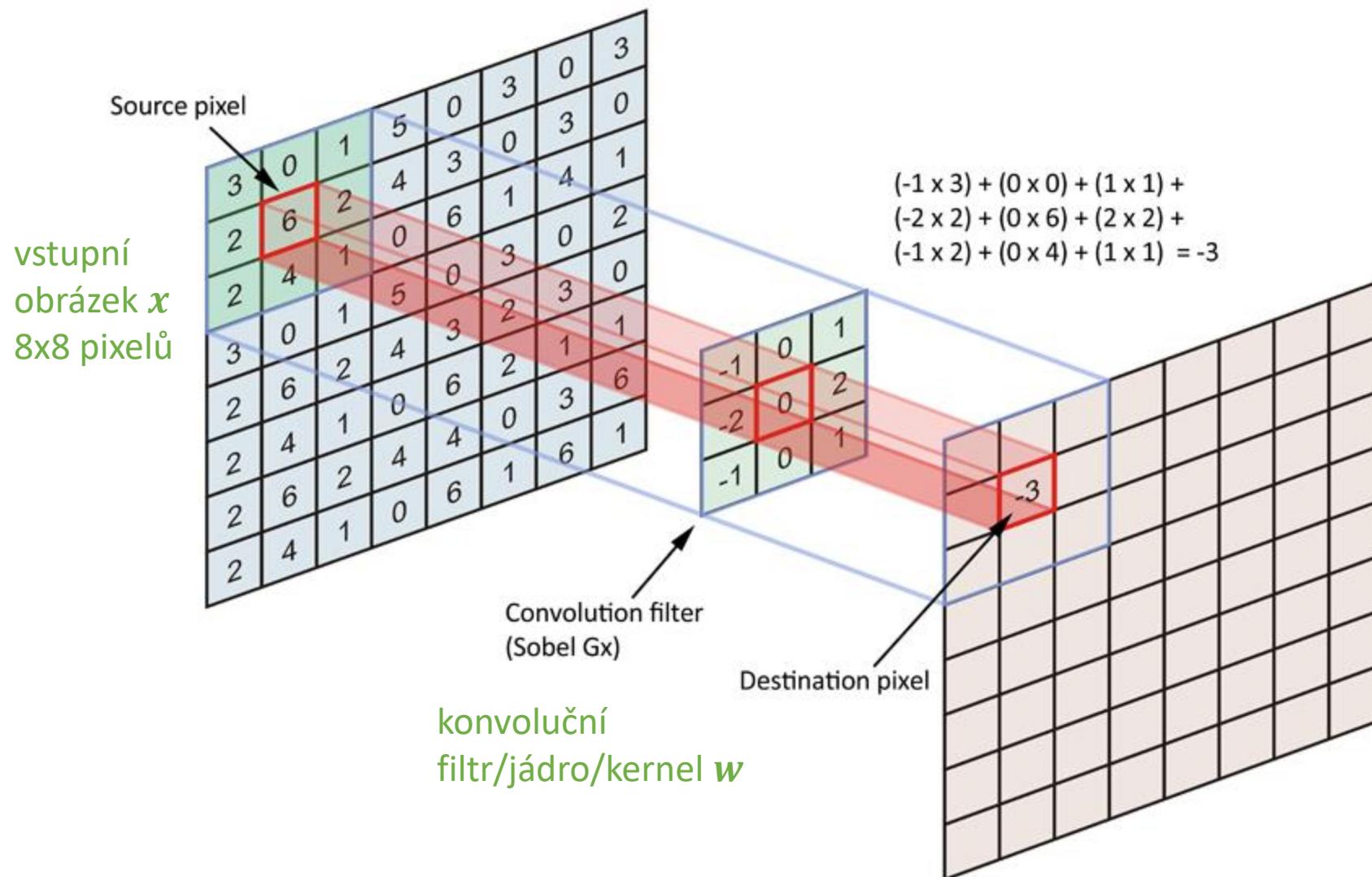


Aplikace neuronových sítí

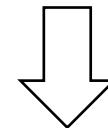
Konvoluční sítě

Dvourozměrná konvoluce



Pro každý výstupní pixel:

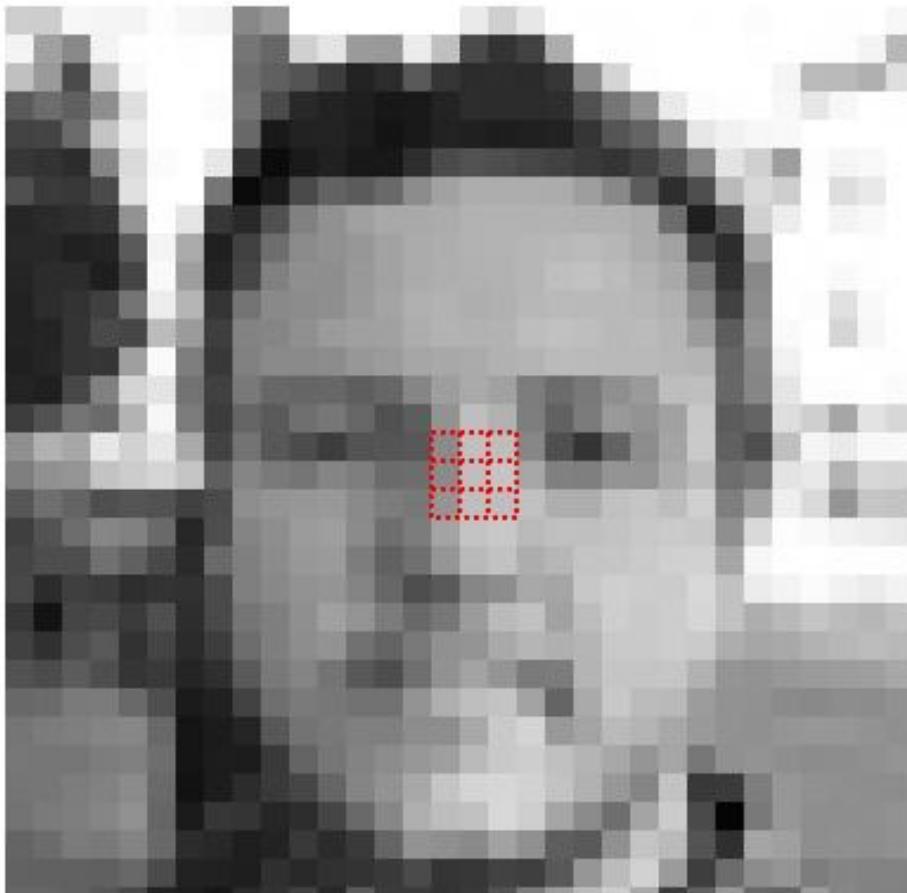
$$z_{uv} = \sum_{i=-k}^{+k} \sum_{j=-k}^{+k} w_{ij} x_{u+i, v+j} + b$$



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

výstupní obrázek z velikost závisí na nastavení konvoluce

Demo

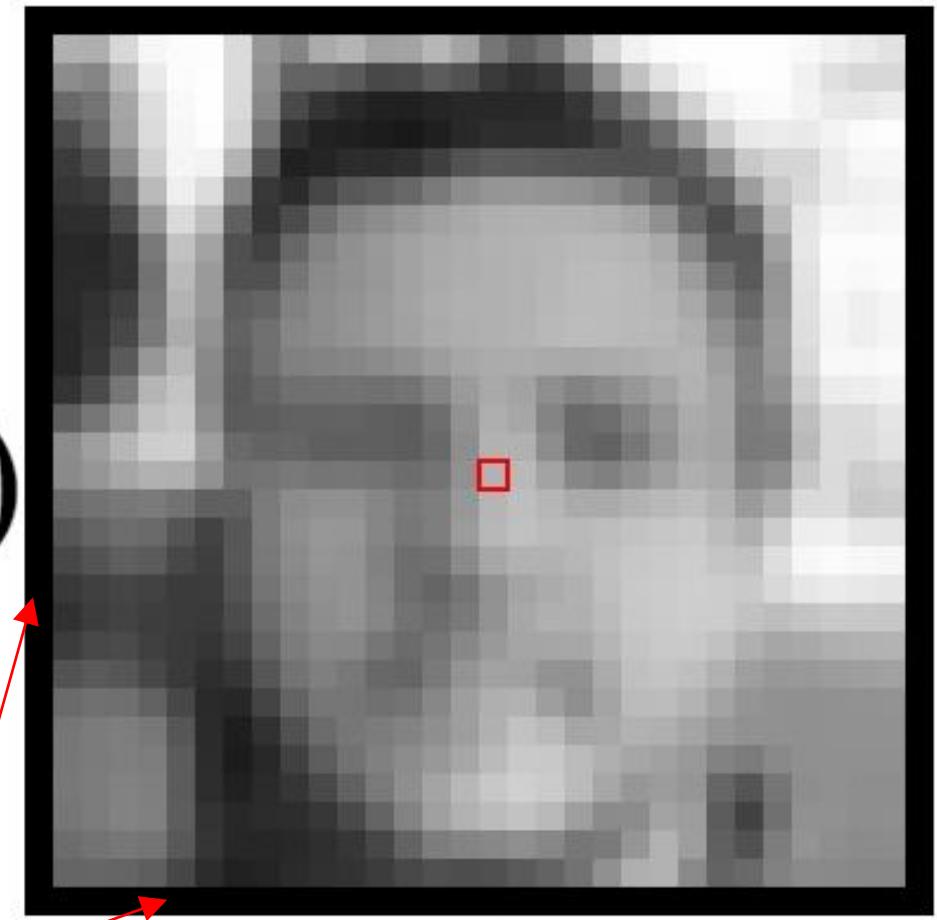


input image

$$\left(\begin{array}{ccc} 139 & + & 192 & + & 190 \\ \times 0.0625 & & \times 0.125 & & \times 0.0625 \\ \\ + & 139 & + & 191 & + & 197 \\ \times 0.125 & & \times 0.25 & & \times 0.125 \\ \\ + & 149 & + & 191 & + & 190 \\ \times 0.0625 & & \times 0.125 & & \times 0.0625 \end{array} \right) = 179$$

kernel:

blur ▾



output image

okraje?

Padding

Jak se vypořádat s okraji?

0 ₀	0 ₁	0 ₂	0	0	0	0
0 ₂	3 ₂	3 ₀	2	1	0	0
0 ₀	0 ₁	0 ₂	1	3	1	0
0	3	1	2	2	3	0
0	2	0	0	2	2	0
0	2	0	0	0	1	0
0	0	0	0	0	0	0

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

6.0	14.0	17.0
14.0	12.0	12.0
8.0	10.0	17.0

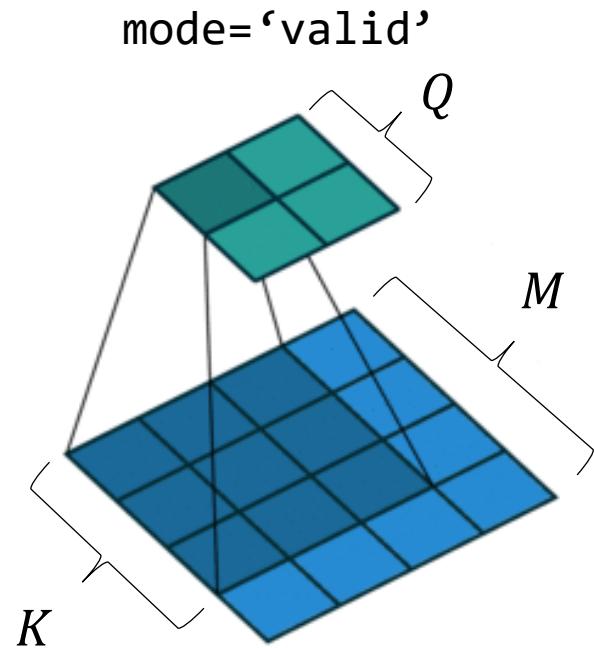
(výsledek je 3x3, protože
v příkladu stride=2)

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

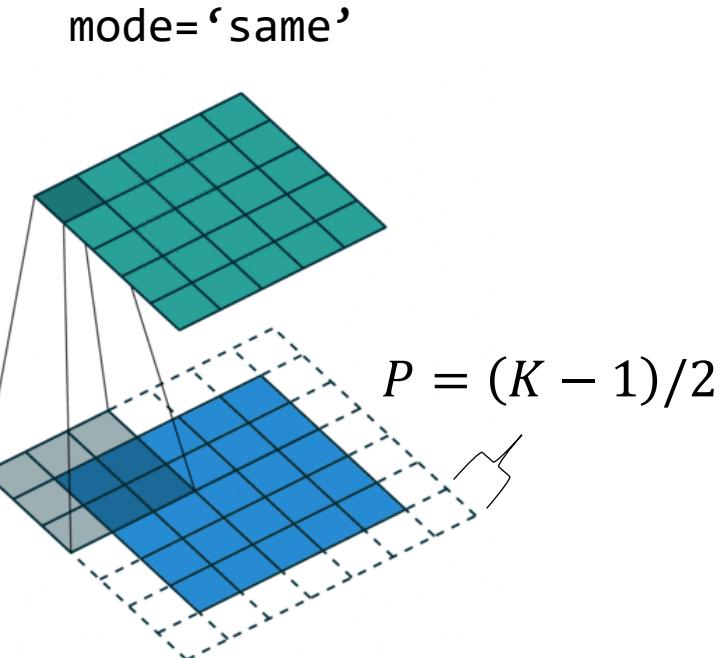
obrázky: https://github.com/vdumoulin/conv_arithmetic

Velikost výstupu v závislosti na paddingu

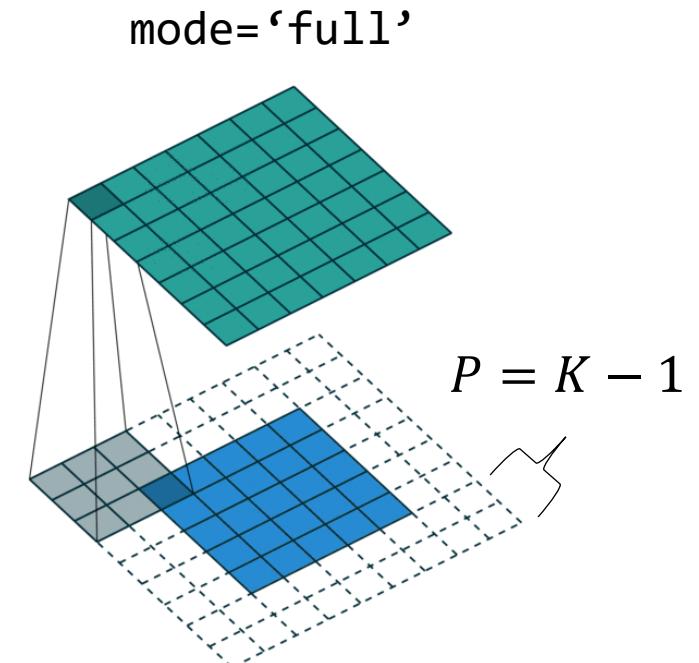
`scipy.signal.convolve2d(x, w, mode, ...)` → výstup $Q \times Q$



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



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



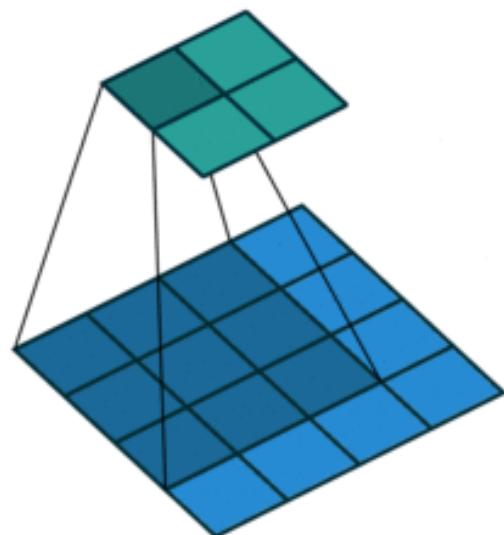
$$\begin{aligned} Q &= M - (K - 1) + 2P \\ &= M + (K - 1) \\ &= 5 + 3 - 1 \\ &= 7 \end{aligned}$$

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

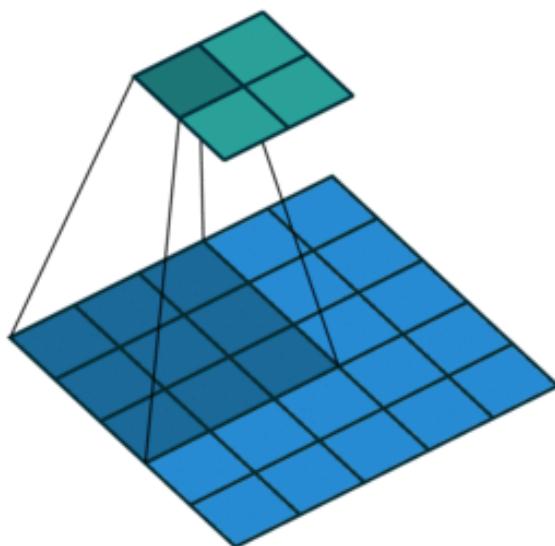
Velikost kroku

V anglické literatuře **stride**

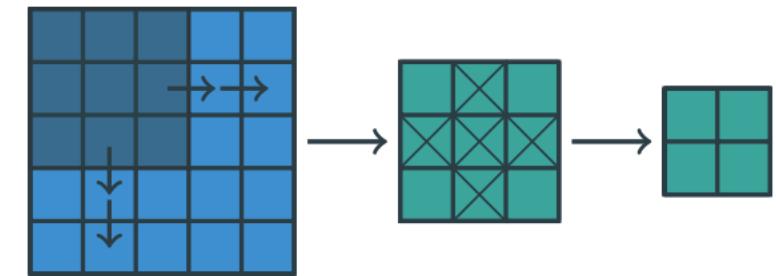
$$S = 1$$



$$S = 2$$



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

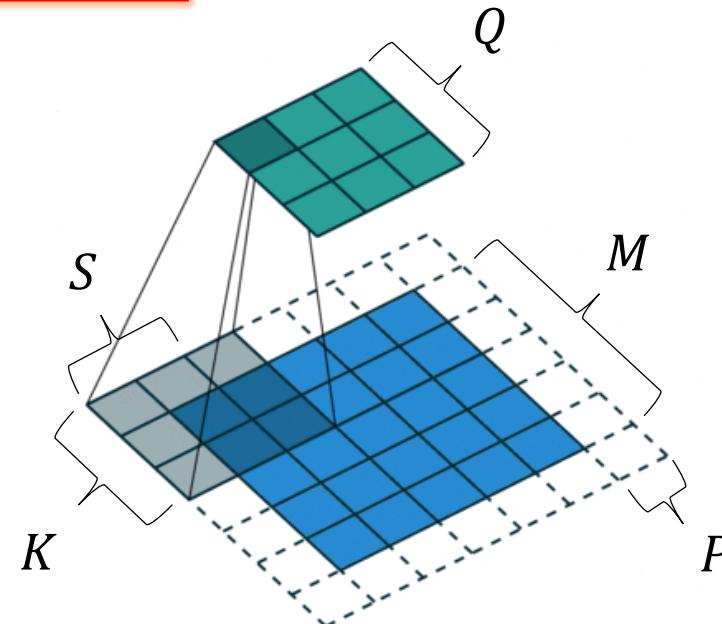
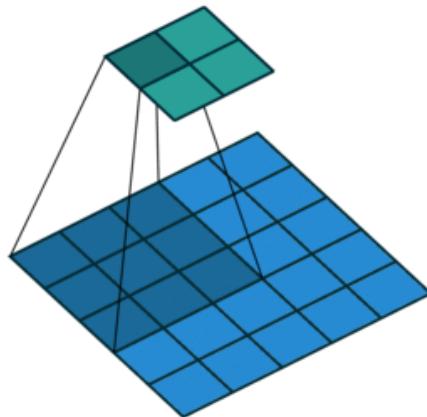


