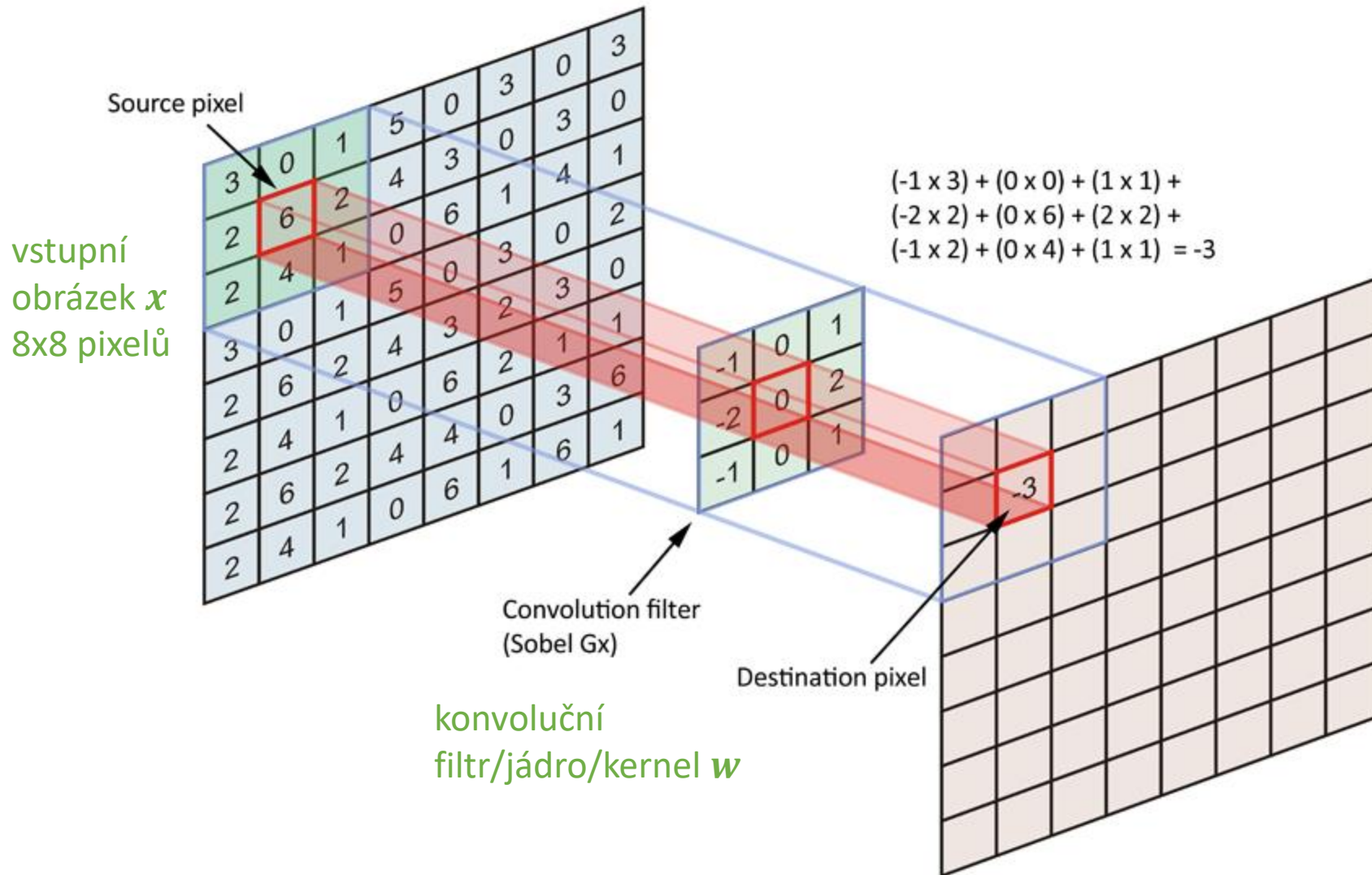


Aplikace neuronových sítí

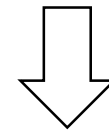
Konvoluční sítě

Dvourozměrná konvoluce



Pro každý výstupní pixel:

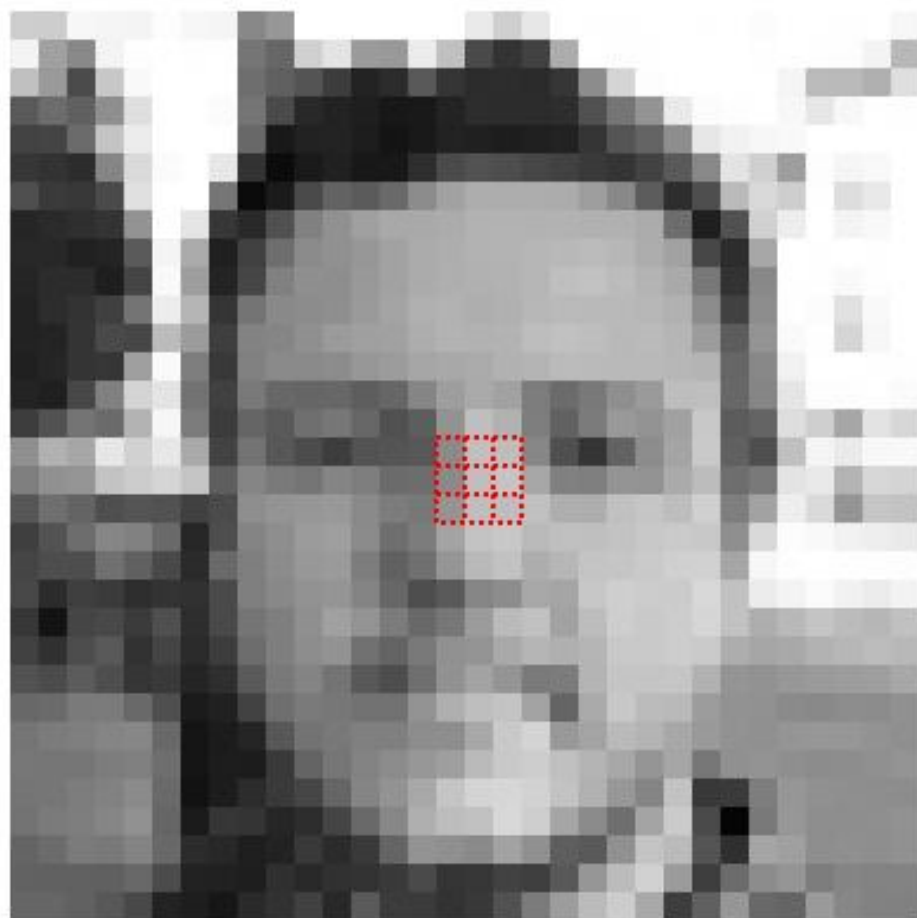
$$z_{uv} = \sum_{i=-k}^{+k} \sum_{j=-k}^{+k} w_{ij} x_{u+i, v+j} + b$$



$$z_{uv} = \mathbf{w}^T \mathbf{x}_{uv} + b$$

výstupní obrázek z
velikost závisí
na nastavení konvoluce

Demo



input image

$$\left(\begin{array}{ccc} 139 & + & 192 & + & 190 \\ \times 0.0625 & \times 0.125 & \times 0.0625 \end{array} \right.$$

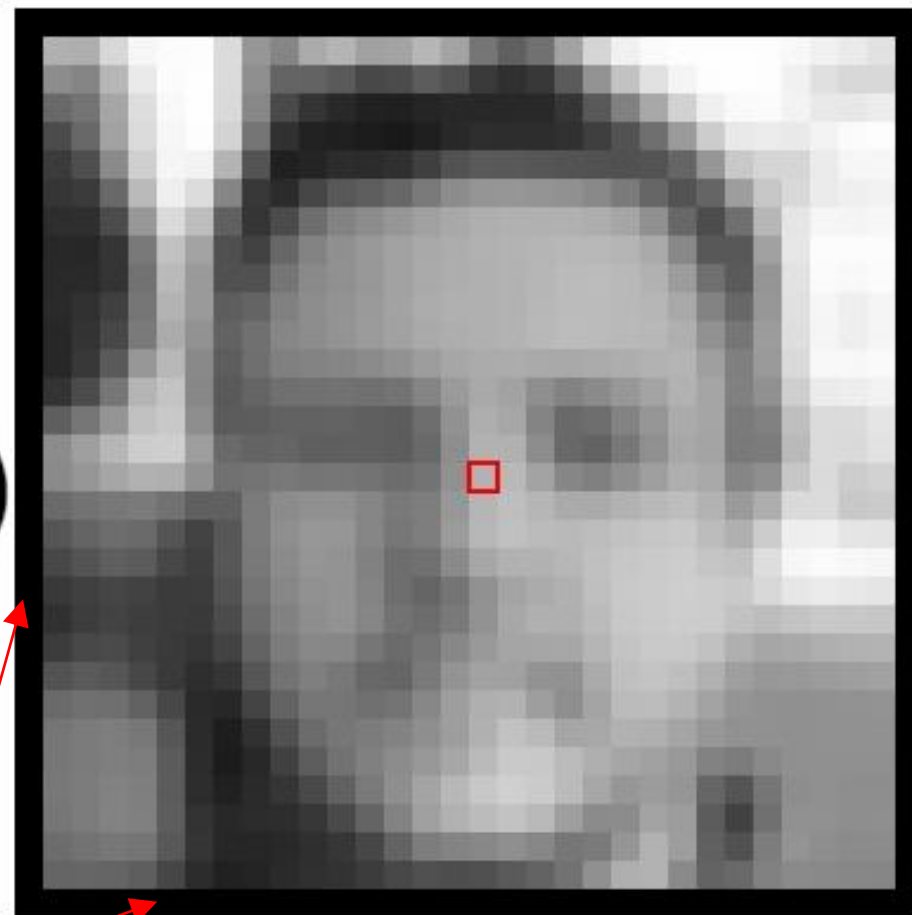
$$+ \begin{array}{ccc} 139 & + & 191 & + & 197 \\ \times 0.125 & \times 0.25 & \times 0.125 \end{array}$$

$$+ \begin{array}{ccc} 149 & + & 191 & + & 190 \\ \times 0.0625 & \times 0.125 & \times 0.0625 \end{array} \right)$$

$$= \begin{array}{c} 179 \end{array}$$

kernel:

blur ▼



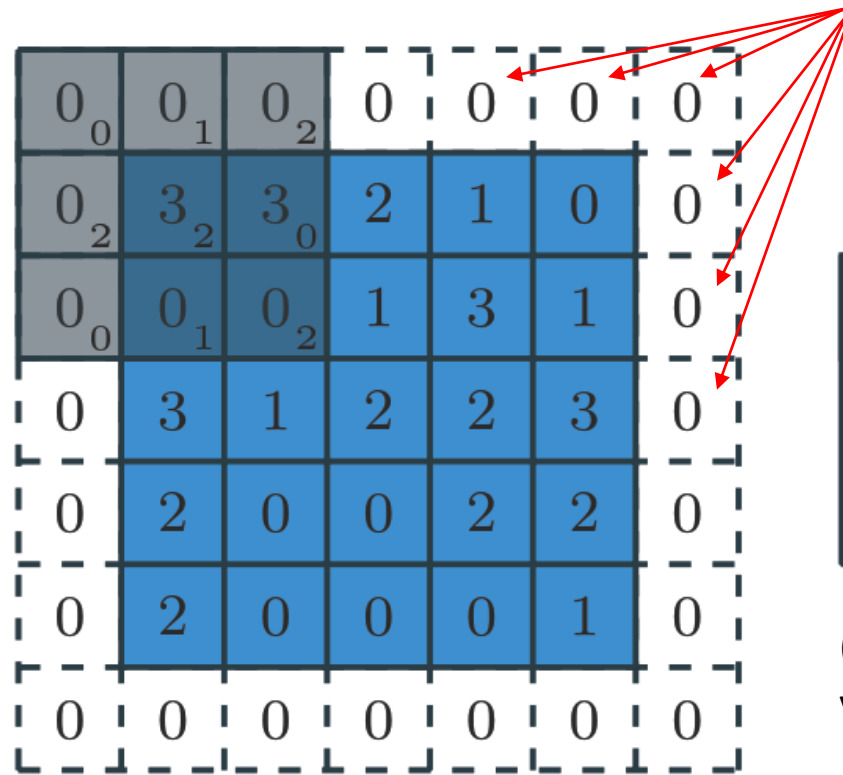
output image

okraje?

Odkaz: <http://setosa.io/ev/image-kernels/>

Padding

Jak se vypořádat s okraji?



Obrázek např. "nastavíme" nulami
→ tzv. zero padding

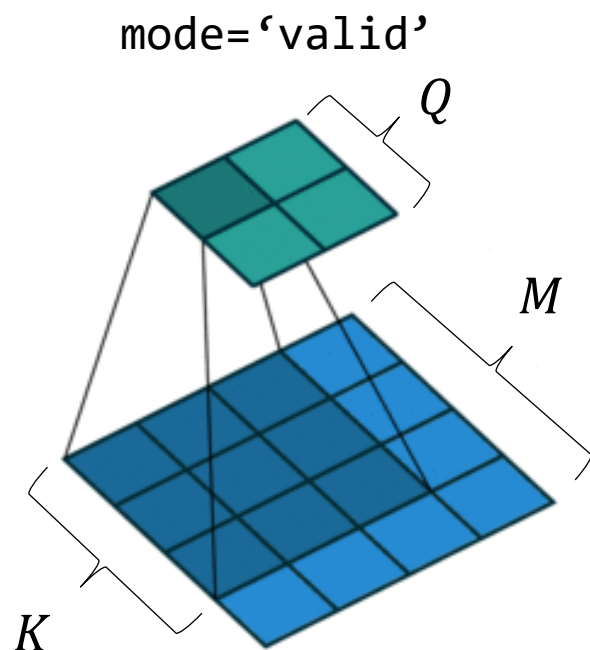
(výsledek je 3x3, protože
v příkladu stride=2)

Existují další způsoby: const, wrap, symm, ...

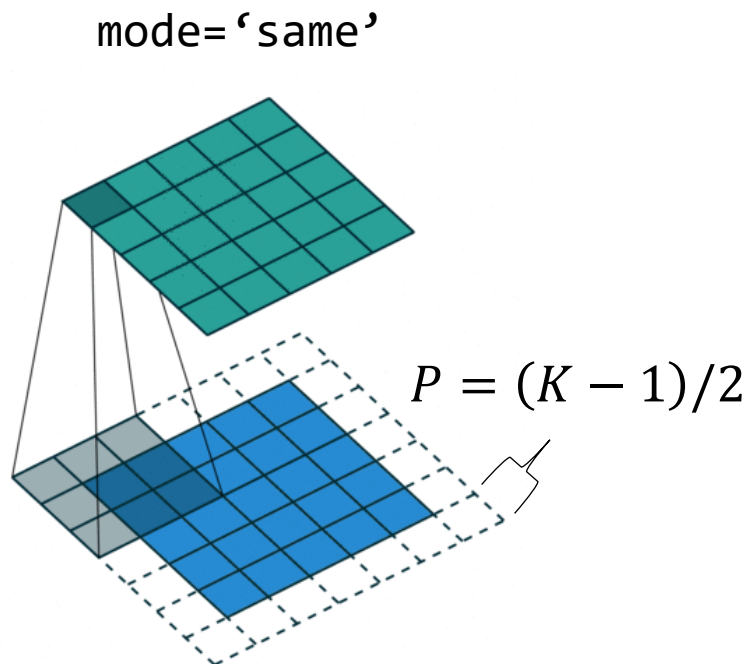
obrázky: https://github.com/vdumoulin/conv_arithmetic

