Detektor objektov YOLO v3
You Only Look Once

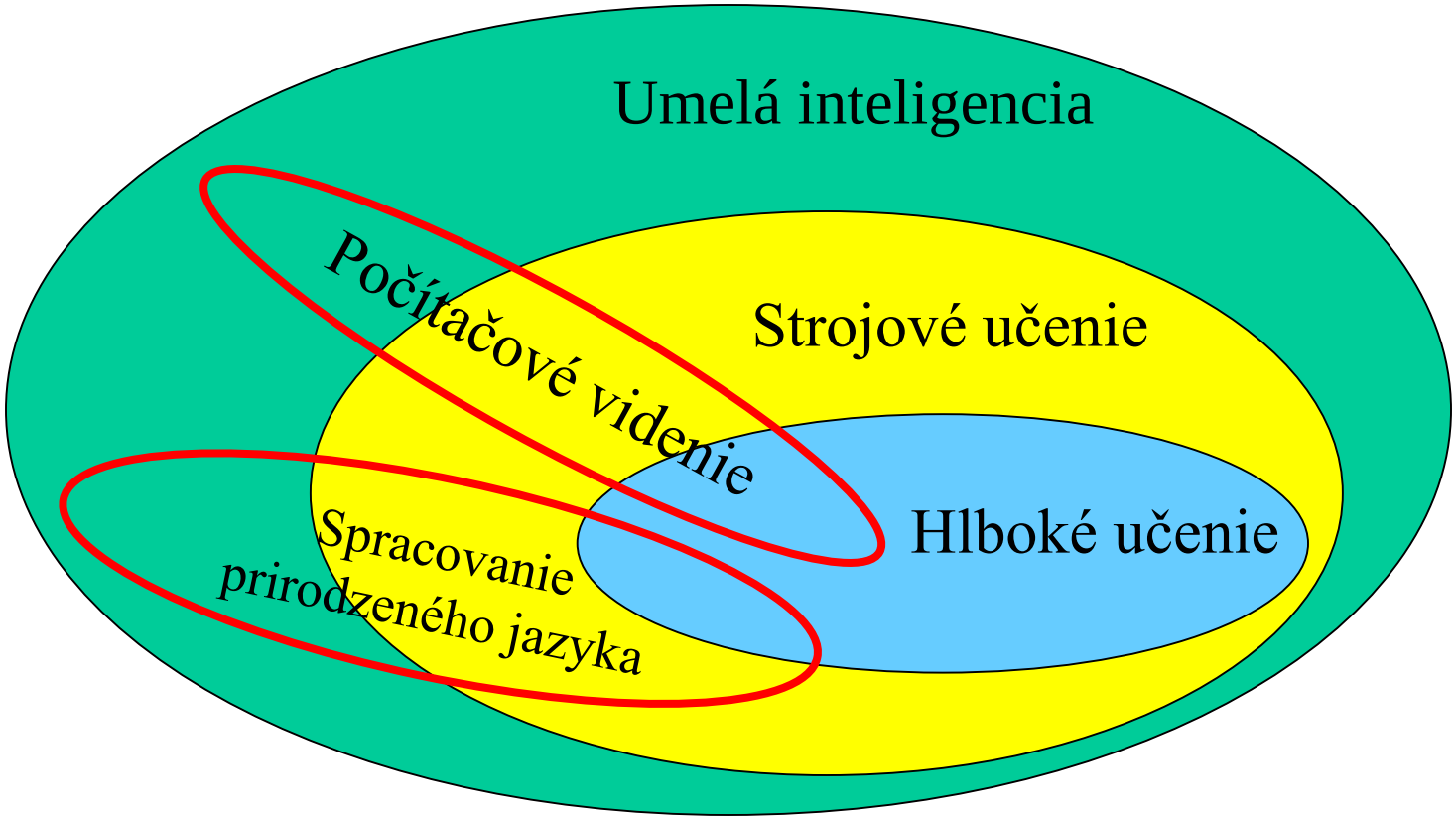
Andrej Lúčny

Katedra aplikovanej informatiky FMFI UK

lucny@fmph.uniba.sk

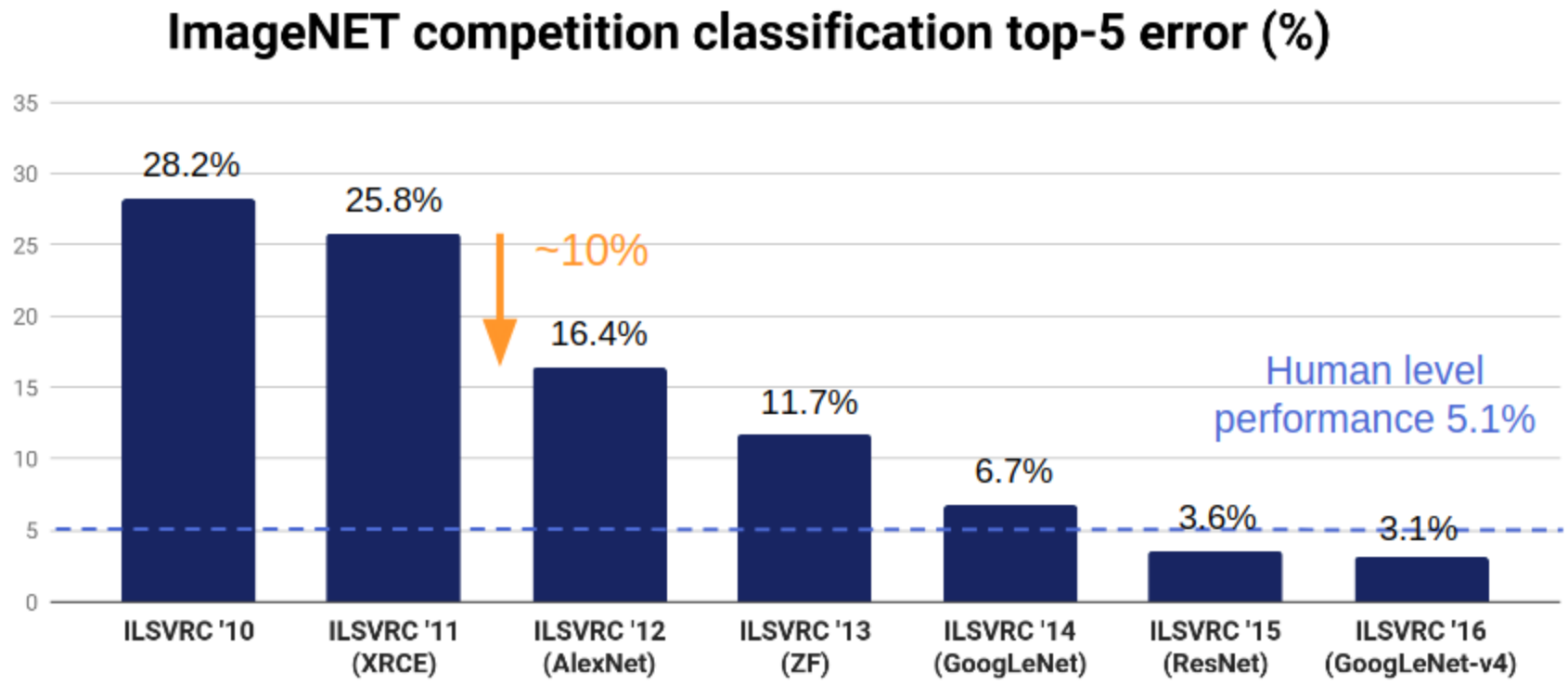
http://dai.fmph.uniba.sk/w/Andrej_Lucny

Pokroky v umelej inteligencii



Boom DNN od 2012

(**D**eep **N**eural **N**etworks)



Boom DNN spôsobilo v prvom rade riešenie problémov s ich trénovaním...

Problémy:

- Preučenie
- Problém strácajúcich sa dát
- Problém strácajúceho sa gradientu

Riešenia:

- Dropout
- Batch normalization
- Xavier initialization
- Reziduálne spojenia

... a dostupnosť výpočtového výkonu

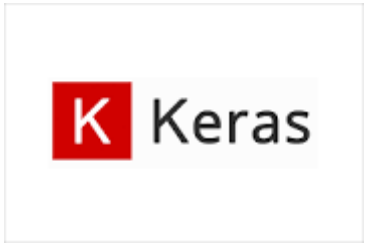
GPU **CPU** **VPU**

NVIDIA. CUDA.

NVIDIA.

intel®

Softwarové nástroje DNN



DataSets

- Mnist
- CIFAR
- ImageNet
- Kaggle
- UC Berkeley
- Caltech
- COCO

...

