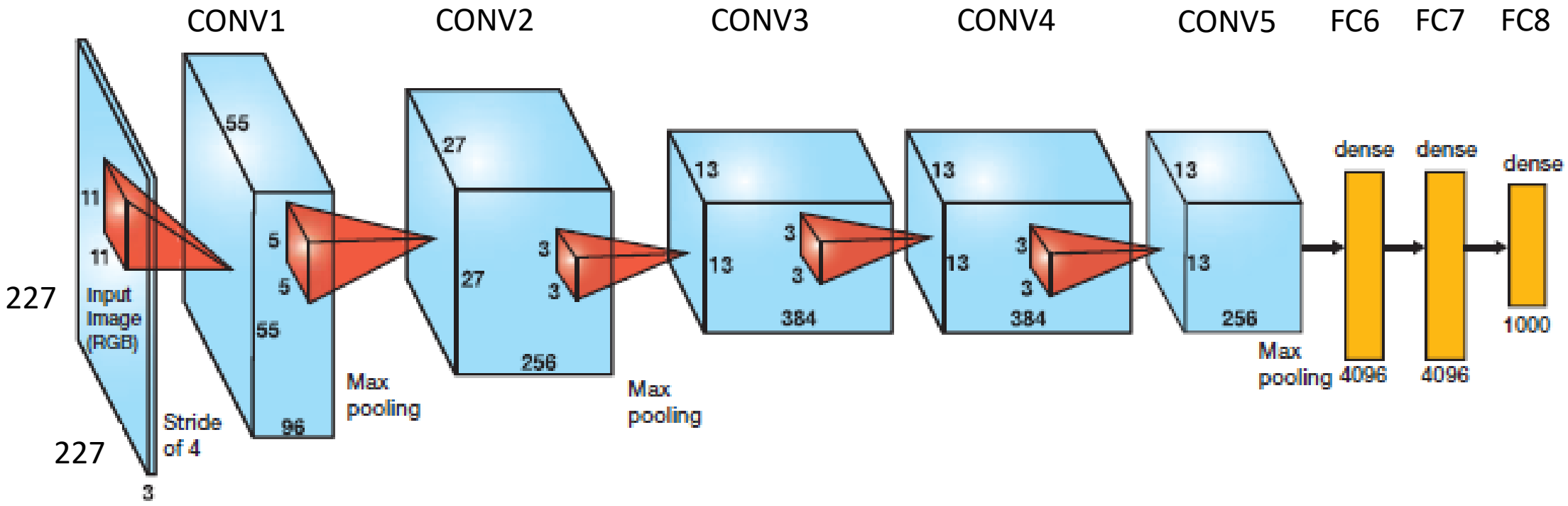
Aplikace neuronových sítí

Trénování sítí v praxi

Alexnet (2012)



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

obrázek: http://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/

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

Úprava dat

Předzpracování dat

- U konvolučních sítí pro obrazová data obvykle jen velmi omezené
- Cílem je end-to-end učení modelu
- Např. odečtení průměrného obrázku

  `out = rgb - mean_image`

  kde `mean_image` je 32x32x3

- Odečtení průměrného pixelu

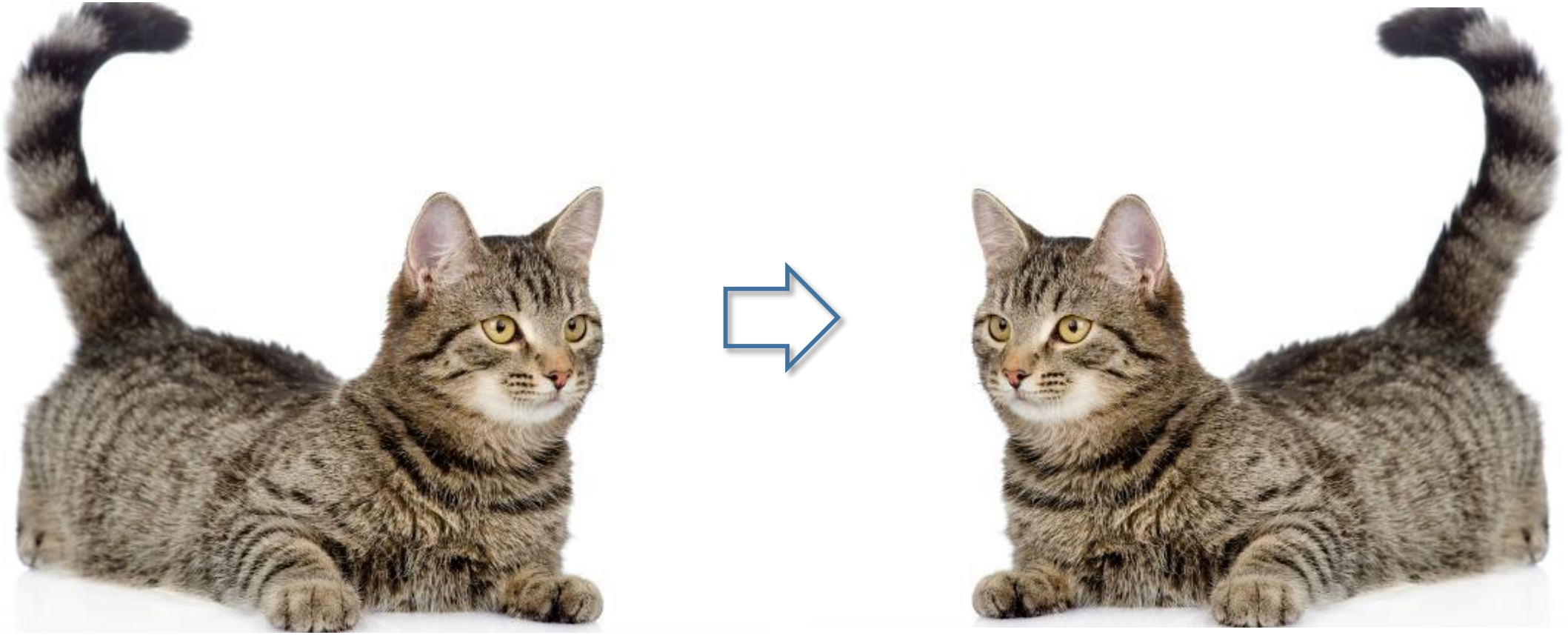
  `out = rgb - mean_pixel`

  kde `mean_pixel` je trojice [r, g, b]

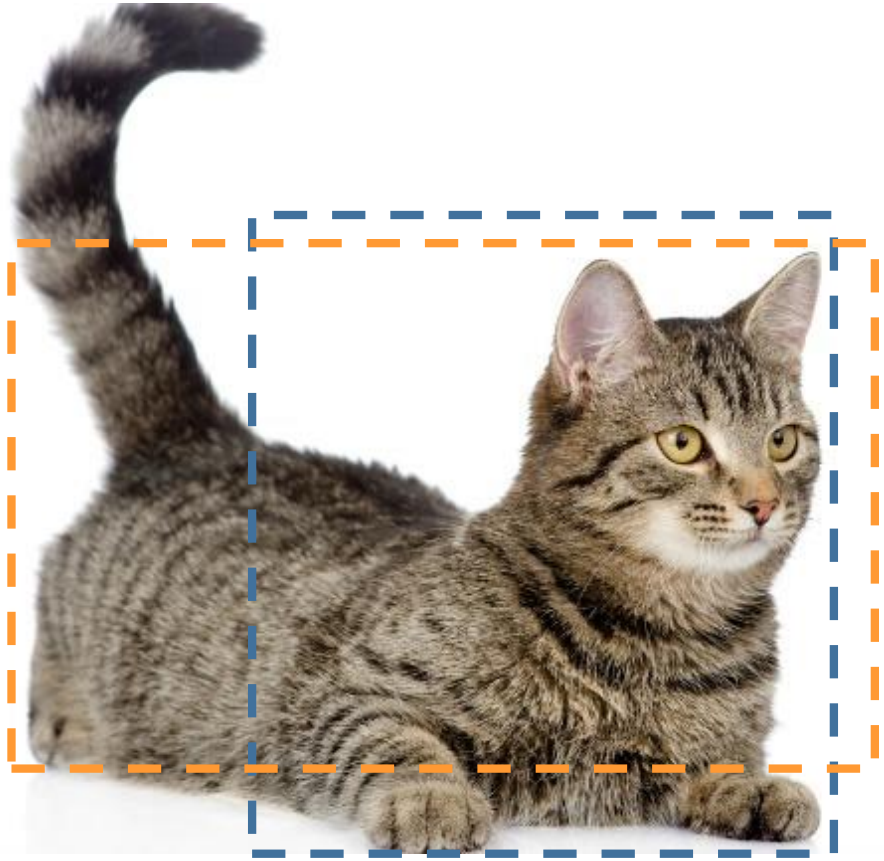
Umělé rozšiřování dat

- Data augmentation
- U neurosítí téměř vždy platí: více dat = lepší výsledky
- Pokud reálná data nejsou, lze je “nafouknout” uměle, např. náhodnými transformacemi obrázků

Zrcadlení



Ořez



Další transformace

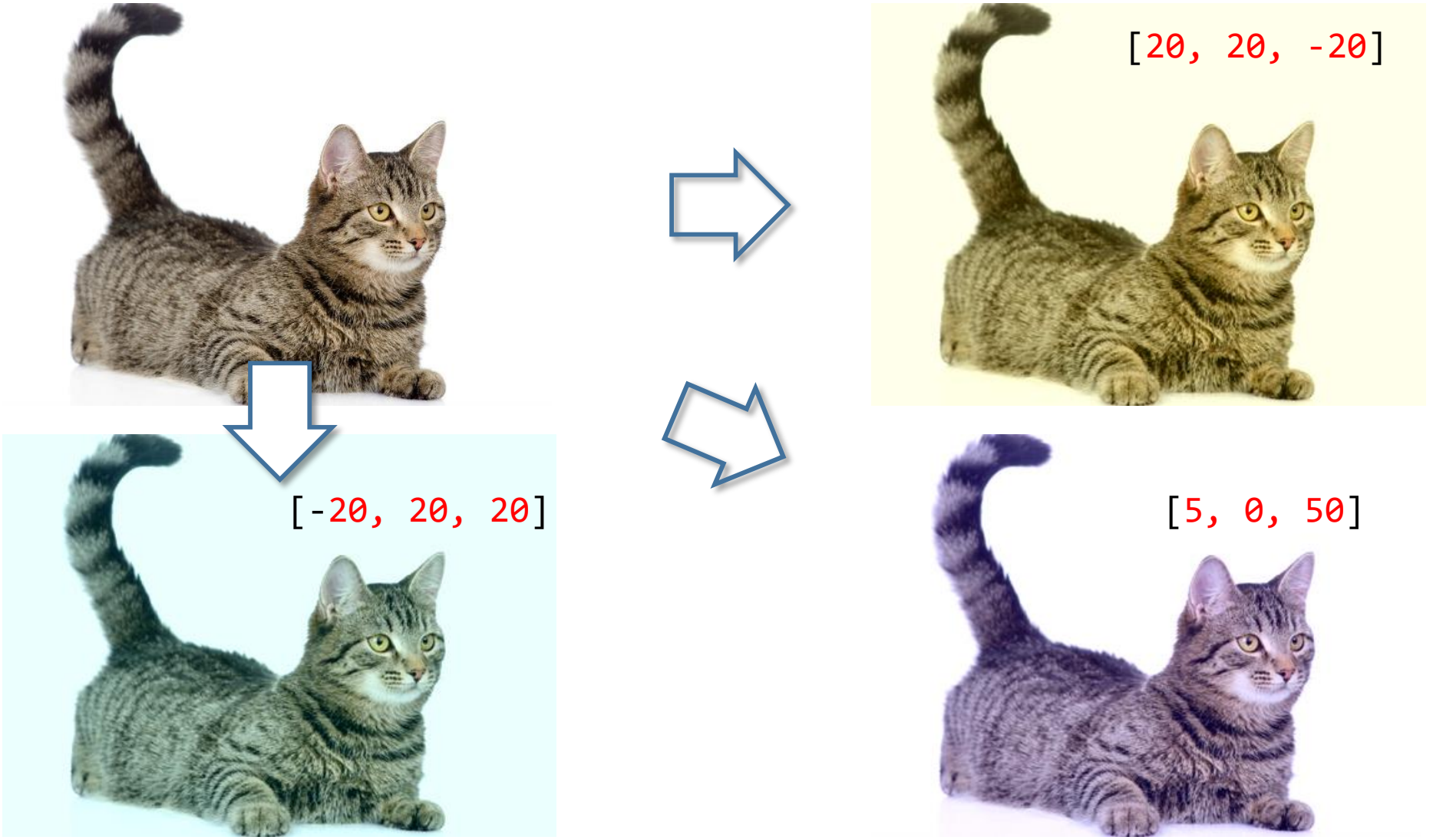
- Otočení
- Zkosení
- Lokální deformace a další
- Ne vždy ale pomáhají

Train augmentation

Name	Accuracy	LogLoss	Comments
Default	**0.471**	**2.36**	Random flip, random crop 128x128 from 144xN, N > 144
Drop 0.1	0.306	3.56	+ Input dropout 10%. not finished, 186K iters result
Multiscale	0.462	2.40	Random flip, random crop 128x128 from (144xN, - 50%, 188xN - 20%, 256xN - 20%, 130xN - 10%)
5 deg rot	0.448	2.47	Random rotation to [0..5] degrees.

https://github.com/ducha-aiki/caffenet-benchmark/blob/master/Augmentation.md

Posun barev




```
out = np.clip(rgb + np.array([r, g, b]), 0, 255).astype(np.uint8)
```

Úprava dat v Kerasu

ImageDataGenerator

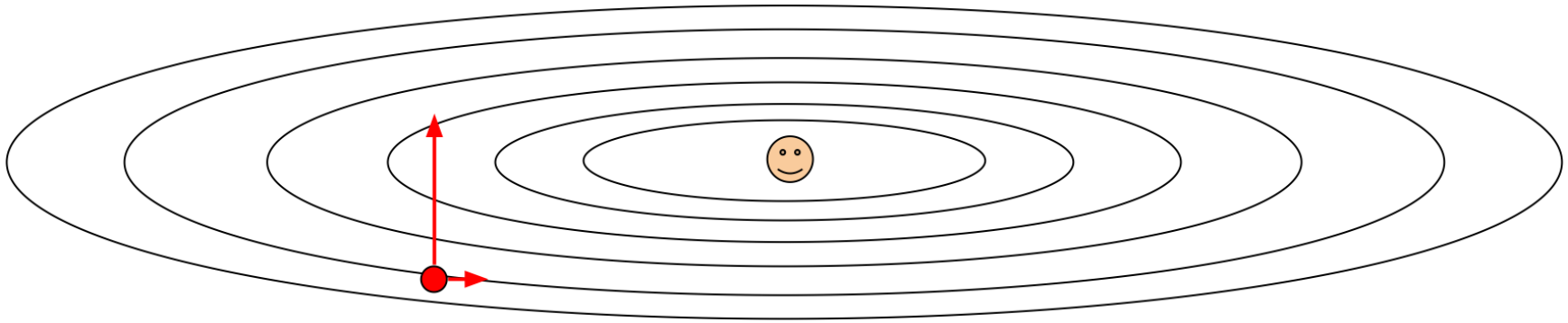
```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-6,
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    shear_range=0.,
    zoom_range=0.,
    channel_shift_range=0.,
    fill_mode='nearest',
    cval=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=K.image_data_format())
```

Generate batches of tensor image data with real-time data augmentation. The data will be looped over (in batches) indefinitely.

