

# Počítačové vidění

---

Detekce a segmentace objektů

# Klasifikace vs lokalizace vs detekce

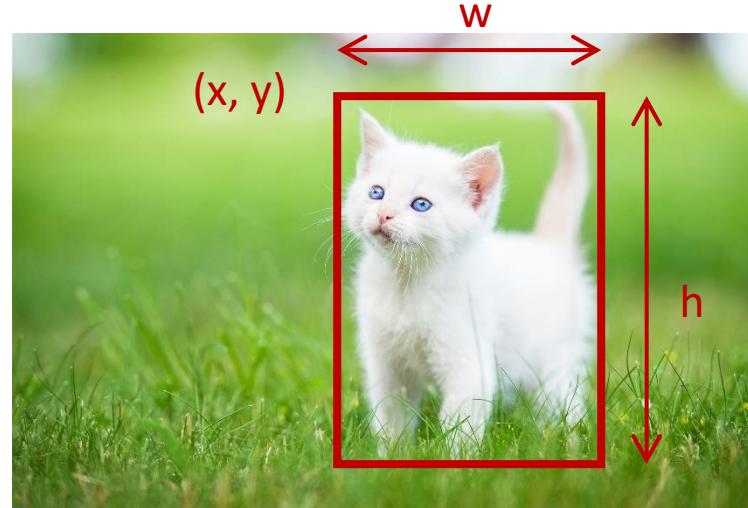
klasifikace



return 1

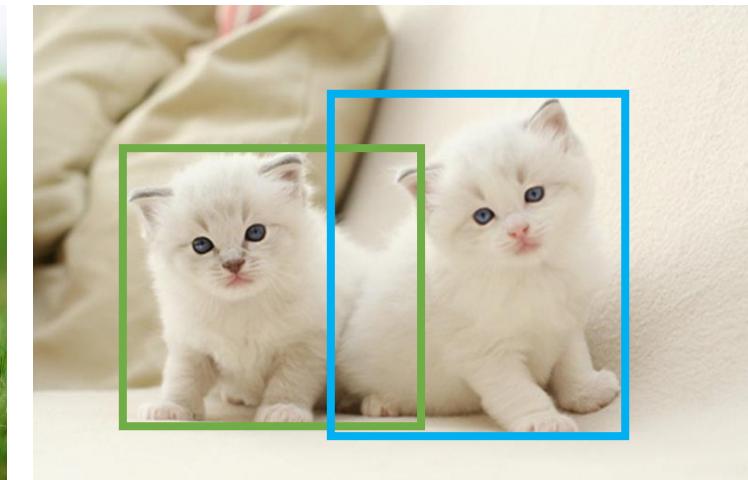
index třídy "kočka"

lokalizace



return  $(1, x, y, w, h)$

detekce



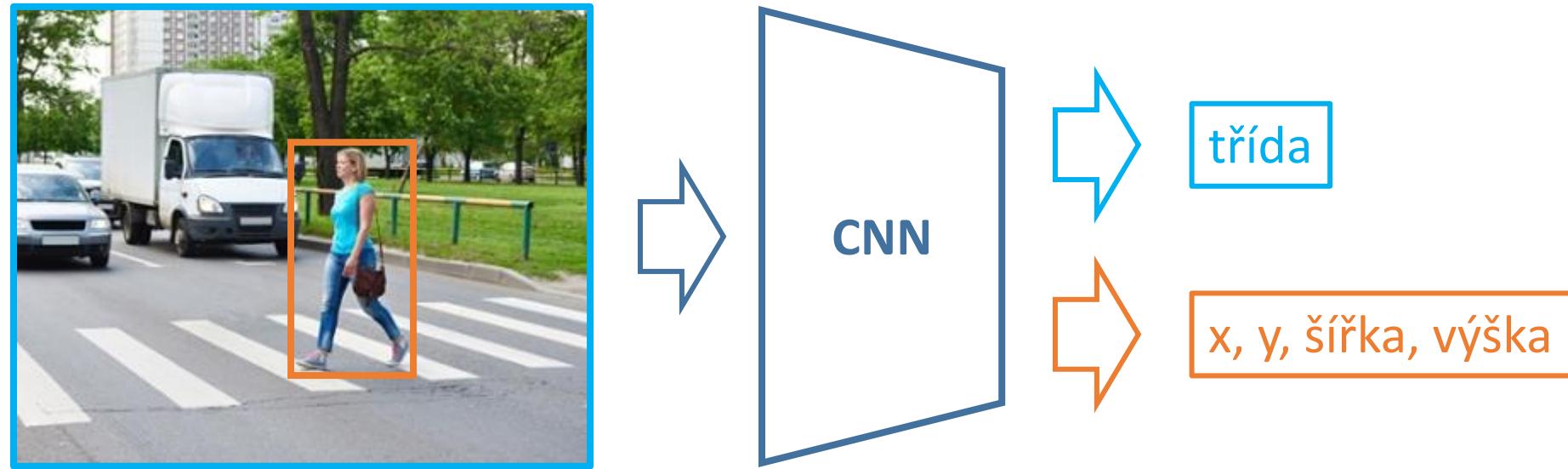
return [  
 $(1, x_1, y_1, w_1, h_1),$   
 $(1, x_2, y_2, w_2, h_2)$   
]

# Lokalizace

---

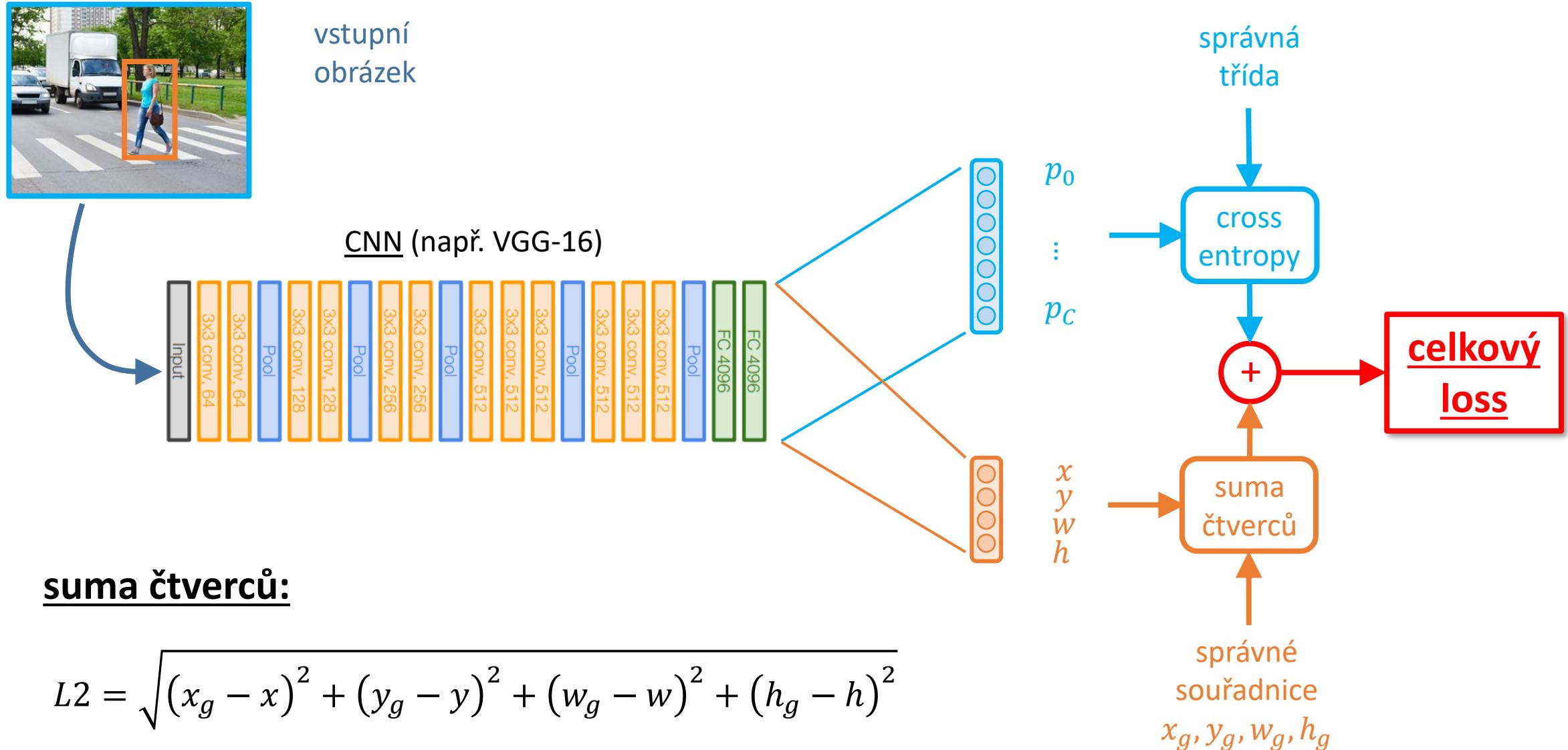
# Lokalizace

kromě třídy bude síť zároveň predikovat pozici objektů ve formě obdélníků



problém transformován na úlohu klasifikace + regrese

# Lokalizace: trénování



# Trénovací obrázky bez objektů

- Celkový loss tedy je

$$L = \underbrace{\log\left(\frac{\exp(s_y)}{\sum_{c=0}^C \exp(s_c)}\right)}_{\text{softmax cross entropy}} + \alpha \|\mathbf{b} - \mathbf{g}\|_2$$

L2 suma čtverců

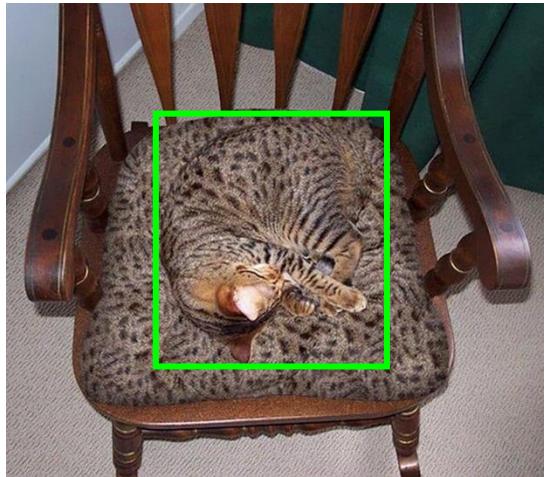
často se používá tzv. Smooth L1

$$L1^s(x) = \begin{cases} 0.5x^2 & |x| < 1 \\ |x| - 0.5 & \text{jinak} \end{cases}$$

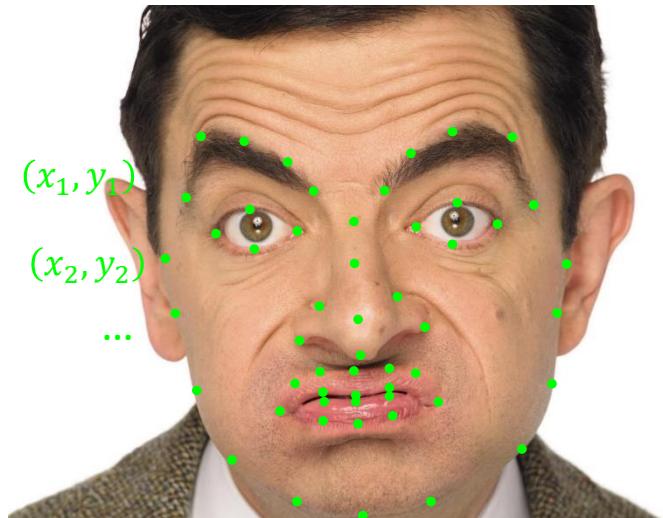
kde  $y$  je správná třída,  $\mathbf{b} = [x, y, w, h]$  je predikovaný box a  $\mathbf{g} = [x_g, y_g, w_g, h_g]$

- Pokud na trénovacím obrázku není žádný objekt, třída  $y = 0$  ("pozadí" může mít libovlný index) a L2 člen můžeme ignorovat, např. tak, že  $\alpha = 0$

# Další příklady lokalizace



$$\begin{bmatrix} x \\ y \\ w \\ h \end{bmatrix}$$



$$\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_{68} \\ y_{68} \end{bmatrix}$$

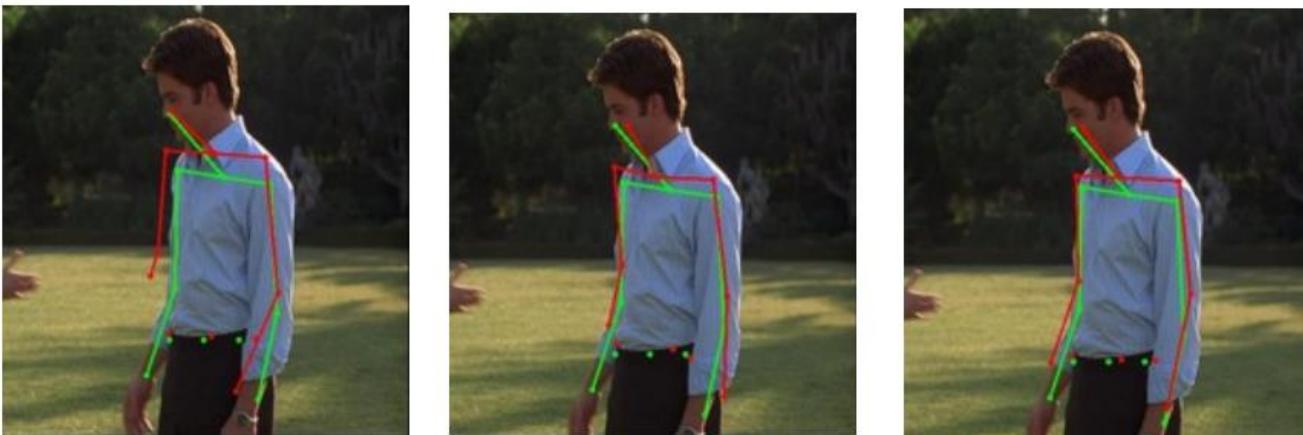


$$\begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_{14} \\ y_{14} \end{bmatrix}$$

# DeepPose



Figure 2. Left: schematic view of the DNN-based pose regression. We visualize the network layers with their corresponding dimensions, where convolutional layers are in blue, while fully connected ones are in green. We do not show the parameter free layers. Right: at stage  $s$ , a refining regressor is applied on a sub image to refine a prediction from the previous stage.



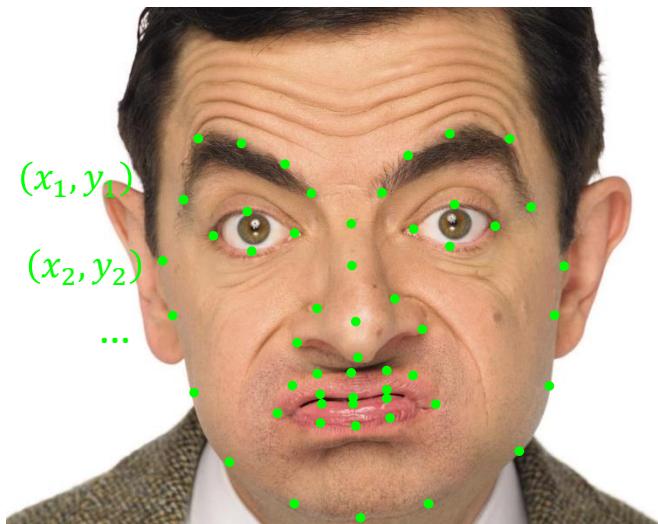
více fází, kdy se predikce pozic kloubů postupně upřesňují

# Detekce

---

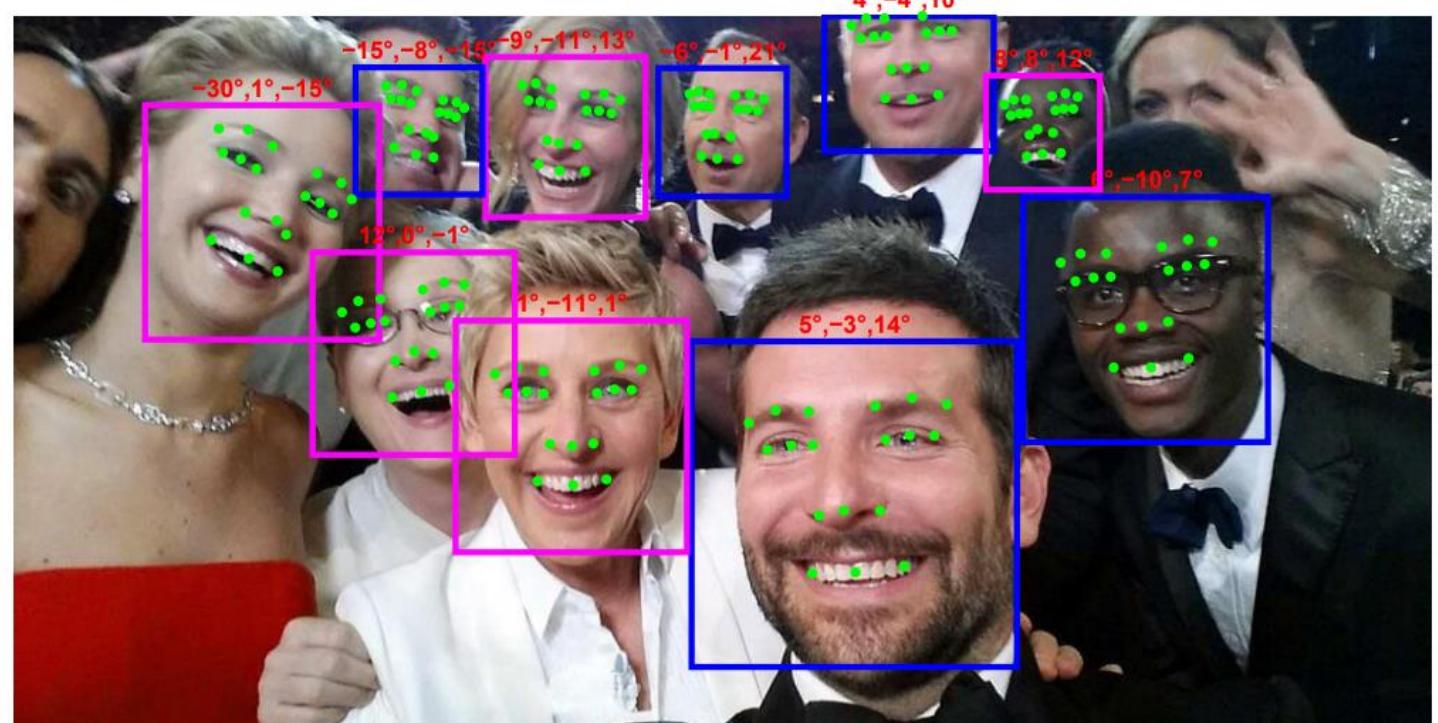
# Detekce jako lokalizace (regrese)?

lokalizace



počet objektů / predikovaných veličin  
je konstantní

detekce

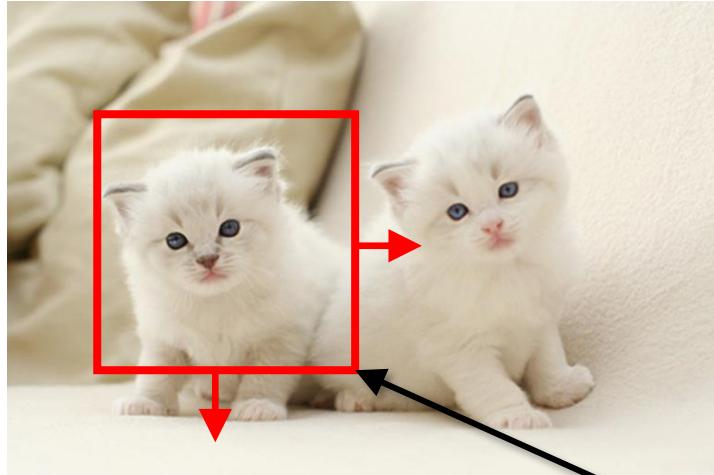


počet objektů / predikovaných veličin  
je variabilní

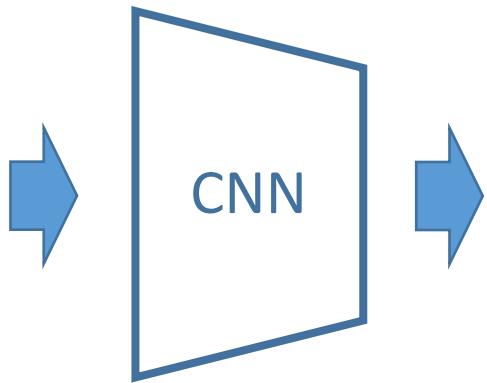
[Ranjan et al.: HyperFace: A Deep Multi-task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition](#)

# Posuvné okénko

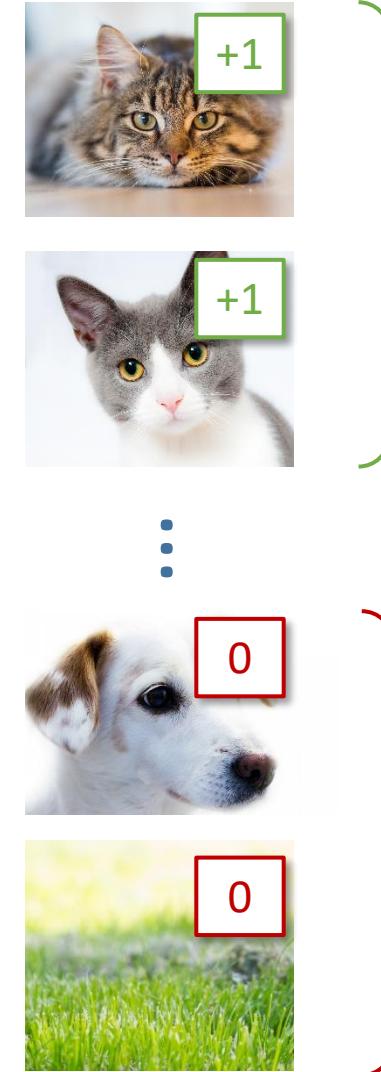
## příklad: detektor koček



v každém kroku se vyřízne  
okénko a klasifikuje



+1 = kočka!



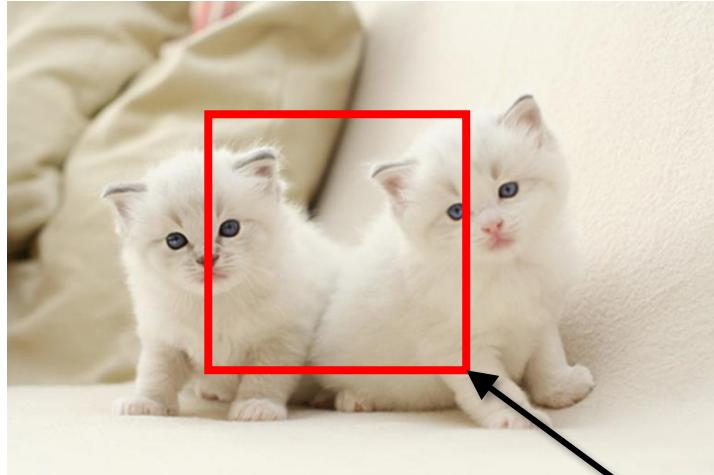
## trénovací data

kladné  
příklady

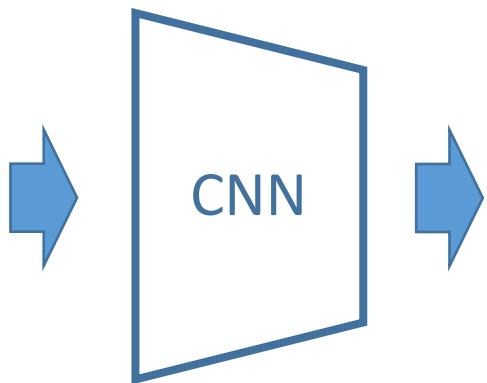
příklady  
“nekoček”  
(pozadí)

# Posuvné okénko

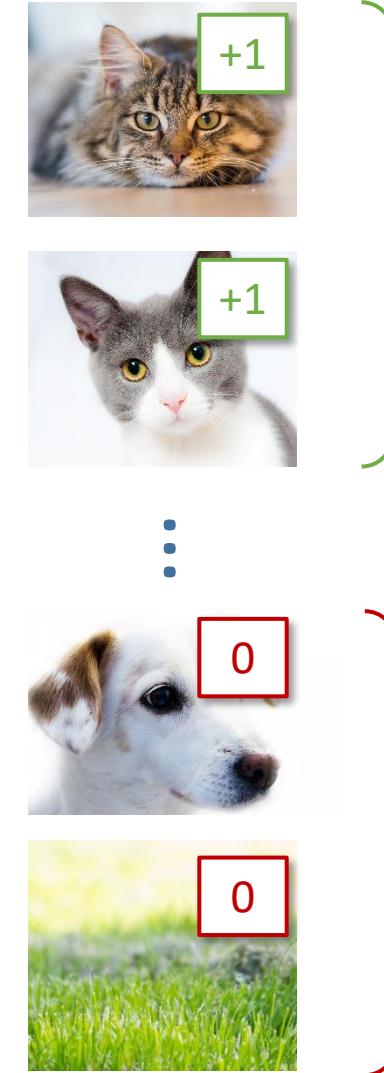
## příklad: detektor koček



v každém kroku se vyřízne  
okénko a klasifikuje



0 = pozadí



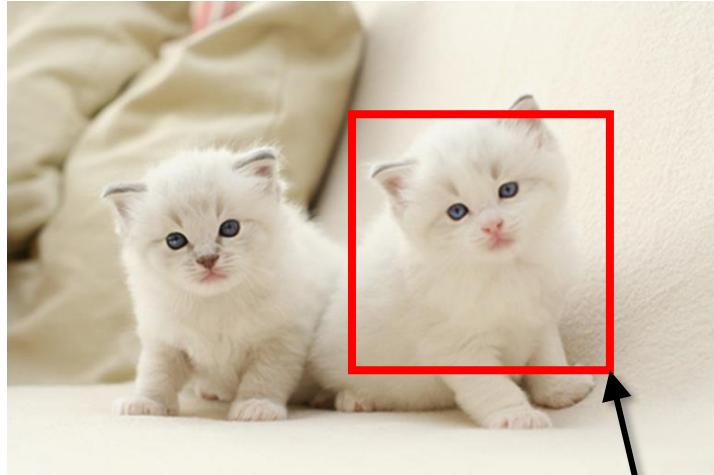
## trénovací data

kladné  
příklady

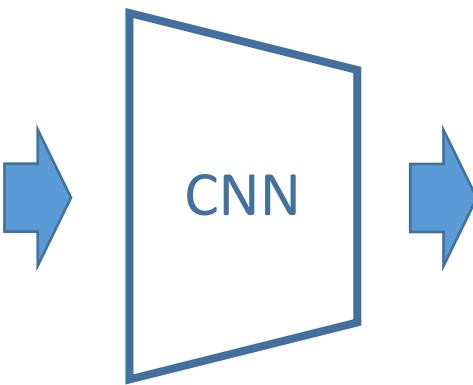
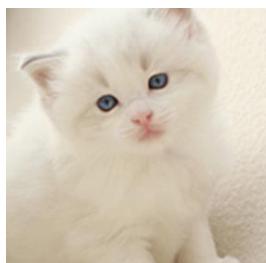
příklady  
„nekoček“  
(pozadí)

# Posuvné okénko

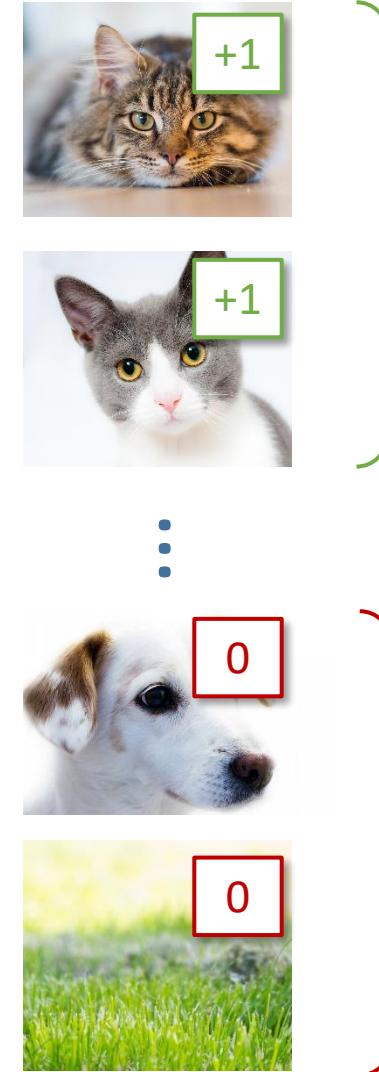
## příklad: detektor koček



v každém kroku se vyřízne  
okénko a klasifikuje



+1 = kočka!



