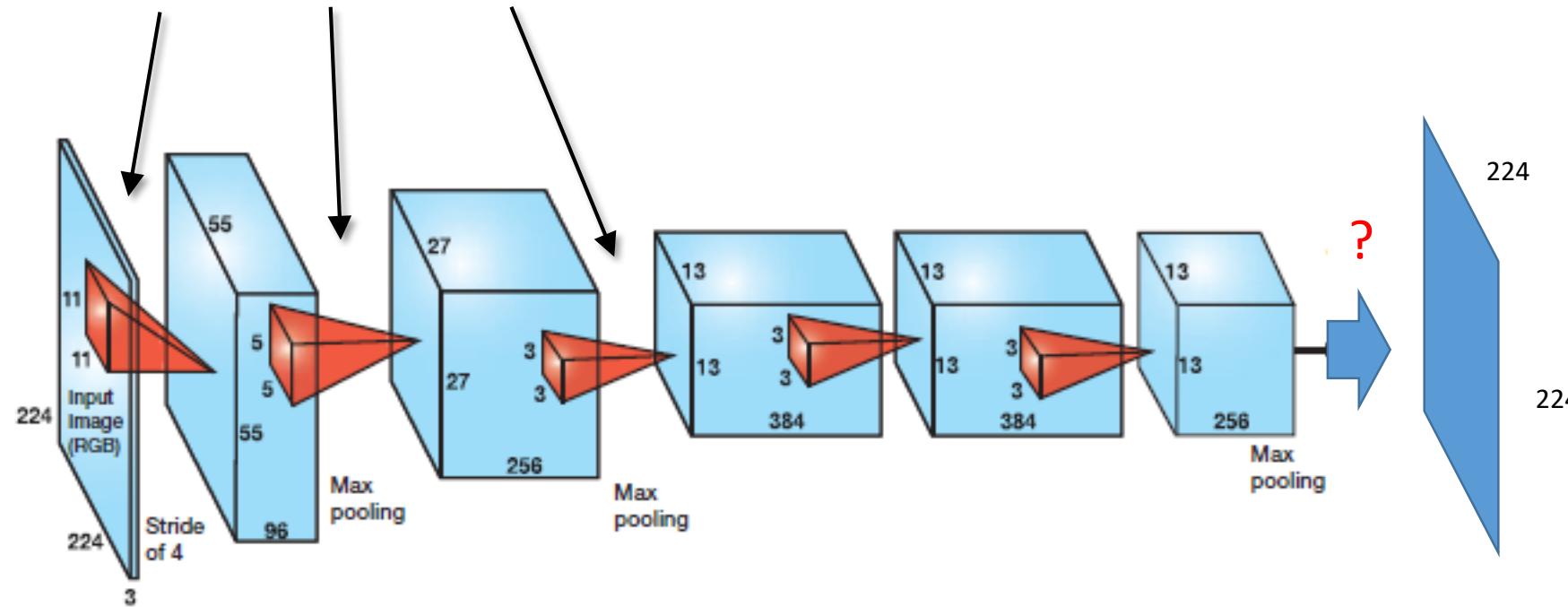


Aplikace neuronových sítí

Transponovaná konvoluce, generativní modely

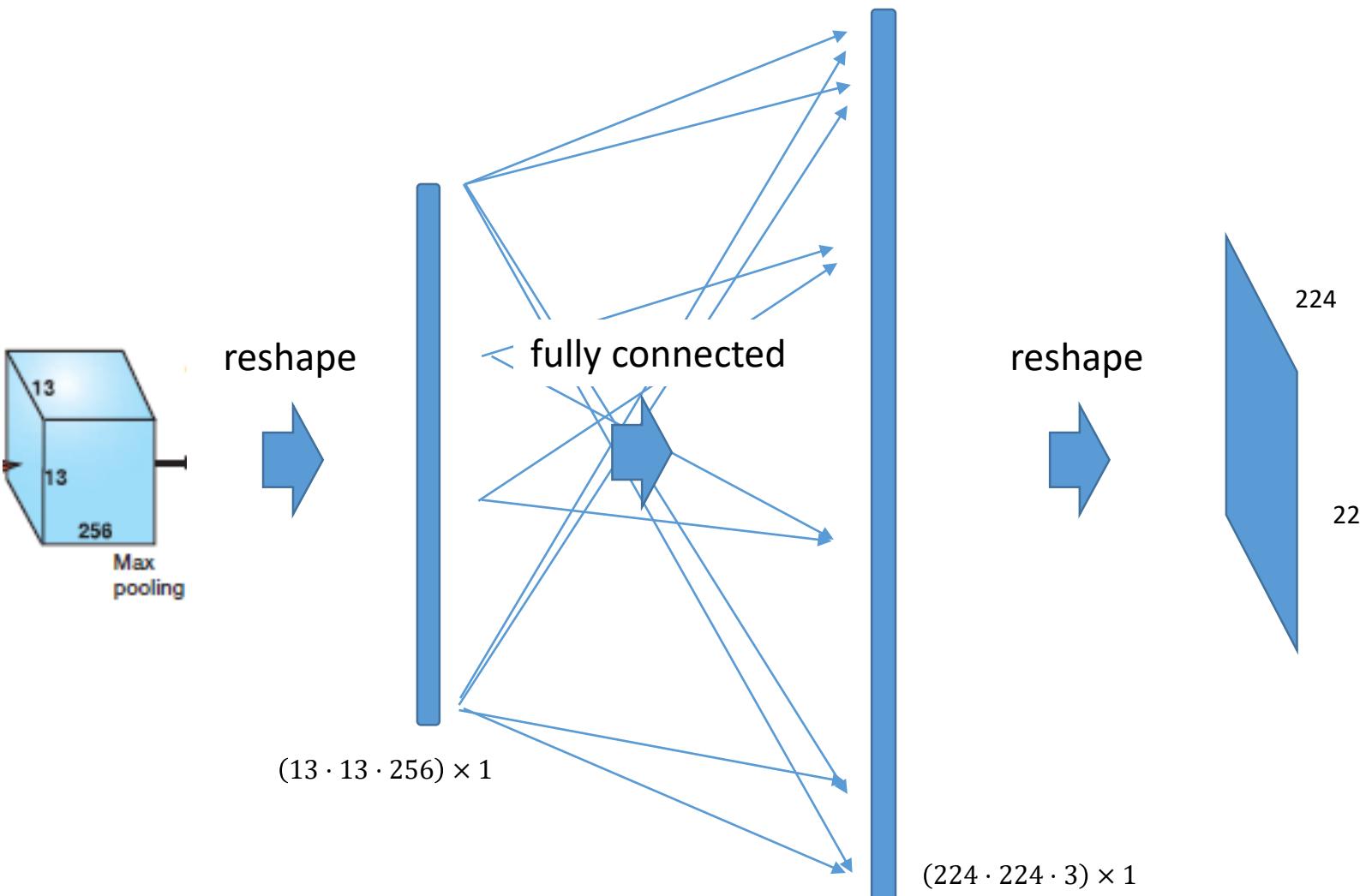
Obrázek jako výstup

2D max pooling či konvoluce s krokem (stride) > 1 pouze zmenšují



jak vyrobit/rekonstruovat obrázek v původním
nebo vyšším rozlišení, než je poslední vrstva?

Triviálně fully connected vrstvou

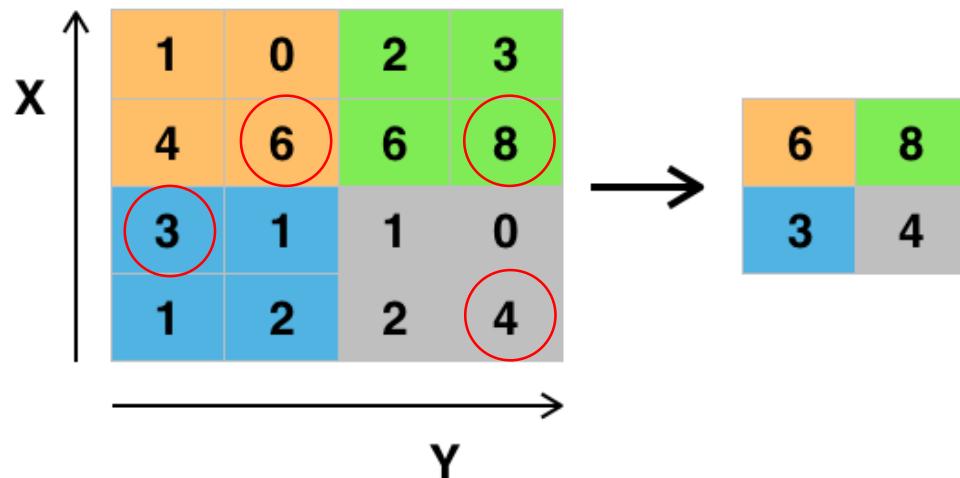


problém: počet parametrů = $13 \cdot 13 \cdot 256 \cdot 224 \cdot 224 \cdot 3 = 6.5 \cdot 10^9$ (!!)

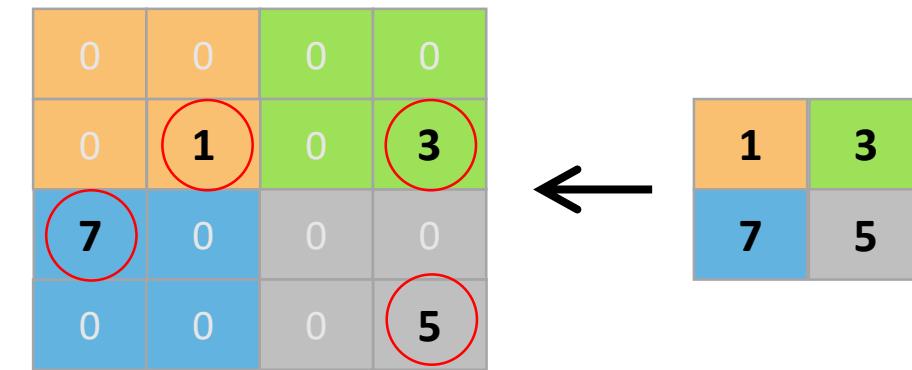
Max Pooling

dopředný průchod

Single depth slice



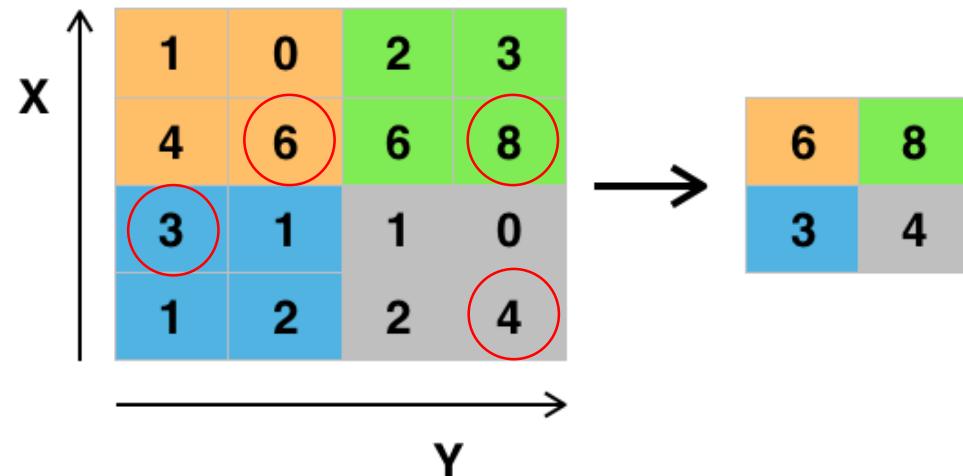
zpětný průchod



Max Unpooling

Max pooling

Single depth slice



Max unpooling

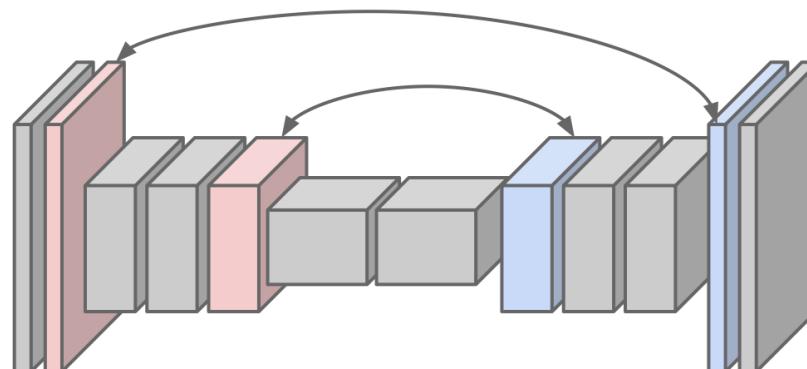
The diagram shows a 2x2 output matrix with values 6, 8, 1, 3. Red circles indicate the positions of the maximum values (6, 8) in the original 4x4 input. The resulting 4x4 output matrix has these values at their original positions, with zeros elsewhere. Red circles also highlight the positions of the maximum values (1, 3) in the output matrix.

0	0	0	0
0	1	0	3
7	0	0	0
0	0	0	5

zapamatují se pozice maxim

předpoklad symetrické konvoluční sítě se stejným počtem poolingů i unpoolingu

hodnoty se zapíšou na pozice dle pozice maxim z odpovídajícího poolingu v první části sítě