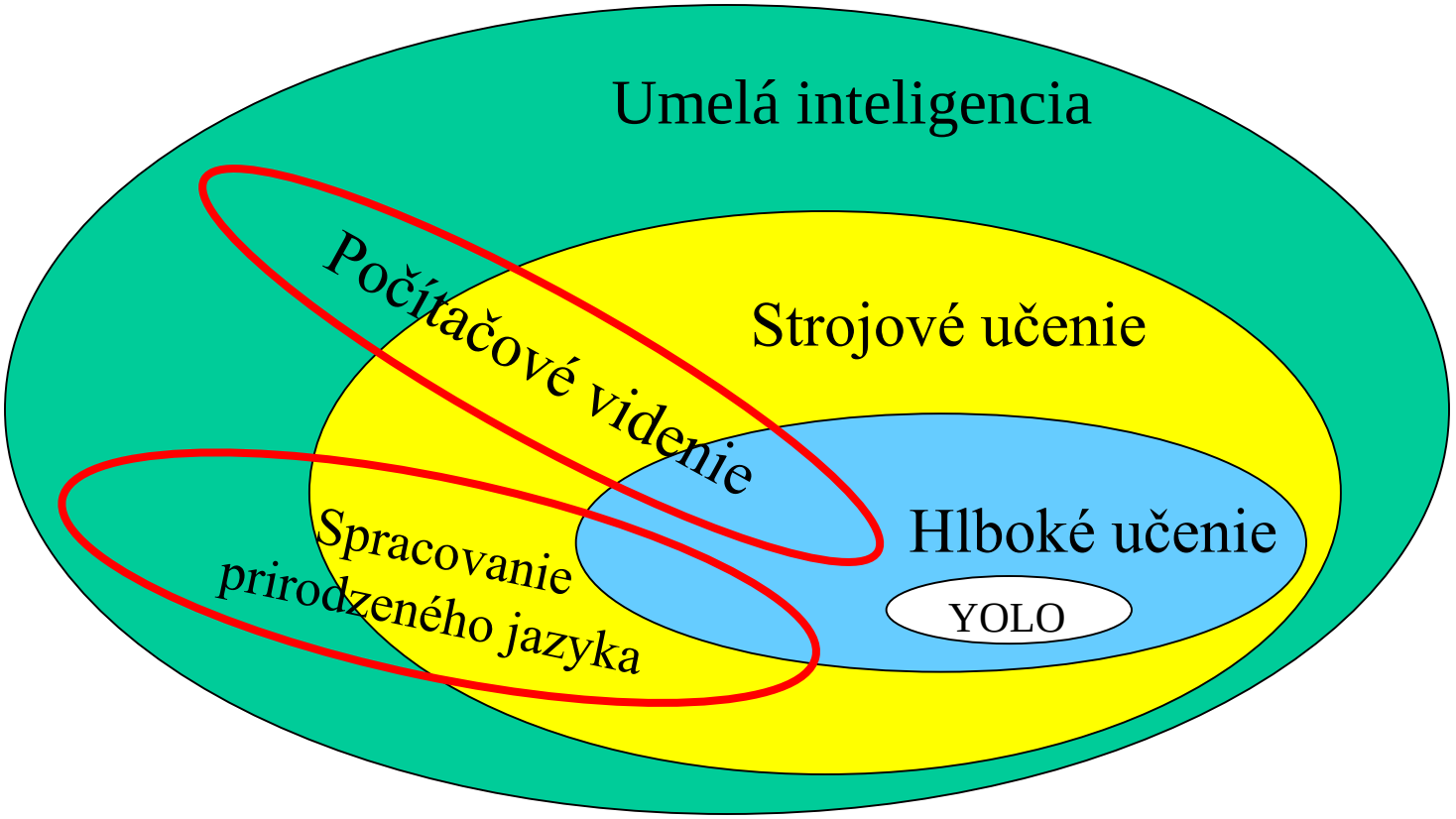
Model ZOO

- VGG
- ResNet
- AlexNet
- DarkNet
- GoogleNet
- SegNet

…

- *Gluon*

Pokroky v umelej inteligencii



YOLO v3 - Ali Farhadi & Joseph Redmon [2018]

YOLO

Ako YOLO použijeme?

- Hardware
- Software
- Dataset
- Predtrénovaný model
- Trénovanie
- Testovanie

Hardware

- Potrebujeme graficku kartu NVIDIA, ktorá má GPU (bežné na hernom notebooku)
- Aktualizujeme jej grafický driver

Dataset

- Potrebujeme dataset obrázkov s detekovaným objektom (môže ich byť aj viac druhov)

`recorder\record.bat`

Dataset

- Ku každému obrázku potrebujeme anotáciu, ktorá nám hovorí kde je na obrázku objekt

`3a1c9e9e582136b45999c2386100ee7a.jpg 3a1c9e9e582136b45999c2386100ee7a.txt`

0 ← *classId*

0.1515625 ← *center-x*

0.27049180327868855 ← *center-y*

0.16145833333333334 ← *width*

0.3296903460837887 ← *height*

`annotator\annotate.bat`

Software

- Potrebujeme software na trenovanie modelu, v danom prípade DarkNet

https://github.com/pjreddie/darknet
https://github.com/AlexeyAB/darknet

- Potrebujeme software na používanie natrénovaného module, použijeme OpenCV

https://opencv.org/

Predtrénovaný model

- Transferred learning: vyjdeme z modelu, ktorý už podstúpil určité trénovanie a je určený na dotrénovanie na konkrétnych dátach
- Pre každú konkrétnu architektúru ako je `darknet-yolov3.cfg` je taký model k dispozícii, napríklad `darknet53.conv.74`

```
wget https://pjreddie.com/media/files/darknet53.conv.74
```

Transfer learning

- Model na detekciu psa sa bude zrejme dosť podobať na model detekujúci snehuliaka, minimálne v prvých vrstvách kde sa spracúva obraz ako taký.
- Je preto výhodnejšie pretrénovať model psa na snehuliaka ako trénovať snehuliaka z ničoho
- Každá knižnica pre DNN preto dodáva aj tzv. pretrained models

Trénovanie

- Vytvoríme súbor s názvom objektu

snowman.

- Vytvoríme konfiguračný súbor trénovania

```
classes = 1
train  = snowman_train.txt
valid  = snowman_test.txt
names = classes.names
backup = weights
```

```
training\*.txt
```

Trénovanie

- Upravíme architektúru
- Hlavne násobíme 2x `subdivisions` kým sa prestane tréning rúbať

```
# Based on cfg/yolov3-voc.cfg

[net]
# Training
batch=64
subdivisions=64
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
learning_rate=0.001
burn_in=400
max_batches=5200
policy=steps
steps=3800
scales=.1

[convolutional]
```

`training\train.bat`

Trénovanie

- Spustíme trénovanie

```
darknet.exe detector train darknet.data
      darknet-yolov3.cfg darknet53.conv.74 >
      train.log
```

- V prípade prerušenia spustíme trénovanie od posledného záznamu

```
darknet.exe detector train darknet.data
      darknet-yolov3.cfg
      weights\darknet-yolov3_last.weights >
      train2.log
```

```
grep "avg" train.log
```

Trénovanie

1: 1926.908691, 1926.908691 avg loss, 0.000000 rate, 16.382236 seconds, 64 images
2: 1928.343750, 1927.052246 avg loss, 0.000000 rate, 16.625786 seconds, 128 images
3: 1929.291260, 1927.276123 avg loss, 0.000000 rate, 16.577889 seconds, 192 images
4: 1928.152588, 1927.363770 avg loss, 0.000000 rate, 16.851934 seconds, 256 images
5: 1927.867798, 1927.414185 avg loss, 0.000000 rate, 16.652522 seconds, 320 images
6: 1928.150391, 1927.487793 avg loss, 0.000000 rate, 16.810422 seconds, 384 images
7: 1928.016602, 1927.540649 avg loss, 0.000000 rate, 16.751157 seconds, 448 images
8: 1928.218262, 1927.608398 avg loss, 0.000000 rate, 16.729694 seconds, 512 images
9: 1928.114624, 1927.659058 avg loss, 0.000000 rate, 15.867022 seconds, 576 images
10: 1927.558716, 1927.649048 avg loss, 0.000000 rate, 15.965221 seconds, 640 images
11: 6243.309570, 2359.215088 avg loss, 0.000000 rate, 35.089396 seconds, 704 images
…
5187: 0.041430, 0.056101 avg loss, 0.000100 rate, 36.932604 seconds, 331968 images
5188: 0.037166, 0.054207 avg loss, 0.000100 rate, 36.947924 seconds, 332032 images
5189: 0.058412, 0.054628 avg loss, 0.000100 rate, 36.836279 seconds, 332096 images
5190: 0.049058, 0.054071 avg loss, 0.000100 rate, 36.974160 seconds, 332160 images
5191: 0.033626, 0.052026 avg loss, 0.000100 rate, 17.876321 seconds, 332224 images
5192: 0.029043, 0.049728 avg loss, 0.000100 rate, 18.346750 seconds, 332288 images
5193: 0.026204, 0.047376 avg loss, 0.000100 rate, 18.115039 seconds, 332352 images
5194: 0.074290, 0.050067 avg loss, 0.000100 rate, 18.119981 seconds, 332416 images
5195: 0.045238, 0.049584 avg loss, 0.000100 rate, 18.342966 seconds, 332480 images
5196: 0.050178, 0.049644 avg loss, 0.000100 rate, 18.377167 seconds, 332544 images
5197: 0.027712, 0.047450 avg loss, 0.000100 rate, 18.085651 seconds, 332608 images
5198: 0.109231, 0.053628 avg loss, 0.000100 rate, 18.165273 seconds, 332672 images
5199: 0.035231, 0.051789 avg loss, 0.000100 rate, 18.154648 seconds, 332736 images
5200: 0.059666, 0.052576 avg loss, 0.000100 rate, 18.214233 seconds, 332800 images

Trénovanie

- Trénovanie trvá dlho (s GeForce GTX1050 dva a pol dňa)
- Výsledkom sú parametre (váhy) architektúry `weights\darknet-yolov3_final.weights`
- Architektúra (.cfg) + parametre (.weights) = model

Testovanie

- Upravíme architektúru

```
# snowman.cfg

[net]
# Testing
batch=1
subdivisions=1
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1
learning_rate=0.001
burn_in=400
max_batches=5200
policy=steps
steps=3800
scales=.1

[convolutional]
```

Testovanie

- Naprogramujeme aplikáciu v OpenCV, ktorá model využíva

```
dnn::Net net = readNetFromDarknet("snowman.cfg","snowman.weights");
net.setPreferableBackend(DNN_BACKEND_OPENCV);
net.setPreferableTarget(DNN_TARGET_OPENCL_FP16); // 3.5 fps
//net.setPreferableTarget(DNN_TARGET_OPENCL); // 2.6 fps
//net.setPreferableTarget(DNN_TARGET_CPU); // 1.75 fps

Mat inputBlob = blobFromImage(frame, 1/255.F, Size(416, 416),
        Scalar(0,0,0), true, false);
net.setInput(inputBlob, "data");
std::vector<Mat> outputBlobs;
net.forward(outputBlobs,outputNames);

// process outputBlobs to vector<Rect>
```