Optimalizační metody

SGD update

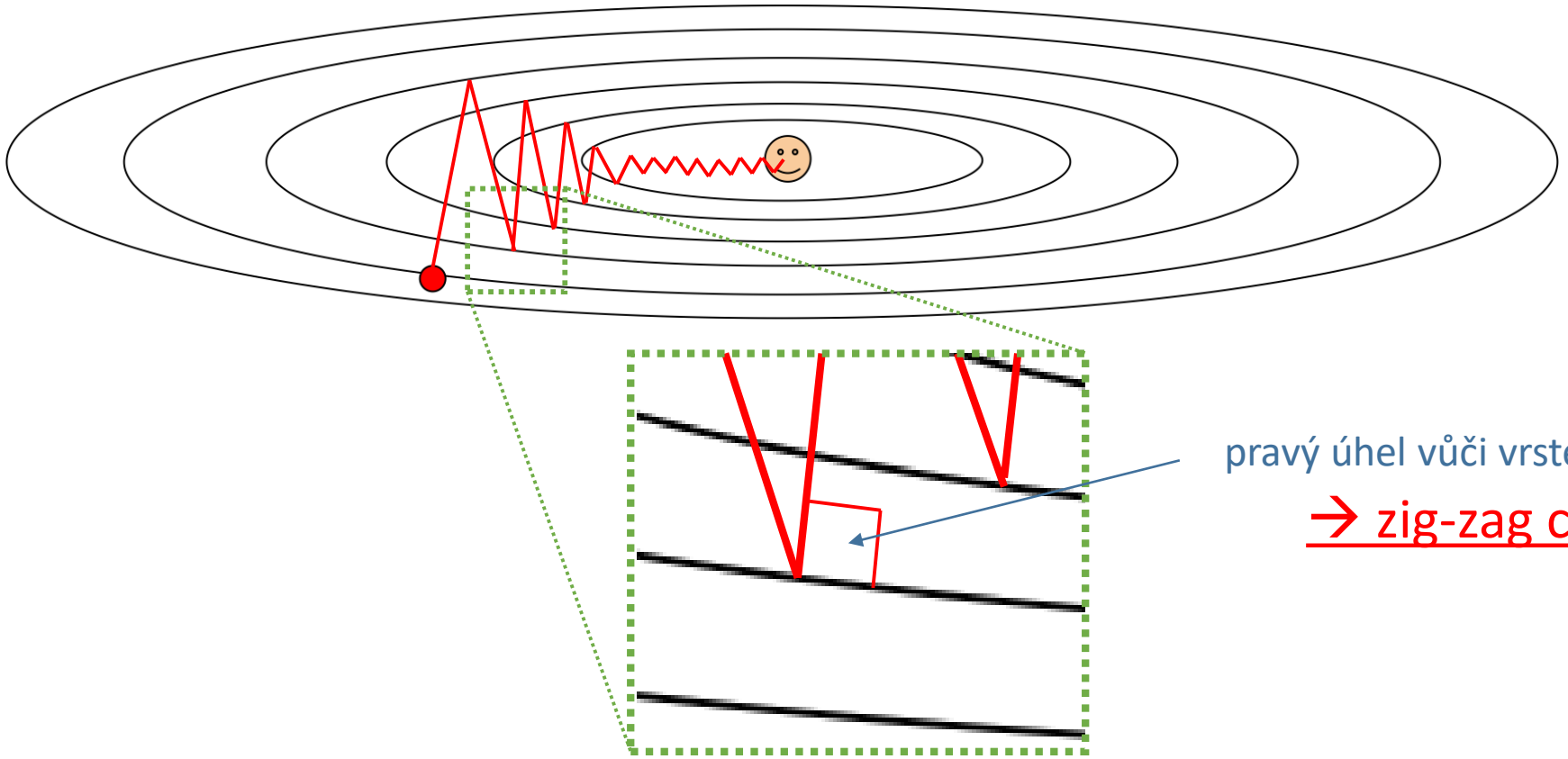
funkce, která je v jednom směru mnohem citlivější na změnu



obrázek: http://cs231n.stanford.edu/

SGD update

funkce, která je v jednom směru mnohem citlivější na změnu



pravý úhel vůči vrstevnici

→ zig-zag cesta

obrázek: http://cs231n.stanford.edu/

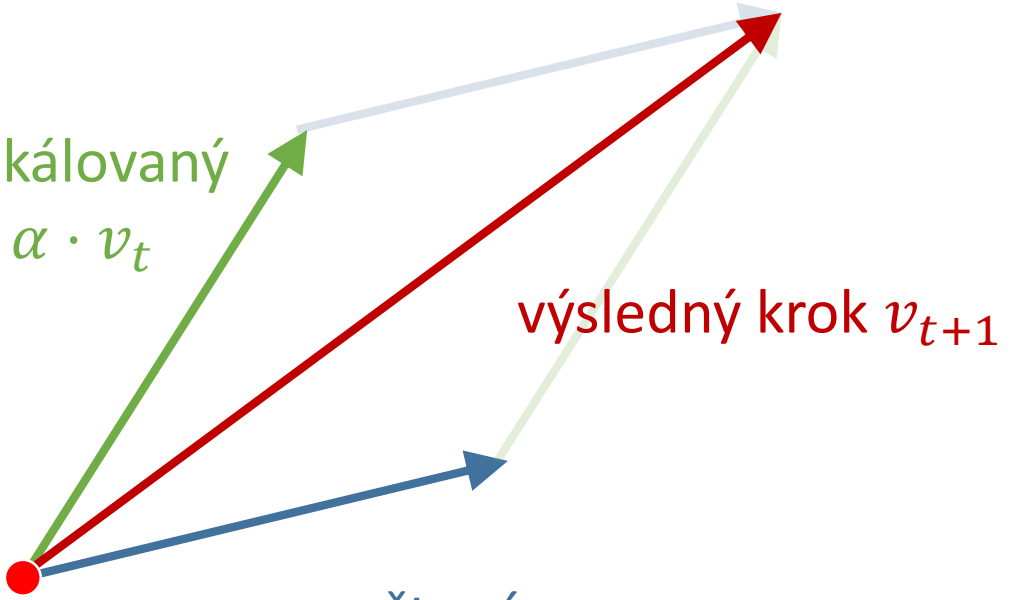
Momentum SGD

- Pamatuje si předchozí update
- Průměruje s novým
- → momentum = hybnost
- Obvykle konverguje rychleji

hyperparametr, např. $\alpha = 0.95$

learning rate (krok)

$$v_{t+1} \leftarrow \alpha \cdot v_t + \gamma \cdot dx_t$$
$$x_{t+1} \leftarrow x_t + v_{t+1}$$



standardní SGD:

$$x_{t+1} \leftarrow x_t + \gamma \cdot dx_t$$

RMSprop

- Root mean square
- Vychází z adaptivních technik jako např. AdaGrad
- Upravuje krok pro jednotlivé parametry
- Pokud je gradient v některých směrech neustále vyšší než jiné → normalizace
- Tzn. zmenšuje “protáhlé” dimenze = zvětšuje “splácnuté” → narovnává

decay rate … hyperparametr, typ. $\beta = 0.99$

průměrná norma gradientů (prvkově pro každý parameter → stejný rozměr jako gradient)

$$u_{t+1} \leftarrow \beta \cdot u_t + (1 - \beta) \cdot (dx_t)^2$$

prvkově na druhou

$$x_{t+1} \leftarrow \gamma \cdot \frac{dx_t}{\sqrt{u_{t+1} + \epsilon}}$$

Adam

- Adaptive momentum
- Kombinace Momentum SGD + RMSprop

momentum: $v_{t+1} = \alpha \cdot m_t + (1 - \alpha) \cdot dx_t$

rmsprop: $u_{t+1} = \beta \cdot u_t + (1 - \beta) \cdot (dx_t)^2$

Adam update: $x_{t+1} = \gamma \cdot \frac{v_{t+1}}{\sqrt{u_{t+1} + \epsilon}}$

- Obvykle funguje dobře i s výchozím nastavením hyperparametrů
- Dobrá výchozí volba

Batch normalizace, SELU

aneb další triky v rukávu

Batch normalizace (BN)

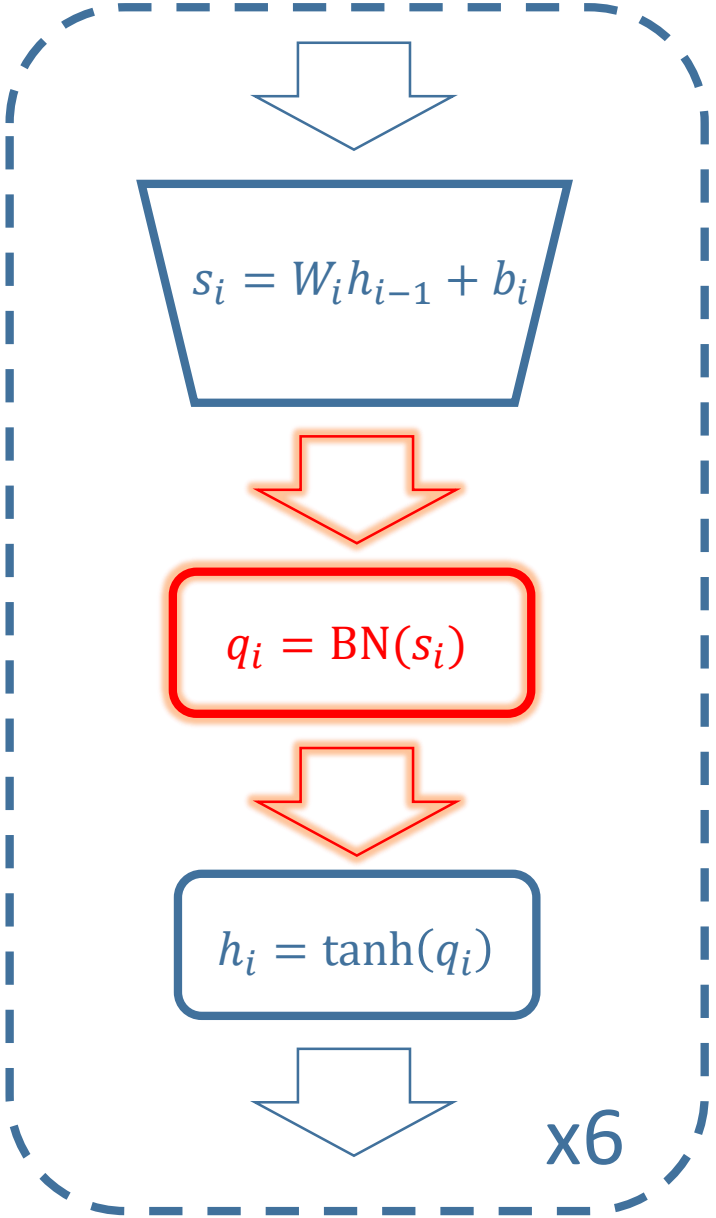
- Chceme podobné rozložení hodnot v různých vrstvách tak, aby žádná vrstva “nezabíjela” gradient
- Obtížné zajistit inicializací a aktivacemi / nelinearitami
- Co prostě výstup vrstvy normalizovat?
- Např. na nulový průměr a std. odchylku 1:

$$\widehat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}}$$

kde
$\mathrm{E}[x]$ je střední hodnota
$\mathrm{Var}[x]$ je rozptyl

- operace je diferencovatelná! → lze počítat gradient

https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html



Batch normalizace

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

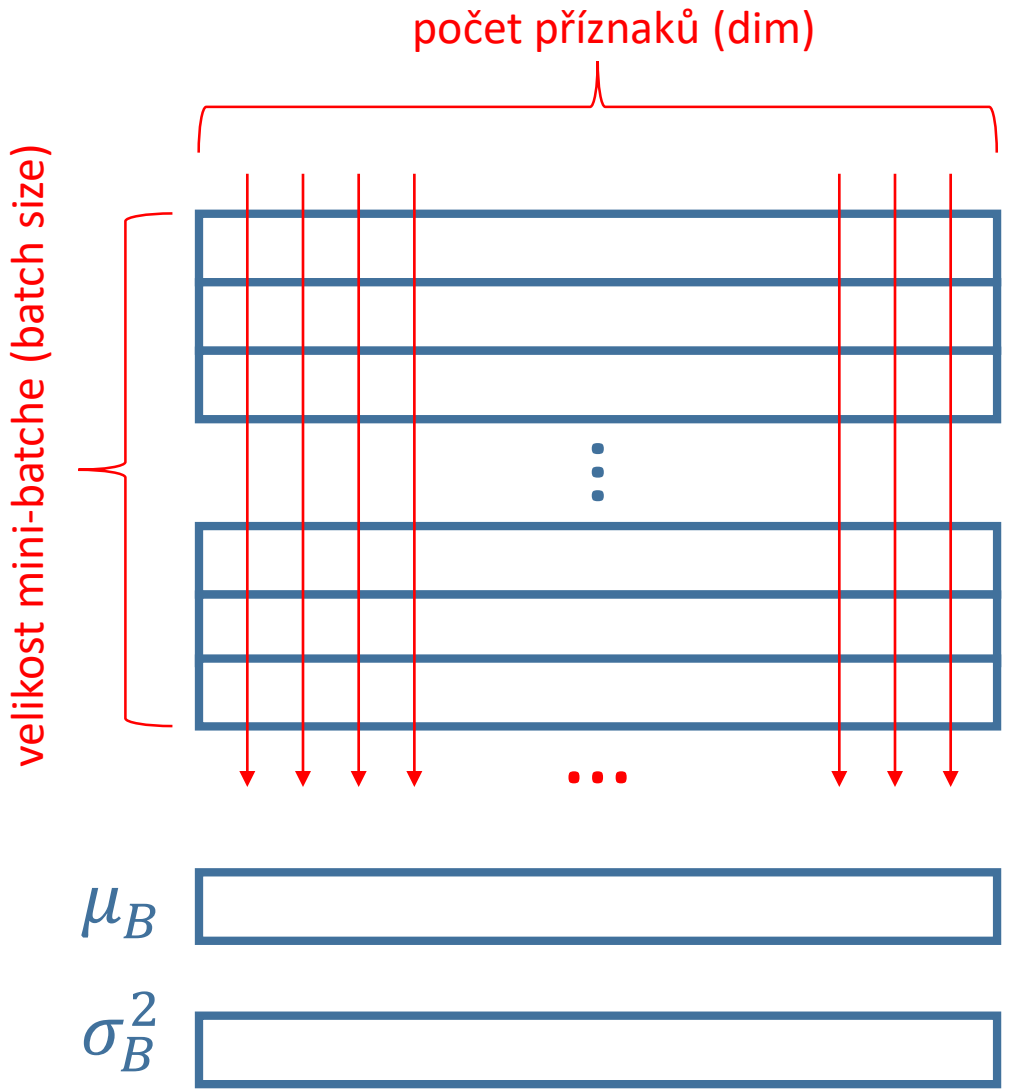
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

naučitelné parametry γ a β



Batch normalizace

- parametry γ a β umožňují nastavit si statistiky výstupu tak, jak to síti vyhovuje
  - pomáhá?
- umístit BN před nebo po nelinearitě?
  - doporučuje se různě
  - např. dle cs231n 2016/lec 5/slide 67 před
  - **dle výsledků však lepší po**

Výsledky pro RELU na ImageNet:

Name	Accuracy	LogLoss	Comments
Before	0.474	2.35	As in paper
Before + scale&bias layer	0.478	2.33	As in paper
After	**0.499**	**2.21**	
After + scale&bias layer	0.493	2.24	

- V testovací fázi se batch statistiky nepočítají
- Použije se naučené průměr a rozptyl z trénovacích dat
- Výpočet např. průměrováním se zapomínaním
- Nebo např. jedním průchodem natrénované sítě trénovacími daty

zdroj: https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md

Batch normalizace

☺obvykle zvyšuje úspěšnost

☺urychluje trénování, lze vyšší learning rate

☺snižuje potřebu dropout

☺snižuje závislost na inicializaci → robustnější

☺stabilní pokud batch size dostatečně velká

☹nepříliš vhodná pro rekurentní sítě

☹nic moc pro malé batche

☹různé chování v train a test

☹zpomaluje

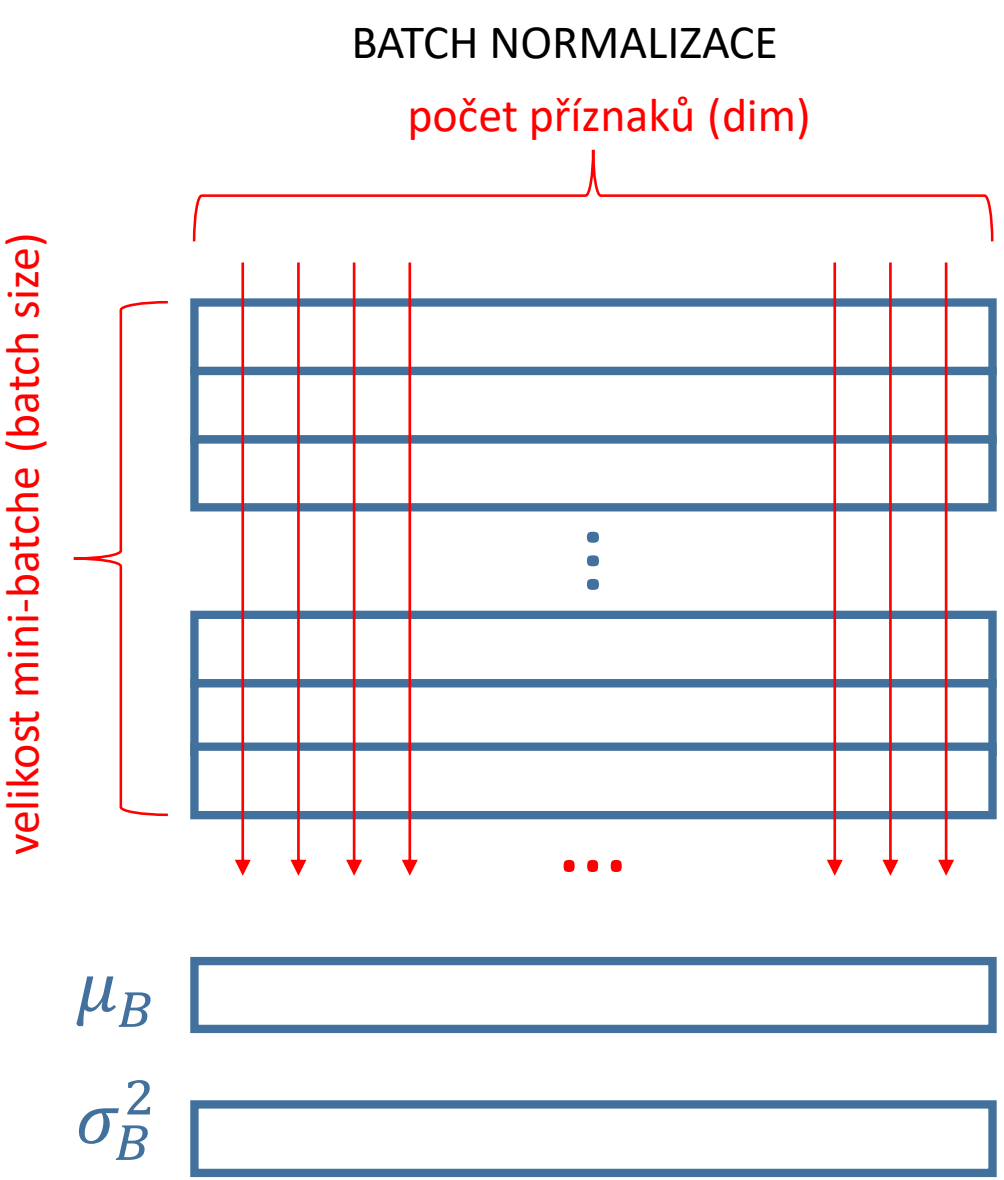
☹take závisí na nelinearitě

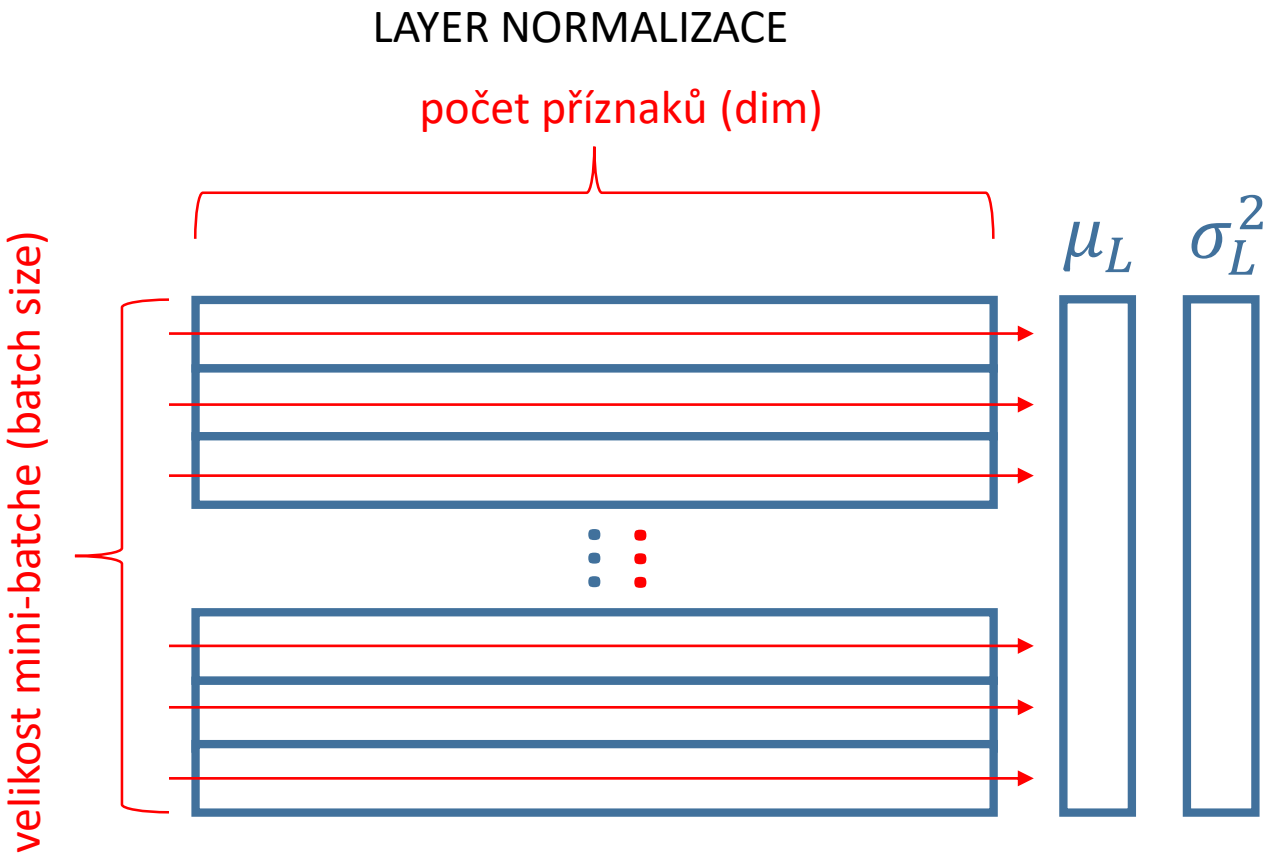
BN and activations

Name	Accuracy	LogLoss	Comments
ReLU	0.499	2.21	
RReLU	0.500	2.20	
PReLU	**0.503**	**2.19**	
ELU	0.498	2.23	
Maxout	0.487	2.28	
Sigmoid	0.475	2.35	
TanH	0.448	2.50	
No	0.384	2.96	

zdroj + další výsledky: https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md

Layer normalizace



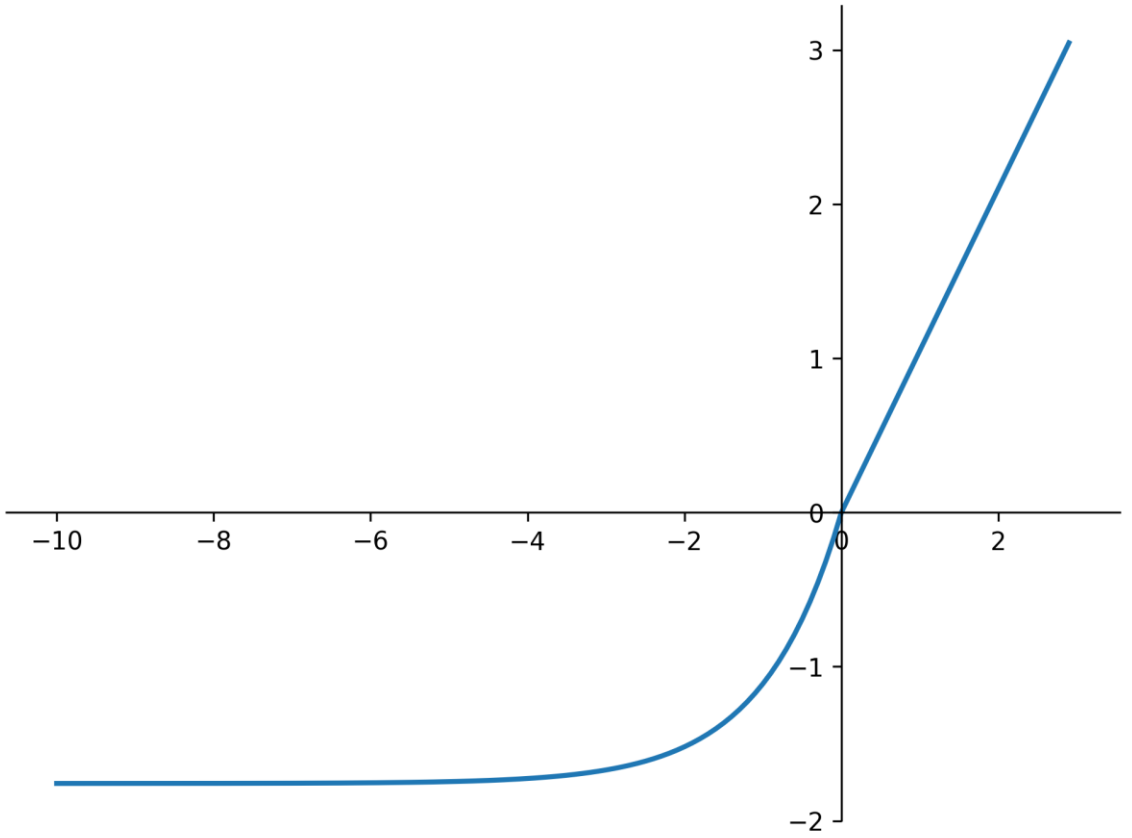


☹ Příliš nefunguje pro konvoluční sítě

Exponential Linear Unit (ELU)