všimněme si: připomíná
zpětný průchod max poolingu

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

1	2
3	4

vstup: 2x2

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

výstup: 4x4

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

všuse, kde filtr překrývá výstup, se
přičte vstup krát váhy filtru

1	2
3	4

vstup: 2x2

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

výstup: 4x4

váhy konvolučního filtru

1	-1	2
1	-2	1
-1	2	-1

x 1



1	-1	2
1	-2	1
-1	2	-1

toto se přičte k výstupu
na odpovídající pozici

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

všuse, kde filtr překrývá výstup, se
přičte vstup krát váhy filtru

1	2
3	4

vstup: 2x2

-2	1	0	0
2	-1	0	0
0	0	0	0
0	0	0	0

výstup: 4x4

váhy konvolučního filtru

1	-1	2
1	-2	1
-1	2	-1

x 1



1	-1	2
1	-2	1
-1	2	-1

toto se přičte k výstupu
na odpovídající pozici

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

všuse, kde filtr překrývá výstup, se
přičte vstup krát váhy filtru

1	2
3	4

vstup: 2x2

-2	1	0	0
2	-1	0	0
0	0	0	0
0	0	0	0

výstup: 4x4

váhy konvolučního filtru

1	-1	2
1	-2	1
-1	2	-1

x 2



2	-2	4
2	-4	2
-2	4	-2

toto se přičte k výstupu
na odpovídající pozici

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

všuse, kde filtr překrývá výstup, se
přičte vstup krát váhy filtru

1	2
3	4

vstup: 2x2

-2	3	-4	2
2	-3	4	-2
0	0	0	0
0	0	0	0

výstup: 4x4

váhy konvolučního filtru

1	-1	2
1	-2	1
-1	2	-1

x 2



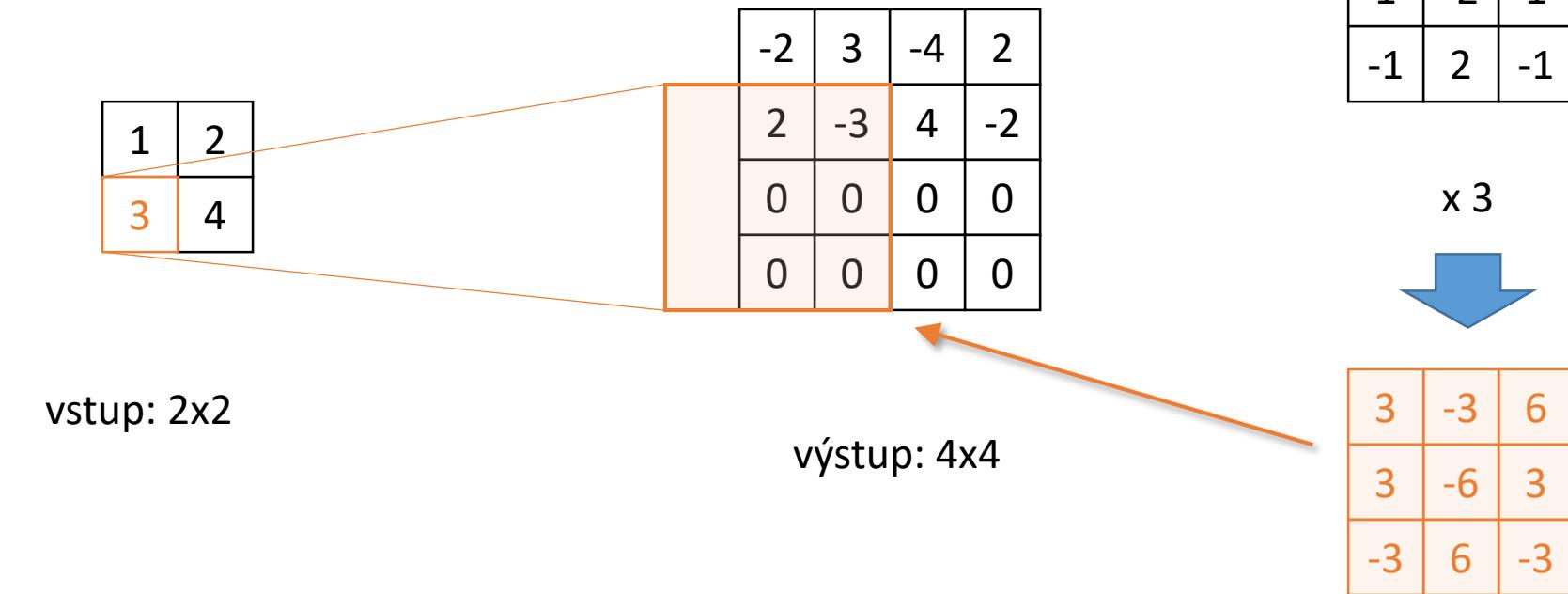
2	-2	4
2	-4	2
-2	4	-2

toto se přičte k výstupu
na odpovídající pozici

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

všuse, kde filtr překrývá výstup, se
přičte vstup krát váhy filtru



váhy konvolučního filtru

1	-1	2
1	-2	1
-1	2	-1

x 3



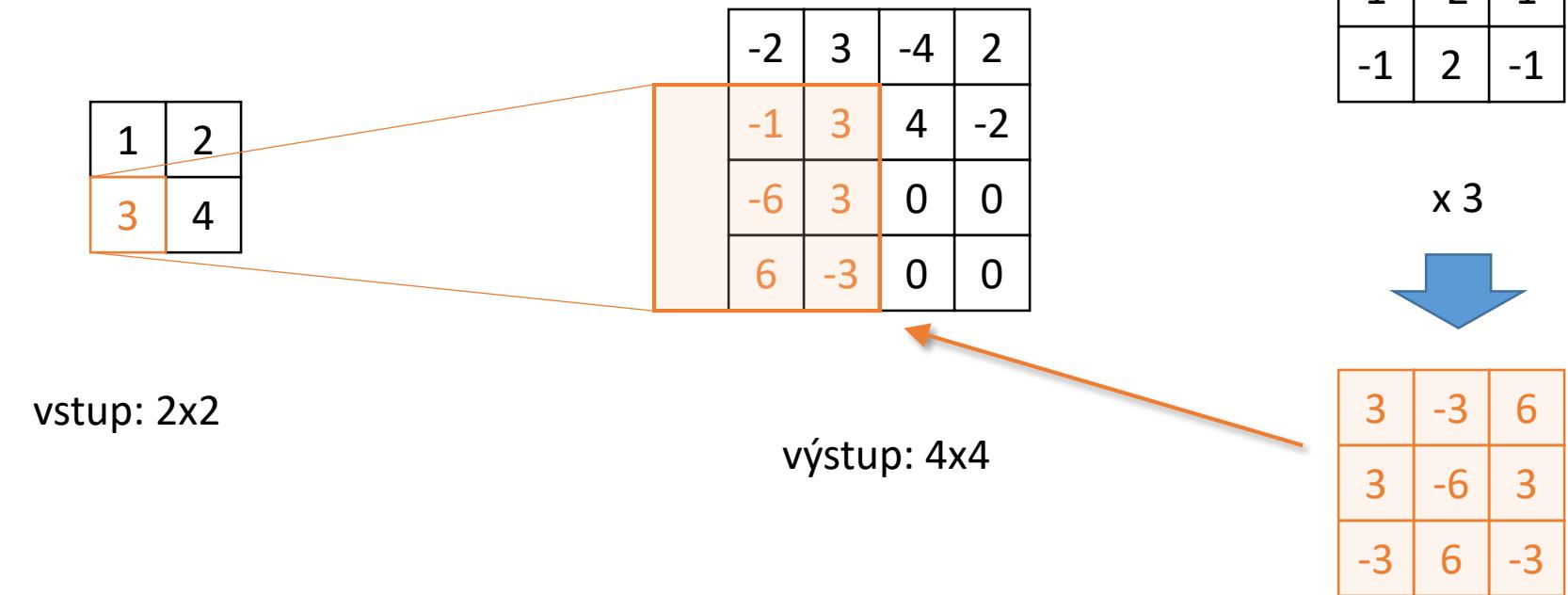
3	-3	6
3	-6	3
-3	6	-3

toto se přičte k výstupu
na odpovídající pozici

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

všuse, kde filtr překrývá výstup, se
přičte vstup krát váhy filtru



toto se přičte k výstupu
na odpovídající pozici

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

všuse, kde filtr překrývá výstup, se
přičte vstup krát váhy filtru

1	2
3	1

vstup: 2x2

-2	3	-4	2
-1	3	4	-2
-6	3	0	0
6	-3	0	0

výstup: 4x4

váhy konvolučního filtru

1	-1	2
1	-2	1
-1	2	-1

x 1



1	-1	2
1	-2	1
-1	2	-1

toto se přičte k výstupu
na odpovídající pozici

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

všuse, kde filtr překrývá výstup, se
přičte vstup krát váhy filtru

1	2
3	1

vstup: 2x2

-2	3	-4	2
-1	4	3	0
-6	4	-2	1
6	-4	2	-1

výstup: 4x4

váhy konvolučního filtru

1	-1	2
1	-2	1
-1	2	-1

x 1



1	-1	2
1	-2	1
-1	2	-1

toto se přičte k výstupu
na odpovídající pozici

Transponovaná konvoluce

transponovaná konvoluce 3x3, stride 2, padding 1

všuse, kde filtr překrývá výstup, se
přičte vstup krát váhy filtru

1	2
3	1

vstup: 2x2

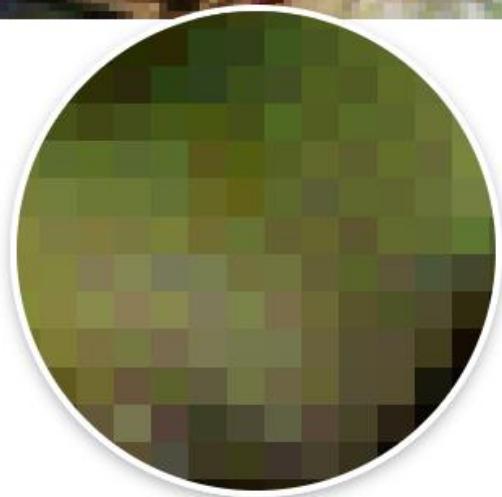
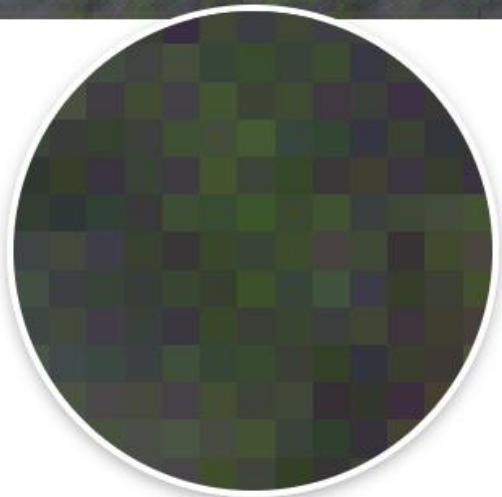
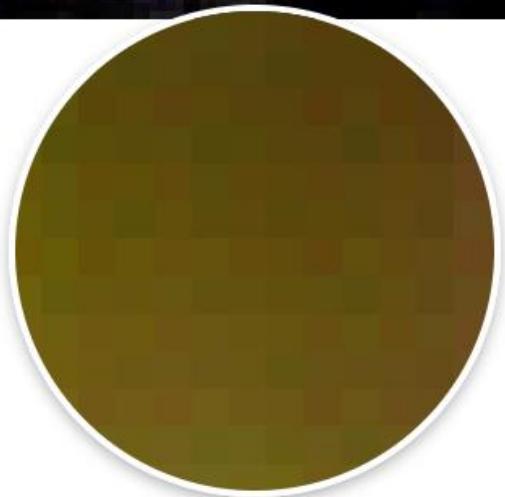
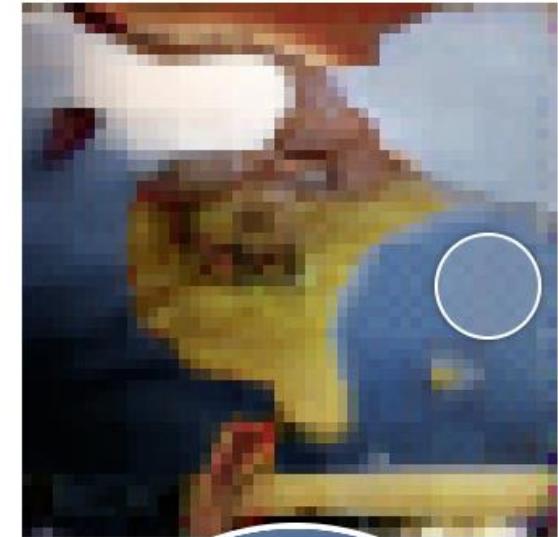
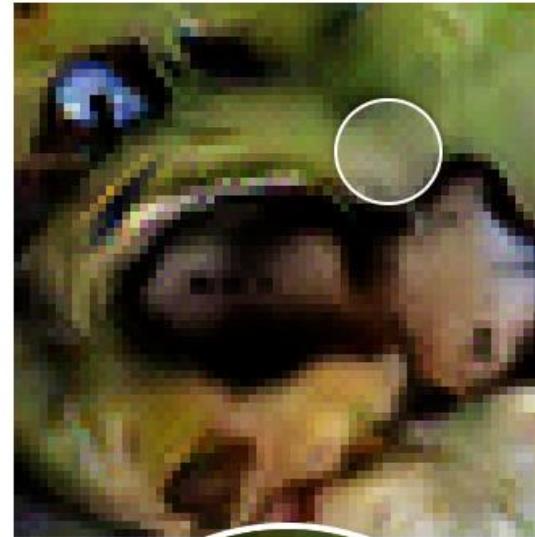
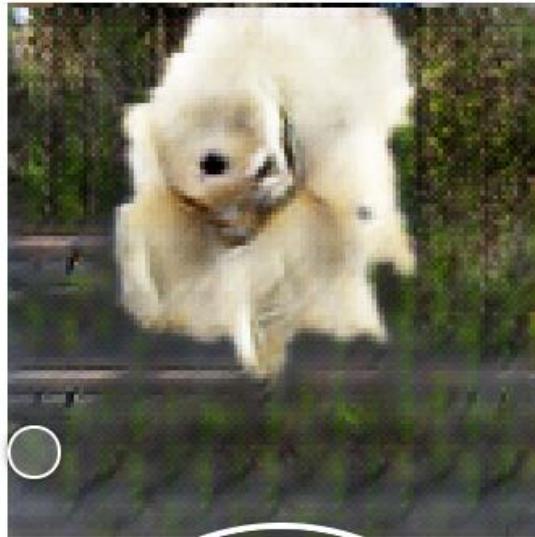
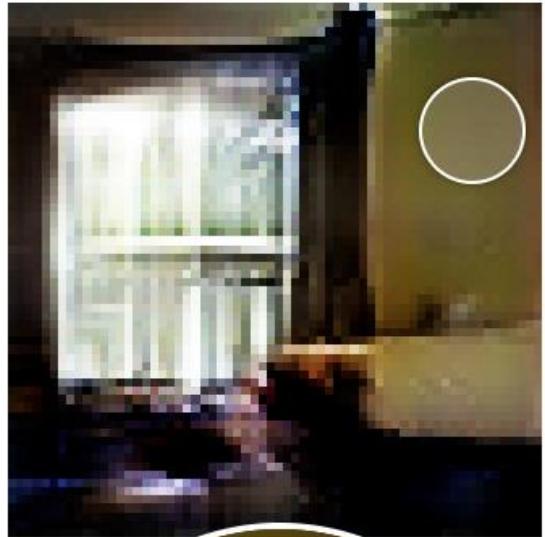
-2	3	-4	2
-1	4	3	0
-6	4	-2	1
6	-4	2	-1

výstup: 4x4

váhy konvolučního filtru

1	-1	2
1	-2	1
-1	2	-1

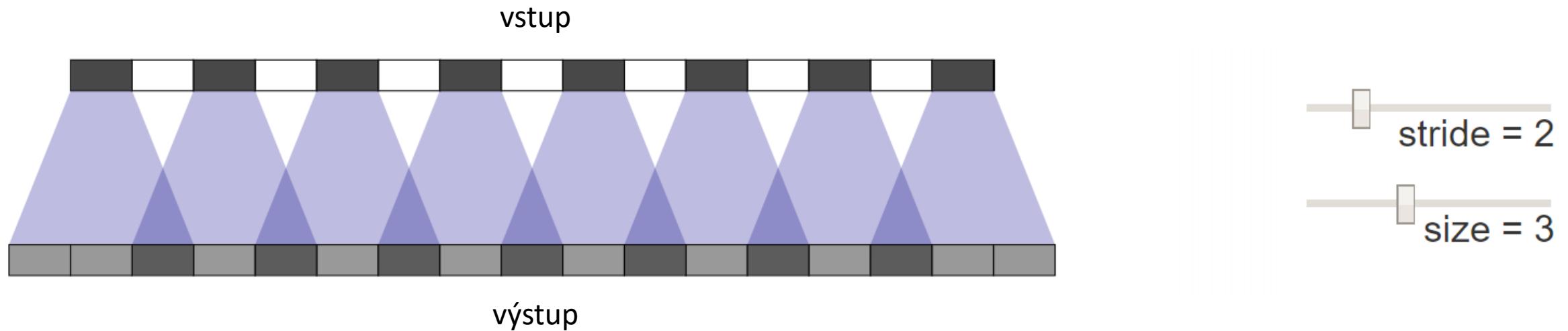
Checkerboard artifact



obrázek: <https://distill.pub/2016/deconv-checkerboard/>

Checkerboard artifact

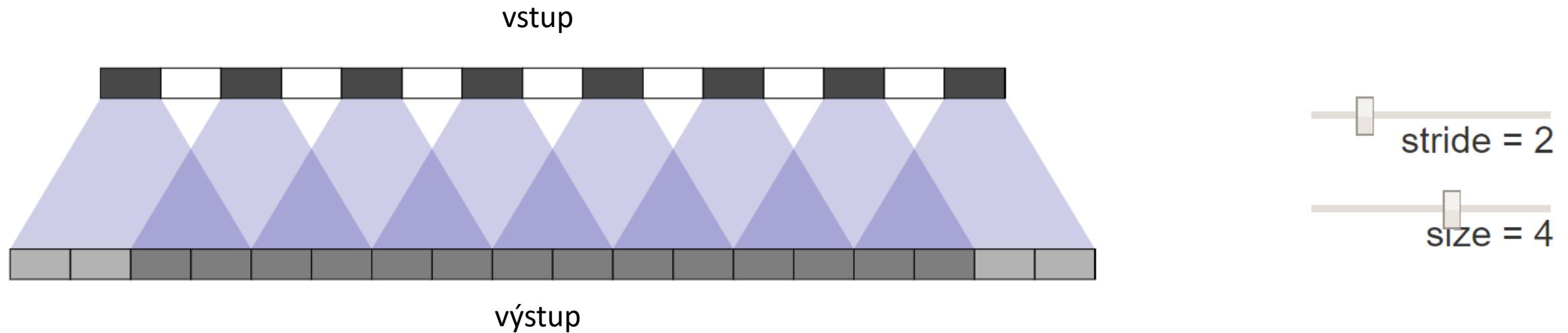
některé pixely výstupu sestávají z více součtů než ostatní → vzniká “šachovnicový” šum



obrázek: <https://distill.pub/2016/deconv-checkerboard/>

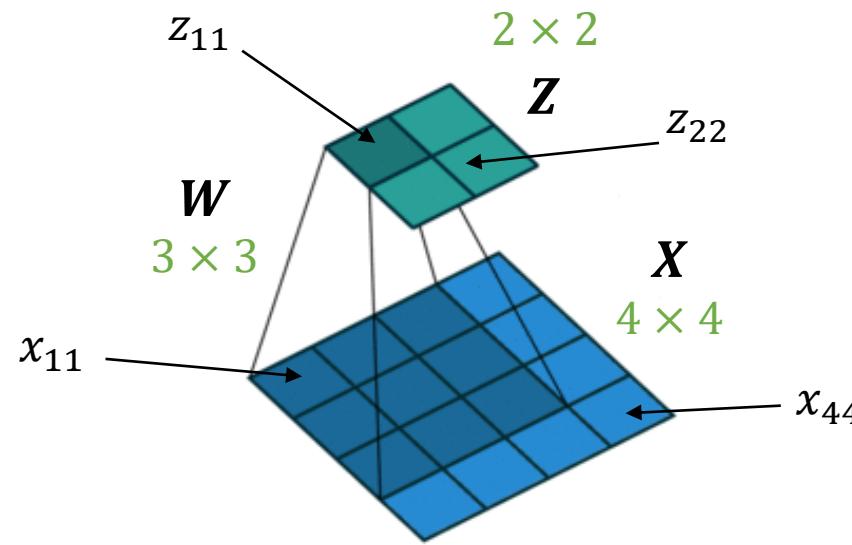
Checkerboard artifact

pokud vhodně zvolíme kombinaci velikosti filtru a kroku, tento artefakt odstraníme



obrázek: <https://distill.pub/2016/deconv-checkerboard/>

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

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

$4 \times 1 \rightarrow 2 \times 2$
 4×16
 $4 \times 4 \rightarrow 16 \times 1$

je gradient na vstup

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

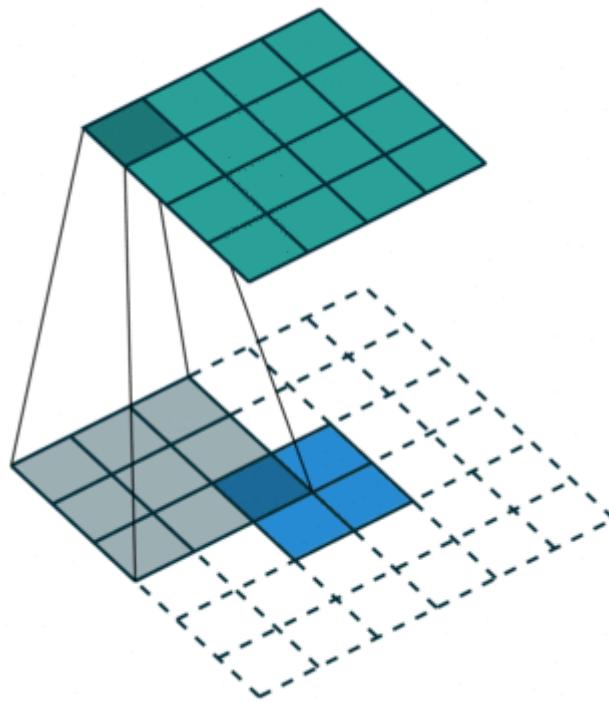
$16 \times 1 \rightarrow 4 \times 4$
 16×4
 $2 \times 2 \rightarrow 4 \times 1$