Testovanie

- Naprogramujeme aplikáciu v OpenCV, ktorá model využíva

- Spustíme ju

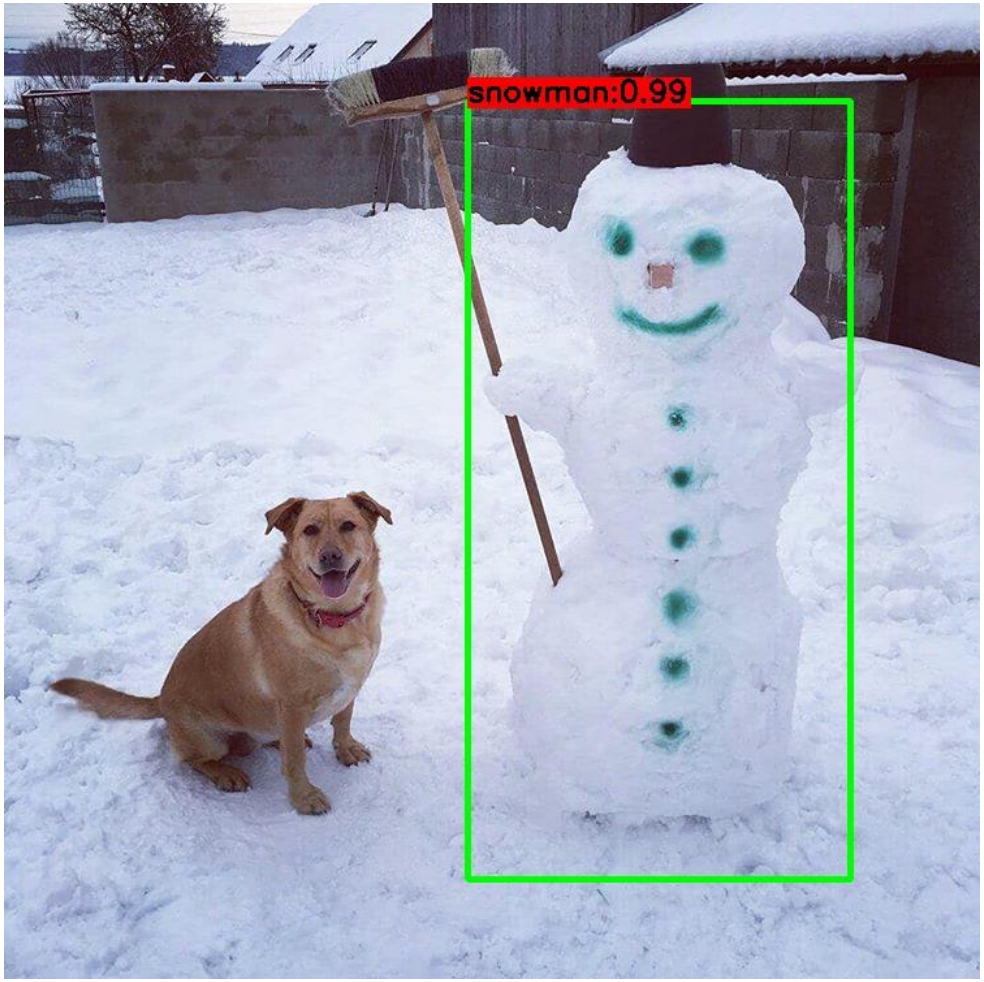
```
set OPENCV_OCL4DNN_CONFIG_PATH=cache
yolov3.exe -cfg=snowman.cfg -model=snowman.weights
    -source=snowmen.mp4 -out=snowmen-labeled.avi
    -class_names=classes.names
```

(prvý obrázok ide pomaly – vytvára sa cache)

Potom s GPU máme 3.6 fps

`testing\apply.bat`

Testovanie


snowman:0.99

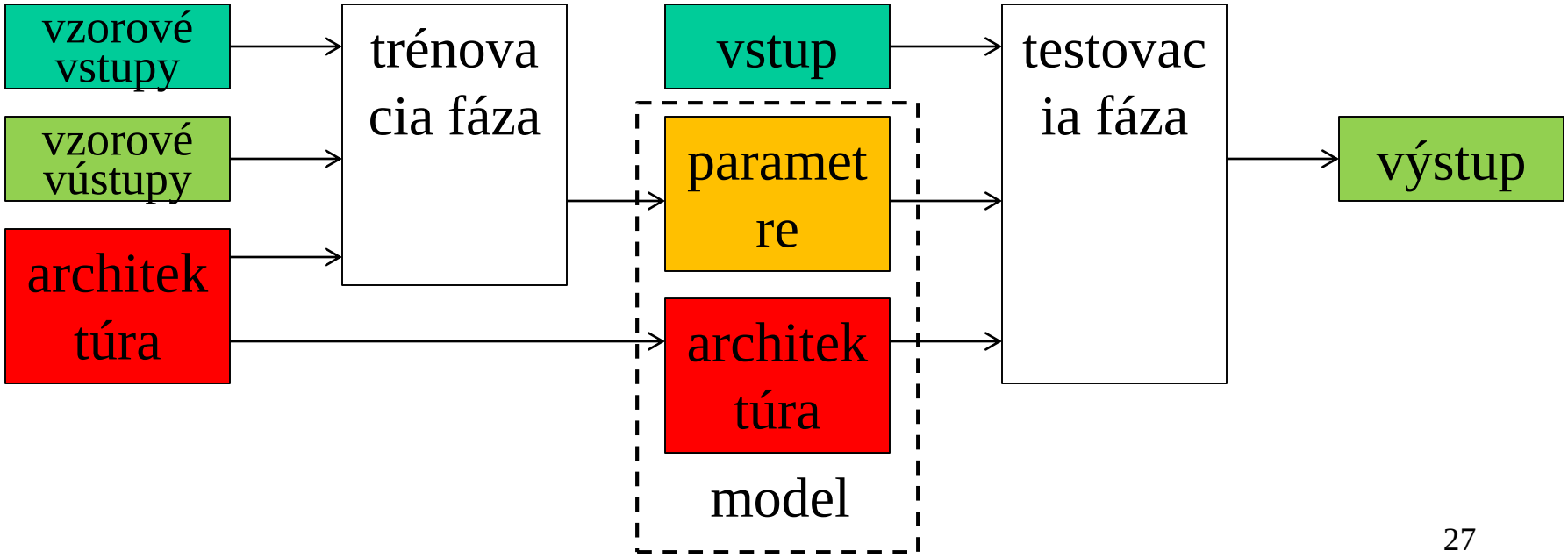

HOUR MINUTE SECOND
10. 16. 28. 88
DATE 06. 05. 2008.
DIRECTOR

Prečo YOLO funguje ?

- Strojové učenie
- Neurónové siete
- Preceptron
- Konvolučné siete
- (Hlboký) autoencoder
- Hlboké učenie
- VGG, ResNet, YOLO

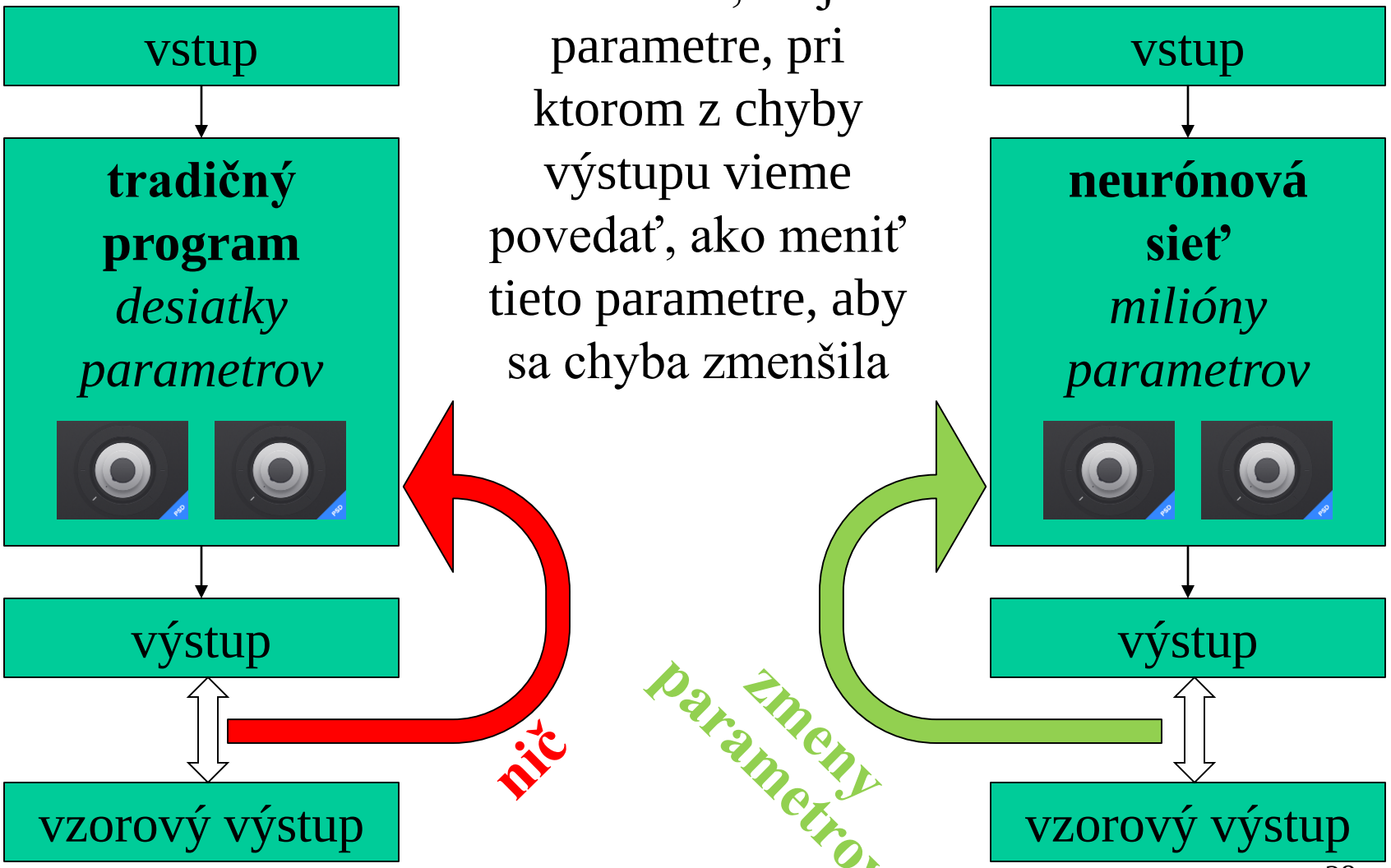
Strojové učenie

- Empirický prístup k programovaniu
- Zo vzorových vstupov a výstupov skonštruujeme model
- Pomocou modelu transformujeme ďalšie vstupy