Velikost výstupu v závislosti na paddingu

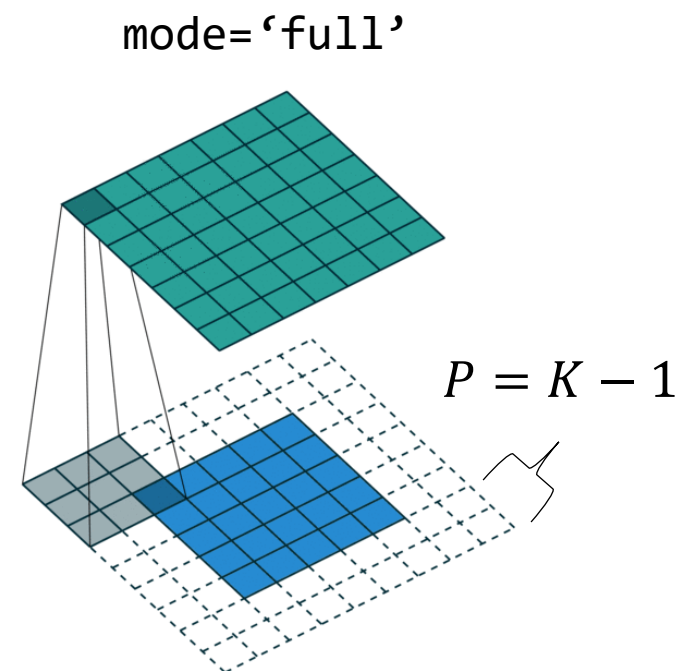
`scipy.signal.convolve2d(x, w, mode, ...)` \Rightarrow výstup $Q \times Q$



$$\begin{aligned} Q &= M - (K - 1) \\ &= 4 - 3 + 1 \\ &= 2 \end{aligned}$$



$$\begin{aligned} Q &= M - (K - 1) + 2P \\ &= M \\ &= 5 \end{aligned}$$

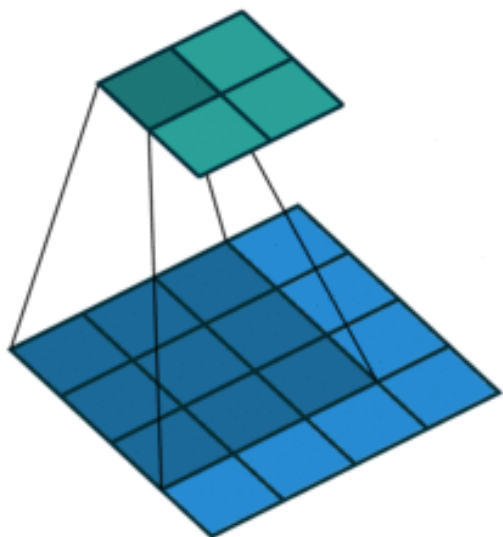


$$\begin{aligned} Q &= M - (K - 1) + 2P \\ &= M + (K - 1) \\ &= 5 + 3 - 1 \\ &= 7 \end{aligned}$$

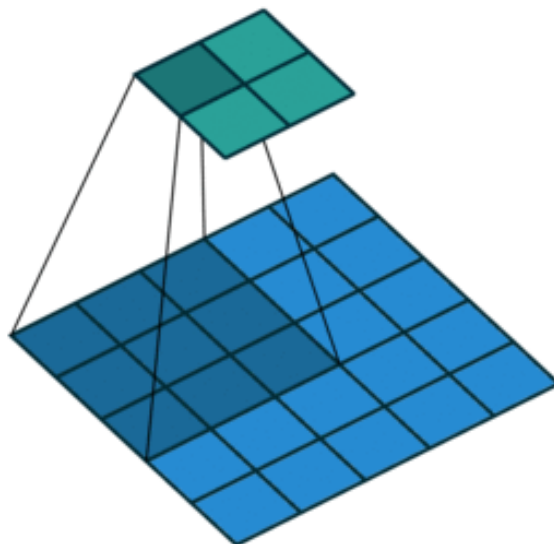
Velikost kroku

V anglické literatuře stride

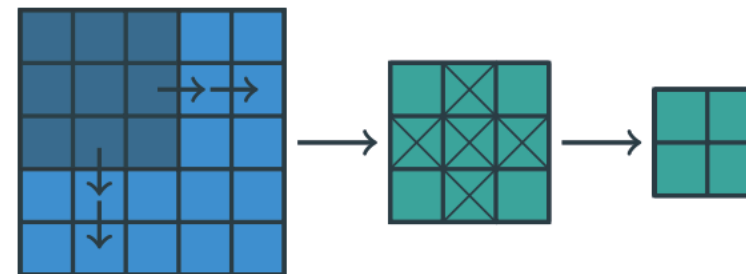
$S = 1$



$S = 2$



$S = 2$ je to samé, jako když $S = 1$,
ale ponecháme pouze každý S -tý výstup:

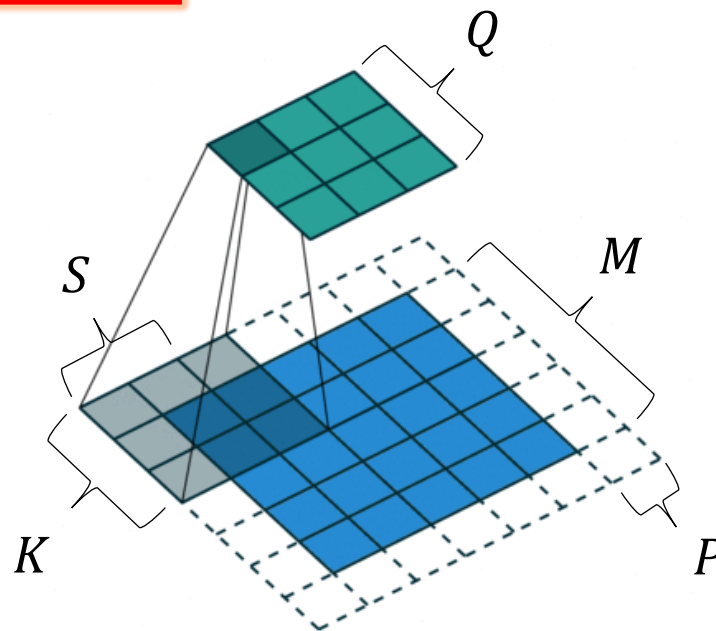
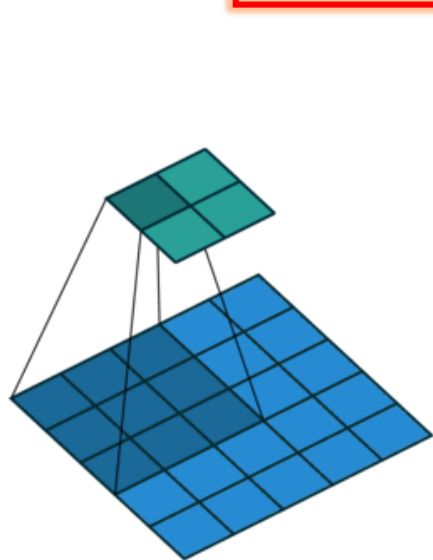


Vidíme tedy, že velikost výstupu
se krokem S dělí

(mezery jsou ve výstupu)

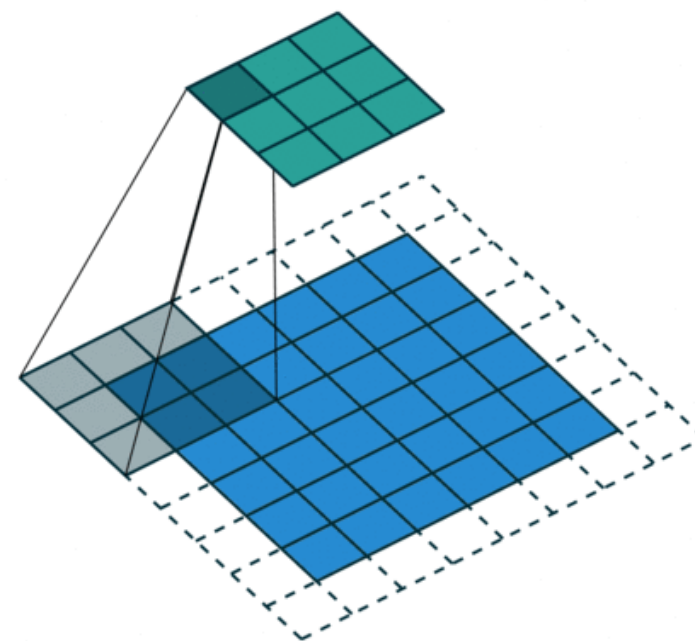
Velikost výstupů konvoluce

$$Q = \left\lfloor \frac{M + 2P - K}{S} \right\rfloor + 1$$



$$Q = \left\lfloor \frac{5 + 2 \cdot 0 - 3}{2} \right\rfloor + 1 = 2$$

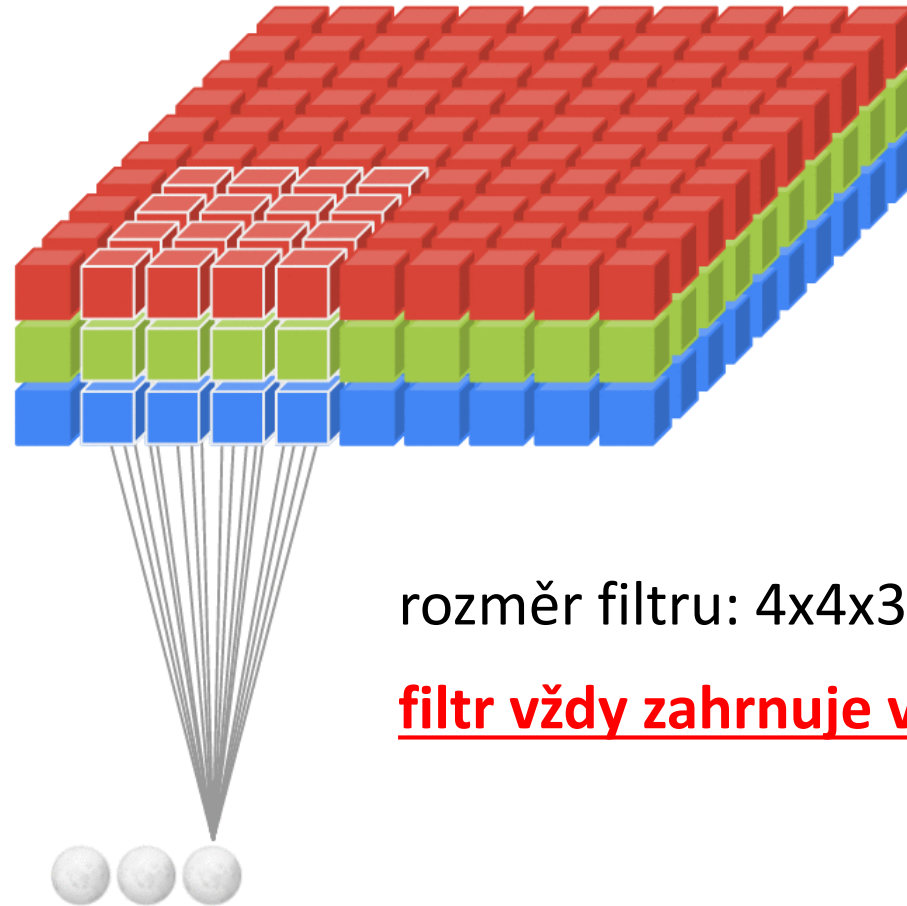
$$Q = \left\lfloor \frac{5 + 2 \cdot 1 - 3}{2} \right\rfloor + 1 = 3$$



$$Q = \left\lfloor \frac{6 + 2 \cdot 1 - 3}{2} \right\rfloor + 1 = 3$$

“Když to nevyjde hezky”:

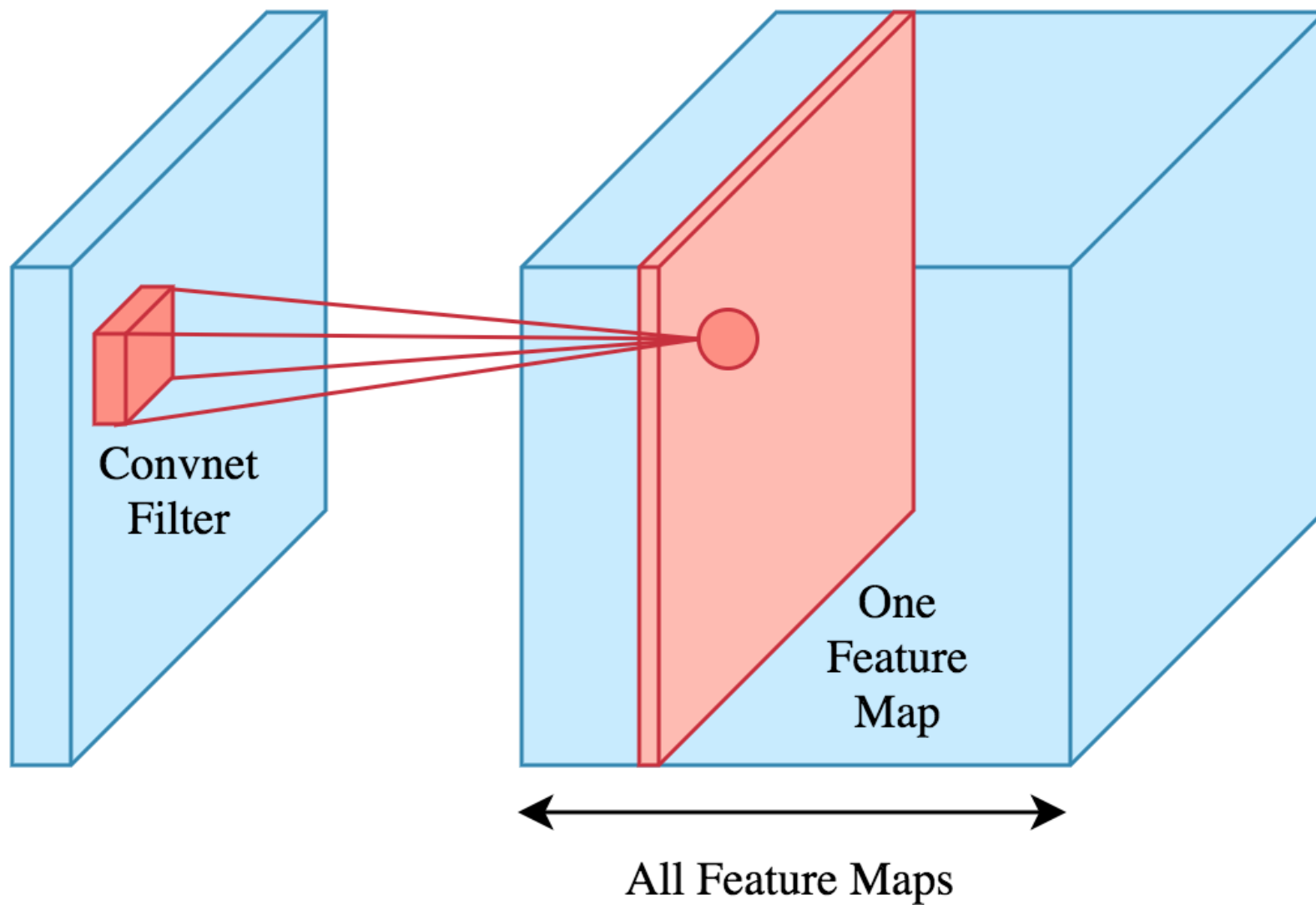
Vícekanálový vstup



rozměr filtru: 4x4x3

filtr vždy zahrnuje všechny vstupní kanály

Více konvolučních filtrů



Konvoluce jako vrstva v neurosíti

Váhy W jsou tensor tvaru

$$K \times K' \times C \times F$$

Bias b je vektor délky

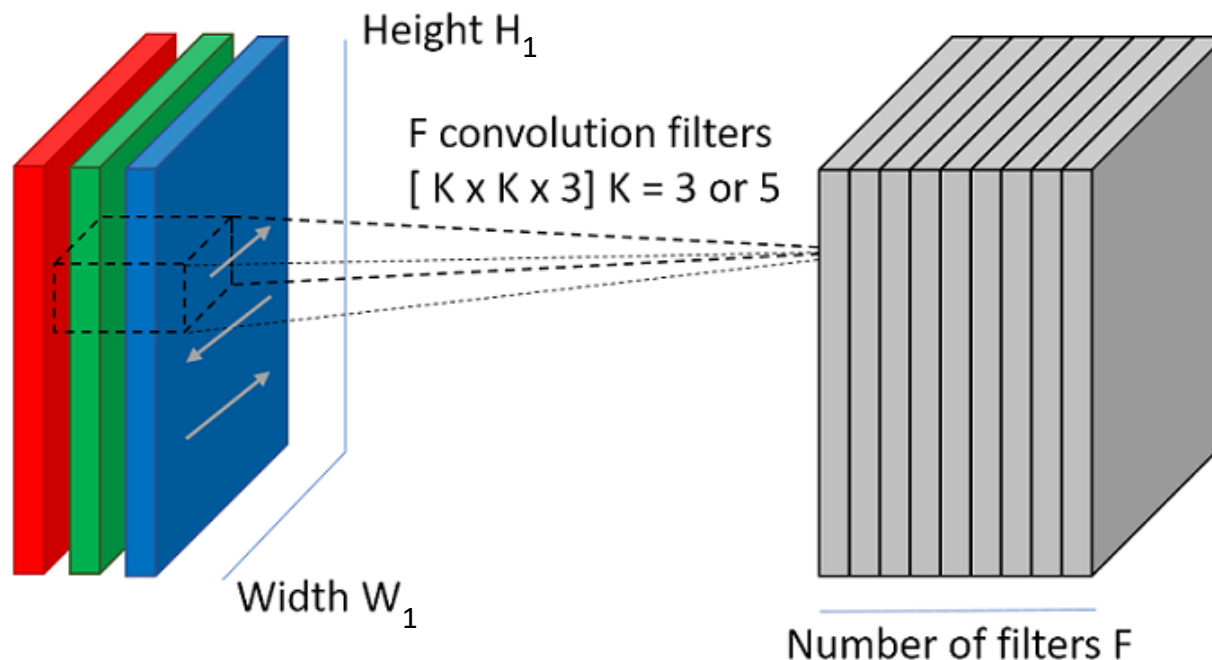
$$F$$

Výstup má rozměr

$$Q = \left\lfloor \frac{M + 2P - K}{S} \right\rfloor + 1$$

Hyperparametry:

- velikost filtru K
- počet filtrů F
- padding (okraj) P
- stride (krok) S



Vstup:

$$M \times M' \times C$$

Výstup:

$$Q \times Q' \times F$$

Příklad: RGB 32x32x3, 10 5x5 filtrů, bez paddingu, stride=1

Váhy W jsou tensor tvaru

$$5 \times 5 \times 3 \times 10$$

Bias b je vector délky

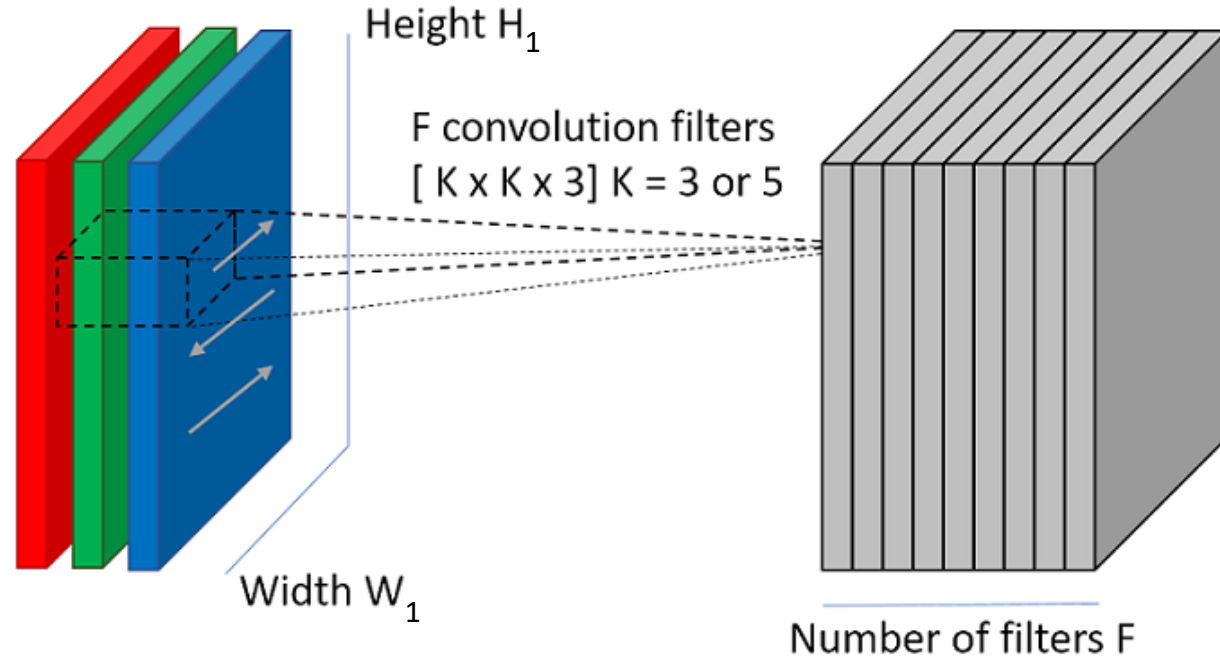
10 (počet filtrů)

Počet parametrů vrstvy

$$5 \cdot 5 \cdot 3 \cdot 10 + 10 = 760$$

$\underbrace{\hspace{1.5cm}}_{\text{váhy}} \quad \underbrace{\hspace{1.5cm}}_{\text{biasy}}$

Pro porovnání lineární vrstva $(32 \cdot 32 \cdot 3) \times (28 \cdot 28 \cdot 10)$
by měla $32 \cdot 32 \cdot 3 \cdot 28 \cdot 28 \cdot 10 \approx 24 \cdot 10^6$ parametrů!



Vstup tvaru

$$32 \times 32 \times 3$$

Výstup má rozměr

$$Q \times Q \text{ kde } Q = \left\lfloor \frac{32 + 2 \cdot 0 - 5}{1} \right\rfloor + 1 = 28$$

a hloubku

10 (počet filtrů)

Zpětný průchod konvoluce: gradient na váhy

- Dopředný průchod

$$z_{uv} = \sum_{i=1}^K \sum_{j=1}^K \sum_{c=1}^C w_{ijc} x_{\textcolor{green}{S}u+i, \textcolor{green}{S}v+j, c} + b$$

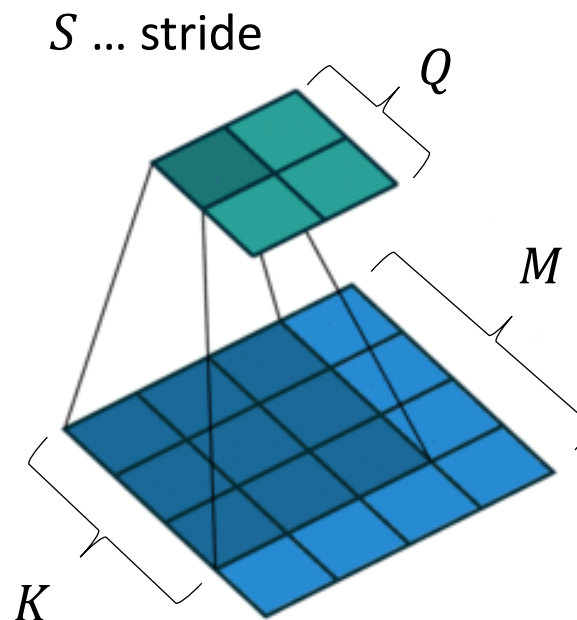
- Lokální gradient

$$\frac{\partial z_{uv}}{\partial w_{ijc}} = x_{i+Su, j+Sv, c}$$

- Celkový gradient na váhy

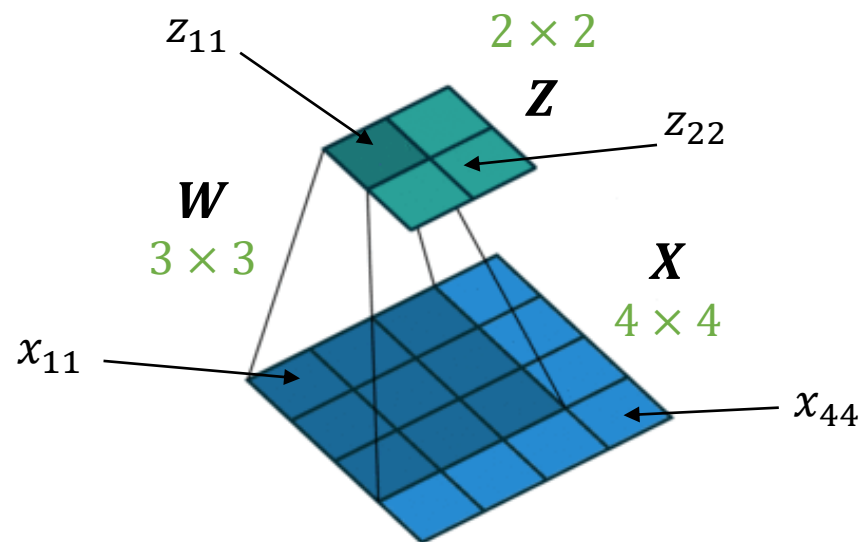
$$\begin{aligned} \frac{\partial L}{\partial w_{ijc}} &= \sum_{u=1}^Q \sum_{v=1}^Q \frac{\partial L}{\partial z_{uv}} \cdot \frac{\partial z_{uv}}{\partial w_{ijc}} \\ &= \sum_{\textcolor{red}{u}=1}^Q \sum_{\textcolor{red}{v}=1}^Q \bar{z}_{uv} \cdot x_{i+\textcolor{red}{S}u, j+\textcolor{red}{S}v, c} \end{aligned}$$

pokud $S > 1$:
“mezery” ve vstupu



- Zpětný průchod na váhy je tedy rovněž konvoluce!

Konvoluce jako lineární vrstva



konvoluci na obrázku lze zapsat maticově:

$2 \times 2 \rightarrow 4 \times 1$

$4 \times 4 \rightarrow 16 \times 1$

$$\begin{bmatrix} z_{11} \\ z_{12} \\ z_{21} \\ z_{22} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{11} & w_{12} & w_{13} & 0 & w_{21} & w_{22} & w_{23} & 0 & w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ \vdots \\ x_{42} \\ x_{43} \\ x_{44} \end{bmatrix}$$

... podobné lineární vrstvě

Zpětný průchod konvoluce: gradient na vstup

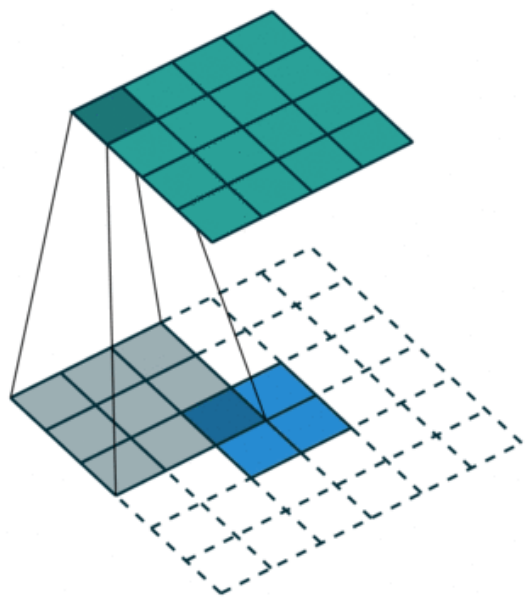
- Připomeňme, že pro lineární vrstvu
$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

je gradient na vstup

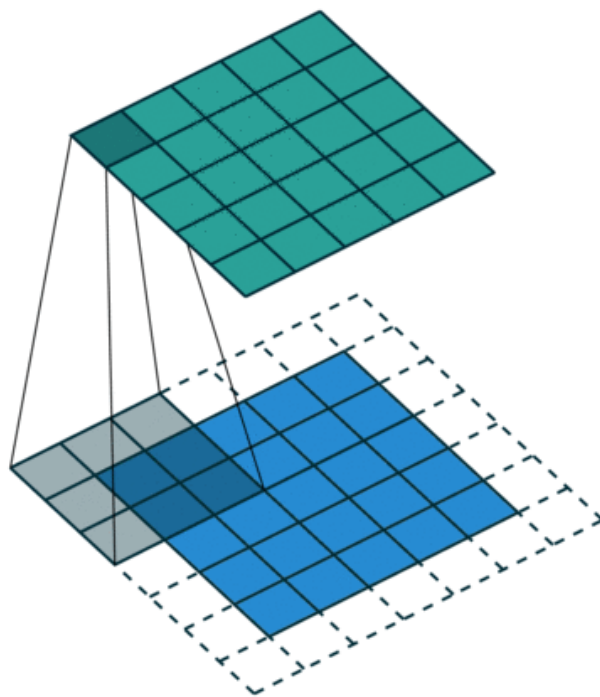
$$\bar{\mathbf{x}} = \mathbf{W}^T \bar{\mathbf{z}}$$

- Zpětná propagace gradientu na vstup konvoluce je tedy opět konvoluce, jejíž lineární forma má transponovanou matici \mathbf{W}
- Odtud anglický název **transposed convolution**
- “Obrácená” konvoluce: z tvaru výstupu \mathbf{z} na tvar vstupu \mathbf{x}
 - Nalezneme dokonce i název dekonvoluce: špatně, ve skutečnosti něco jiného

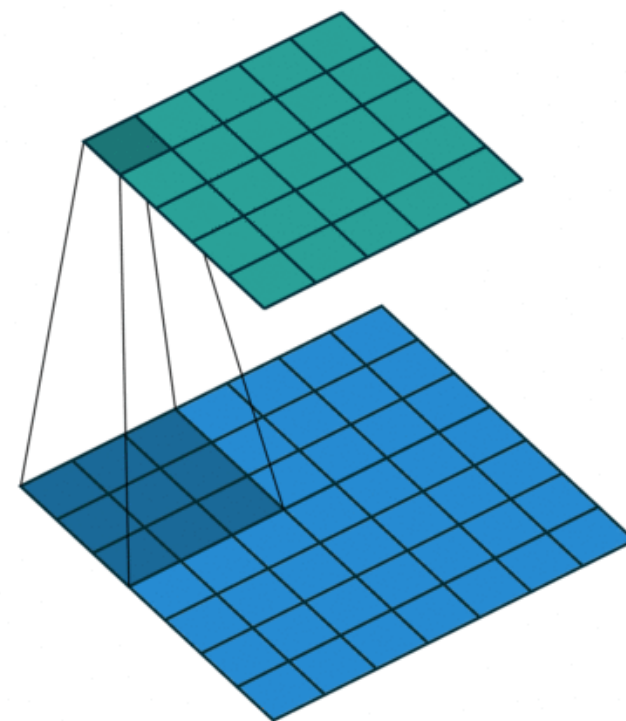
Ilustrace transponované konvoluce, stride=1



bez paddingu, stride=1

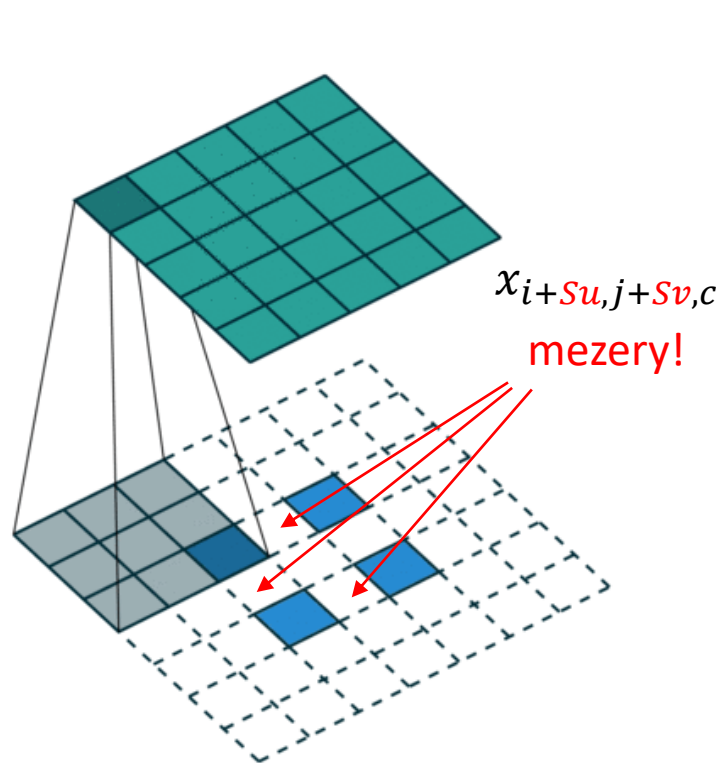


poloviční padding, stride=1

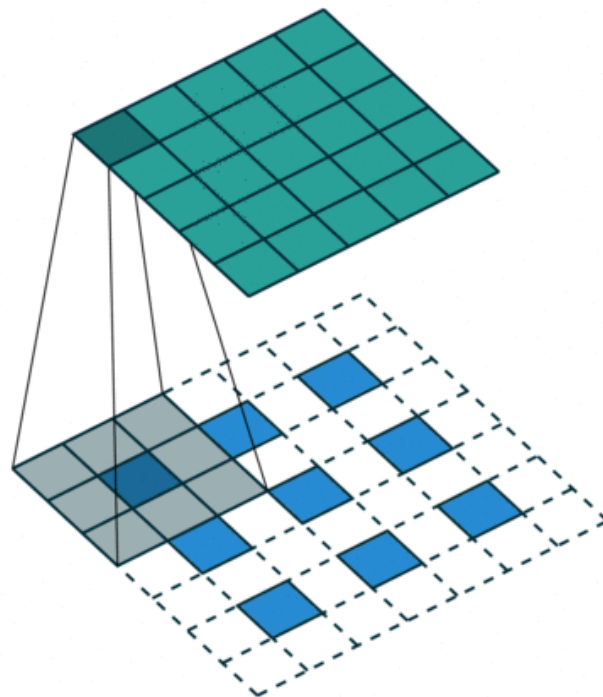


full padding, stride=1

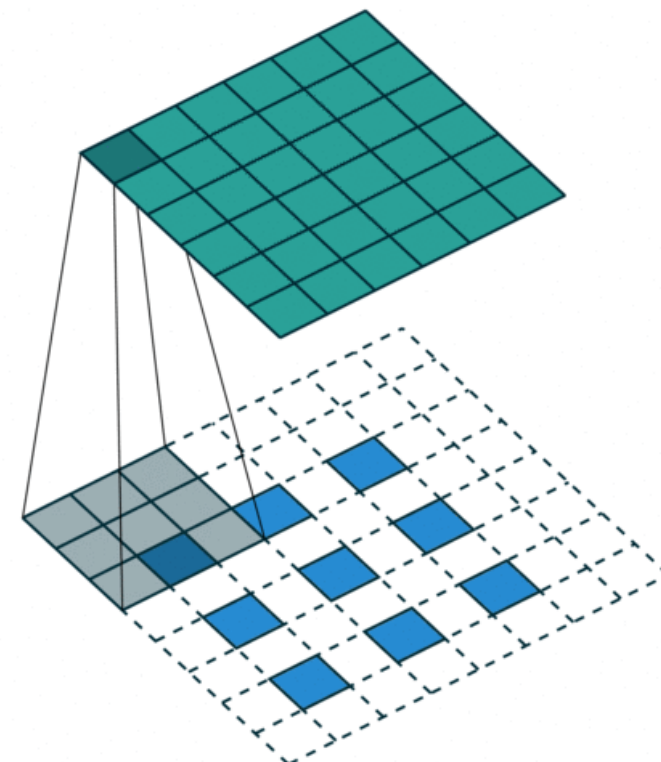
Ilustrace transponované konvoluce, stride > 1