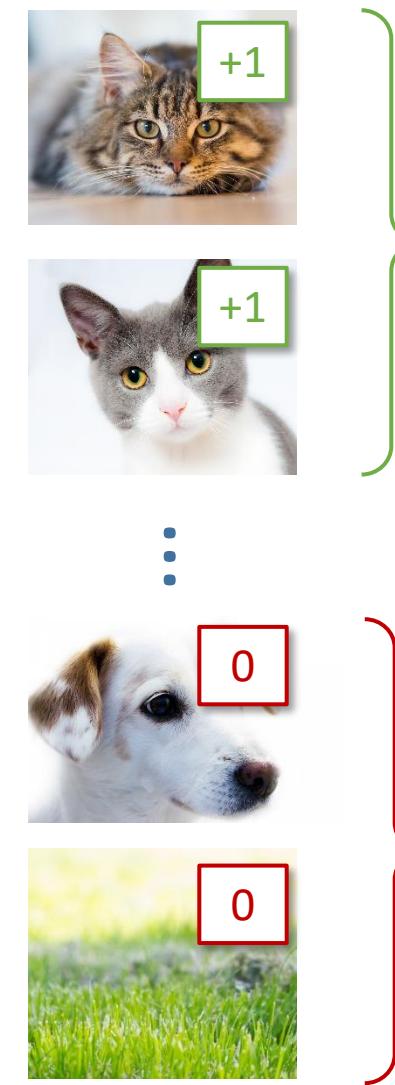
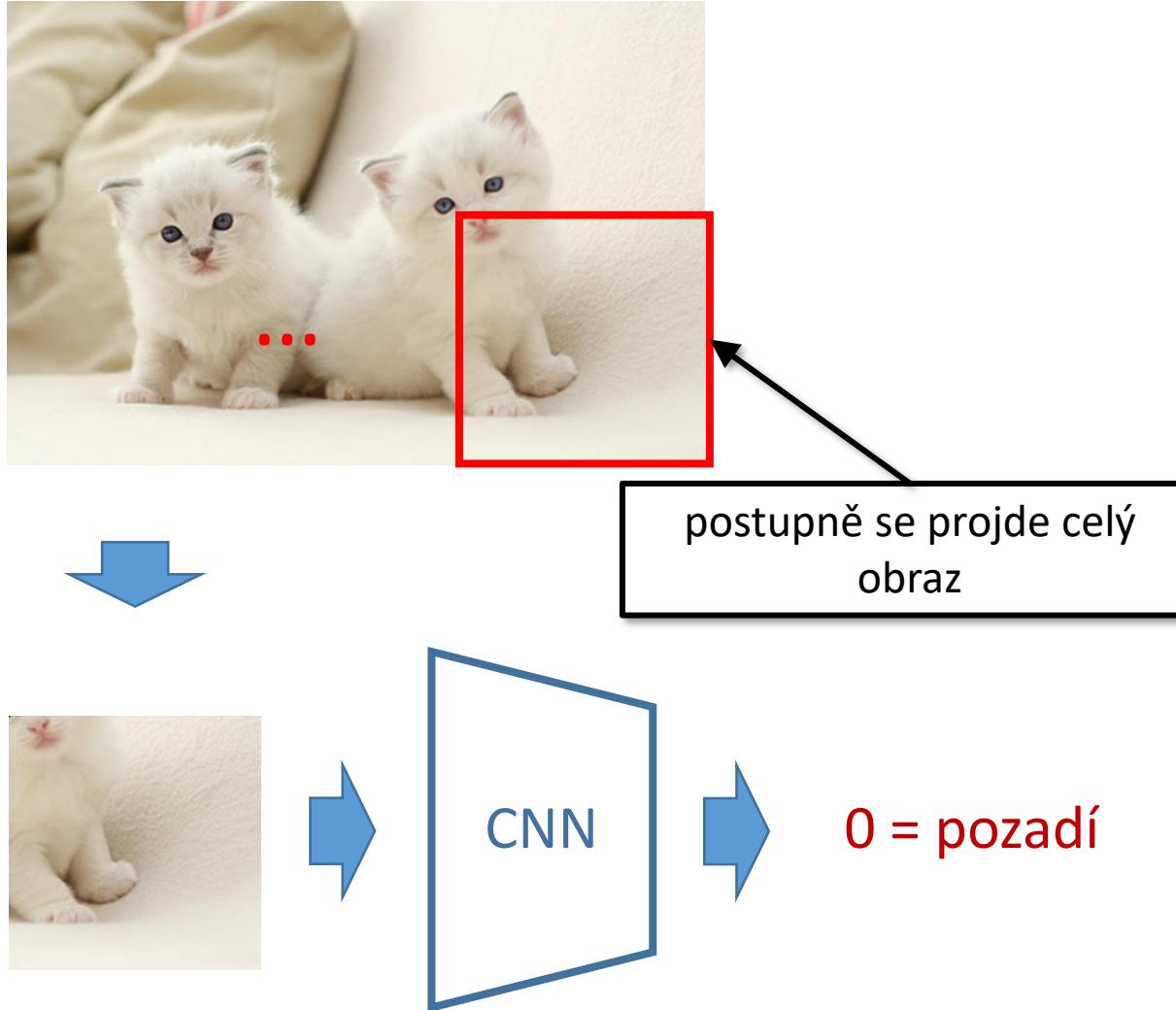
## trénovací data

kladné  
příklady

příklady  
“nekoček”  
(pozadí)

# Posuvné okénko

## příklad: detektor koček



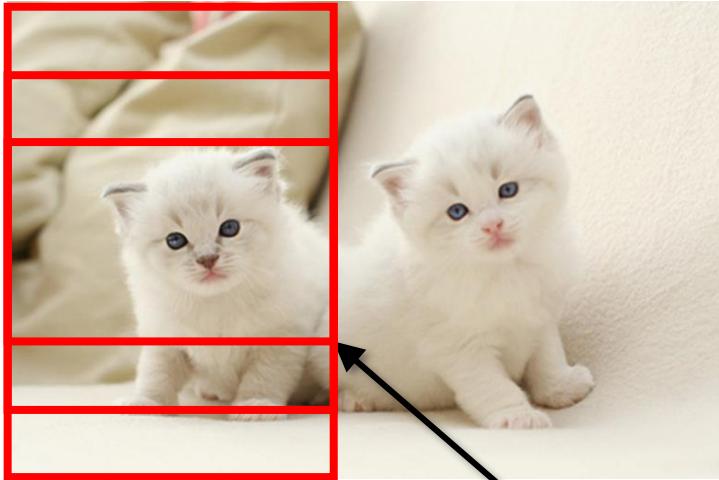
## trénovací data

kladné  
příklady

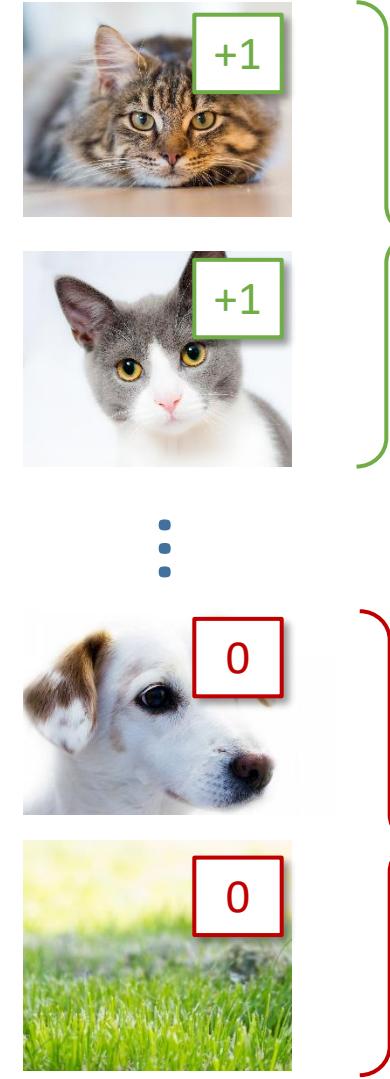
příklady  
"nekoček"  
(pozadí)

# Posuvné okénko

## příklad: detektor koček



poté to samé pro větší  
okénko, abychom  
mohli detektovat různě  
velké objekty



## trénovací data

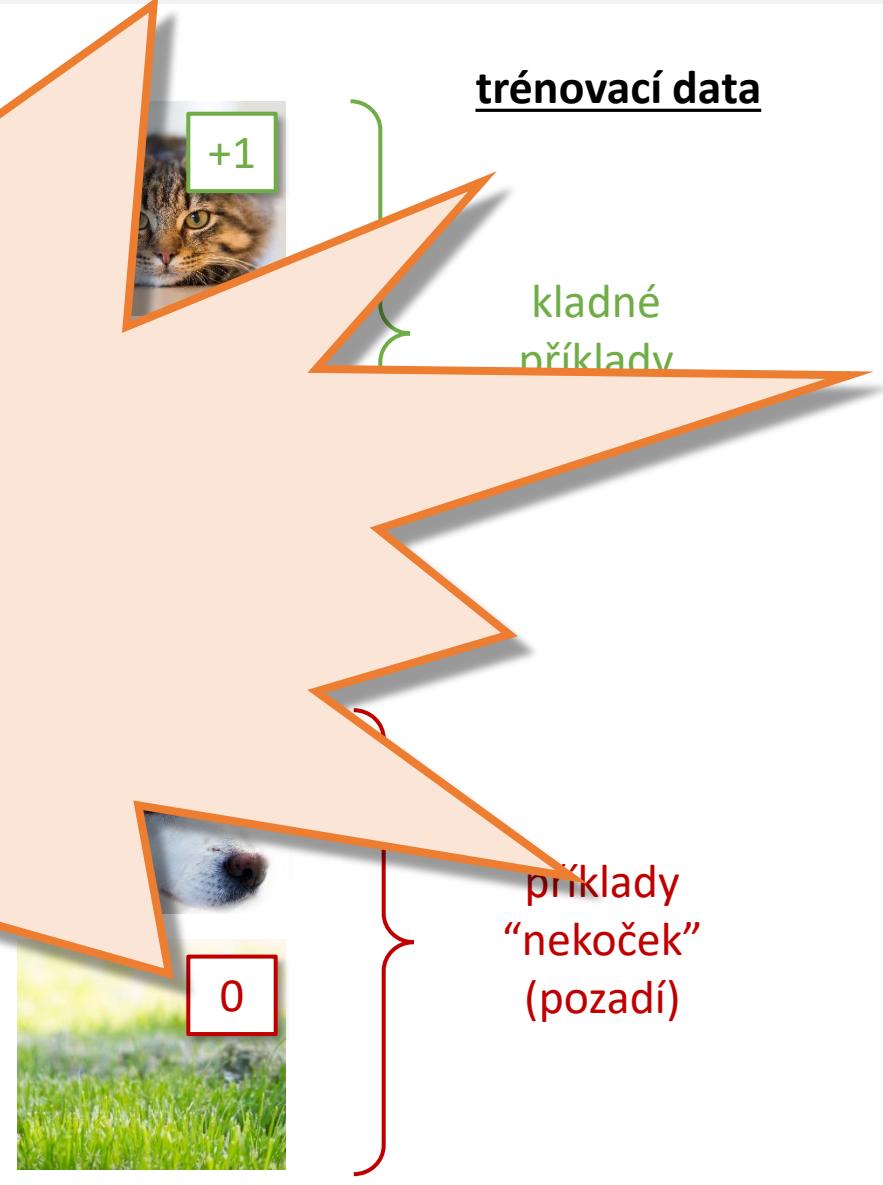
kladné  
příklady

příklady  
“nekoček”  
(pozadí)

# Posuvné okénko

příklad: detektor koček

Pomalé 😞

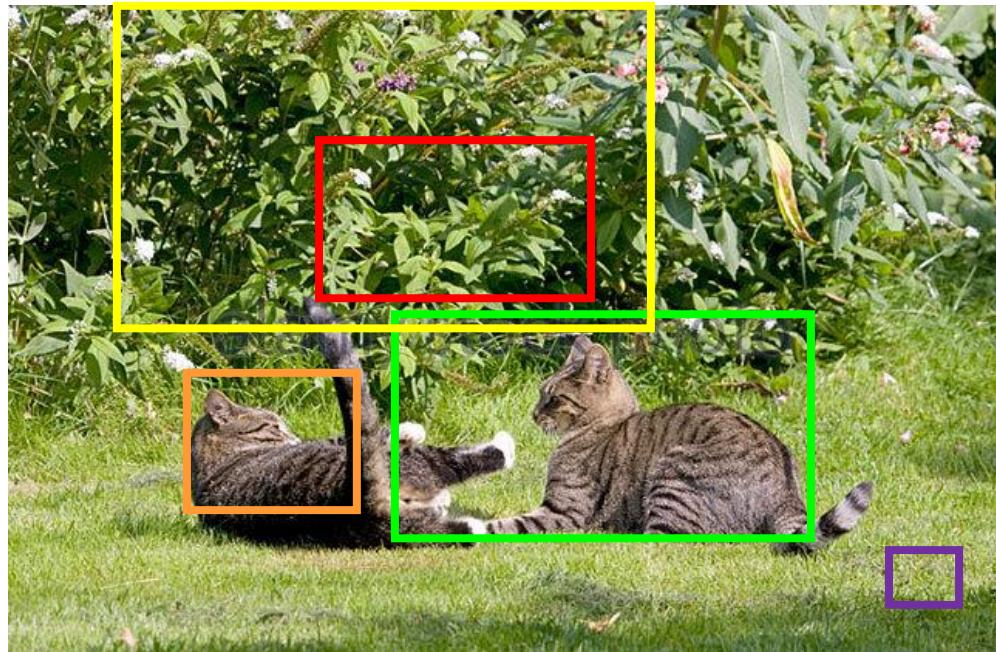


# R-CNN

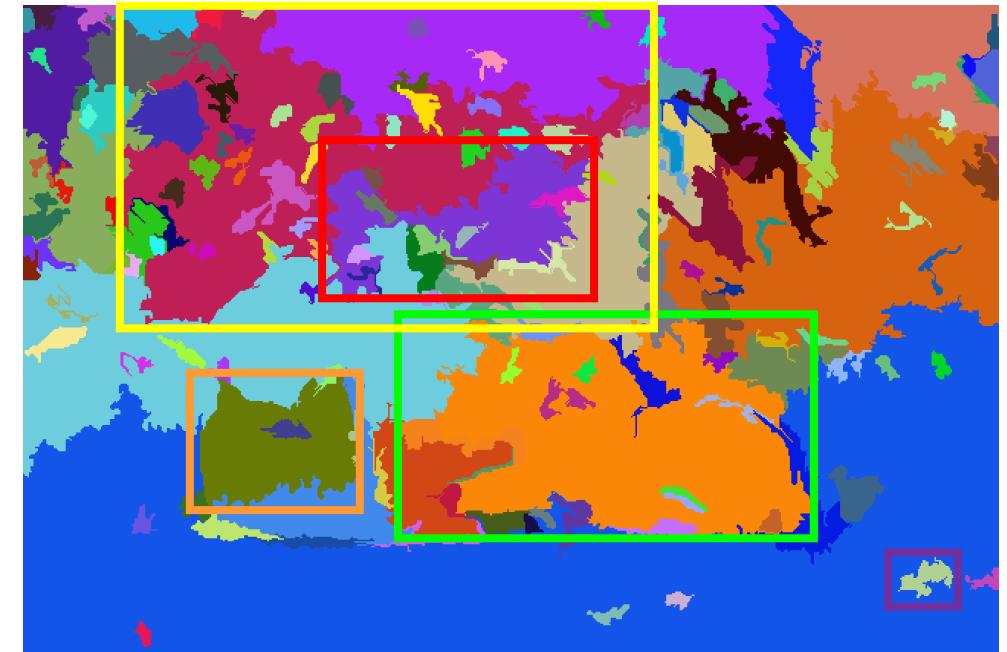
---

# Region proposal

vstupní obrázek



segmentace obrázku



zde příklad selective search

místo exhaustivního procházení obrázku se vyhodnotí **jen vybrané výřezy (tzv. region proposals)** → tzv. Region-based Convolutional Neural Networks (R-CNN)

[Girshick et al: Rich feature hierarchies for accurate object detection and semantic segmentation](#)

# R-CNN

Vstupní  
obraz



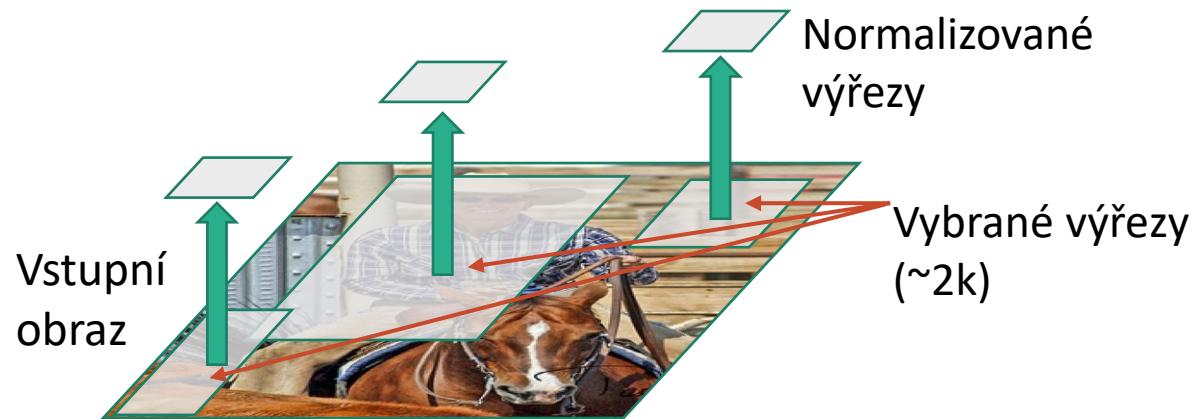
obrázek: <https://dl.dropboxusercontent.com/s/vlyrkgd8nz8gy5l/fast-rcnn.pdf?dl=0>

# R-CNN



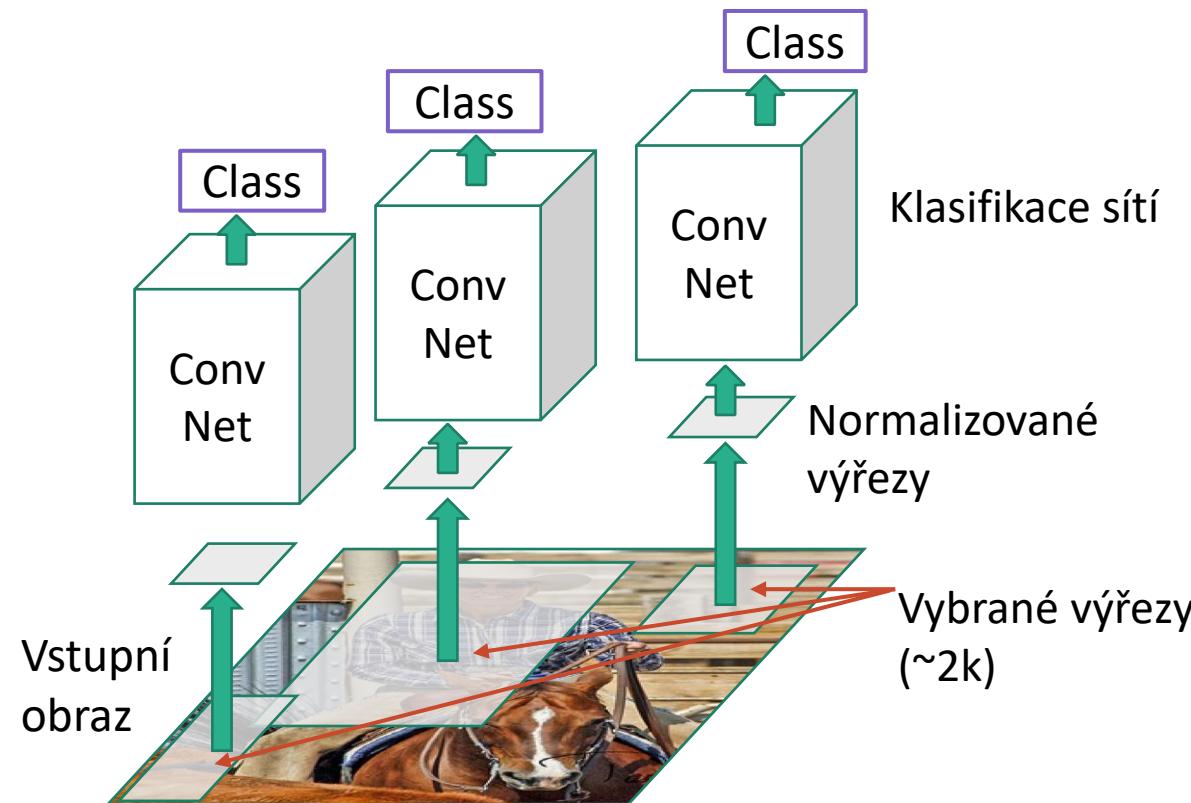
obrázek: <https://dl.dropboxusercontent.com/s/vlyrkgd8nz8gy5l/fast-rcnn.pdf?dl=0>

# R-CNN



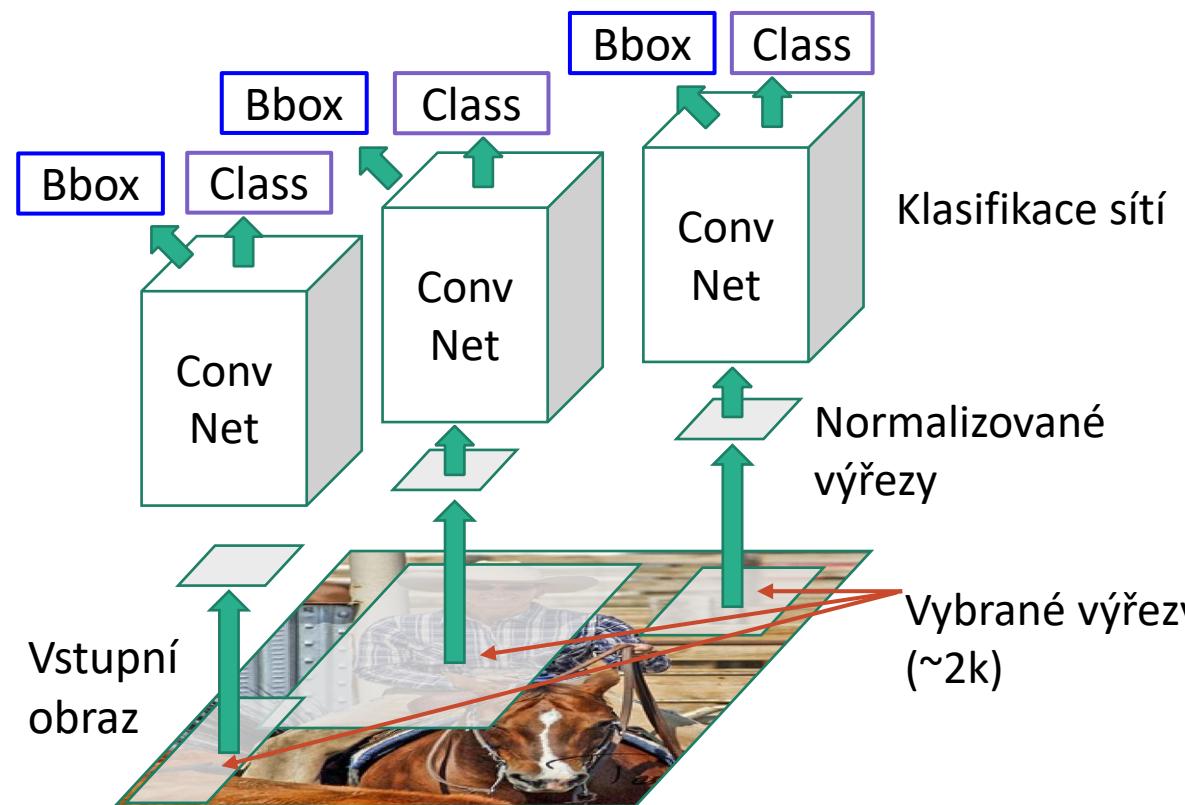
obrázek: <https://dl.dropboxusercontent.com/s/vlyrkgd8nz8gy5l/fast-rcnn.pdf?dl=0>

# R-CNN



Klasifikace výřežů

# R-CNN



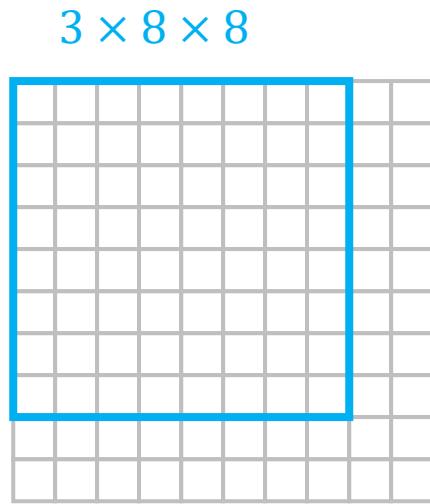
Klasifikace výřezů

Regresce bounding boxů:  
predikuje se transformace RoI:  
 $(t_x, t_y, t_h, t_w)$

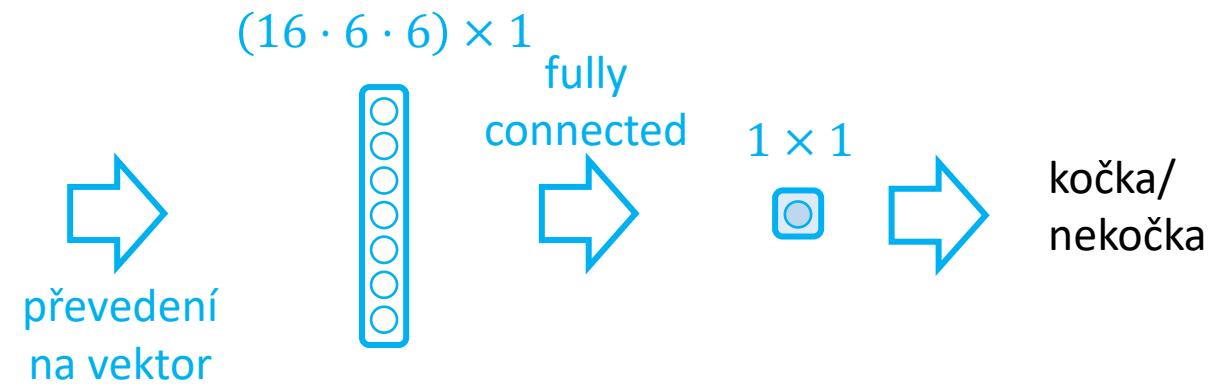
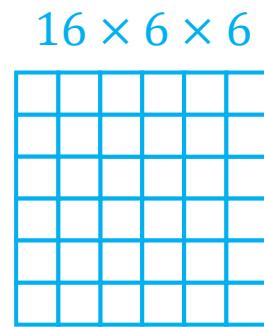
# Posuvné okno jako konvoluce

dva regiony v  $3 \times 10 \times 10$

obrázku: modrý a červený



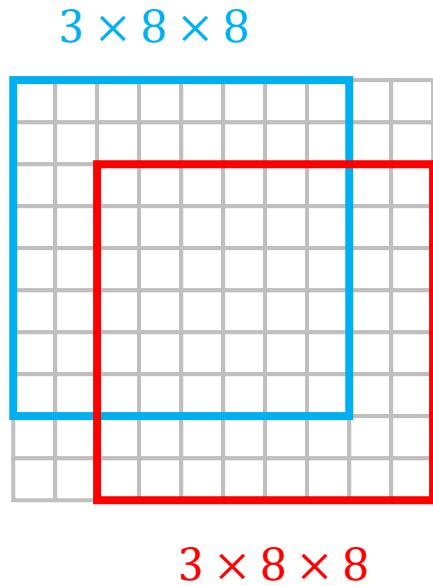
3x3 konv.  
→



# Posuvné okno jako konvoluce

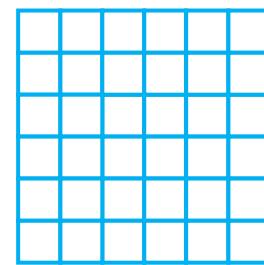
dva regiony v  $3 \times 10 \times 10$

obrázku: modrý a červený



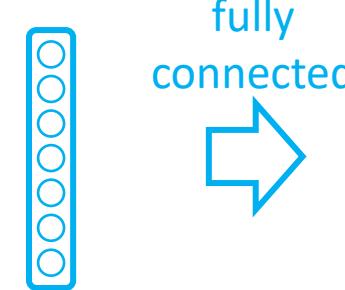
3x3 konv.  
3x3 konv.

$16 \times 6 \times 6$



převedení  
na vektor

$(16 \cdot 6 \cdot 6) \times 1$

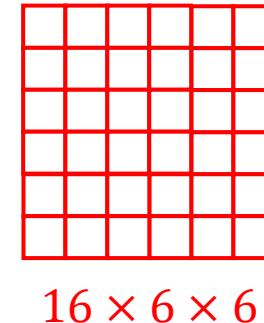


fully  
connected

$1 \times 1$

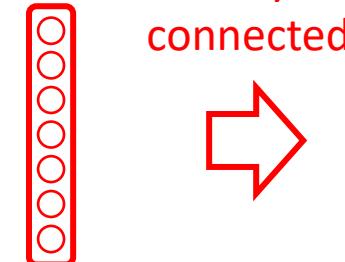


kočka/  
nekočka



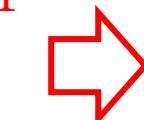
převedení  
na vektor

$(16 \cdot 6 \cdot 6) \times 1$



fully  
connected

$1 \times 1$



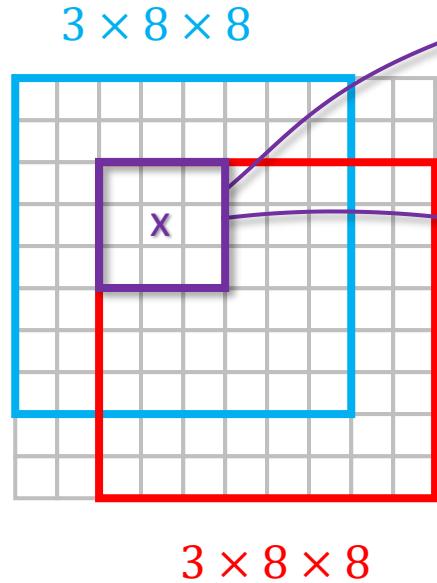
kočka/  
nekočka

- dva samostatné a oddělené průchody modrým a červeným regionem

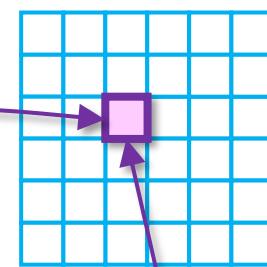
# Posuvné okno jako konvoluce

dva regiony v  $3 \times 10 \times 10$

obrázku: modrý a červený

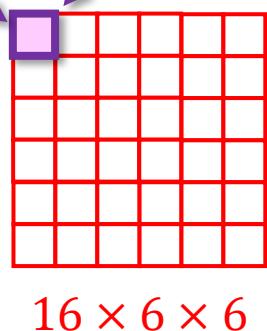


$16 \times 6 \times 6$



převedení  
na vektor

hodnoty jsou stejné!



převedení  
na vektor

$(16 \cdot 6 \cdot 6) \times 1$

fully  
connected



$1 \times 1$



kočka/  
nekočka

$(16 \cdot 6 \cdot 6) \times 1$

fully  
connected



$1 \times 1$



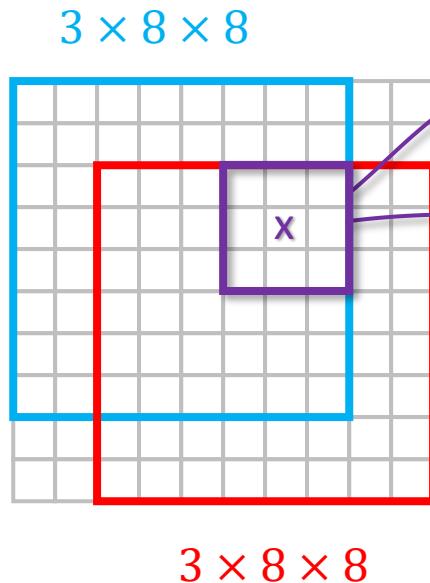
kočka/  
nekočka

- dva samostatné a oddělené průchody modrým a červeným regionem

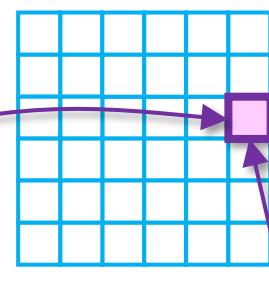
# Posuvné okno jako konvoluce

dva regiony v  $3 \times 10 \times 10$

obrázku: modrý a červený

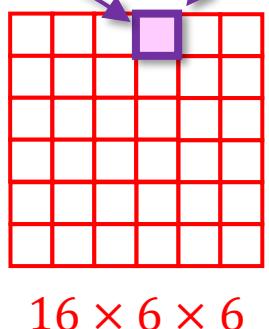


$16 \times 6 \times 6$



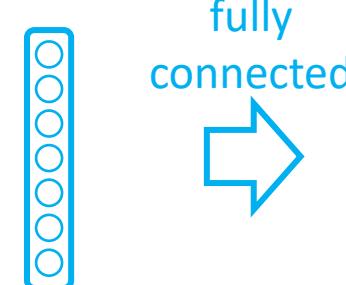
převedení na vektor

hodnoty jsou stejné!



převedení na vektor

$(16 \cdot 6 \cdot 6) \times 1$



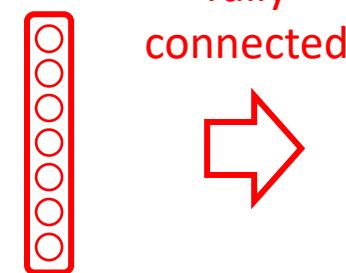
fully  
connected

$1 \times 1$



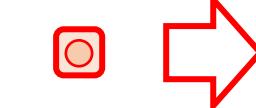
kočka/  
nekočka

$(16 \cdot 6 \cdot 6) \times 1$



fully  
connected

$1 \times 1$



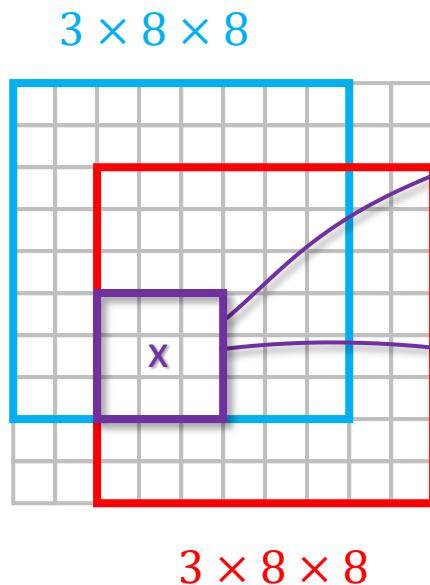
kočka/  
nekočka

- dva samostatné a oddělené průchody modrým a červeným regionem

# Posuvné okno jako konvoluce

dva regiony v  $3 \times 10 \times 10$

obrázku: modrý a červený



3x3 konv.