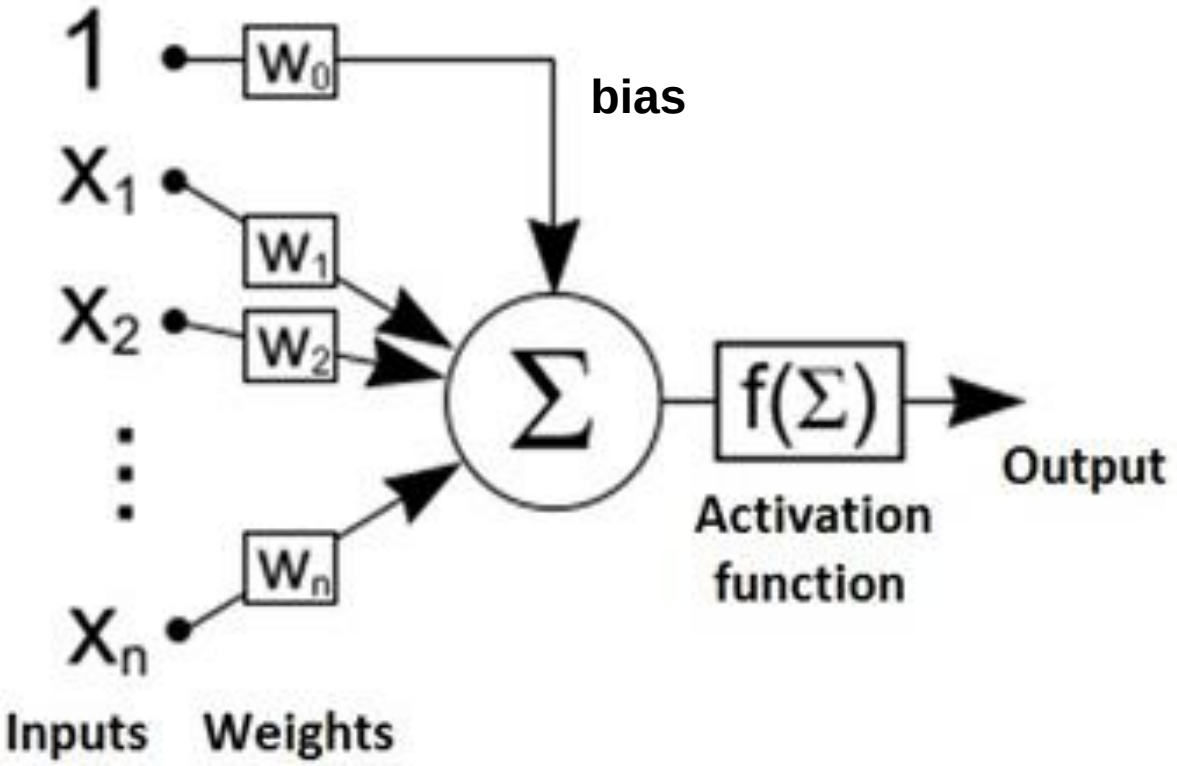


Neurónová sieť

= čokoľvek, majúce parametre, pri ktorom z chyby výstupu vieme povedať, ako meniť tieto parametre, aby sa chyba zmenšila



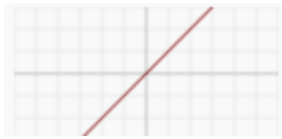
Neurón


1
w_0
bias
x_1
w_1
x_2
w_2
Σ
$f(\Sigma)$
Output
Activation function
w_n
x_n
Inputs
Weights

Aktivácia

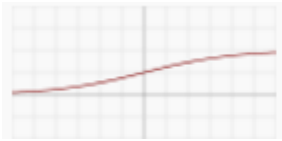
- Na výstup z neurónu je možné aplikovať aktivačnú funkciu

`linear`



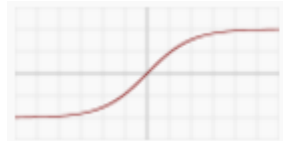
$$f(x) = x$$

`sigmoid`



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

`tanh`



$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

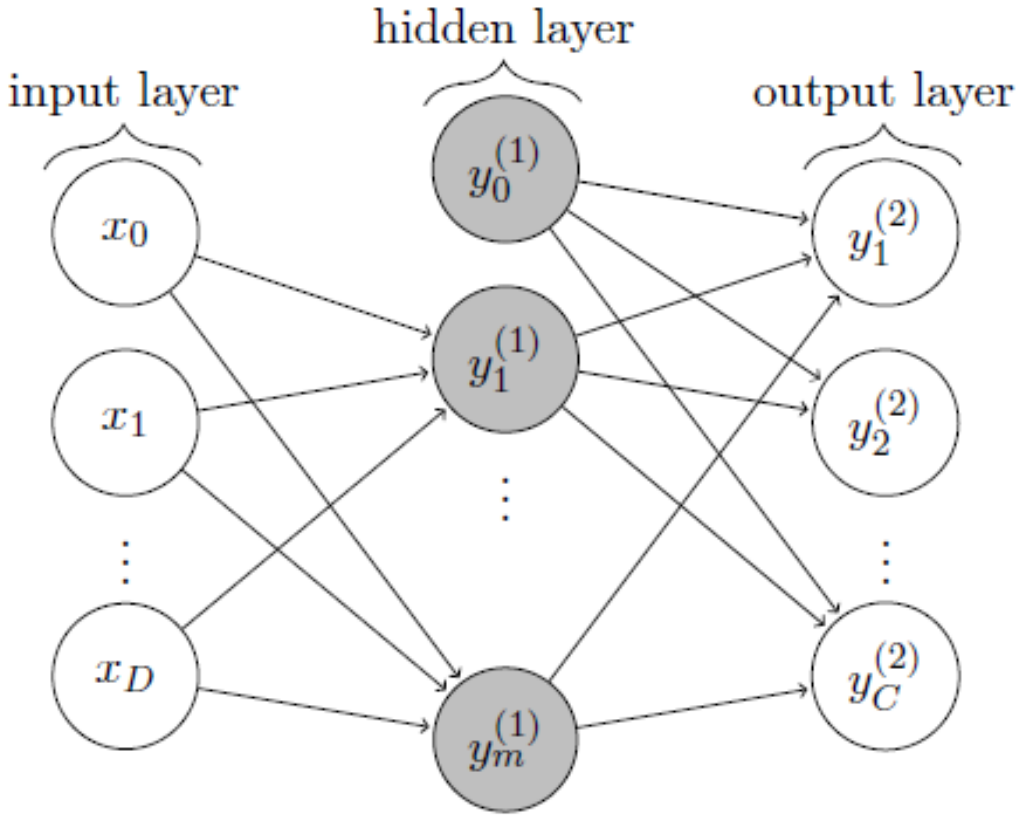
`relu`

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

`softmax`

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

3 vrstvový Perceptron

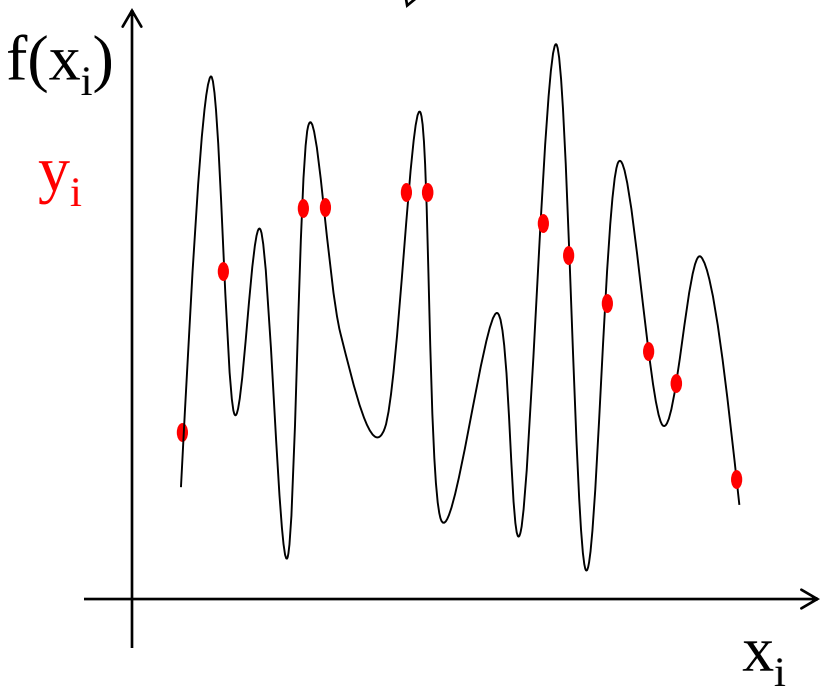


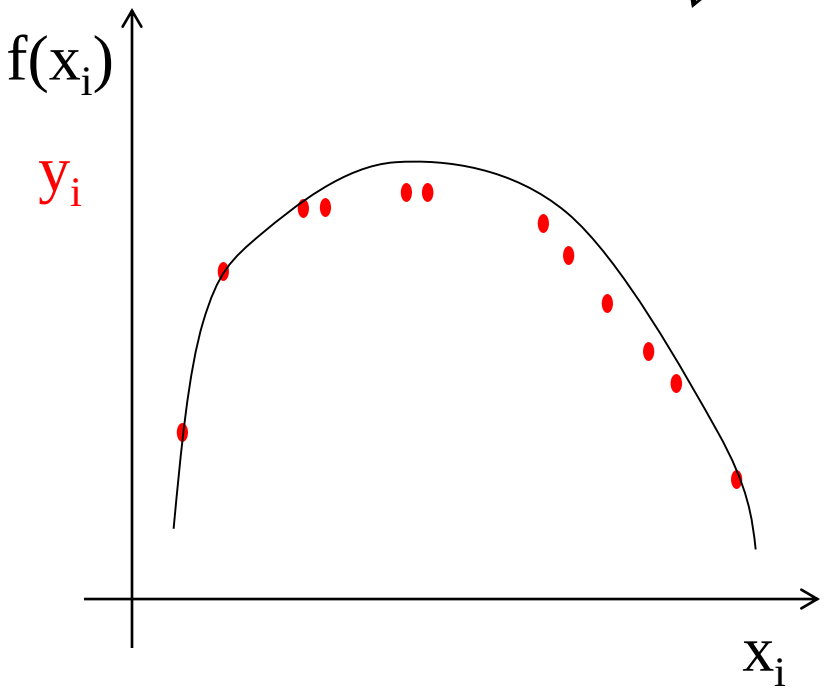
univerzálny aproximátor

Problém

Preučenie (overfitting):