- Snaží se kombinovat lineární a exp aktivace
- Přibližuje výstupní hodnoty nulovému průměru
- Scaled exponential linear units (SELU)
  - Klambauer et al.: “Self-Normalizing Neural Networks” (2017)
  - Cílem dosáhnout m=0 a std=1
  - Při správném nastavení λ a α nahrazuje batch normalizaci! → navíc urychluje!
  - $\lambda = 1.0507009873554804934193349852946$
    $\alpha = 1.6732632423543772848170429916717$

$$\mathrm{ELU}(x) = \lambda \begin{cases} x & \text{if} \quad x > 0 \\ \alpha \exp(x) - \alpha & \text{if} \quad x \leq 0 \end{cases}$$



Scaled Exponential Linear Unit (SELU)

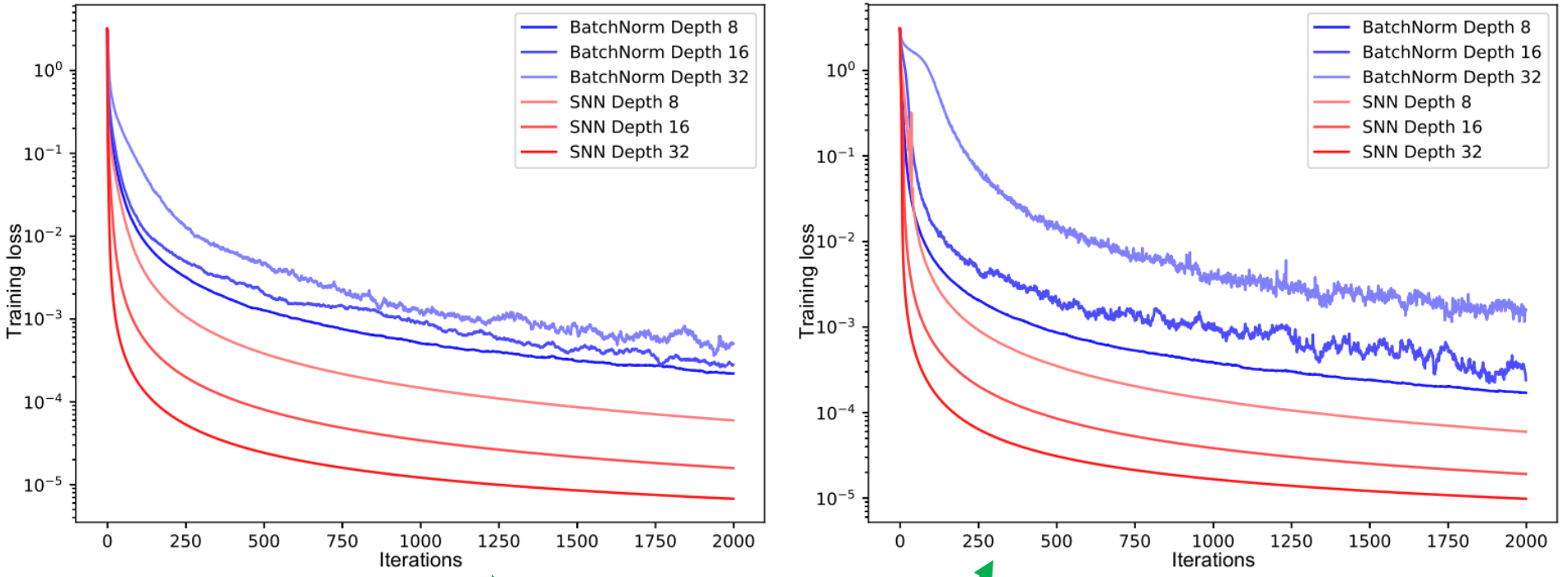
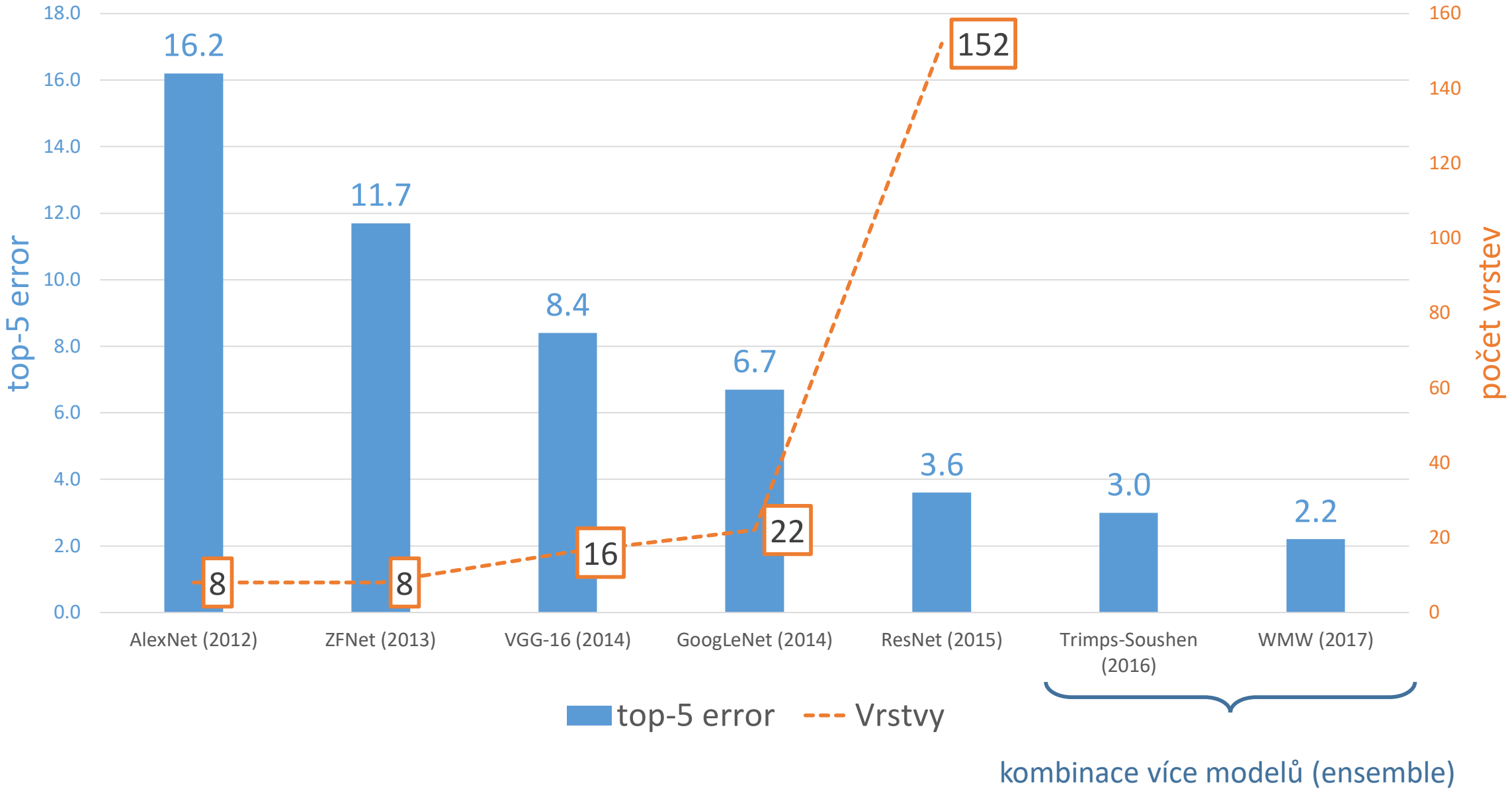


Figure 1: The left panel and the right panel show the training error (y-axis) for feed-forward neural networks (FNNs) with batch normalization (BatchNorm) and self-normalizing networks (SNN) across update steps (x-axis) on the MNIST dataset the CIFAR10 dataset, respectively. We tested networks with 8, 16, and 32 layers and learning rate 1e-5. FNNs with batch normalization exhibit high variance due to perturbations. In contrast, SNNs do not suffer from high variance as they are more robust to perturbations and learn faster.

zdroj: Klambauer et al.: “Self-Normalizing Neural Networks” (2017)

Další konvoluční sítě

ImageNet klasifikace

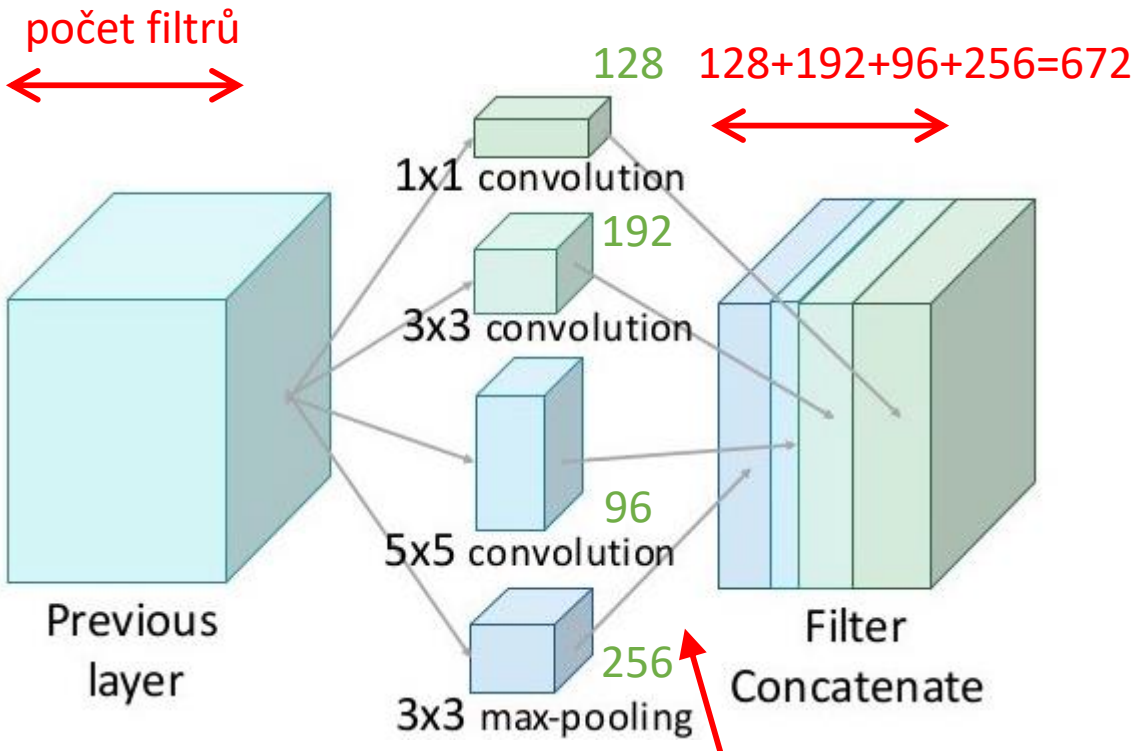


GoogLeNet (2014)

- 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



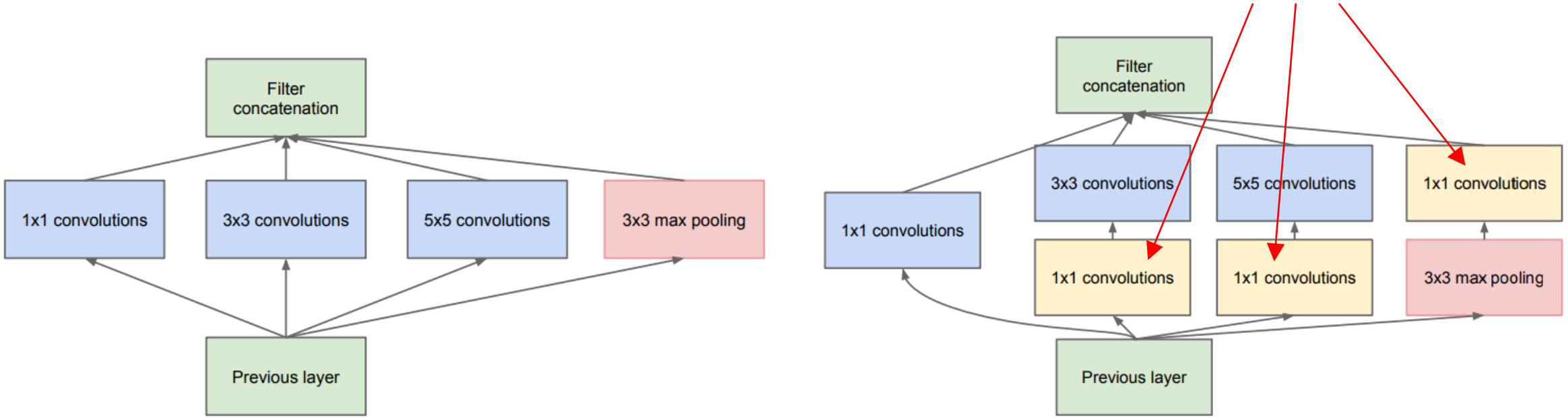
zero padding tak, aby výsledek konvoluce měl vždy stejnou velikost (mode='same')

obrázek: https://www.kdnuggets.com/2017/08/intuitive-guide-deep-network-architectures.html