Vidíme tedy, že velikost výstupu se krokem S dělí
(mezery jsou ve výstupu)

Velikost výstupu konvoluce

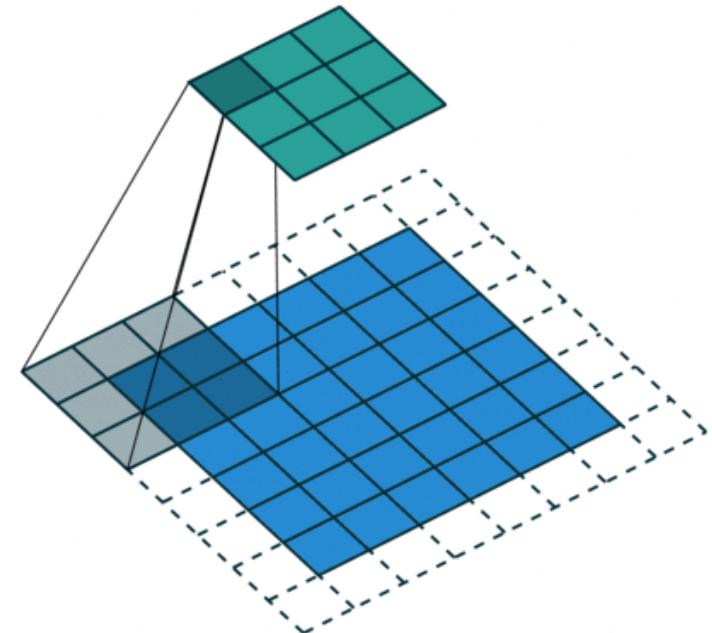
$$Q = \left\lceil \frac{M + 2P - K}{S} \right\rceil + 1$$

“Když to nevyjde hezky”:



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

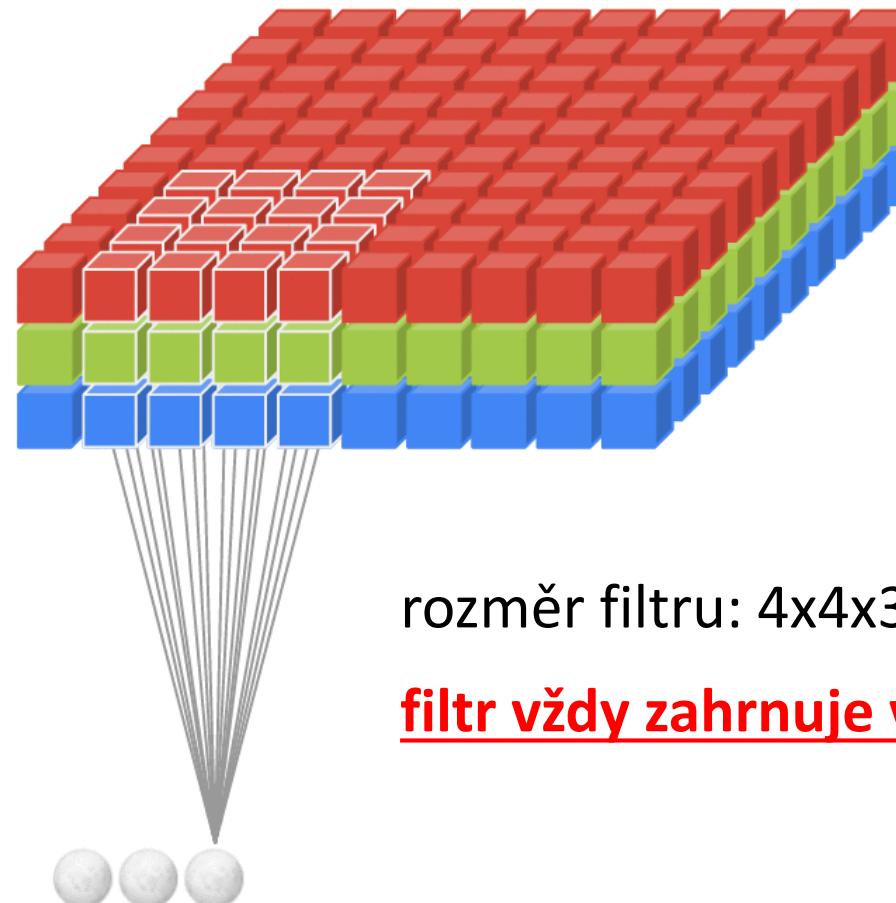
$$Q = \left\lceil \frac{5 + 2 \cdot 1 - 3}{2} \right\rceil + 1 = 3$$



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

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

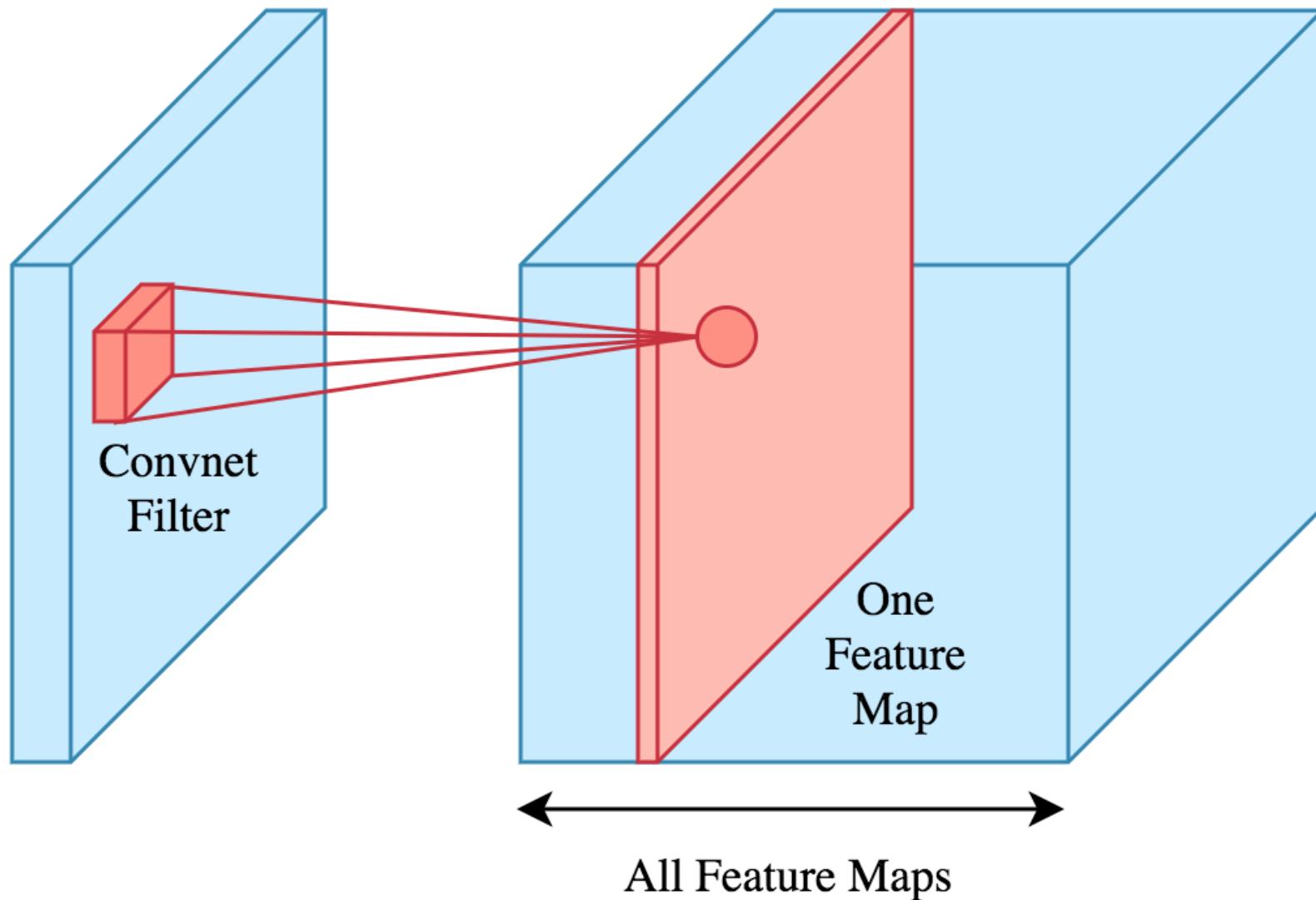
Vícekanálový vstup



rozměr filtru: $4 \times 4 \times 3$

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

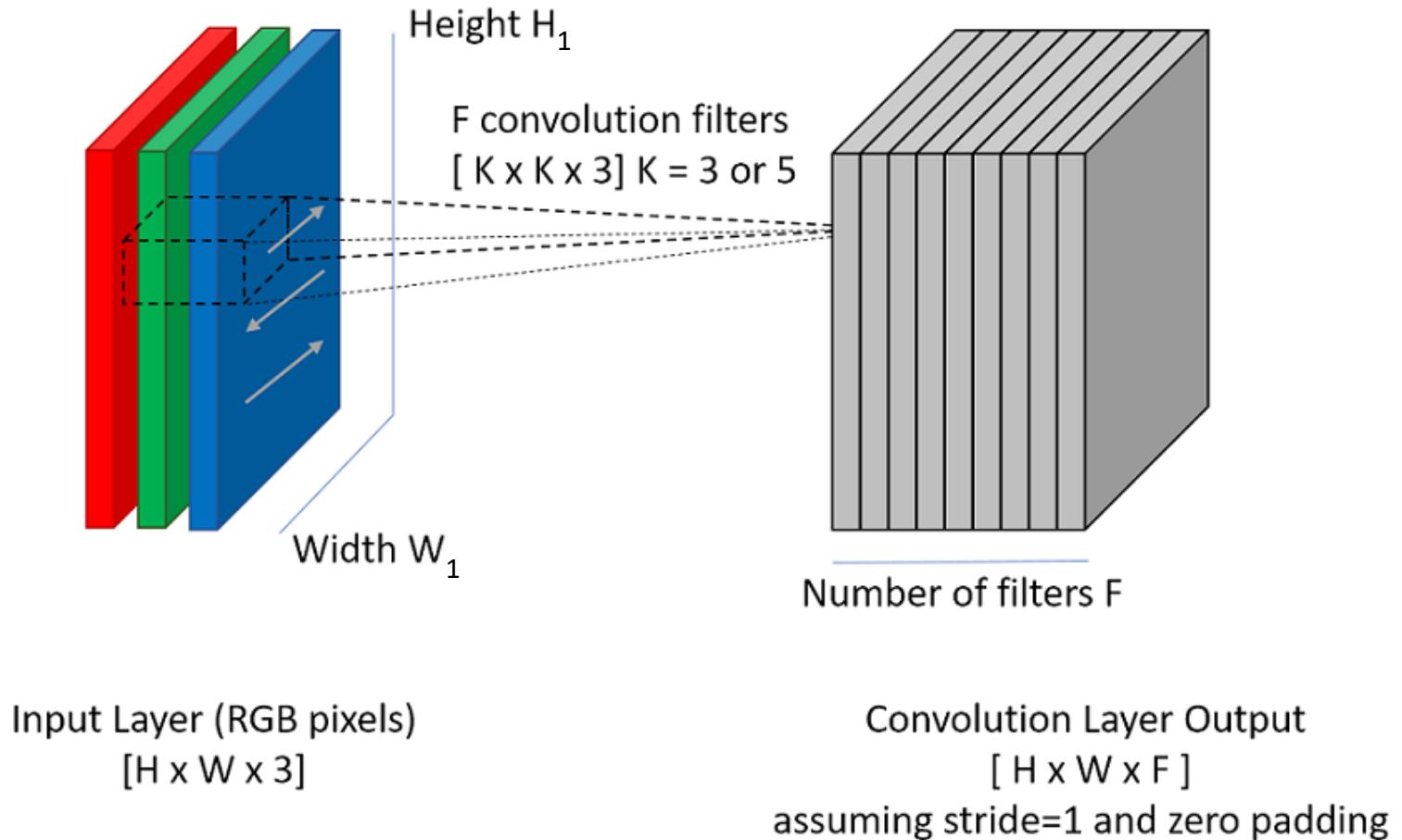
$$F$$

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



Příklad: RGB $32 \times 32 \times 3$, 10 5×5 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

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

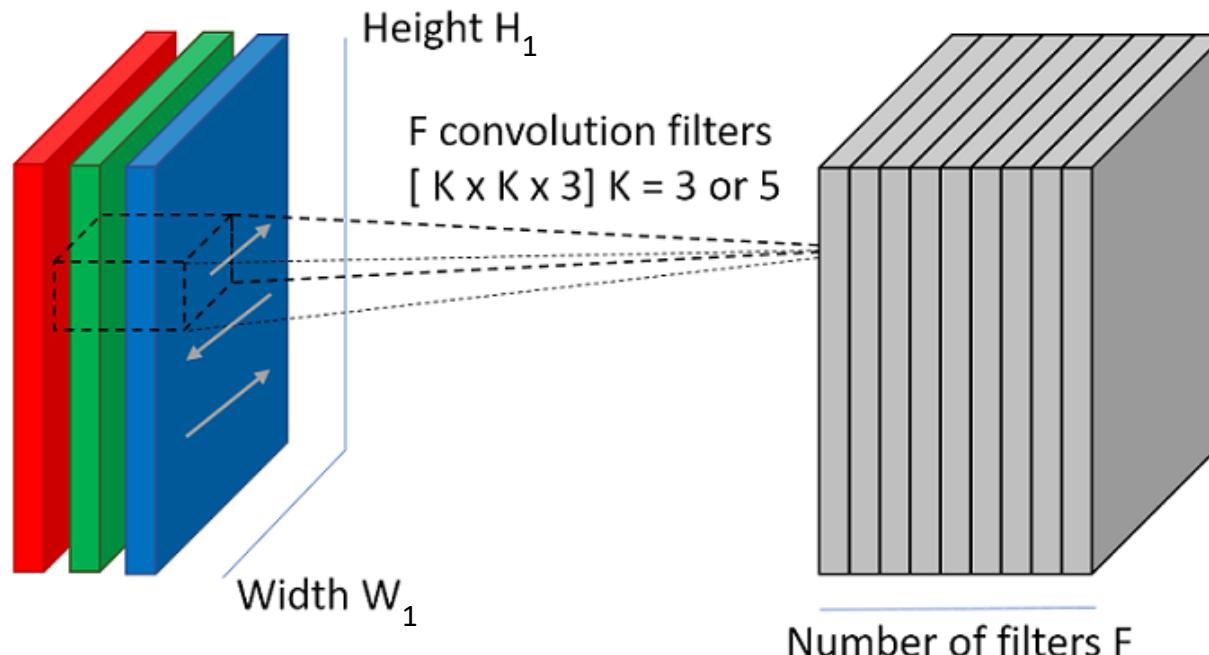
váhy

biasy

Vstup tvaru

$$32 \times 32 \times 3$$

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!



obrázek: <http://john-cd.com/blog/2017/03/08/Deep-Learning>

Výstup má rozměr

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

a hloubku

10 (počet filtrů)

Zpětný průchod konvoluce: gradient na váhy

- Dopředný průchod

$$z_{uv} = \sum_{i=1}^K \sum_{j=1}^K \sum_{c=1}^C w_{ijc} x_{su+i, sv+j, c} + b$$

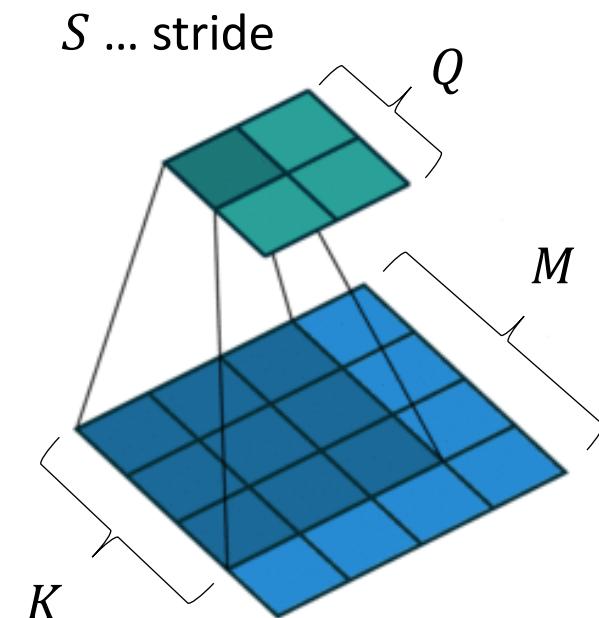
- Lokální gradient

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

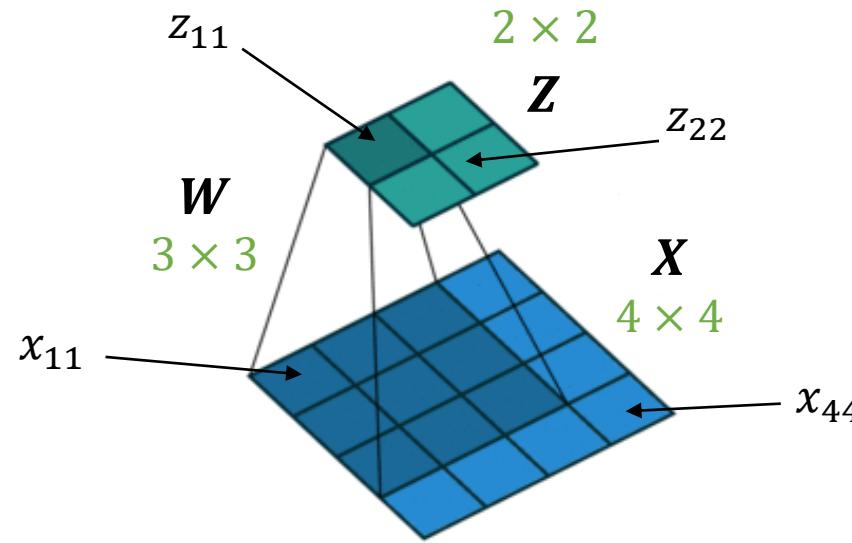
- Celkový gradient na váhy

$$\begin{aligned} \frac{\partial L}{\partial w_{ijc}} &= \sum_{u=1}^Q \sum_{v=1}^Q \frac{\partial L}{\partial z_{uv}} \cdot \frac{\partial z_{uv}}{\partial w_{ijc}} \quad \text{pokud } S > 1: \\ &= \sum_{u=1}^Q \sum_{v=1}^Q \overline{z_{uv}} \cdot x_{i+su, j+sv, c} \quad \text{"mezery" } \underline{\text{ve vstupu}} \end{aligned}$$

- Zpětný průchod na váhy je tedy rovněž konvoluce!



