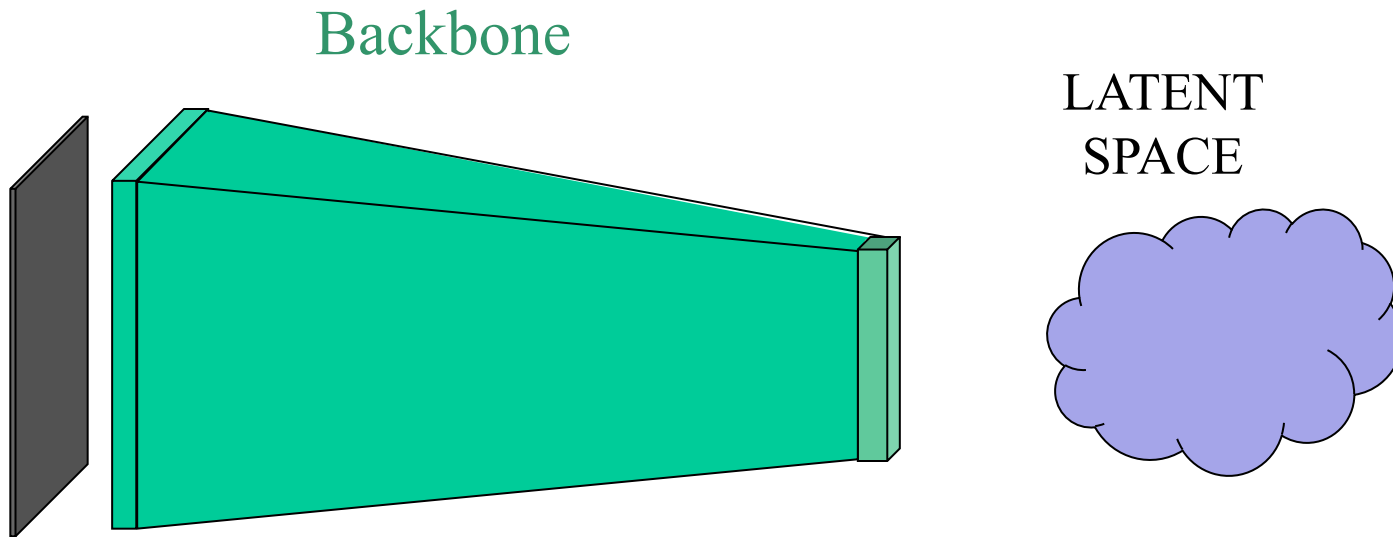


Backbone

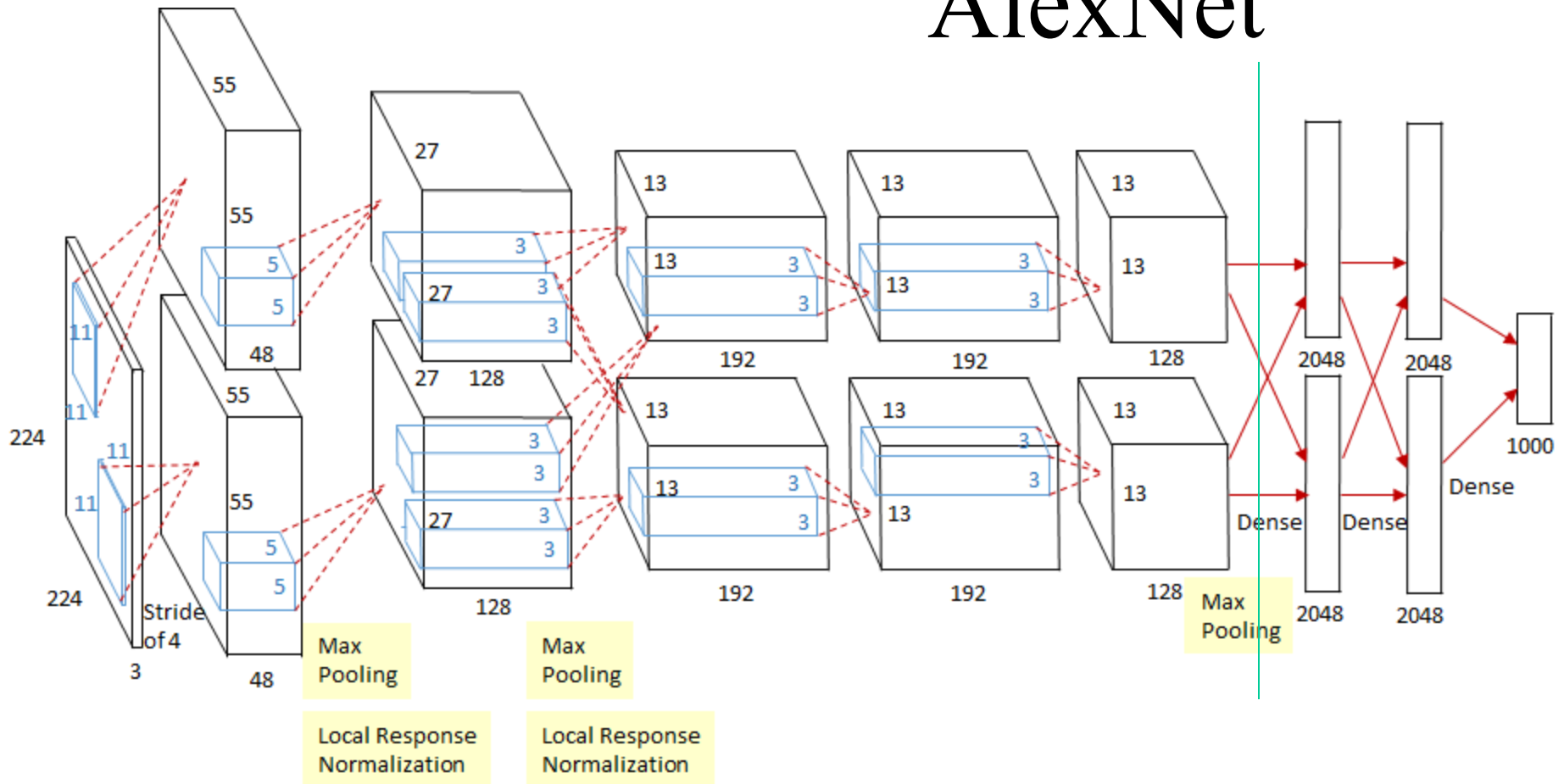


- Časť siete, ktorá obraz premení na príznaky

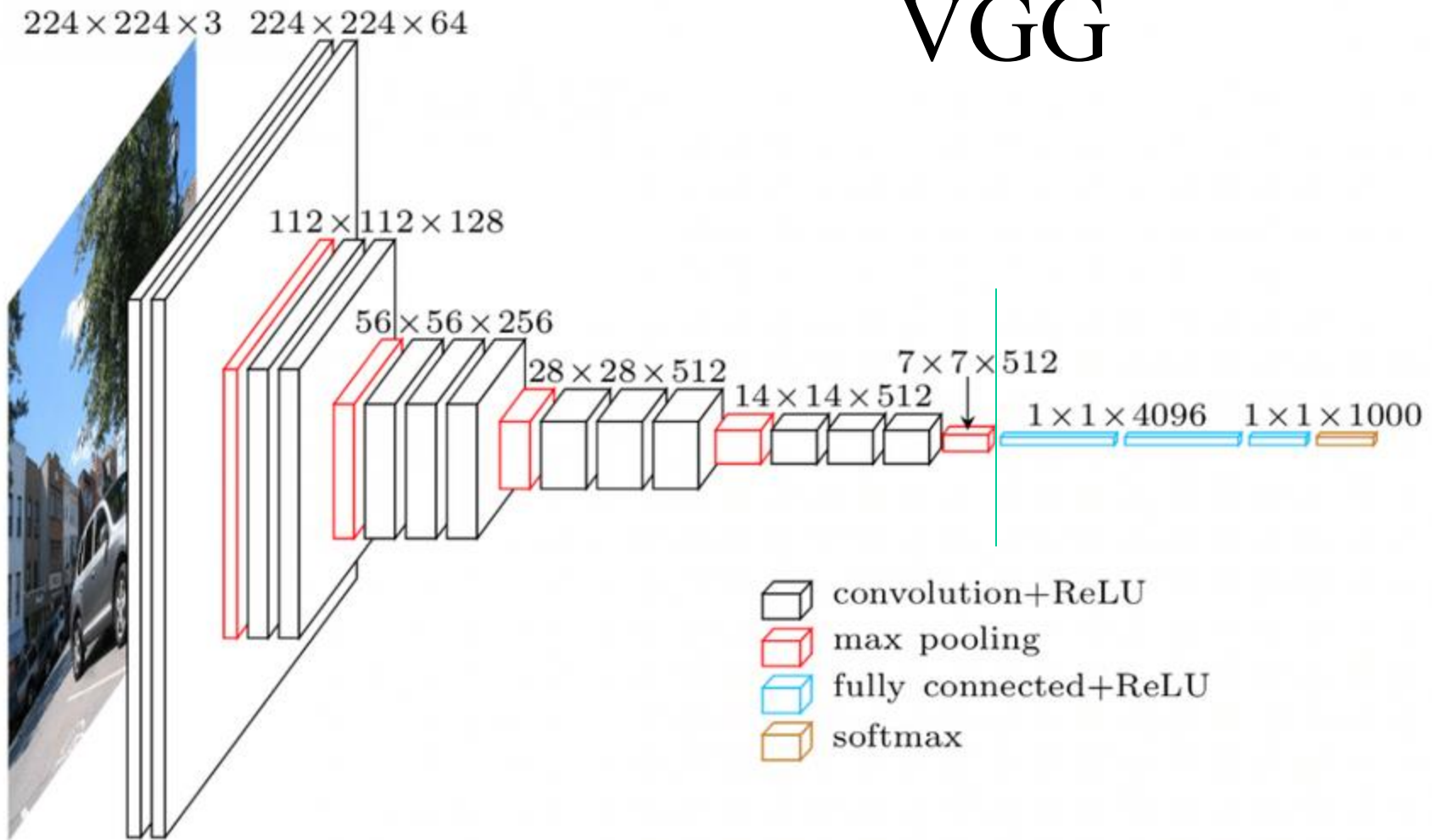