- 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 z na tvar vstupu x

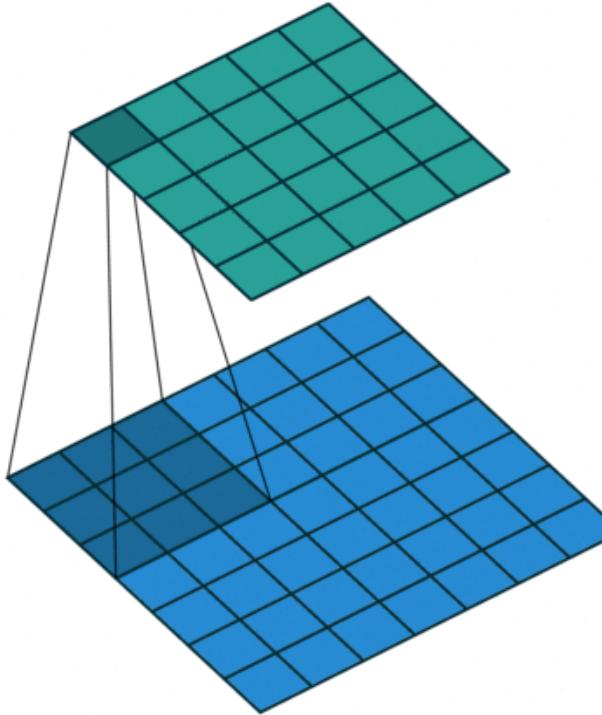
Jiná vizualizace transponované konvoluce

pozn.: vizualizace odpovídají transponované variantě standardní konvoluce s uvedenými parametry!

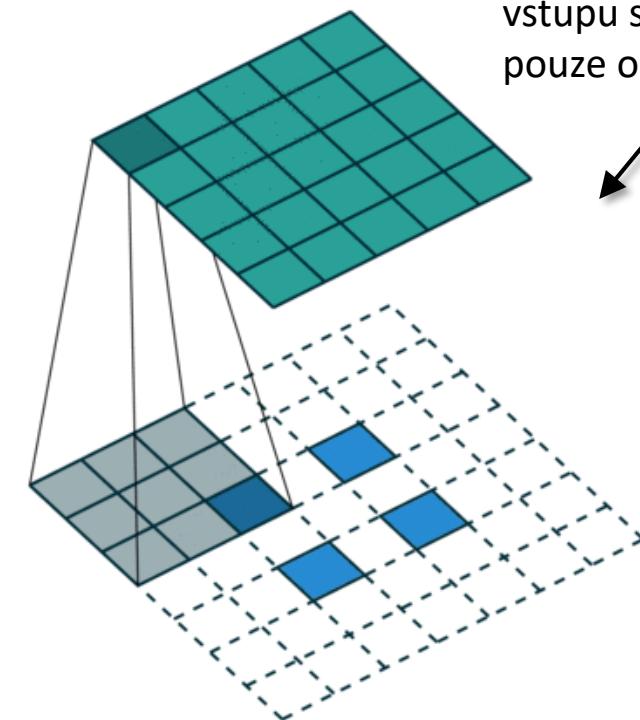
tj. kdybychom měli std. konvoluci s parametry uvedenými dole, tak její zpětný (transponovaný) průchod by vypadal právě takto



3x3, stride=1, padding=0



3x3, stride=1, padding=2



3x3, stride=2, padding=0
(někdy jako à trous/dilated convolution)

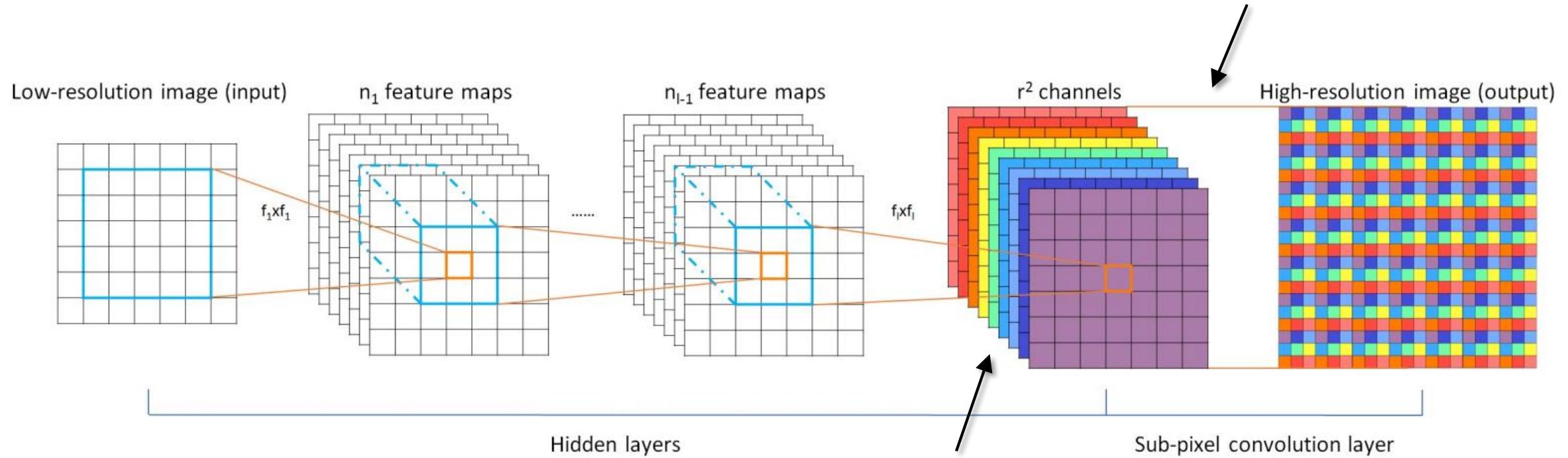
Další názvy transponované konvoluce

- Transposed convolution
 - zřejmě “nejsprávnější” název
 - někdy se však používá jen pro speciální konfigurace stride a paddingu
- Deconvolution
 - špatně, tento termín je již zaveden pro jiný typ operace
- Upconvolution
 - Příliš se neuchytilo
- Fractionally strided convolution
 - vychází z definice kroku (stride) jako **poměru** pohybu po vstupu ku výstupu
 - u zpětného průchodu tak může být < 1
- Upsampling
 - v některých frameworkech, v literatuře ne
- Dilated / à trous convolution
 - spec. případ transposed convolution se $\text{stride} > 1$

Sub-pixelová konvoluce

efektivnější než transponovaná konvoluce, přitom bez artefaktů!

poslední krok je vlastně jen reshape!



samostatný model pro každé požadované zvětšení r

vygenerujeme $r^2 C$ -kanálovou konvoluční mapu
 C je počet kanálů výstupu, r je faktor zvětšení

poslední reshape krok → v PyTorch třída [nn.PixelShuffle](#)

Super resolution



- poměrně jednoduché nasbírat trénovací data, stačí jen obrázky
- učíme sítě z malého obrázku predikovat velký
- kritérium bud' mean square error (suma čtverců)
- nebo např. perceptual loss – podobné jako content loss u style transfer = porovnávají se konvoluční mapy
- namísto RGB se typicky používá YCbCr a zvětšuje se pouze kanál Y

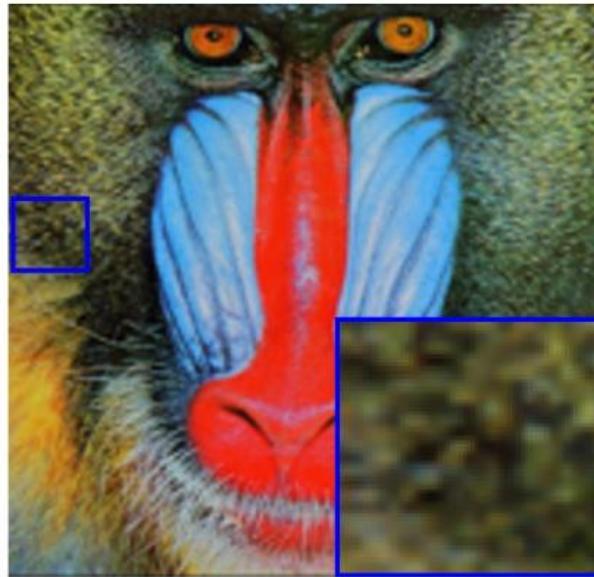
obrázek: <https://sites.google.com/a/udayton.edu/rhardie1/research/super-resolution>

Super resolution

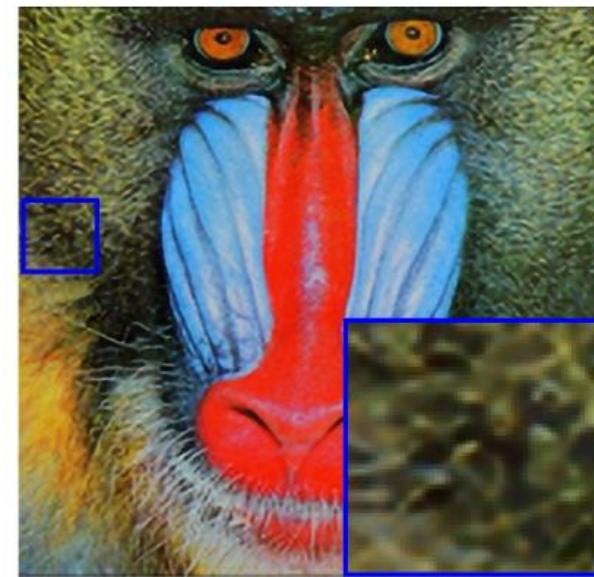
trojnásobné zvětšení



(a) Baboon Original



(b) Bicubic / 23.21db



(e) ESPCN / **23.72db**

Super resolution

trojnásobné zvětšení



(f) 335094 Original



(g) Bicubic / 22.24db



(j) ESPCN / 24.14db

Super resolution

trojnásobné zvětšení



(k) Monarch Original



(l) Bicubic / 29.43db



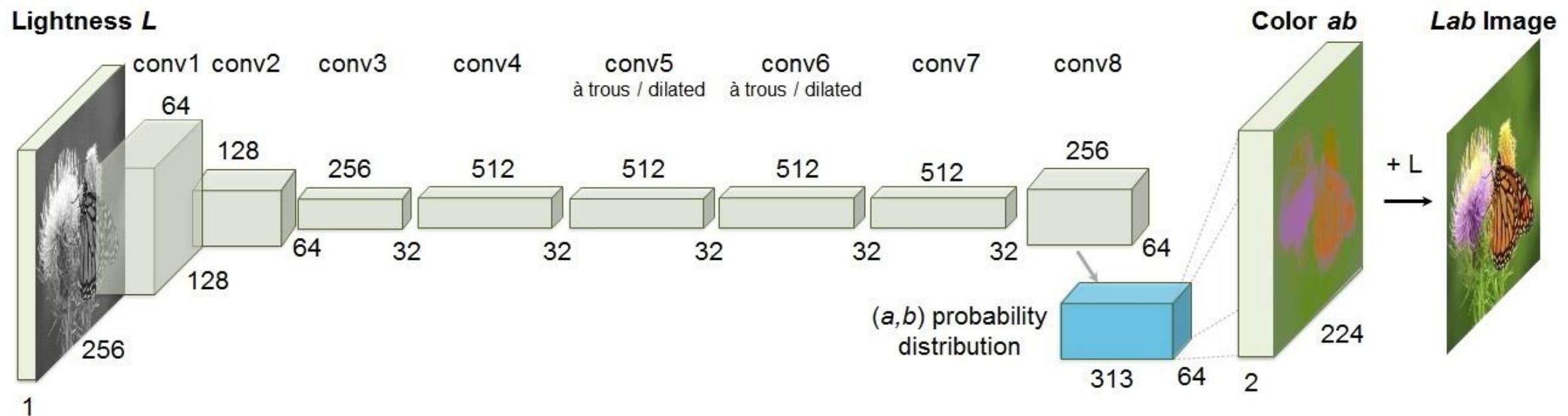
(o) ESPCN / **33.66db**

Automatické barvení obrázků (image colorization)

z jedné složky (L)

3

predikujeme zbylé dvě (ab)



opět ne RGB, ale jiný obarevný prostor: zde typicky Lab či LUV (jas → dvě barevné složky)

zdroj: Zhang et al.: Colorful Image Colorization

Automatické barvení obrázků (image colorization)



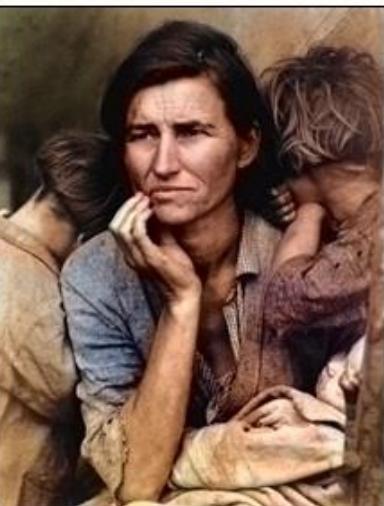
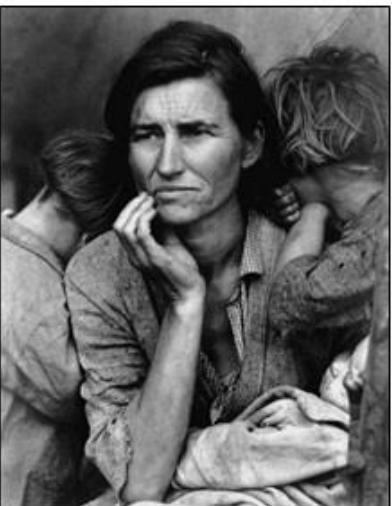
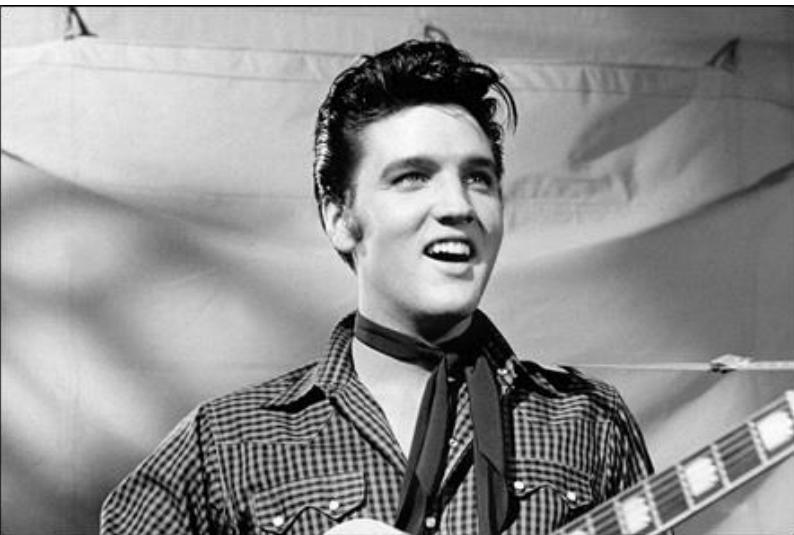
zdroj: [Zhang et al.: Colorful Image Colorization](#)