Konvoluce jako lineární vrstva



konvoluci na obrázku lze zapsat maticově:

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

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

... podobné lineární vrstvě

Zpětný průchod konvoluce: gradient na vstup

- Připomeňme, že pro lineární vrstvu

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

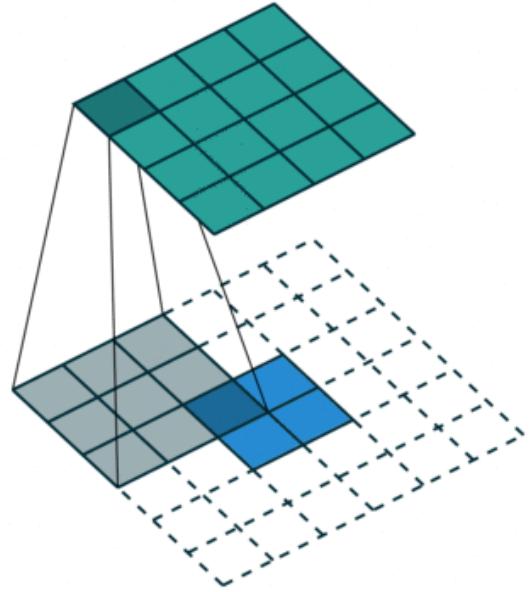
$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

je gradient na vstup

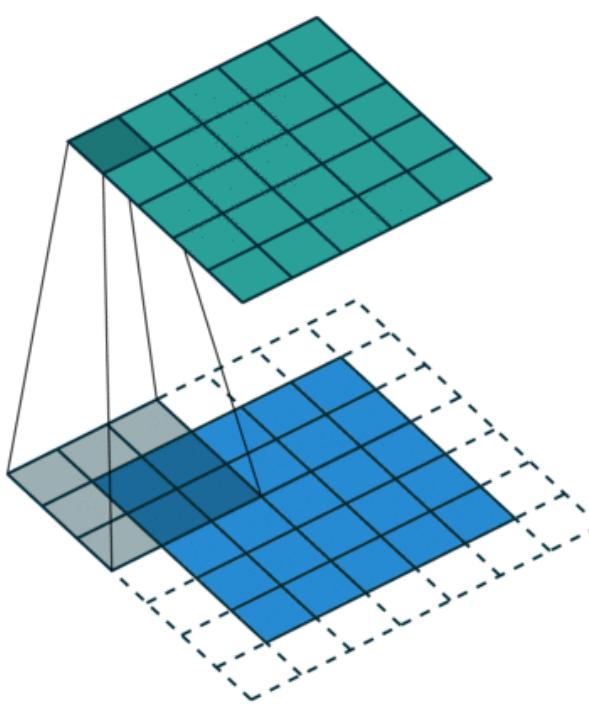
$$16 \times 1 \rightarrow 4 \times 4 \quad 16 \times 4 \quad 2 \times 2 \rightarrow 4 \times 1$$
$$\bar{\mathbf{x}} = \mathbf{W}^T \bar{\mathbf{z}}$$

- 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

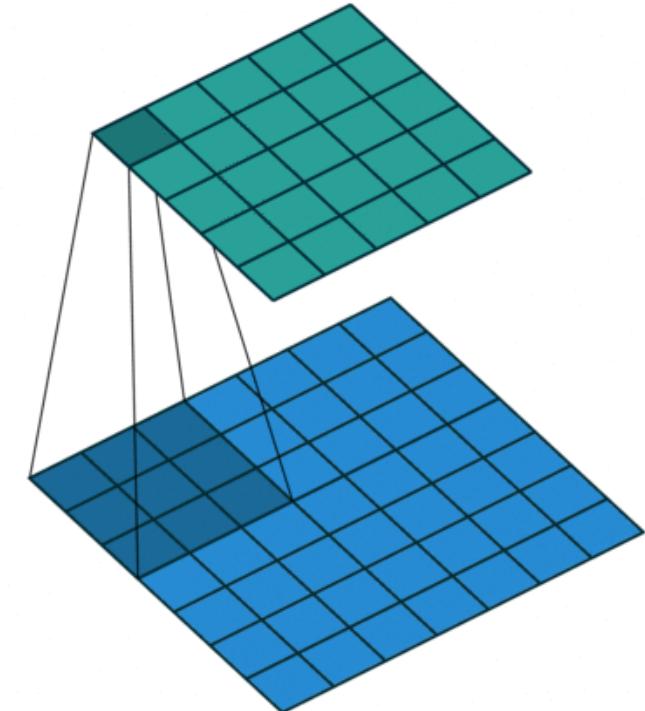
Ilustrace transponované konvoluce, stride=1



bez paddingu, stride=1

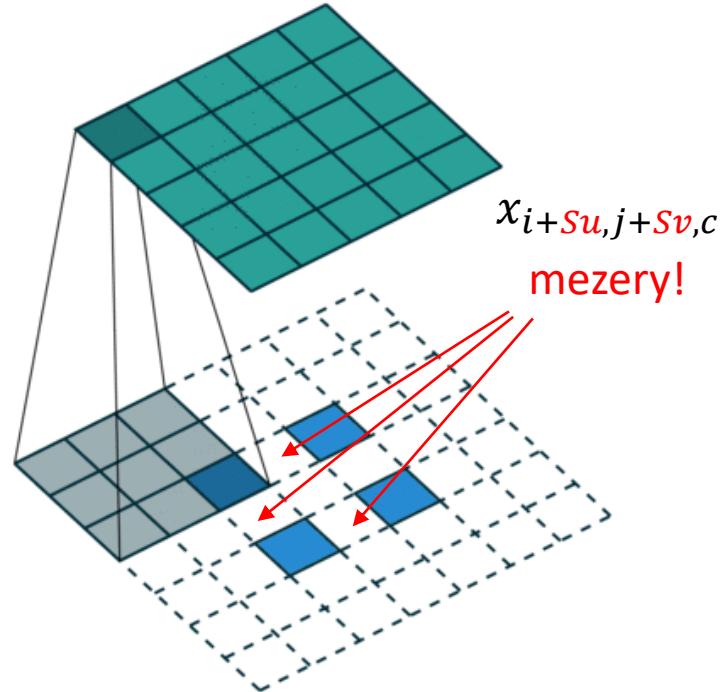


poloviční padding, stride=1

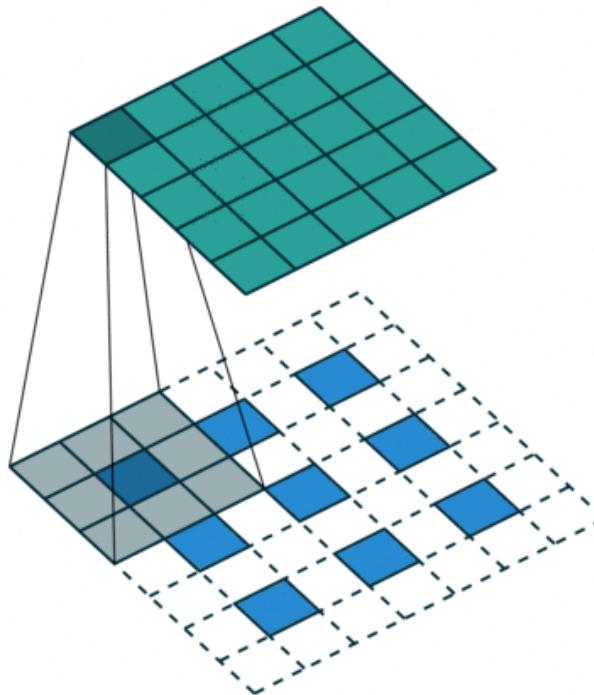


full padding, stride=1

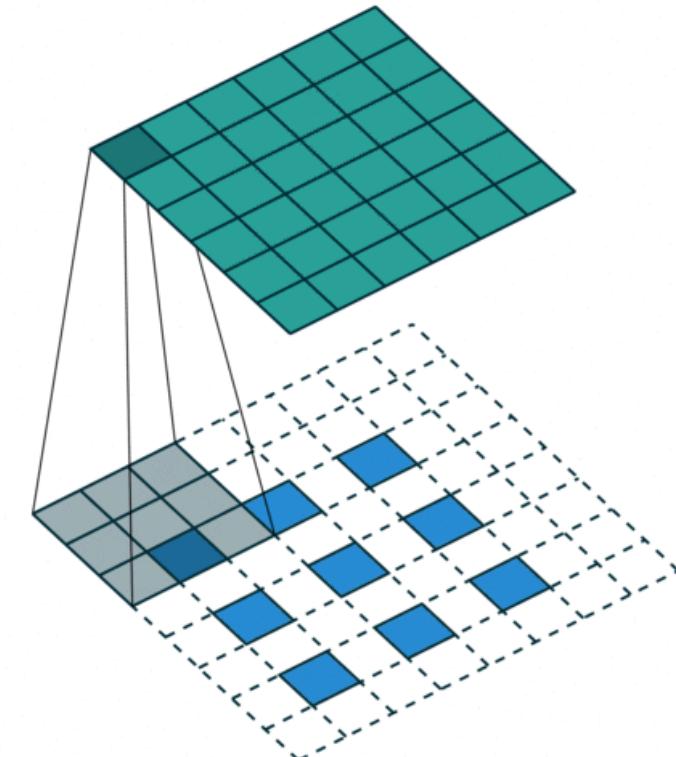
Illustrace transponované konvoluce, stride > 1



bez paddingu, stride=2



poloviční padding, stride=2

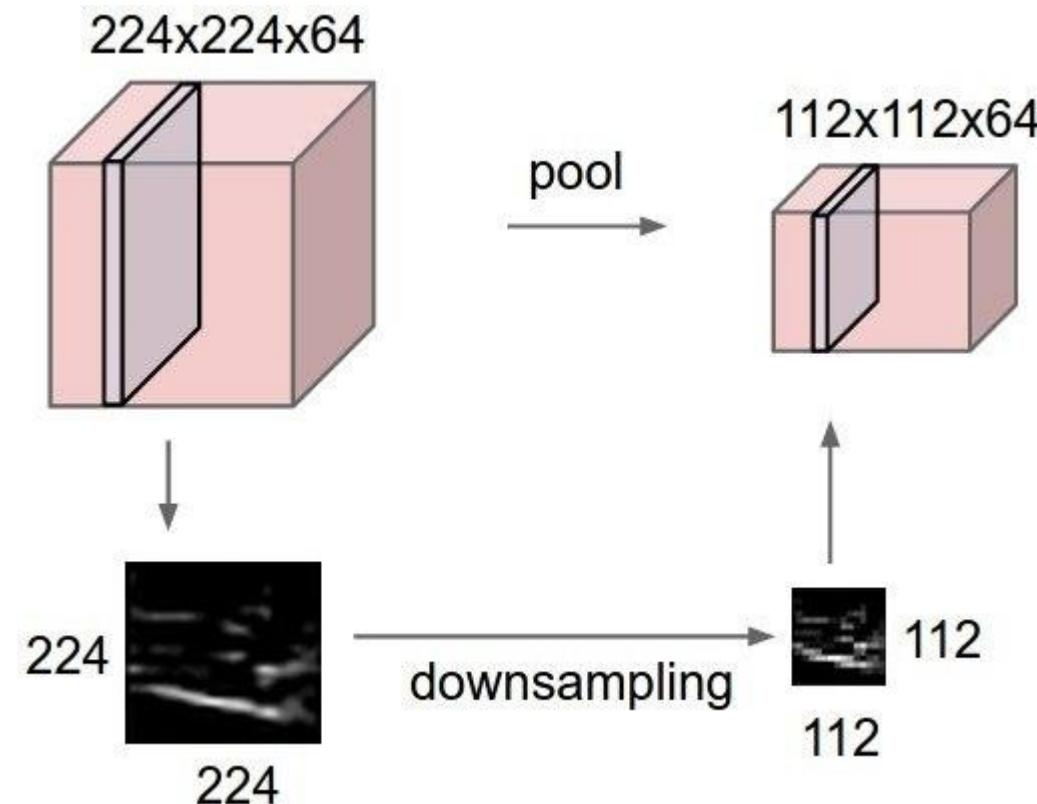


poloviční padding, stride=2
"když to nevyjde hezky"

mezery → anglický název **fractionally strided convolution**

Pooling

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



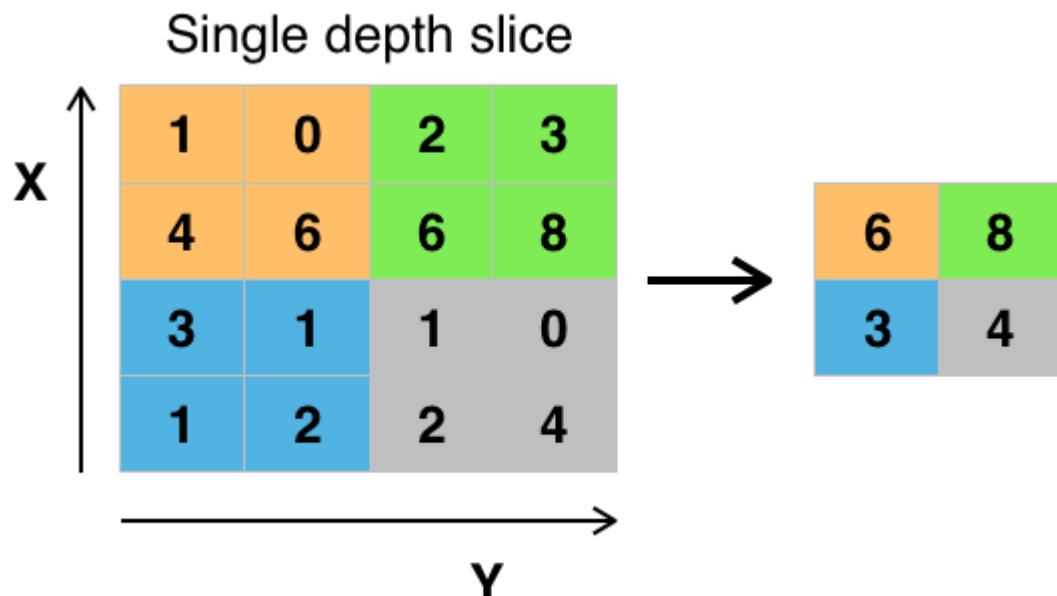
obrázek: <http://cs231n.github.io/>

Max pooling

- Nejčastější forma poolingu
- Robustní vůči malému posunu vstupu

Počet parametrů: 0

např. 2x2 max pooling, stride=2:



výstup:

maximum přes každé okénko

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

příklad:

vstup: 32x32x3

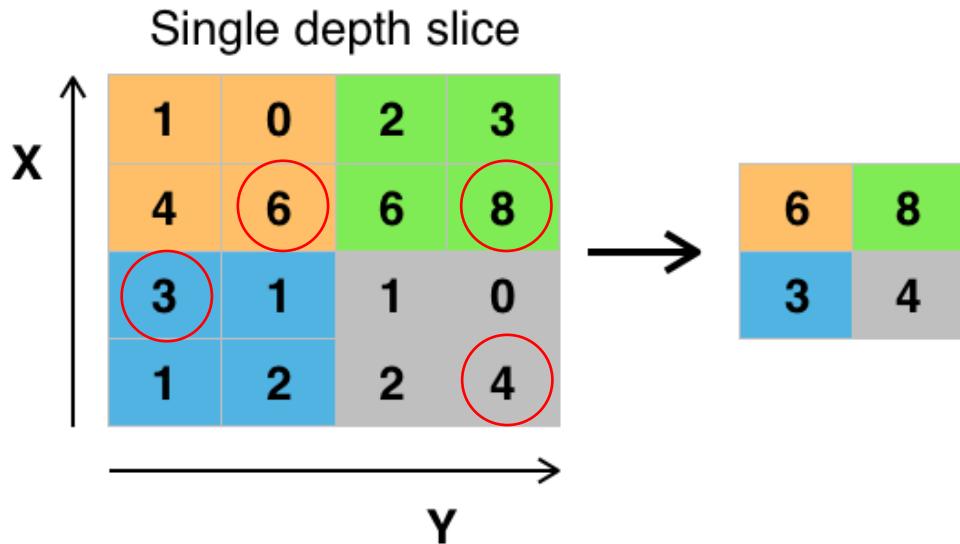
výstup: 16x16x3

Max pooling: zpětný průchod

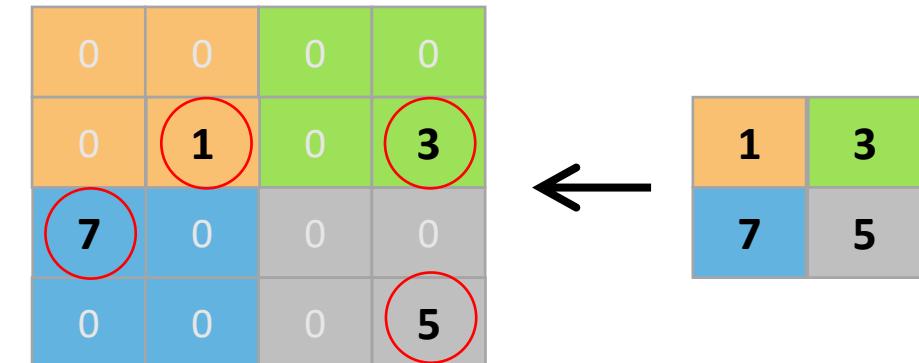
funkce max zapsána jinak: $z = \begin{cases} x_1 & \text{pokud } x_1 \geq x_i \forall x_i \in x \\ & \vdots \\ x_D & \text{pokud } x_D \geq x_i \forall x_i \in x \end{cases}$ **výběr prvku z pole**

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

dopředný průchod

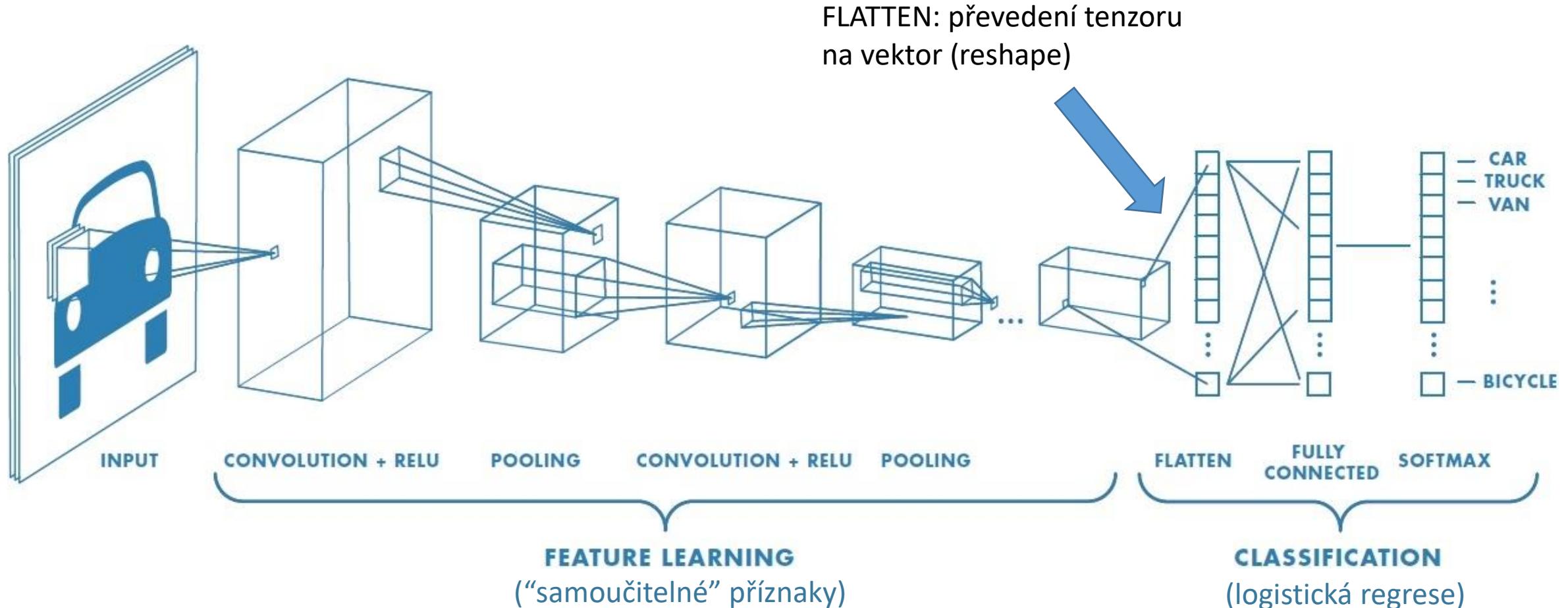


zpětný průchod



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 číslovek: LeNet-5 (1998)