Backbones

- AlexNet
- VGG (VGG16, VGG19)
- ResNet (ResNet50, ResNet18)
- DarkNet
- Inception (Inception v3)
- Xception
- MobileNet

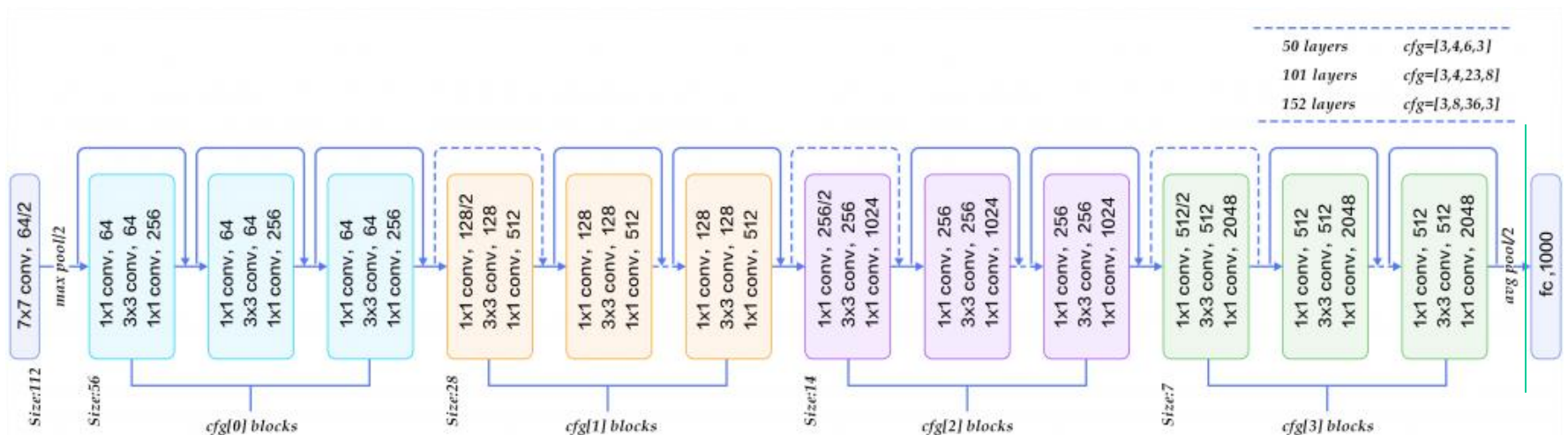
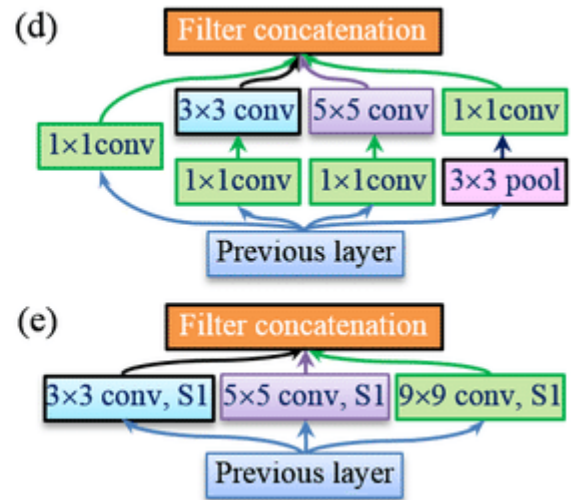
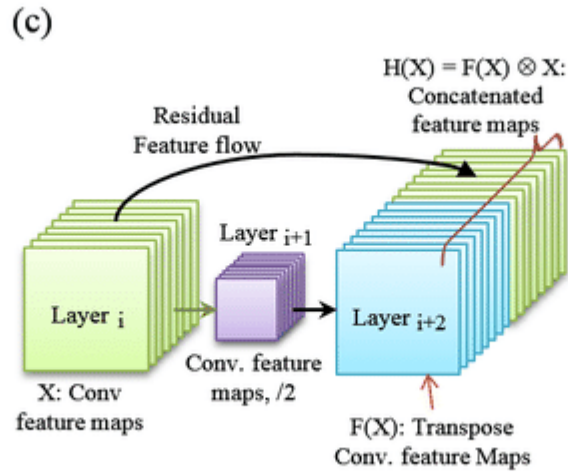
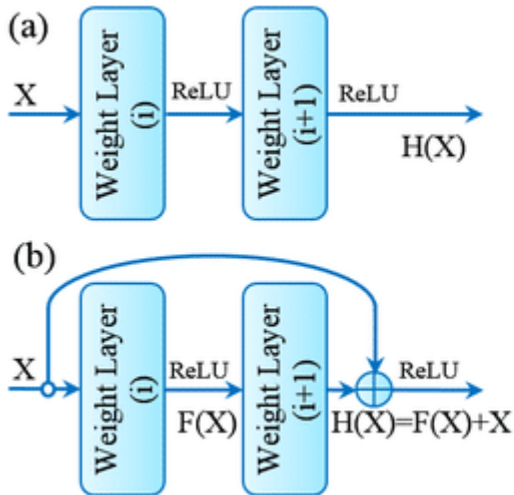
AlexNet



