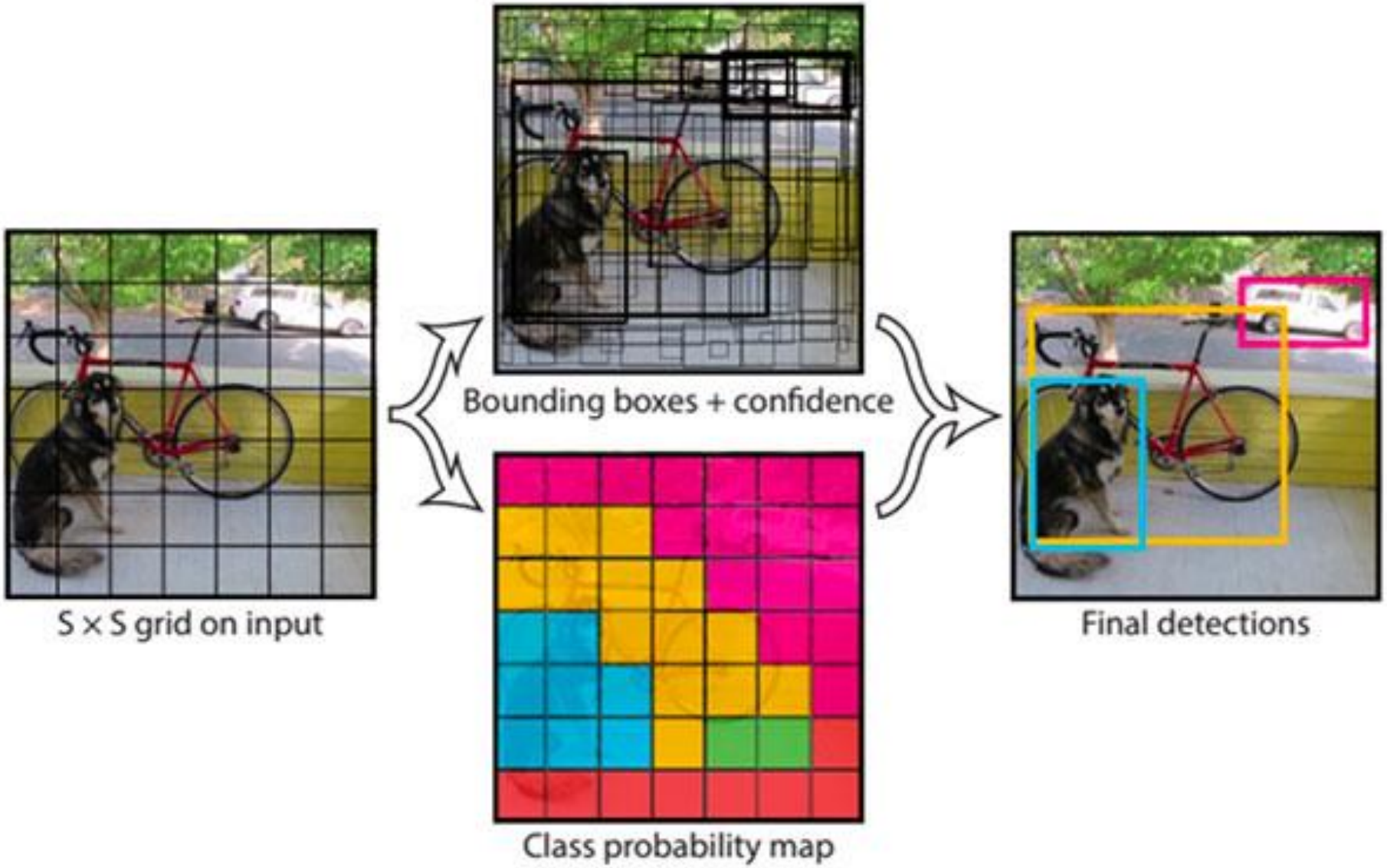


YOLO Detektor

- You Look Only Once
- Obrázok pokryjeme štvorcovými neprekrývajúcimi sa regiónmi
- Pre každý región klasifikujeme pravdepodobnosti jednotlivých typov objektov
- Paralelne k tomu robíme pre každý región regresiu dvoch obdĺžnikov ohraničujúcich objekt, ktorého by mohol byť región súčasťou
- Výsledky klasifikácie a regresie zosumarizujeme (non-maximum supression)