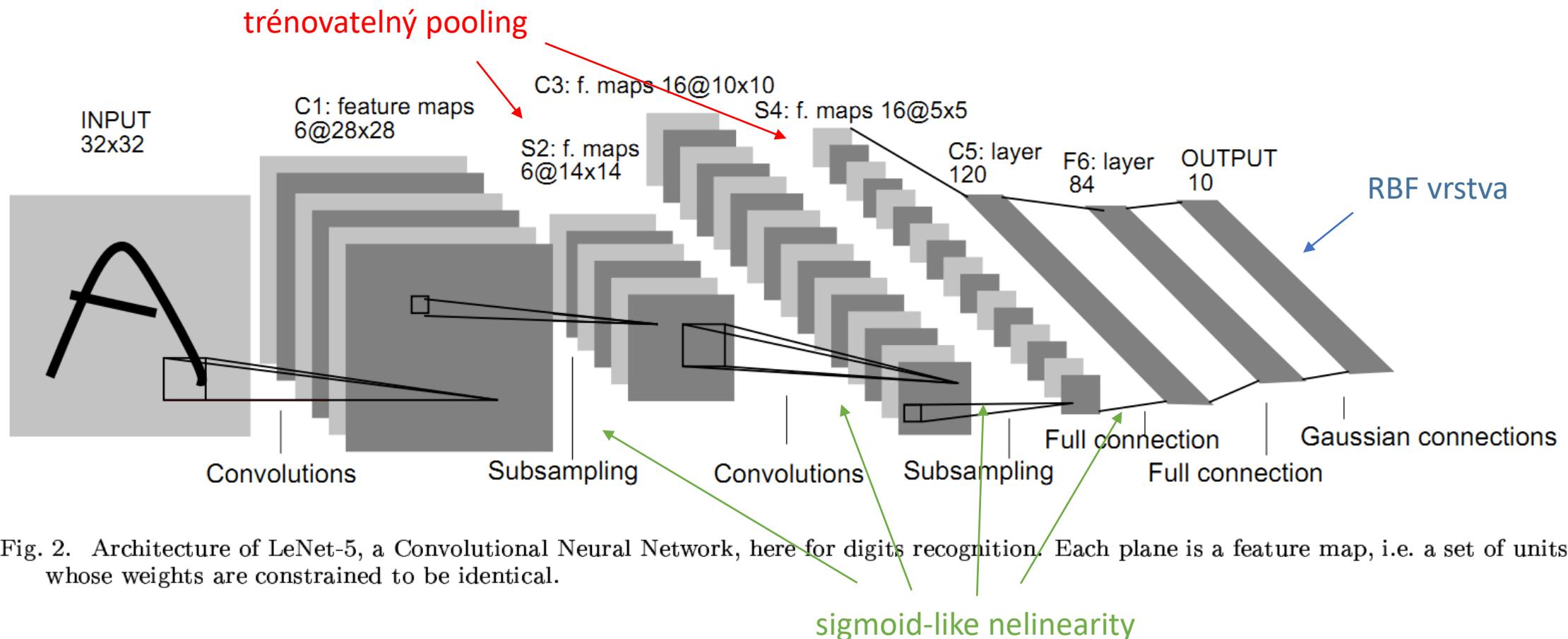


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

FC ... Fully Connected

Rozpoznávání MNIST číslovek: LeNet-5 (1998)

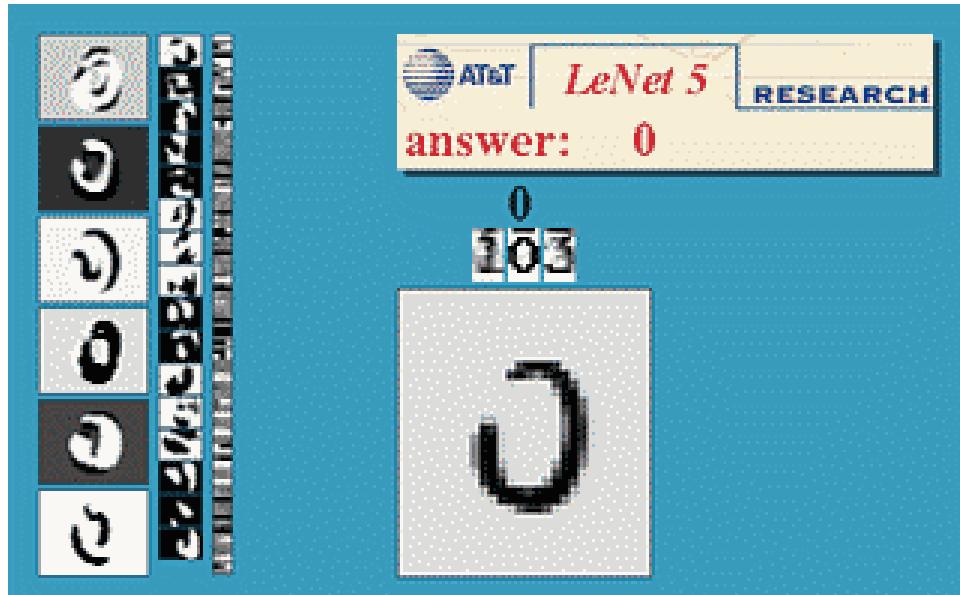


Fig. 4. Size-normalized examples from the MNIST database.
MNIST ... 60000 obrázků číslovek

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

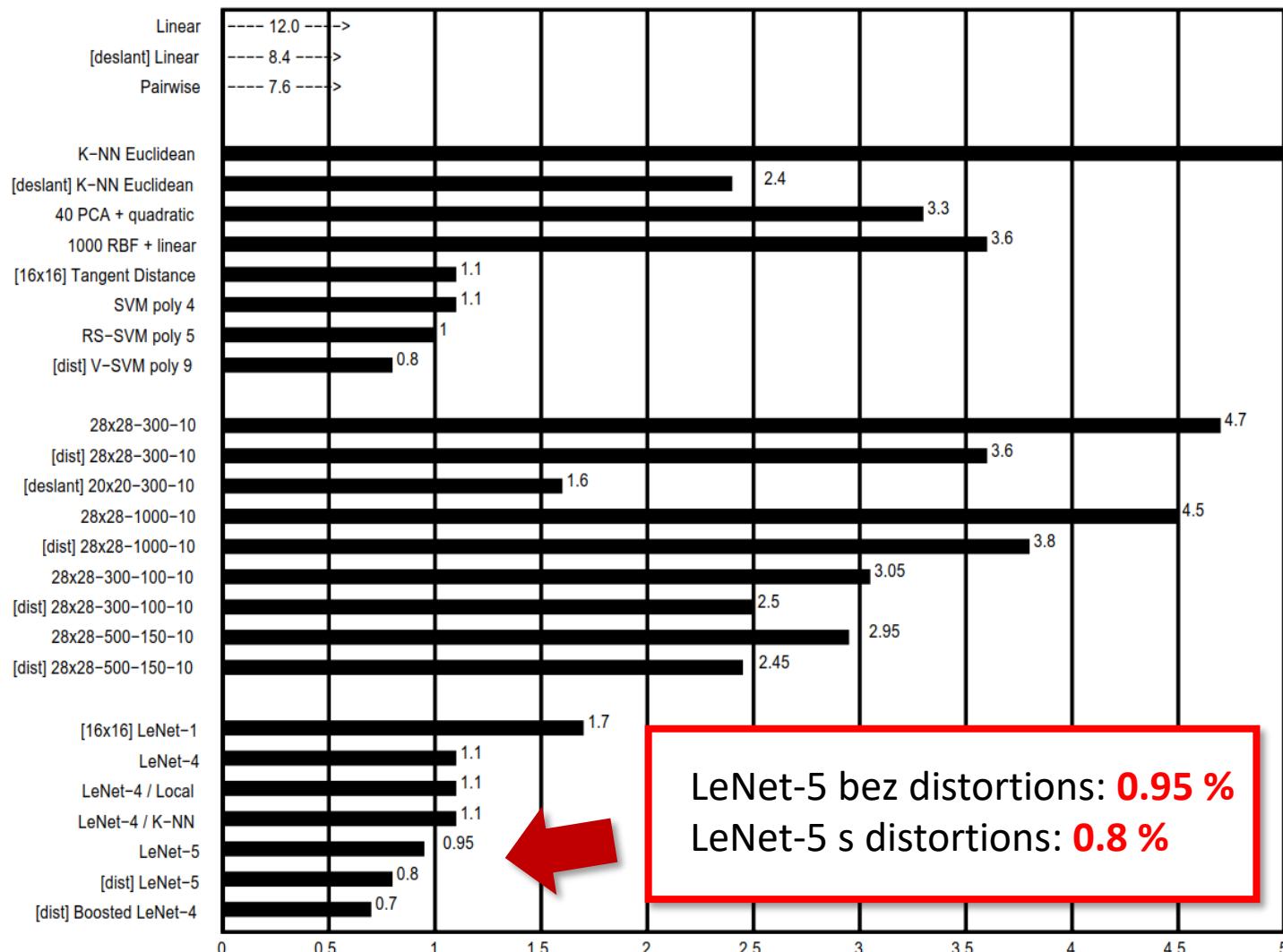
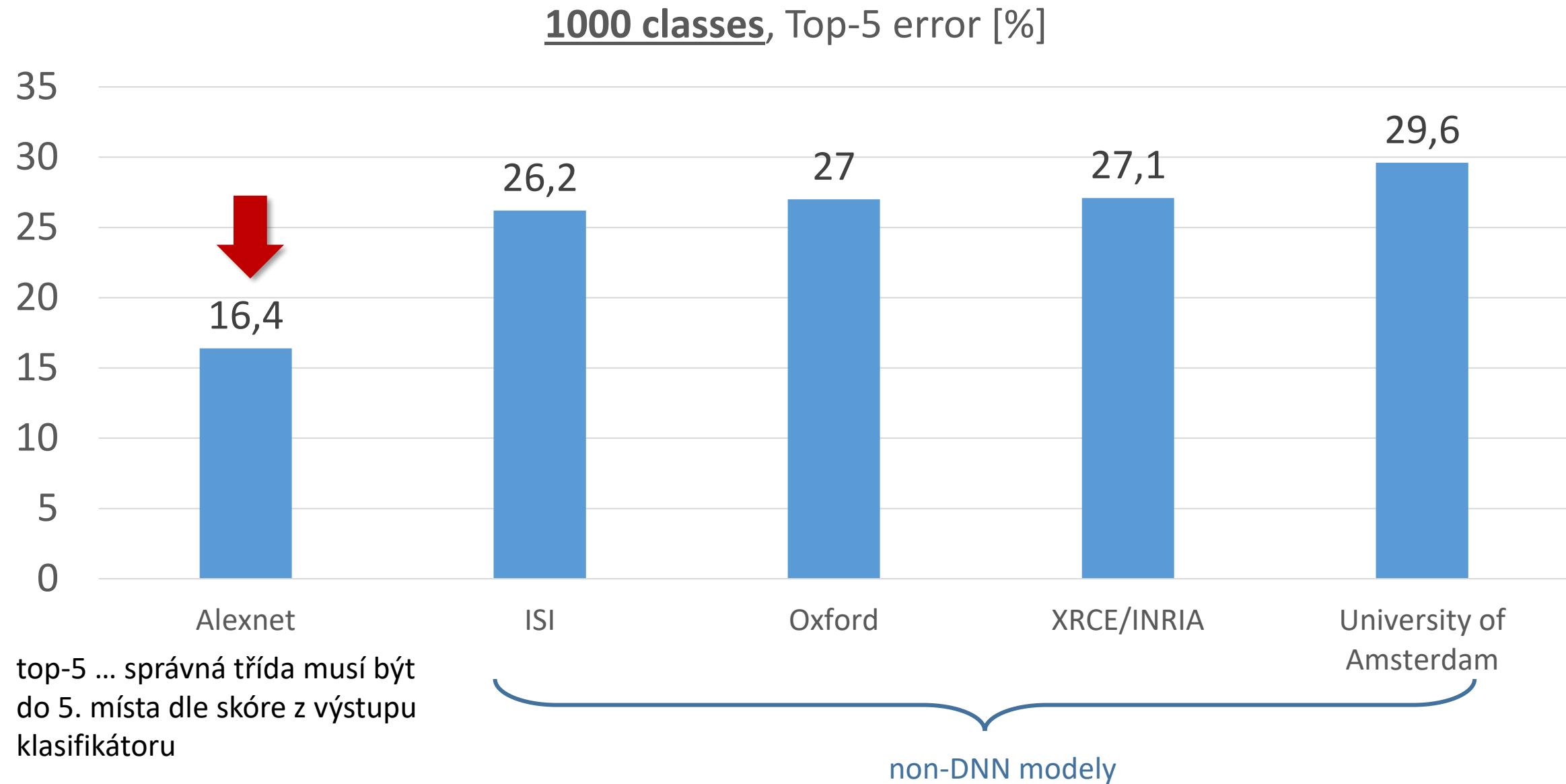
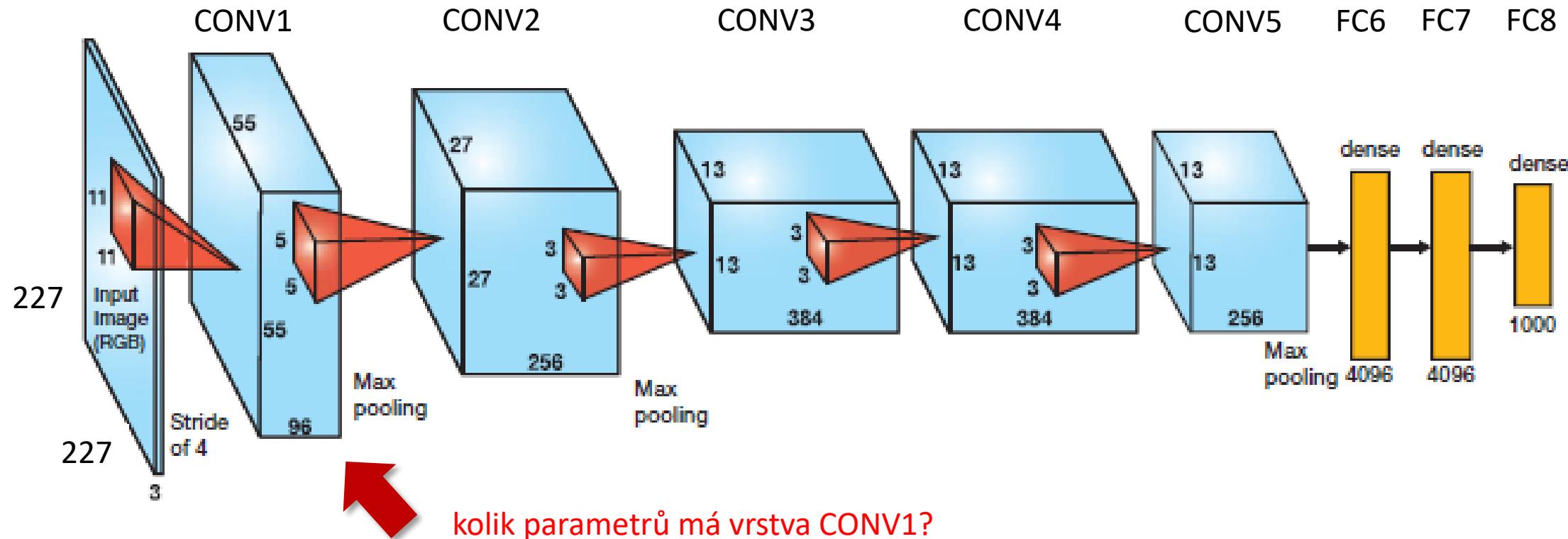


Fig. 9. Error rate on the test set (%) for various classification methods. [deslant] indicates that the classifier was trained and tested on the deslanted version of the database. [dist] indicates that the training set was augmented with artificially distorted examples. [16x16] indicates that the system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

Rozpoznávání ImageNet: Alexnet (2012)

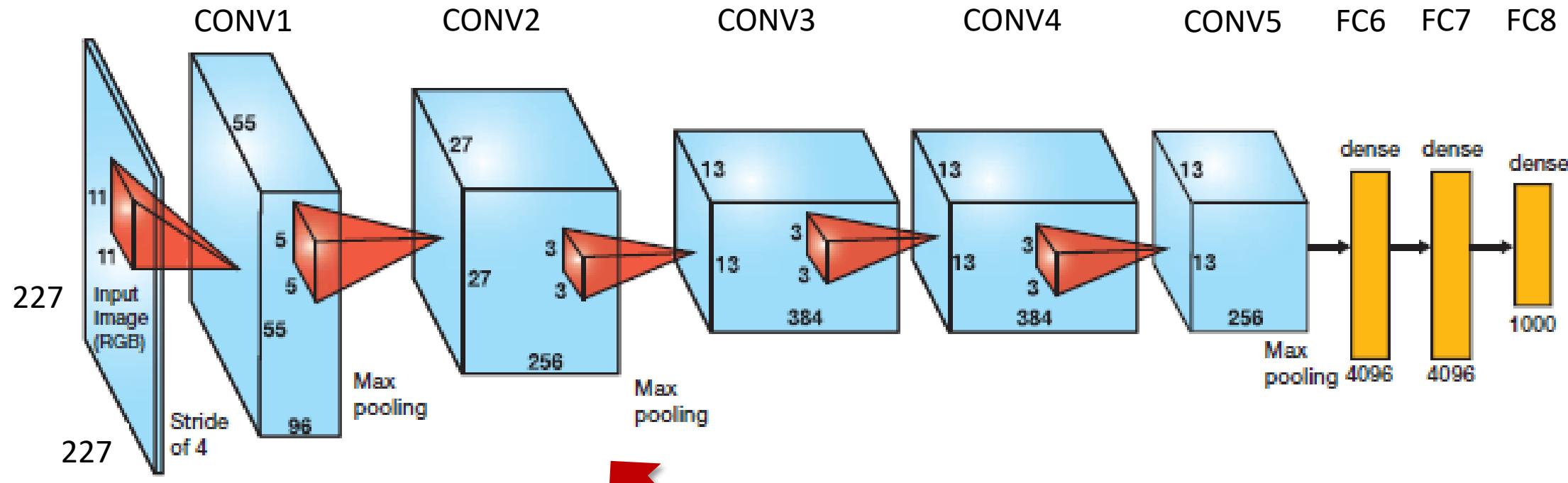


Alexnet (2012)



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

Alexnet (2012)

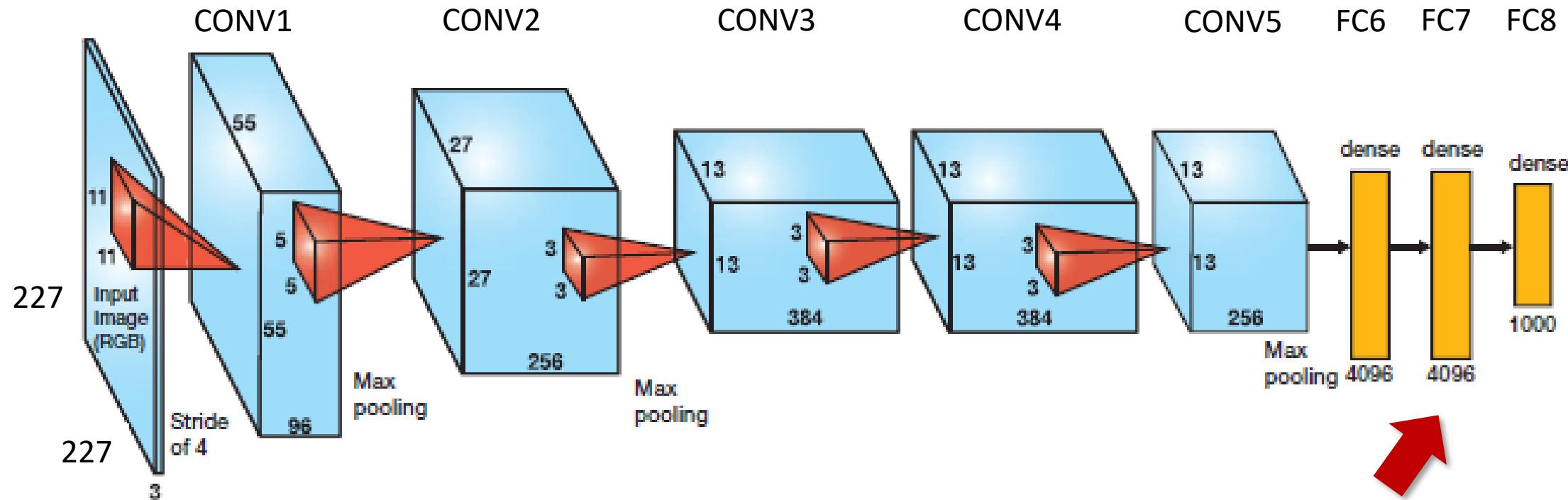


kolik parametrů má vrstva CONV2?