3x3 konv.

$16 \times 6 \times 6$

$16 \times 6 \times 6$

převedení  
na vektor

převedení  
na vektor

$(16 \cdot 6 \cdot 6) \times 1$

fully  
connected



$1 \times 1$



kočka/  
nekočka

$(16 \cdot 6 \cdot 6) \times 1$

fully  
connected



$1 \times 1$



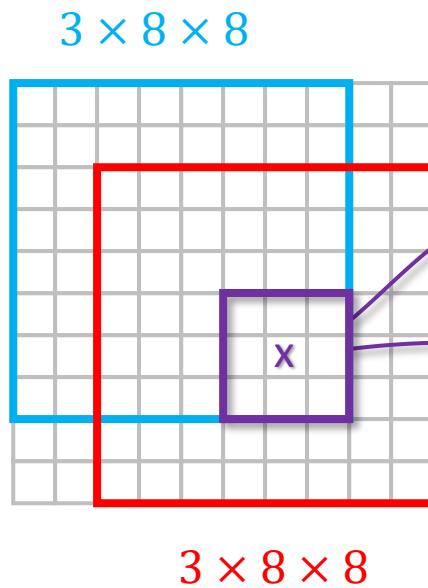
kočka/  
nekočka

- dva samostatné a oddelené průchody modrým a červeným regionem

# Posuvné okno jako konvoluce

dva regiony v  $3 \times 10 \times 10$

obrázku: modrý a červený



3x3 konv.

3x3 konv.

$16 \times 6 \times 6$

$16 \times 6 \times 6$

převedení  
na vektor

převedení  
na vektor

$(16 \cdot 6 \cdot 6) \times 1$  fully connected



fully connected

$1 \times 1$

$1 \times 1$

kočka/  
nekočka

kočka/  
nekočka

hodnoty jsou stejné!

$(16 \cdot 6 \cdot 6) \times 1$  fully connected



fully connected

$1 \times 1$

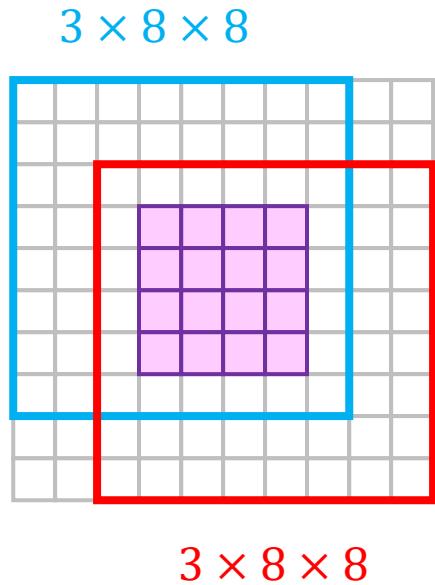
$1 \times 1$

- dva samostatné a oddělené průchody modrým a červeným regionem

# Posuvné okno jako konvoluce

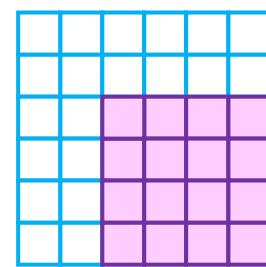
dva regiony v  $3 \times 10 \times 10$

obrázku: modrý a červený

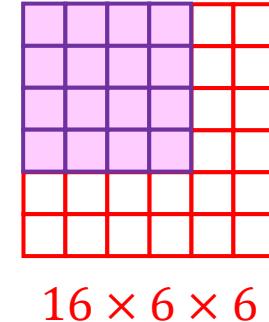


3x3 konv.  
3x3 konv.

$16 \times 6 \times 6$



převedení  
na vektor



převedení  
na vektor

$(16 \cdot 6 \cdot 6) \times 1$  fully  
connected



fully  
connected

$1 \times 1$

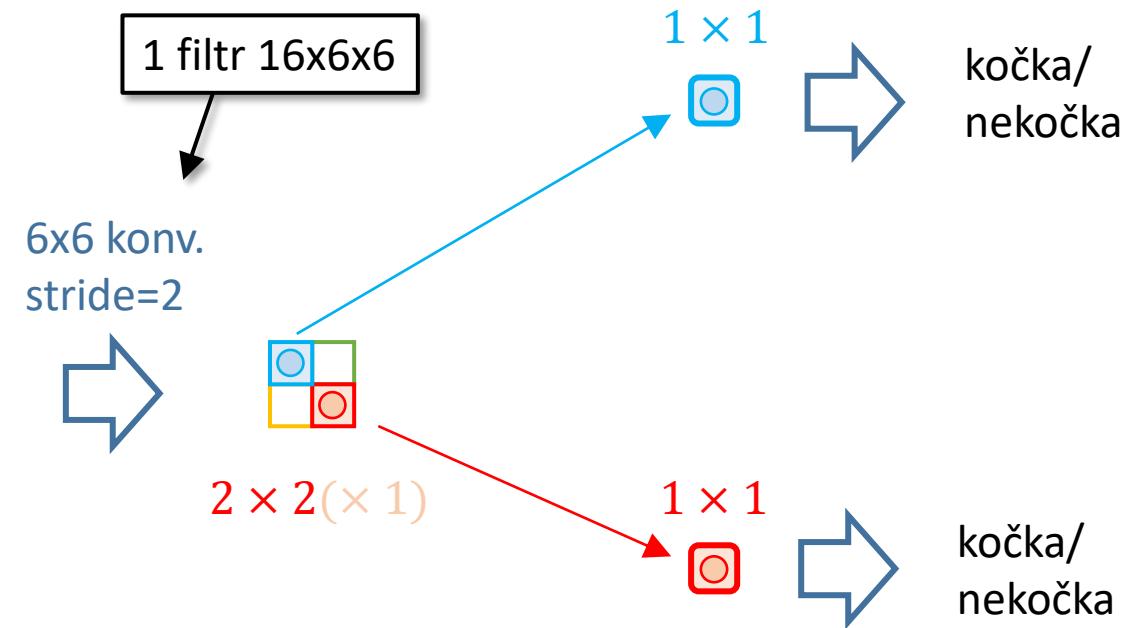
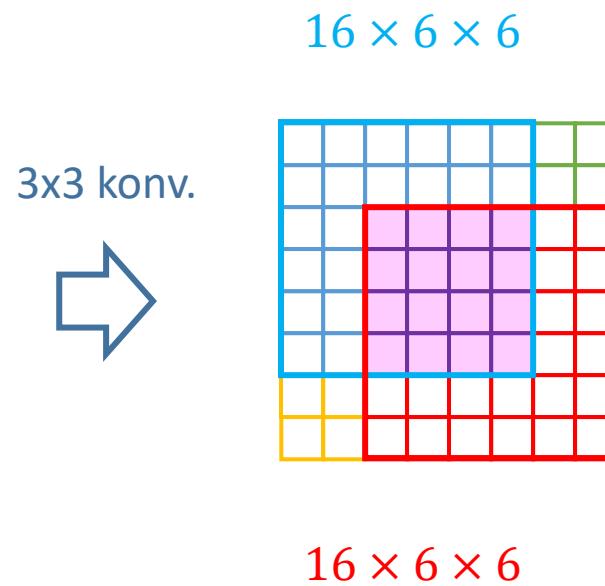
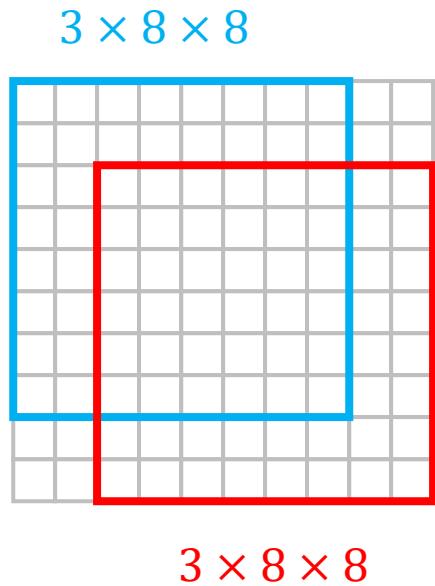


kočka/  
nekočka

- dva samostatné a oddelené průchody modrým a červeným regionem
- ve fialové oblasti jsou výpočty shodné pro modré i červené okno
- pokud jsou průchody nezávislé, zbytečně se opakují

# Posuvné okno jako konvoluce

dva regiony v  $3 \times 10 \times 10$  obrázku: modrý a červený

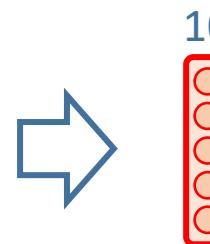
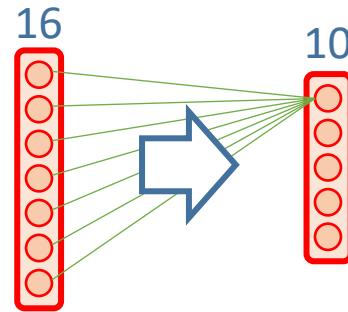
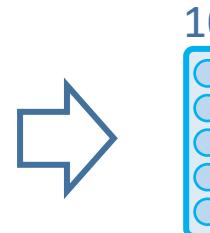
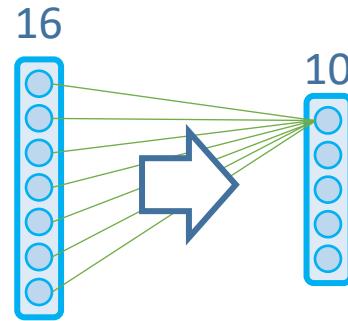


- dopřednou (fully connected) vrstvu jsme nahradili konvolucí
- stačí tedy, aby celý obrázek prošel sítí pouze jednou → jedna velká konvoluce
- tím i výpočty ve fialové oblasti budou provedeny pouze jednou!

# Fully connected jako 1x1 konvoluce

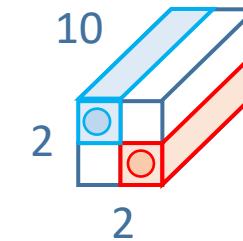
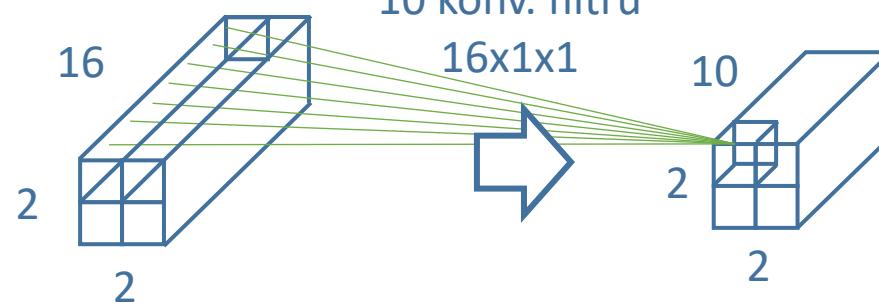
## samosatné průchody

(multi-layer perceptron)



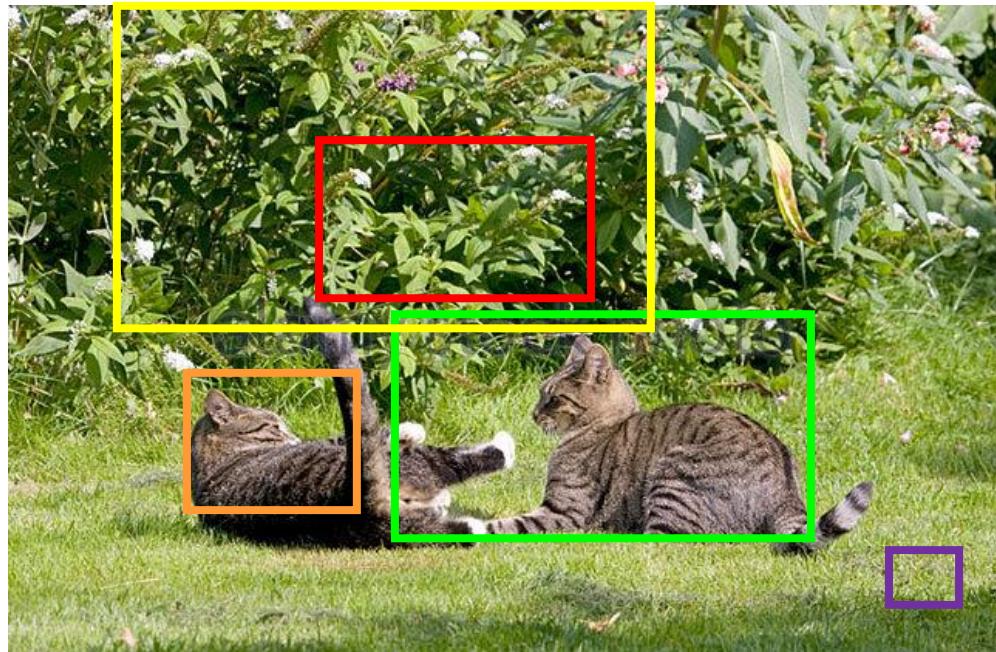
oba průchody ekvivalentní,  
počítají to samé

## jako konvolute

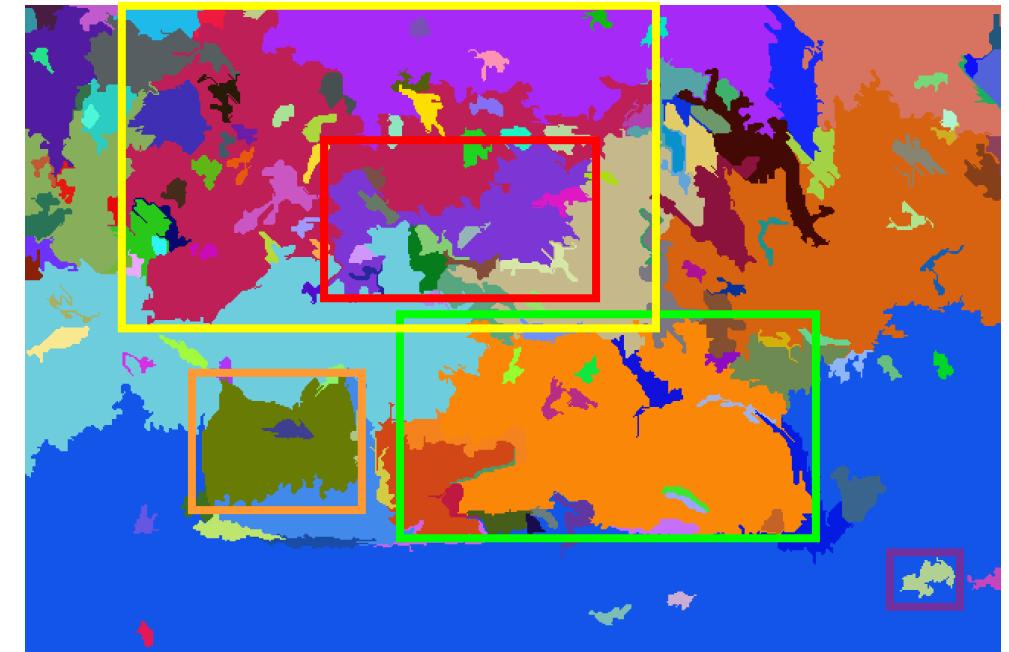


# Fast R-CNN

vstupní obrázek



segmentace obrázku

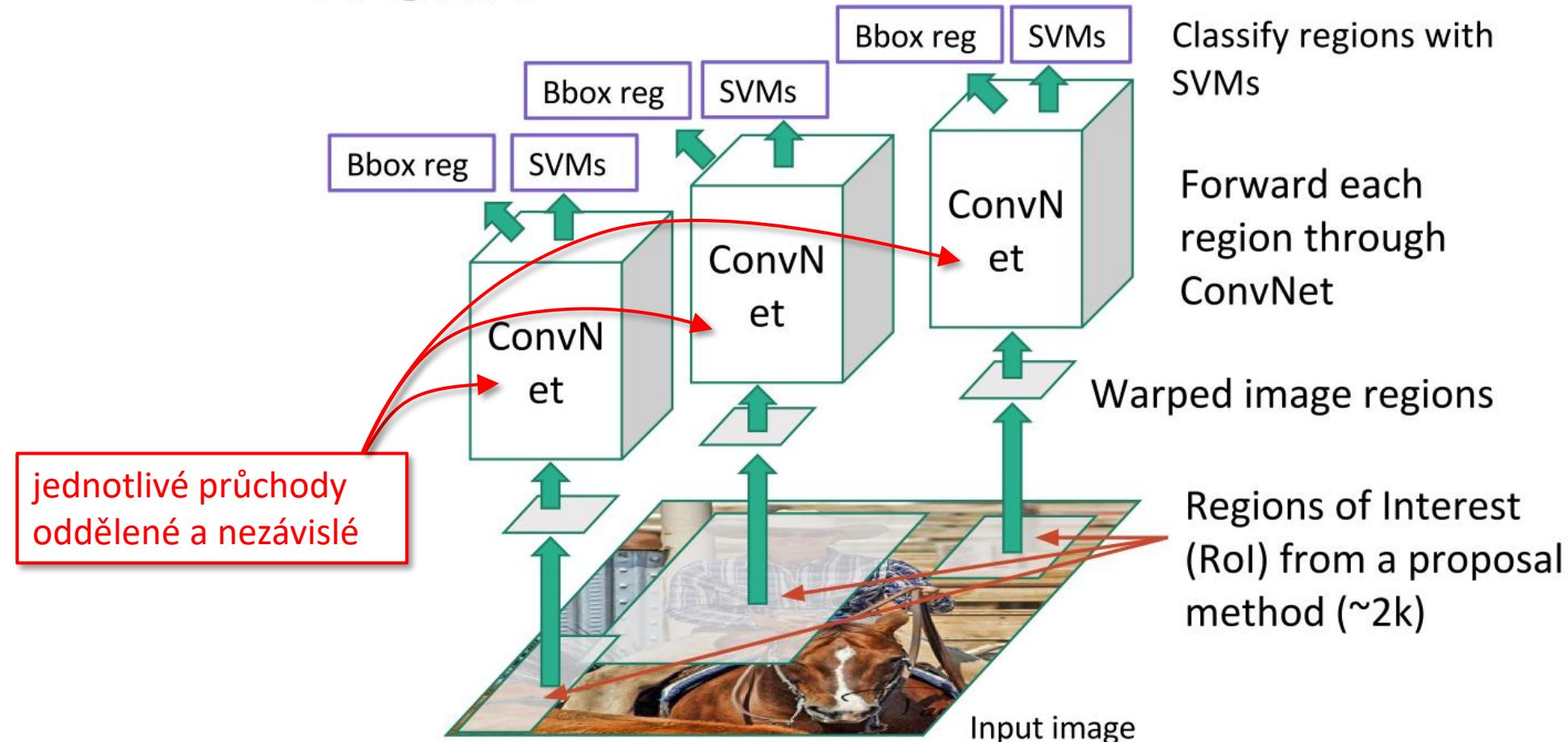


zde příklad selective search

- konvoluce se tedy provede jen jedna přes celý obrázek
- souřadnice vstupních obdélníků se přepočtou do výsledné konvoluční mapy
- klasifikace každé oblasti je pak samostatný průchod

# R-CNN vs Fast R-CNN

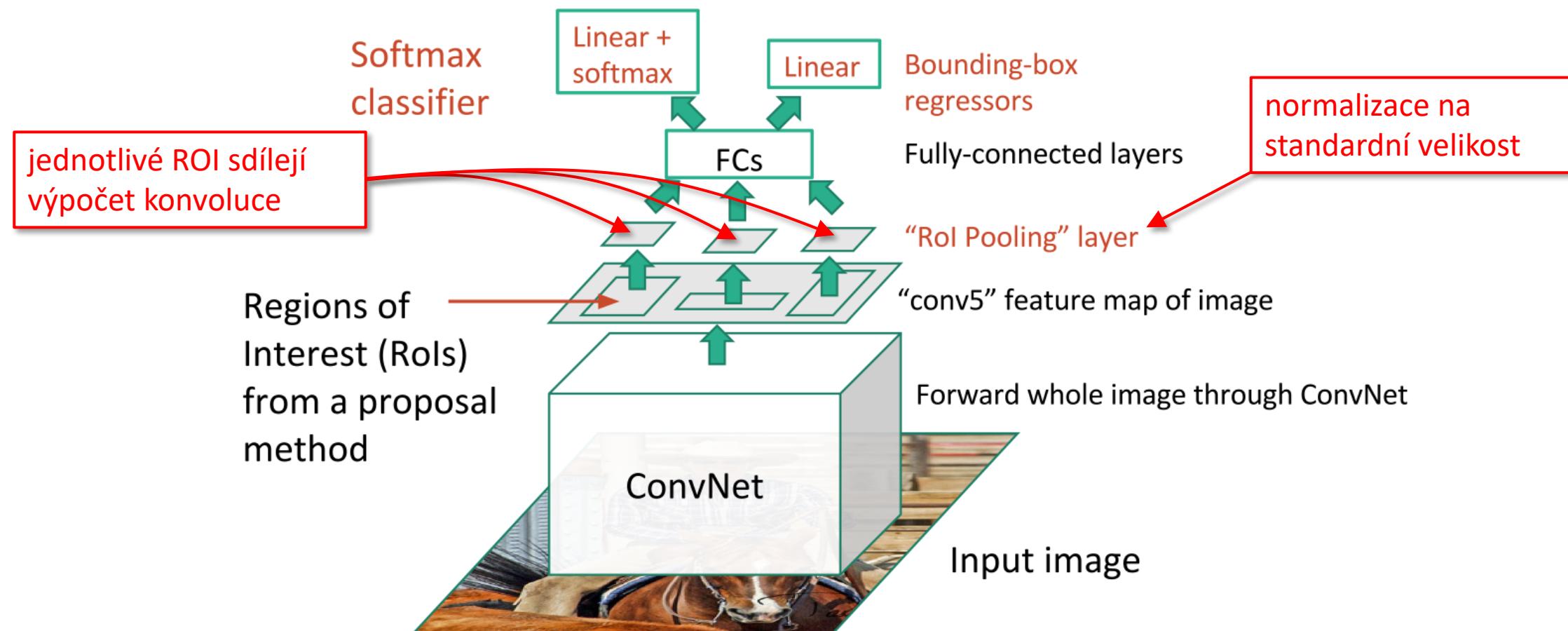
## R-CNN



obrázek: <https://dl.dropboxusercontent.com/s/vlyrkgd8nz8gy5l/fast-rcnn.pdf?dl=0>

# R-CNN vs Fast R-CNN

## Fast R-CNN



obrázek: <https://dl.dropboxusercontent.com/s/vlyrkgd8nz8gy5l/fast-rcnn.pdf?dl=0>

# Rychlost Fast R-CNN

	R-CNN	Fast R-CNN
Trénování	84 hod.	9.5 hod.
Zrychlení trénování	1x	8.8x
Test průchod	47.0 s	0.32 s
Zrychlení testovacího průchodu	1x	146x
mAP (přesnost)	66.0 %	66.9 %

časy jsou bez externího region proposal algoritmu,  
který v případě cca 640x480 obrázku trvá cca 2 s; při  
započtení je zrychlení cca 21x

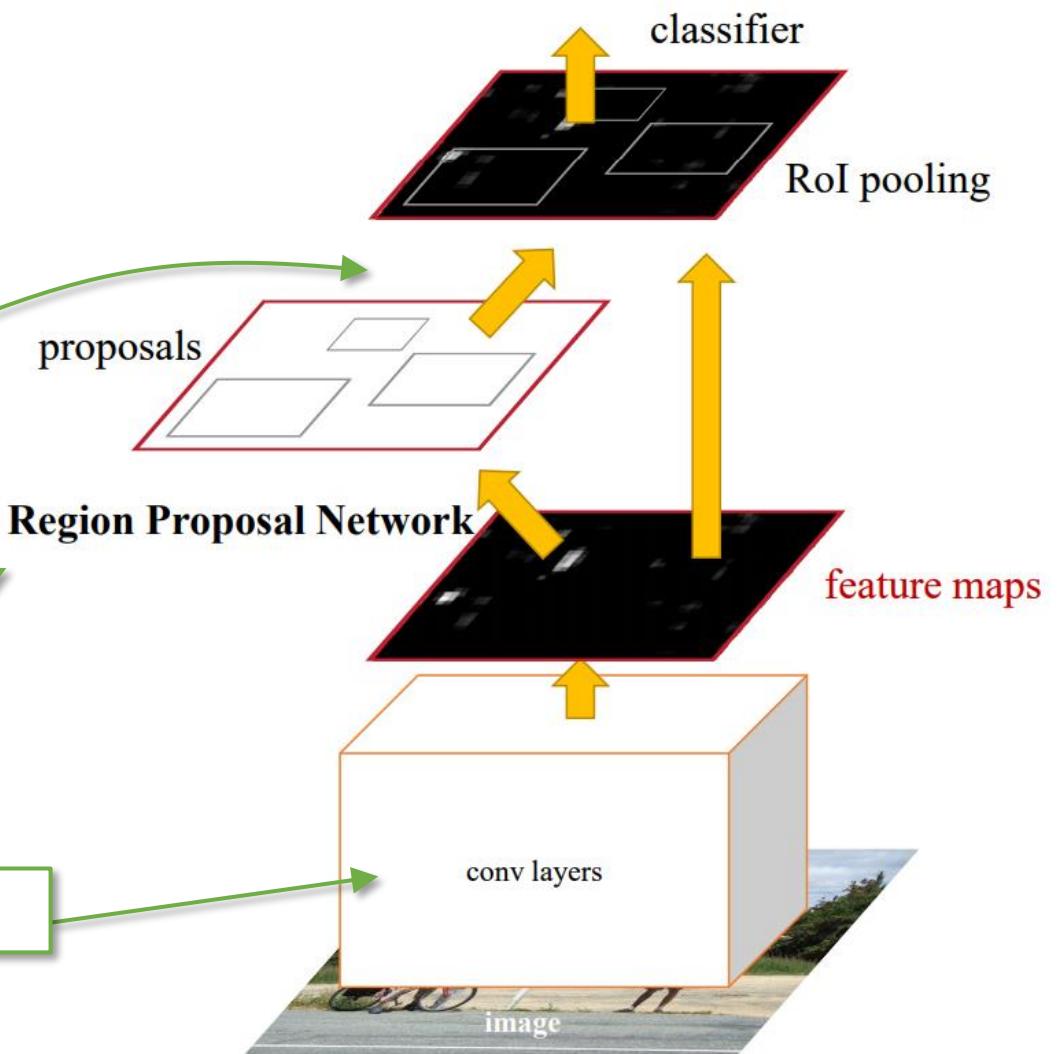
# Faster R-CNN

- všechny komponenty systému jsou trénovatelné, včetně návrhu oblastí (ROI)
- Fast R-CNN má mezi vstupem a konvolucemi nediferencovatelný návrh oblastí

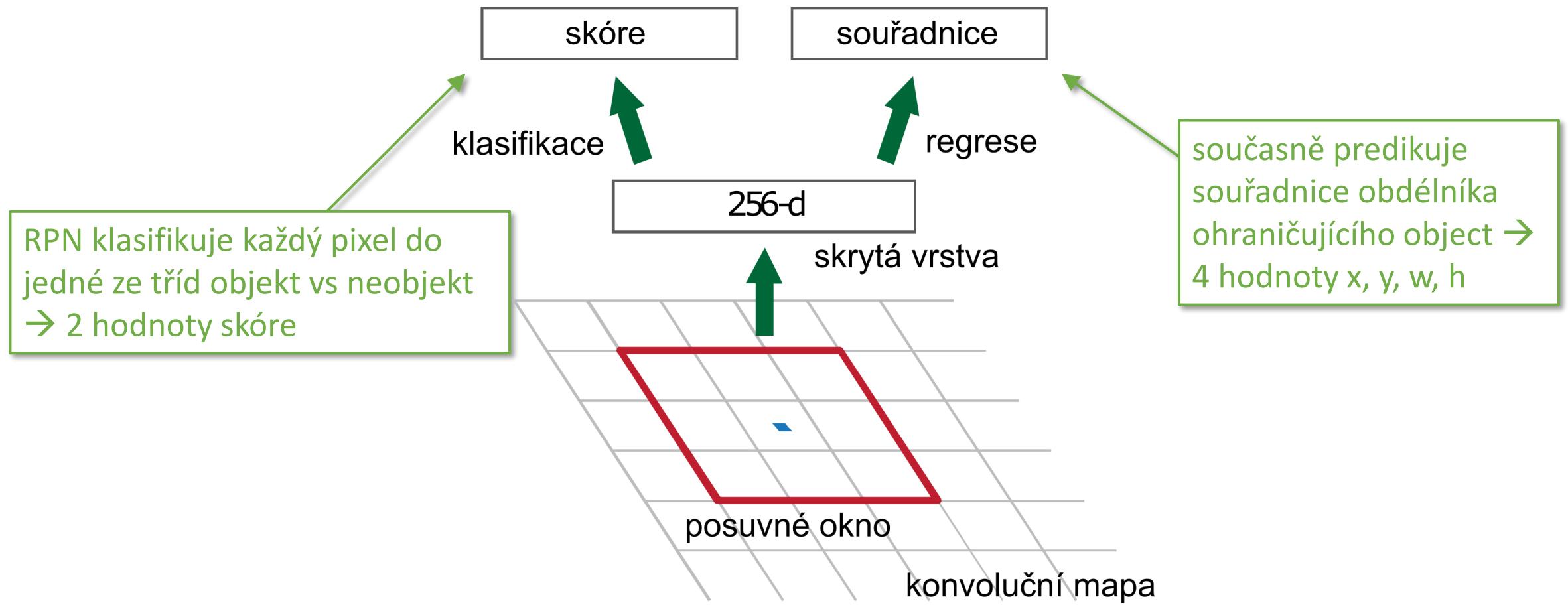
sítí vybere několik ROI (typicky 300) a ty následně klasifikuje

region proposals zajišťuje součást neurosítě, tzv. Region Proposal Net (RPN)

VGG / ResNet / Inception / ...