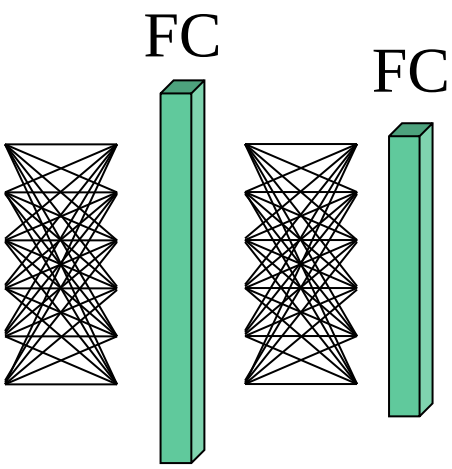
[2016 Redmon] [2018 Redmon & Farhadi]

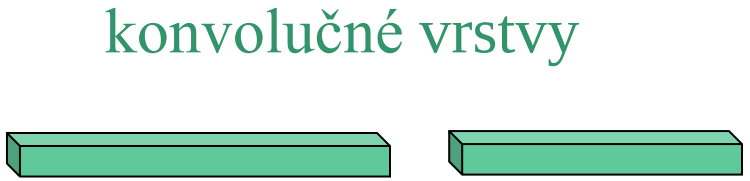
regresor, ktorý vracia parametre obdĺžnika, ktorý sa snaží celý objekt obsiahnuť



klasifikátor, ktorý vráti pravdepodobnosti objektov, ktoré sa tam nachádzajú

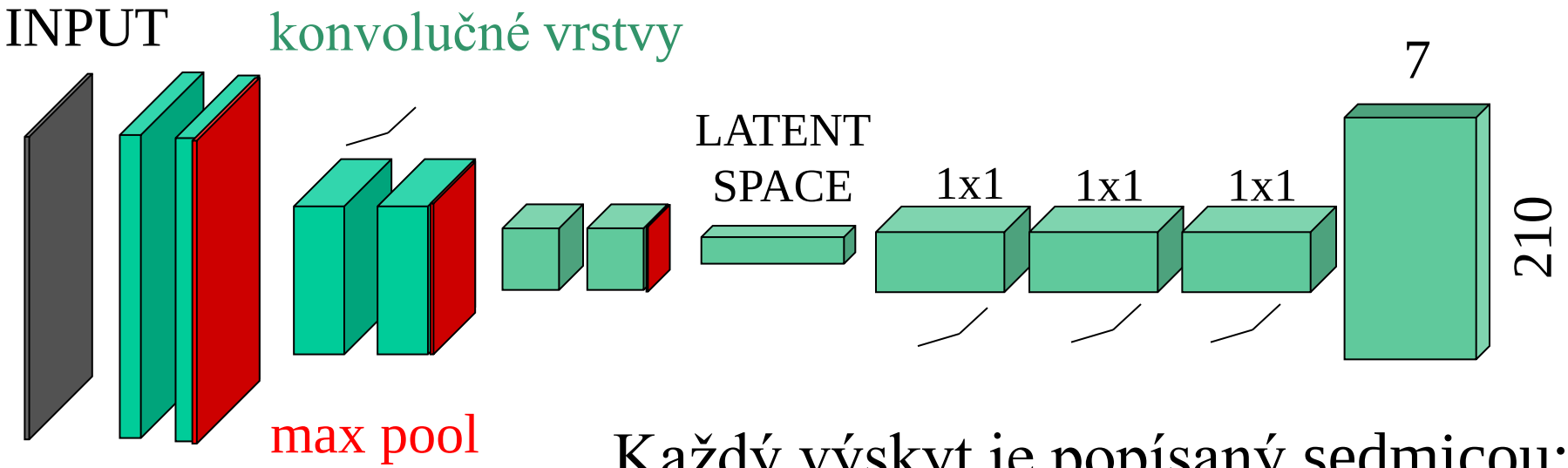
Ako pustíme viacej perceptronov naraz?