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

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

Alexnet (2012)



kolik parametrů má vrstva FC7?

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

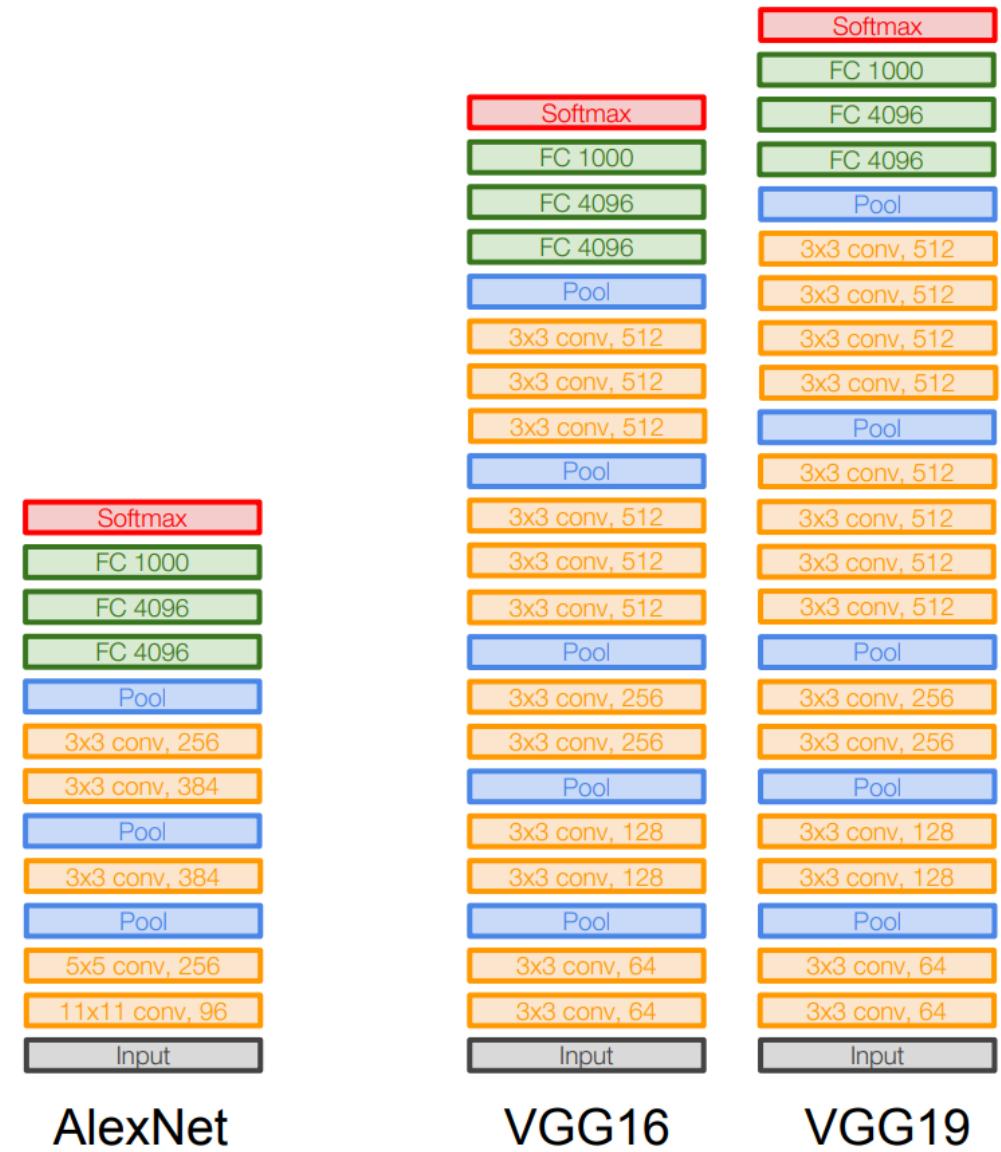
- architektura: CONV-POOL-NORM-CONV-POOL-NORM-CONV-CONV-FC-FC-FC
- “naškálovaná” LeNet-5

Alexnet (2012)

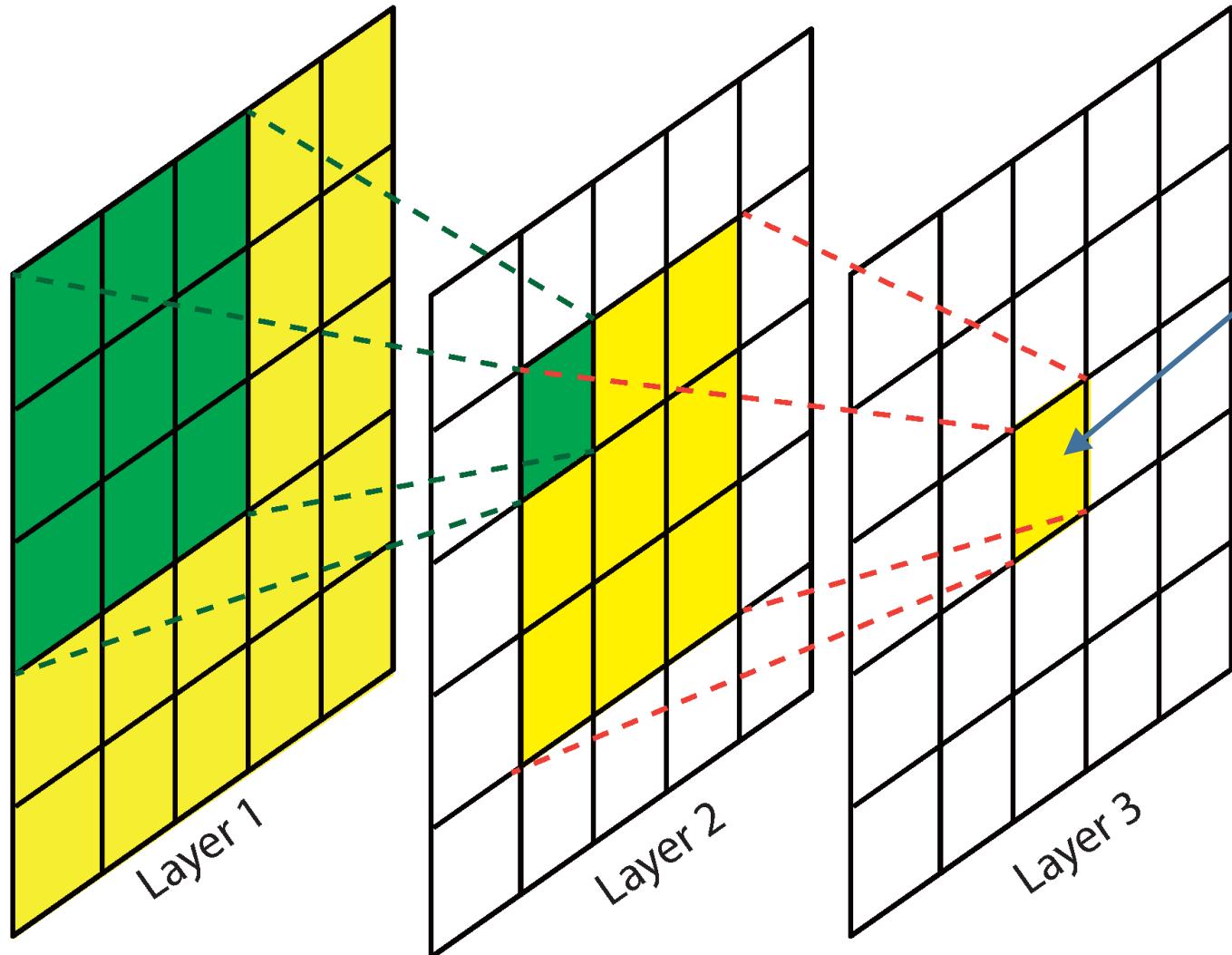
- Krizhevsky, Sutskever, Hinton: “ImageNet Classification with Deep Convolutional Neural Networks”
- Sít, která “nastartovala DNN/CNN revoluci”
- Autoři nevyvinuli žádný nový algoritmus, “pouze” ukázali, jak správně CNN používat
- Místo sigmoid aktivací přechod na ReLU
- Kromě klasické L2 regularizace 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 3×3 nad
druhou vrstvou závisí na 5×5
oblasti v první vrstvě

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

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

VGG (2014)

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

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

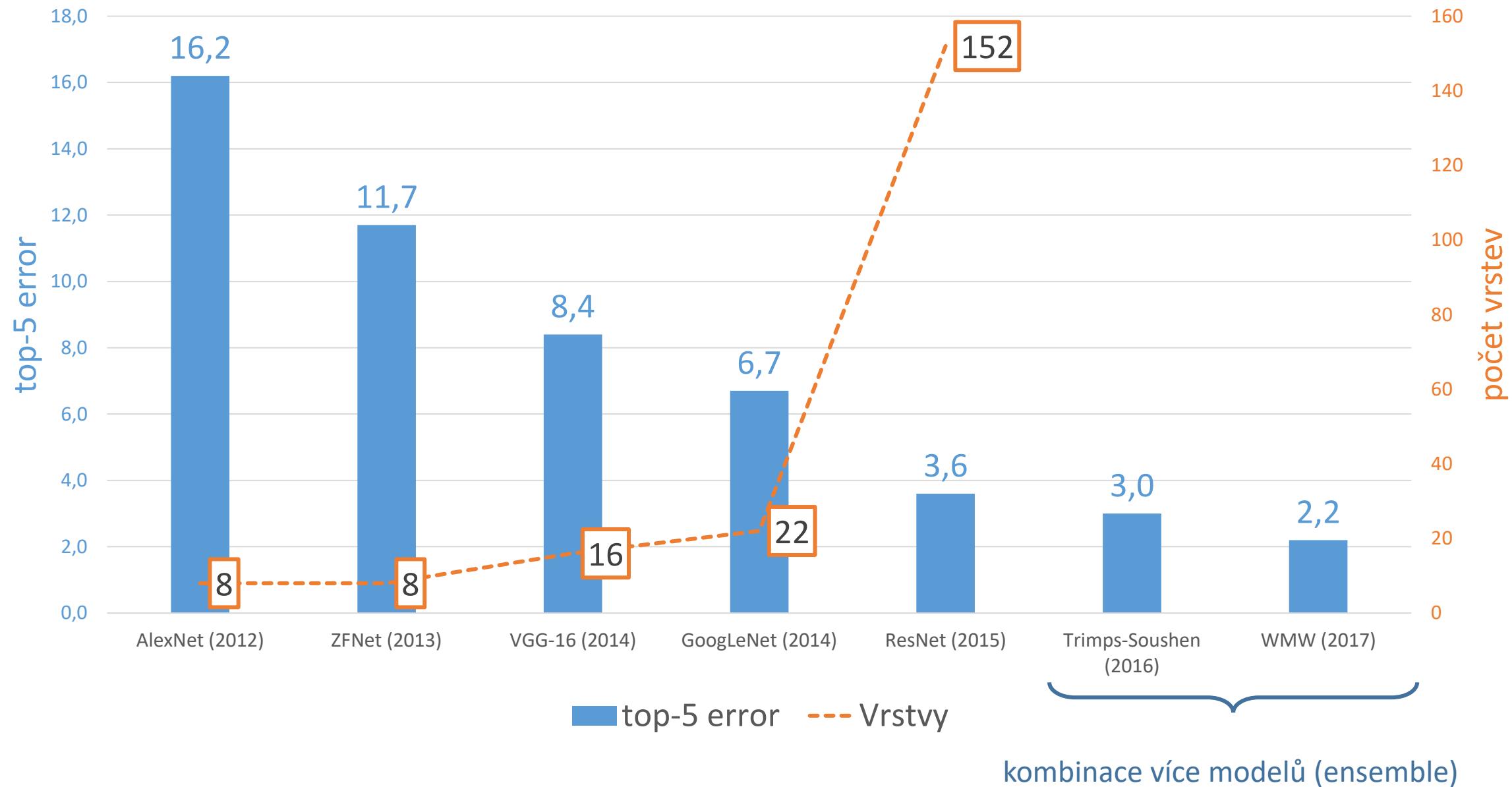
vs

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

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



ImageNet klasifikace



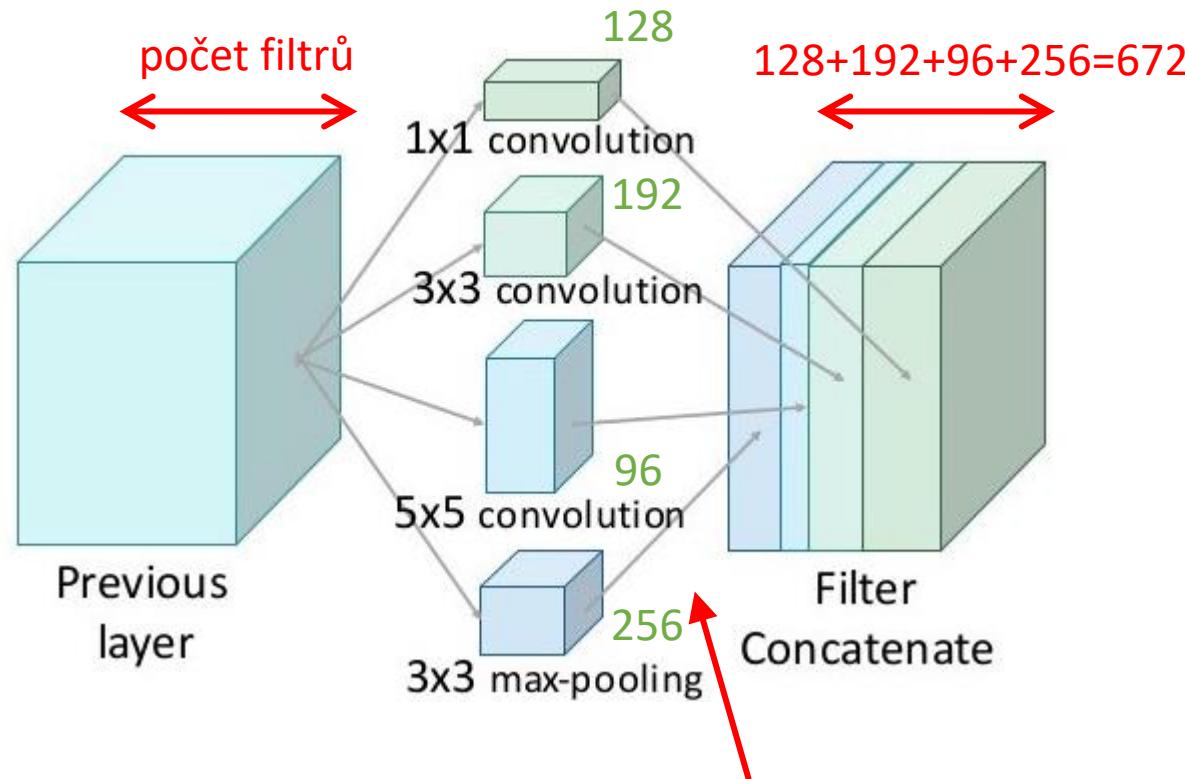
GoogLeNet (2014)

- Szegedy et al.: Going Deeper with Convolutions
- Navrženo s ohledem na výpočetní náročnost a celkový počet parametrů
- Skládá se z tzv. **Inception** modulů, které kombinují více typů konvolucí v jedné vrstvě (vychází z Lin et al.: “Network in network”)



obrázek: <http://knowyourmeme.com/memes/we-need-to-go-deeper>

Inception modul v1

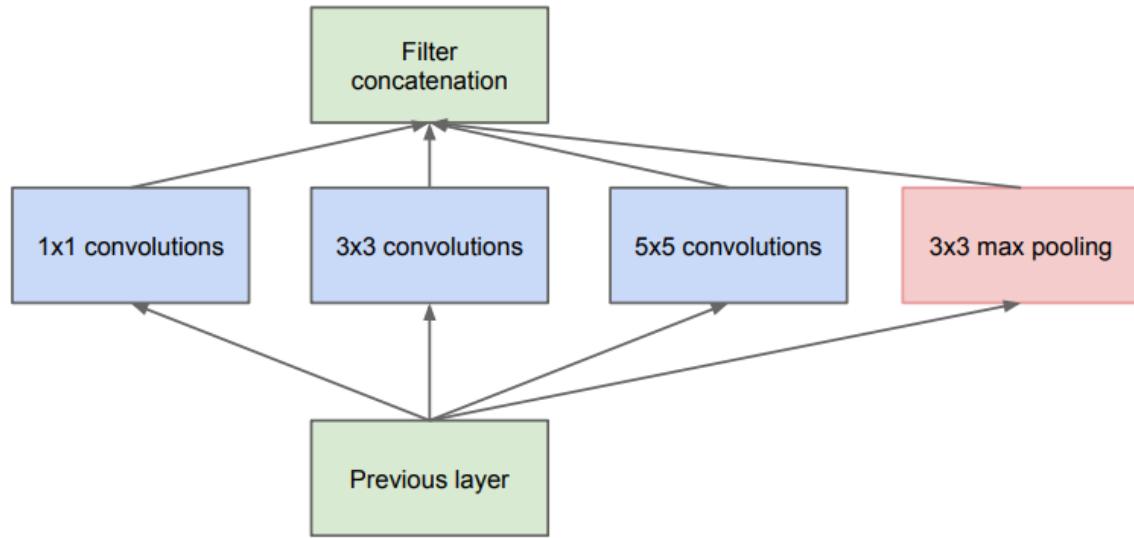


zero padding tak, aby výsledek konvoluce
měl vždy stejnou velikost (mode='same')