Automatické barvení obrázků (image colorization)



zdroj: [Zhang et al.: Colorful Image Colorization](#)

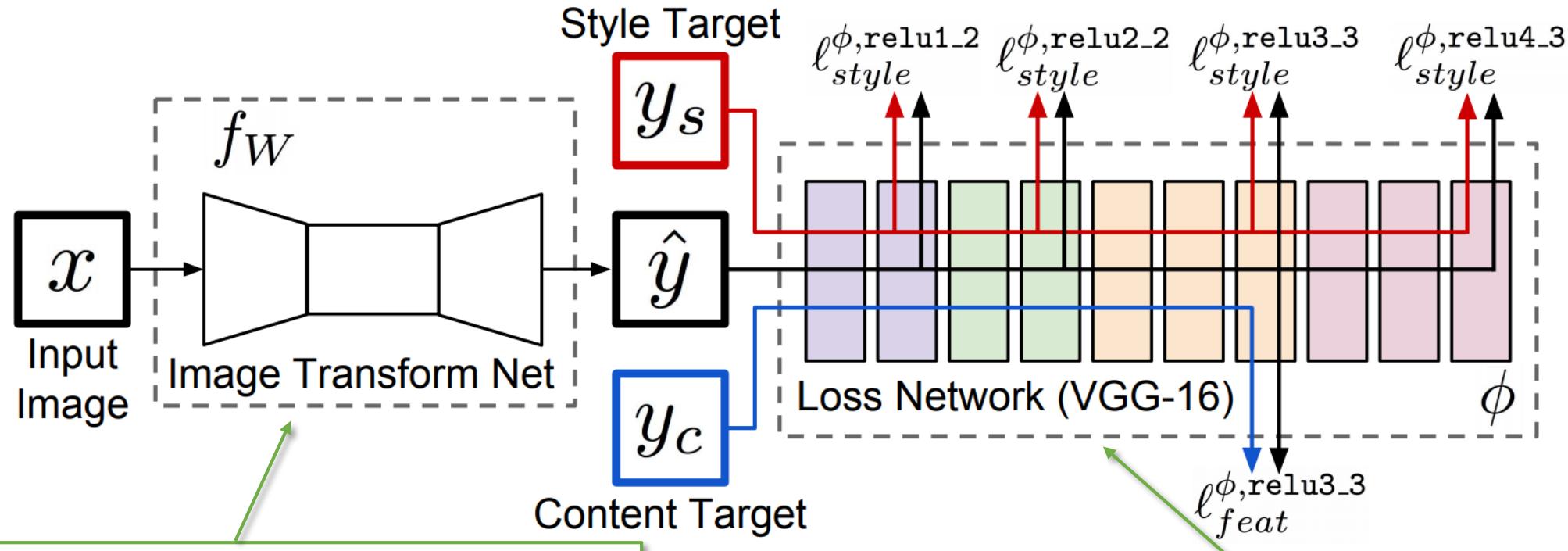
Automatické barvení obrázků (image colorization)



zdroj: [Zhang et al.: Colorful Image Colorization](#)

Fast style transfer

nahrazuje postupnou a pomalou optimalizaci vstupu učením sítě, která, jakmile naučená, stylizuje obrázek jediným průchodem



natrénovaná konvoluční síť, která ze vstupu
přímo predikuje stylizovaný obrázek

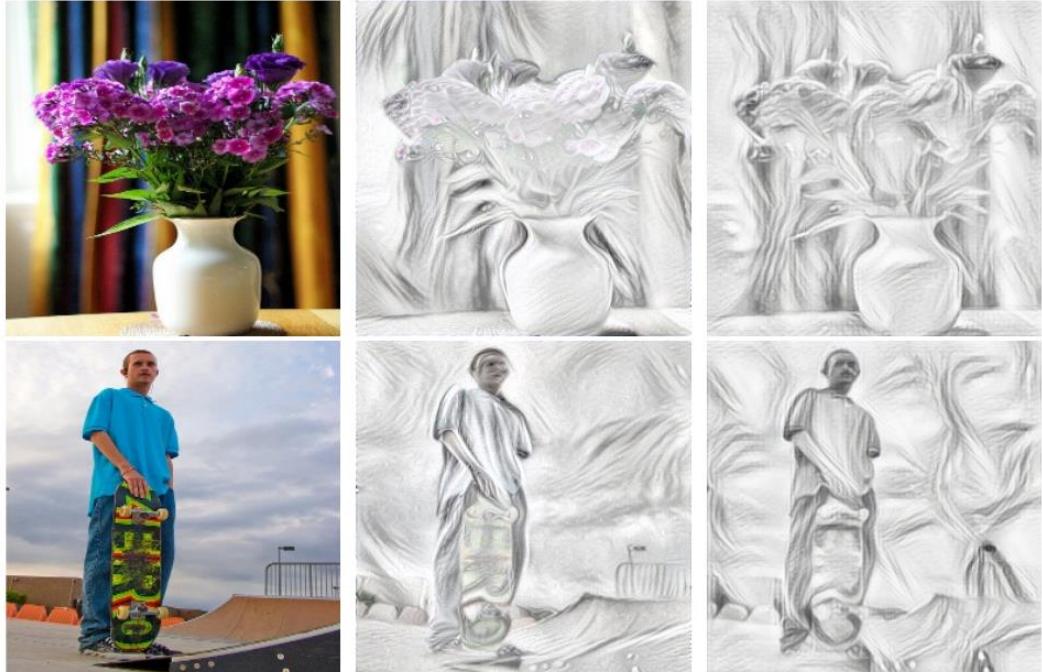
tato část zůstává stejná, tj. porovnávají se konvoluční
mapy (perceptual loss) a jejich style (gram matice)

je potřeba samostatnou síť pro každý styl; toto omezení však bylo odstraněno v dalších pracích

článek: [Johnson et al.: Perceptual Losses for Real-Time Style Transfer and Super-Resolution](#)

Fast style transfer

Style
Sketch



obsah (content)

původní transfer

fast transfer

Style
The Simpsons



obsah (content)

původní transfer

fast transfer

článek: [Johnson et al.: Perceptual Losses for Real-Time Style Transfer and Super-Resolution](#)

Fast style transfer → super resolution

stejný algoritmus (perceptual loss) autoři aplikovali i na problém super resolution!



Ground Truth	Bicubic	Ours (ℓ_{pixel})	SRCCN [11]	Ours (ℓ_{feat})
This image	31.78 / 0.8577	31.47 / 0.8573	32.99 / 0.8784	29.24 / 0.7841
Set5 mean	28.43 / 0.8114	28.40 / 0.8205	30.48 / 0.8628	27.09 / 0.7680

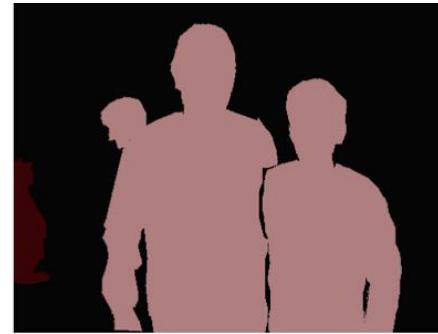
Fast style transfer → super resolution

stejný algoritmus (perceptual loss) autoři aplikovali i na problém super resolution!



článek: [Johnson et al.: Perceptual Losses for Real-Time Style Transfer and Super-Resolution](#)

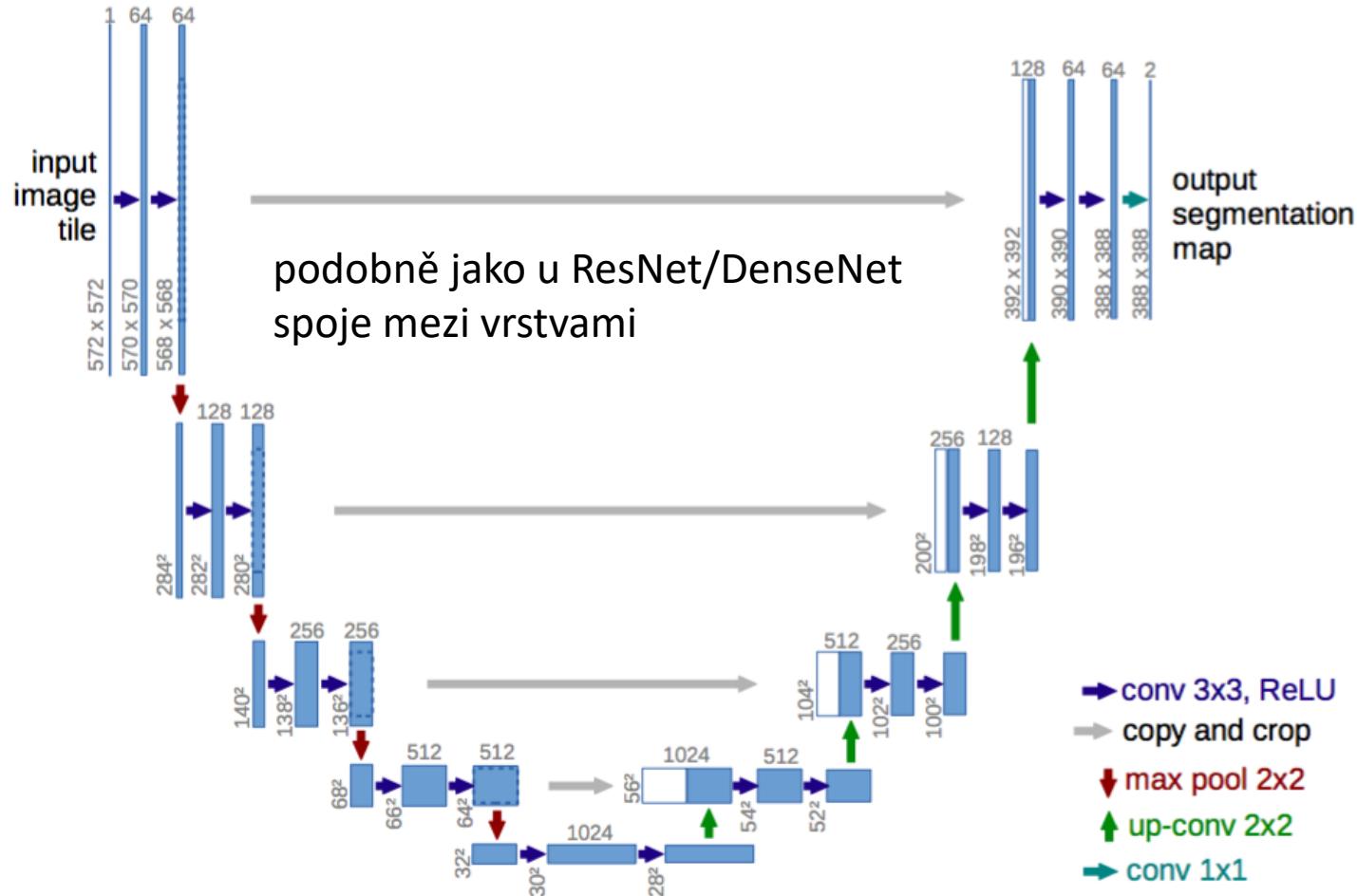
Sémantická segmentace



- lze zformulovat jako klasifikaci
- výstup ze sítě je predikce třídy pro každý pixel
- lze tedy použít standardní softmax, loss pro obrázek je pak suma přes všechny pixely
- drahá trénovací data – potřebujeme manuálně per-pixel segmentované obrázky!

Sémantická segmentace U-net

architektura typu enkodér - dekodér



článek: [Ronneberger et al.: U-Net: Convolutional Networks for Biomedical Image Segmentation](#)

Sémantická segmentace U-net

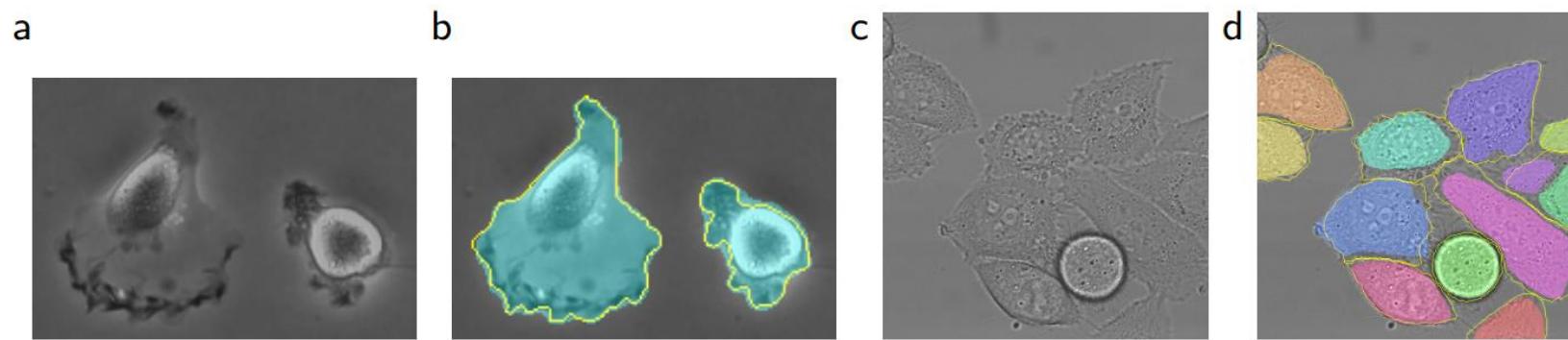


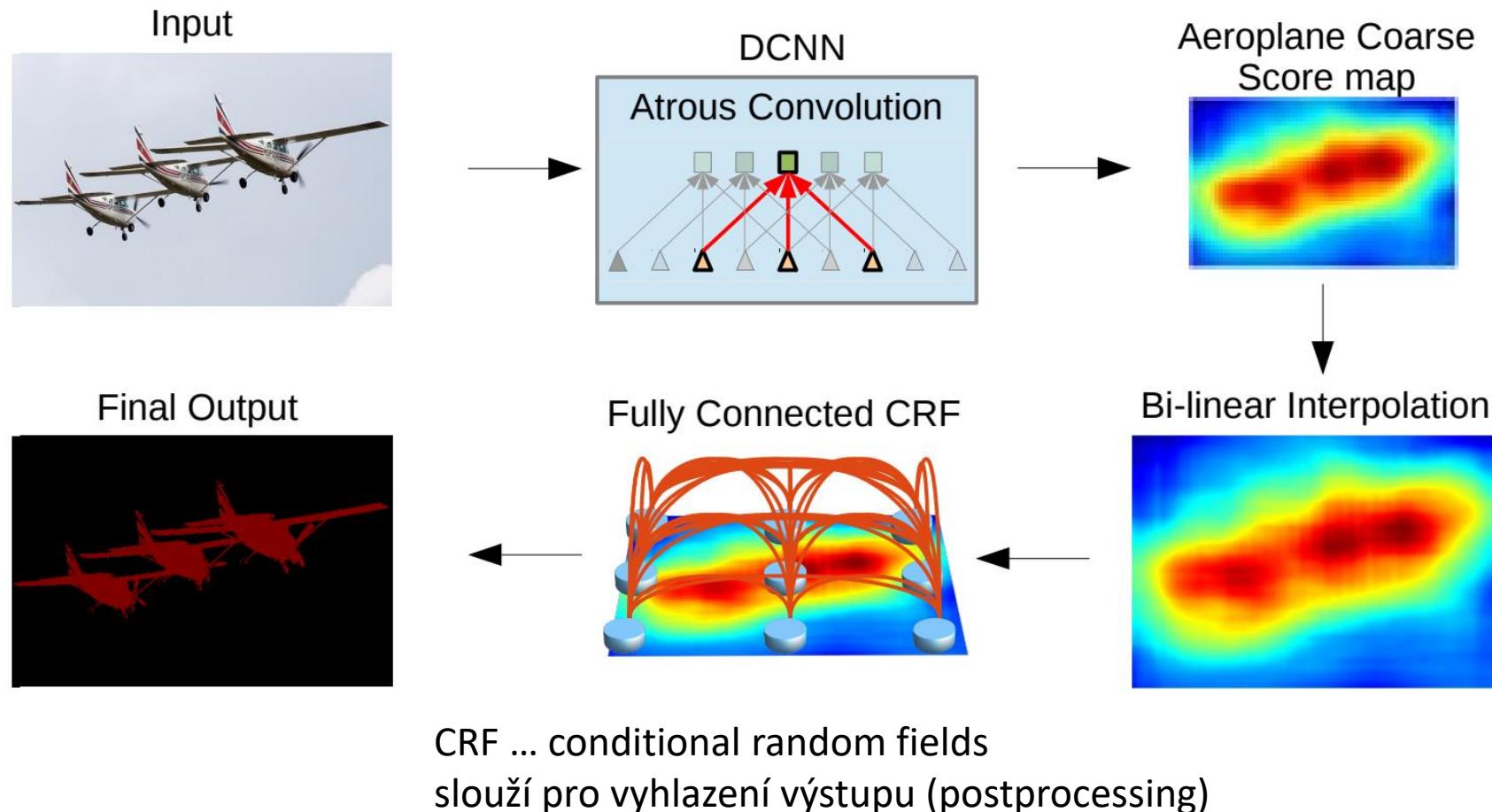
Fig. 4. Result on the ISBI cell tracking challenge. (a) part of an input image of the “PhC-U373” data set. (b) Segmentation result (cyan mask) with manual ground truth (yellow border) (c) input image of the “DIC-HeLa” data set. (d) Segmentation result (random colored masks) with manual ground truth (yellow border).