Optimalizace Inception modulu

[Szegedy et al.: Going Deeper with Convolutions]

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



(a) Inception module, naïve version

(b) Inception module with dimension reductions

Figure 2: Inception module

ve skutečnosti použita tato varianta

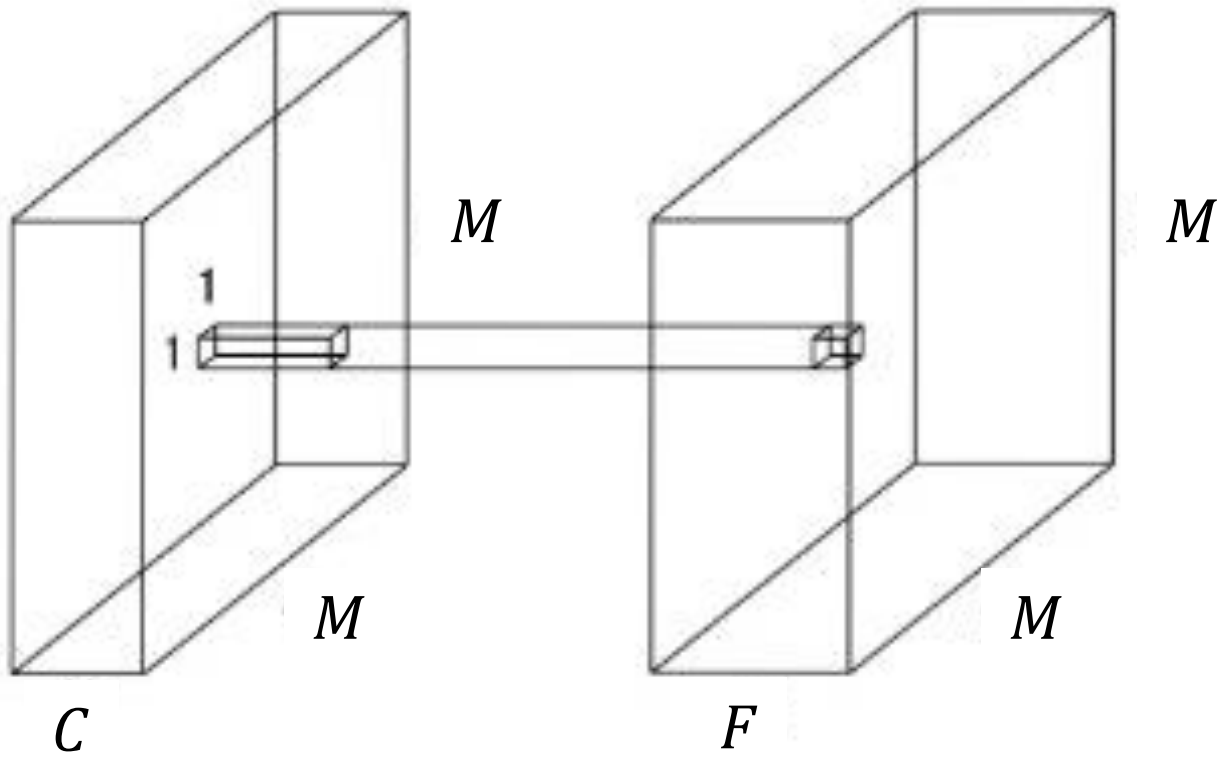
obrázek: https://arxiv.org/abs/1409.4842

1x1 konvoluce

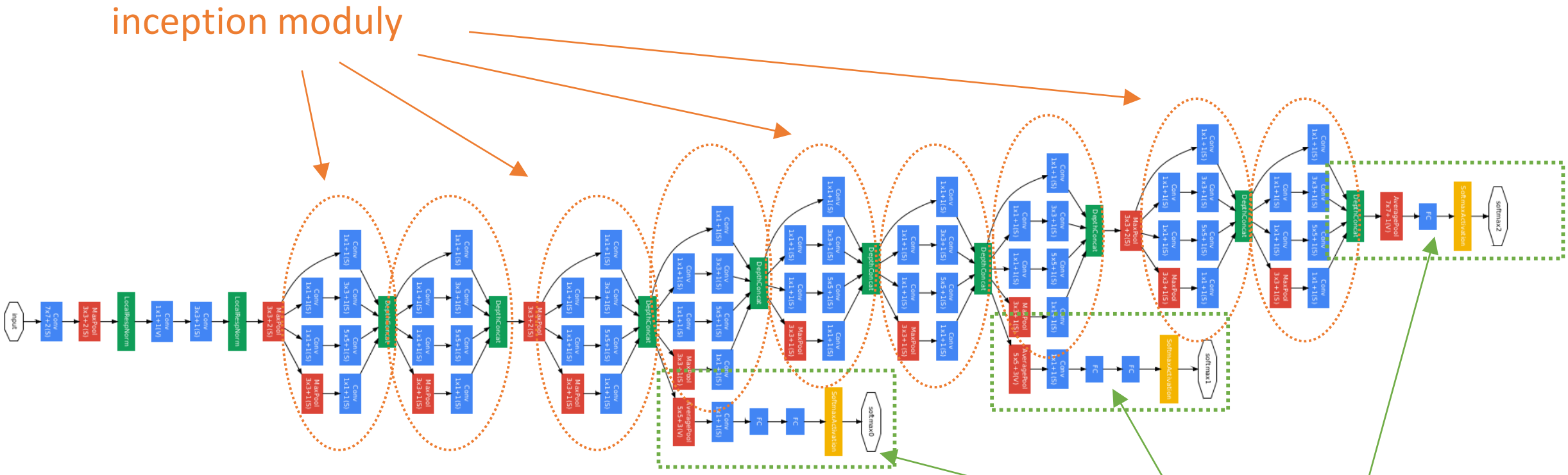
vzpomeňme: filtr vždy zahrnuje všechny kanály

tzn., že i při 1x1 konvoluci (bez okolí) je výsledek stále lineární kombinací přes kanály

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



GoogLeNet (2014)



- vítěz ImageNet 2014 s 6.7 % top-5 error
- celkem 22 vrstev
- žádné skryté fully-connected (lineární) vrstvy, téměř plně konvoluční síť
- 12x méně parametrů než AlexNet

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: s více vrstvami klesá i trénovací chyba!

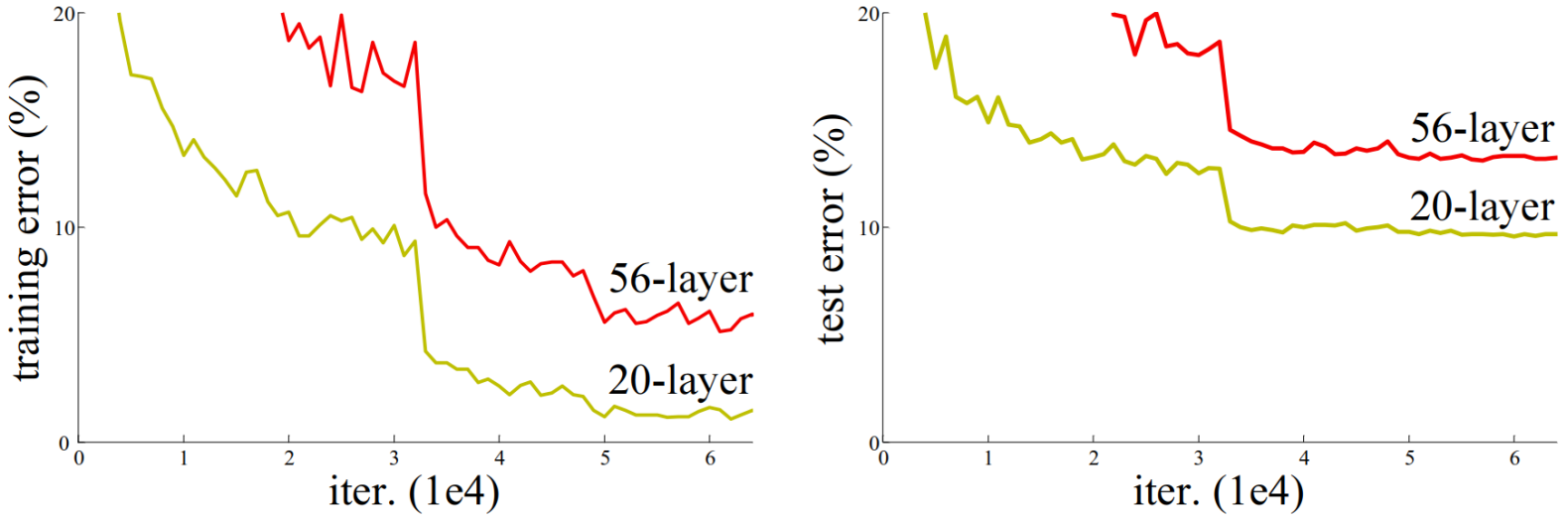


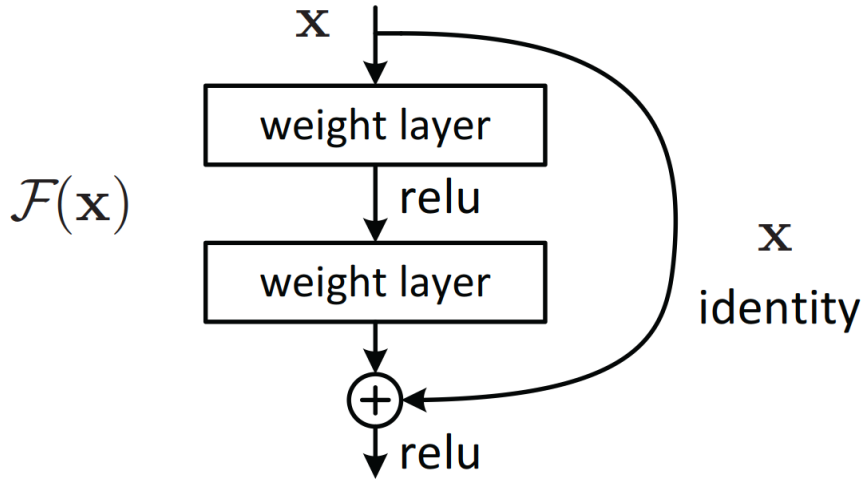
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

Reziduální blok

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

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

- "Naučit se nuly je jednodušší než identitu"



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

Figure 2. Residual learning: a building block.

Bottleneck residual blok

Podobně jako u Inception i ResNet optimalizuje pomocí 1x1 bottleneck vrstev uvnitř reziduálních bloků

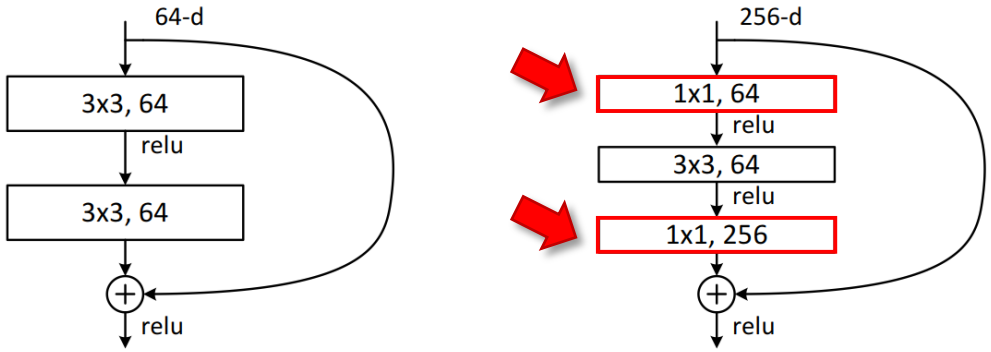
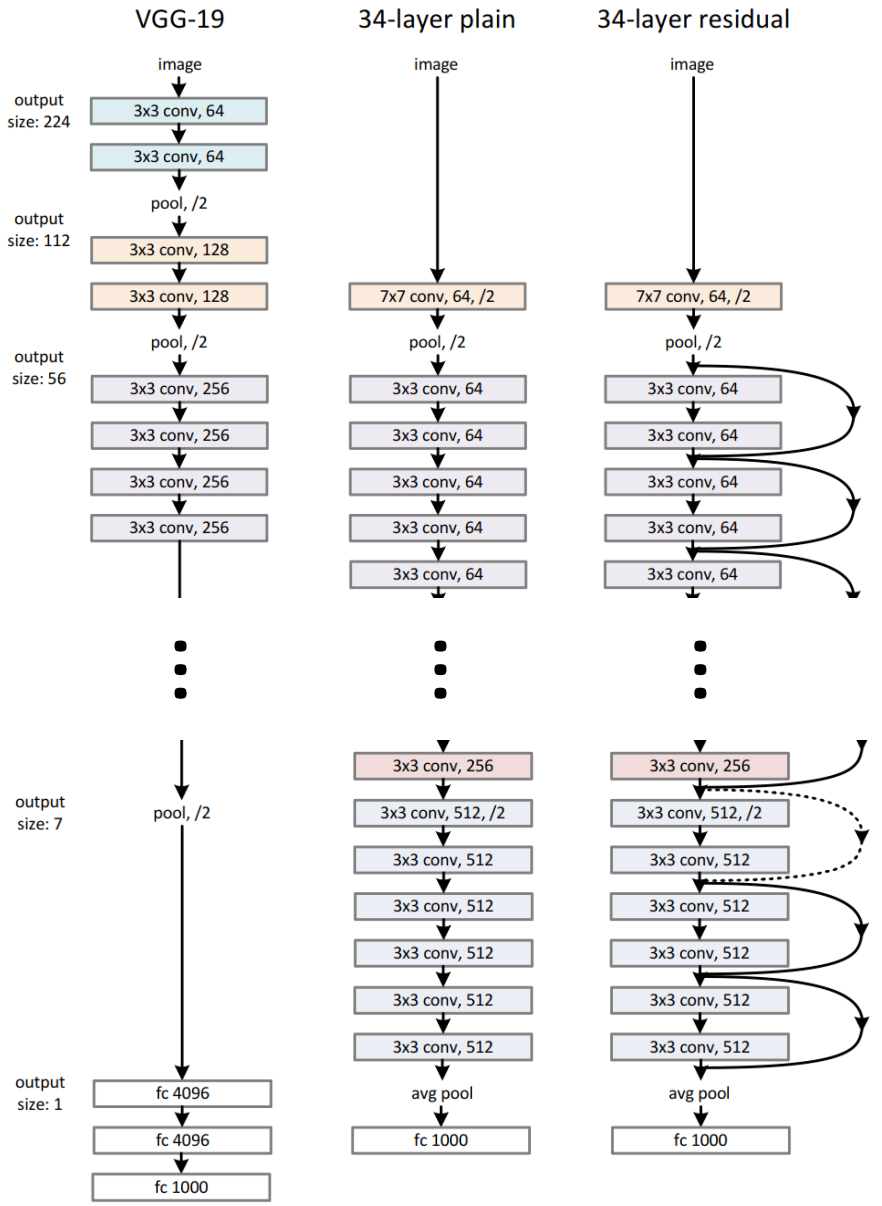


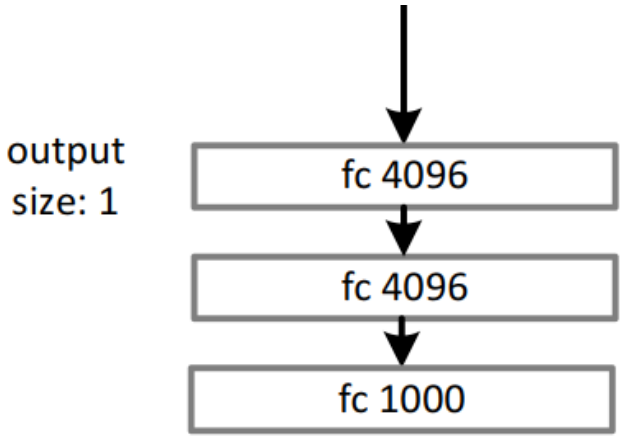
Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.