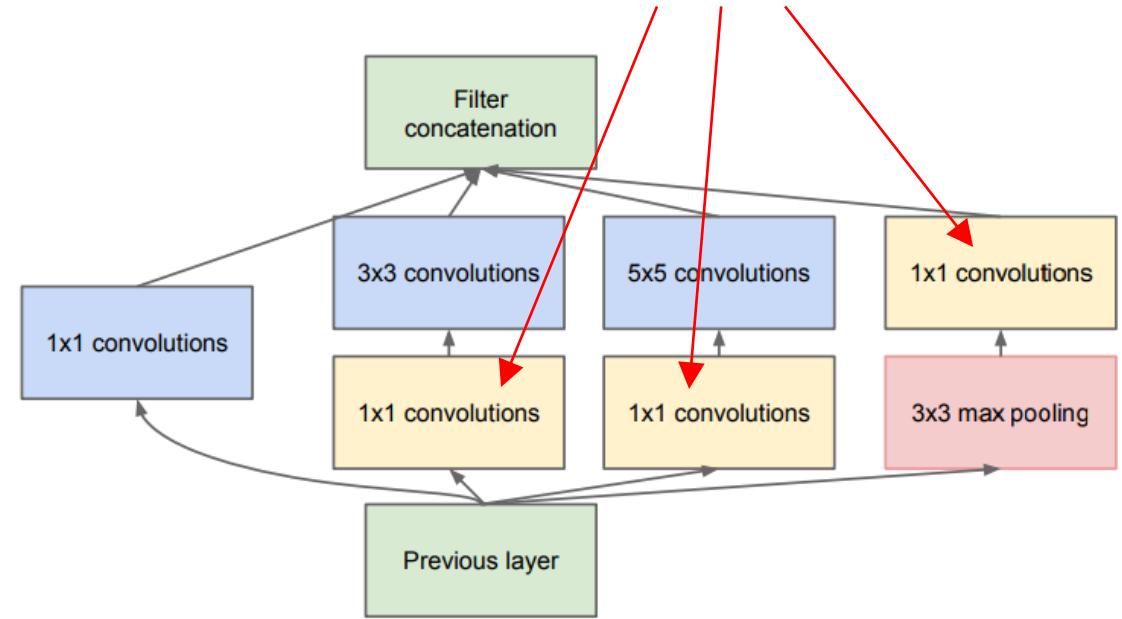
Optimalizace Inception modulu

[Szegedy et al.: Going Deeper with Convolutions](#)

1x1 “bottleneck” vrstvy s méně filtry: redukují dimenze a urychlují



(a) Inception module, naïve version



(b) Inception module with dimension reductions

Figure 2: Inception module

obrázek: <https://arxiv.org/abs/1409.4842>

ve skutečnosti použita tato varianta

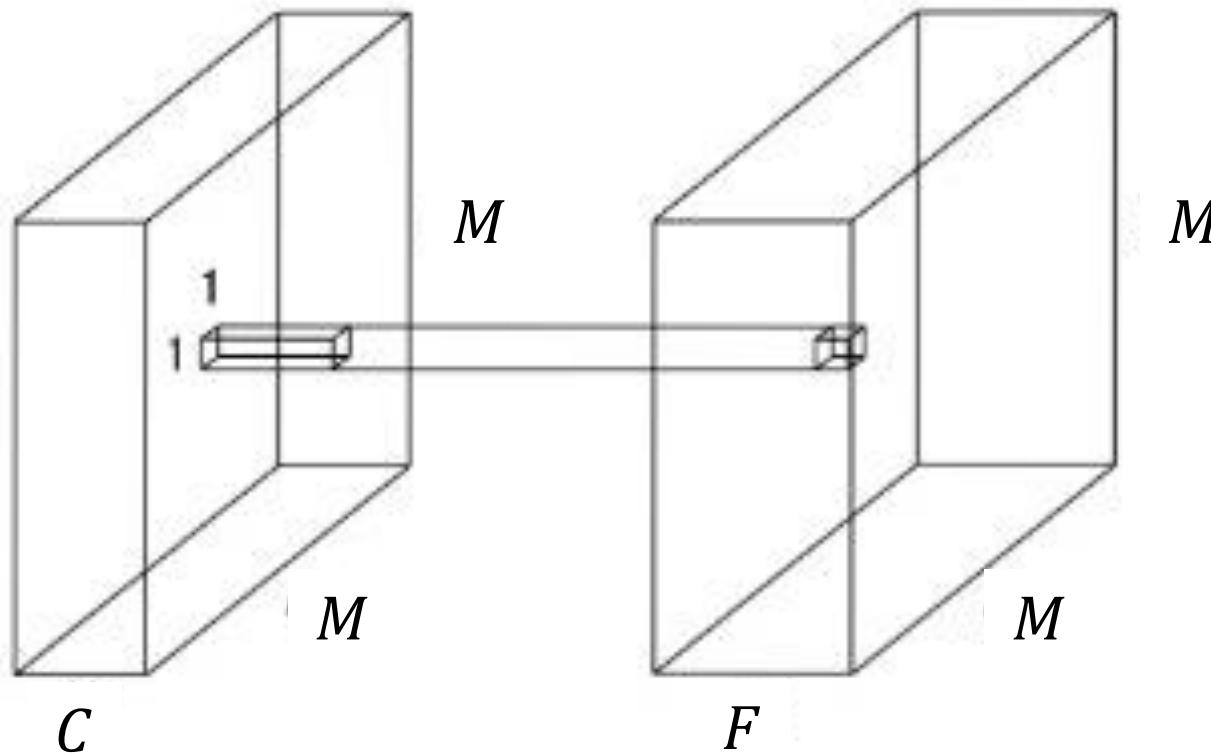


1×1 konvoluce

vzpomeňme: filtr vždy zahrnuje všechny kanály

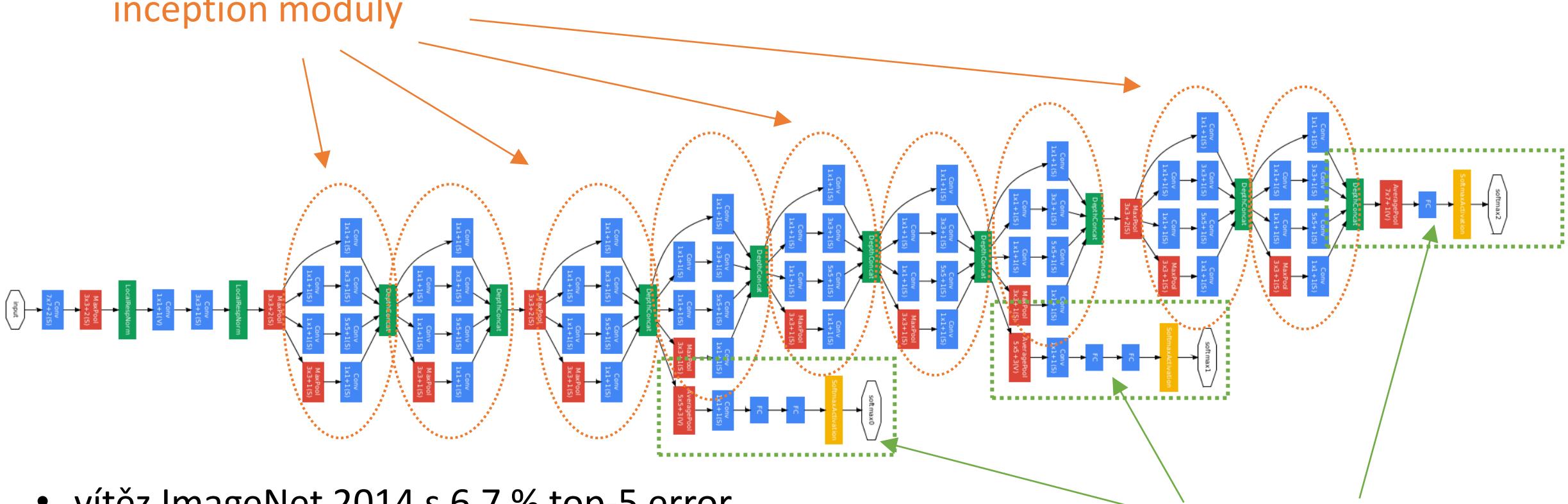
tzn., že i při 1×1 konvoluci (bez okolí) je výsledek stále lineární kombinací přes kanály

filtr je tedy $1 \times 1 \times C \rightarrow$ počet parametrů je $C \cdot F$



GoogLeNet (2014)

inception moduly



- vítěz ImageNet 2014 s 6.7 % top-5 error
- celkem 22 vrstev
- minimum fully-connected (lineárních) vrstev, téměr plně konvoluční síť
- 12x méně parametrů než AlexNet

loss na více místech sítě, ne
pouze na vrchu

ResNet (2015)

- He et al.: “Deep Residual Learning for Image Recognition”
- Cílem návrhu být co nejhlubší → 152 vrstev!
- Vítěz ImageNet 2015 ve všech kategoriích
- Vítěz MS COCO challenge
- 3.6 % top-5 error na ImageNet: lepší než člověk (cca 5 %)
- Problém: přidávání vrstev pomáhá jen do určité chvíle, pak už ne
- Overfitting?

Příliš mnoho vrstev

Overfitting? Ne: kromě testovací chyby s více vrstvami roste i trénovací chyba!

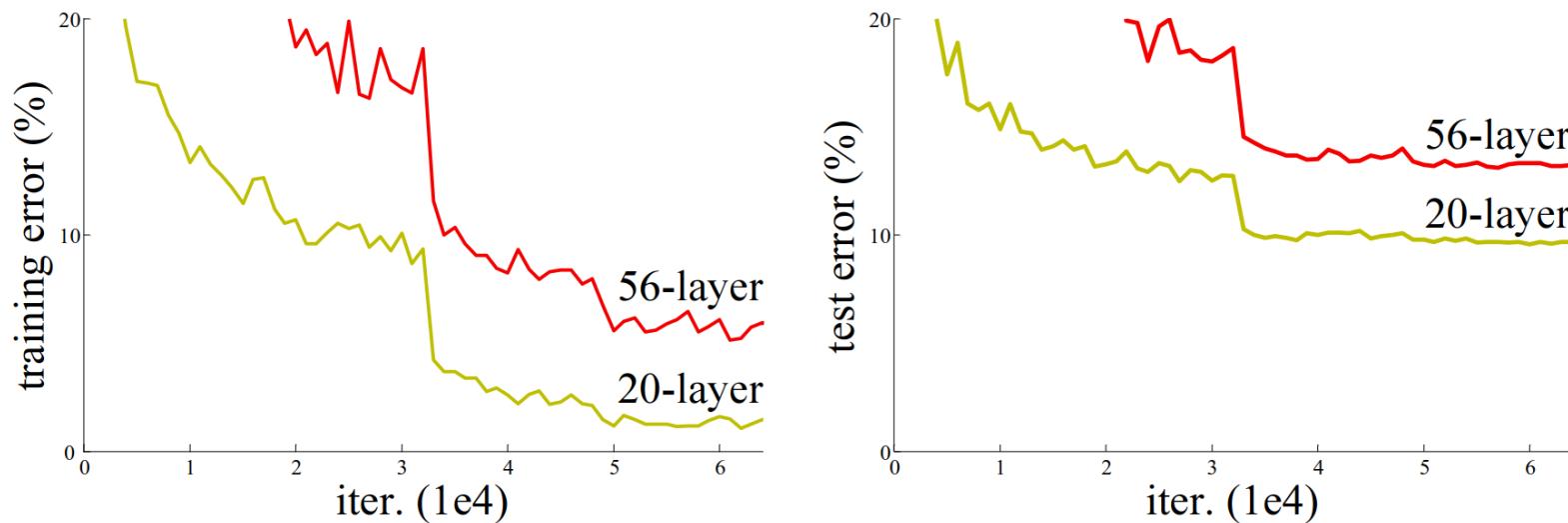


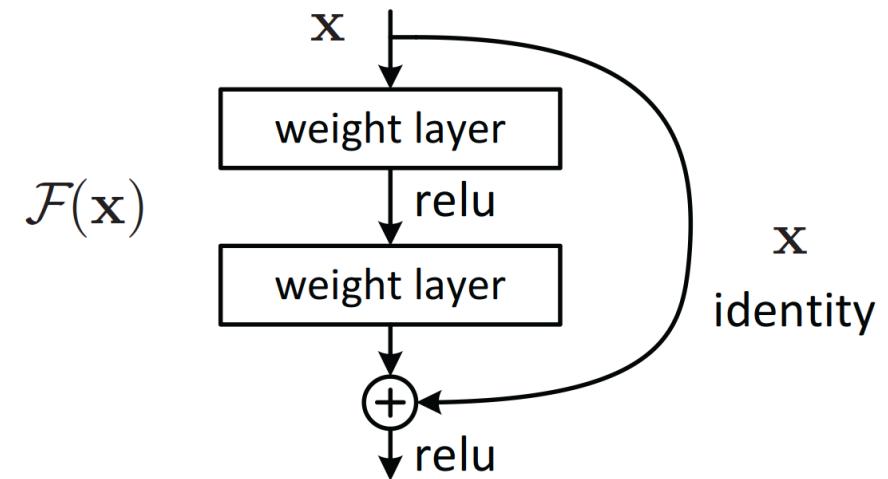
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Reziduální blok

- Podobně jako inception používá složitější bloky
- Výstup sestává ze součtu konvoluce a přímo mapovaného vstupu (identity)
- Sítí se tedy učí pouze rezidua

$$\mathcal{F}(x) = \mathcal{H}(x) - x$$

- “Naučit se nuly je jednodušší než identitu”



$$\mathcal{H}(x) = \mathcal{F}(x) + x$$

Figure 2. Residual learning: a building block.

Bottleneck residual blok

Podobně jako u Inception i ResNet
optimalizuje pomocí 1×1 bottleneck
vrstev uvnitř reziduálních bloků

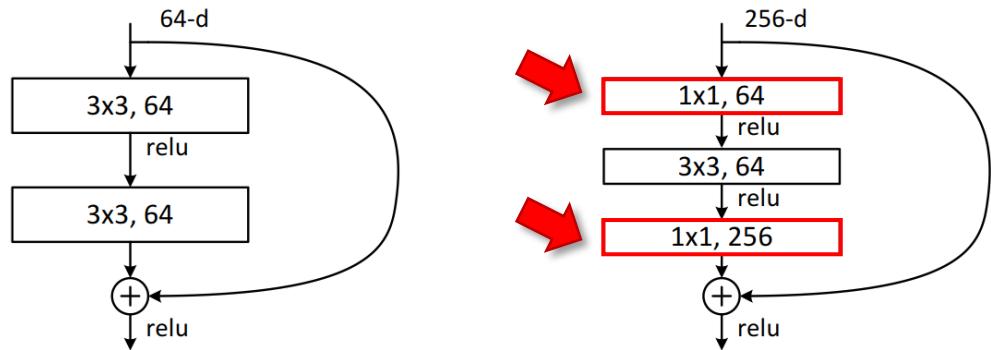
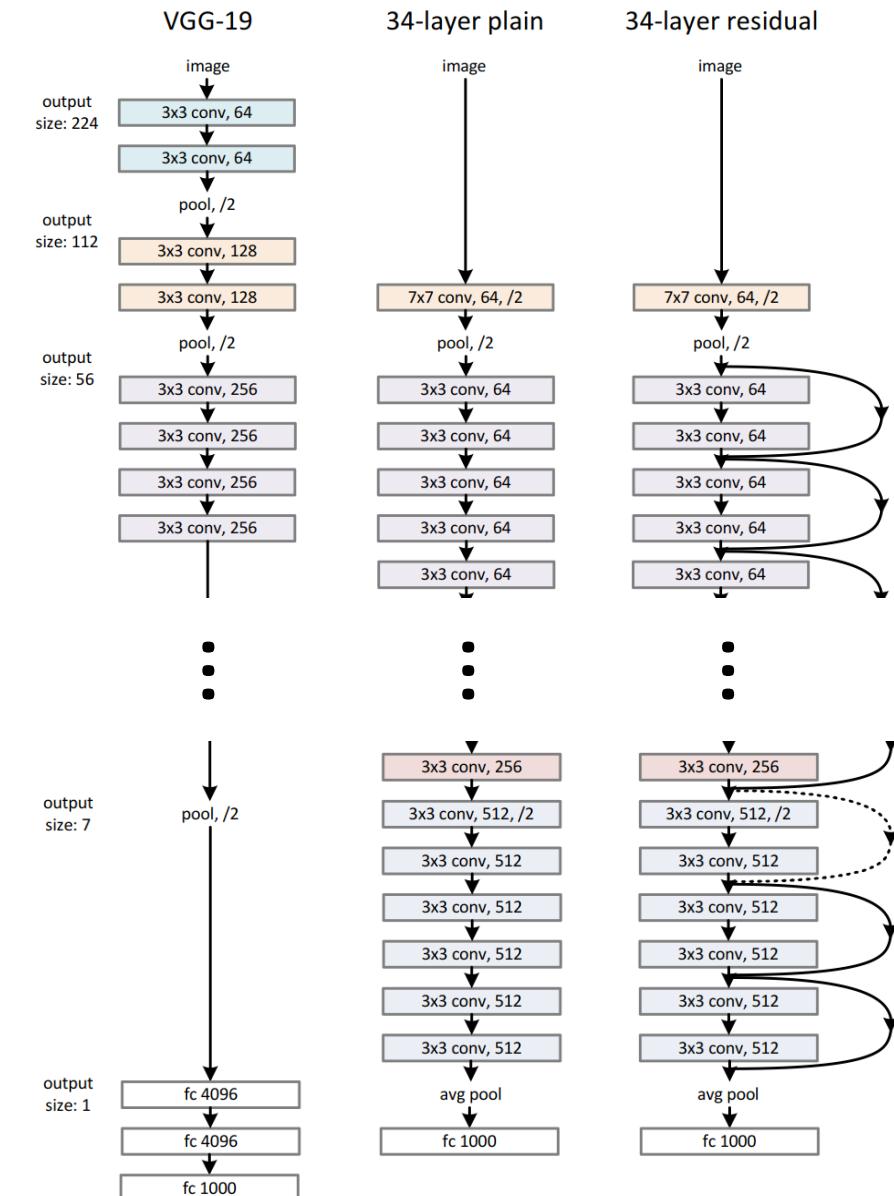


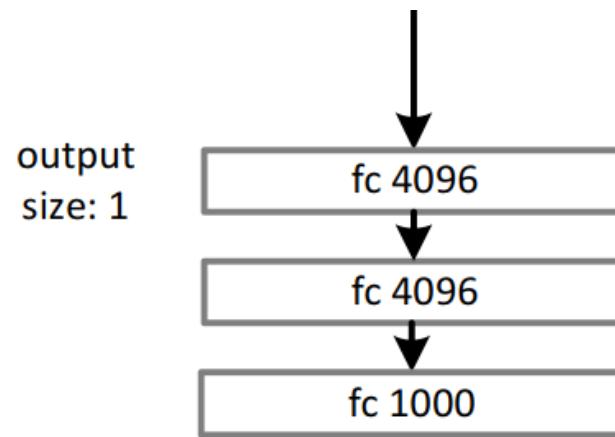
Figure 5. A deeper residual function \mathcal{F} for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a “bottleneck” building block for ResNet-50/101/152.