Table 2. Segmentation results (IOU) on the ISBI cell tracking challenge 2015

Name	PhC-U373	DIC-HeLa
IMCB-SG (2014)	0.2669	0.2935
KTH-SE (2014)	0.7953	0.4607
HOUS-US (2014)	0.5323	-
second-best 2015	0.83	0.46
u-net (2015)	0.9203	0.7756

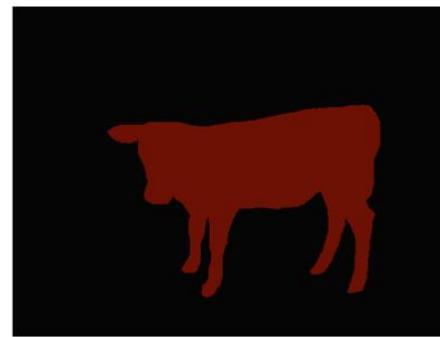
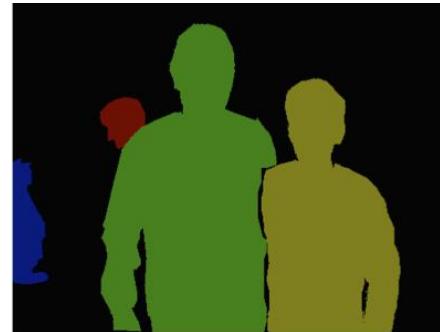
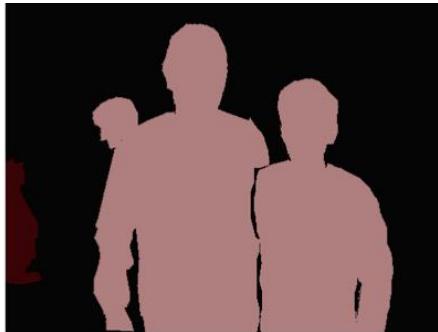
Sémantická segmentace skrze dilated konvoluce

- používá VGG, ze které autoři odstranili všechny max poolingy a nahradili konvolucemi se stride > 1
- upsampling potom opět pomocí dilated konvolucí = transponovaná konvolece se stride > 1



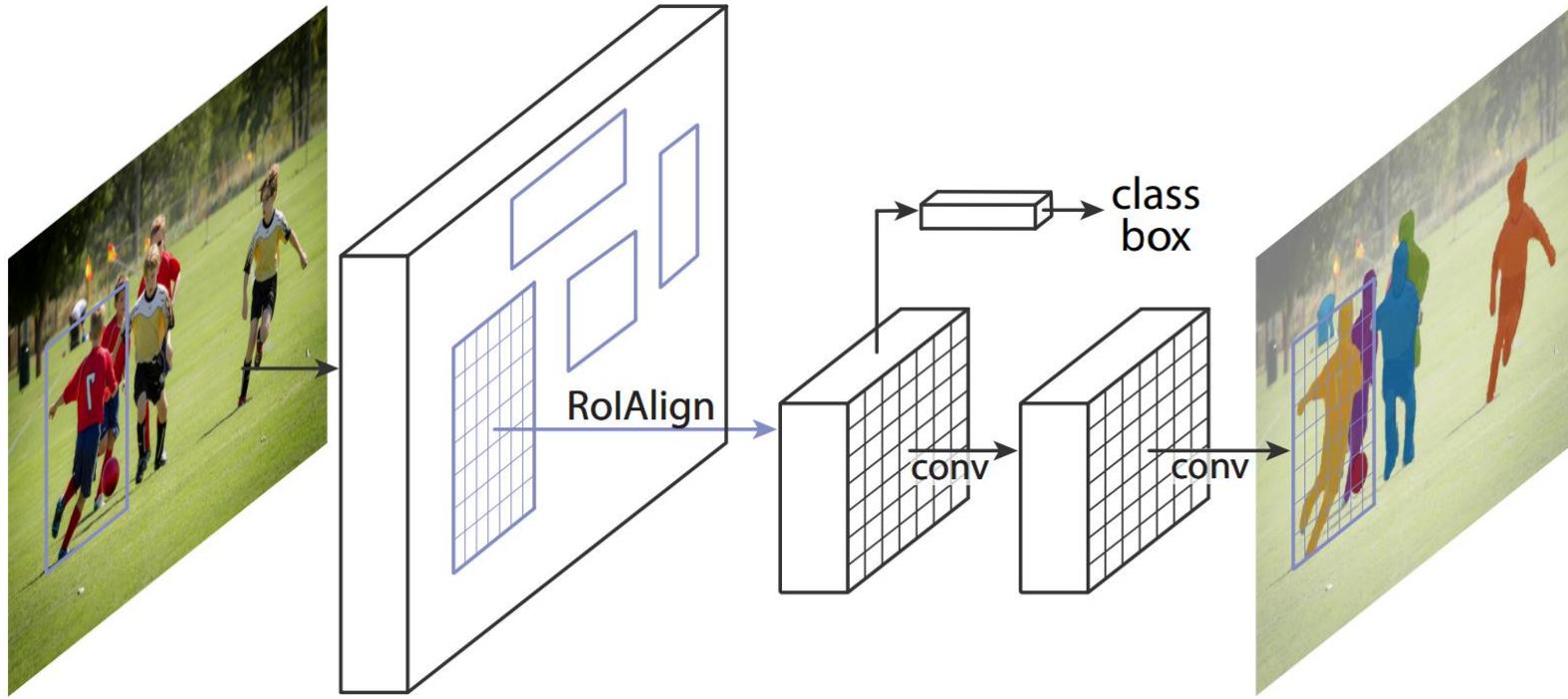
článek: [Chen et al: DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs](#)

Segmentace objektů (instance segmentation)



- na rozdíl od sémantické segmentace rozlišujeme mezi jednotlivými instancemi tříd (objekty)!

Mask R-CNN



- kombinace Faster R-CNN pro detekci objektů a segmentace
- pro každý výstupní region je zároveň predikovaná i maska → segmentace objektů, nikoliv tříd
- segmentace per-pixelově predikuje objekt vs neobjekt
- oproti Faster R-CNN tedy obsahuje jednu segmentační větev navíc

článek: [He et al.: Mask R-CNN](#)

Mask R-CNN

	backbone	AP ^{bb}	AP ^{bb} ₅₀	AP ^{bb} ₇₅	AP ^{bb} _S	AP ^{bb} _M	AP ^{bb} _L
Faster R-CNN+++ [19]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [27]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [21]	Inception-ResNet-v2 [41]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [39]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
Faster R-CNN, RoIAlign	ResNet-101-FPN	37.3	59.6	40.3	19.8	40.2	48.8
Mask R-CNN	ResNet-101-FPN	38.2	60.3	41.7	20.1	41.1	50.2
Mask R-CNN	ResNeXt-101-FPN	39.8	62.3	43.4	22.1	43.2	51.2

Table 3. **Object detection single-model** results (bounding box AP), vs. state-of-the-art on test-dev. Mask R-CNN using ResNet-101-FPN outperforms the base variants of all previous state-of-the-art models (the mask output is ignored in these experiments). The gains of Mask R-CNN over [27] come from using RoIAlign (+1.1 AP^{bb}), multitask training (+0.9 AP^{bb}), and ResNeXt-101 (+1.6 AP^{bb}).

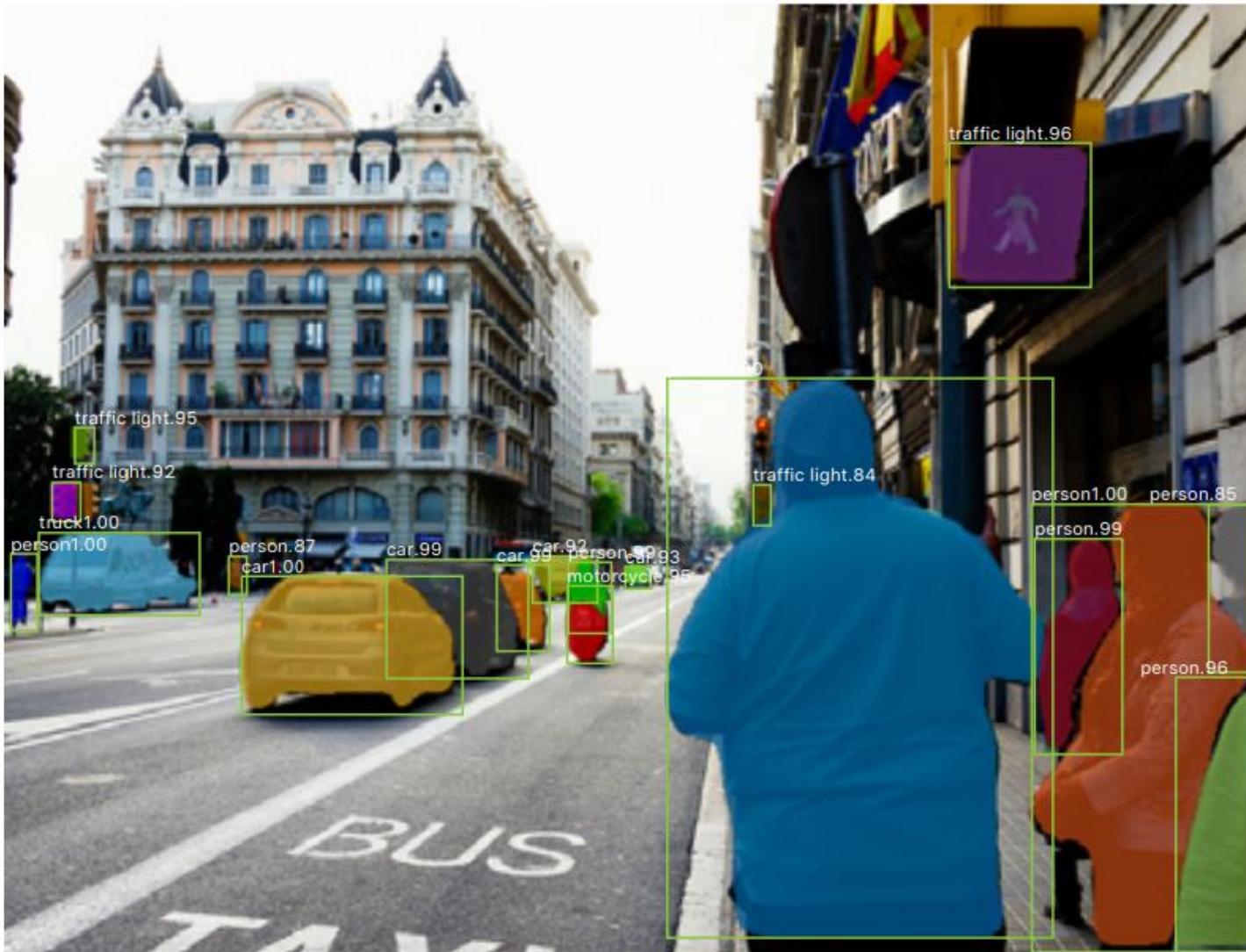
- dosahuje state-of-the-art i pro obyčejnou detekci objektů (bounding boxy!)
- cca 200 ms (5 FPS) na obrázek na NVidia Tesla M40 GPU
- cca 400 ms (2.5 FPS) na obrázek, pokud region proposal net (RPN) a segmentační části sítě nesdílí váhy

Mask R-CNN



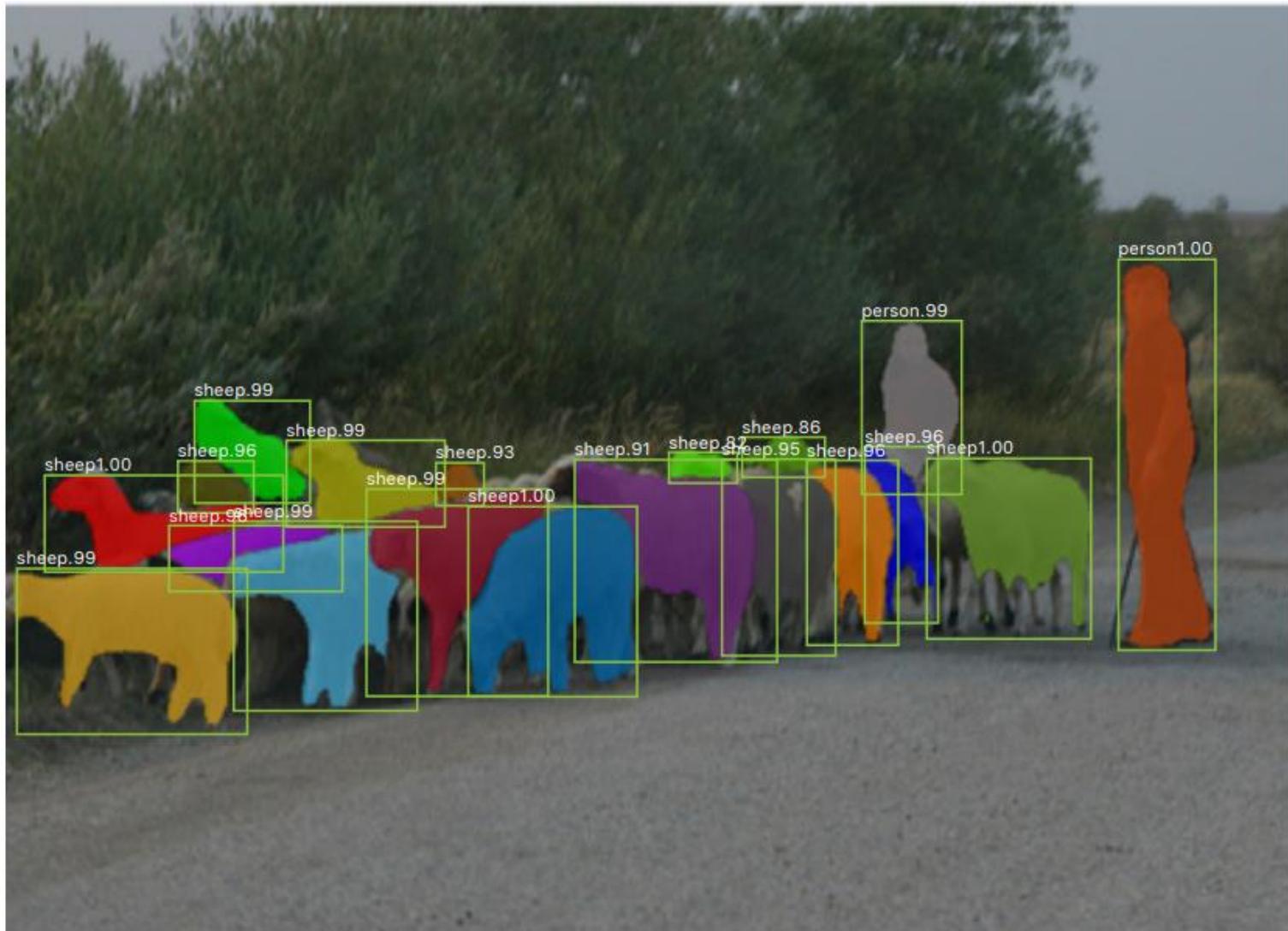
článek: [He et al.: Mask R-CNN](#)

Mask R-CNN



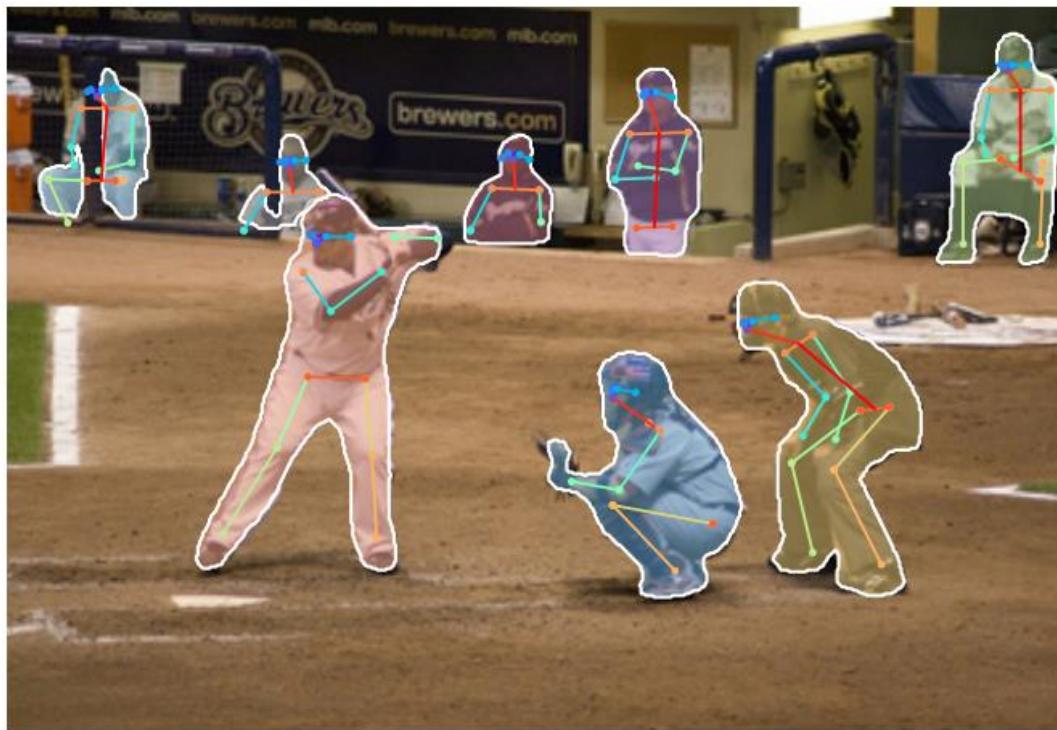
článek: [He et al.: Mask R-CNN](#)