bez paddingu, stride=2



poloviční padding, stride=2

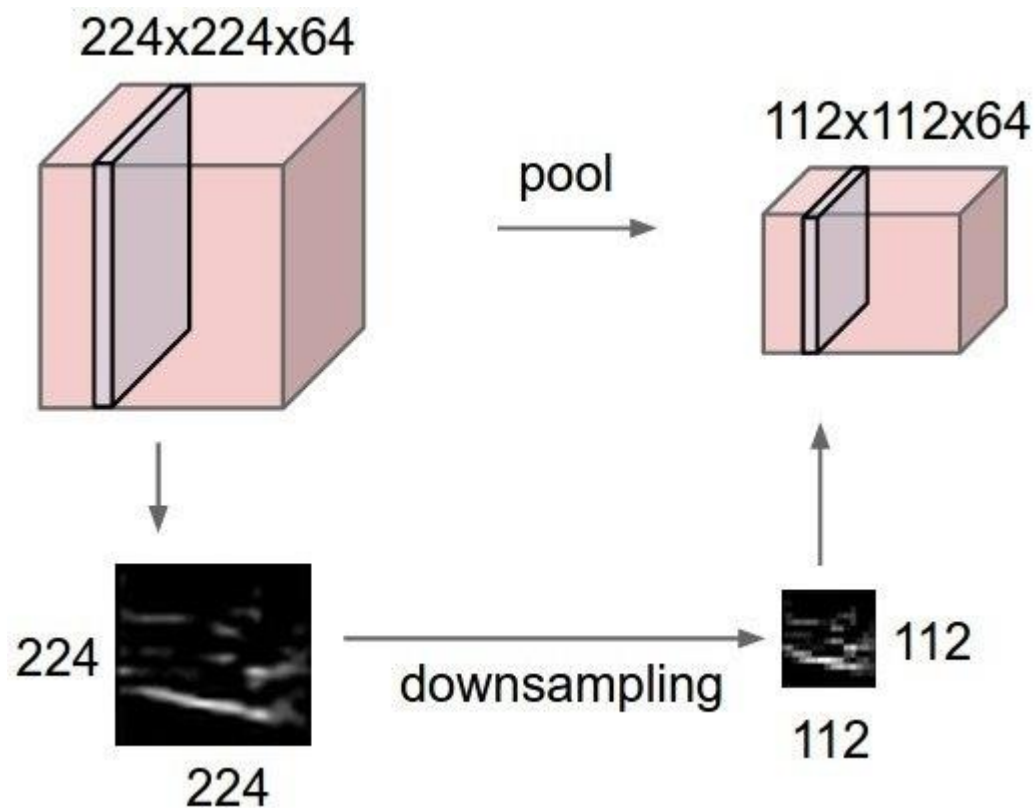


poloviční padding, stride=2
“když to nevyjde hezky”

mezery → anglický název **fractionally strided convolution**

Pooling

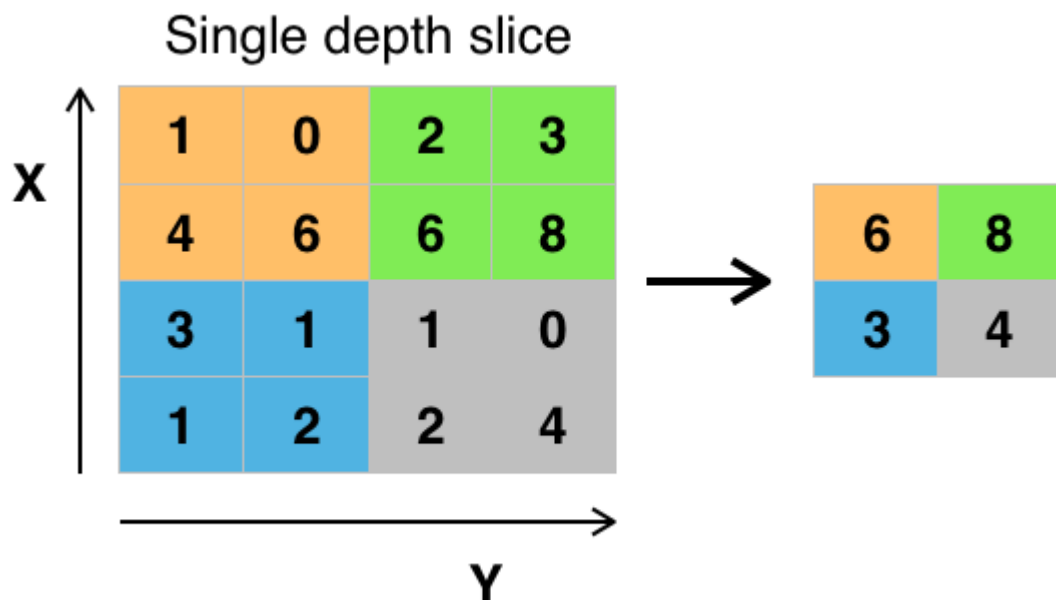
cílem zmenšit objem dat → méně paměti, tlak na kompresi příznakového prostoru



Max pooling

- Nejčastější forma poolingů
- Robustní vůči malému posunu vstupu

např. 2x2 max pooling, stride=2:



Počet parametrů: 0

výstup:

maximum přes každé okénko

vždy pouze pro jeden kanál vstupu →
redukuje pouze v x a y prostoru

příklad:

vstup: 32x32x3

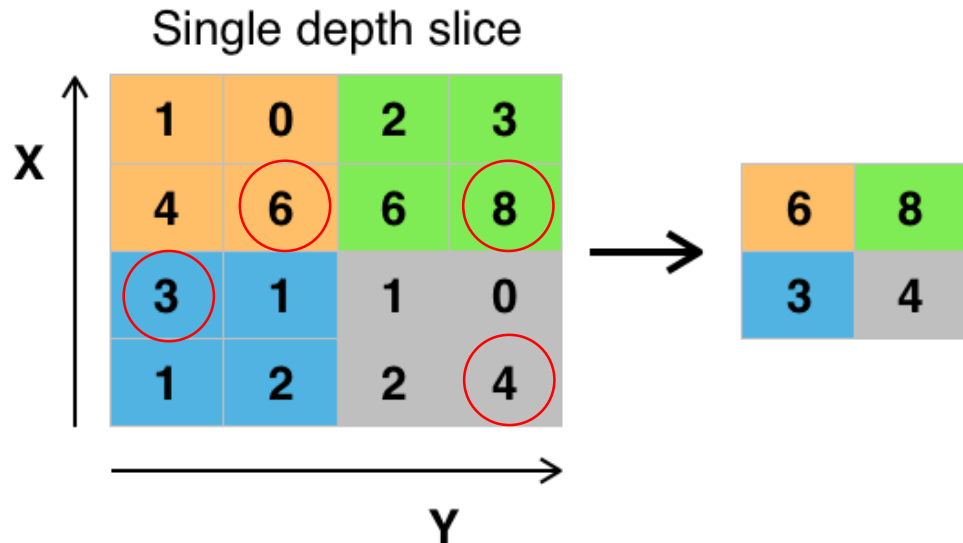
výstup: 16x16x3

Max pooling: zpětný průchod

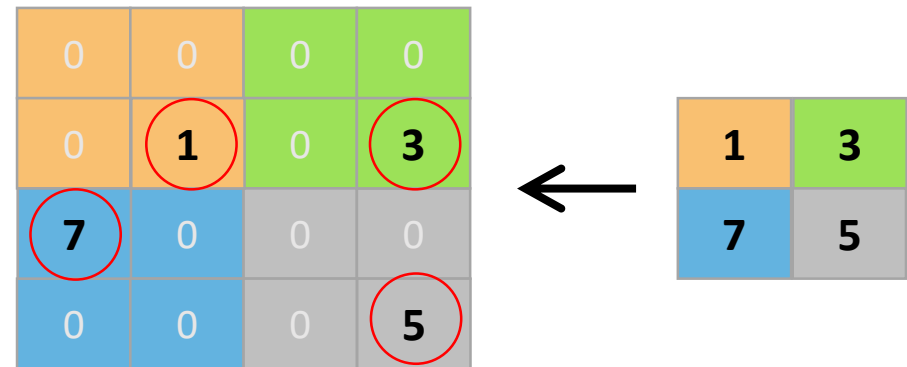
funkce max zapsána jinak: $z = \begin{cases} x_1 & \text{pokud } x_1 \geq x_i \forall x_i \in x \\ \vdots \\ x_D & \text{pokud } x_D \geq x_i \forall x_i \in x \end{cases} \Rightarrow \text{výběr prvku z pole}$

gradient pak je: $\overline{x_d} = \begin{cases} 1 & \text{pokud } x_d \geq x_i \forall x_i \in x \\ 0 & \text{jinak} \end{cases}$

dopředný průchod

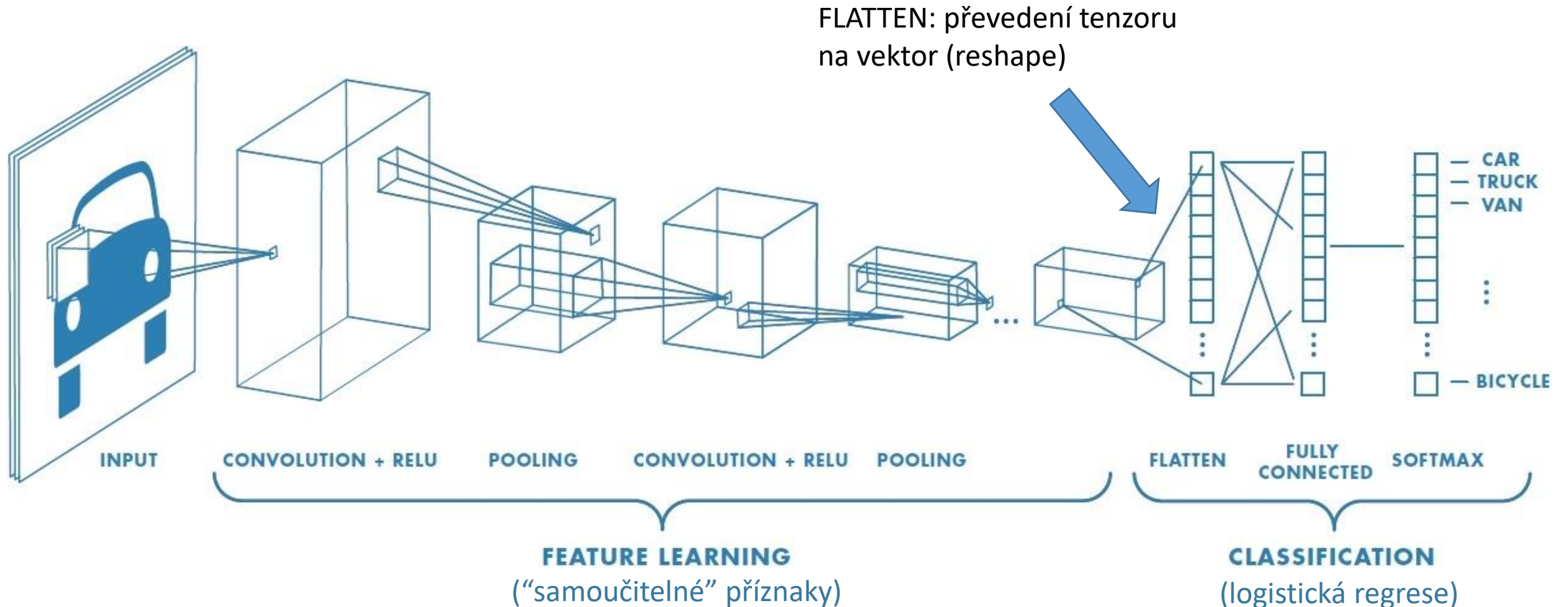


zpětný průchod



Konvoluční síť (Convolutional Neural Network, CNN)

- zadefinováním konvoluce jako bloku v neurosíti nyní můžeme libovolně kombinovat s ostatními vrstvami



obrázek: <https://ch.mathworks.com/solutions/deep-learning/convolutional-neural-network.html>

Rozpoznávání MNIST čísel: LeNet-5 (1998)

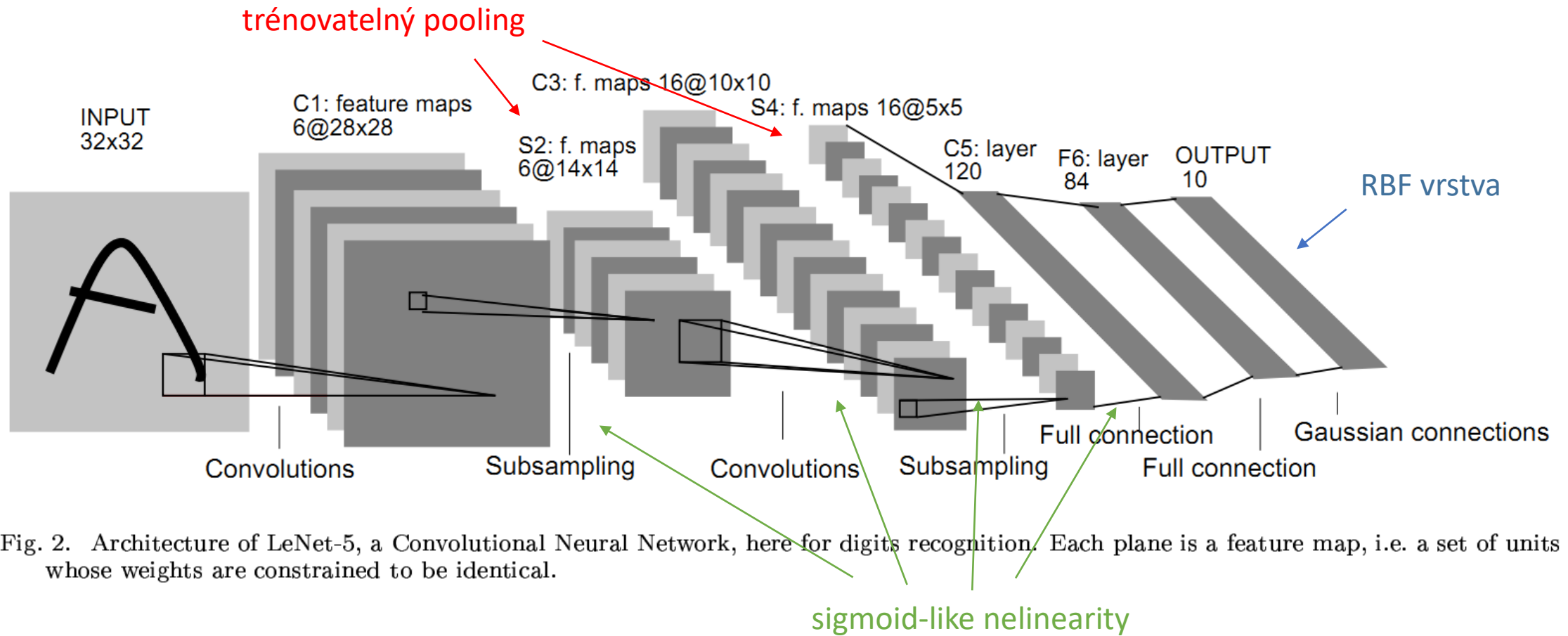


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- první známá a úspěšná konvoluční síť
- architektura: CONV-POOL-CONV-POOL-FC-FC-FC

FC ... Fully Connected

[Lecun et al. \(1998\): GradientBased Learning Applied to Document Recognition](#)

Rozpoznávání MNIST čísel: LeNet-5 (1998)

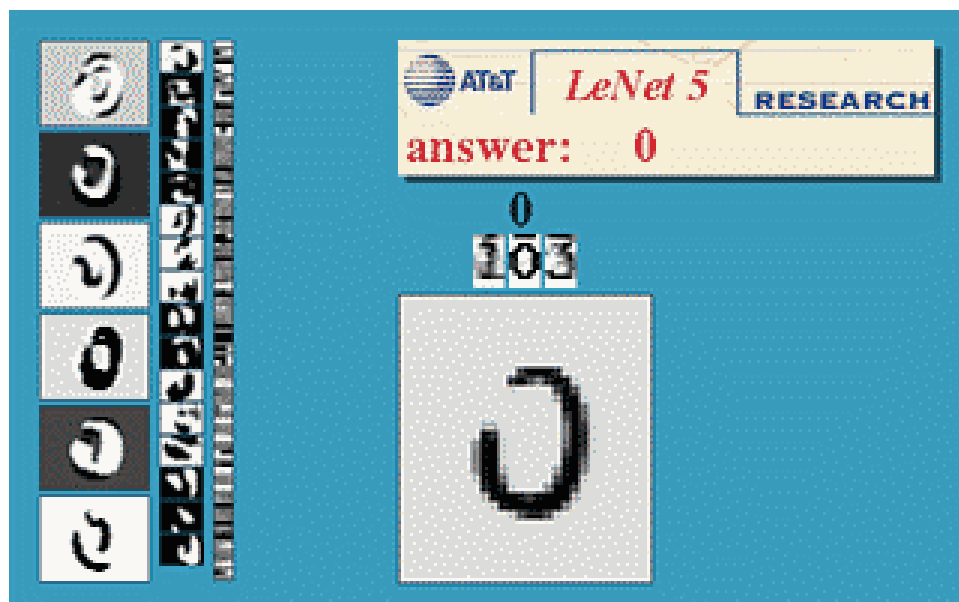


Fig. 4. Size-normalized examples from the MNIST database.