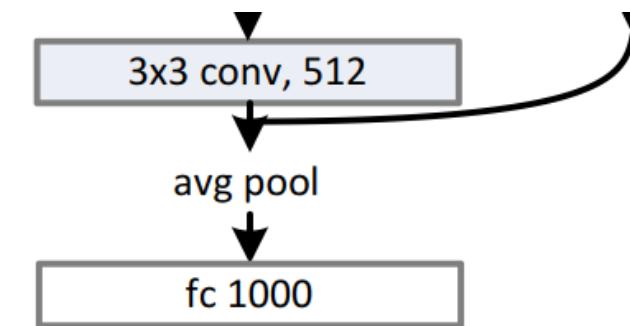
Téměr plně konvoluční

bez skrytých lineárních vrstev

VGG:



ResNet:



pouze lineární klasifikátor

Average pooling

1	0	2	3
4	6	6	8
3	1	1	0
1	2	2	4

max pooling

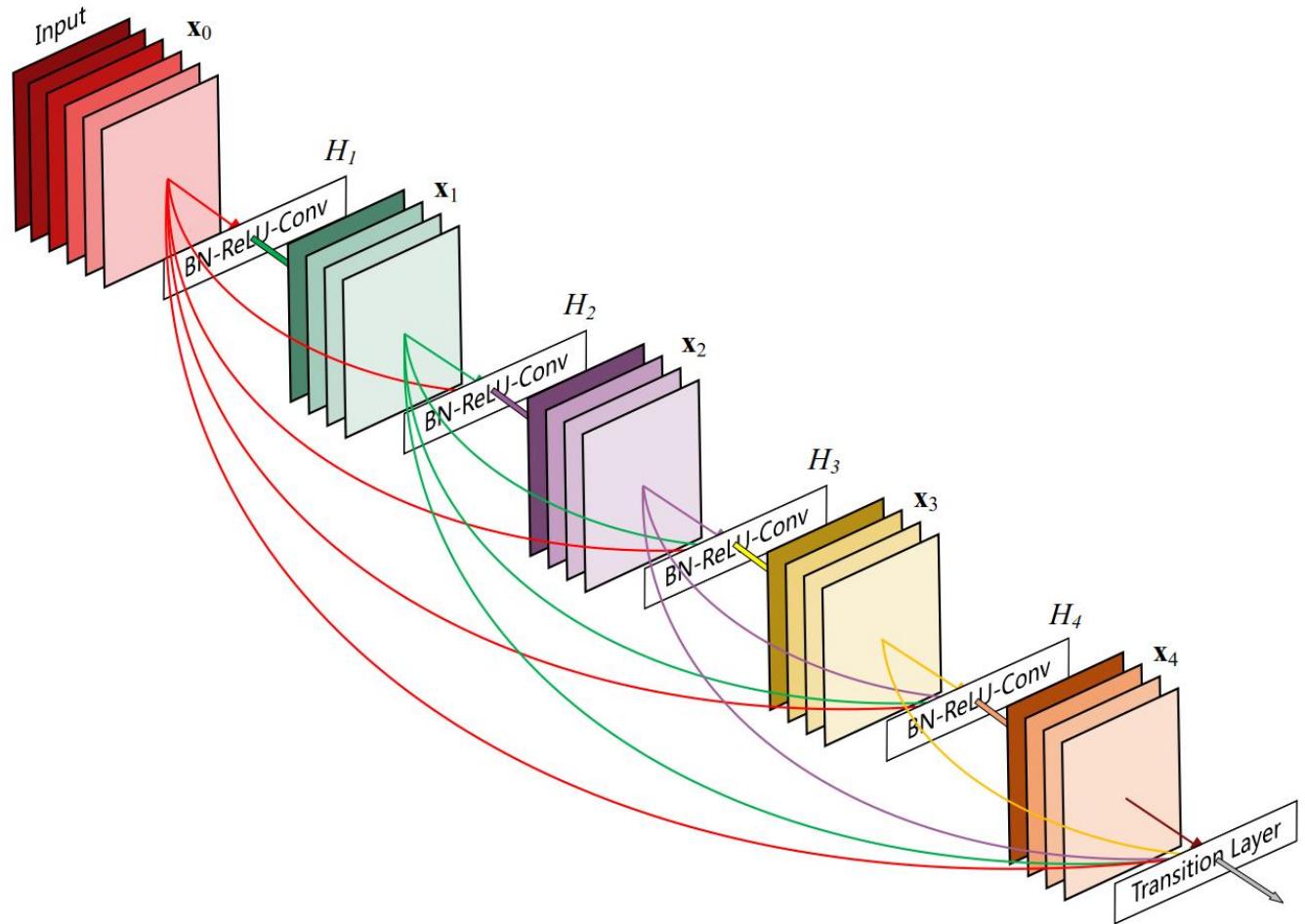
6	8
3	4

average pooling

3	5
2	2

DenseNet (2016)

- Huang et al.: “Densely Connected Convolutional Networks”
- Výstup vrstvy je připojen na vstup každé další vrstvy
- “ResNet do extrému”



SqueezeNet (2016)

- Iandola et al: “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size”
- Cílem co nejmenší a nejfektivnější síť

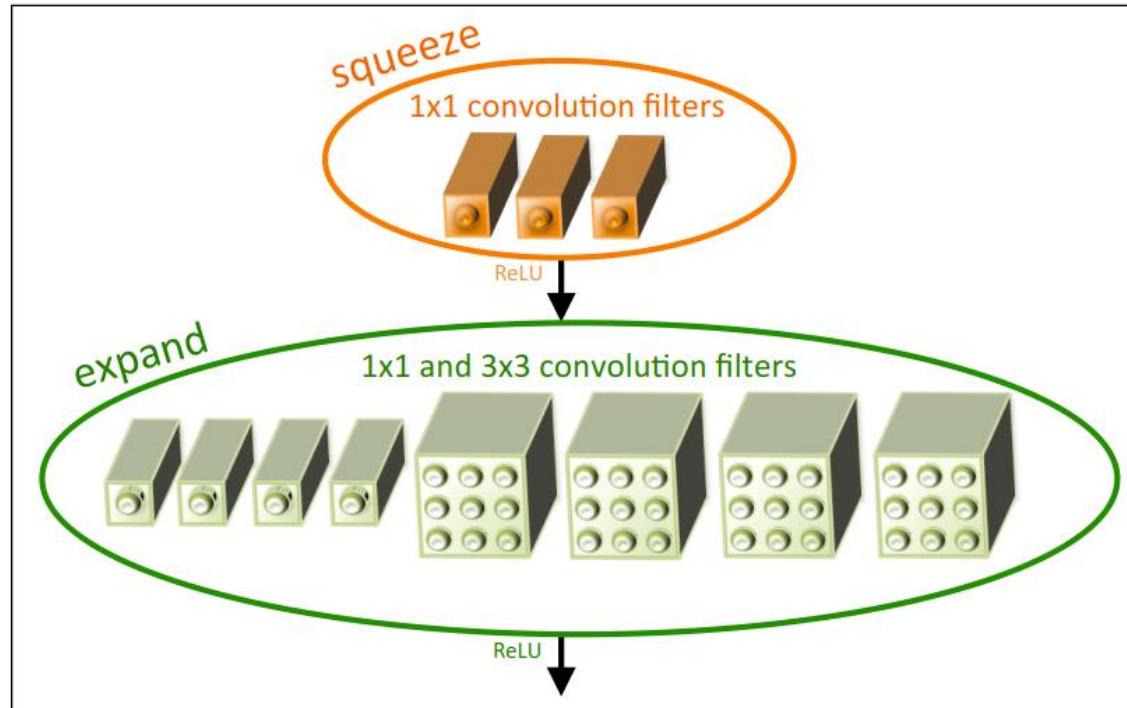
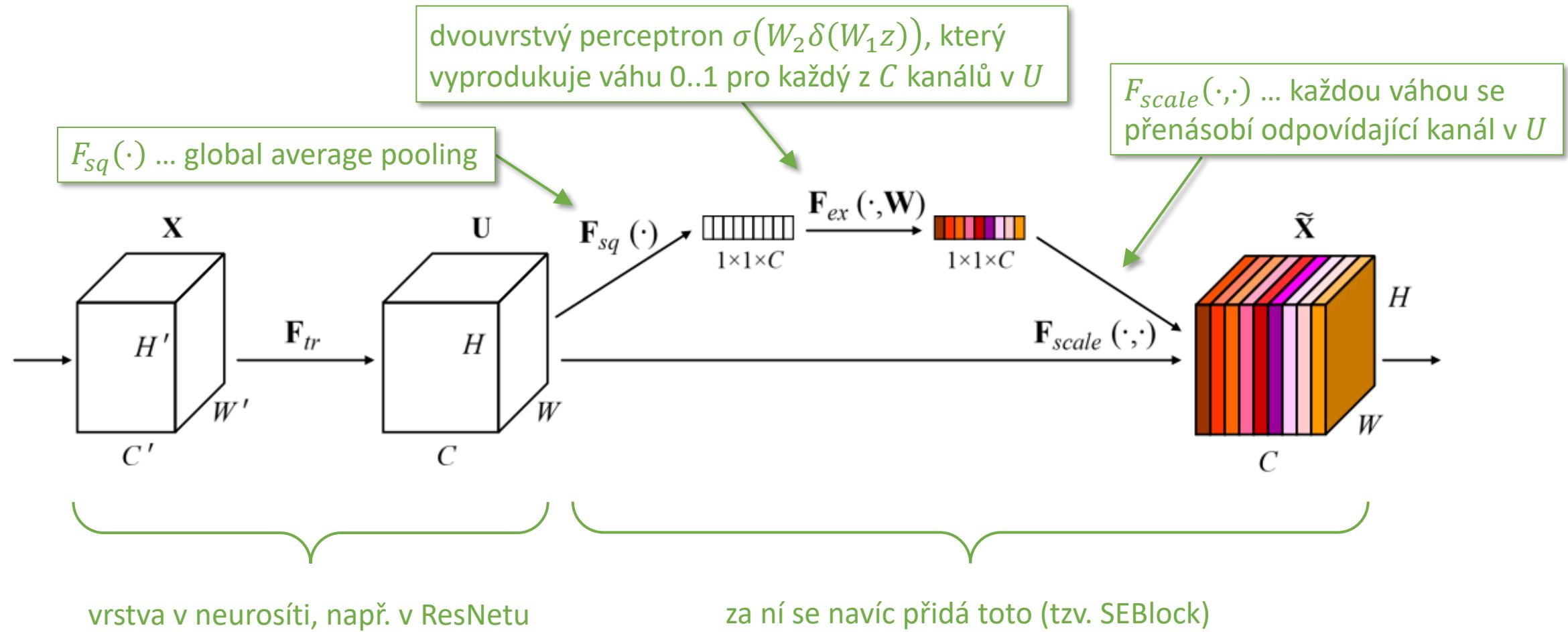


Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

SENet (2017)

- [Hu et al.: Squeeze-and-excitation Networks](#)
- Cílem nějak explicitně modelovat závislost kanálů



SENet (2017)

```
class SE_Block(nn.Module):
    "credits: https://github.com/moskomule/senet.pytorch"
    def __init__(self, c, r=16):
        super().__init__()
        self.squeeze = nn.AdaptiveAvgPool2d(1)
        self.excitation = nn.Sequential(
            nn.Linear(c, c // r, bias=False),
            nn.ReLU(inplace=True),
            nn.Linear(c // r, c, bias=False),
            nn.Sigmoid()
        )

    def forward(self, x):
        bs, c, _, _ = x.shape
        y = self.squeeze(x).view(bs, c)
        y = self.excitation(y).view(bs, c, 1, 1)
        return x * y.expand_as(x)
```

plugin do existujících sítí

TABLE 4
Classification error (%) on CIFAR-10.

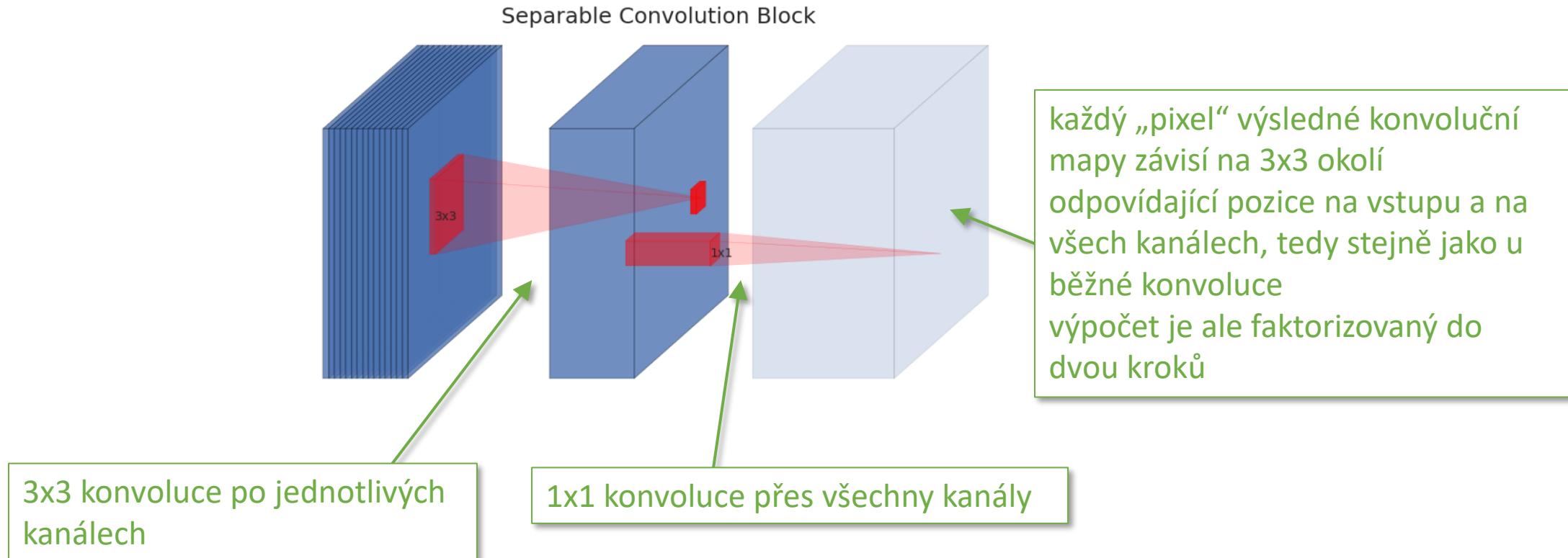
	original	SENet
ResNet-110 [14]	6.37	5.21
ResNet-164 [14]	5.46	4.39
WRN-16-8 [67]	4.27	3.88
Shake-Shake 26 2x96d [68] + Cutout [69]	2.56	2.12

TABLE 5
Classification error (%) on CIFAR-100.

	original	SENet
ResNet-110 [14]	26.88	23.85
ResNet-164 [14]	24.33	21.31
WRN-16-8 [67]	20.43	19.14
Shake-Even 29 2x4x64d [68] + Cutout [69]	15.85	15.41

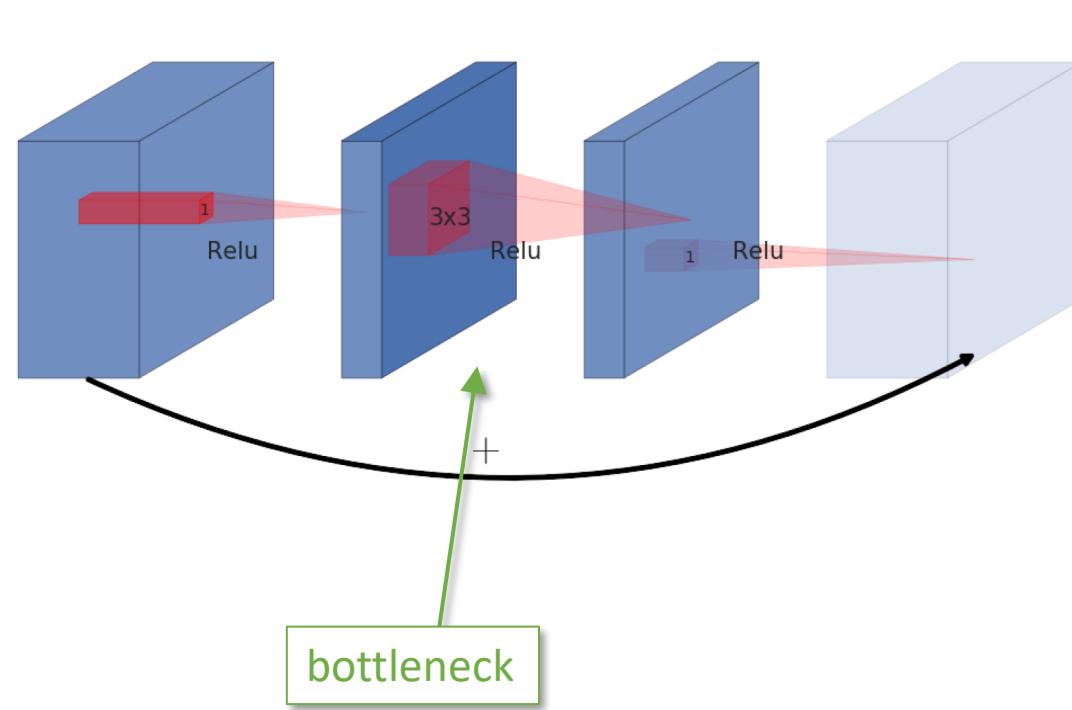
MobileNetV2 (2018)

- Sandler et al.: MobileNetV2: Inverted Residuals and Linear Bottlenecks
- Depthwise separable konvoluce



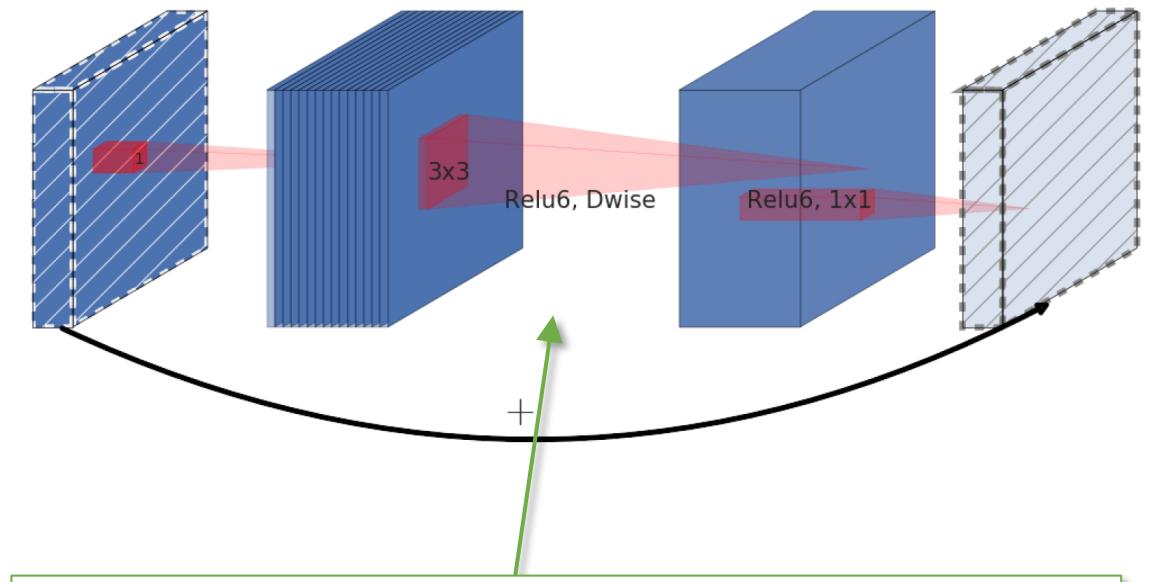
MobileNetV2 (2018)

standardní ResNet



upravený ResNet s depthwise separable konvolucemi

(b) Inverted residual block



- 1. konvolve je nahrazena depthwise separable verzí
- 2. místo bottlenecku expansion (= více kanálů)
- 3. čárkované konv. mapy neprocházejí nelinearitou

MobileNetV2 (2018)

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	3.4M	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	72.0	3.4M	300M	75ms
MobileNetV2 (1.4)	74.7	6.9M	585M	143ms

Table 4: Performance on ImageNet, comparison for different networks. As is common practice for ops, we count the total number of Multiply-Adds. In the last column we report running time in milliseconds (ms) for a single large core of the Google Pixel 1 phone (using TF-Lite). We do not report ShuffleNet numbers as efficient group convolutions and shuffling are not yet supported.