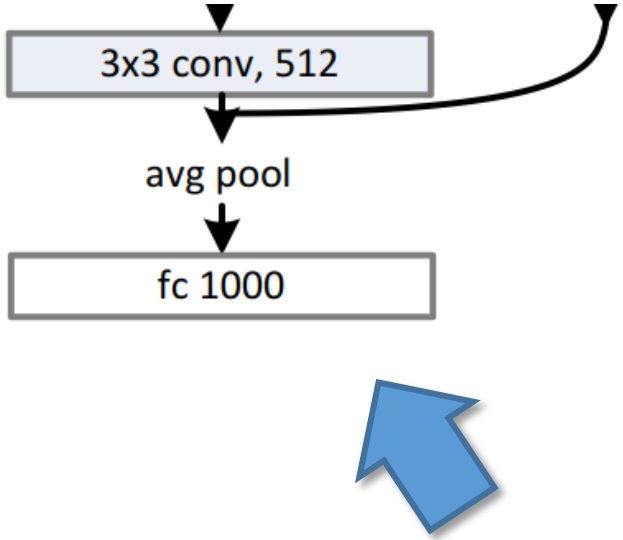
Téměř plně konvoluční

bez skrytých lineárních vrstev

VGG:

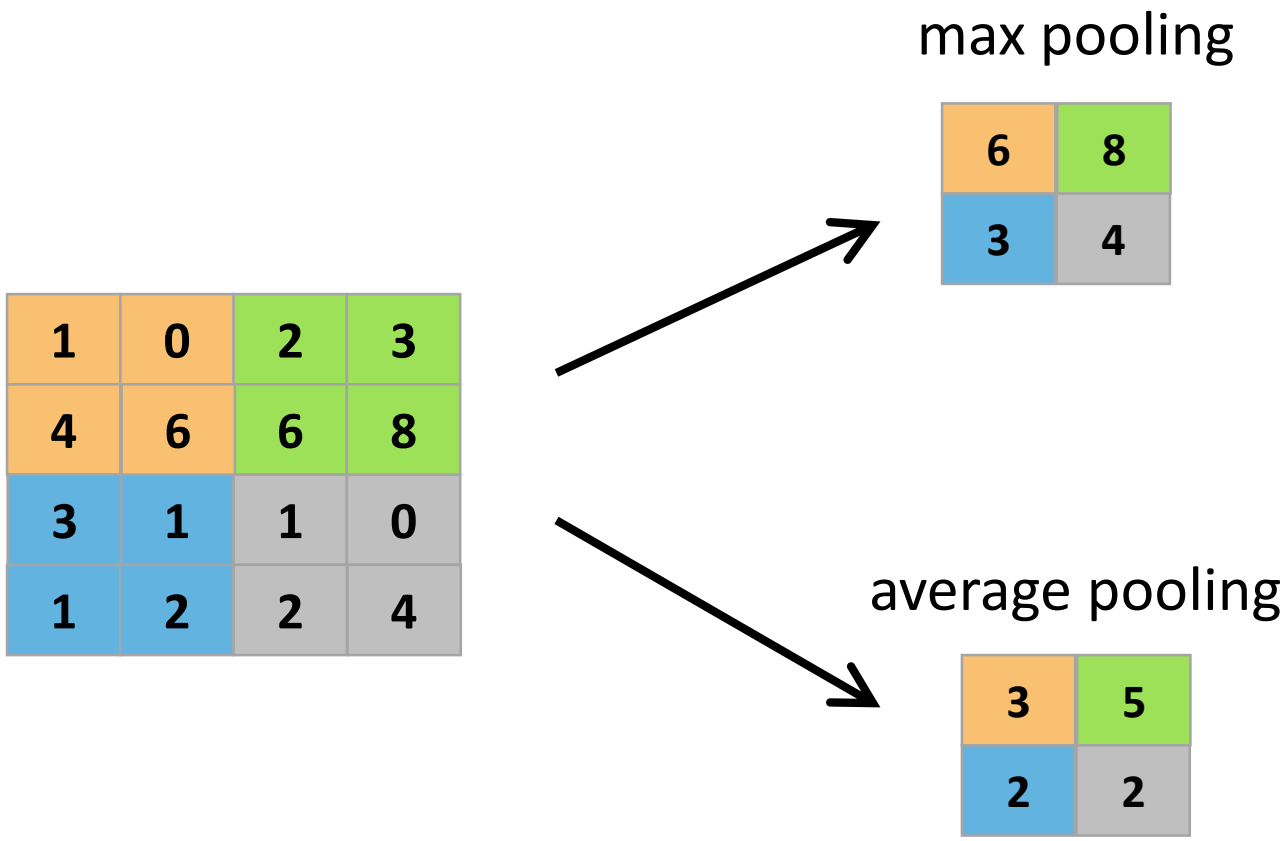


ResNet:



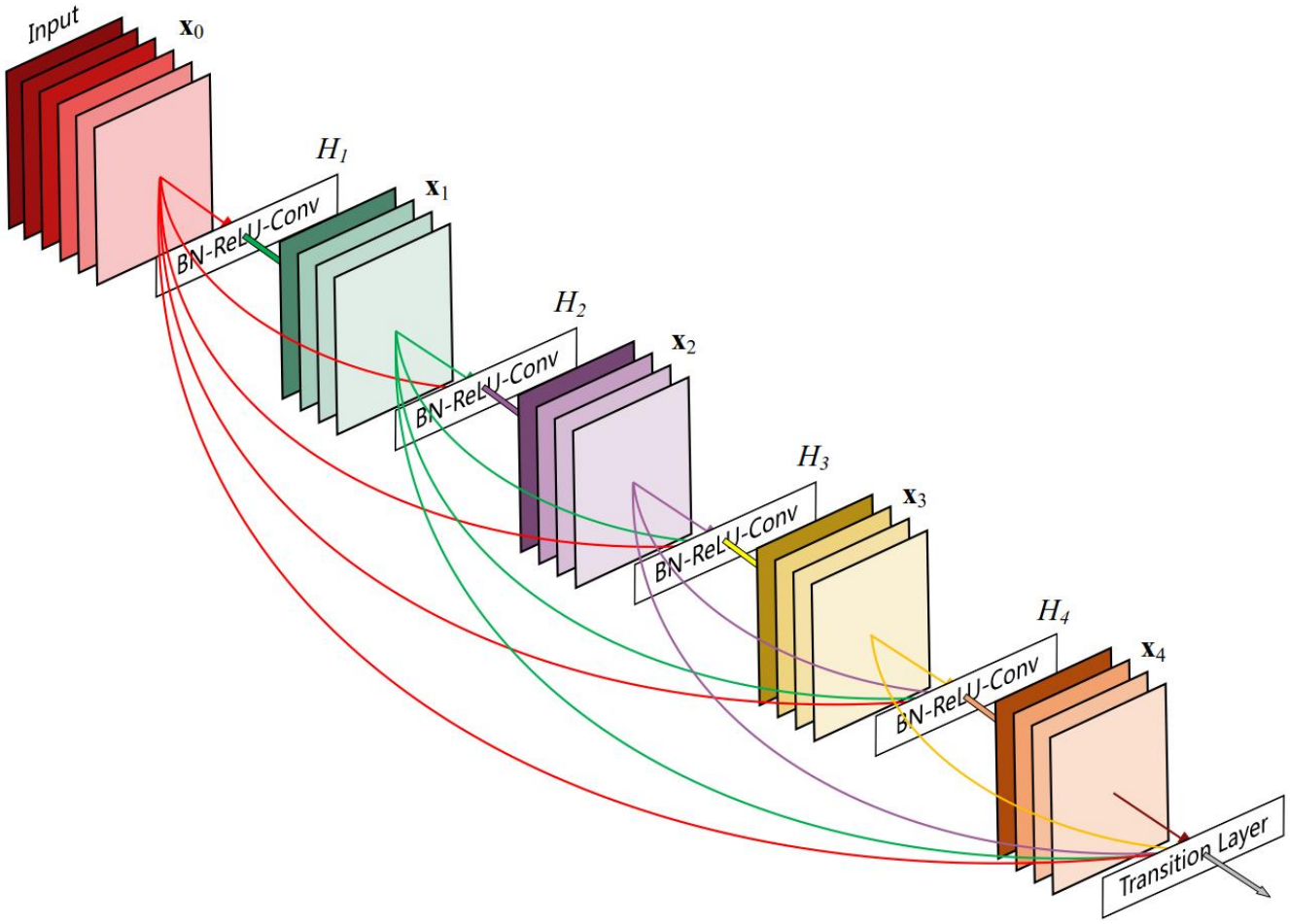
pouze lineární klasifikátor

Average pooling



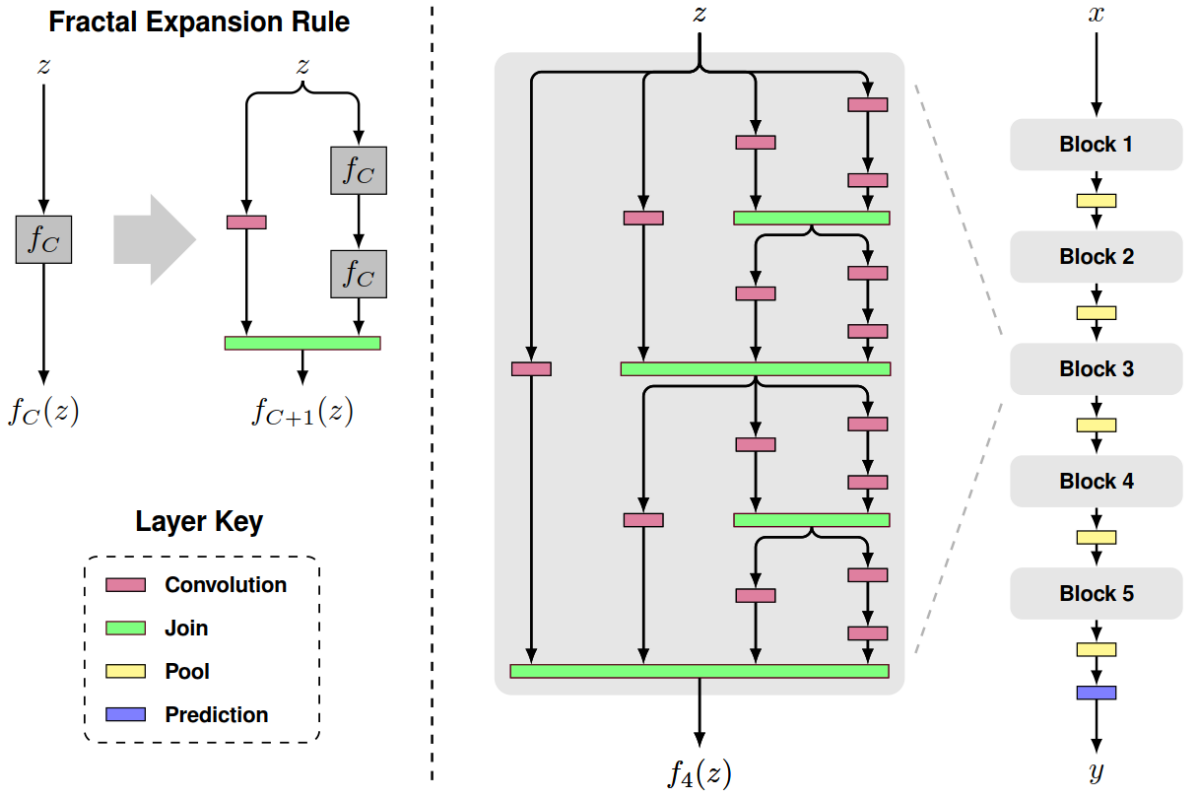
DenseNet (2016)

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



FractalNet (2017)

- Larsson et al.: “FractalNet: Ultra-Deep Neural Networks without Residuals”
- “Rekurzivní ResNet”



SqueezeNet (2016)