- Radšej väčšiu chybu a lepšiu predikčnú schopnosť ako menšiu chybu a horšiu predikčnú schopnosť





CNN - Konvolučná neurónová sieť (príklad): hranový operátor

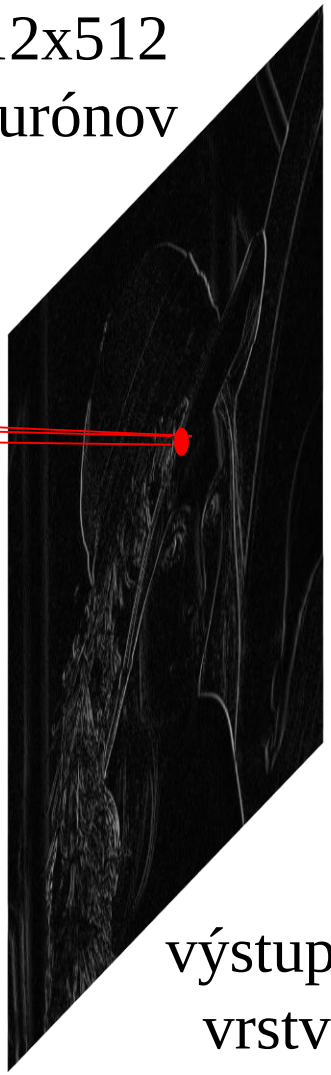
512x512 neurónov

vstupná vrstva

kernel 3x3

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

512x512 neurónov



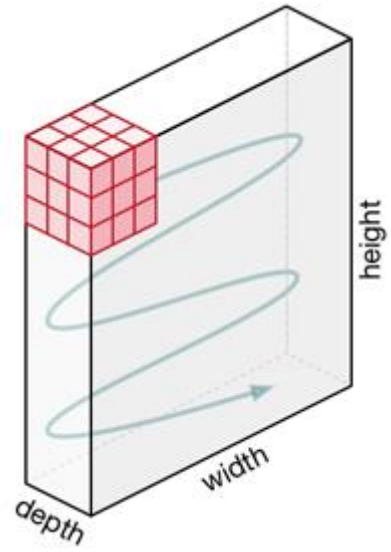
výstupná vrstva

Každý neurón výstupnej vrstvy má spojenia na 3x3 = 9 neurónov vstupnej vstvy

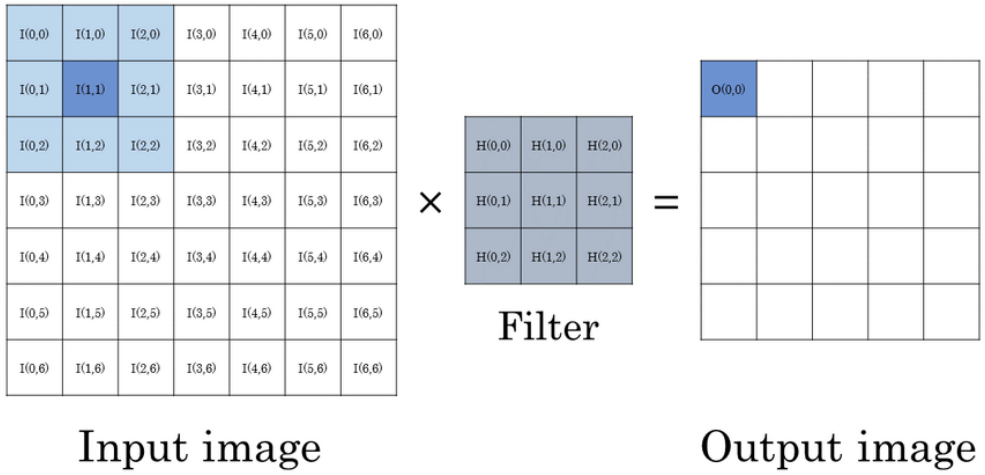
Každý neurón má na spojeniach rovnaké váhy, tj. sieť má 9 parametrov

Sieť môže mať ako 10. parameter bias, ktorý pri ReLU aktivácii predstavuje prah

Tenzor

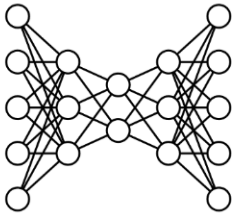


Kernel



Hlboké učenie
(deep learning)