- Perceptron spracúvajúci vstup o veľkosti 128 z dvoch plne prepojených vrstiev s 512 a 256 neurónmi možno zrealizovať ako dva bloky konvolučných vrstiev s kernelom 1x1, jedna hlboká 512 vrstiev, druhá 256 vrstiev, spracúvajúcich vstup o veľkosti 512 x 1 x 1
- Keď potom do toho vložíme vstup 512 x 3 x 4 tak nám to pustí vlastne 12 perceptronov paralelne

YOLO v1

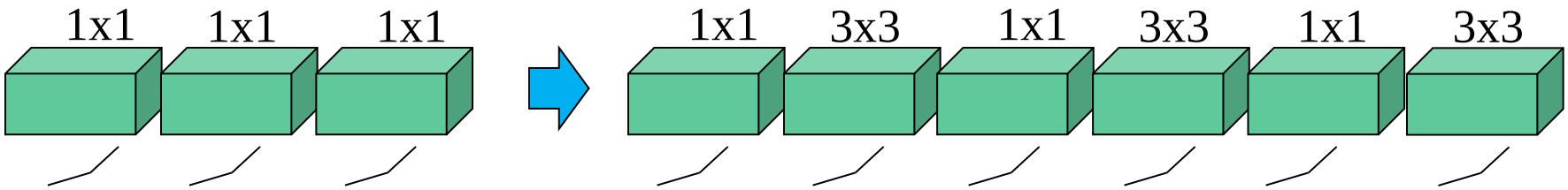


Každý výskyt je popísaný sedmicou:
- či sa výskyt berie v úvahu
- aký je to objekt (kategória 0-...)
- nakoľko je detekcia spoľahlivá
- pozícia obdĺžnika a jeho veľkosť

Dokážeme zdetekovať maximálne 210 objektov

Ako vezmeme v úvahu vplyv okolitých regiónov?

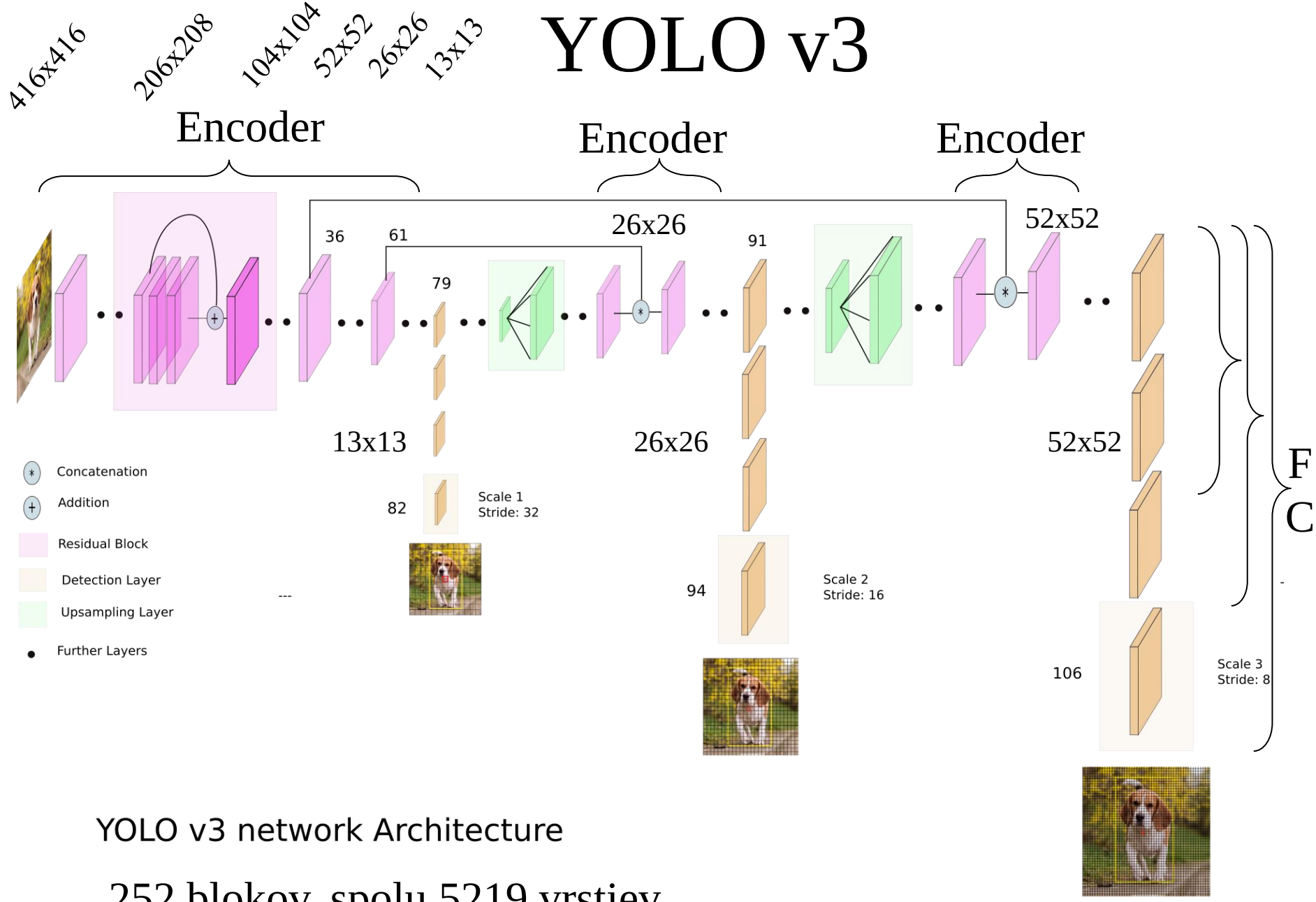
- ak v okolitých regiónoch vidíme hodinky, tak v našom regióne budú asi hodinky
- preto prekladáme konvolúcie 1x1 s konvolúciami 3x3