MNIST ... 60000 obrázků čísel

animace: <http://yann.lecun.com/exdb/lenet/>

LeNet-5: dobové výsledky (error rate) na MNIST

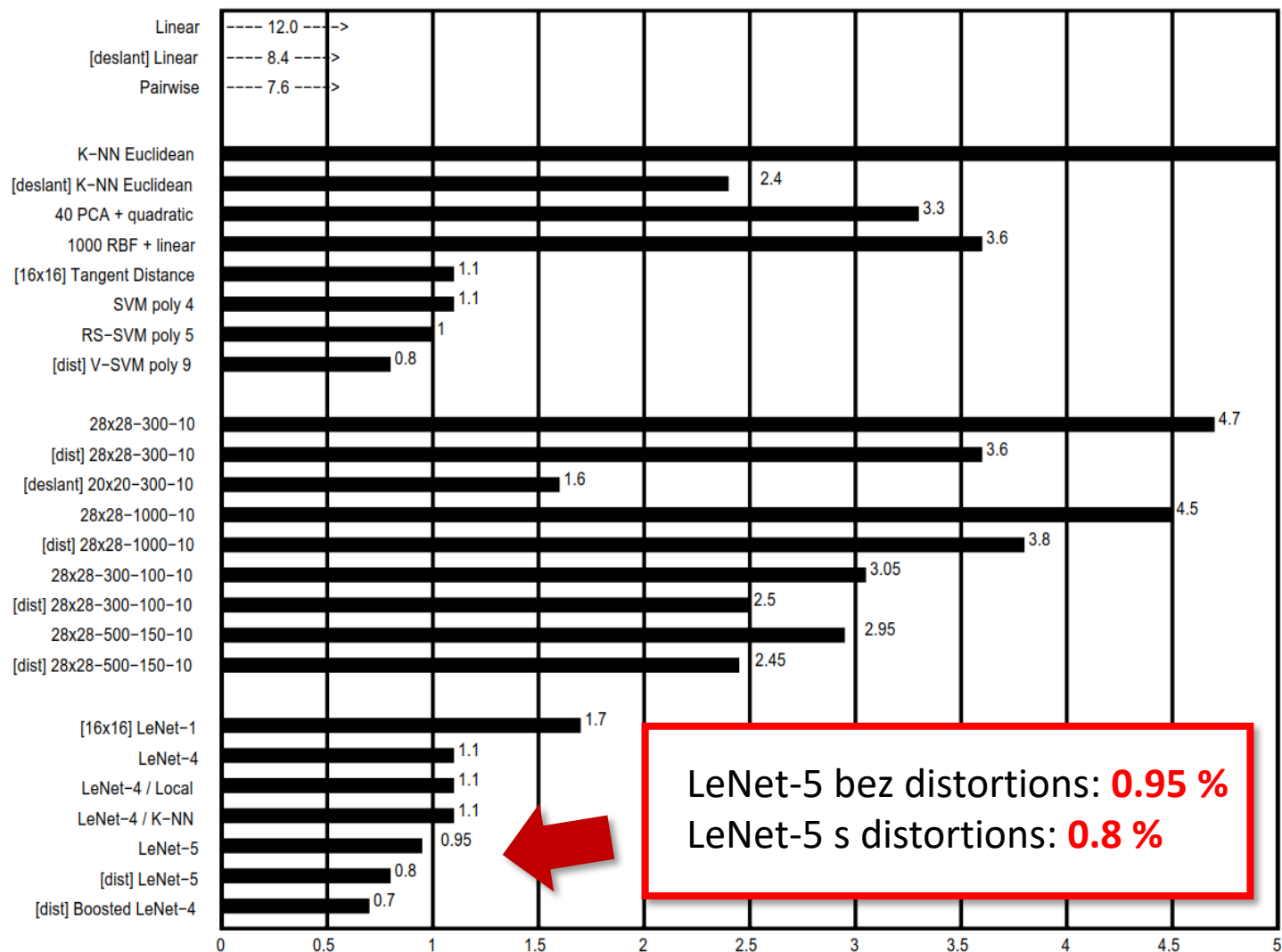
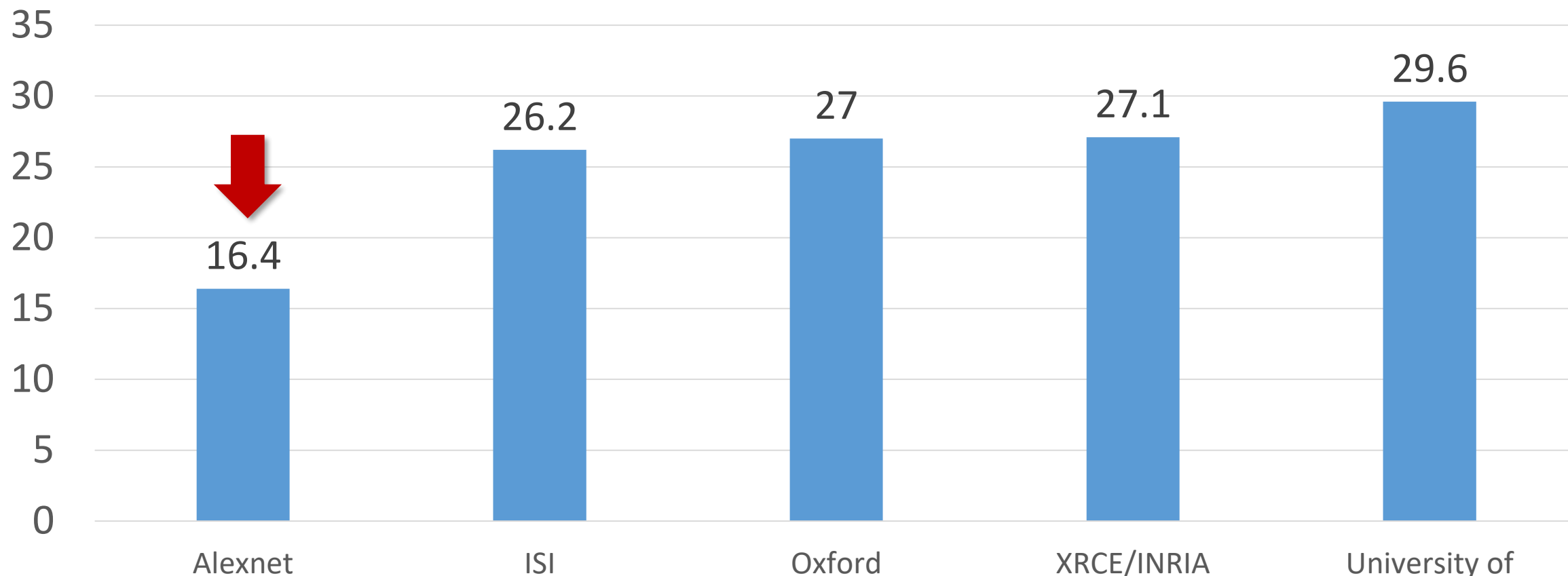


Fig. 9. Error rate on the test set (%) for various classification methods. [deslant] indicates that the classifier was trained and tested on the deslanted version of the database. [dist] indicates that the training set was augmented with artificially distorted examples. [16x16] indicates that the system used the 16x16 pixel images. The uncertainty in the quoted error rates is about 0.1%.

Rozpoznávání ImageNet: Alexnet (2012)

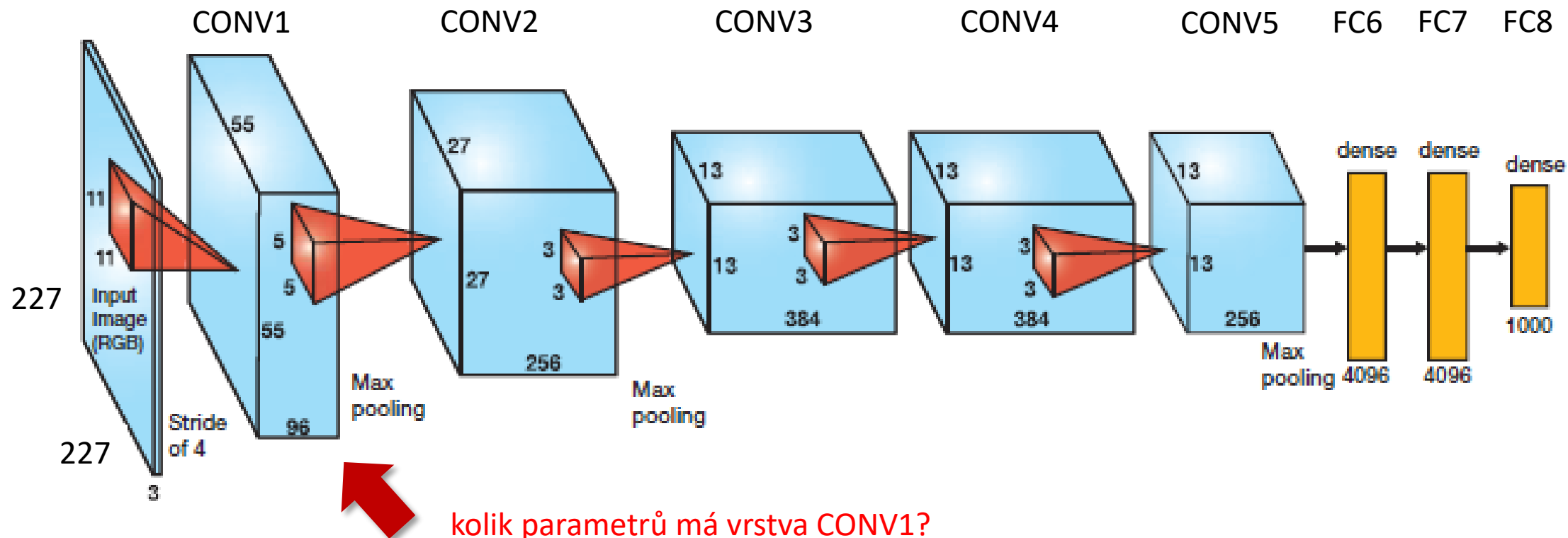
1000 classes, Top-5 error [%]



top-5 ... správná třída musí být
do 5. místa dle skóre z výstupu
klasifikátoru

non-DNN modely

Alexnet (2012)

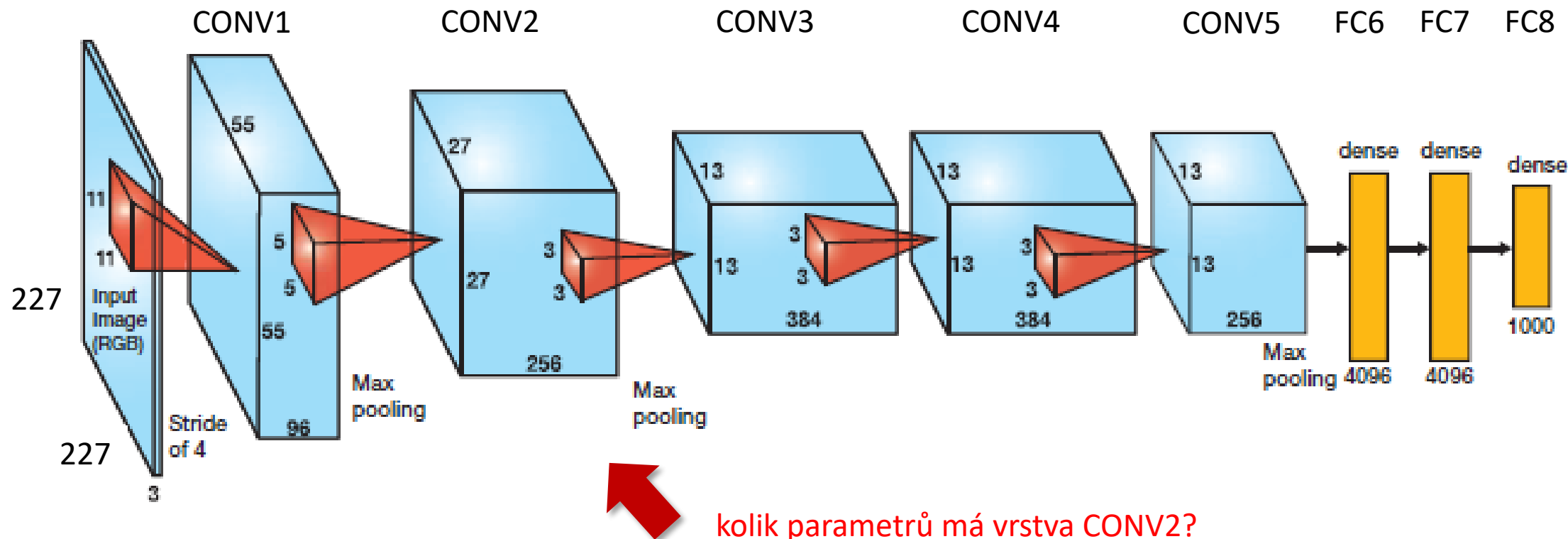


kolik parametrů má vrstva CONV1?

$$11 \cdot 11 \cdot 3 \cdot 96 = 34\,848$$

- architektura: CONV-POOL-NORM-CONV-POOL-NORM-CONV-CONV-CONV-FC-FC-FC
- “naškálovaná” LeNet-5

Alexnet (2012)

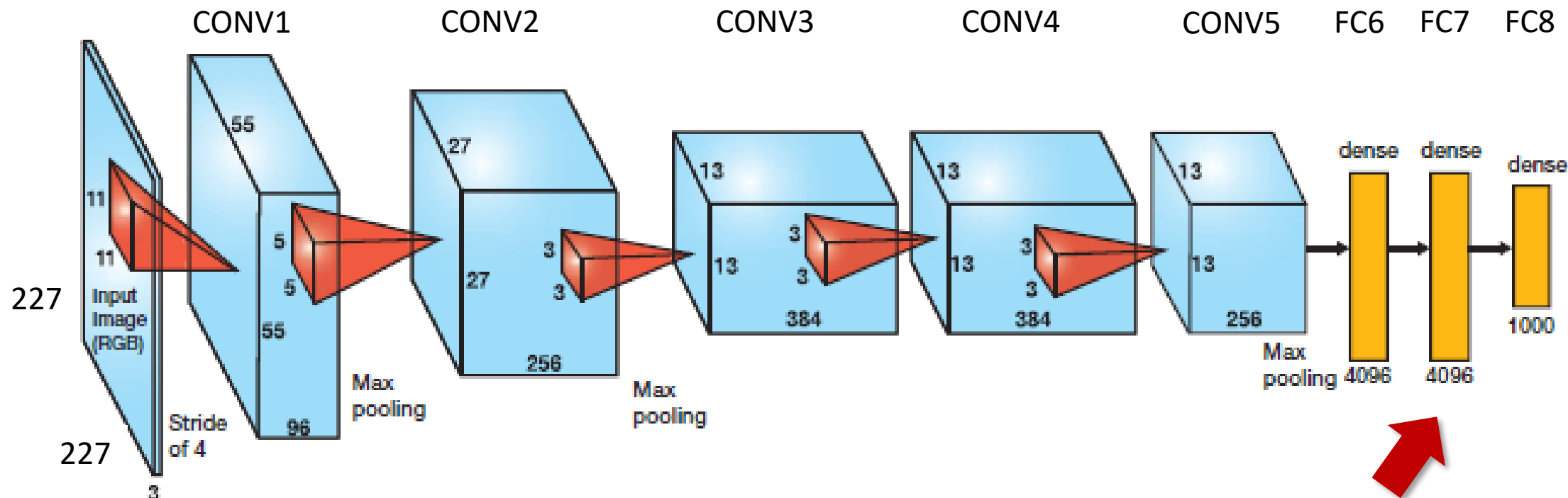


kolik parametrů má vrstva CONV2?

$$5 \cdot 5 \cdot 96 \cdot 256 = 614\,400$$

- architektura: CONV-POOL-NORM-CONV-POOL-NORM-CONV-CONV-CONV-FC-FC-FC
- “naškálovaná” LeNet-5

Alexnet (2012)



kolik parametrů má vrstva FC7?

$$4096 \cdot 4096 = 16\,777\,216 (!)$$

- architektura: CONV-POOL-NOI
- “naškálovaná” LeNet-5

Konvoluční vrstvy mají výrazně nižší počet parametrů oproti lineárnímu, což je jeden z důvodů, proč se snáze trénují a lépe fungují. Dokonce současný trend jsou tzv. Fully Convolutional Nets, kde je vše vyjádřeno jako konvoluce!