Mask R-CNN



článek: [He et al.: Mask R-CNN](#)

Mask R-CNN pro pose estimation



- autoři rozšířili i pro pose estimation
- pro každý kloub (keypoint) síť predikuje one-hot masku = pouze jediný pixel je “1”, ostatní “0”

Učení bez učitele a generativní modely

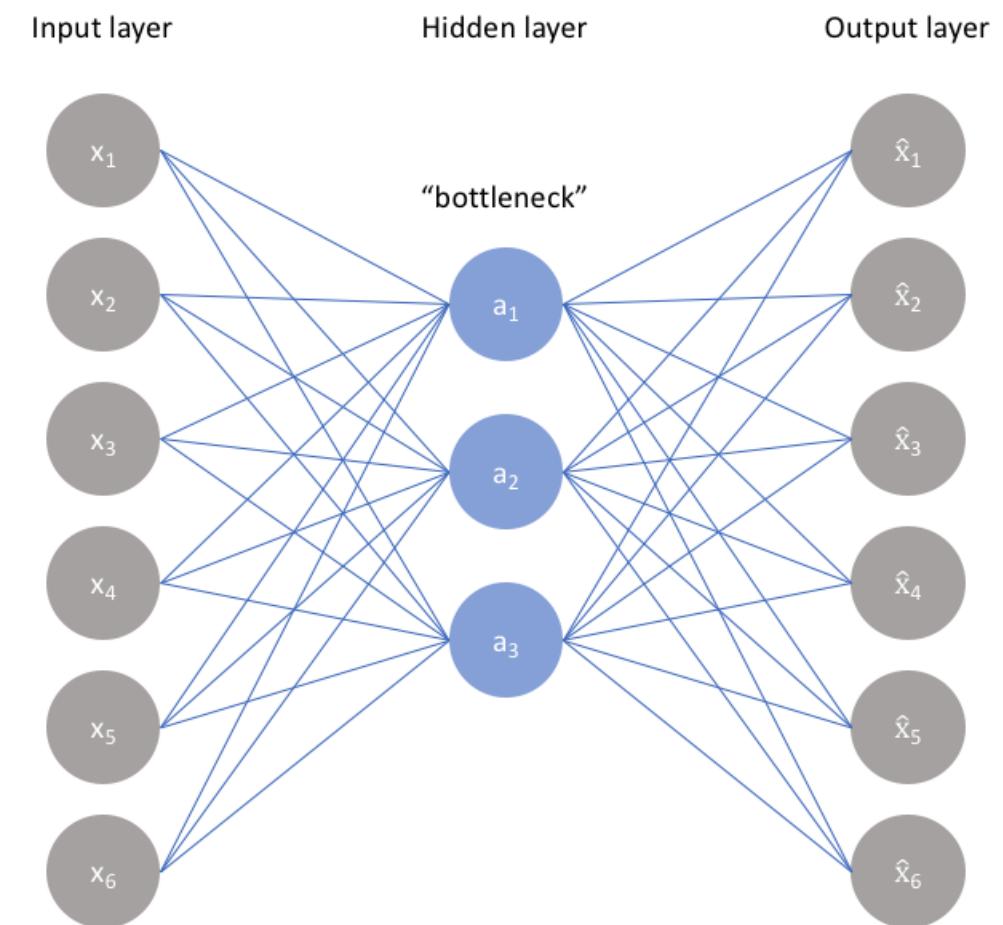
Autoenkodér

- úkolem rekonstrukce vstupních dat
- architektura typu encoder - dekodér
- vstupní vektor/obrázek je redukován enkodérem na nějaký *bottleneck*
- poté znova rekonstruován dekodérem
- lze nahlížet jako na kompresi
- cílem co nejmenší rekonstrukční odchylka, např. mean square error (MSE)

$$L(x, x') = \frac{1}{D} \|x - x'\|^2$$

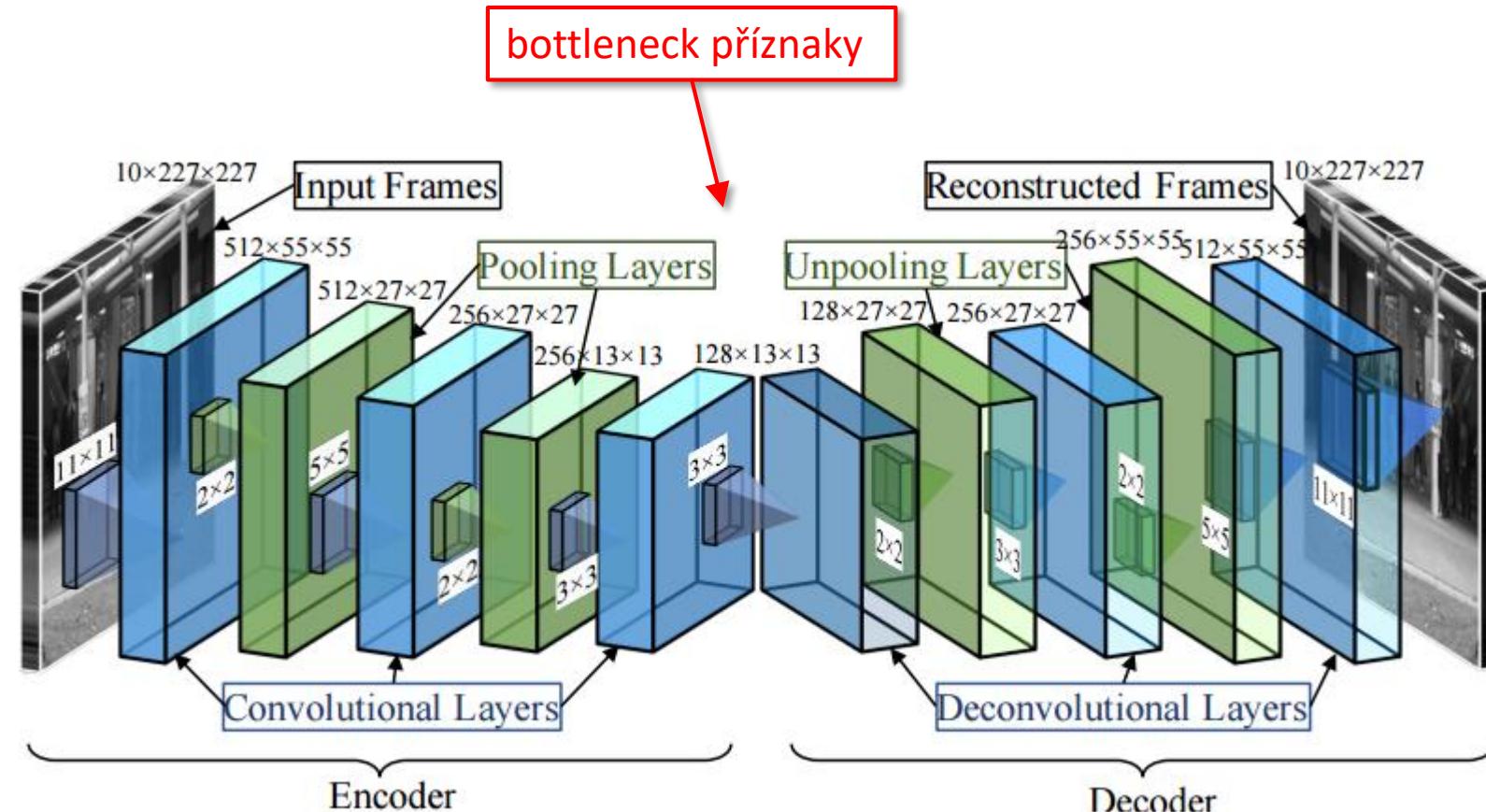
kde D je délka vektoru x

- Spec. případ s jednou skrytou vrstvou a bez nelinearity
≈ PCA
 - nejsou to samé, ale výsledné bottleneck vektory pokrývají stejný prostor
 - PCA má navíc omezení jejich vzájemné ortogonality



obrázek: <https://www.jeremyjordan.me/autoencoders/>

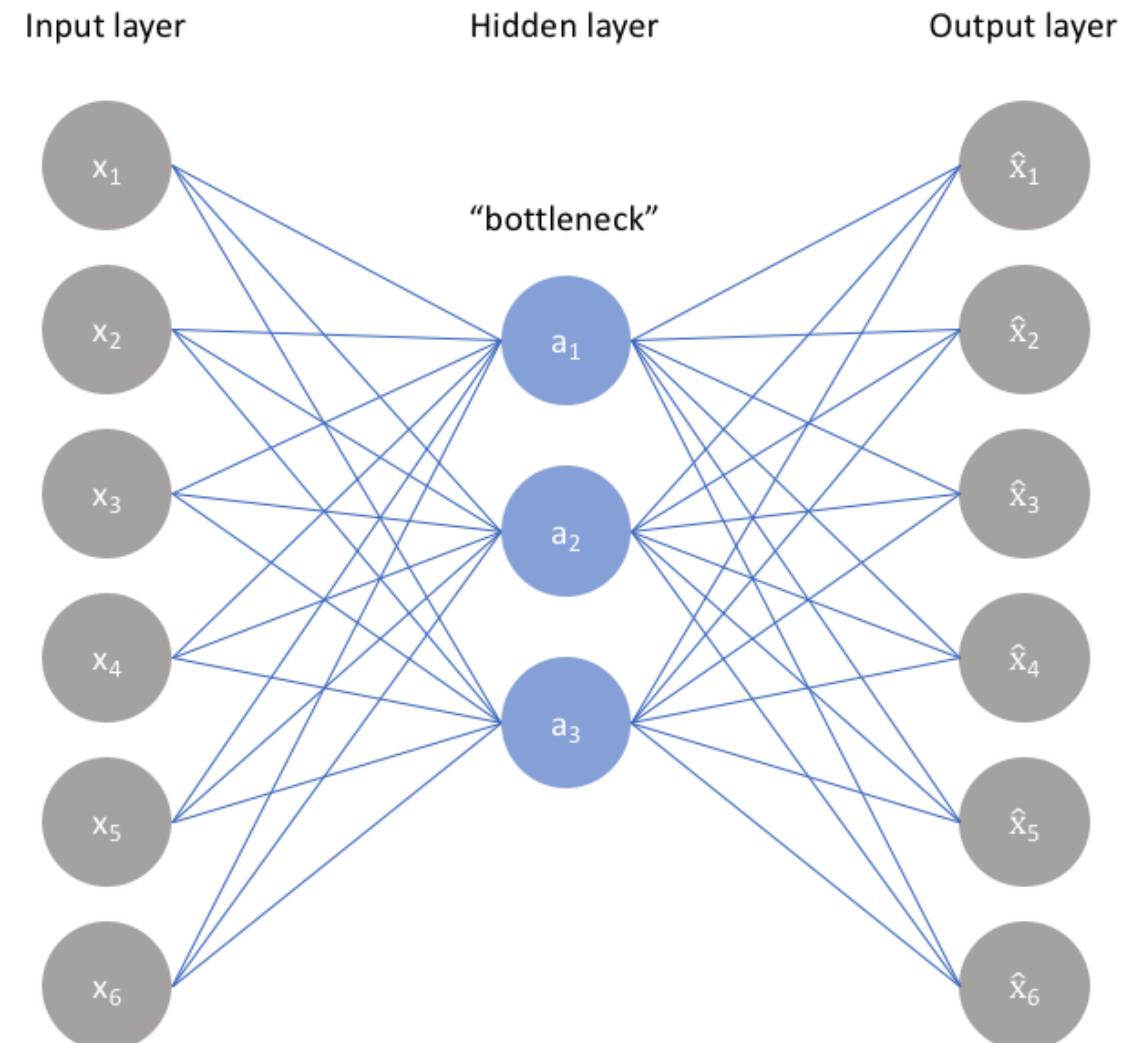
Konvoluční autoenkodér



- nahrazuje fully connected vrstvy konvolučními

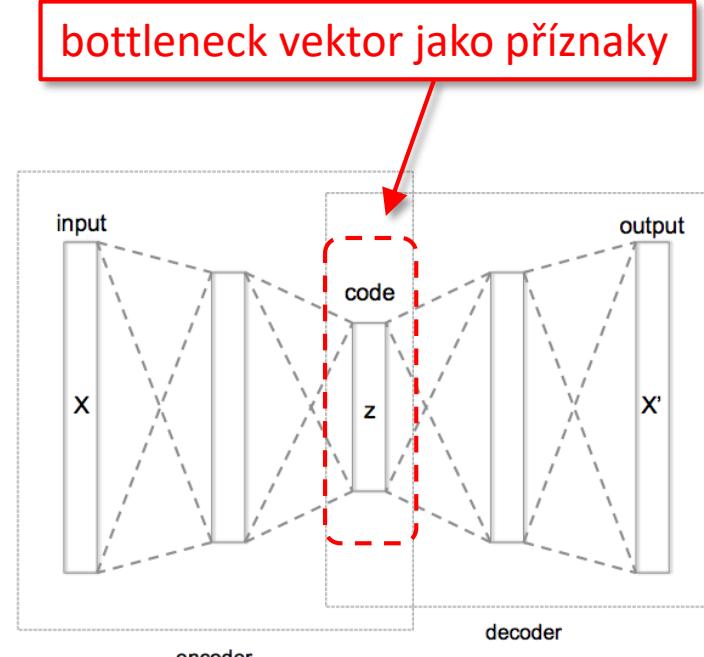
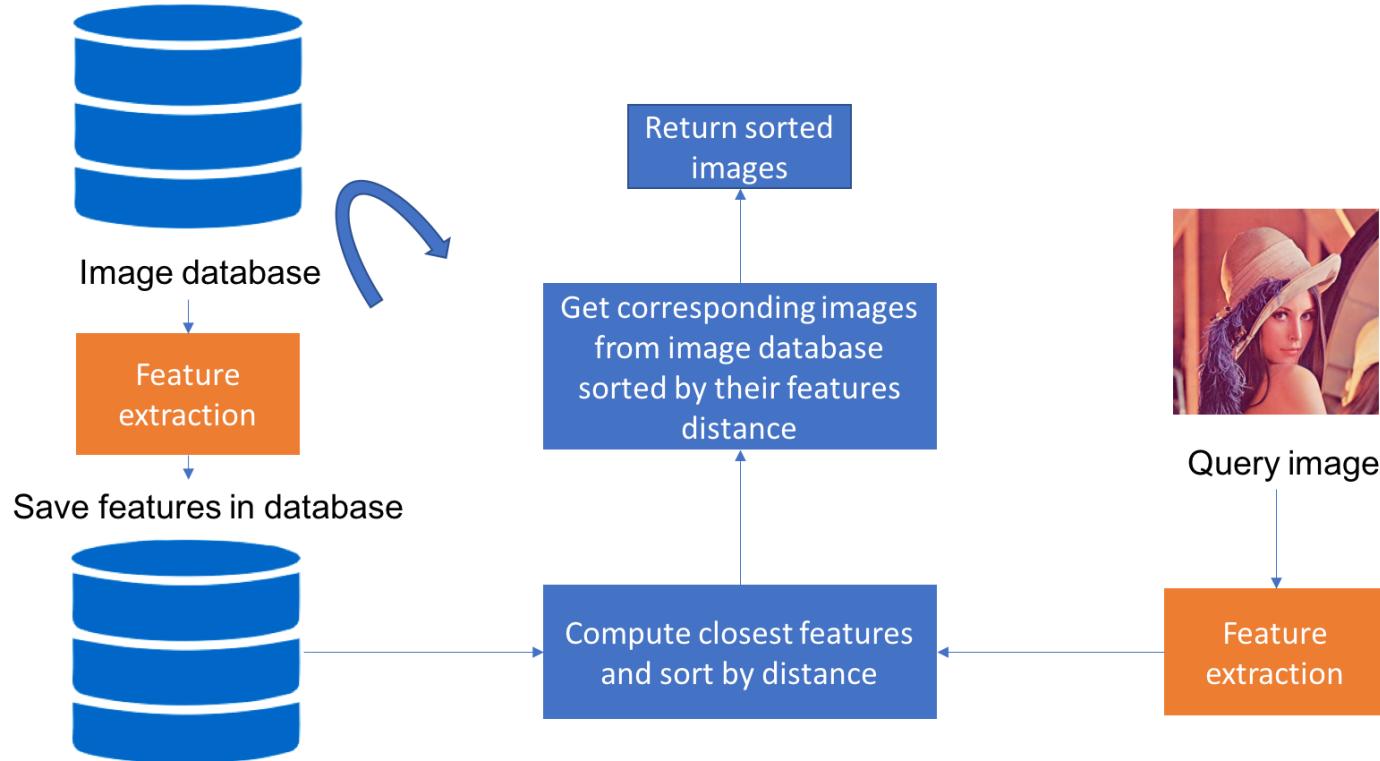
Autoenkodér pro detekci anomálií

- myšlenka: natrénujeme na “čistých” datech
- v testovací fázi:
 - známá data, která odpovídají trénovacím → malá rekonstrukční odchylka
 - neznámá data (anomálie) → velká odchylka
- V literature jako anomalies či outlier detection
- lze využít např. pro detekci nečistot v materiálu apod.