MNASNet (2018)

- [Tan et al.: MnasNet: Platform-Aware Neural Architecture Search for Mobile \(2018\)](#)
- Automatické hledání optimální architektury

Formulace úlohy jako optimalizace:

$$\underset{m}{\text{maximize}} \quad ACC(m) \times \left[\frac{LAT(m)}{T} \right]^w$$

$ACC(m)$... správnost modelu m

T ... cílová latence = čas potřebný pro dopředný průchod

$LAT(m)$... latence změřená přímo na mobilních telefonech

$w = -0.07$

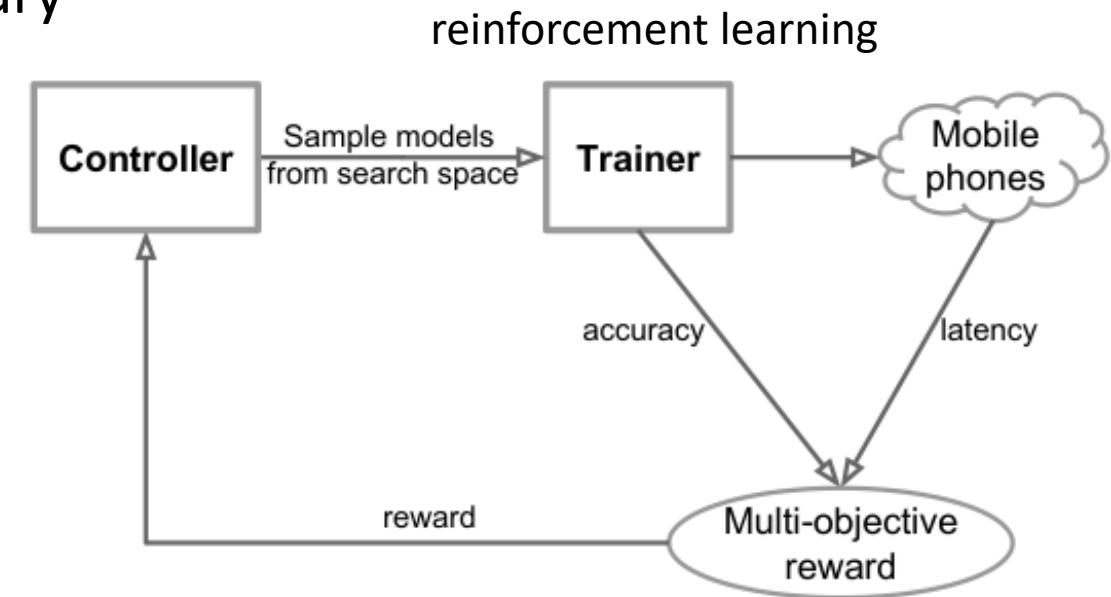


Figure 1: An Overview of Platform-Aware Neural Architecture Search for Mobile.

Hledání není exhaustivní – základním stavebním kamenem je Inverted Residual Block (MobileNetV2), u něhož se pouze upravují hloubka a velikost filtru

EfficientNet (2019)

- [Tan, Le: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks](#)

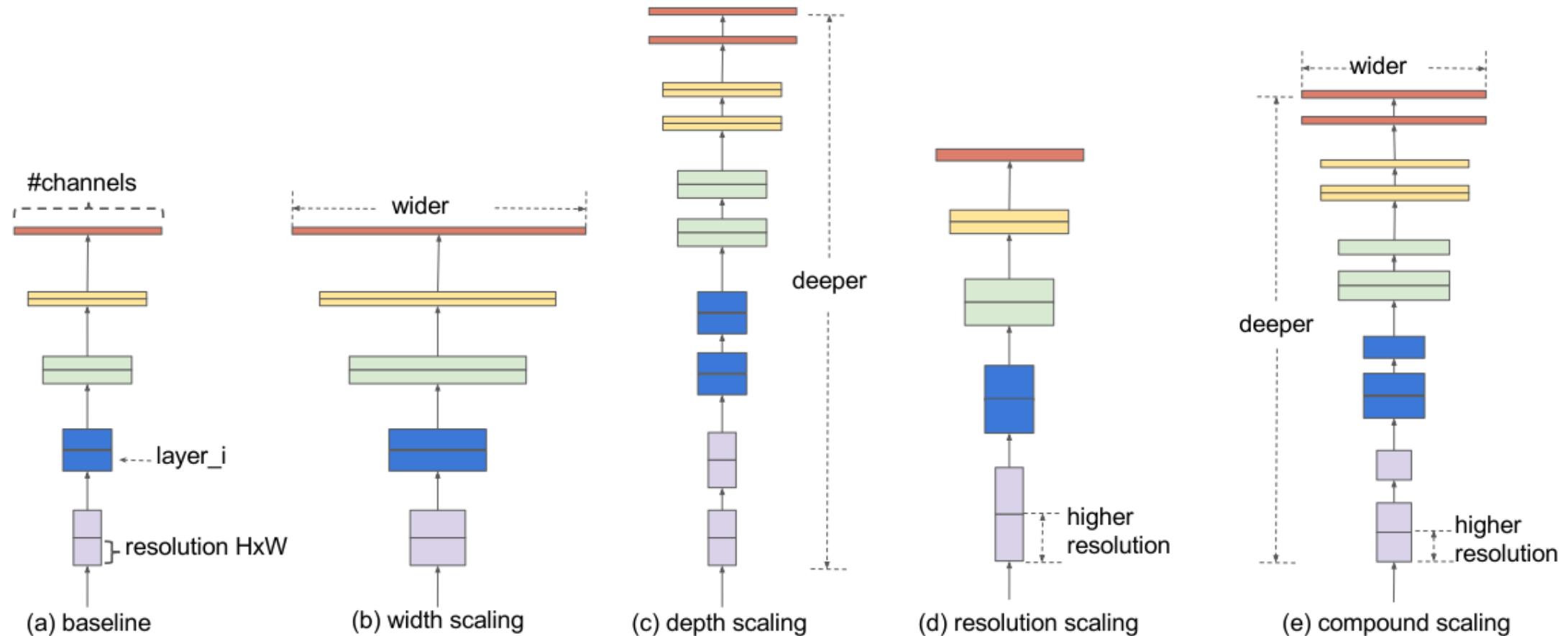


Figure 2. Model Scaling. (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

EfficientNet (2019)

- [Tan, Le: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks](#)
- Podobná revoluce jako v roce 2015 přinesl ResNet, stav poznání ve všech oblastech počítačového vidění
- **EfficientNet = neural architecture search + compound scaling**
- Neural architecture search (NAS) - formulováno jako reinforcement learning s expected reward (proximal policy)

$$ACC(m) \cdot [FLOPS(m)/T]^w$$

kde namísto latence je teoretická výpočetní náročnost ve floating point operacích

- Cílovým FLOPS je cca 400 MFLOPS

EfficientNet (2019)

- Automatickým hledáním NAS vznikne EfficientNet-B0, která má nějaké cílové FLOPs
- Nyní chceme větší síť, ale bez NAS, protože to by nebylo upočítatelné
- Compound scaling
 - Větší síť bude „násobkem“ (škálováním) základní sítě
 - Síť vyjádříme jako spojení s bloků, z nichž každý L_i -krát opakuje operaci \mathcal{F}_i

$$\mathcal{N} = \bigodot_{i=1 \dots s} \mathcal{F}_i^{L_i}(X_{(H_i, W_i, C_i)}) \quad (1)$$

- Škálovat budeme uvnitř bloků, tj. síť bude mít stejný počet bloků, ty ale budou větší
- Musíme najít optimální poměr toho, jak moc má být větší hloubka (# vrstev), šířka (# filtrů ve vrstvách) a rozlišení (velikost konv. map)
- Zároveň chceme, aby větší verze měla cca 2x více FLOPs

EfficientNet (2019)

$$\begin{aligned} \text{depth: } d &= \alpha^\phi \\ \text{width: } w &= \beta^\phi \\ \text{resolution: } r &= \gamma^\phi \\ \text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\ \alpha \geq 1, \beta \geq 1, \gamma \geq 1 & \end{aligned}$$

velikost sítě lze ovládat jediným parametrem Φ tak, že zvětnutí jeho hodnoty o +1 zdvojnásobí FLOPs (viz podmínka)

- Základní síť EfficientNet-B0 má $\Phi = 0$
- Větší síť získáme
 - počet vrstev uvnitř každého bloku vynásobíme α -krát (a zaokrouhlíme)
 - počet filtrů konv. map vynásobíme β -krát
 - rozlišení vstupu zvětšíme γ -krát
- Neznáme hodnoty $\alpha, \beta, \gamma \rightarrow$ provedeme grid search přes různé kombinace, které splňují podmínu $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ a vybereme takovou, která maximalizuje skóre po natrénovalání přeškálované sítě (v článku vyšlo $\alpha = 1.2, \beta = 1.1, \gamma = 1.15$)
- Vznikne EfficientNet-B1, protože má $\Phi = 1$
- Další sítě až do EfficientNet-B6 získáme dosazením $\Phi = 2 \dots 6$ do škálovacího schématu

EfficientNet (2019)

Table 1. EfficientNet-B0 baseline network – Each row describes a stage i with \hat{L}_i layers, with input resolution $\langle \hat{H}_i, \hat{W}_i \rangle$ and output channels \hat{C}_i . Notations are adopted from equation 2.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels \hat{C}_i	#Layers \hat{L}_i
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

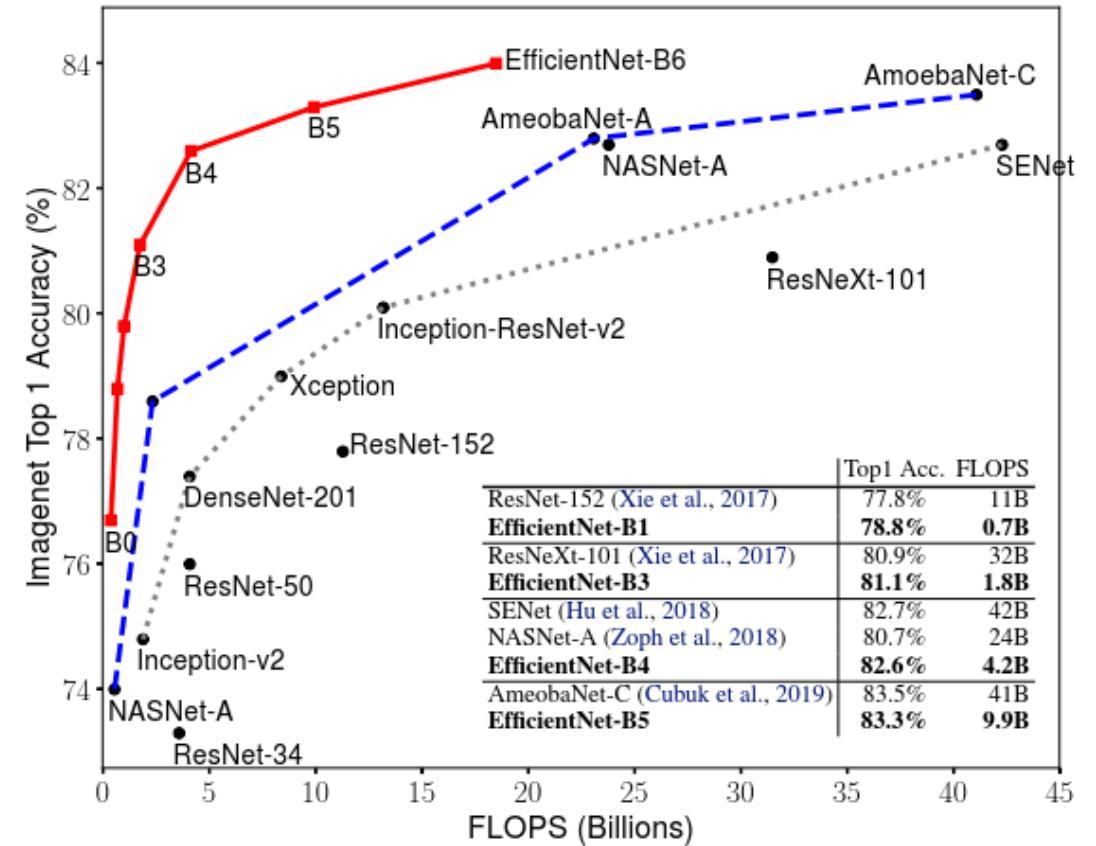
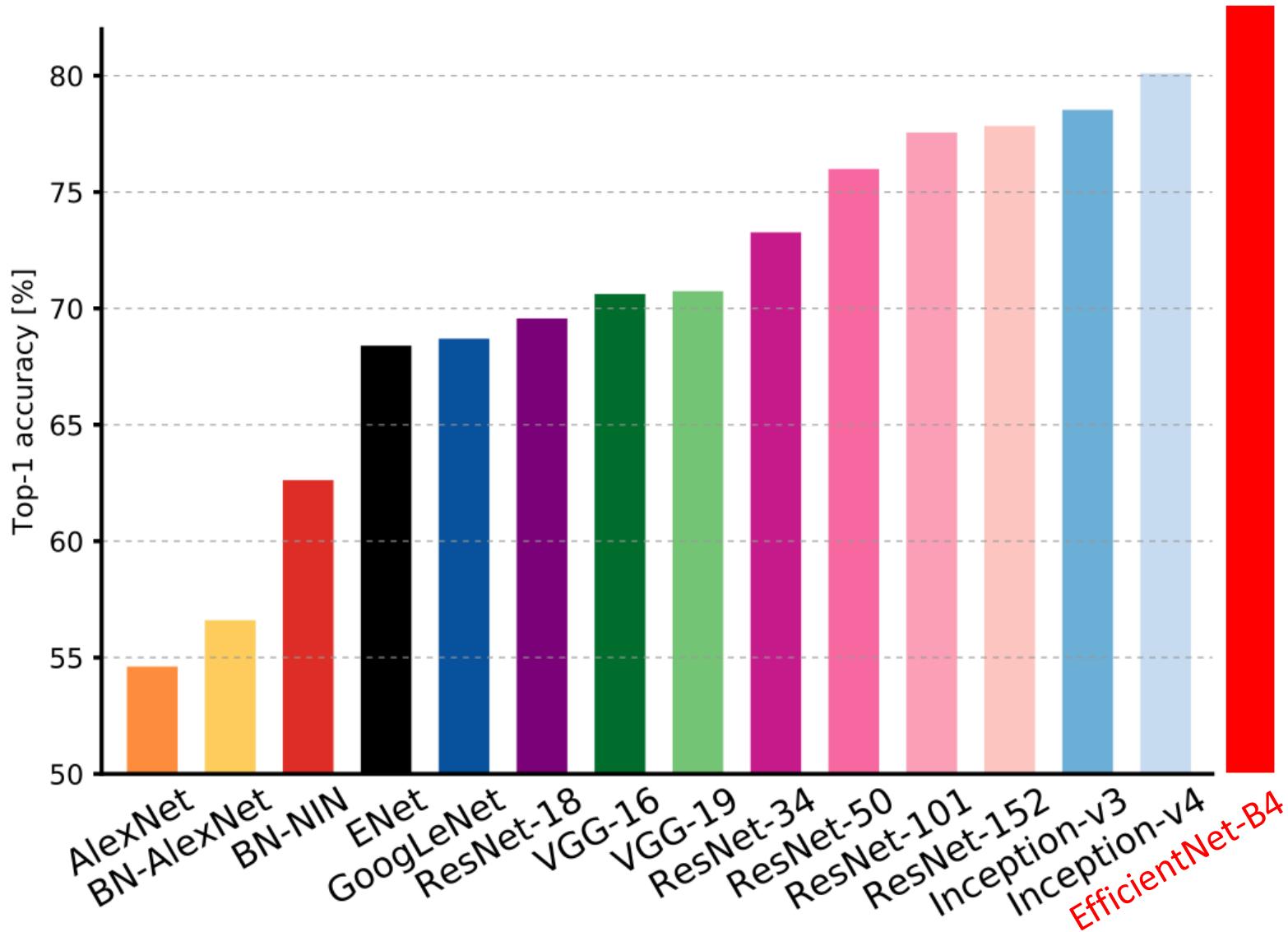


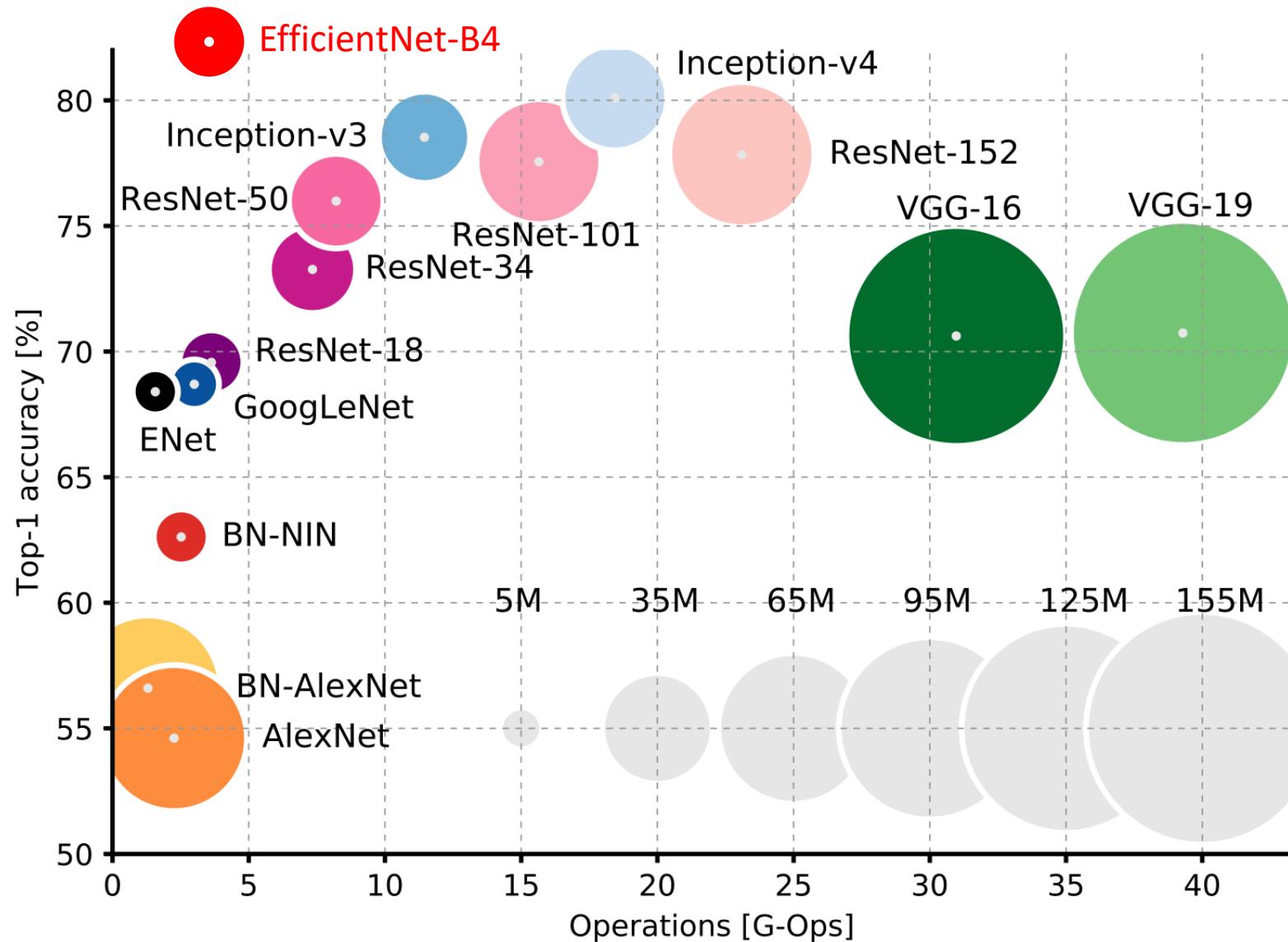
Figure 5. FLOPS vs. ImageNet Accuracy – Similar to Figure 1 except it compares FLOPS rather than model size.

Srovnání nejpoužívanějších CNN architektur



obrázek: [Canziani et al.: "An Analysis of Deep Neural Network Models for Practical Applications"](#)

Srovnání nejpoužívanějších CNN architektur



velikost znázorňuje
celkový počet
parametrů

obrázek: [Canziani et al.: "An Analysis of Deep Neural Network Models for Practical Applications"](#)