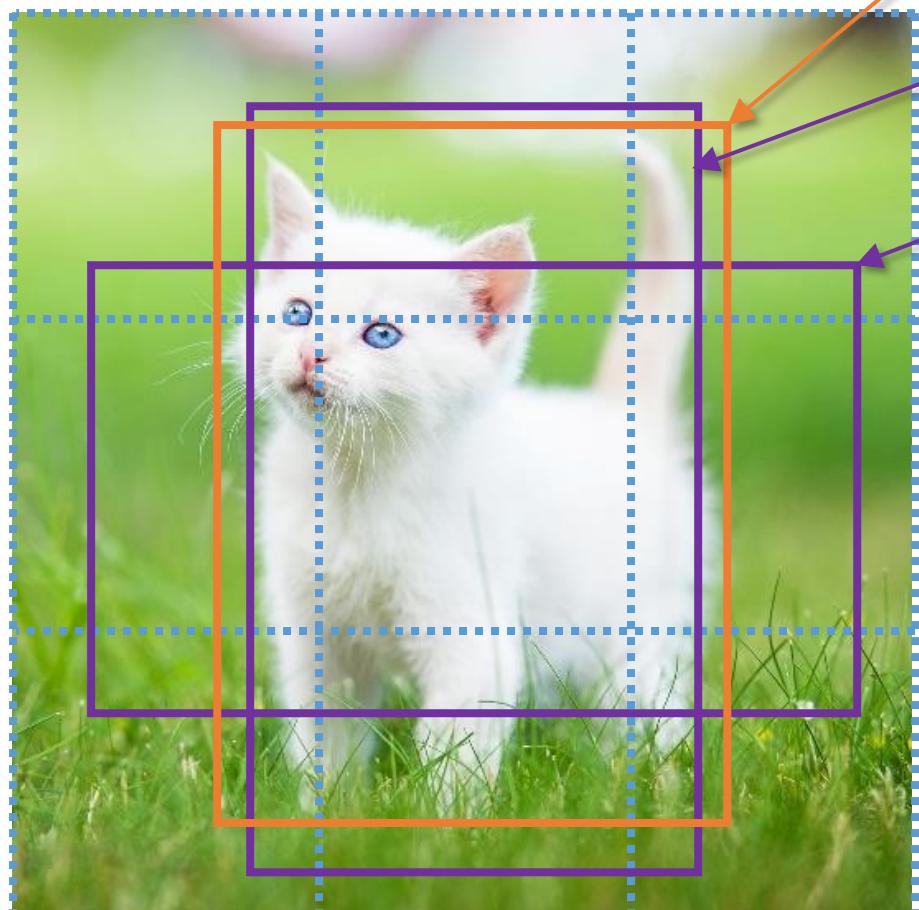


# Region proposal network (RPN)



# Predikce obdélníků: anchor boxes

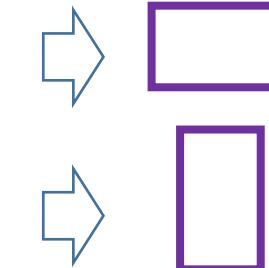
pro ilustraci obrázek rozdělen na  
3x3 “pixely”



anotovaný obdélník R1

anchor box B1

anchor box B2



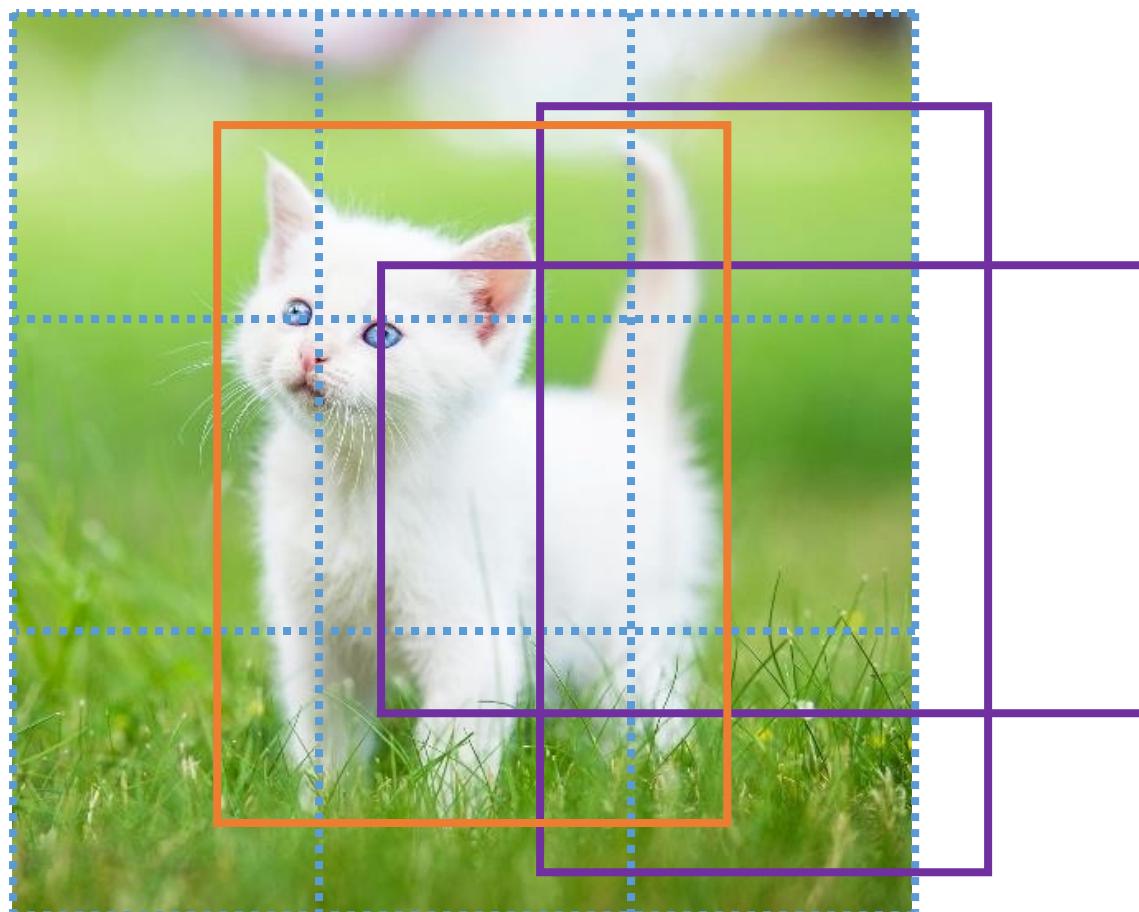
zde má každý “pixel” dva  
anchor boxy

sít’ se pro každý anchor box učí predikovat:

- $S_{\text{objekt}}, S_{\text{neobjekt}} \dots$  skóre / pr., zda box obsahuje objekt
- $b_x, b_y, b_w, b_h \dots$  čísla relativní vůči velikosti anchor boxu, která ho upravují na pozici a rozměry přiřazeného anotovaného obdélníku
- pokud se anchor box dostatečně nepřekrývá s žádným obdélníkem, pak se korekce souřadnic ignoruje

# Predikce obdélníků: anchor boxes

pro ilustraci obrázek rozdělen na  
3x3 “pixely”

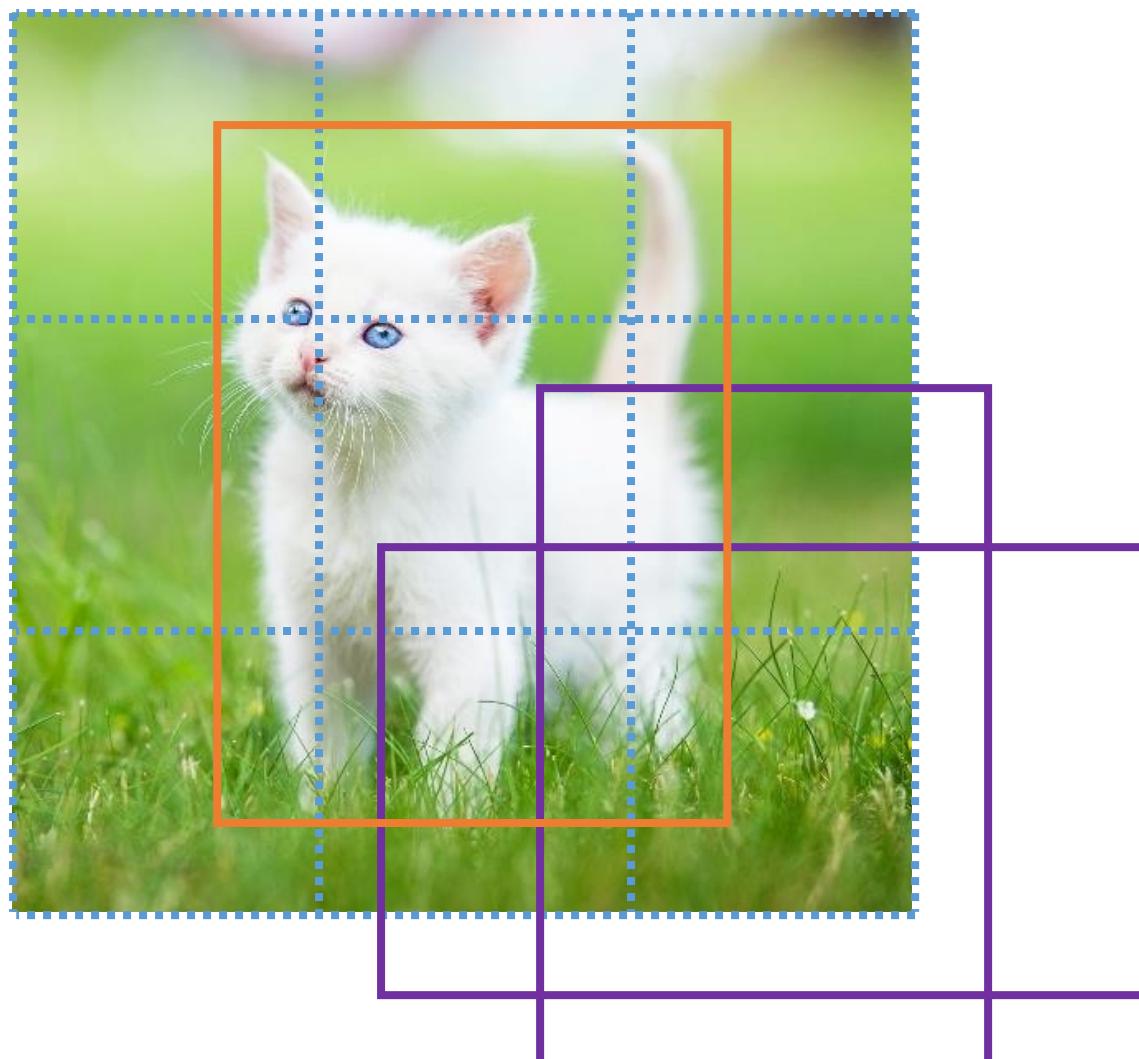


to samé pro všechny prohledávací pozice

ani jeden z anchor boxů se dostatečně  
nepřekrývá s anotovaným obdélníkem

# Predikce obdélníků: anchor boxes

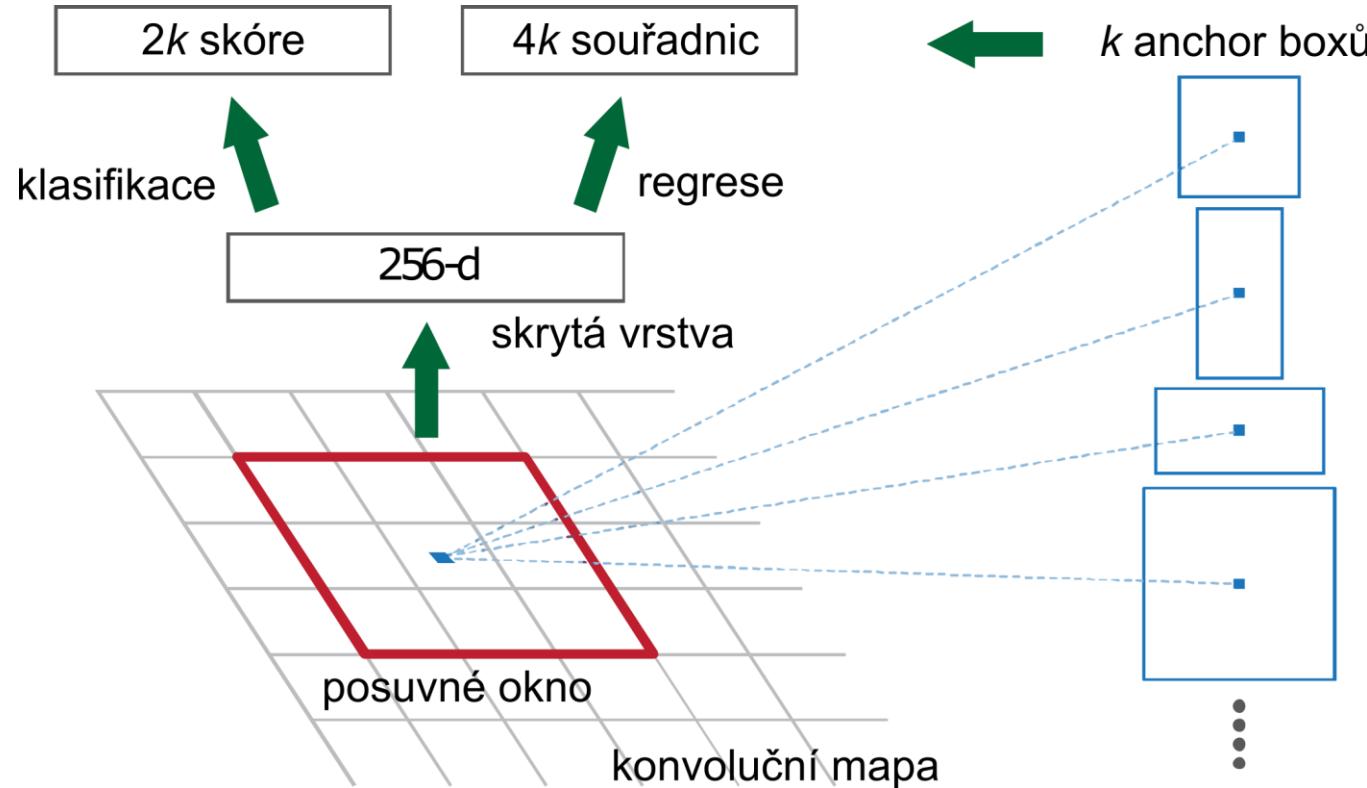
pro ilustraci obrázek rozdělen na  
3x3 “pixely”



to samé pro všechny prohledávací pozice

ani jeden z anchor boxů se dostatečně  
nepřekrývá s anotovaným obdélníkem

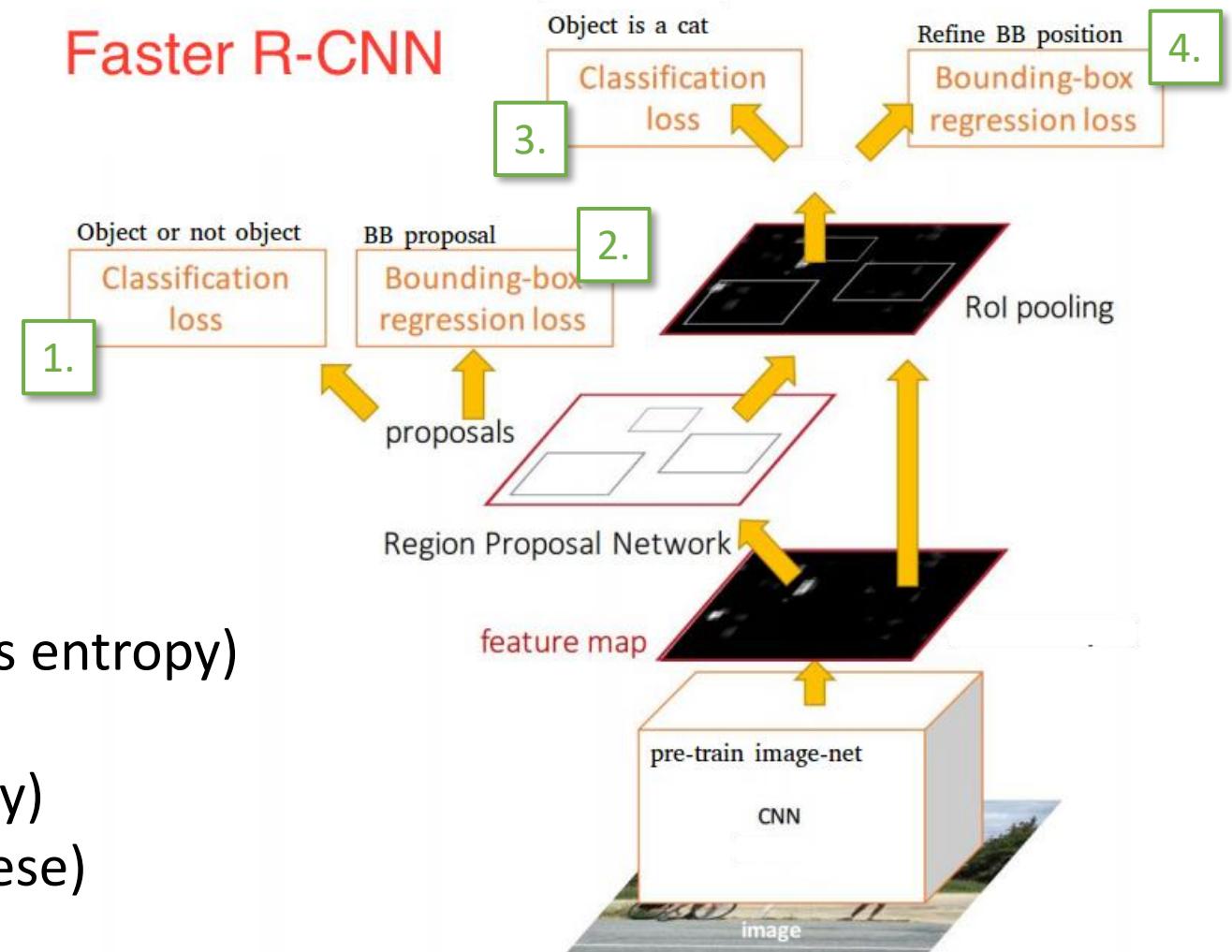
# Region proposal network (RPN)



jednomu pixelu odpovídá  $k$  anchor boxes (v orig. článku  $k = 9$ )

RPN predikuje pro každé z nich  
→ proto má  $2k$  klasifikačních a  
 $4k$  regresních výstupů

# Trénování Faster R-CNN



Celkový loss je suma 4 dílčích:

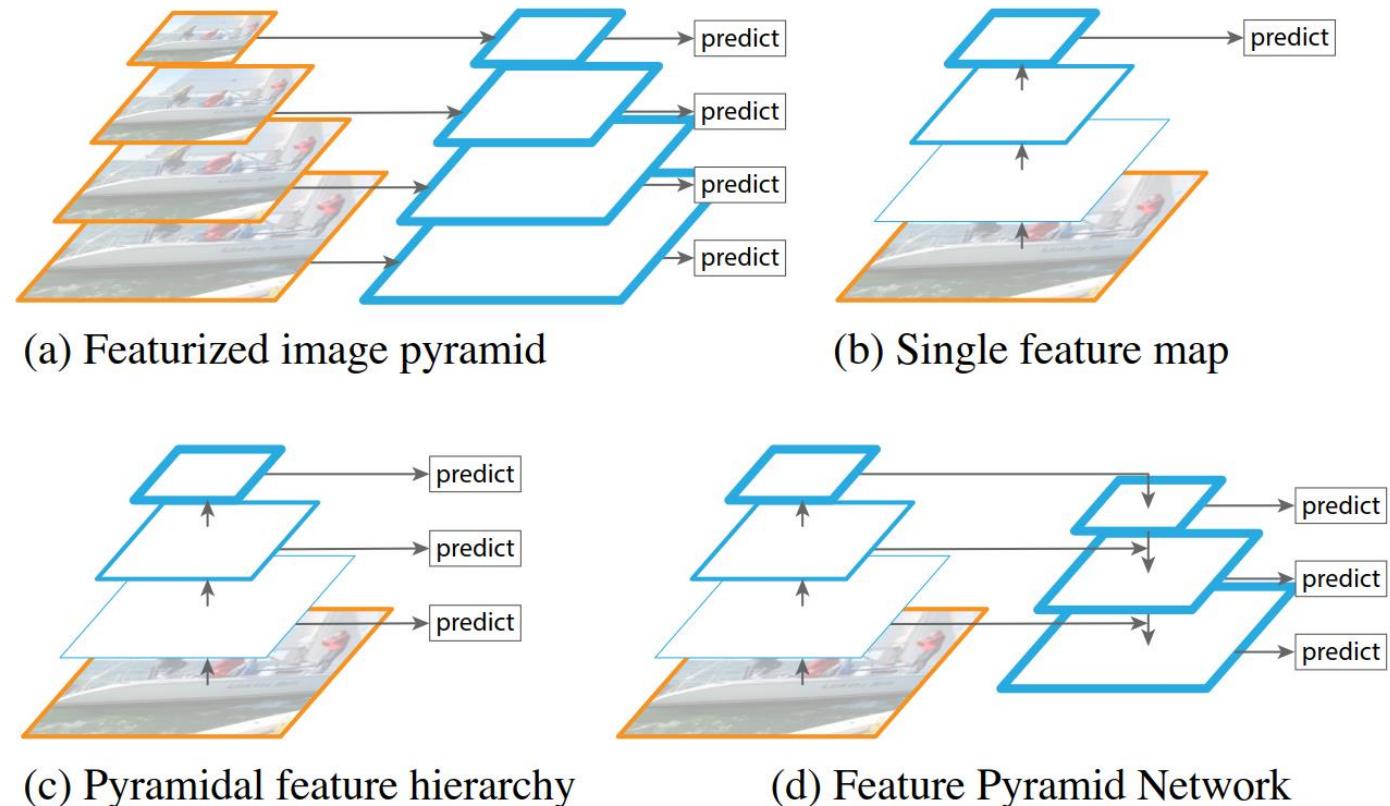
1. RPN objekt/neobjekt (softmax cross entropy)
2. RPN obdélník (Smooth L1 regrese)
3. třída objektu (softmax cross entropy)
4. korekce obdélníka (Smooth L1 regrese)

# R-CNN vs Fast R-CNN vs Faster R-CNN

	R-CNN	Fast R-CNN	Faster R-CNN
Test průchod	49.0 s	2.32 s	0.2 s
Zrychlení testovacího průchodu	1x	21x	245x
mAP (přesnost)	66.0 %	66.9 %	66.9 %

# Feature Pyramid Network (FPN)

- Jedním ze zdrojů chyb variabilní velikosti objektů
- Ke konvoluční síti extrahující příznaky (tzv. backbone síť) se připojí paralelní síť s opačným tokem odshora dolů
- Predikce (RPN, boxy, klasifikace) se provádějí na všech vrstvách FPN
- Např. RPN má na každé vrstvě FPN anchor boxy jen s jednou velikostí
- Lze použít s libovolným detektorem
- Zlepšuje skóre o několik procent



# Detekce jako regrese

---

# Detekce jako regrese: You only look once (YOLO)

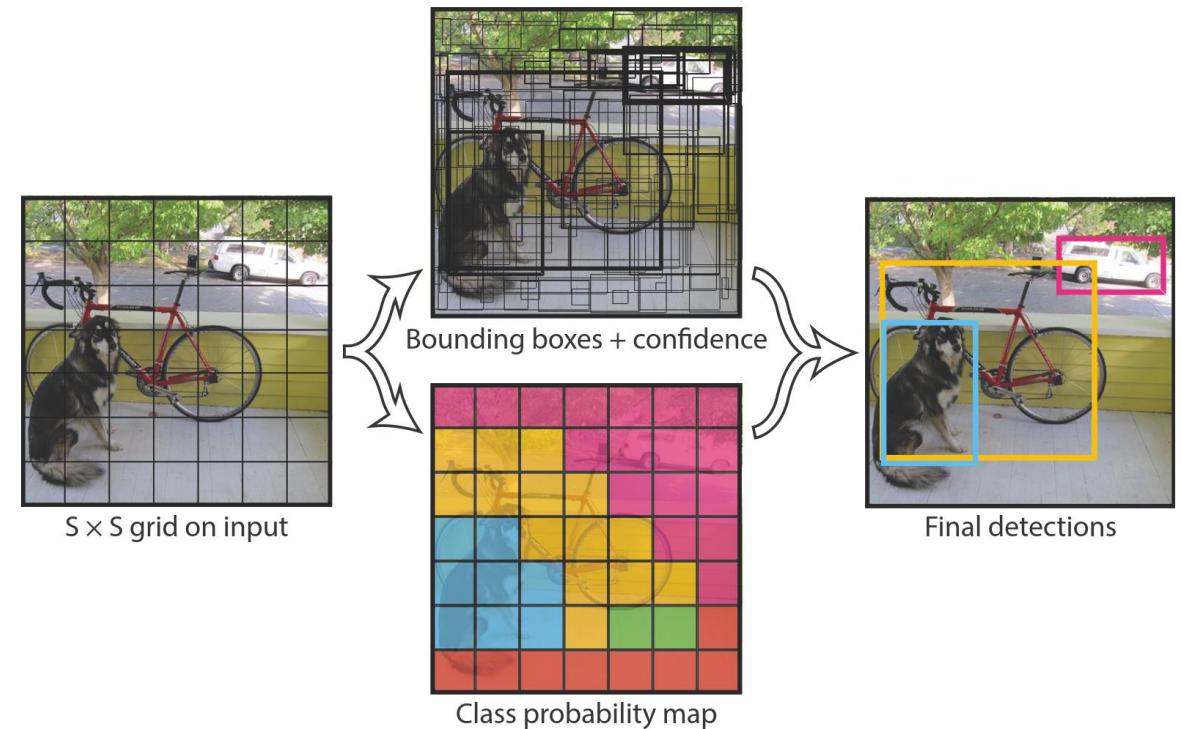
- podobné jako region proposal network (RPN) u Faster R-CNN, ovšem zde rovnou i s predikcí třídy
- obrázek se rozdělí na  $7 \times 7$  buněk
- v každé buňce predikce 2 anchor boxů

**každý anchor je  $4 + 1 + C$  čísel:**

- $p_{\text{obj}}$  ... pr., zda box obsahuje objekt
- $b_x, b_y, b_w, b_h$  ... čísla relativní vůči velikosti anchor boxu, která ho upravují na pozici a rozměry přiřazeného anotovaného obdélníka
- $p_0, p_1, \dots, p_C$  ... pravděpodobnosti všech tříd

**dohromady tedy výstup ze sítě má rozměr**

$$7 \times 7 \times (2 \cdot 5 + C)$$



**Figure 2: The Model.** Our system models detection as a regression problem. It divides the image into an  $S \times S$  grid and for each grid cell predicts  $B$  bounding boxes, confidence for those boxes, and  $C$  class probabilities. These predictions are encoded as an  $S \times S \times (B * 5 + C)$  tensor.