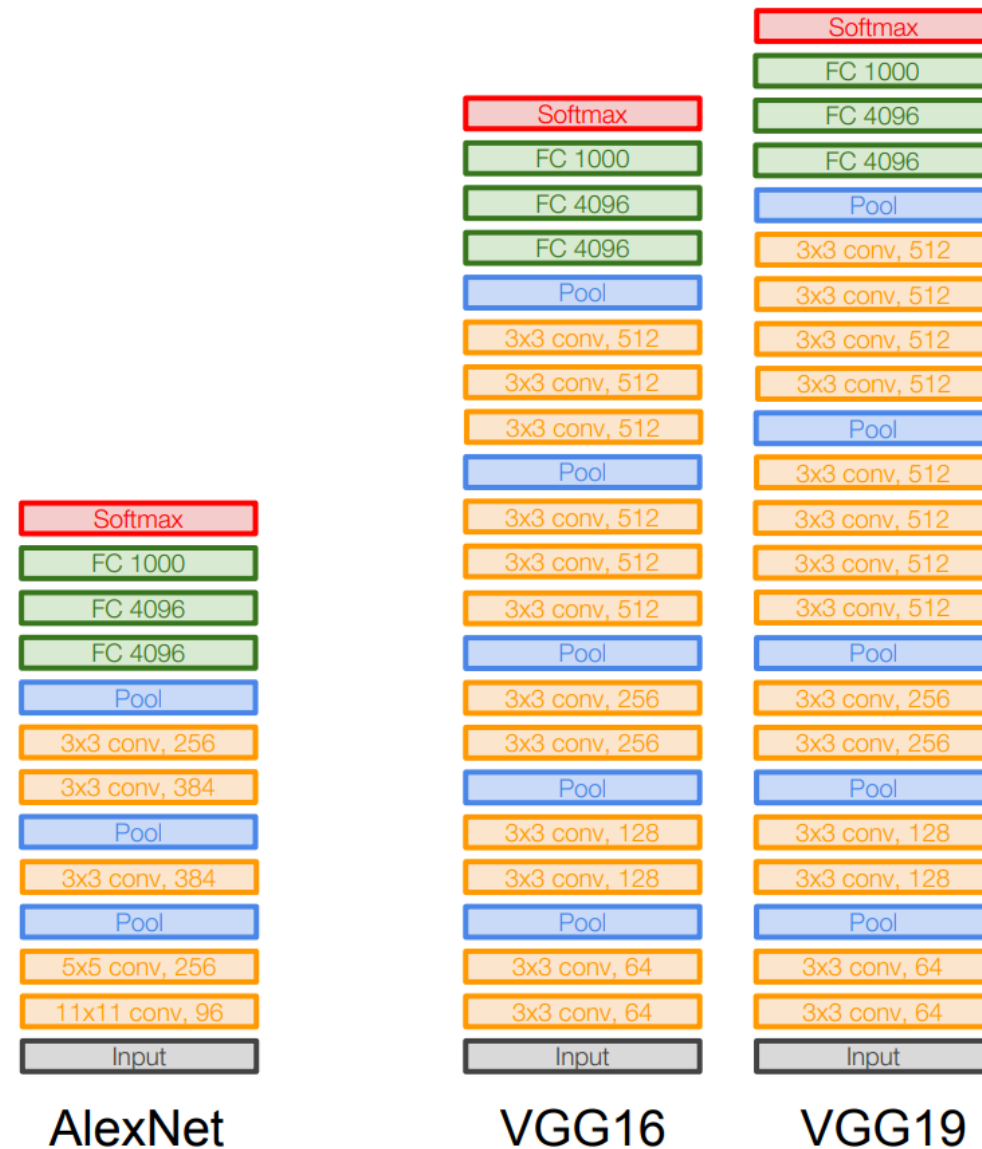
-FC-FC

Alexnet (2012)

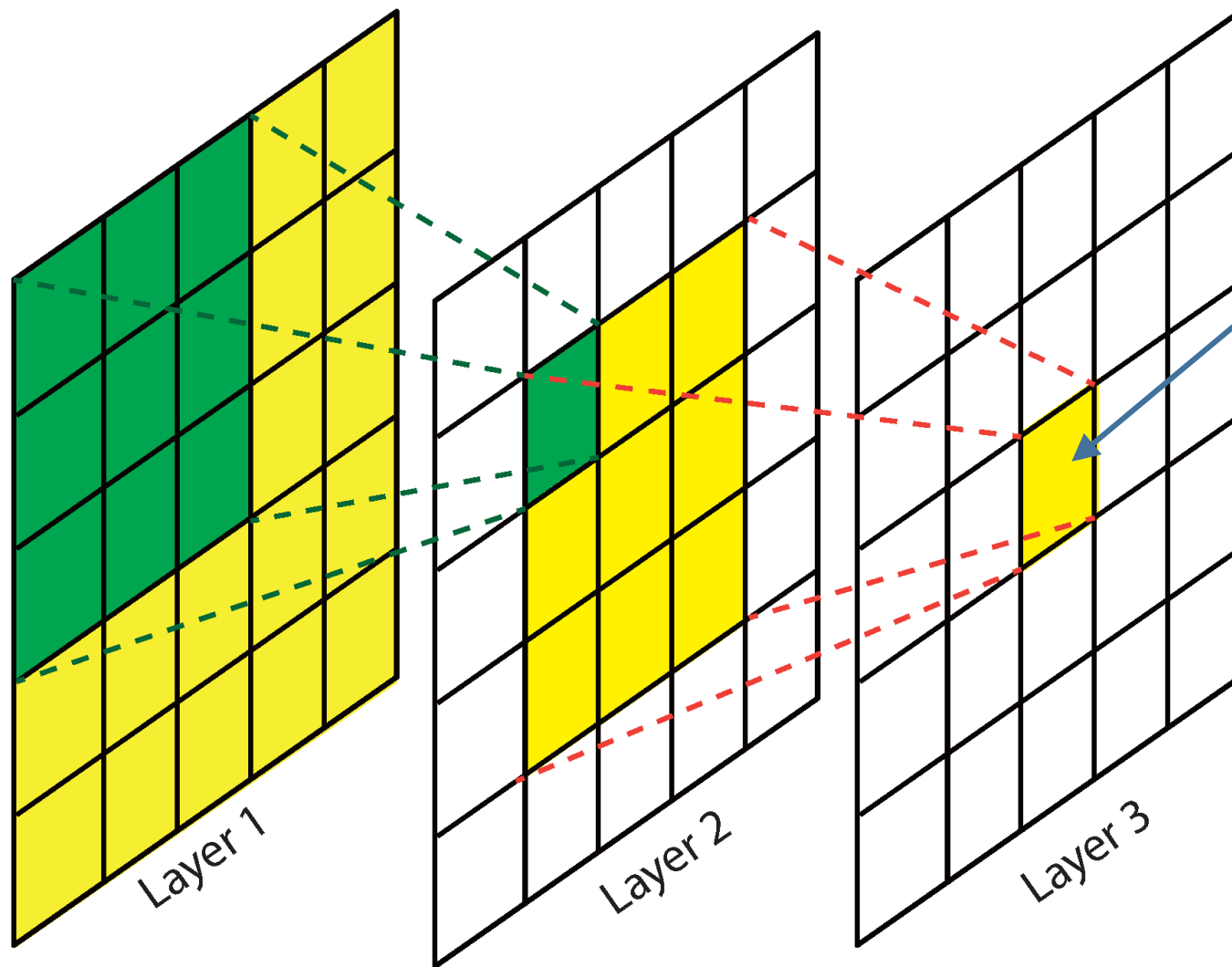
- [Krizhevsky, Sutskever, Hinton: “ImageNet Classification with Deep Convolutional Neural Networks”](#)
- Síť, která “nastartovala DNN/CNN revoluci”
- Autoři nevyvinuli žádný nový algoritmus, “pouze” ukázali, jak správně CNN používat
- Místo sigmoid aktivací přechod na ReLU
- Kromě klasické L2 regularizace navíc Dropout
- Výrazné umělé rozšiřování dat (data augmentation)
- Místo SGD → Momentum SGD
- Postupné snižování learning rate
- Trénováno na dvou GTX 580 celkem 5-6 dní

VGG (2014)

- [Simonyan, Zisserman: “Very Deep Convolutional Networks for Large-Scale Image Recognition”](#)
- Druhé místo ImageNet competition 2014
- Mnohem jednodušší architektura než vítěz (GoogLeNet)
- Velmi podobné AlexNet
- Místo 11x11 apod. konvolucí pouze 3x3
- Pouze 2x2 max-pooling
- Žádná lokální normalizace
- 16 a 19 vrstev
- VGG-16: 8.4 % top-5 error



Receptive field



výsledek konvoluce 3x3 nad druhou vrstvou závisí na 5x5 oblasti v první vrstvě

pro 3. vrstvu by to bylo 7x7, pro 4. vrstvu 9x9, atd.

jednotlivé neurony každé další vrstvy tedy popisují větší a větší část obrázku

VGG (2014)

- VGG tedy nahrazuje jedinou 7x7 konvoluci vrstvenými 3x3 konvolucemi
- Stejné “zorné pole” (receptive field)
- Méně parametrů:

$$3 \times (3 \times 3 \times C \times C) = 27C^2$$

vs

$$1 \times (7 \times 7 \times C \times C) = 49C^2$$

- Zároveň více nelinearit
- Výsledné “FC7” příznaky (4096D) lze použít i pro jiné úlohy (viz transfer learning)
- Poslední klasická CNN

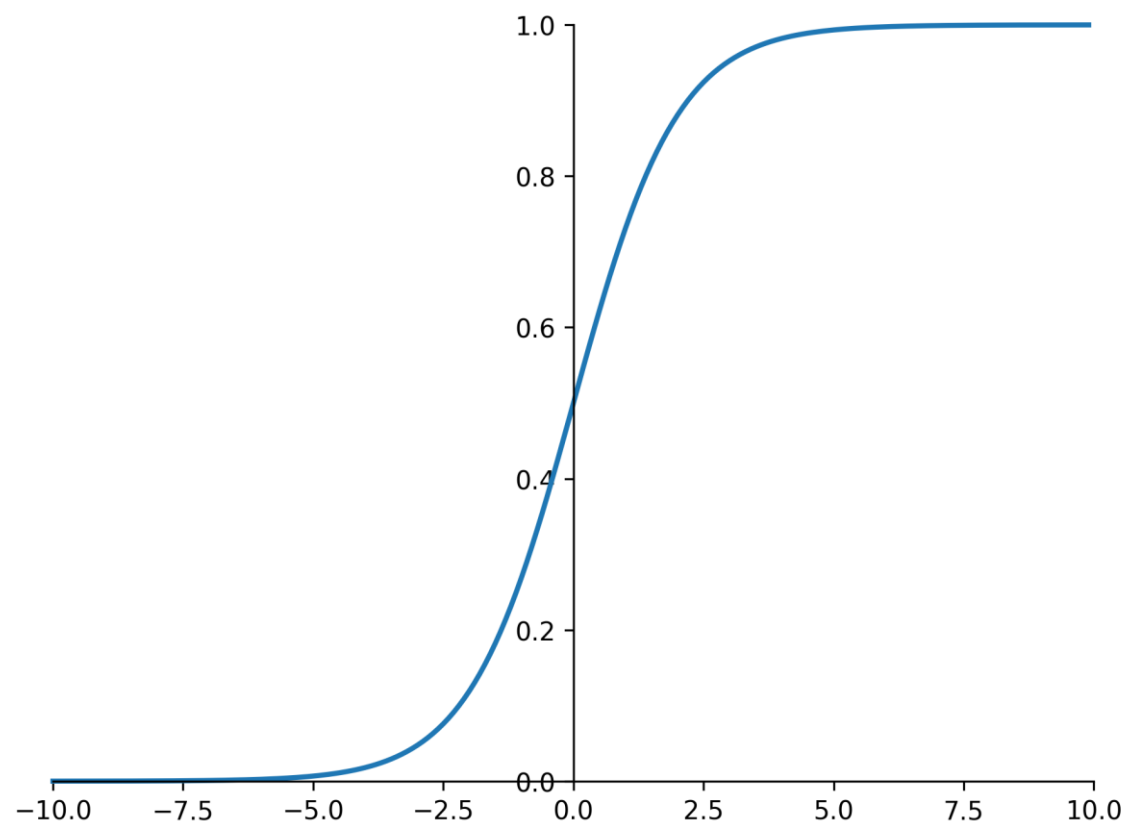


Aktivace

Sigmoid

- Před Alexnet prakticky jediná používaná aktivace
- Převádí vstup na pravděpodobnost, tj. do intervalu $\langle 0, 1 \rangle$
- Vstup x jsou typicky skóre s z předchozí lineární vrstvy
- Problémy:
 1. "umírající" gradient
 2. pouze kladné hodnoty
 3. exp funkce zbytečně náročná na výpočet

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



Sigmoid

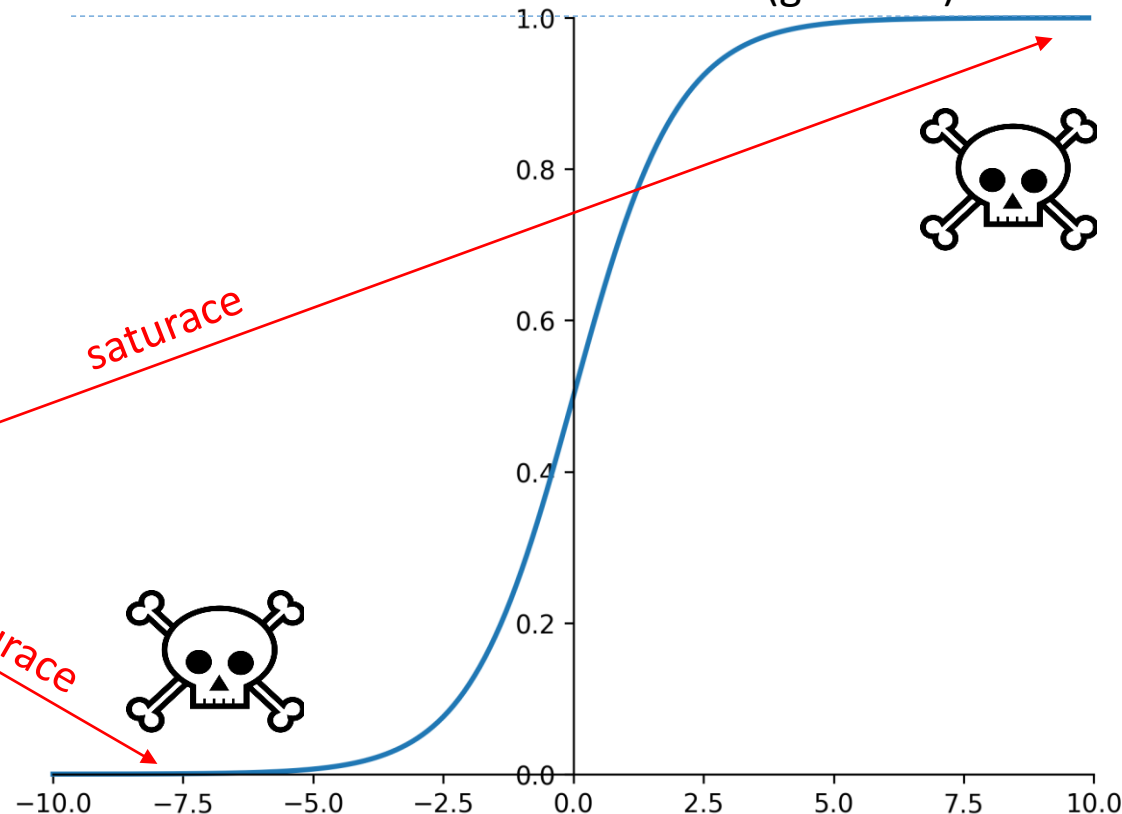
- Před Alexnet prakticky jediná používaná aktivace
- Převádí vstup na pravděpodobnost, tj. do intervalu $\langle 0, 1 \rangle$
- Vstup x jsou typicky skóre s z předchozí lineární vrstvy
- Problémy:
 1. **“umírající” gradient**
 2. pouze kladné hodnoty
 3. exp funkce zbytečně náročná na výpočet



vanishing gradient

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

vodorovná tečna = nulová derivace (gradient)