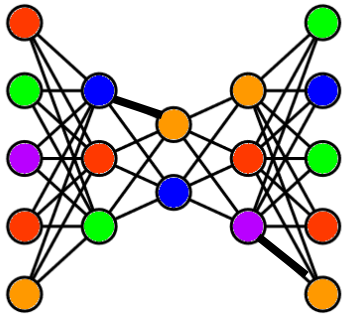
Architektúra

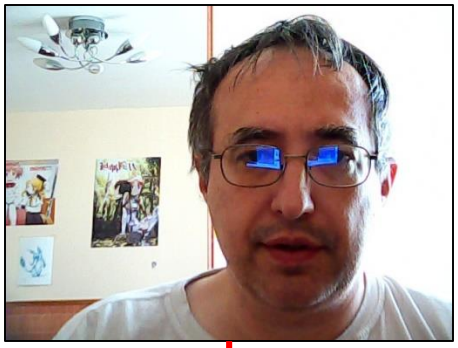


DL knižnica trénujúca DNN

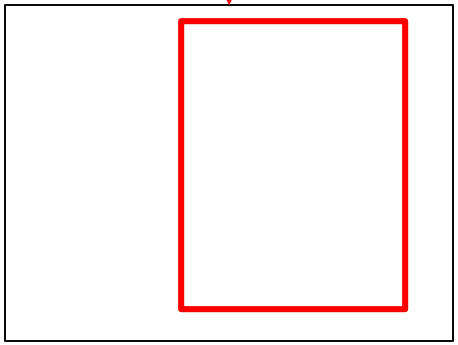
DataSet

Model

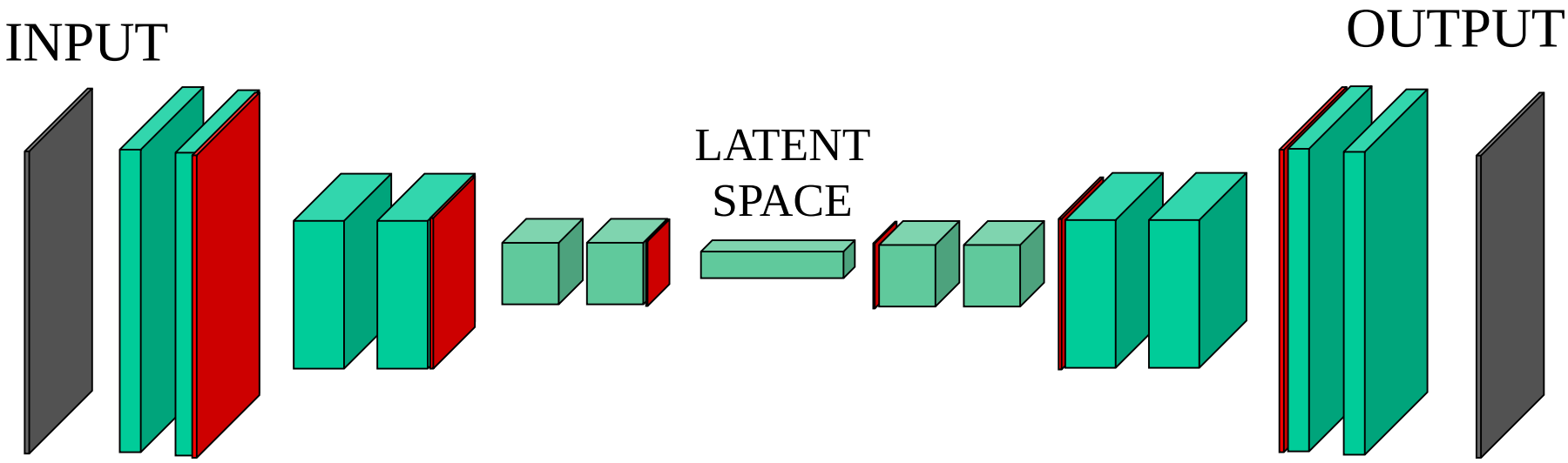




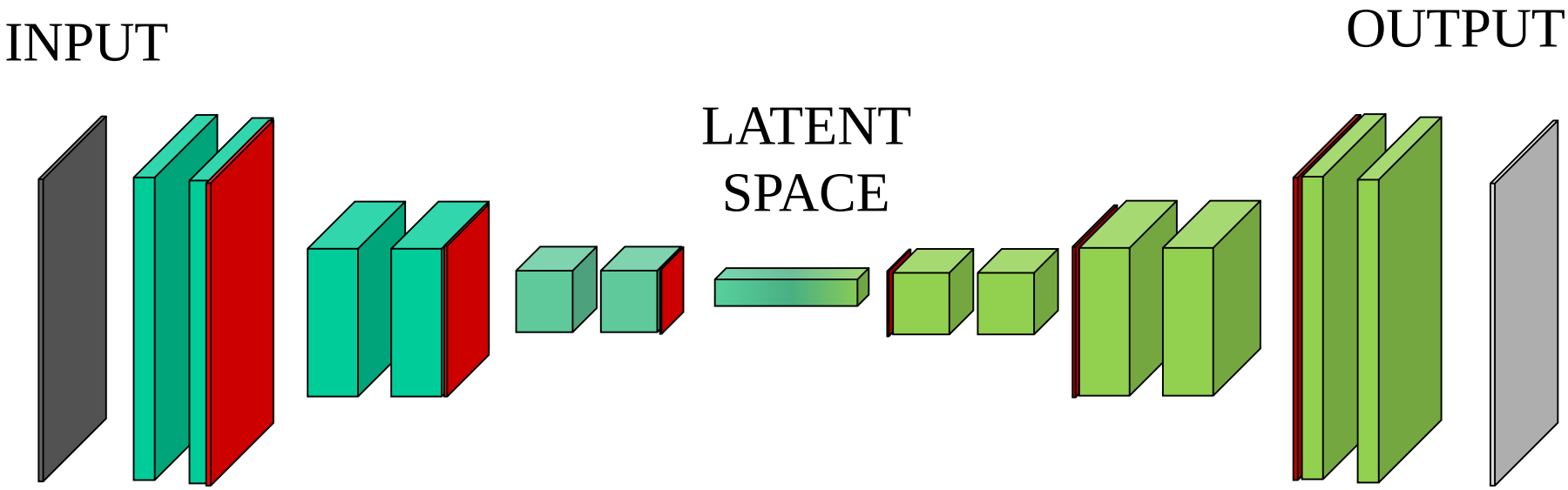
DL knižnica interpretujúca DNN



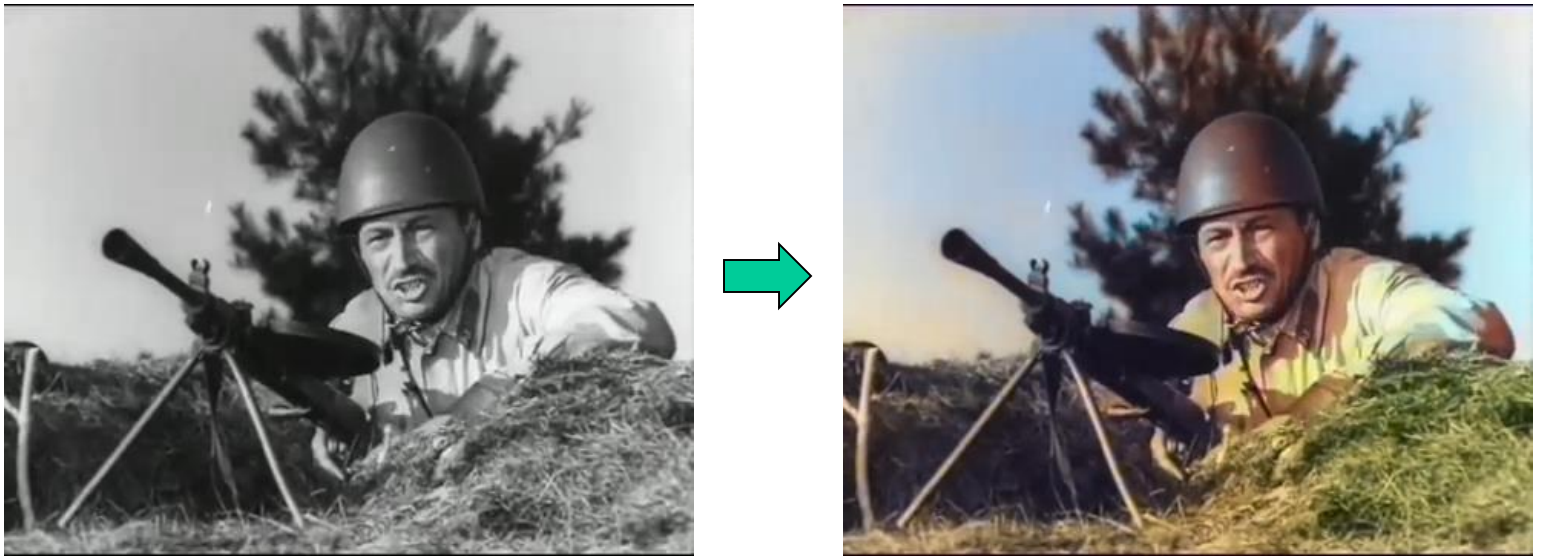
(Hlboký) Autoencoder



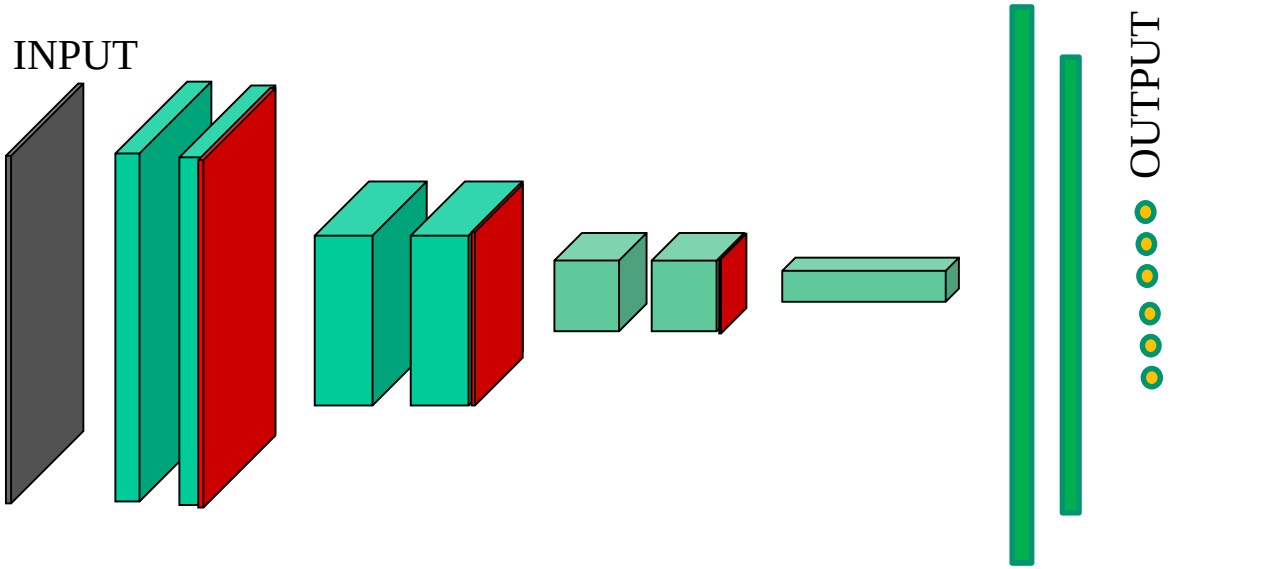
Encoder - Decoder



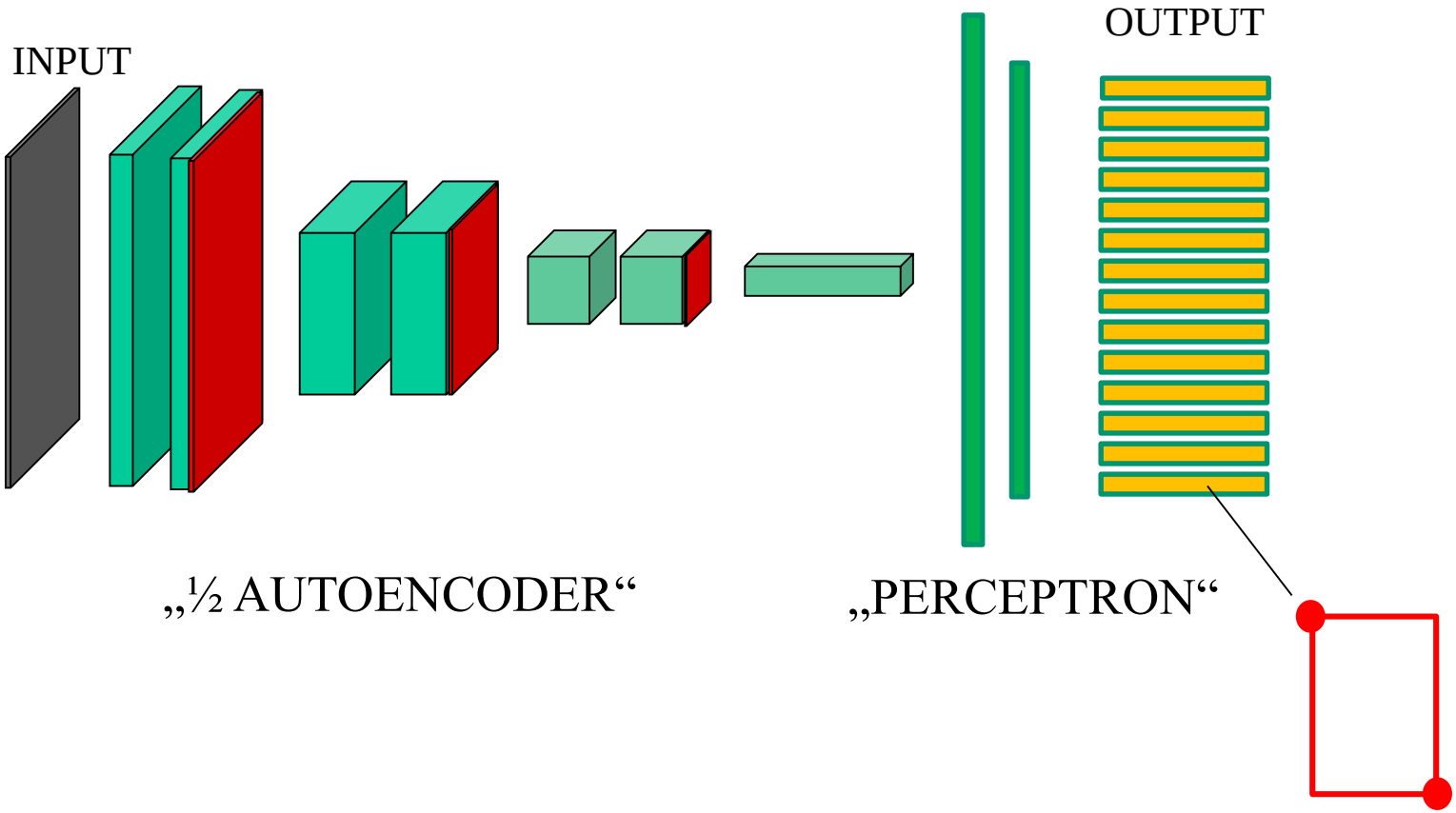