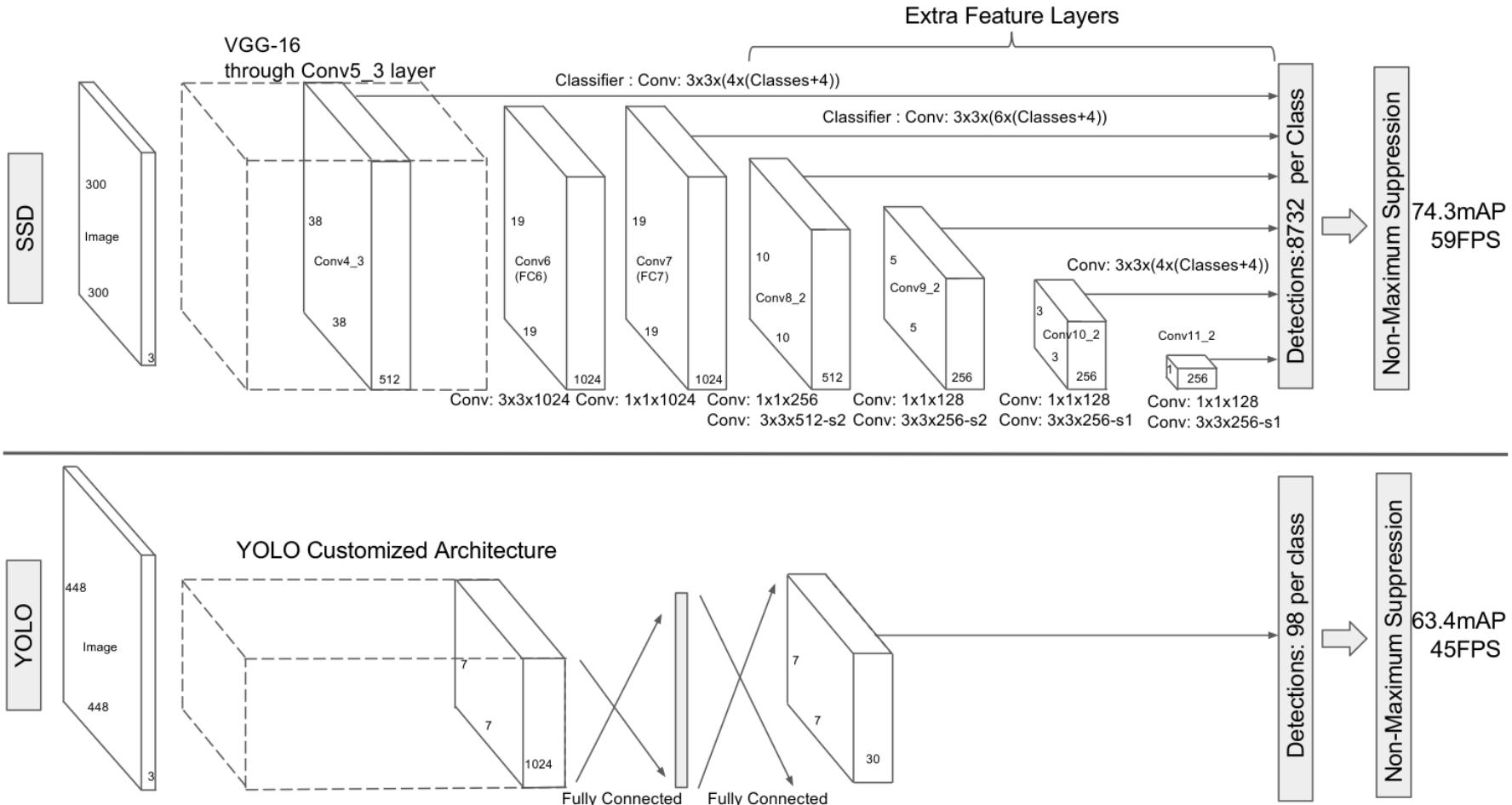
# Detekce jako regrese: You only look once (YOLO)

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	<b>155</b>
YOLO	2007+2012	<b>63.4</b>	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

**Table 1: Real-Time Systems on PASCAL VOC 2007.** Comparing the performance and speed of fast detectors. Fast YOLO is the fastest detector on record for PASCAL VOC detection and is still twice as accurate as any other real-time detector. YOLO is 10 mAP more accurate than the fast version while still well above real-time in speed.

# Detekce jako regrese: Single shot detection (SSD)

velmi podobné jako YOLO, predikuje boxy na základě více vrstev, ne pouze poslední → lépe řeší měřítka



# Detekce jako regrese: RetinaNet

- SSD detektory klasifikují každou oblast obrázku → tzv. dense predictions (hustá mapa)
- To vede na velkou nevyváženosť objektov a pozadí (class imbalance)
- Lin et al. to řeší úpravou cross entropy lossu → **focal loss**
- Idea je položit väčšiu váhu na těžko klasifikovateľné obasti
- Trochu podobné SVM, kde body "mimo pásmo" nepriispívajú do lossu
- Od svojho uvedenia veľmi populárni, zlepšuje skóre SSD-like detektorov

[Lin et al: "Focal Loss for Dense Object Detection"](#)

# RetinaNet: focal loss

- Standardní cross entropy je

$$\text{CE}(p, y) = \begin{cases} -\log(p) & \text{if } y = 1 \\ -\log(1 - p) & \text{otherwise.} \end{cases}$$

- Zavedeme

$$p_t = \begin{cases} p & \text{if } y = 1 \\ 1 - p & \text{otherwise,} \end{cases}$$

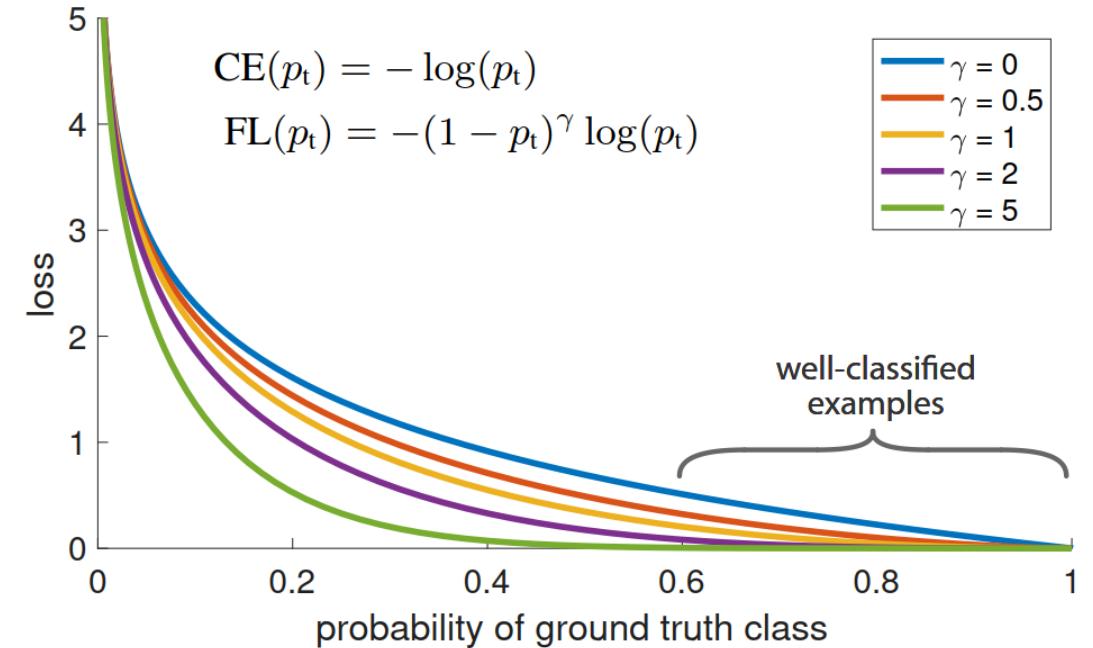
- Výsledný focal loss

$$\text{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t)$$

$\gamma$  ... hyperparametr, v praxi se používá  $\gamma = 2$

$\alpha_t$  ... pro balancování tříd

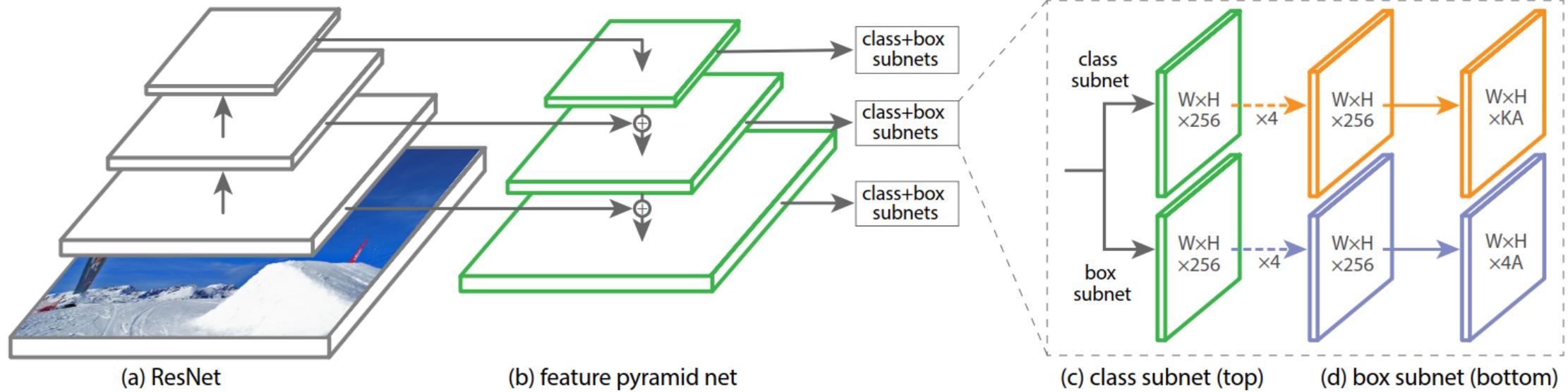
cross entropy vs focal loss



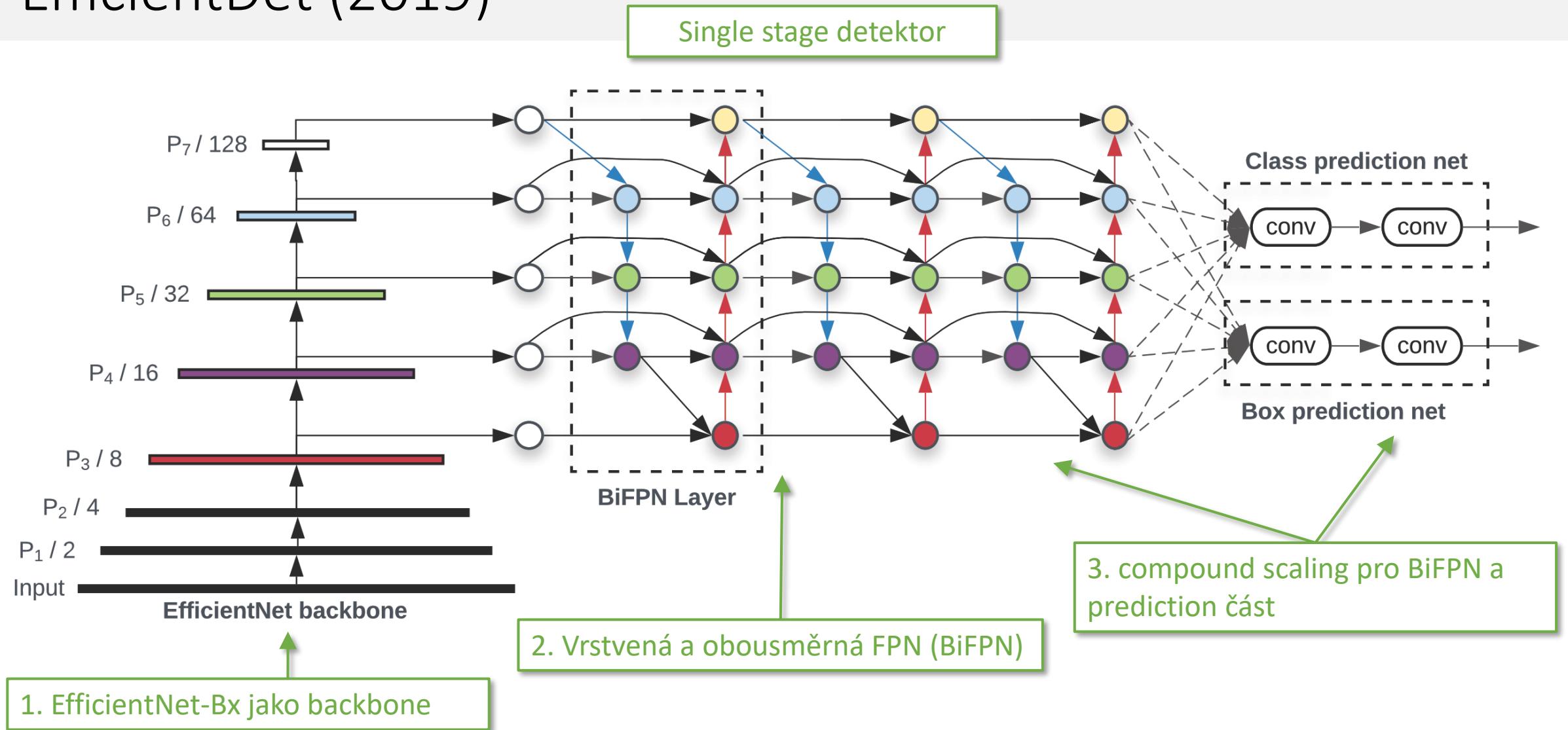
standardní cross entropy ( $\gamma = 0$ , modrá křivka) má velkou hodnotu lossu i např. pro  $p_t = 0.8$

větší hodnota hyperparametru  $\gamma \rightarrow$  menší loss pro "sebevědomé" predikce s velkou  $p_t$  (easy examples)

# RetinaNet: architektura



# EfficientDet (2019)



Tan et al: “EfficientDet: Scalable and Efficient Object Detection”

# EfficientDet (2019)

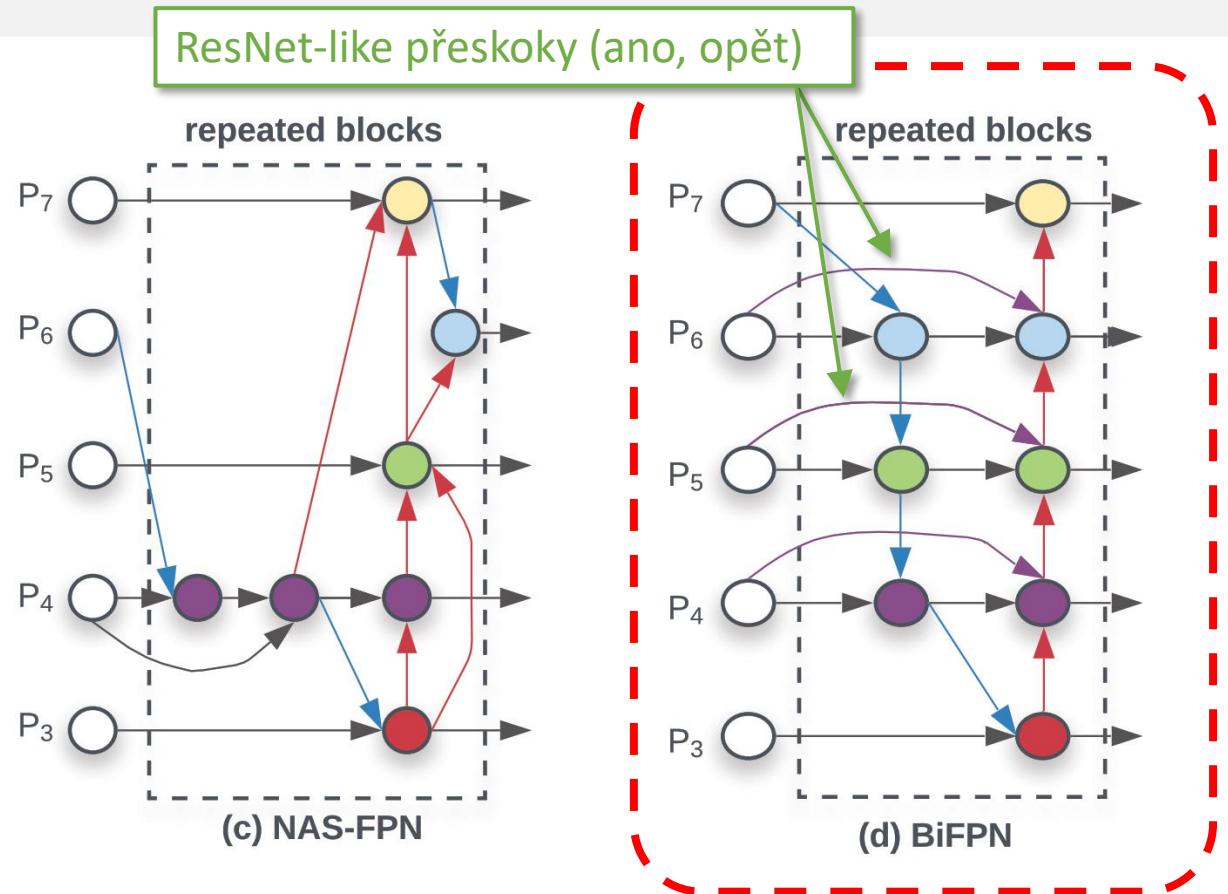
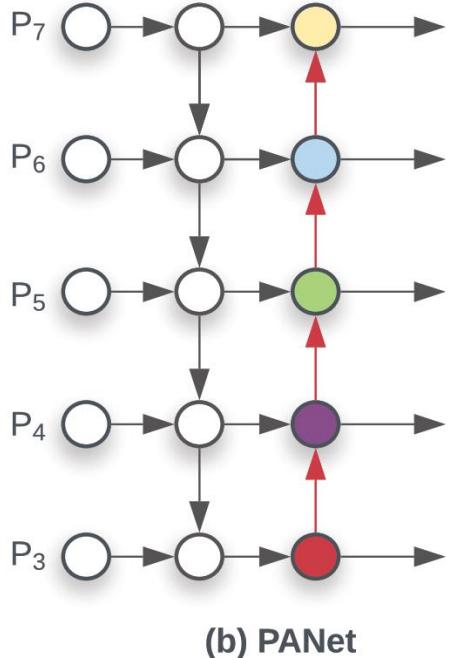
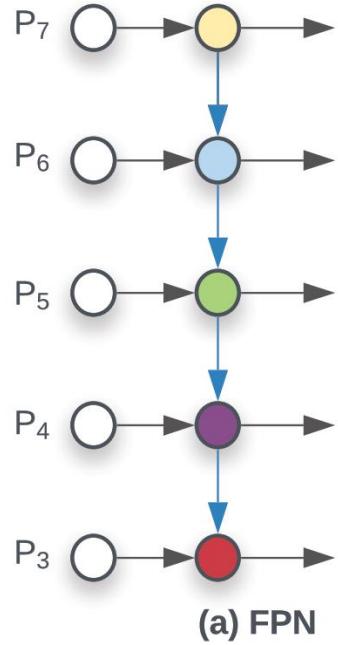


Figure 2: **Feature network design** – (a) FPN [23] introduces a top-down pathway to fuse multi-scale features from level 3 to 7 ( $P_3 - P_7$ ); (b) PANet [26] adds an additional bottom-up pathway on top of FPN; (c) NAS-FPN [10] use neural architecture search to find an irregular feature network topology and then repeatedly apply the same block; (d) is our BiFPN with better accuracy and efficiency trade-offs.

# EfficientDet (2019)

	AP	Parameters	FLOPs
ResNet50 + FPN	37.0	34M	97B
<b>EfficientNet-B3 + FPN</b>	40.3	21M	75B
<b>EfficientNet-B3 + BiFPN</b>	44.4	12M	24B

Table 4: **Disentangling backbone and BiFPN** – Starting from the standard RetinaNet (ResNet50+FPN), we first replace the backbone with EfficientNet-B3, and then replace the baseline FPN with our proposed BiFPN.

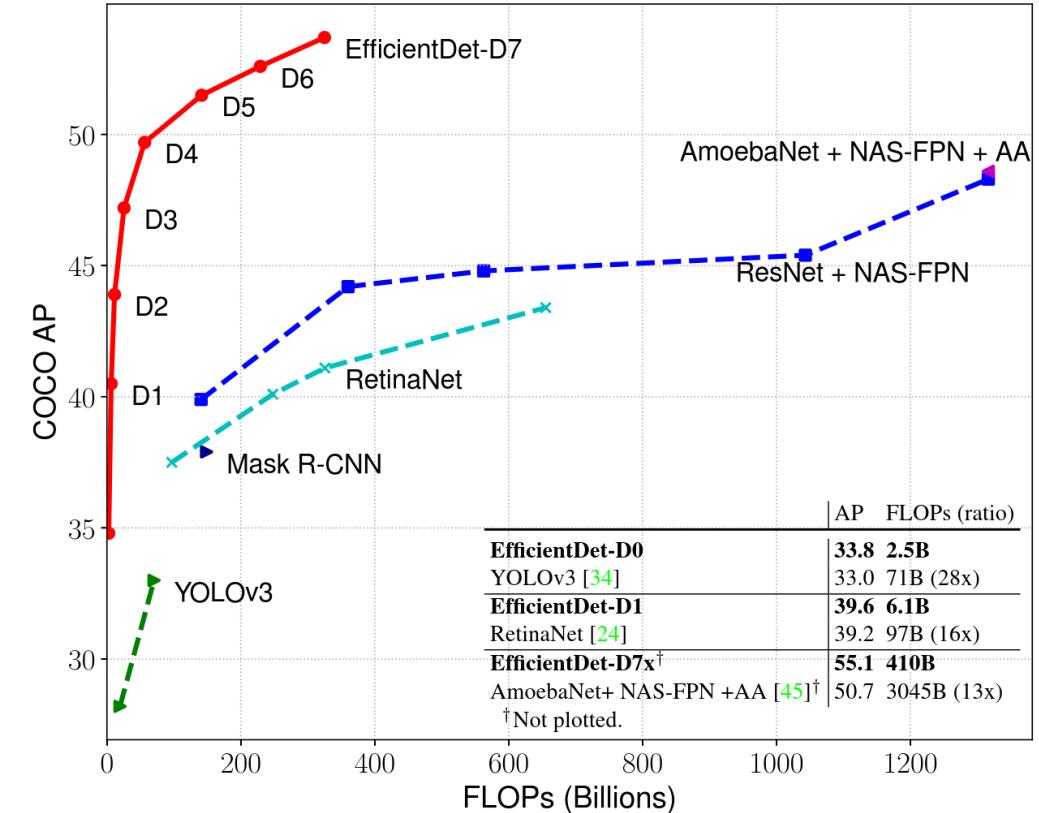


Figure 1: **Model FLOPs vs. COCO accuracy** – All numbers are for single-model single-scale. Our EfficientDet achieves new state-of-the-art 55.1% COCO AP with much fewer parameters and FLOPs than previous detectors. More studies on different backbones and FPN/NAS-FPN/BiFPN are in Table 4 and 5. Complete results are in Table 2.

# YOLOv4 (2020)

- Lepší architektura
  - CSPDarkNet53 (Cross-Stage Partial spoje), Mish aktivace, Vážené sčítání residuálních spojů
- Integrace feature map z různých rozlišení
  - Spatial Pyramid Pooling, Path Aggregation Network
- Regularizace
  - DropBlock, Cross-minibatch Batch Normalizace, CloU loss
- Augmentace
  - CutMix, Mosaic, Self-Adversarial Training
- A další triky
  - ...

[YOLOv4: Optimal Speed and Accuracy of Object Detection](#)

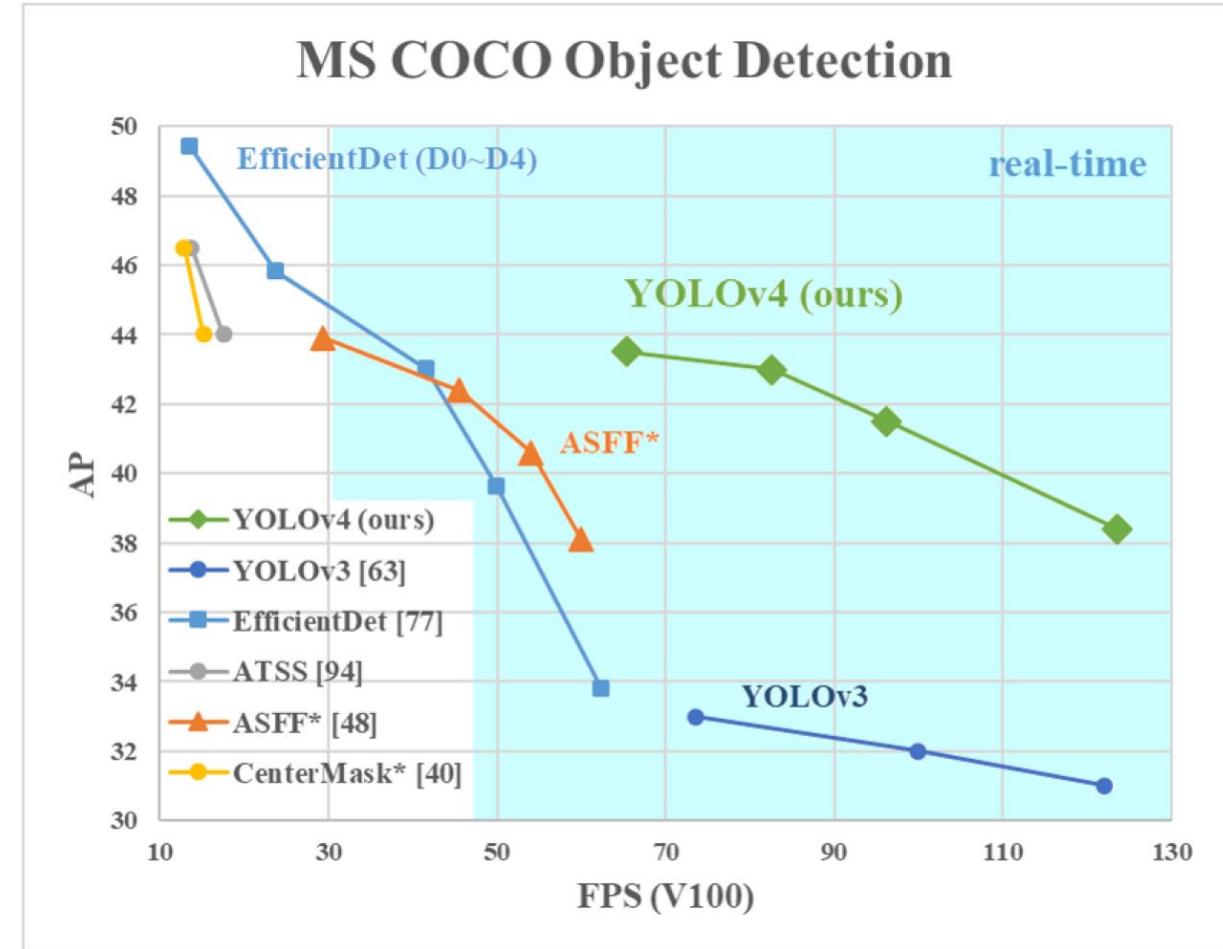
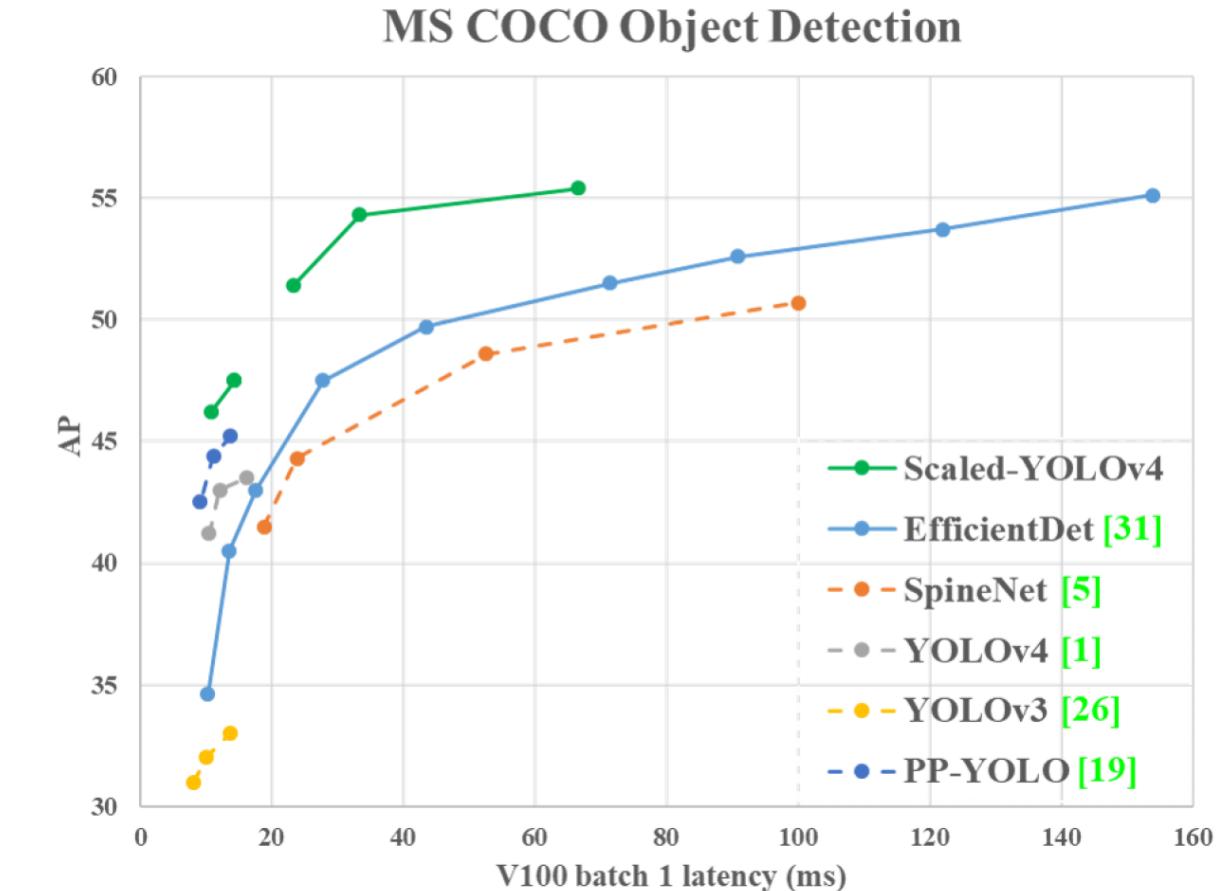


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively.

# Scaled YOLOv4 (2020)

- Cross-Stage Partial spoje úplně všude, nejen backbone
- Kombinují některé věci s EfficientDet
- Různé backbone pro různé cílové náročnosti
  - YOLOv4-tiny
  - YOLOv4-large
  - YOLOv4-Px ( $x=5,6,7$ )



[Scaled-YOLOv4: Scaling Cross Stage Partial Network](#)

Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. The dashed line means only latency of model inference, while the solid line include model inference and post-processing.