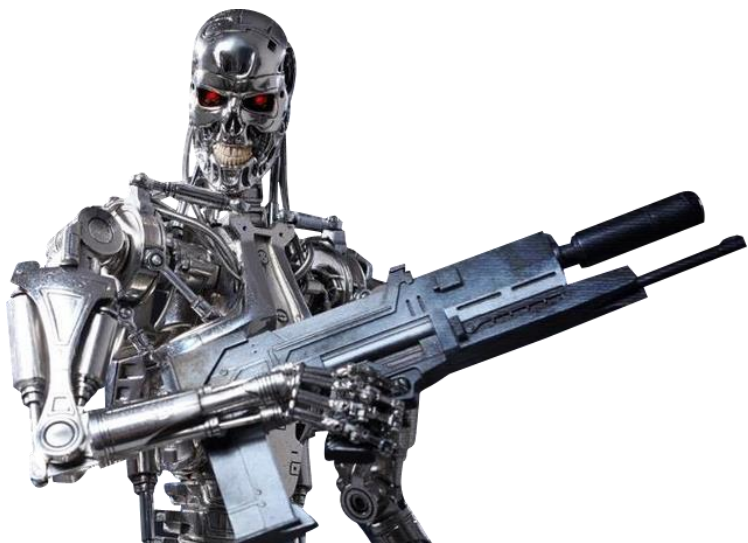


Hyperbolický tangens

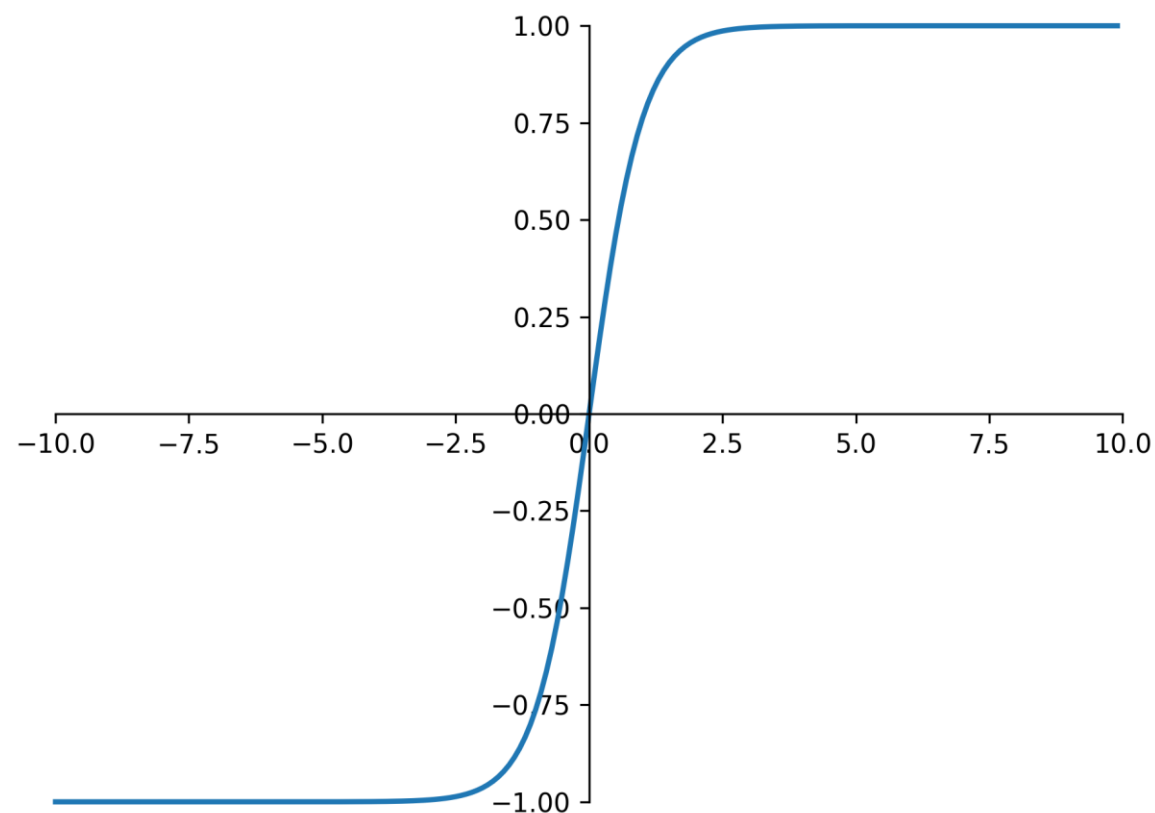
- Vlastně jen přeškálovaný sigmoid

$$\tanh(x) = 2 \cdot \sigma(2x) - 1$$

- Pouze tedy vycentruje sigmoid
- Ale stále “zabíjí” gradient



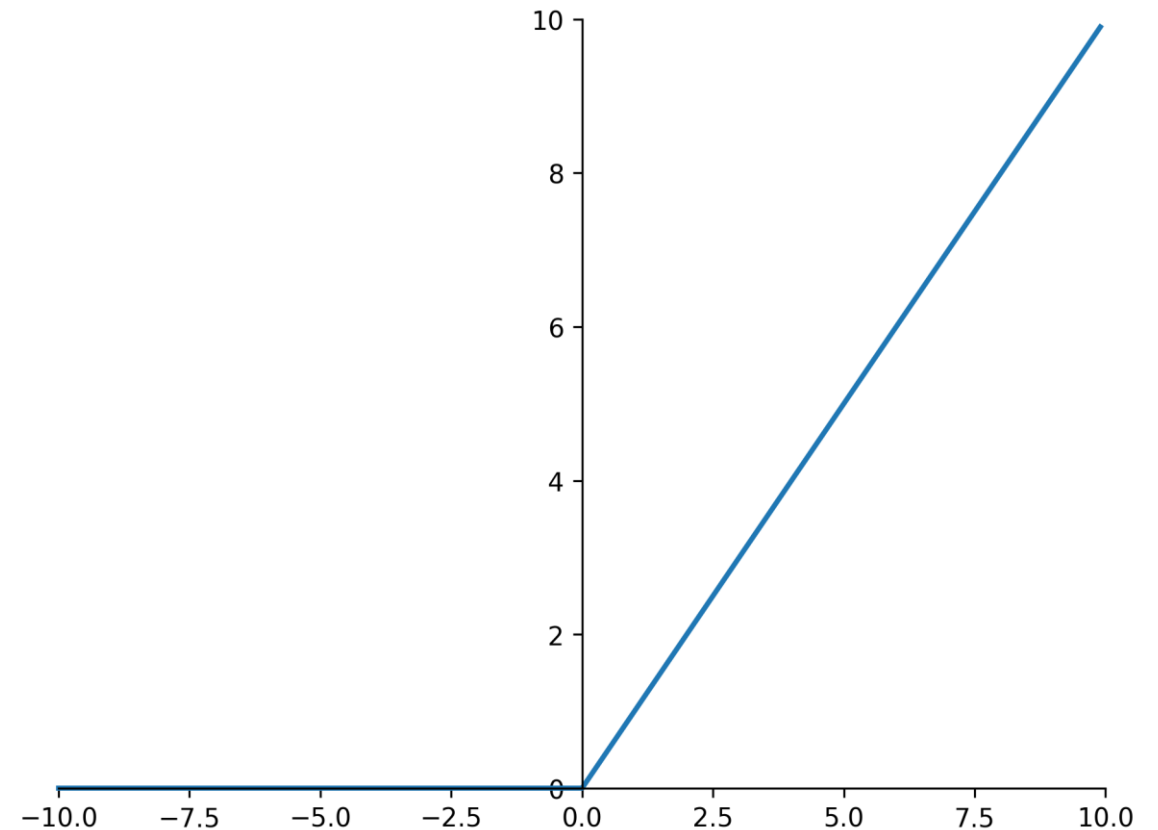
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



ReLU

$$\text{ReLU}(x) = \max(0, x)$$

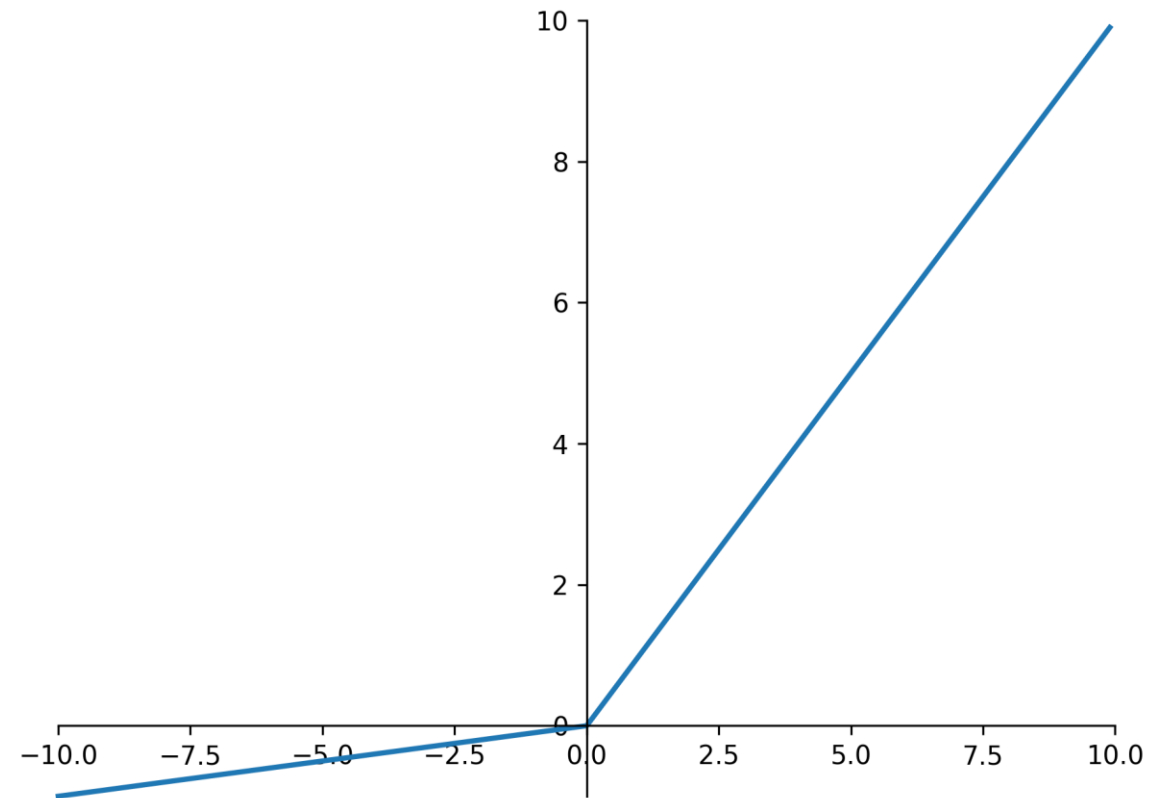
- Rectified Linear Unit
- Saturuje pouze v záporu
- Výpočetně nenáročné (bez expů)
- Trénování je mnohem rychlejší než se sigmoid či tanh
- Defaultní volba pro vnitřní nonlinearity



Leaky/Parametric ReLU

$$\text{PReLU}(x) = \max(\alpha x, x)$$

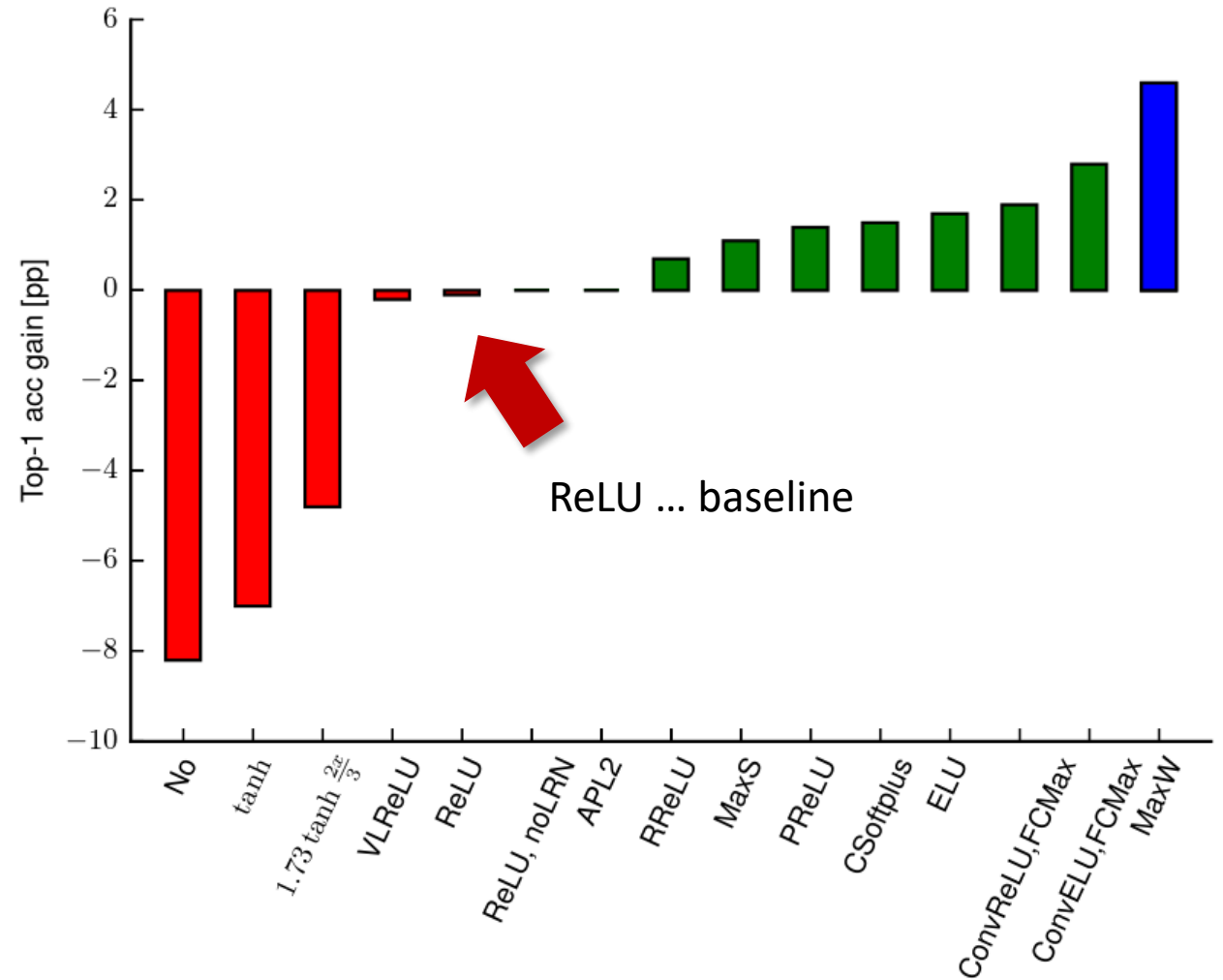
- Snaží se řešit problémy ReLU
 - nemá saturaci → neumírá gradient
 - zachovává rychlost
- Parametr α buď jako
 - konst. (např. 0.01) → Leaky ReLU
 - učitelný parameter → Parametric ReLU



Další nonlinearity

Table 3: Non-linearities tested.