Príklad prekladača: ofarbenie



Klasifikátory

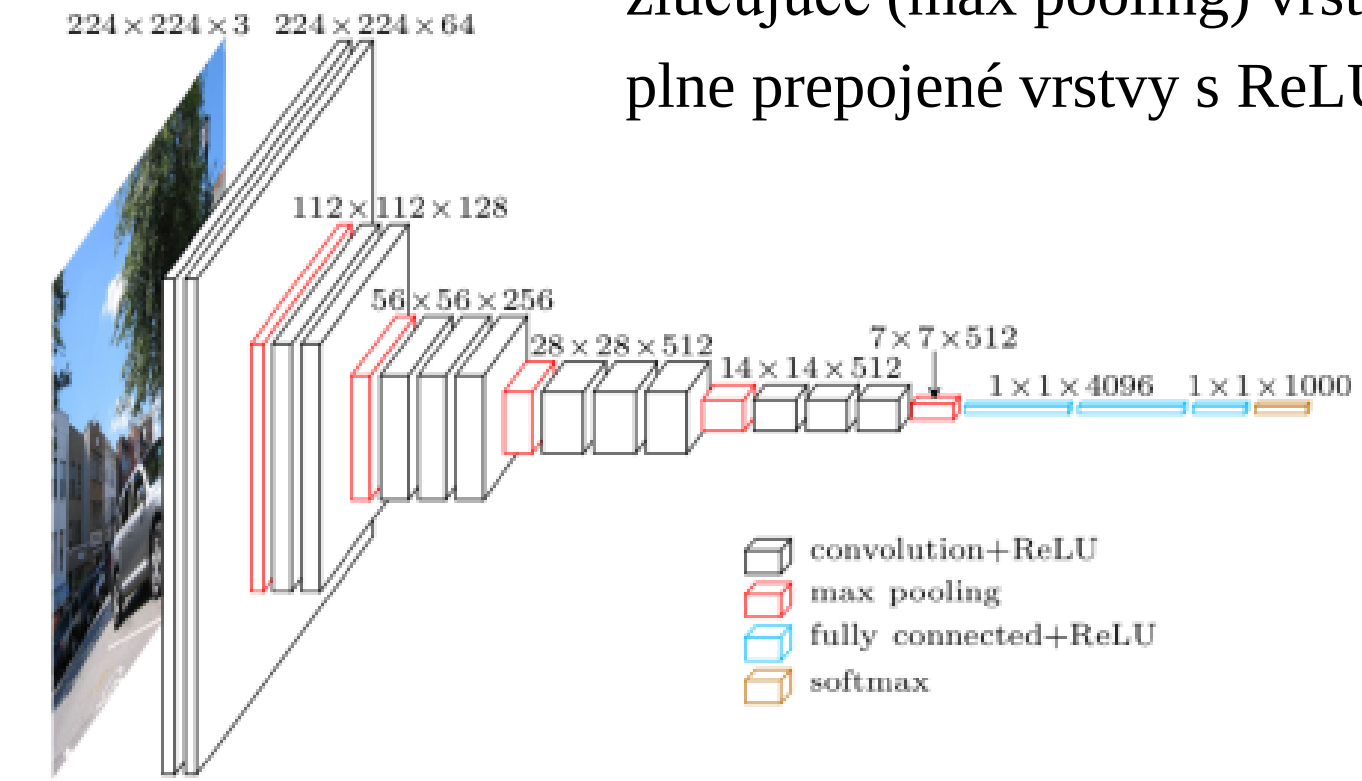


Detektory

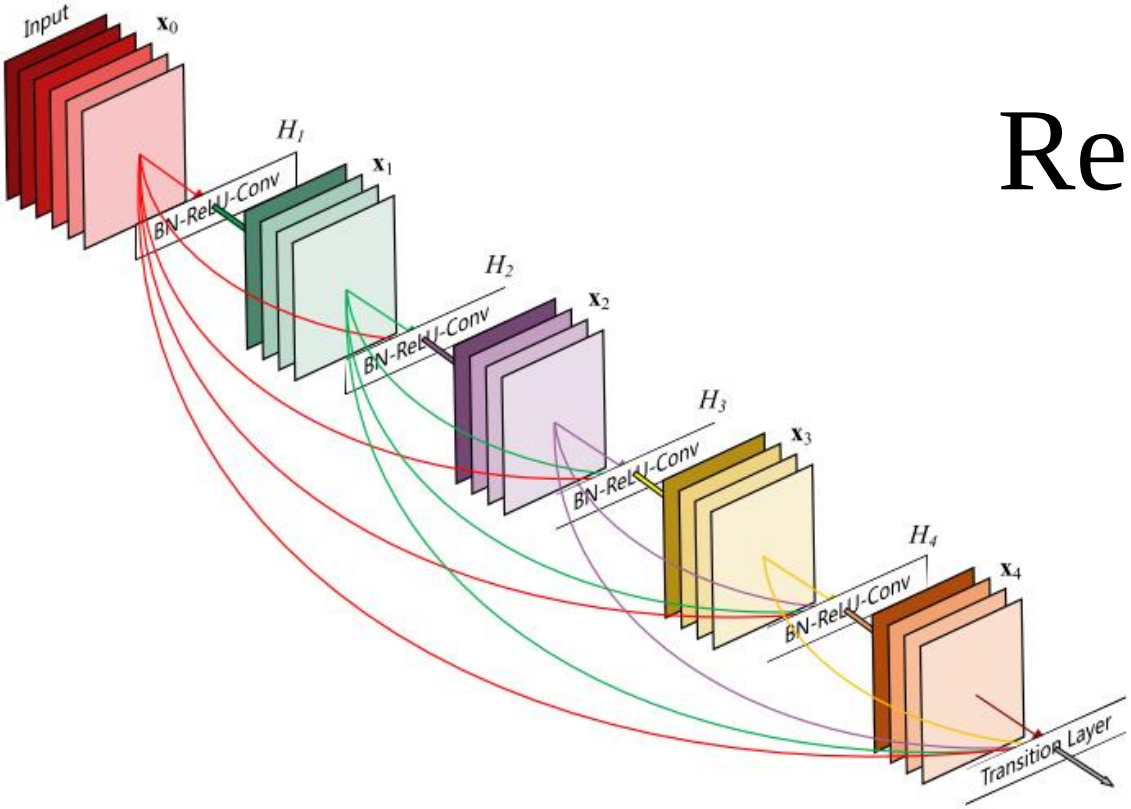


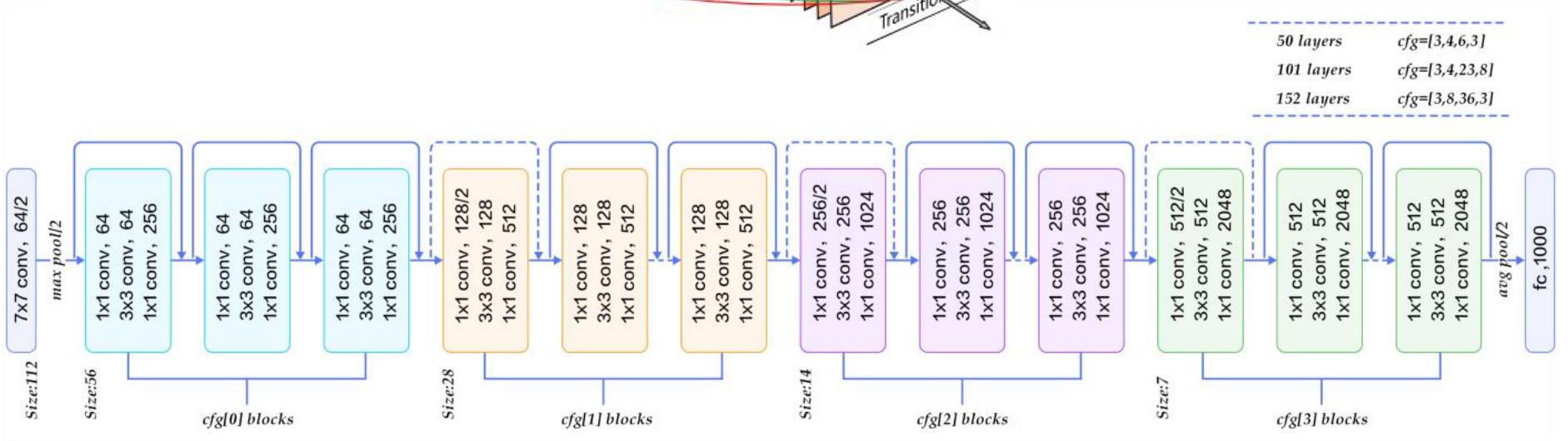
VGG

- VGG je DNN navrhnutá pôvodne pre ImageNet, t.j. klasifikáciu obrázkov, ako každá DNN spracúva tenzory pomocou kernelov.
- Používa 3 stavebné prvky: konvolučné vrstvy s ReLU aktiváciou
  zlučujúce (max pooling) vrstvy
  plne prepojené vrstvy s ReLU a Softmax