Ako riešiť, že objekty sú rôzne veľké?

- Skúsime tri rôzne veľkosti mriežky ktorou obraz pokrývame.
- Príznakový vektor jemnejších odvodíme od:
  - príznakového vektora hrubších
  - čiastočného výsledku detekcie v hrubších

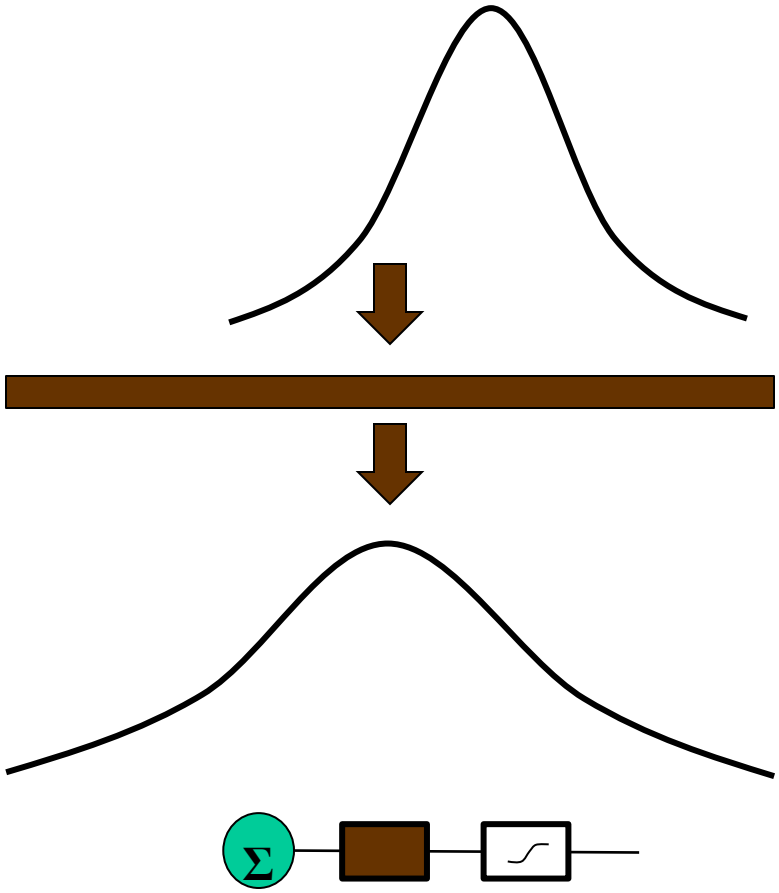
YOLO v3



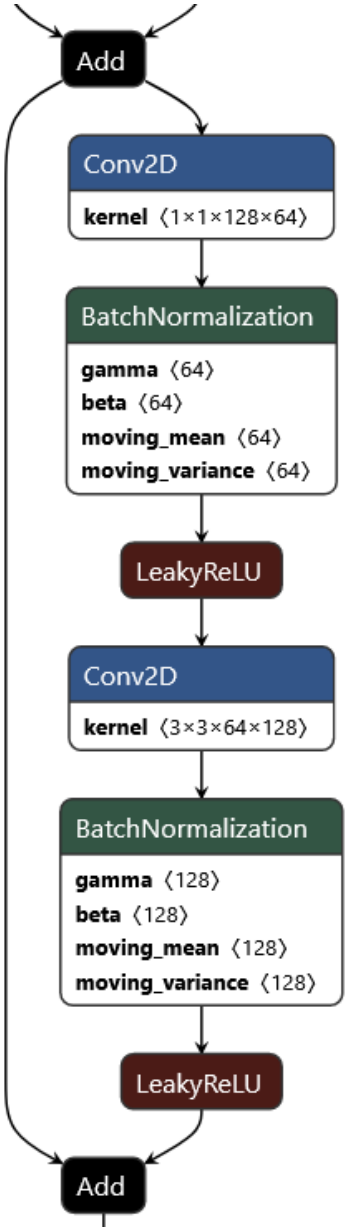
YOLO v3 network Architecture

252 blokov, spolu 5219 vrstiev

Batch Normalization



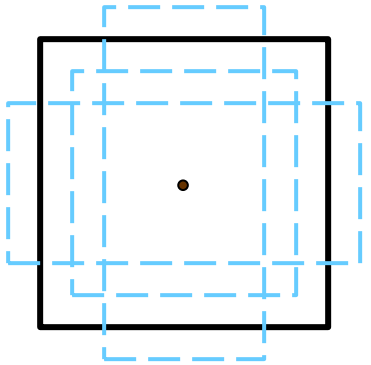
Residuals



Výstup z YOLO detektora

- 13x13x255, 26x26x255, 52x52x255
- Každý paralelne pôsobiaci perceptrón aproximuje tri detekcie 255 = 3 x 85
- detekcia 85 = 4+1+80
  - 4 … stred-x, stred-y, šírka, výška
  - 1 … dôveryhodnosť 0-1
  - 80 … pravdepodobnosti príslušnosti k jednotlivým kategóriám

Kotvy anchors



- Každá detekcia definuje obdĺžnik vyjadrený ako násobok voči tzv. kotvám (anchors), čo sú obdĺžniky určitej konkrétnej veľkosti položené na miesto, kde sa nachádza