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

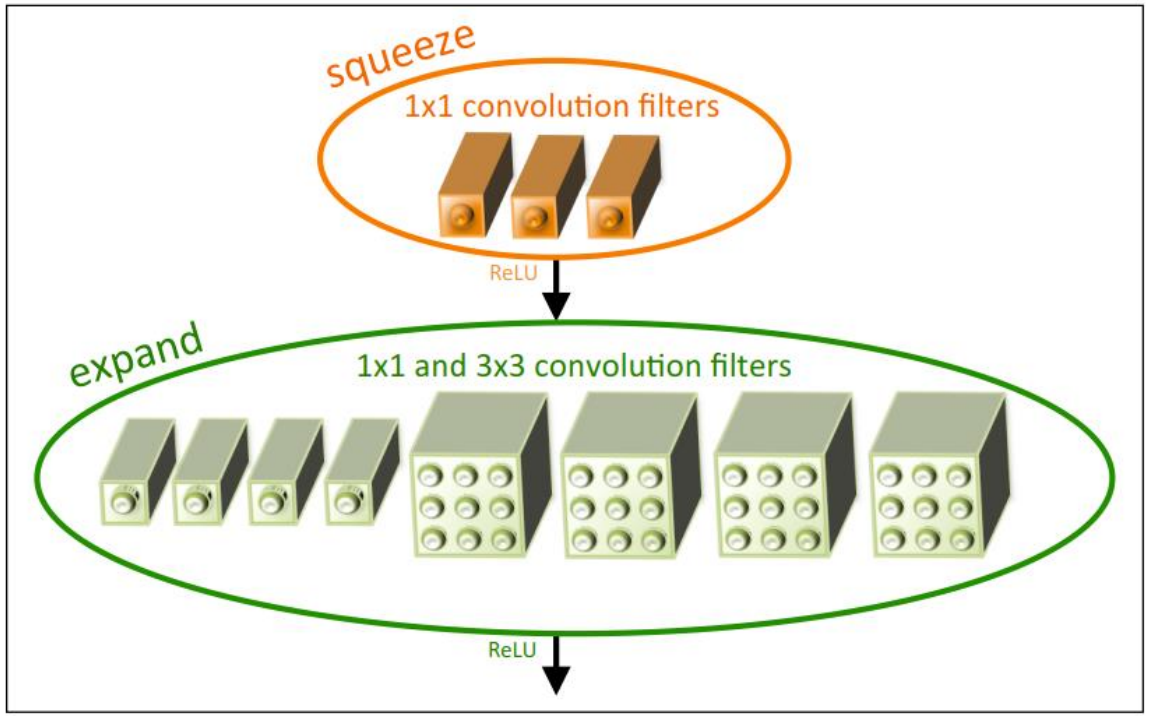
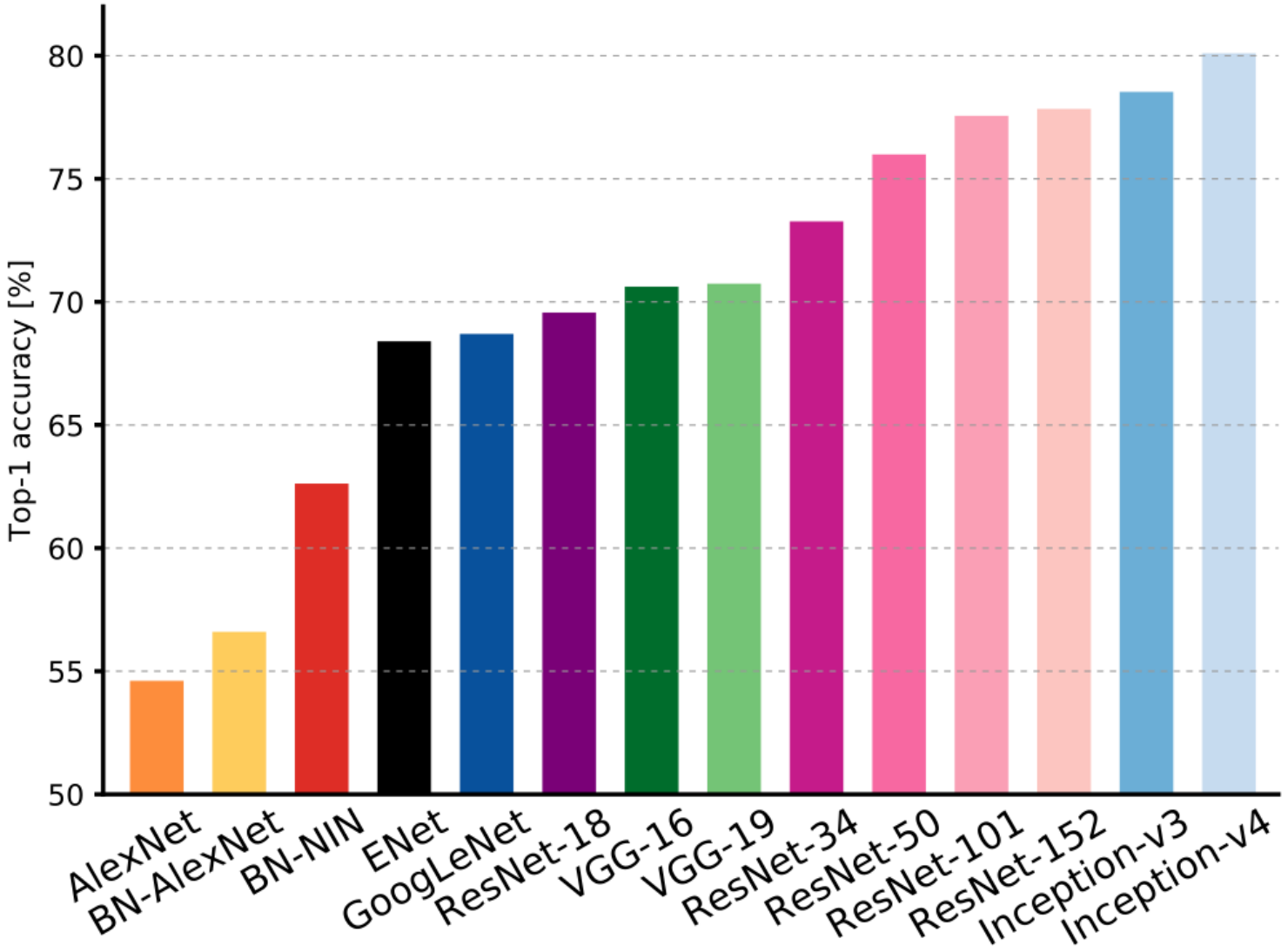


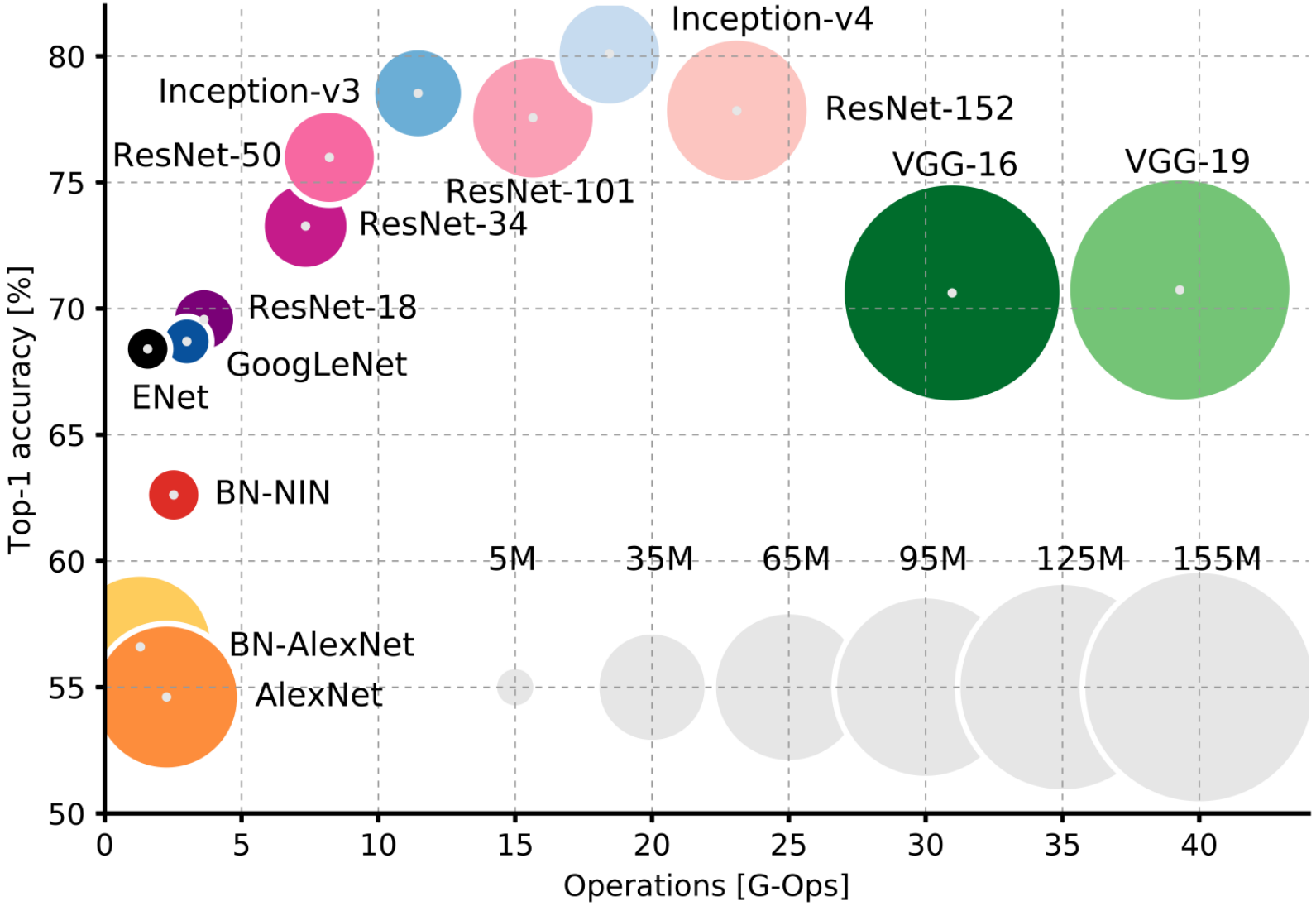
Figure 1: Microarchitectural view: Organization of convolution filters in the **Fire module**. In this example, $s_{1x1} = 3$, $e_{1x1} = 4$, and $e_{3x3} = 4$. We illustrate the convolution filters but not the activations.

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"

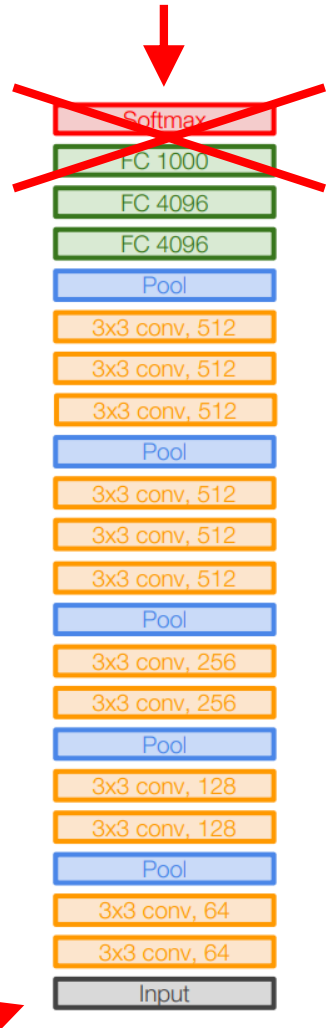
Transfer learning

Trénování konvolučních sítí při málo datech

- Popsané architektury mají obvykle miliony parametrů
- Malé datasety na jejich trénování nestačí → výrazný overfit
- I pokud data máme: trénování VGG na ImageNet trvalo autorům 2-3 týdny, a to i s 4x NVIDIA Titan Black GPU
- **Naštěstí lze obejít!**
  1. Můžeme vzít existující již natrénovaný model (např. VGG-16)
  2. Odstraníme poslední klasifikační vrstvu
  3. Nahradíme vlastní