ResNet

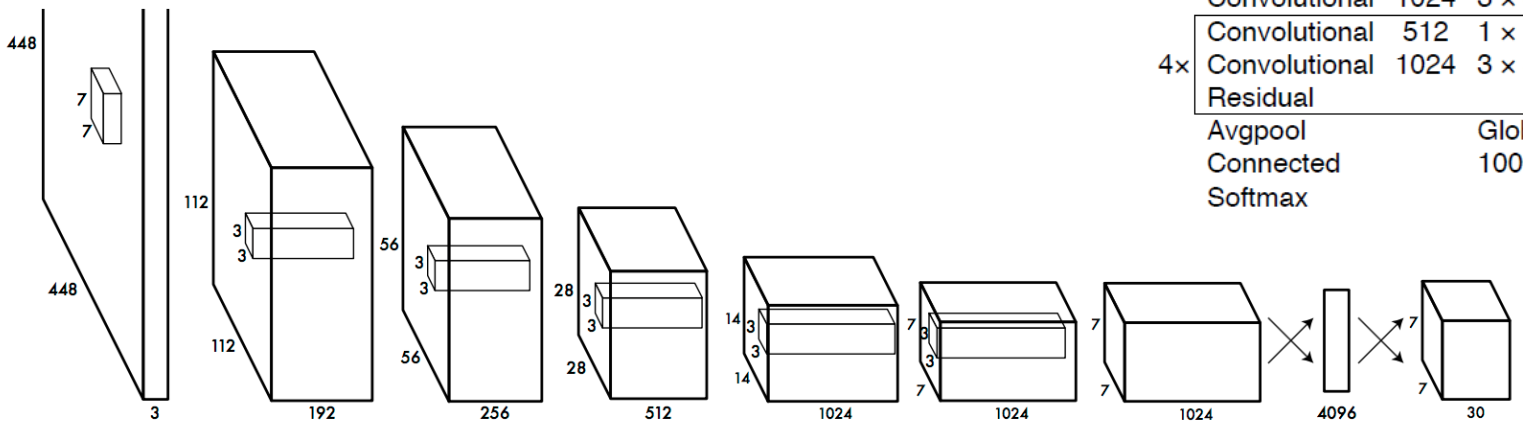

Input
$\mathbf{x}_0$
H_1
BN-ReLU-Conv
$\mathbf{x}_1$
H_2
BN-ReLU-Conv
$\mathbf{x}_2$
H_3
BN-ReLU-Conv
$\mathbf{x}_3$
H_4
BN-ReLU-Conv
$\mathbf{x}_4$
Transition Layer


50 layers cfg=[3,4,6,3]
101 layers cfg=[3,4,23,8]
152 layers cfg=[3,8,36,3]
7x7 conv, 64/2
max pool/2
1x1 conv, 64
3x3 conv, 64
1x1 conv, 256
1x1 conv, 128/2
3x3 conv, 128
1x1 conv, 512
1x1 conv, 128
3x3 conv, 128
1x1 conv, 512
1x1 conv, 256/2
3x3 conv, 256
1x1 conv, 1024
1x1 conv, 256
3x3 conv, 256
1x1 conv, 1024
1x1 conv, 512/2
3x3 conv, 512
1x1 conv, 2048
1x1 conv, 512
3x3 conv, 512
1x1 conv, 2048
avg pool/2
fc ,1000
Size:112
Size:56
cfg[0] blocks
Size:28
cfg[1] blocks
Size:14
cfg[2] blocks
Size:7
cfg[3] blocks

YOLO

v.3

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			



Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2

Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2

Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2

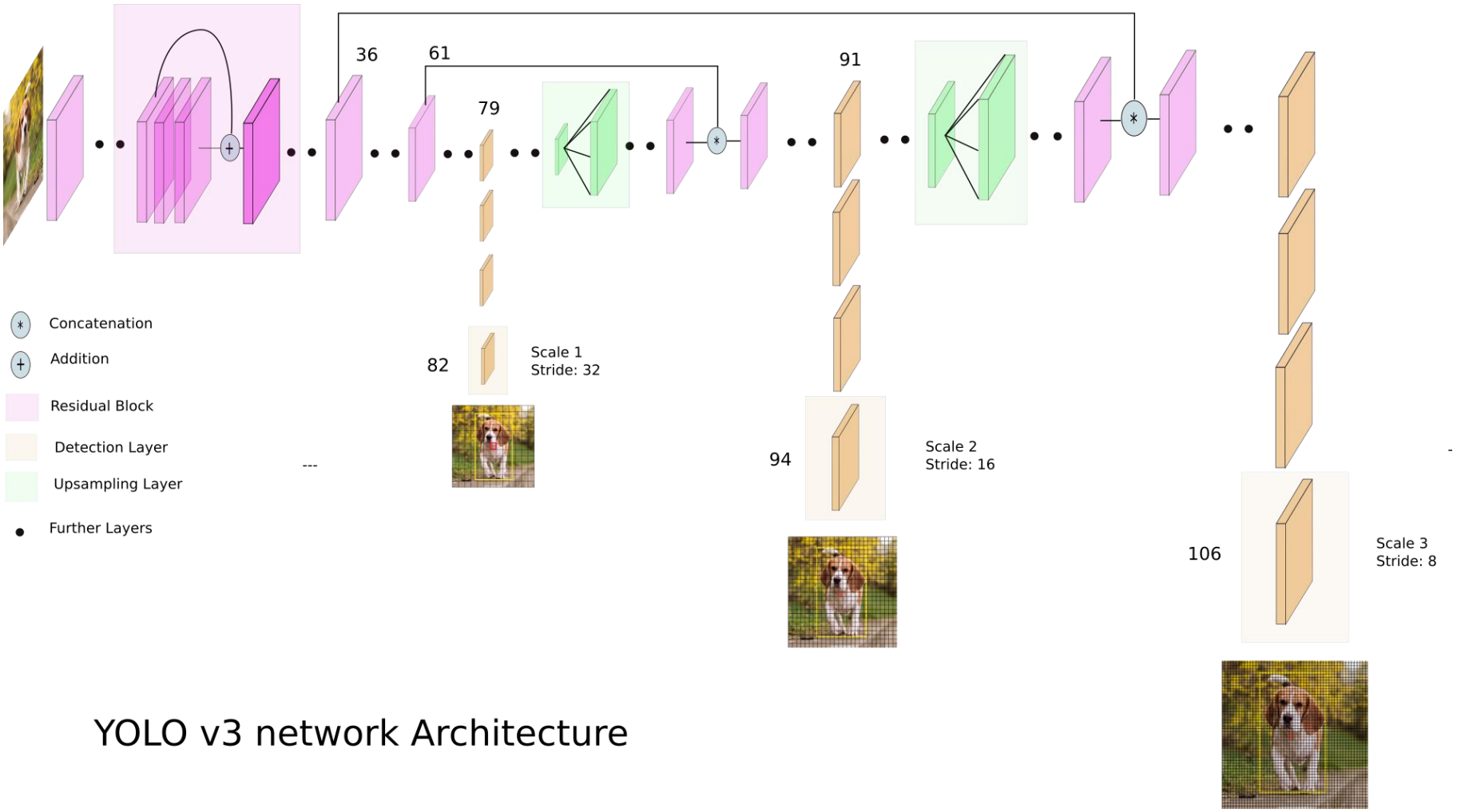
Conv. Layers
1x1x256
3x3x512
} ×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2

Conv. Layers
1x1x512
3x3x1024
} ×2
3x3x1024
3x3x1024-s-2

Conv. Layers
3x3x1024
3x3x1024

Conn. Layer

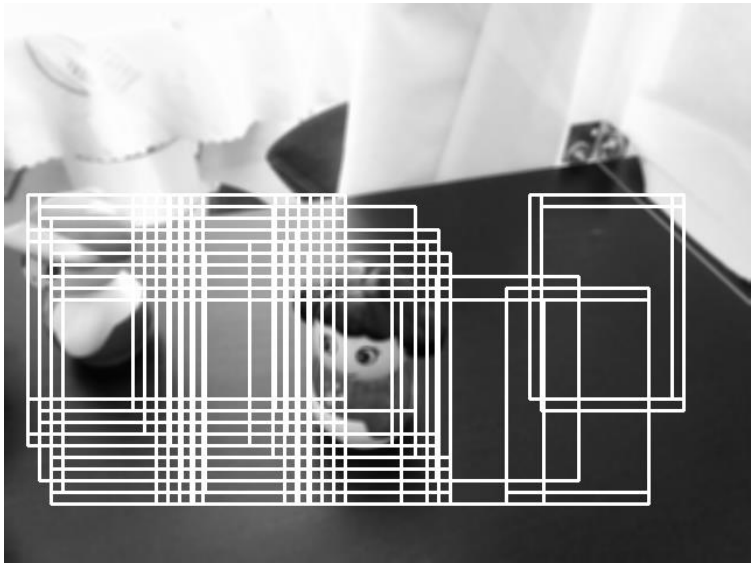
Conn. Layer



YOLO v3 network Architecture

Tradičný detektor

- Jadrom tradičného detektora je klasifikátor, ktorý je spustený na každé možné miesto výskytu objektu v rôznych veľkostiach
- Tento **Sliding window algorithm** je potom zákonite veľmi pomalý



You Only Look Once

- YOLO detektor počíta obrázka priamo obdĺžniky v ktorých sa detekované objekty nachádzajú
- Tieto **Region Proposal Algoritmy** sú oveľa rýchlejšie.
- Existujú aj tradičné Region Proposal Algoritmy ako je Selective Search, YOLO implementuje túto metódu cez Deep Learning

Zdroje

- https://pjreddie.com/media/files/papers/YOLOv3.pdf https://pjreddie.com/darknet/
- https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b
- https://www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/
- https://www.learnopencv.com/training-yolov3-deep-learning-based-custom-object-detector/

Zdroje

www.robotika.sk/seminar/2019/yolo-distrib-win10.zip

www.robotika.sk/seminar/2019/ yolo-distrib-ubuntu.tar.gz

www.robotika.sk/seminar/2019/opencv401.zip

https://github.com/andylucny

Ďakujem za pozornosť!

Detektor objektov YOLO v3
You Only Look Once

Andrej Lúčny

Katedra aplikovanej informatiky FMFI UK

lucny@fmph.uniba.sk

http://dai.fmph.uniba.sk/w/Andrej_Lucny