# YOLOv7 (2022)

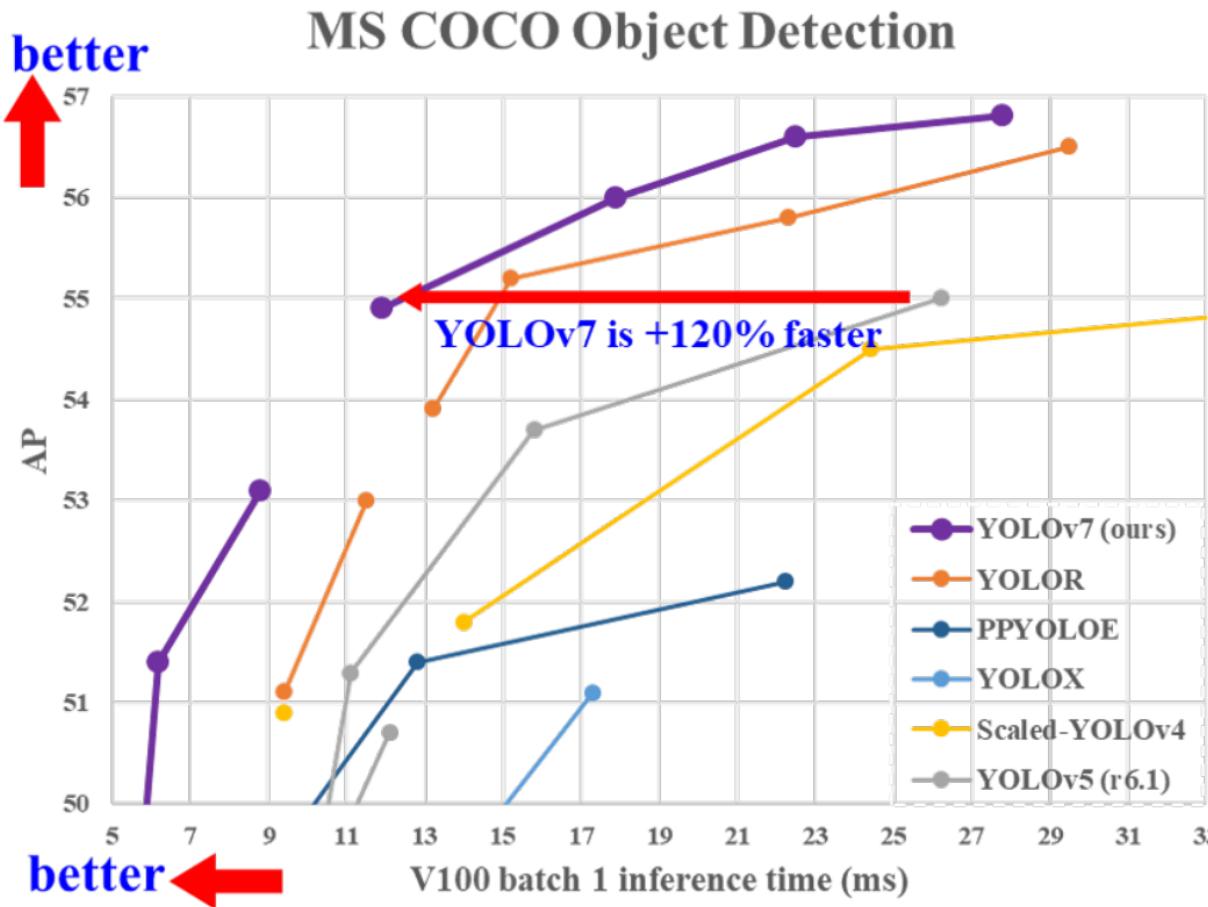


Figure 1: Comparison with other real-time object detectors, our proposed methods achieve state-of-the-arts performance.

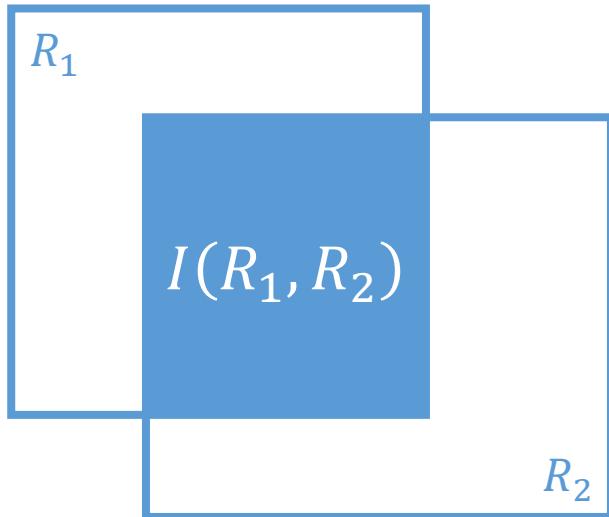
[YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors](#)

## Vyhodnocení detektorů

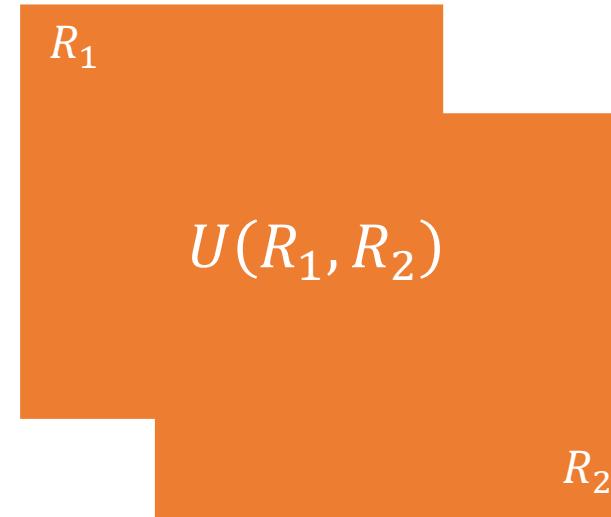
---

# Překryv obdélníků: intersection over union (IoU)

průnik (intersection)



sjednocení (union)



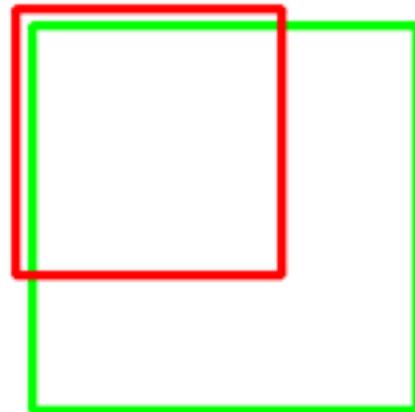
$$IoU(R_1, R_2) = \frac{\text{průnik}}{\text{sjednocení}}$$

- IoU se někdy také nazývá Jaccard index

# Překryv obdélníků: intersection over union (IoU)

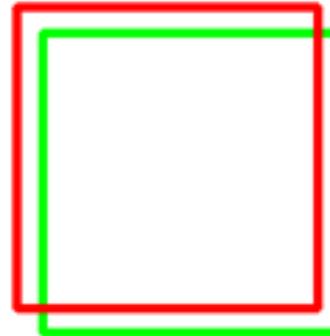
pro účely vyhodnocení se za úspěšně nalezený objekt typicky považuje  $IoU \geq 0.5$

$IoU: 0.4034$



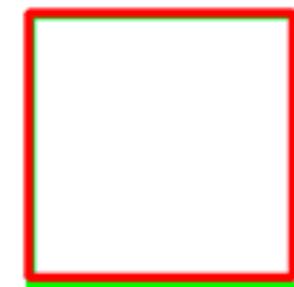
**Poor**

$IoU: 0.7330$



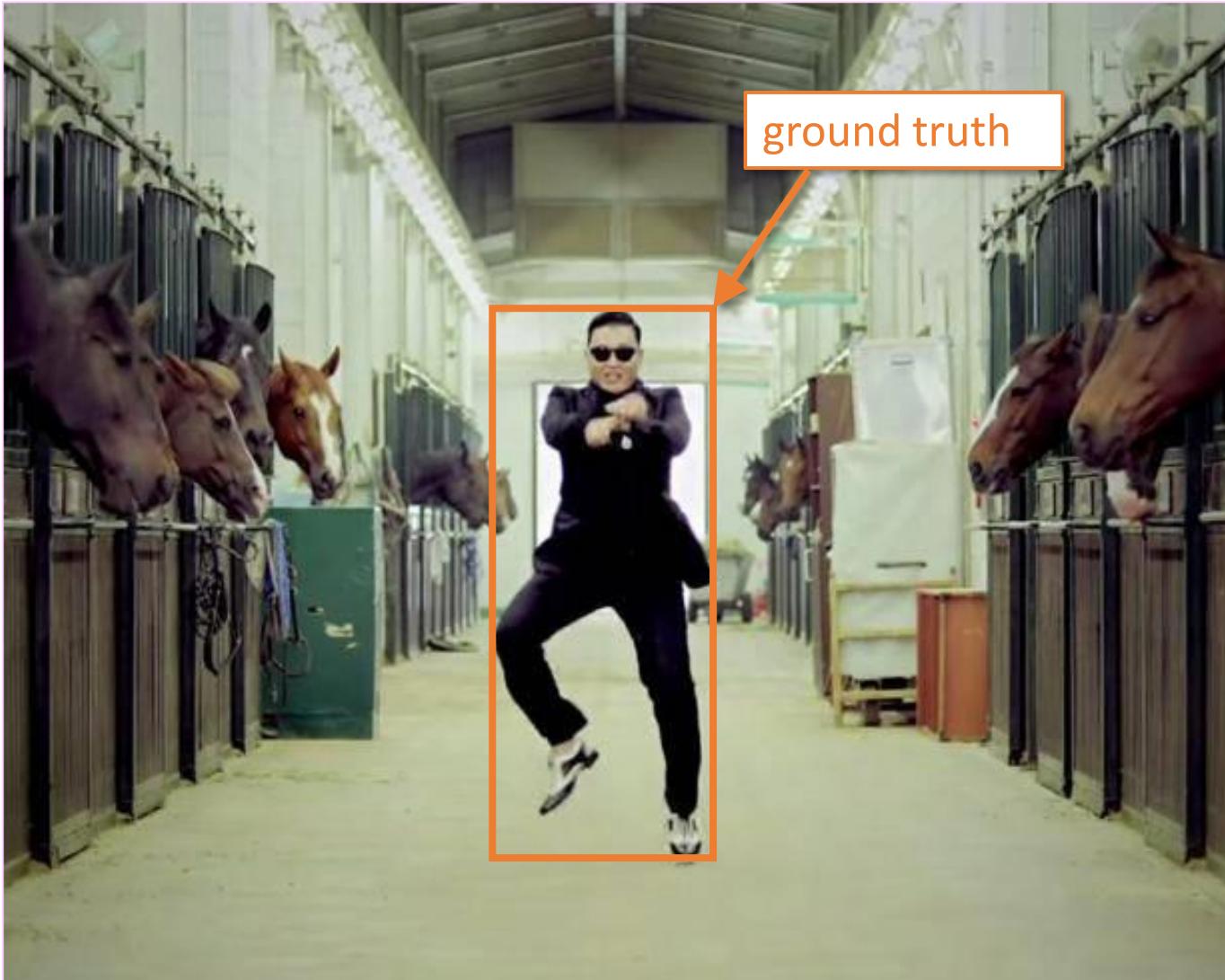
**Good**

$IoU: 0.9264$

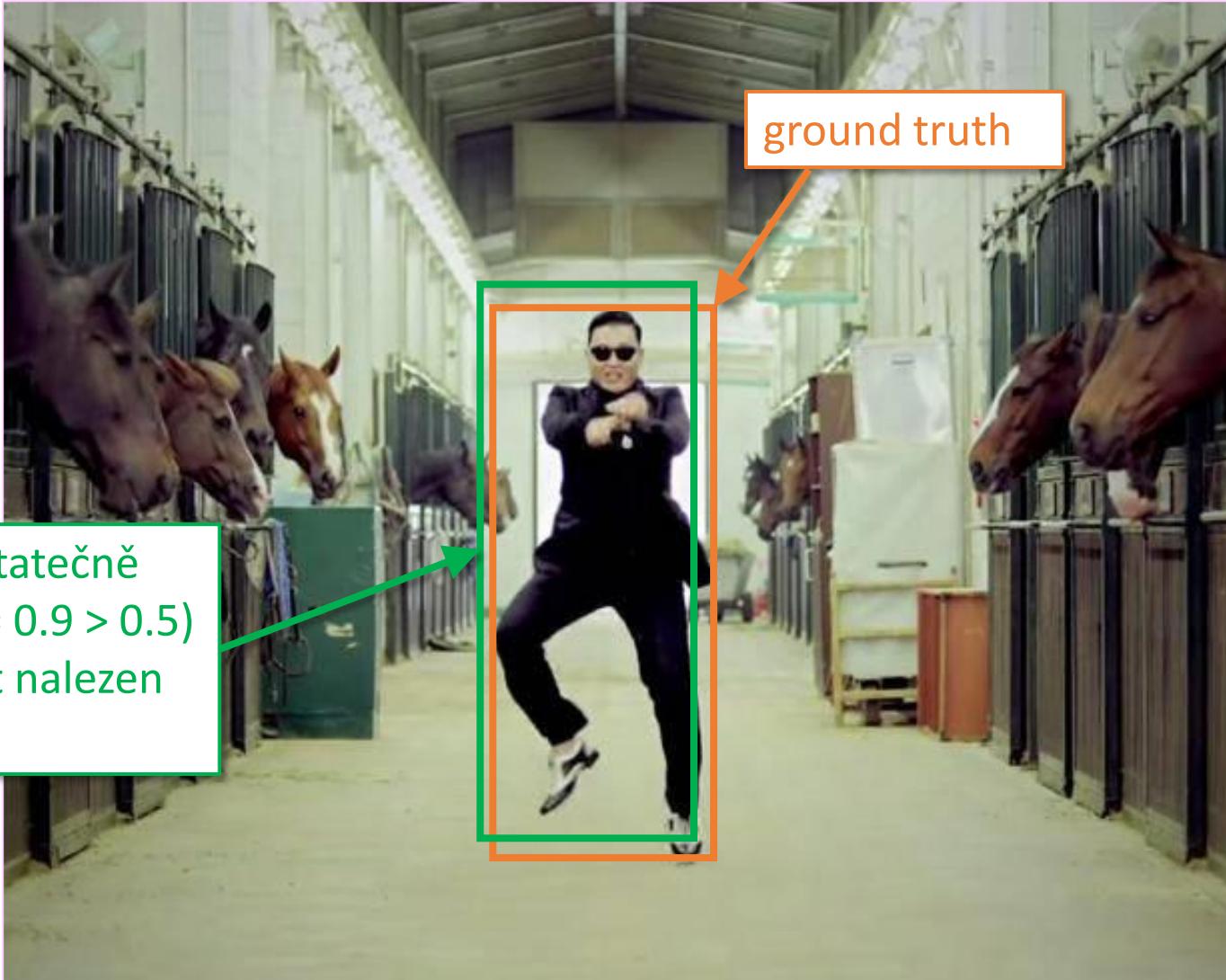


**Excellent**

# True positives (TP) a false positives (FP)

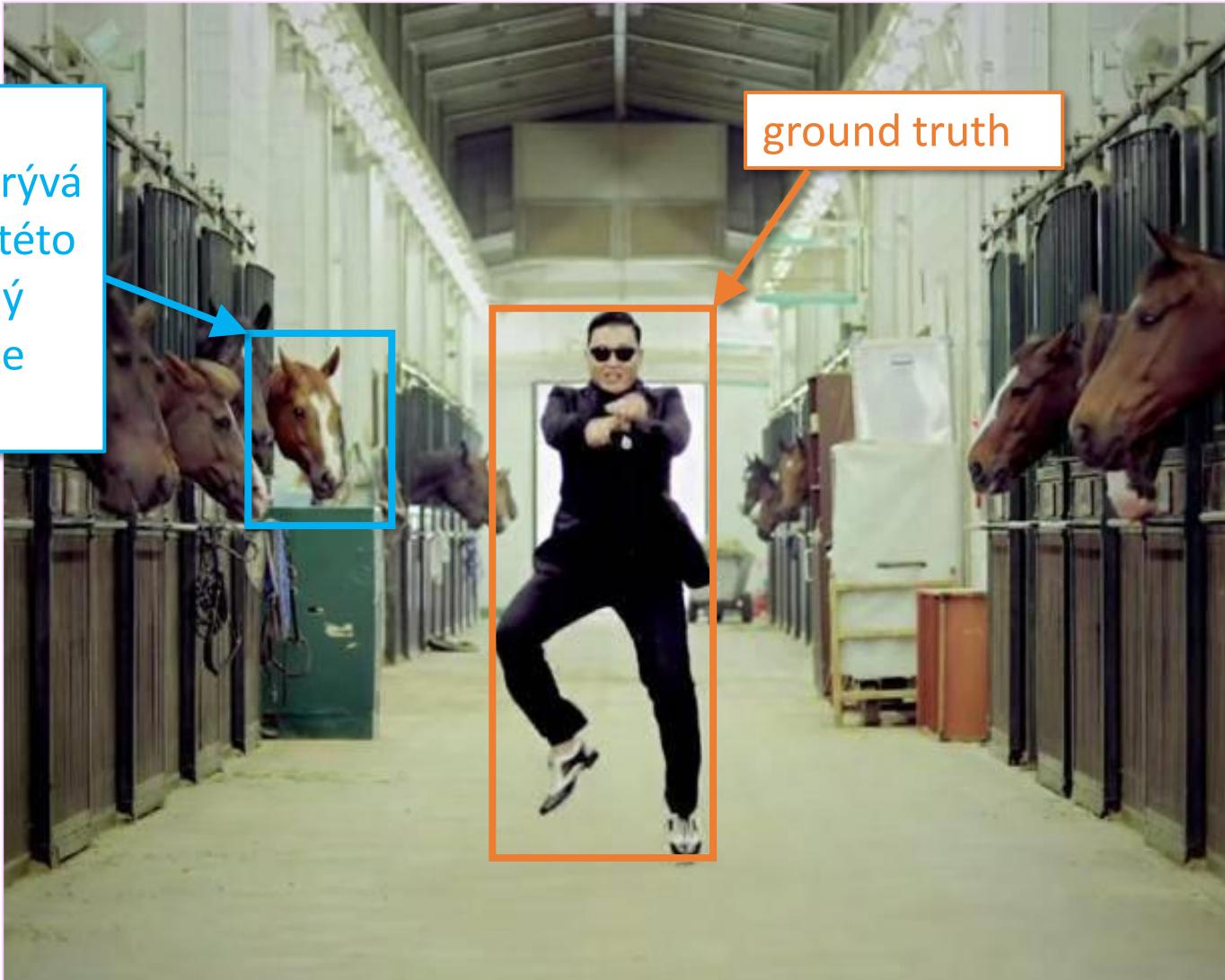


# True positives (TP) a false positives (FP)

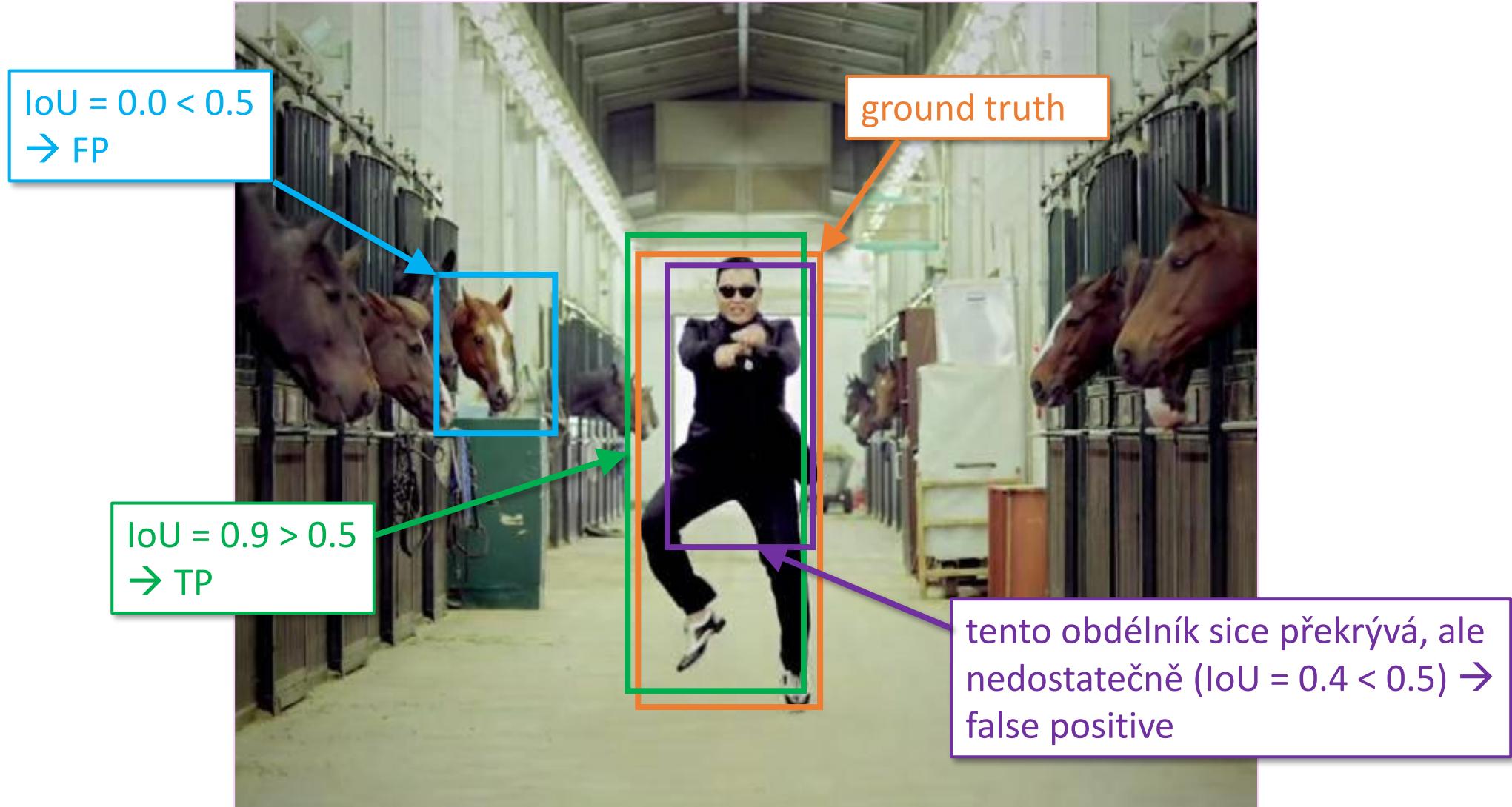


# True positives (TP) a false positives (FP)

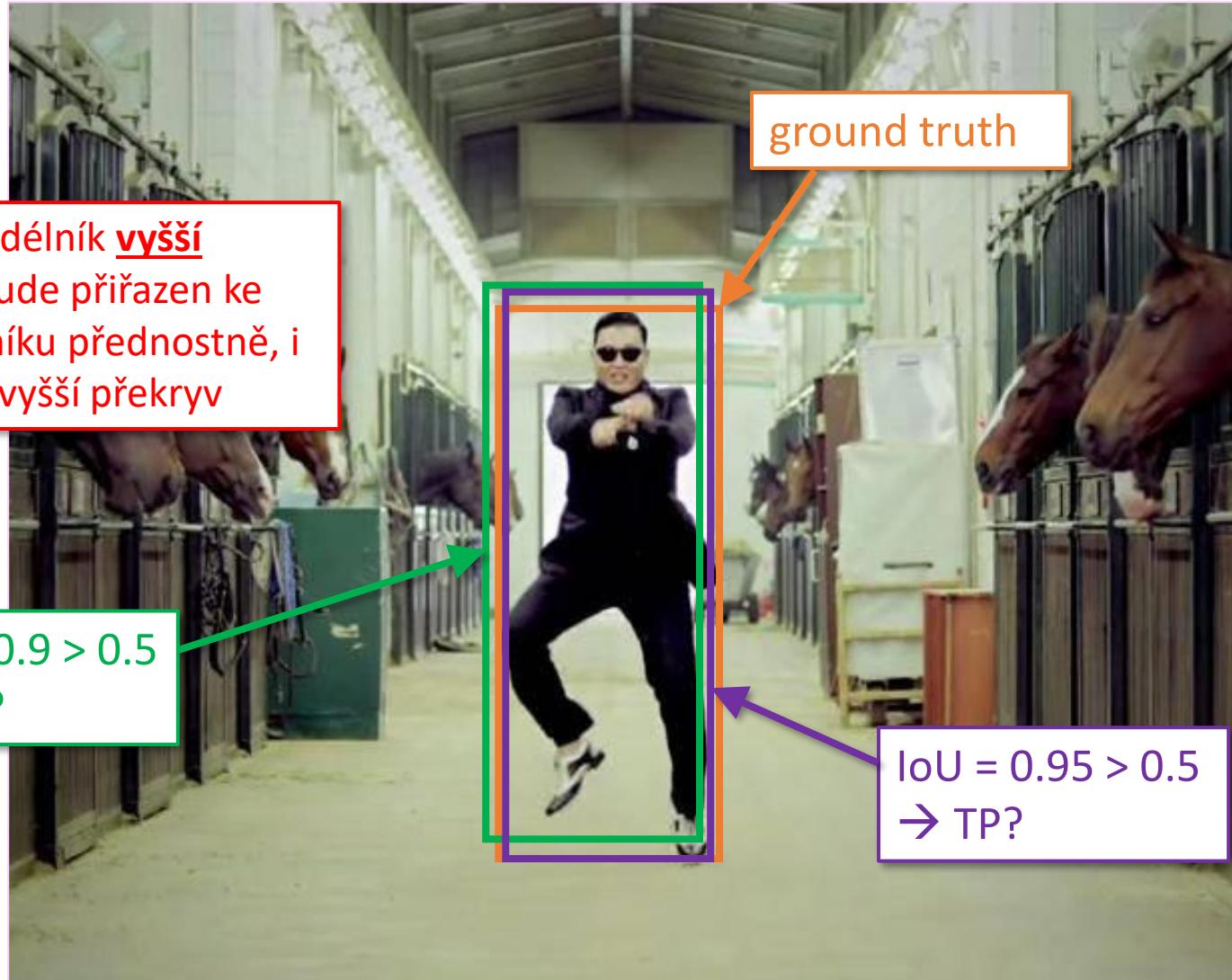
nalezen obdélník, ale  
anotaci vůbec nepřekrývá  
(IoU =  $0.0 < 0.5$ ) = na této  
pozici neměl být žádný  
objekt nalezen → false  
positive



# True positives (TP) a false positives (FP)



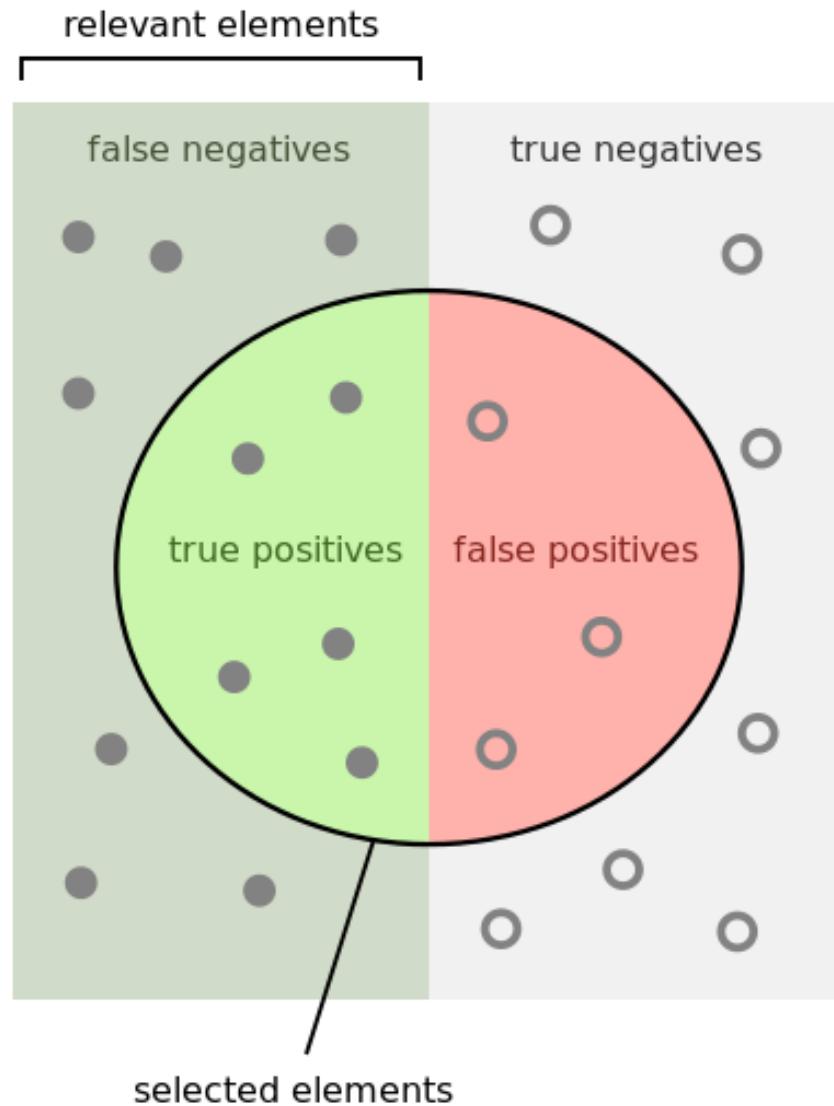
# Více obdélníků na jeden objekt: hladové vyhodnocení



# Více obdélníků na jeden objekt: hladové vyhodnocení



# Precision a recall



How many selected items are relevant?