- Kotvy sú nasledovné:
  [116x90, 156x198, 373x326] (výstup 13x13)
  [30x61, 62x45, 59x119] (výstup 26x26)
  [10x13, 16x30, 33x23] (výstup 13x13)

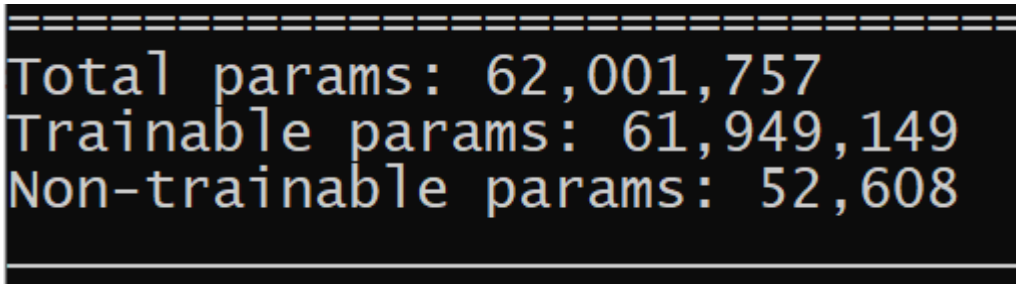
Predtrénovaný model

Trénuje sa z datasetu COCO, ktorý obsahuje 80 kategórii:
person, bicycle, car, motorbike, aeroplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports, ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush

Pri použití sa pretrénuje na kategórie užívateľa (transfer learning)

Trénovanie YOLO v3

- DarkNet, CUDA, GPU



- 3 hodiny (pre normálne veľký dataset by to trvalo 3 dni)

Použitie YOLO v3

- OpenCV 4.3, CUDA

Výsledok detekcie

Ďakujem za pozornosť !

https://www.learnopencv.com
https://towardsdatascience.com
https://www.kaggle.com

https://github.com/andylucny/YOLOtools
https://www.agentspace.org/download/darknet.zip