Content-based image retrieval

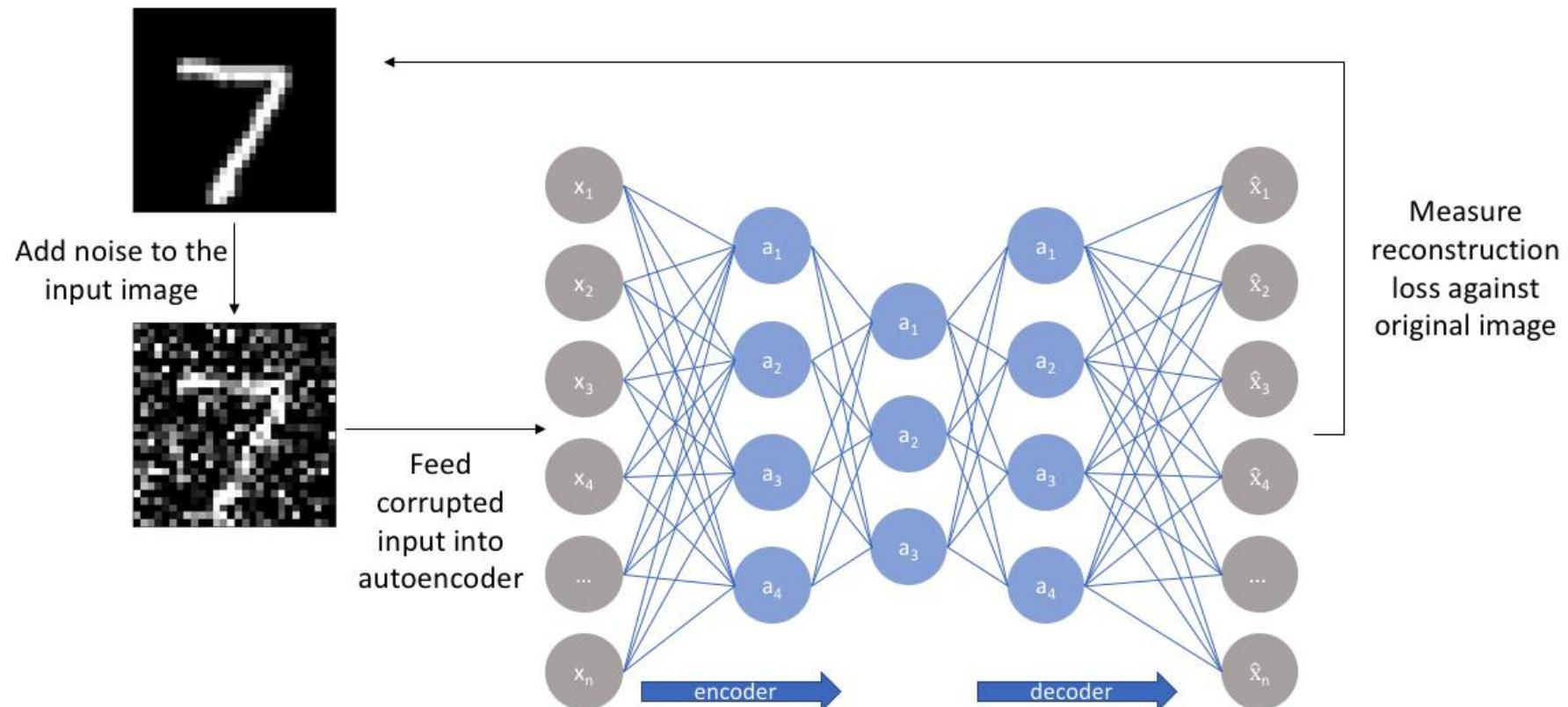
- myšlenka: autoenkovdér natrénovaný na velkém množství obrázků použít pro extrakci příznaků



- nejpodobnější obrázek vyhledáme např. metodou nejbližšího soused nad extrahovanými příznaky

obrázek: <https://blog.sicara.com/keras-tutorial-content-based-image-retrieval-convolutional-denoising-autoencoder-dc91450cc511>

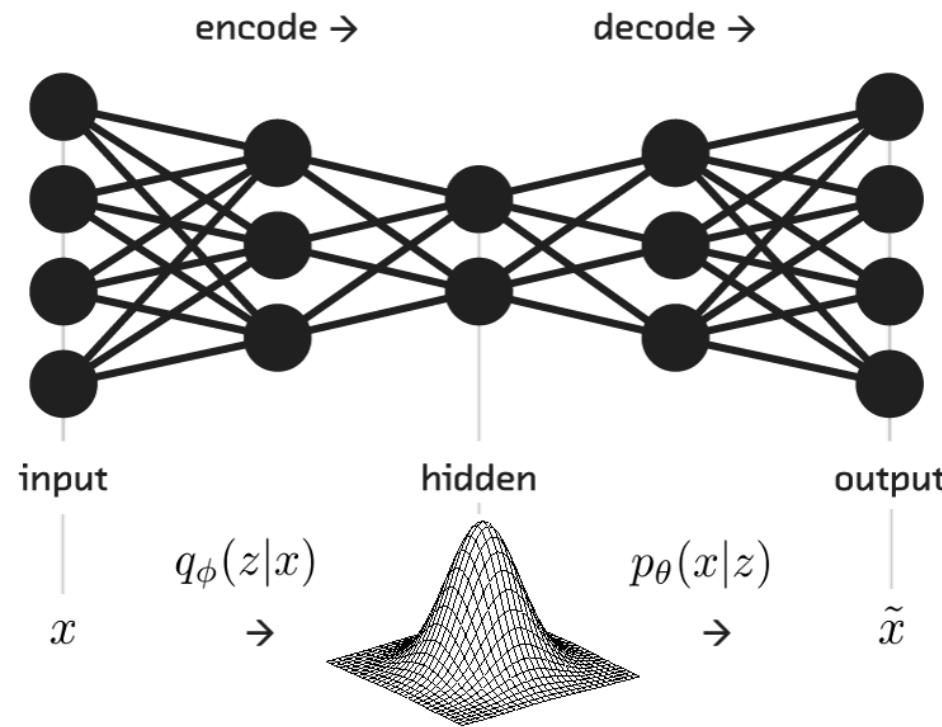
Denoising autoencoder



- vstup je zašuměný → síť rekonstruuje čistý obrázek
- natrénovaný autoenkovdér odstraní šum
- lze využít i pro inpainting apod.
- princip formulace podobný jako u colorization, super resolution, ...

obrázek: <https://www.jeremyjordan.me/autoencoders/>

Variační autoenkodér



celkový loss

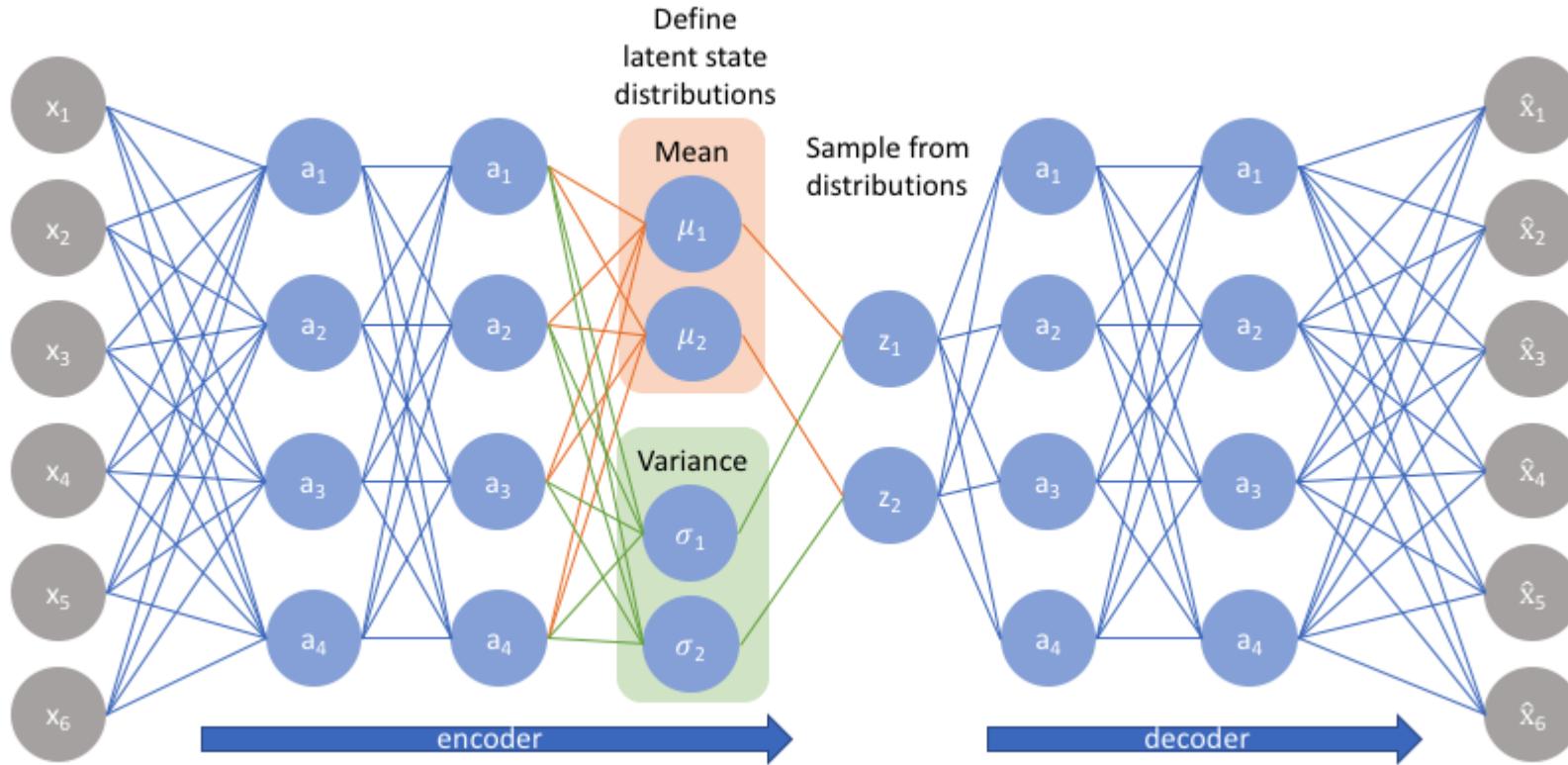
$$\mathcal{L}(x, \hat{x}) + \sum_j KL(q_j(z|x) || p(z))$$

Kullback-Liebler divergence

- omezuje bottleneck příznaky na gaussovské (normální) rozložení
- kromě rekonstrukční odchylky navíc Kullback-Liebler divergence jako kritérium na bottleneck

$$D_{\text{KL}}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \quad \dots \text{měří podobnost, na kolik je bottleneck } (Q) \text{ gaussovský } (P)$$

Implementace variačního autoenkodéru

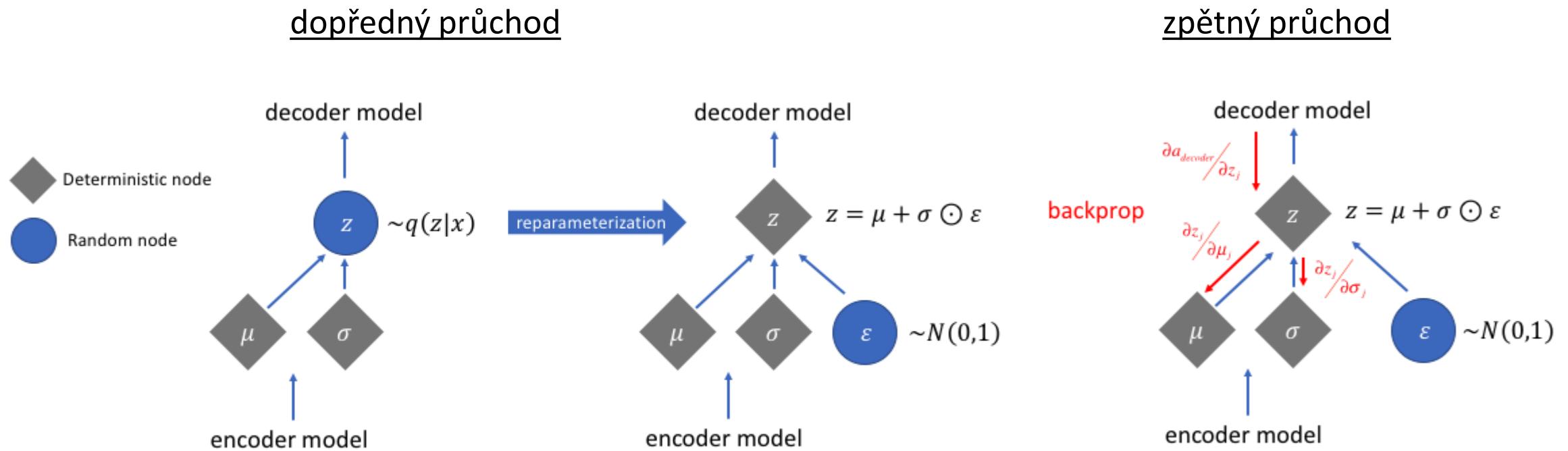


- KL nelze v praxi efektivně vyhodnotit, je to střední hodnota (integrál) přes všechny možné hodnoty
- namísto síť vygeneruje dva vektory průměru μ a rozptylu σ^2 gaussovského rozložení
- pak se využije následujícího:

$$-\mathcal{D}_{KL}(q_\phi(z|x)||p_\theta(z)) = \frac{1}{2} \sum (1 + \log(\sigma^2) - \mu^2 - \sigma^2)$$

obrázek: <https://www.jeremyjordan.me/variational-autoencoders/>

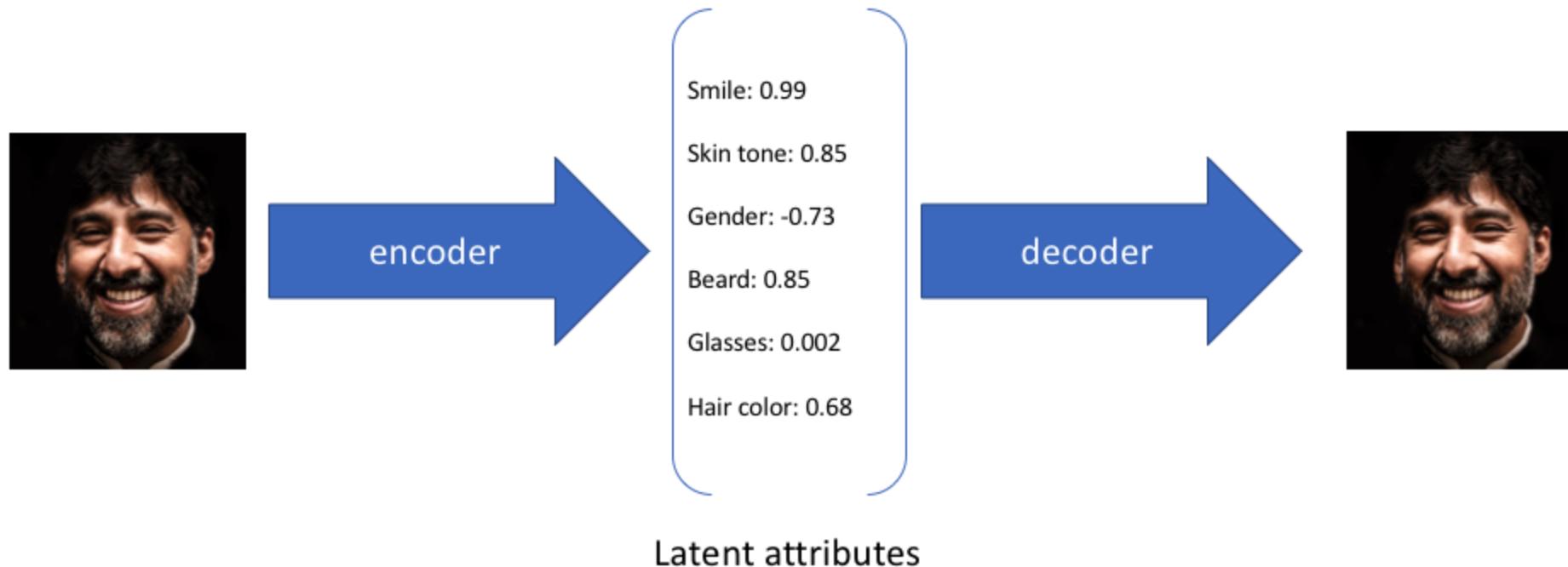
Reparametizační trik



- vzorkuje se vždy ze standardního $N(0, 1)$ gaussovského rozložení s nulovým průměrem a jednotkovým rozptylem
- průměr a rozptyl se řeší affinní transformací (škálování = std. odchylka, posun = průměr) vzorkovaných dat