$$\text{Precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

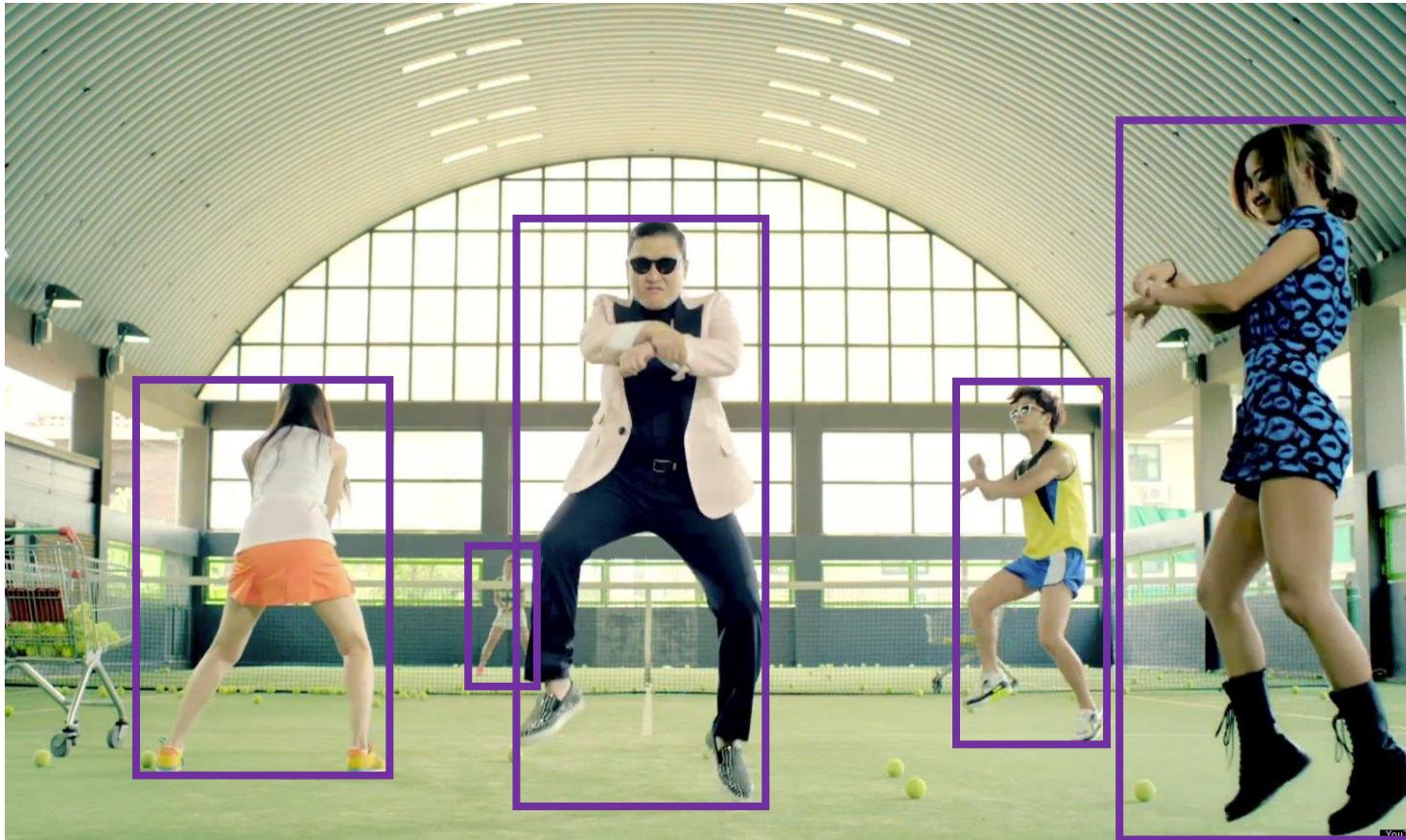
How many relevant items are selected?

$$\text{Recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

detektor nalezl 9 objektů, ale pouze 7 z nich bylo správně, ostatní byly falešné detekce → precision = 7/9 = 78 %

detektor měl najít najít 10 objektů, ale správně jich nalezl pouze 7 → recall = 70%

# Precision a recall: příklad



**ground truth**  
celkem 5 osob

# Precision a recall: příklad

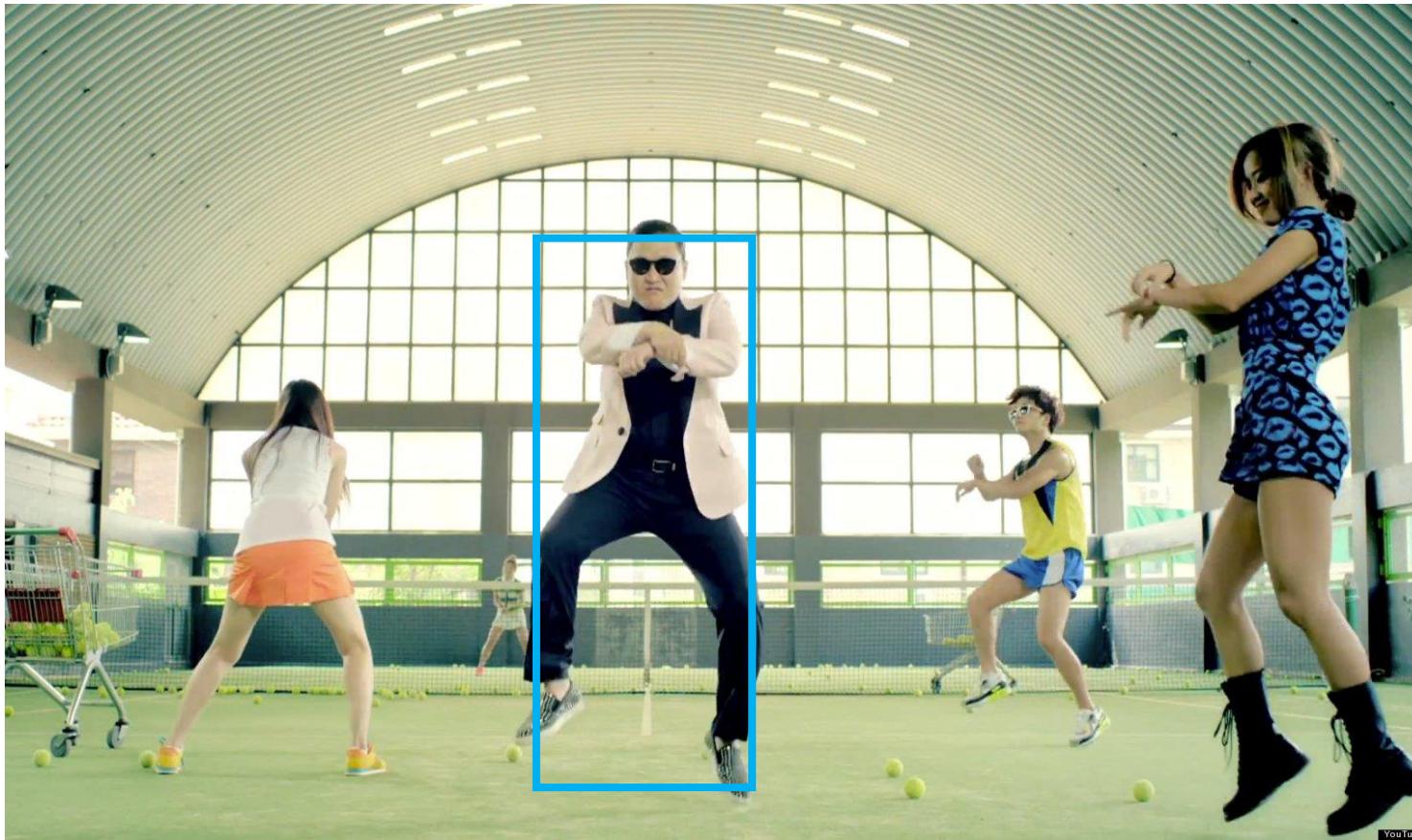


**ground truth**  
celkem 5 osob

pouze detekce s  $P(\text{osoba}) = 1.0$

0 detekovaných osob  
 $\text{recall} = 0/5 = 0\%$   
 $\text{precision} = 0/0 = 100\%$

# Precision a recall: příklad

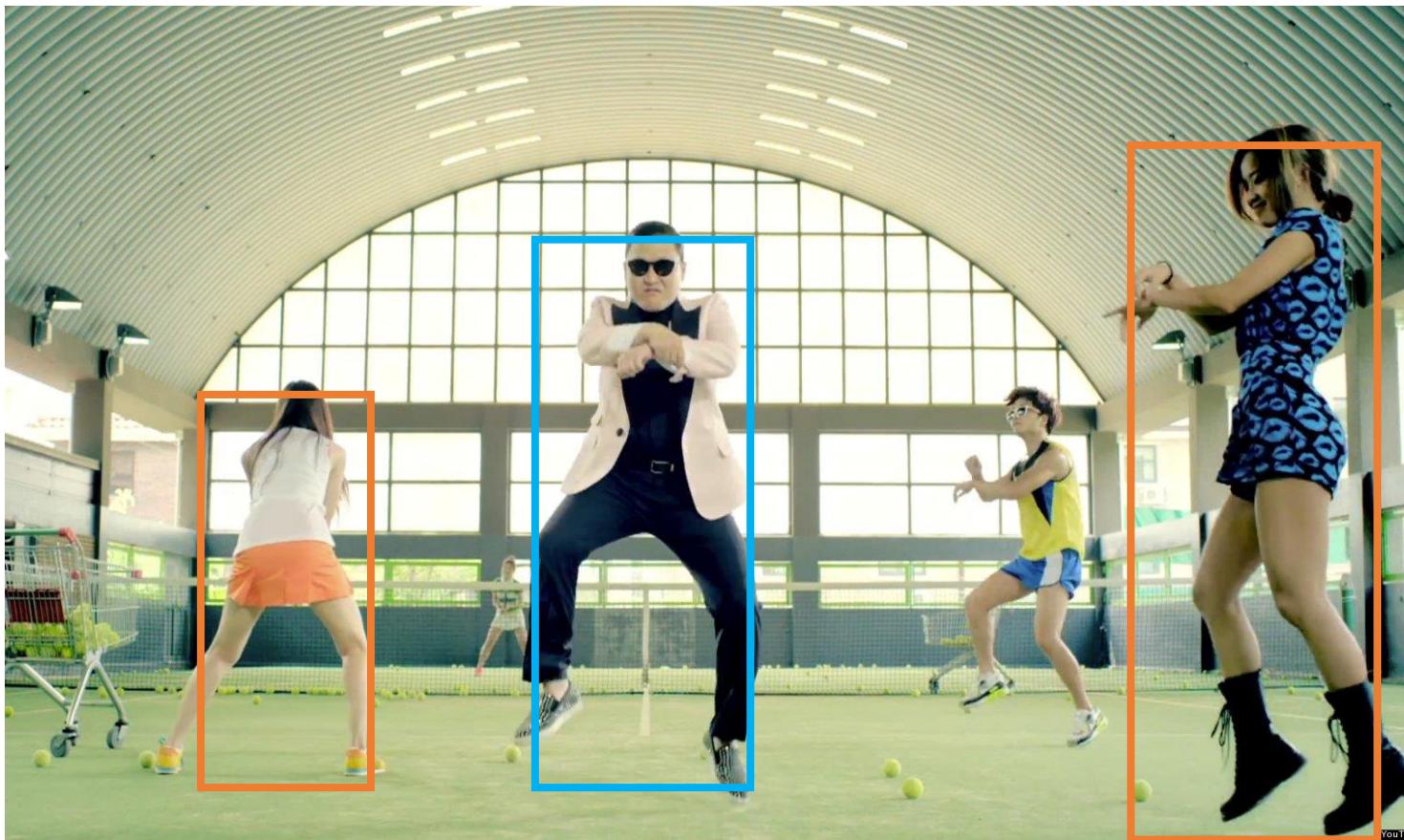


**ground truth**  
celkem 5 osob

pouze detekce s  $P(\text{osoba}) > 0.9$

1 detekovaná osoba  
 $\text{recall} = 1/5 = 20\%$   
 $\text{precision} = 1/1 = 100\%$

# Precision a recall: příklad

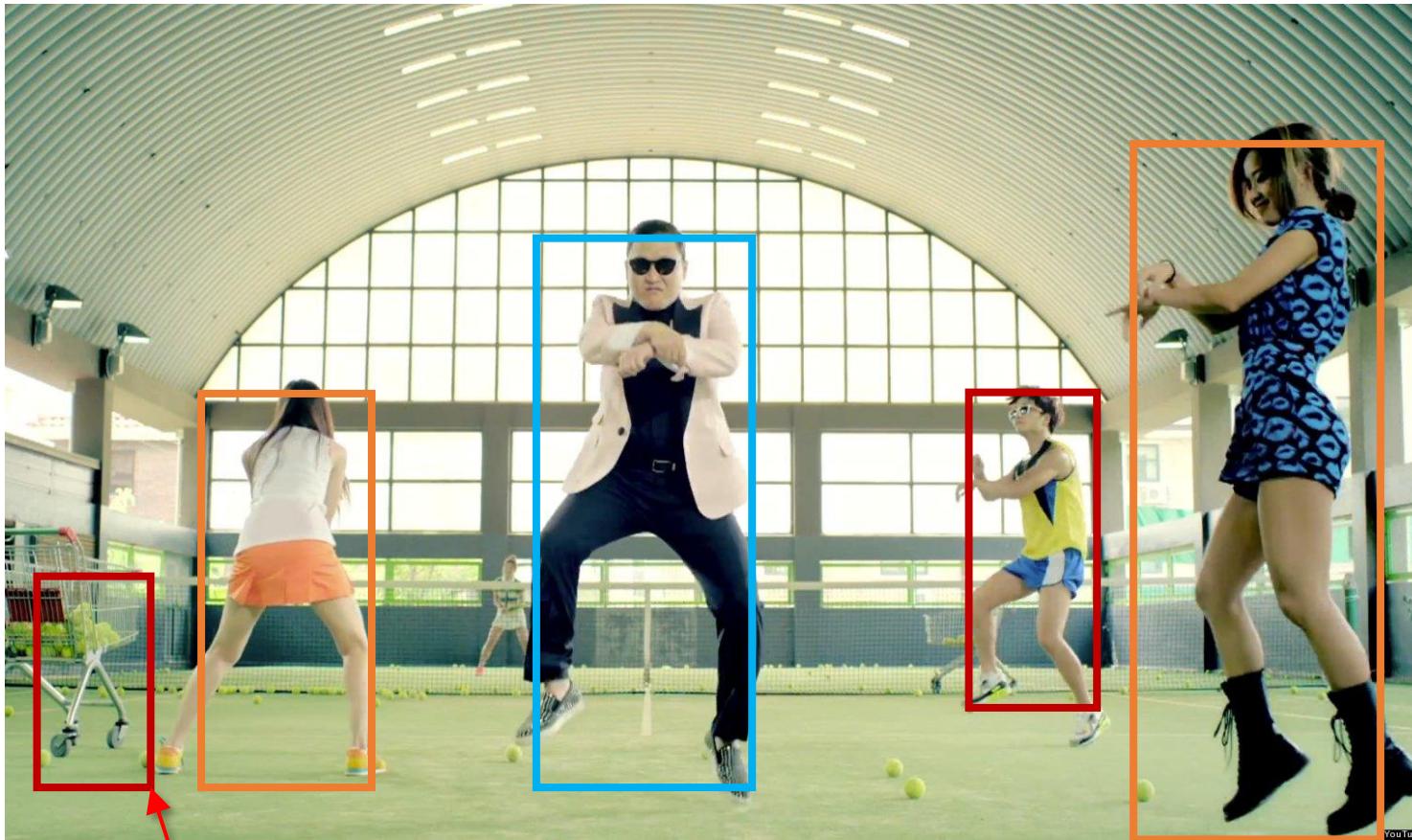


**ground truth**  
celkem 5 osob

pouze detekce s  $P(\text{osoba}) > 0.8$

3 detekované osoby  
 $\text{recall} = 3/5 = 60\%$   
 $\text{precision} = 3/3 = 100\%$

# Precision a recall: příklad



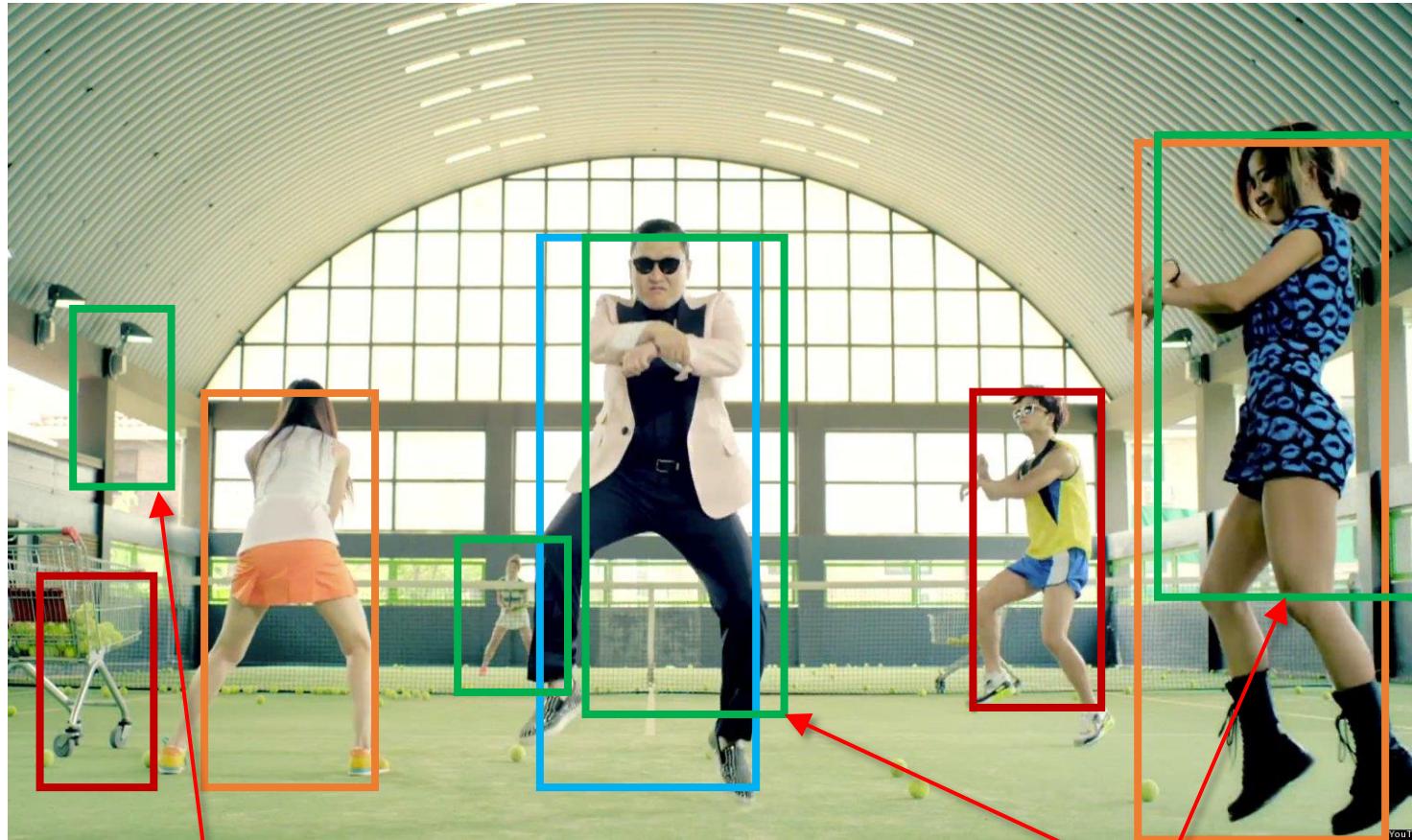
false positive

**ground truth**  
celkem 5 osob

pouze detekce s  $P(\text{osoba}) > 0.7$

5 detekovaných osob  
 $\text{recall} = 4/5 = 80\%$   
 $\text{precision} = 4/5 = 80\%$

# Precision a recall: příklad



false positive

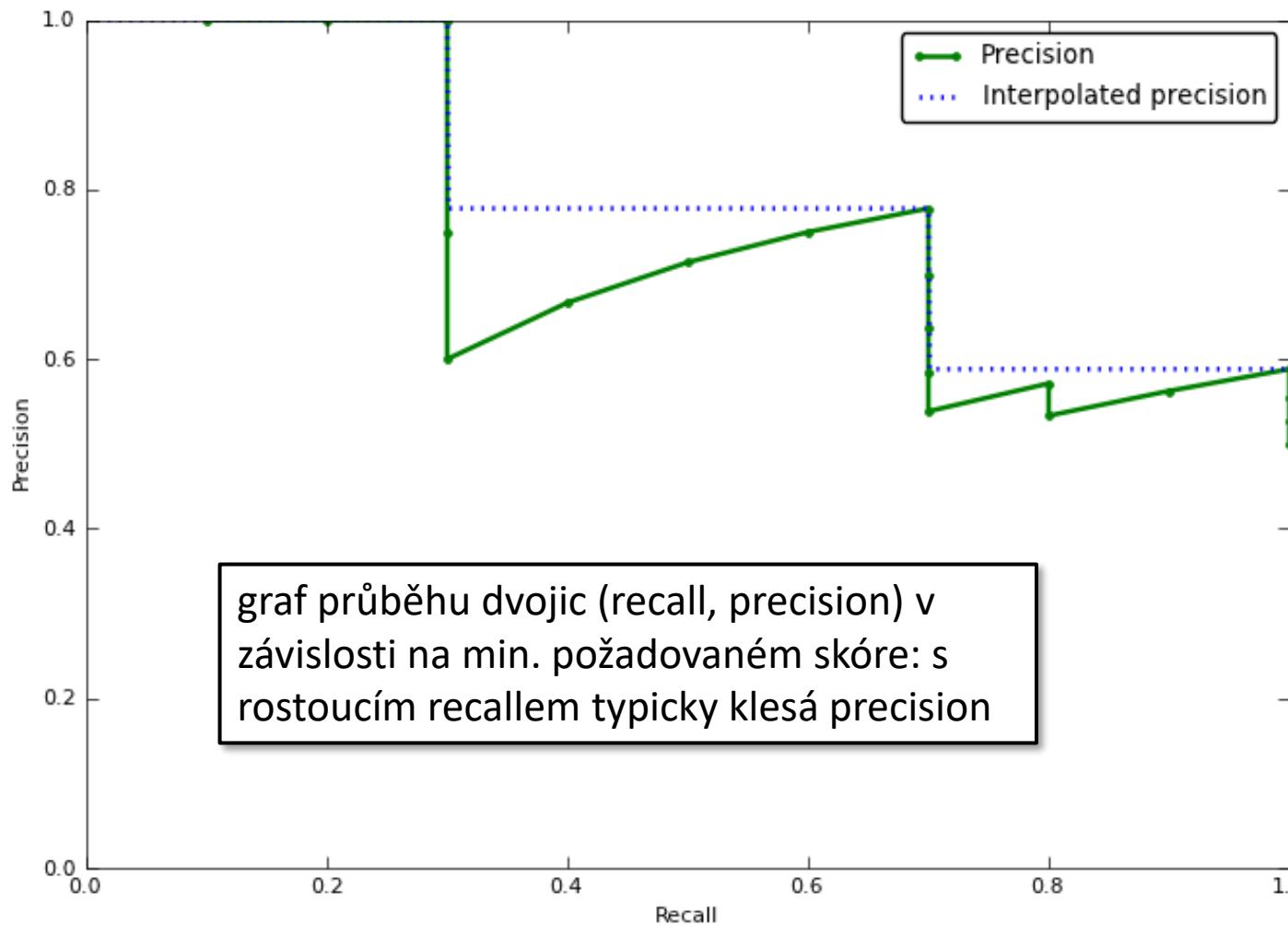
také false positives, protože překrývají osoby, které už byly přiřazeny jiným detektům s vyšší  $P(\text{osoba})$

**ground truth**  
celkem 5 osob

pouze detekce s  $P(\text{osoba}) > 0.5$

9 detekovaných osob  
 $\text{recall} = 5/5 = 100\%$   
 $\text{precision} = 5/9 = 56\%$

# Mean average precision (mAP)



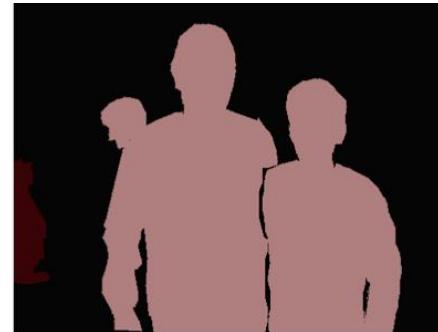
**Mean average precision (mAP)**  
= precision zprůměrovaná přes všechny hodnoty recallu

v případě více tříd (detektor osob, koček, aut, ..., zároveň) se mAP uvádí jako průměr přes všechny třídy

# Segmentace

---

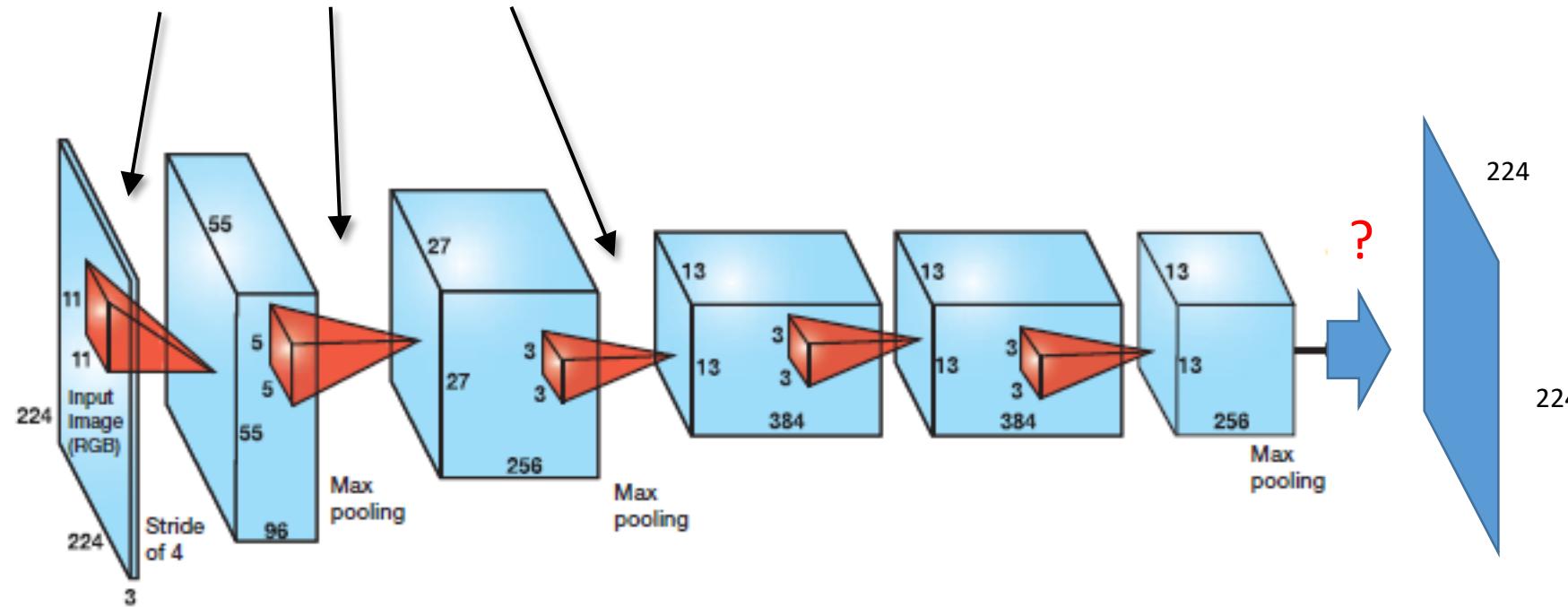
# Sémantická segmentace



- 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!

# Obrázek jako výstup

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

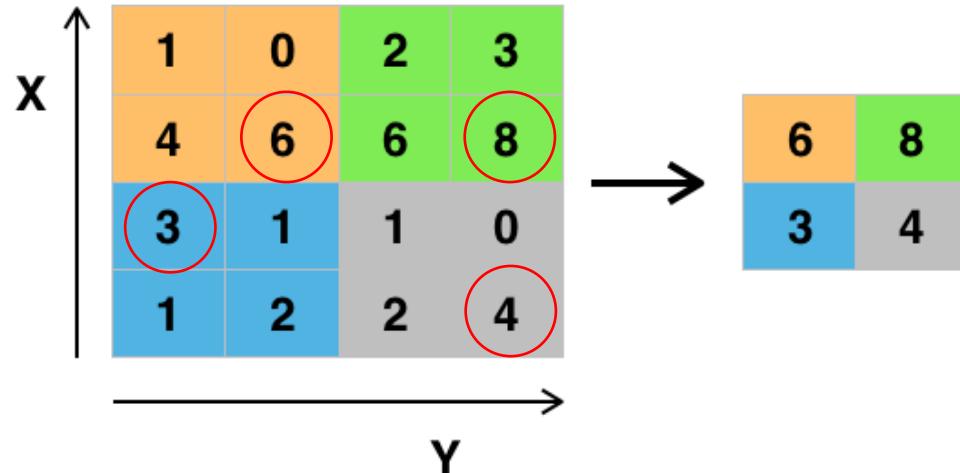


jak vyrobit/rekonstruovat obrázek v původním  
nebo vyšším rozlišení, než je poslední vrstva?