Name	Formula	Year
none	$y = x$	-
sigmoid	$y = \frac{1}{1+e^{-x}}$	1986
tanh	$y = \frac{e^{2x}-1}{e^{2x}+1}$	1986
ReLU	$y = \max(x, 0)$	2010
(centered) SoftPlus	$y = \ln(e^x + 1) - \ln 2$	2011
LReLU	$y = \max(x, \alpha x), \alpha \approx 0.01$	2011
maxout	$y = \max(W_1x + b_1, W_2x + b_2)$	2013
APL	$y = \max(x, 0) + \sum_{s=1}^S a_i^s \max(0, -x + b_i^s)$	2014
VReLU	$y = \max(x, \alpha x), \alpha \in 0.1, 0.5$	2014
RReLU	$y = \max(x, \alpha x), \alpha = \text{random}(0.1, 0.5)$	2015
PReLU	$y = \max(x, \alpha x), \alpha \text{ is learnable}$	2015
ELU	$y = x, \text{ if } x \geq 0, \text{ else } \alpha(e^x - 1)$	2015



Aktivace & Inicializace

- Nejjednodušší zkusit ReLU → když funguje, vyzkoušet její variant nebo např. SELU
- U ReLU namísto Glorot inicializace vhodnější **He**:

```
Wi = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in / 2.)
```

- Odvození: <http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization>
- Co biasy?
 - většinou inicializujeme na nuly: `b = np.zeros(fan_out)`
 - U ReLU možné malé kladné hodnoty: `b = 0.01 * np.ones(fan_out)`

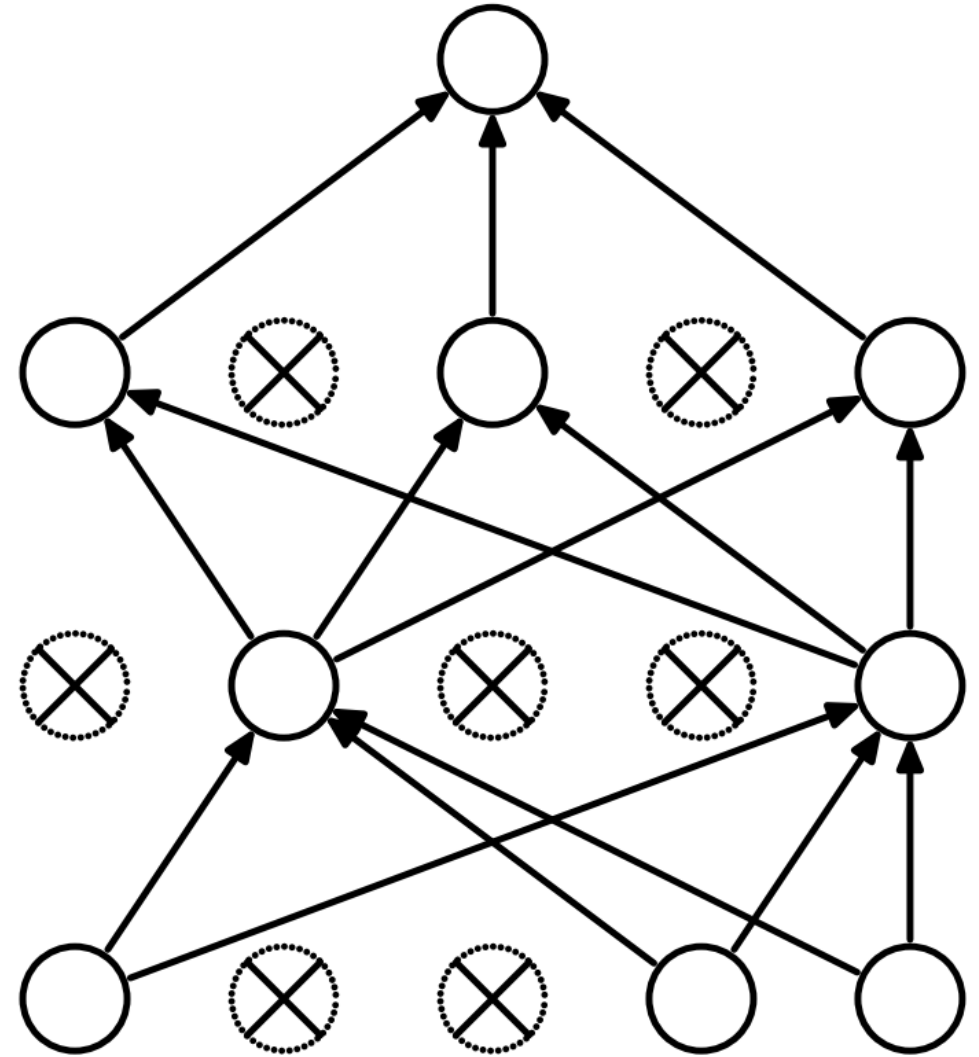
Dropout regularizace

Dropout

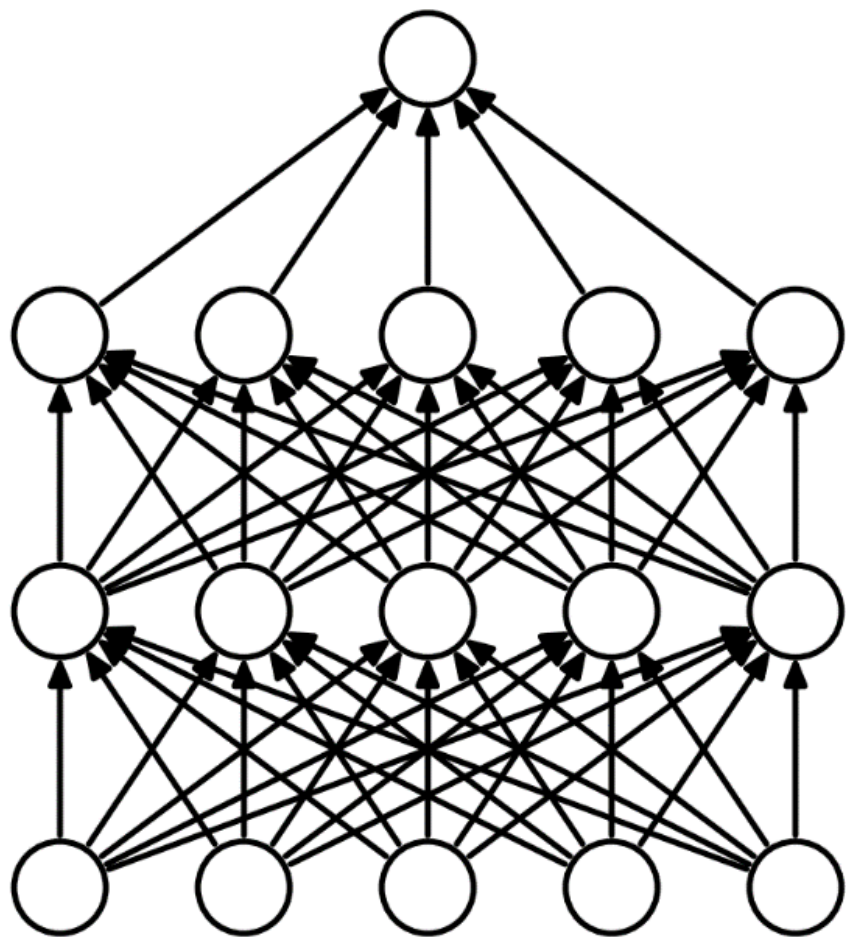
- Náhodně nastavuje výstupy na nulu
- Např. s pravděpodobností 40 %:

```
1 scores = np.dot(x, w) + b
2 hidden = np.maximum(0., scores)
3 mask = np.random.rand(*hidden.shape) < 0.4
4 hidden[mask] = 0.
```

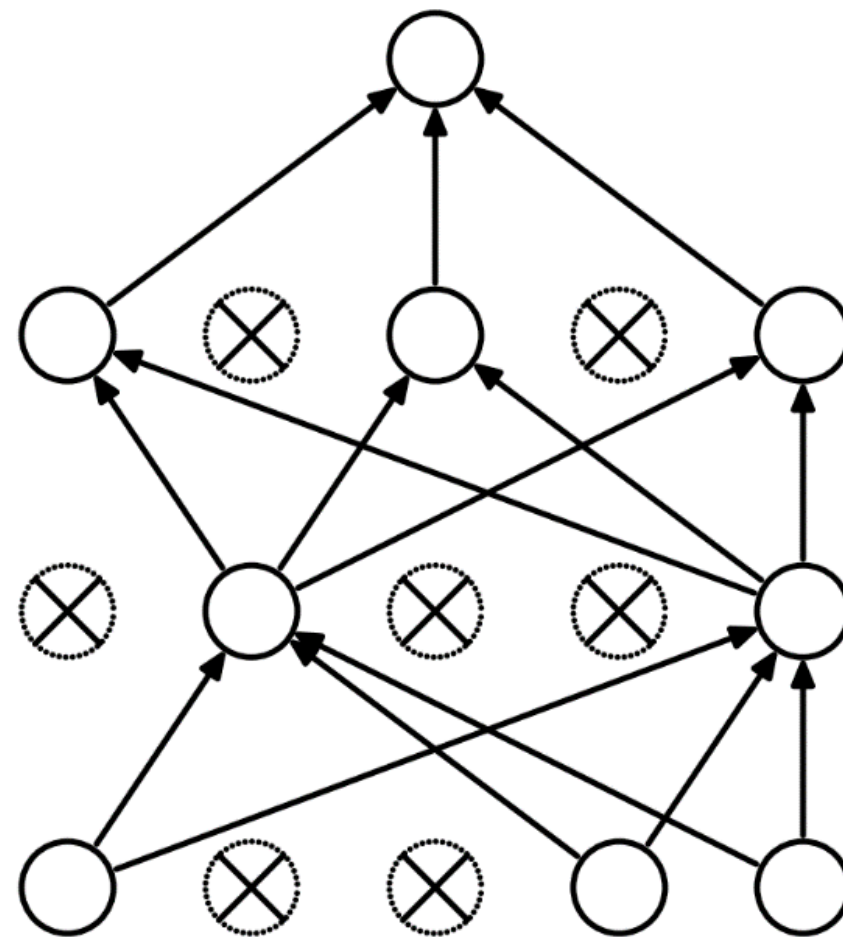
- slouží jako regularizace → prevence overfitu



Dropout



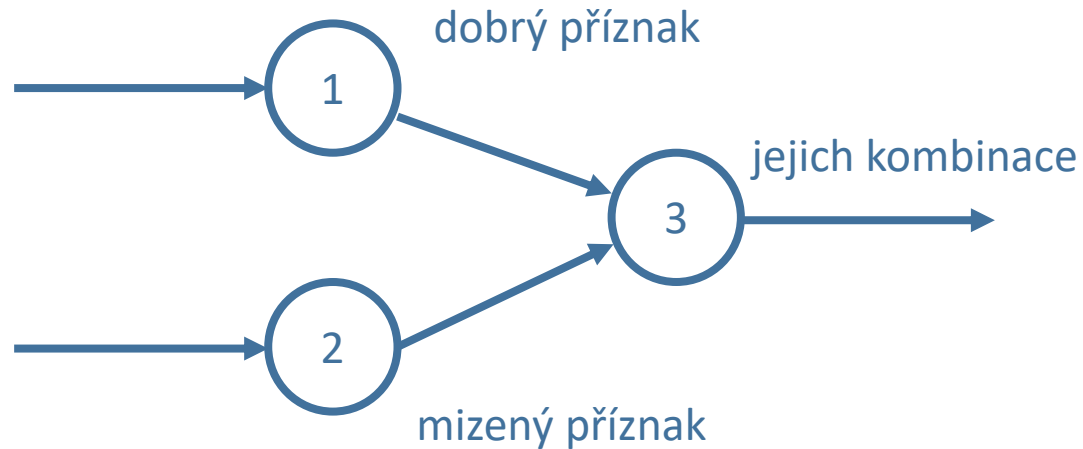
(a) Standard Neural Net



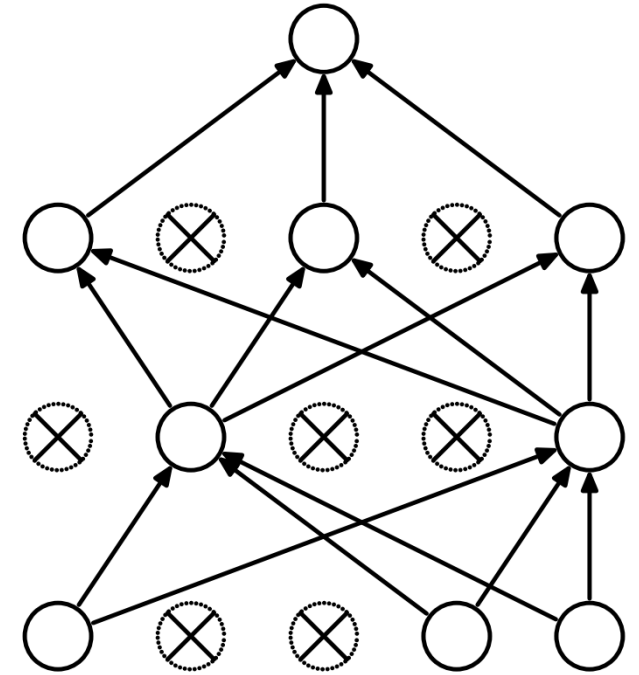
(b) After applying dropout.

Dropout

- Nutí síť vytvářet robustní a redundantní příznaky
 - náhodně vypadávají → tlak, aby **všechny** dobře reprezentovaly



- Model ensemble
 - de facto vytváří kombinaci více modelů, které sdílejí váhy
 - každá maska reprezentuje jednu síť
 - jedna síť = jeden trénovací vektor



Dropout v trénovací a testovací fázi

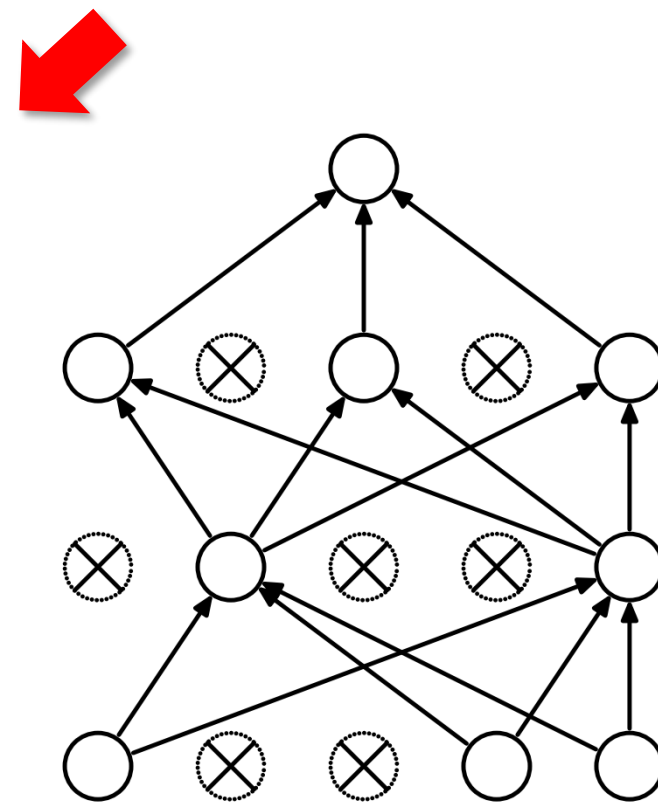
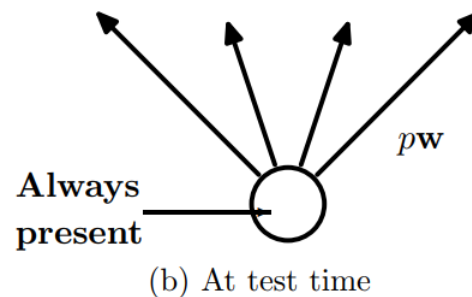
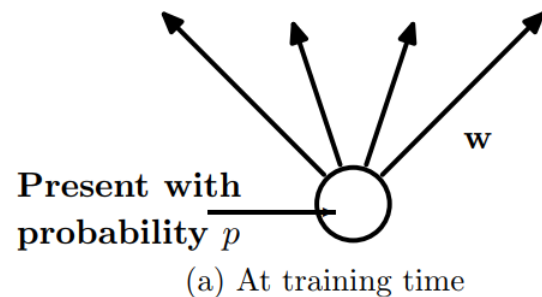
**Rozdílné chování v
trénovací a testovací fázi!**

- Model ensemble

- teoreticky v test. fázi forward např. 100x a zprůměrovat → pomalé
- chceme pouze jeden forward průchod → **v testu se dropout nedělá**

- Problém

- $s = w_1x_1 + w_2x_2 + w_3x_3$
- $p = 1/3 \rightarrow$ v průměru jedno w_i vypadne
- průměrná hodnota s bude o $1/3$ nižší → tomu se přizpůsobí váhy a aktivace
- bez dropoutu v testovací fázi je pak příliš velká “energie” výstupu do další vrstvy




Dropout v trénovací a testovací fázi

- Režim 1: základní (direct) dropout

- v testovací fázi vynásobíme výstup dropout pravděpodobností p


```
1 def train(x):
2     hidden = relu(np.dot(x, w1) + b1)
3     mask = np.random.rand(*hidden.shape) < 0.4
4     hidden[mask] = 0.
5     prob = np.dot(hidden, w2) + b2
6
7     # loss
8     # update gradientu
9
10 def predict(x):
11     hidden = relu(np.dot(x, w1) + b1)
12     hidden *= 0.4
13     prob = np.dot(hidden, w2) + b2
14
15     # argmax / klasifikace
```



- Režim 2: inverted dropout

- vyřešíme už v trénovací fázi, kde výstup vydělíme $1 - p$, aby měl stejnou “energii”, jako kdyby žádný dropout nebyl

```
1 def train(x):
2     hidden = relu(np.dot(x, w1) + b1)
3     mask = np.random.rand(*hidden.shape) < 0.4
4     hidden[mask] = 0.
5     prob = np.dot(hidden / (1 - 0.4), w2) + b2
6
7     # loss
8     # update gradientu
9
10 def predict(x):
11     hidden = relu(np.dot(x, w1) + b1)
12     prob = np.dot(hidden, w2) + b2
13
14     # argmax / klasifikace
```



testovací fáze pak nemusí být upravována
škáluje → vhodné použít ještě s další regularizací

Jak moc dropoutu?



- Optimální většinou 40-60 %, ale není pravidlem ☹️
- Nejlépe nahlížet jako na hyperparametr → křížová validace
- Při správném nastavení obvykle přinese cca 2 % accuracy navíc, někdy ale nic či dokonce zhorší
- Diskuze např. zde:
https://www.reddit.com/r/MachineLearning/comments/3oztvk/why_50_when_using_dropout/
<https://pgaleone.eu/deep-learning/regularization/2017/01/10/anaysis-of-dropout/>