obrázek: <https://www.jeremyjordan.me/variational-autoencoders/>

Interpretace latentních “bottleneck” parametrů

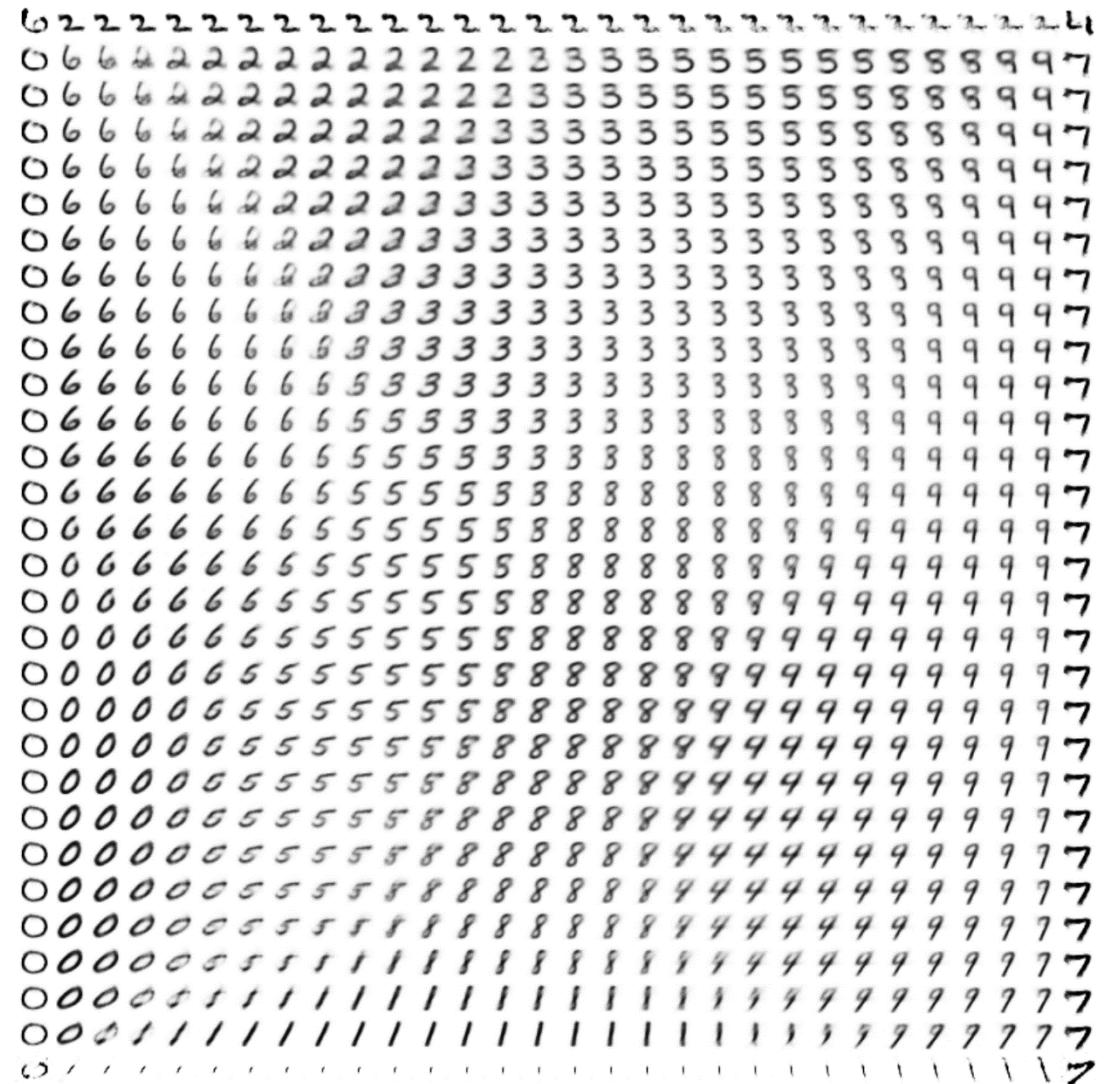


- na bottleneck vektor lze nahlížet jako na skryté parametry
- samplováním parametrů z normálního rozložení lze inicializovat dekodér a generovat různé obrázky
- variační autoenkovdér tedy patří mezi generativní modely

obrázek: <https://www.jeremyjordan.me/variational-autoencoders/>

Skryté parametry variačního autoenkodéru

- na obrázku příklad dvourozměrného normálního rozložení (dva skryté parametry)
- pohybem v tomto 2D prostoru generujeme různé MNIST číslice



obrázek: <http://blog.fastforwardlabs.com/2016/08/22/under-the-hood-of-the-variational-autoencoder-in.html>

Skryté parametry variačního autoenkodéru



Obrázky vygenerované variačním autoenkodérem



Labeled faces in the wild

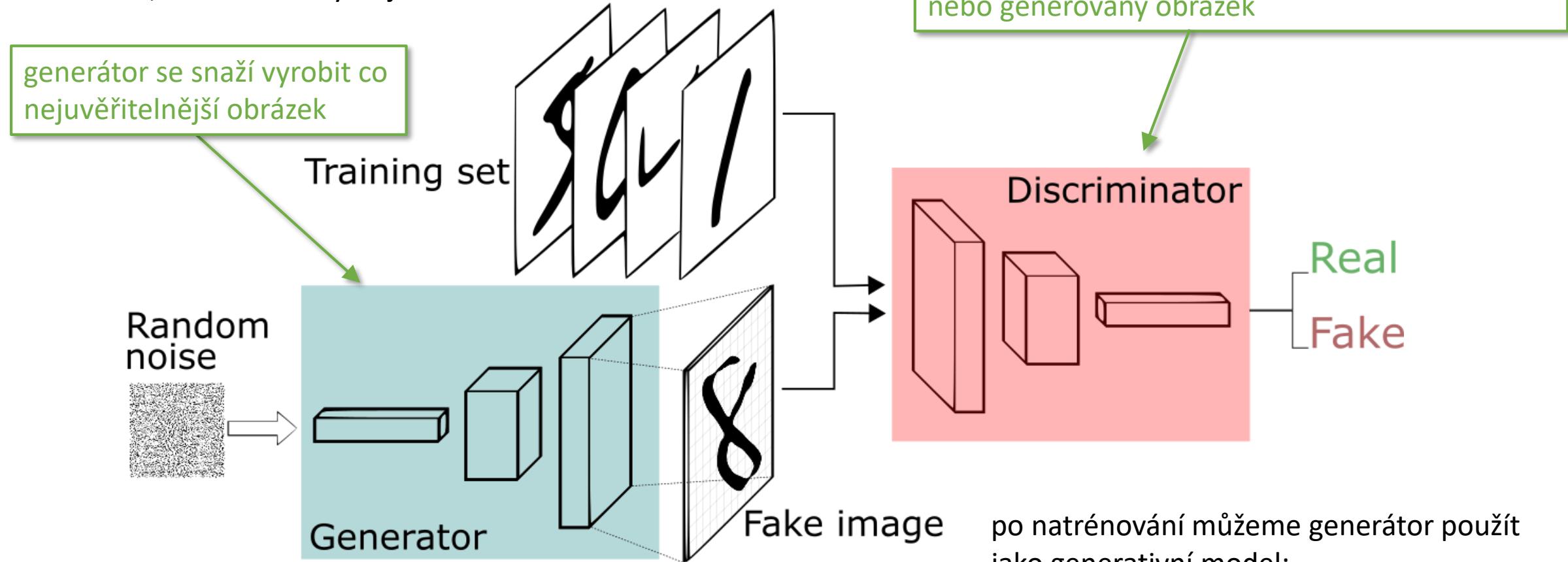


CIFAR-10

obrázek: <https://jaan.io/what-is-variational-autoencoder-vae-tutorial/>

Generative adversarial network (GAN)

- [Goodfellow et al.: Generative Adversarial Networks](#)
- učení dvou sítí zároveň: generátor vs diskriminátor
- "soutěž", která z nich vyhraje



obrázek: <https://deeplearning4j.org/generative-adversarial-network>

po natrénování můžeme generátor použít jako generativní model:
random inicializace → generovaný obrázek

Kritérium generativní adversarial sítě

celkový loss je

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))]$$

výstup diskriminátoru

výstup diskriminátoru

maximalizujeme přes diskriminátor tak, aby $D(x)$ bylo 1 pro reálné obrázky, 0 pro fake

generátor se naopak snaží, aby $D(G(z))$ bylo 1 pro falešné obrázky

Trénování generativní adversarial sítě

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for k steps do
        • Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
        • Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{\text{data}}(x)$ .
        • Update the discriminator by ascending its stochastic gradient:
            
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[ \log D(x^{(i)}) + \log (1 - D(G(z^{(i)}))) \right].$$

    end for
    • Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
    • Update the generator by descending its stochastic gradient:
            
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(z^{(i)}))).$$

end for
```

update diskriminátoru

gradient na diskriminátor

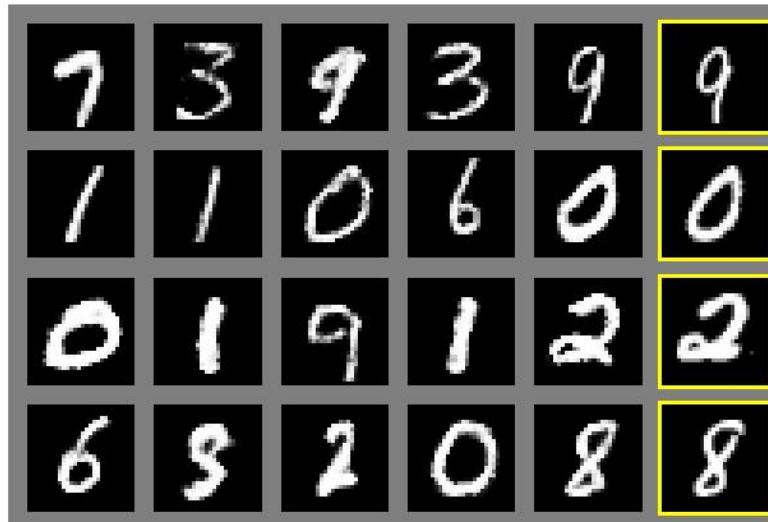
update generátoru

gradient na generátor

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

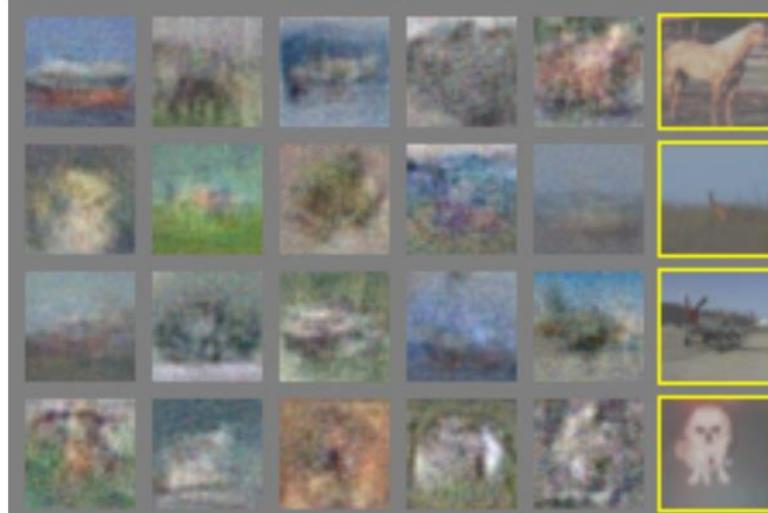
GAN ukázky v původním článku

MNIST



a)

CIFAR



c)
zdroj: [Goodfellow et al.: Generative Adversarial Networks](#)

TFD

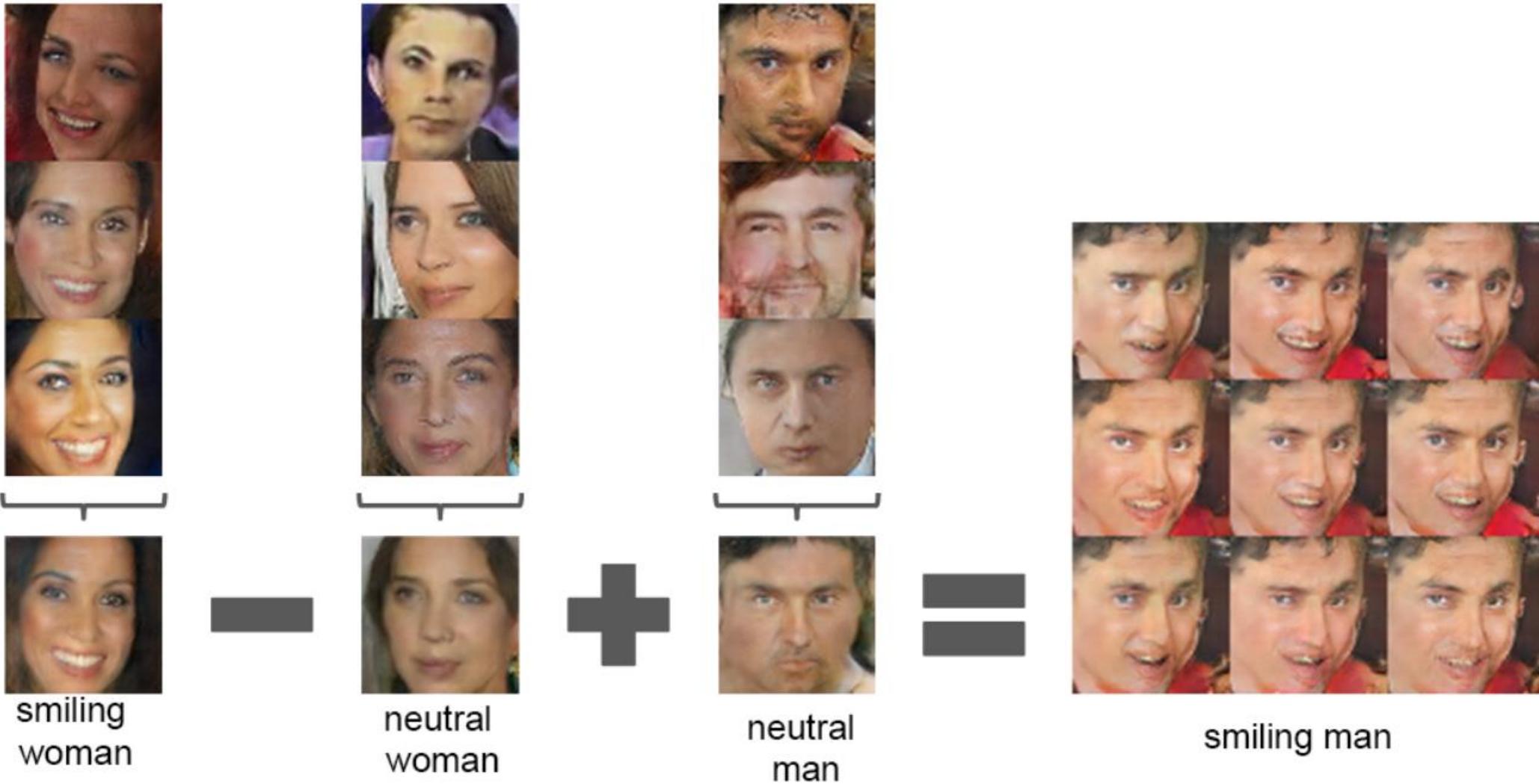


b)



d)

Vektorová aritmetika



článek: [Radford et al.: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks](#)

Ukázka GAN obrázků z poslední doby



Figure 5: 1024×1024 images generated using the CELEBA-HQ dataset. See Appendix F for a larger set of results, and the accompanying video for latent space interpolations. On the right, two images from an earlier megapixel GAN by Marchesi (2017) show limited detail and variation.

Ukázka GAN obrázků z poslední doby



Mao et al. (2016b) (128×128)

Gulrajani et al. (2017) (128×128)

Our (256×256)

Figure 6: Visual quality comparison in LSUN BEDROOM; pictures copied from the cited articles.