# Max Unpooling

## Max pooling

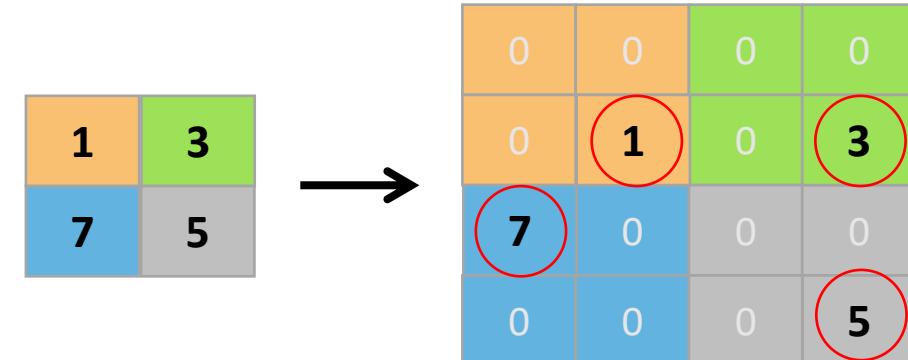
Single depth slice



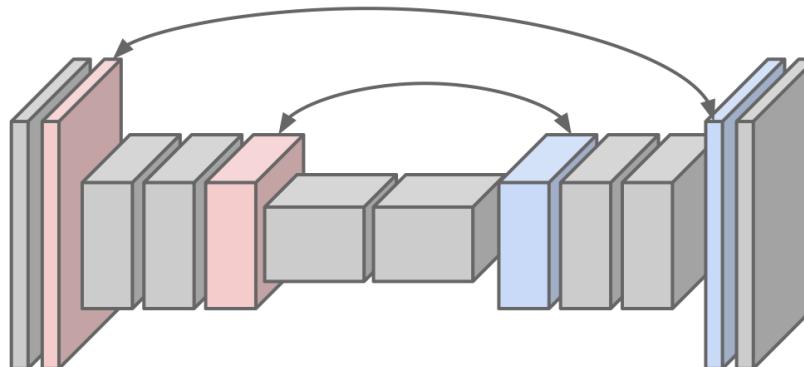
zapamatují se pozice maxim

předpoklad symetrické konvoluční sítě se stejným počtem poolingů i unpoolingu

## Max unpooling



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

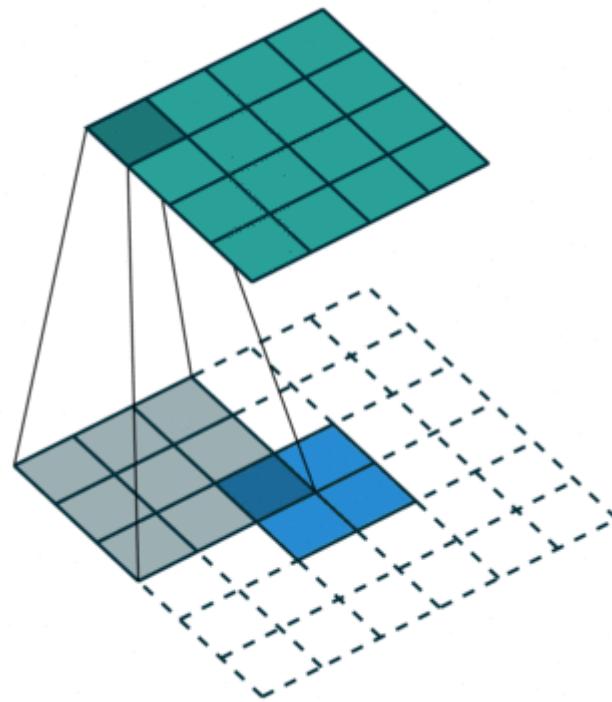


všimněme si: připomíná  
zpětný průchod max poolingu

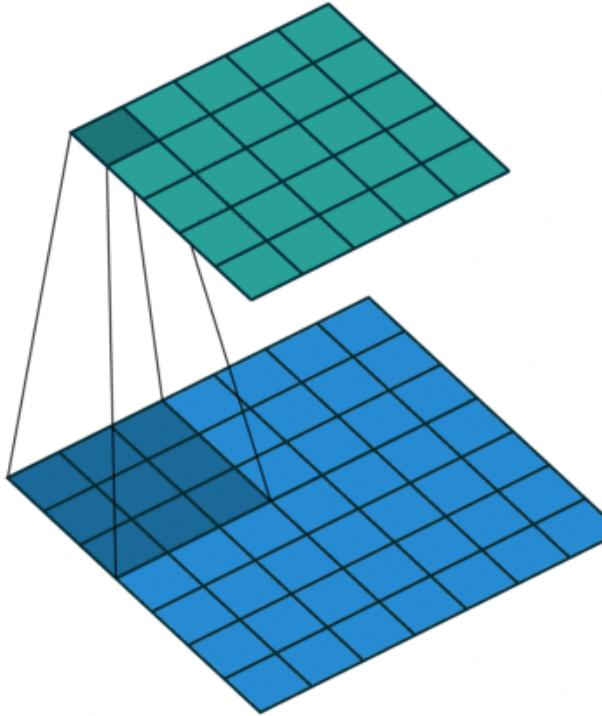
# Transponovaná konvoluce (transposed convolution)

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

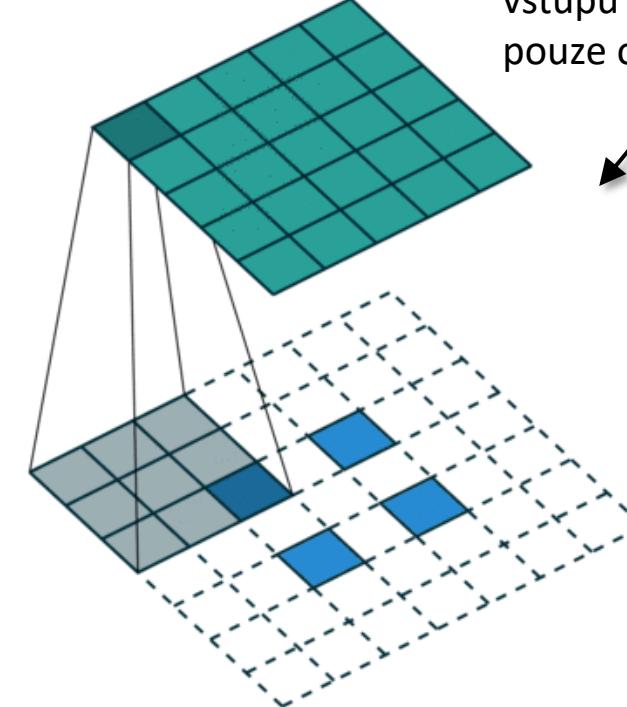
tj. kdybychom měli std. konvoluci s parametry uvedenými dole, tak její zpětný (transponovaný) průchod by vypadal právě takto



3x3, stride=1, padding=0



3x3, stride=1, padding=2

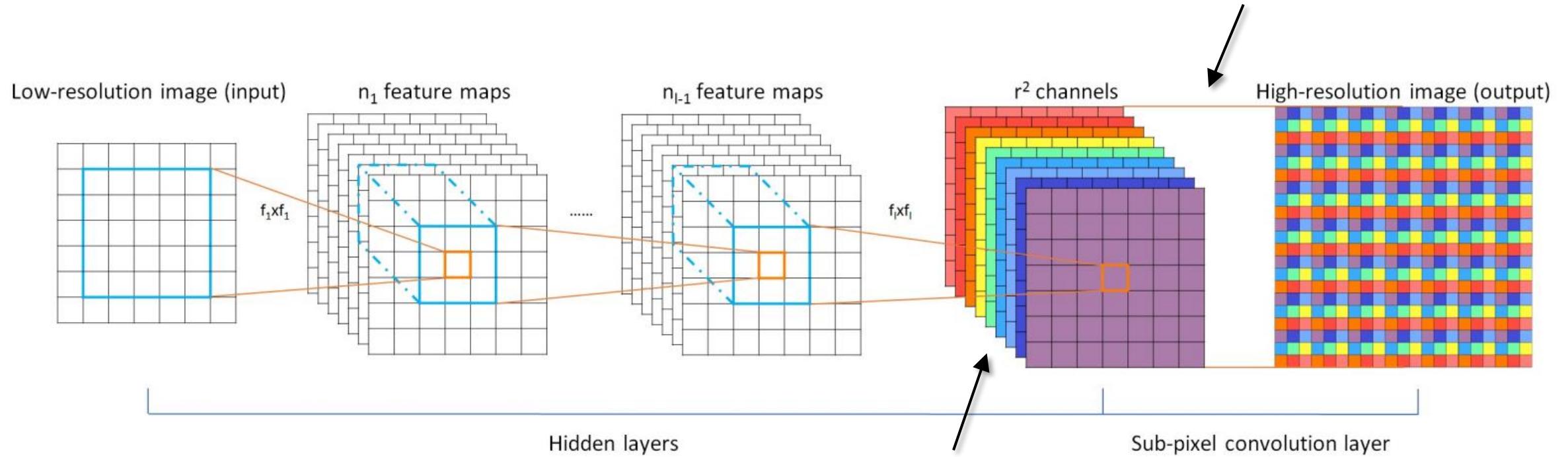


3x3, stride=2, padding=0  
(někdy jako à trous/dilated convolution)

# Sub-pixelová konvoluce

efektivnější než transponovaná konvoluce, přitom bez artefaktů!

poslední krok je vlastně jen „reshape“!



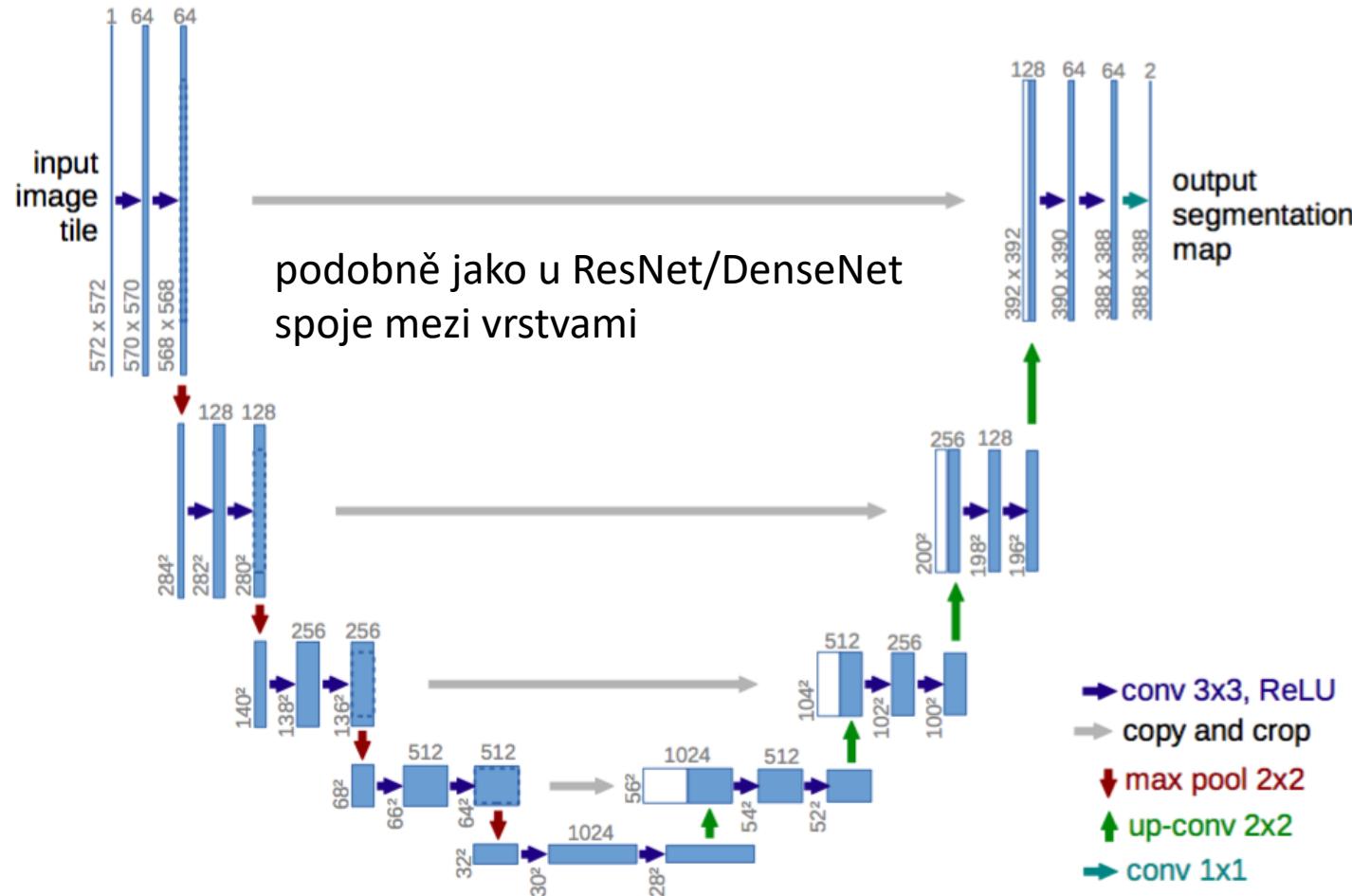
samostatný model pro každé  
požadované zvětšení  $r$

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

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

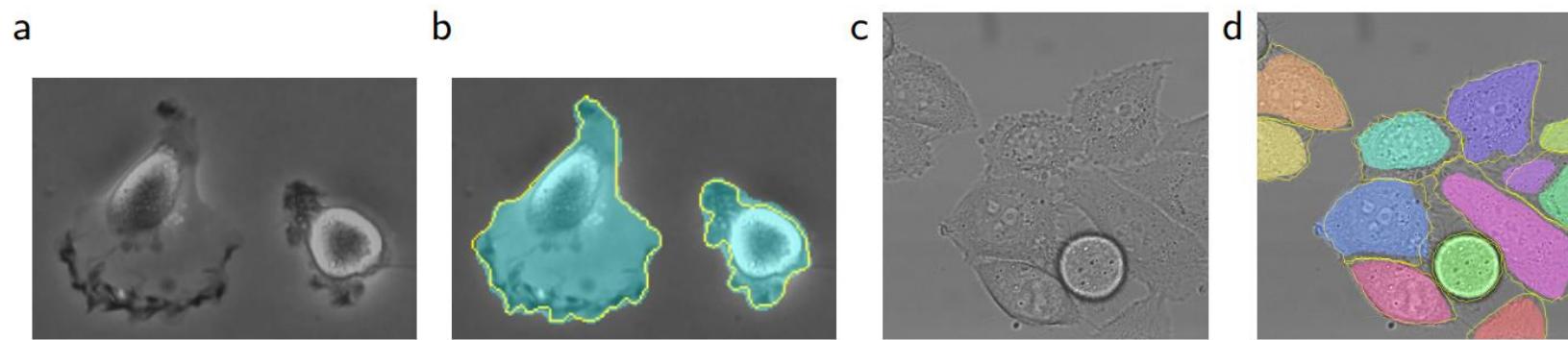
# Sémantická segmentace U-net

architektura typu enkodér - dekodér



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

# Sémantická segmentace U-net



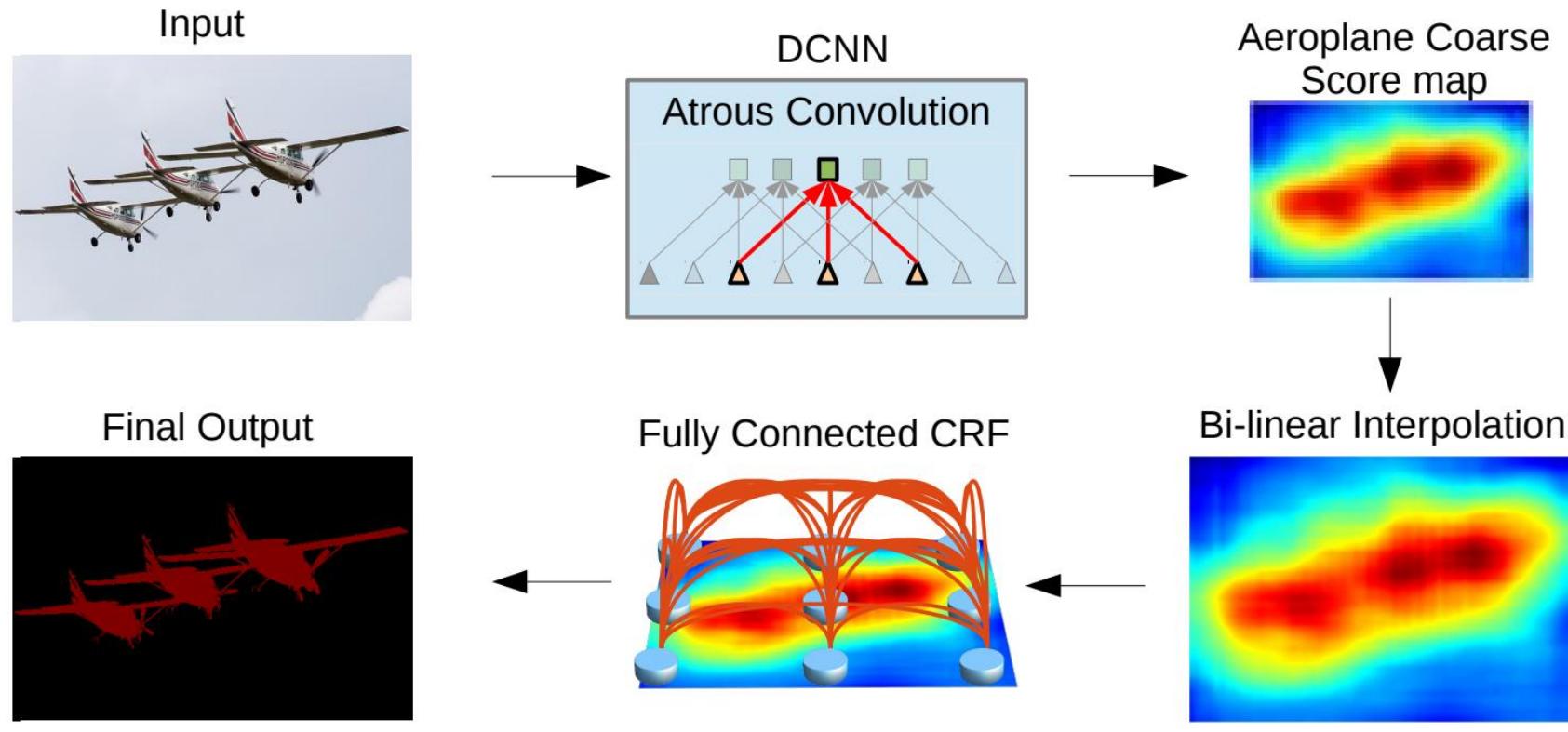
**Fig. 4.** Result on the ISBI cell tracking challenge. (a) part of an input image of the “PhC-U373” data set. (b) Segmentation result (cyan mask) with manual ground truth (yellow border) (c) input image of the “DIC-HeLa” data set. (d) Segmentation result (random colored masks) with manual ground truth (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)	<b>0.9203</b>	<b>0.7756</b>

# Sémantická segmentace skrze dilated konvoluce

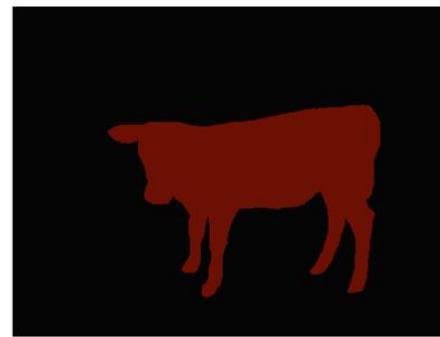
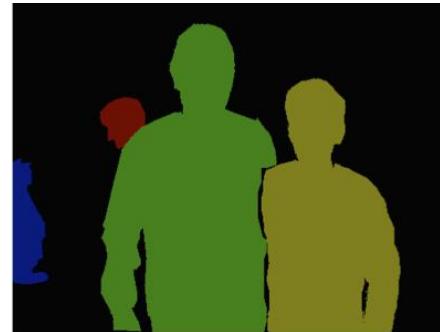
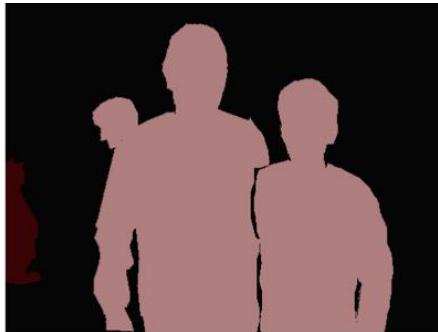
- používá VGG, ze které autoři odstranili všechny max poolingy a nahradili konvolucemi se stride > 1
- upsampling potom opět pomocí dilated konvolucí = transponovaná konvolece se stride > 1



CRF ... conditional random fields  
slouží pro vyhlazení výstupu (**postprocessing**)

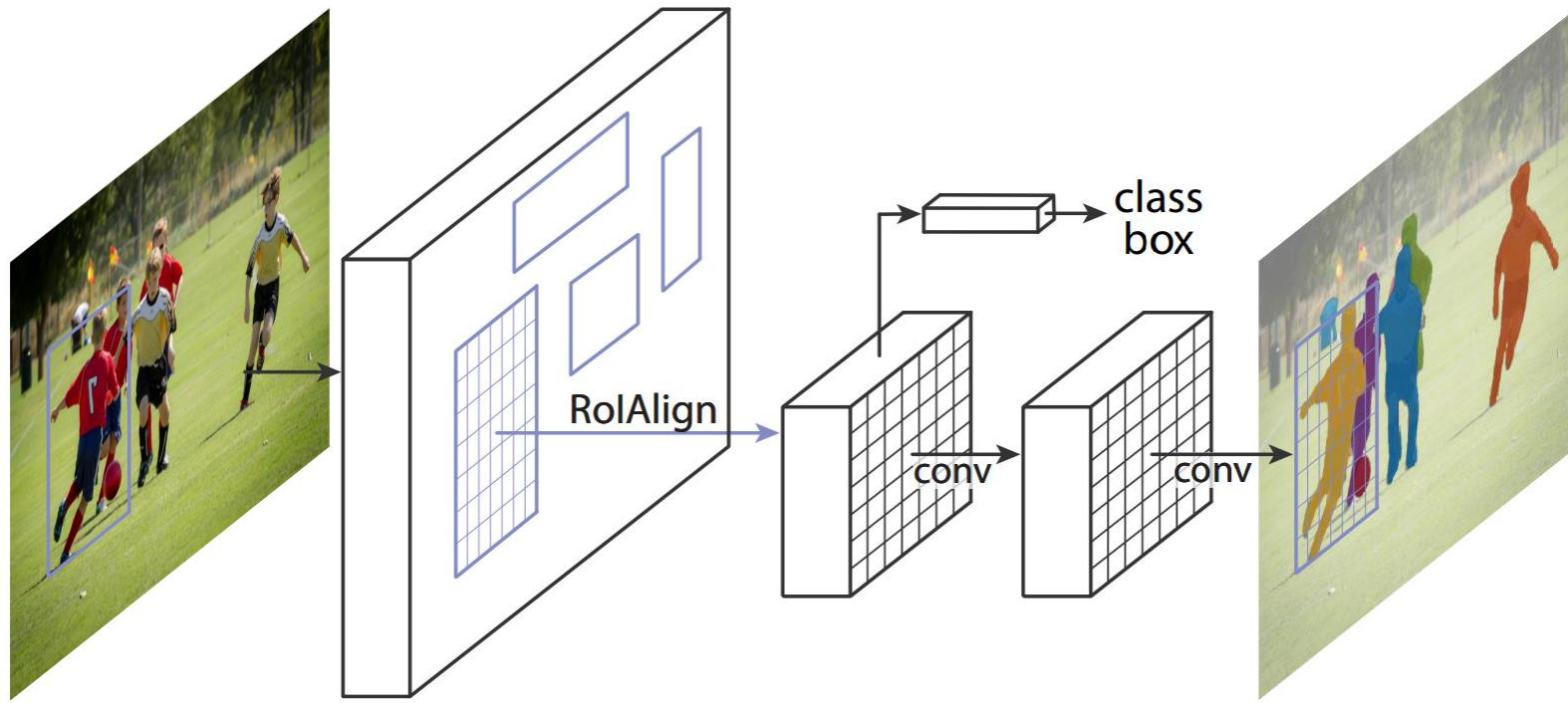
článek: [Chen et al: DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs](#)

# 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 <sup>bb</sup>	AP <sup>bb</sup> <sub>50</sub>	AP <sup>bb</sup> <sub>75</sub>	AP <sup>bb</sup> <sub>S</sub>	AP <sup>bb</sup> <sub>M</sub>	AP <sup>bb</sup> <sub>L</sub>
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	<b>52.1</b>
Faster R-CNN, RoIAlign	ResNet-101-FPN	37.3	59.6	40.3	19.8	40.2	48.8
<b>Mask R-CNN</b>	ResNet-101-FPN	38.2	60.3	41.7	20.1	41.1	50.2
<b>Mask R-CNN</b>	ResNeXt-101-FPN	<b>39.8</b>	<b>62.3</b>	<b>43.4</b>	<b>22.1</b>	<b>43.2</b>	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<sup>bb</sup>), multitask training (+0.9 AP<sup>bb</sup>), and ResNeXt-101 (+1.6 AP<sup>bb</sup>).

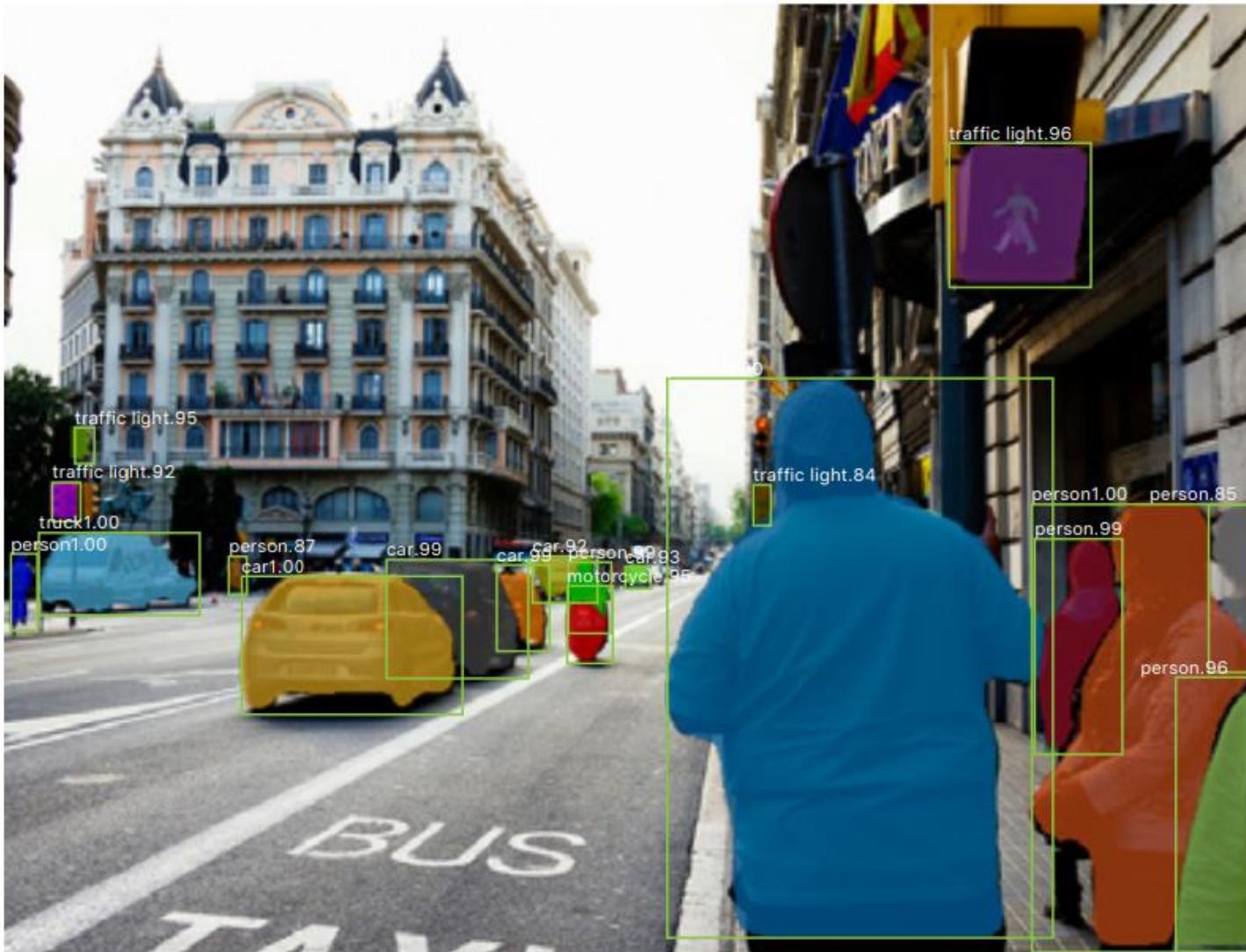
- dosahuje state-of-the-art i pro obyčejnou detekci objektů (bounding boxy!)
- cca 200 ms (5 FPS) na obrázek na NVidia Tesla M40 GPU
- cca 400 ms (2.5 FPS) na obrázek, pokud region proposal net (RPN) a segmentační části sítě nesdílí váhy

# Mask R-CNN



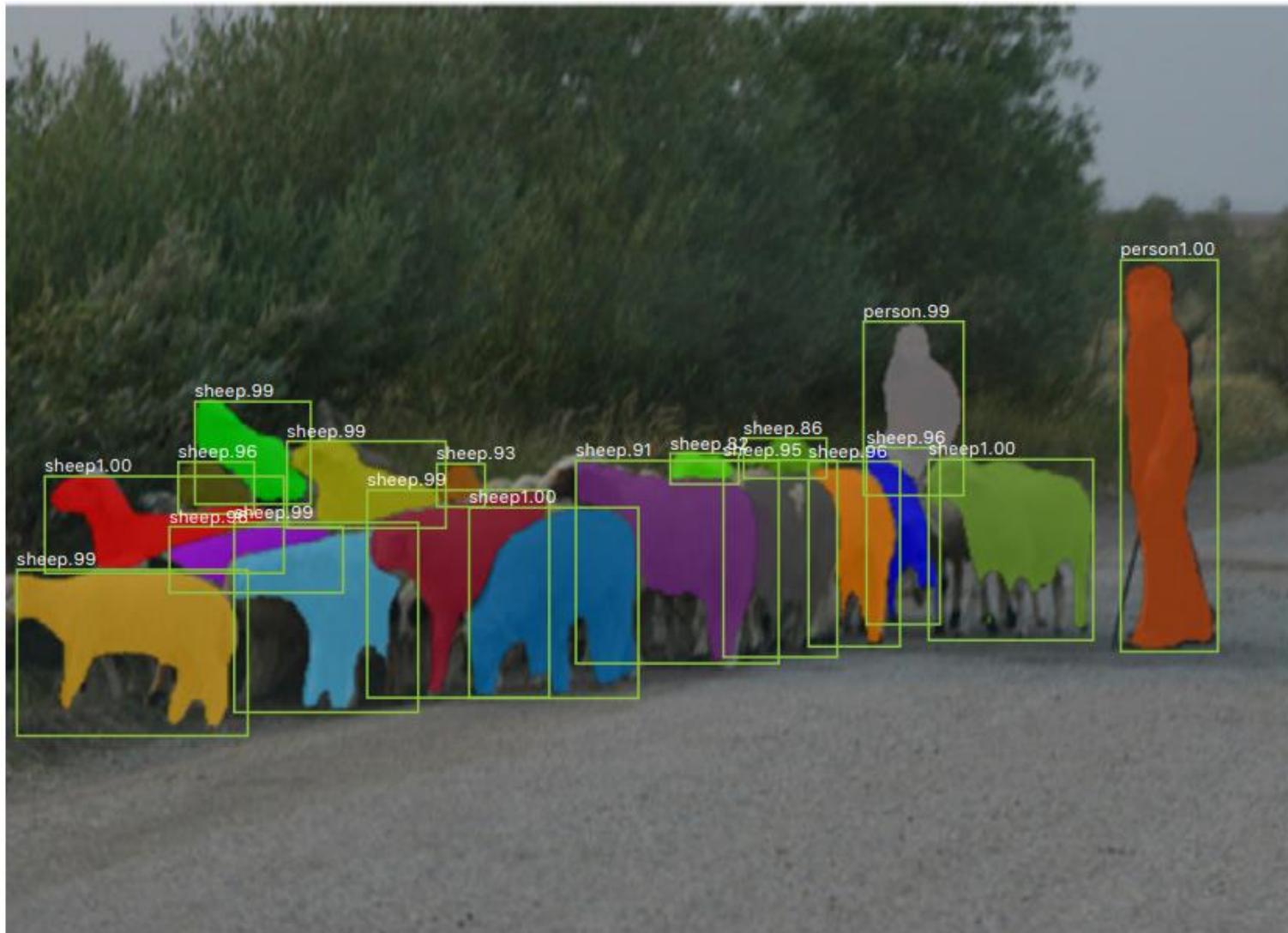
článek: [He et al.: Mask R-CNN](#)

# 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”