Transfer learning

1000 tříd pro ImageNet



VGG16

ImageNet data

poslední lineární vrstvu tvaru 4096×1000 zahodíme

2 třídy: kočky vs psi

VGG16

připojíme vlastní lineární vrstvu tvaru 4096×2 a náhodně inicializujeme

pokud máme málo dat, je lepší konvoluční vrstvy netrénovat → **layer freeze**

můžeme použít vlastní data

Transfer learning: příklad v Kerasu

odstranění lineárních vrstev

použít verzi předtrénovanou na ImageNet

```
58  # build the VGG16 network
59  model = applications.VGG16(weights='imagenet', include_top=False)
60  print('Model loaded.')
```

přidat vlastní vršek sítě

```
62  # build a classifier model to put on top of the convolutional model
63  top_model = Sequential()
64  top_model.add(Flatten(input_shape=model.output_shape[1:]))
65  top_model.add(Dense(256, activation='relu'))
66  top_model.add(Dropout(0.5))
67  top_model.add(Dense(1, activation='sigmoid'))

69  # note that it is necessary to start with a fully-trained
70  # classifier, including the top classifier,
71  # in order to successfully do fine-tuning
72  top_model.load_weights(top_model_weights_path)
73
74  # add the model on top of the convolutional base
75  model.add(top_model)
```

layer freezing: nech konvoluční vrstvy na pokoji

```
77  # set the first 25 layers (up to the last conv block)
78  # to non-trainable (weights will not be updated)
79  for layer in model.layers[:25]:
80      layer.trainable = False
```

vlastní data

```
88  # prepare data augmentation configuration
89  train_datagen = ImageDataGenerator(
90      rescale=1. / 255,
91      shear_range=0.2,
92      zoom_range=0.2,
93      horizontal_flip=True)

97  train_generator = train_datagen.flow_from_directory(
98      train_data_dir,
99      target_size=(img_height, img_width),
100     batch_size=batch_size,
101     class_mode='binary')
```

fine-tuning

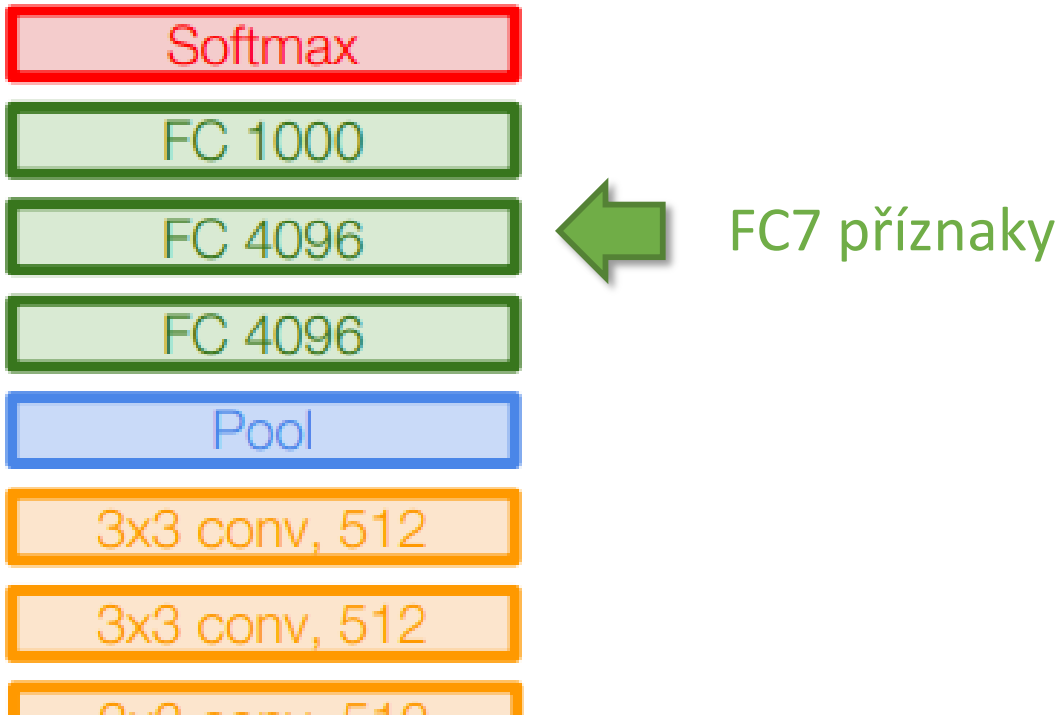
```
109 # fine-tune the model
110 model.fit_generator(
111     train_generator,
112     samples_per_epoch=nb_train_samples,
113     epochs=epochs,
114     validation_data=validation_generator,
115     nb_val_samples=nb_validation_samples)
```

https://gist.github.com/fchollet/7eb39b44eb9e16e59632d25fb3119975

CNN příznaky

- Výstup z posledních lineárních vrstev lze použít např. jako příznaky (tzv. FC7) → CNN jako “feature extractor”
- Např. VGG-16 předposlední vrstva má rozměr 4096
- Nad těmito příznaky je možné natrénovat libovolný klasifikátor, třeba i rozhodovací stromy/lesy, bayesovské klasifikátory, …
- Lze take využít pro urychlení trénování: celý dataset projet sítí a pro každý obrázek uložit na disk FC7 příznaky
- Během trénování se pak nemusí znovu a znovu provádět dopředný průchod celou sítí, pouze těmi posledními

např. VGG-16:



Transfer learning: shrnutí

specifičtější příznaky (obličeje, text, …)

obecné příznaky (hrany, bloby, …)

	podobná data	**odlišná data**
málo dat	trénovat spíše jen poslední vrstvu	problém ☺
hodně dat	fine tune několika vrstev (lze ale i celou síť)	fine tune více vrstev nebo i celé sítě

slide: http://cs231n.stanford.edu/

Shrnutí

Návrh vlastní sítě

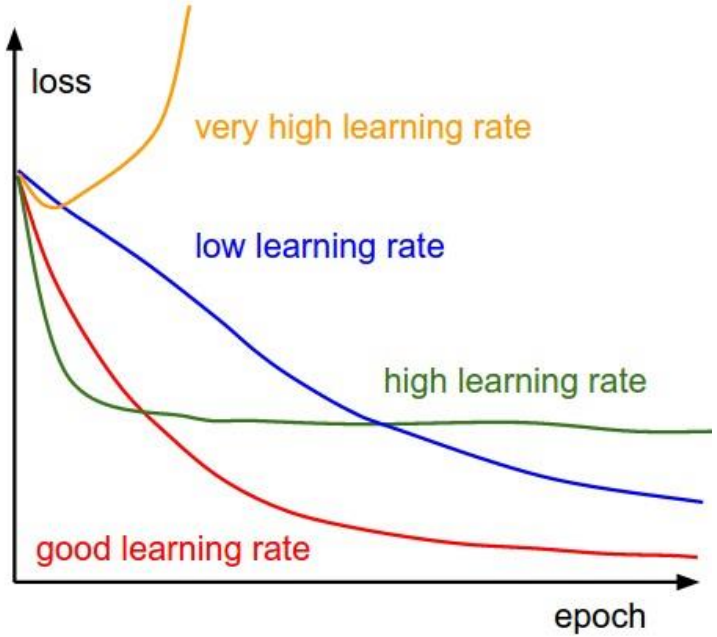
- méně parametrů, více nelinearit
- využívat sdílení parametrů → např. více konvoluce, méně lineárních vrstev
- použít spíše ReLU-like nelinearity, sigmoid ne
- použít batchnorm, pravděpodobně pomůže
- vršek sítě dle úlohy:
  - klasifikace = sigmoid / softmax
  - regrese = bez nelinearity
- pokud navrhujeme vlastní vrstvy a nelze využít autograd → gradient check

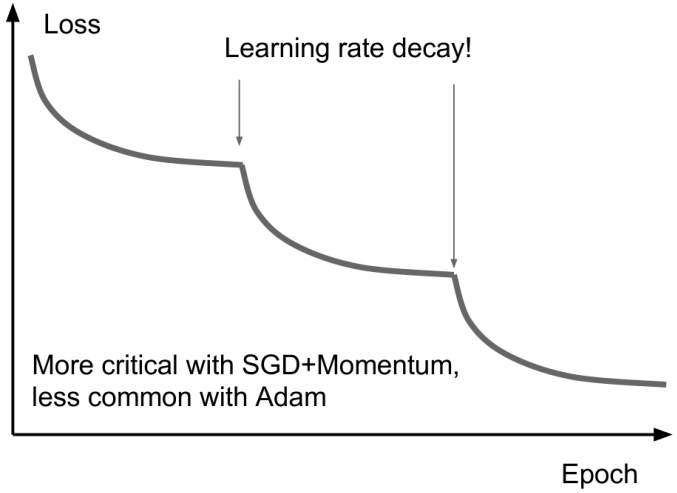
Aplikace existující sítě (transfer learning)

- raději implementace na githubu než se snažit o vlastní – často velmi obtížné replikovat pouze z článku
- stáhnout předučené váhy
- u menších sítí jako AlexNet lze natrénovat “from scratch”, u větších velmi obtížné
- pokud máme málo dat, zablokovat trénování u nižších vrstev

Trénování

- Ověřit overfittingem: zkusit malý vzorek, na nemž model musí dosáhnout 100%
- Monitorovat hodnotu lossu a podle toho nastavit learning rate
- Poté případně podle průběhu postupně learning rate snižovat
  - existují i alternující schémata, viz např. Smith: "Cyclical Learning Rates for Training Neural Networks"
- Pokud vše funguje, zkusit optimalizovat hyperparametry

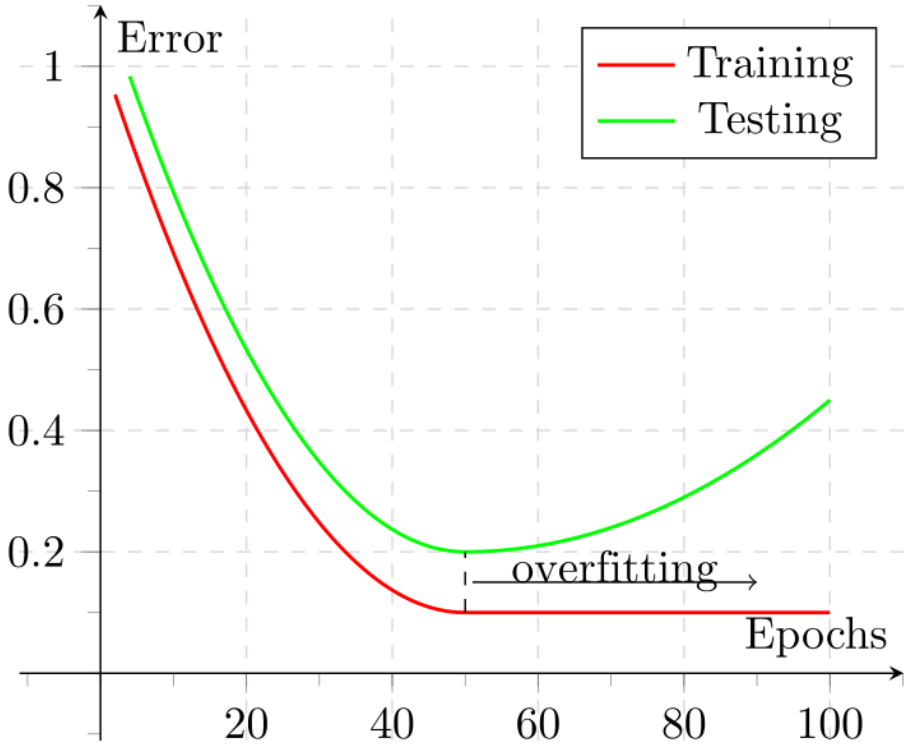




obrázky: http://cs231n.stanford.edu/

Prevence overfitu & optimalizace skóre

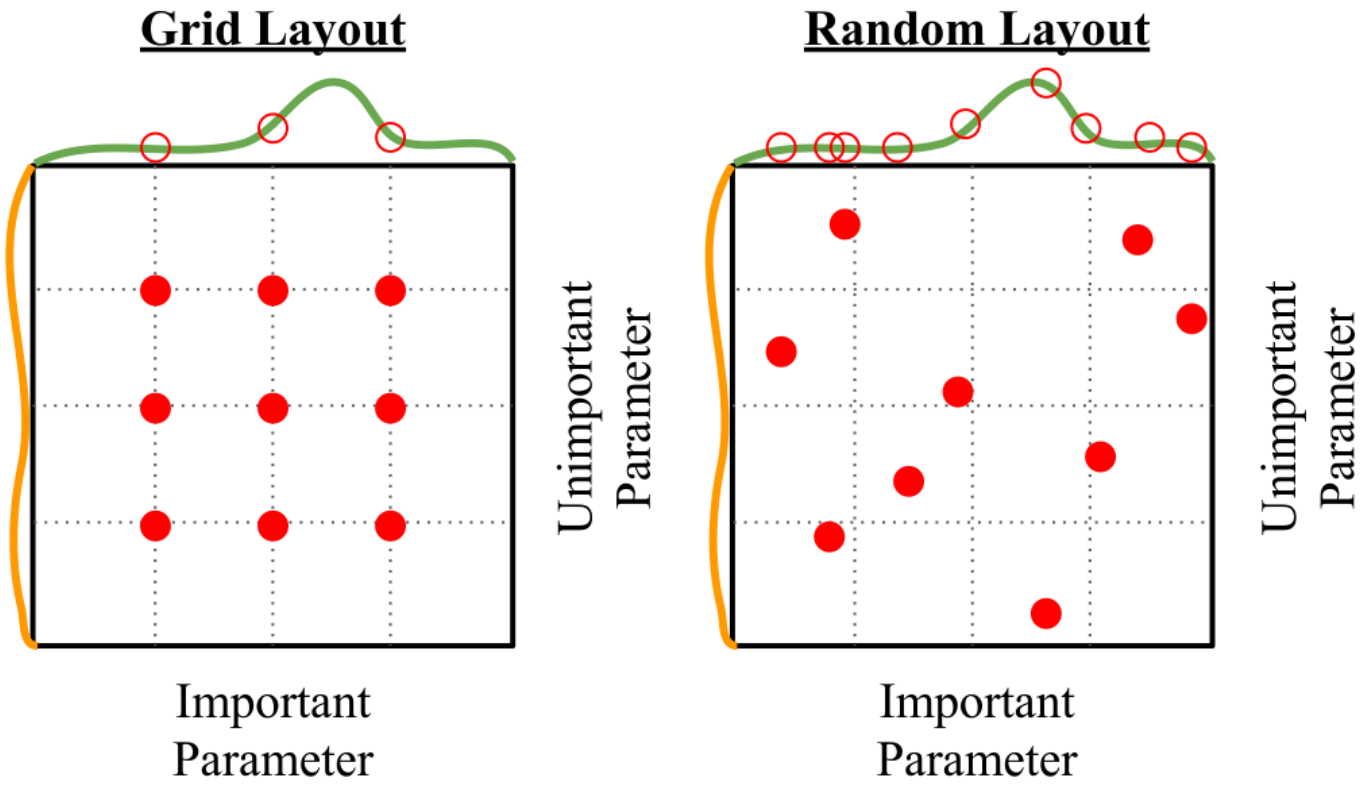
1. Nasbírat více dat
2. Uměle rozšířit data
3. Aplikovat vhodnější architekturu sítě
4. Pokud overfit: regularizace, dropout
5. Pokud stále overfit: zmenšit síť



obrázek: https://commons.wikimedia.org/wiki/File:2d-epochs-overfitting.svg

Optimalizace hyperparametrů

Co když výsledné skóre závisí více na jednom parametru než na jiném?



Náhodné zkoušení lépe pokrývá prostor možností než předdefinované kombinace

obrázek: http://cs231n.stanford.edu/