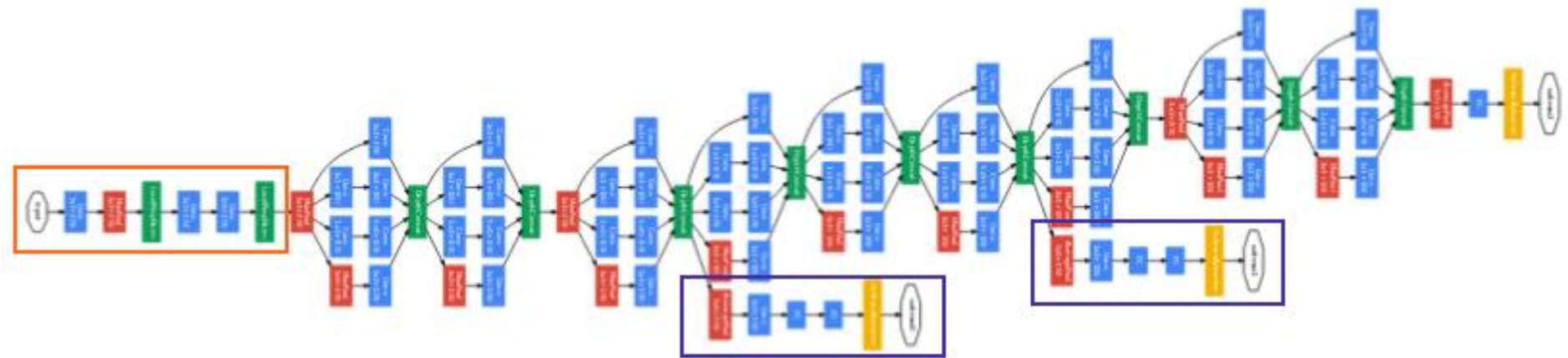
VGG



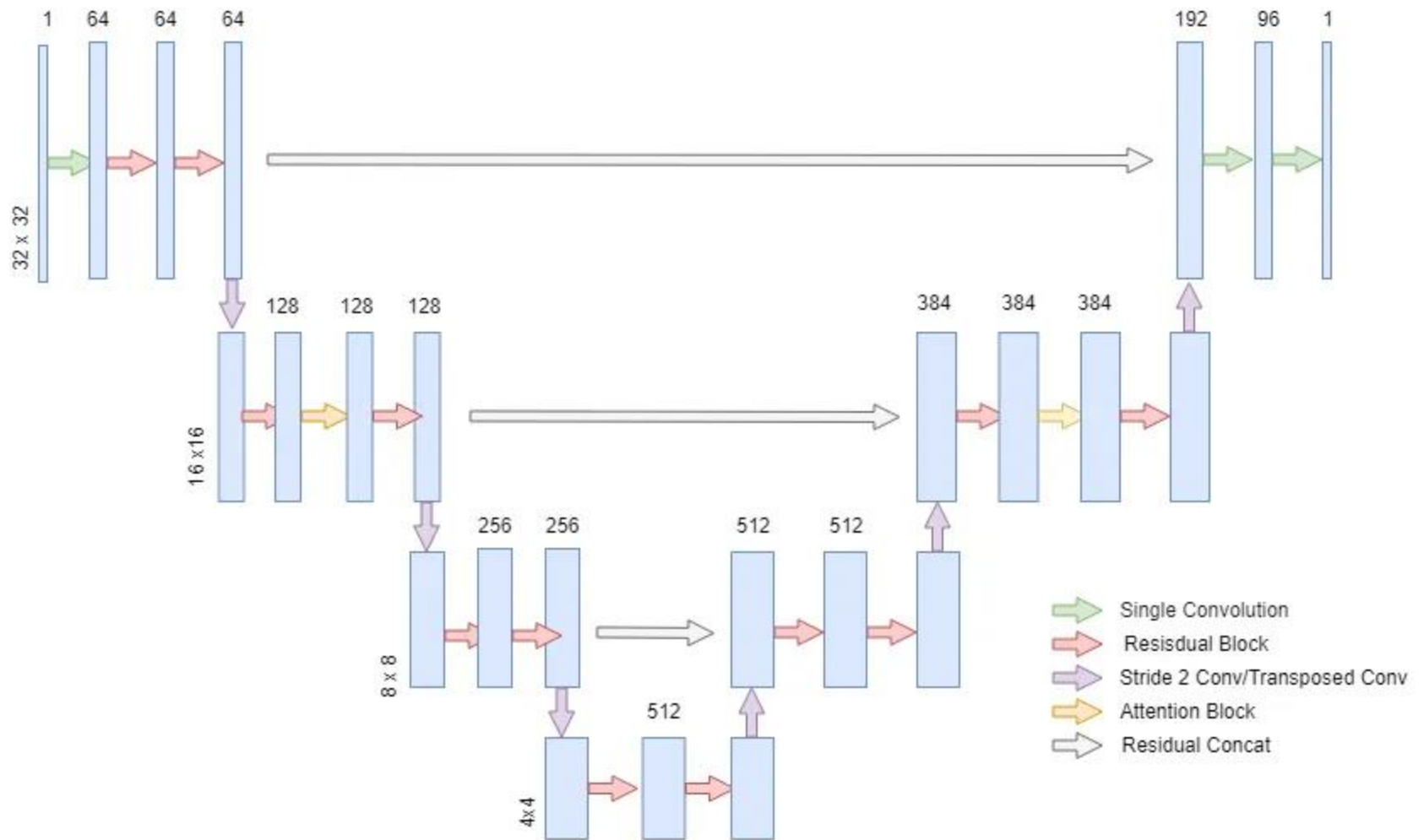
ResNet



Inception V1



U-Net



Predtrénovaný model

- Každá backbone má aj svoje Zoo – zbierku modelov natrénovaných na rôznych datasetoch, napr. na COCO, PascalVOC ...
- Trénovanie siete z predtrénovaného modelu, za ktorý zaradíme svoje (custom) výstupné vrstvy, sa nazýva **transfer learning**

Transfer learning

- Je pravdepodobné, že model na klasifikáciu auta bude v značnej miere podobný modelu pre bicykel
- Preto pri tvorbe druhého môžeme radšej začať od druhého, než na zelenej lúke (from scratch)
- Z modelu natrénovaného pre určitý dataset, napríklad COCO alebo VOC necháme len backbone a ten budeme distribuovať ako **predtrénovaný model**

Fine tuning

- K predtrénovanému modelu pripojíme napr perceptron a trénujeme už len váhy perceptrónu, aby sme získali napr. custom classifier
- Takéto trénovanie je aj oveľa rýchlejšie. Opiera sa však len o hotové features
- Má zmysel preto skúsiť pokračovať v trénovaní, pričom už dovolíme meniť aj váhy backbone. Táto fáza trénovania sa nazýva Fine tuning (jemné doladenie)

Trénovanie hlbokej siete

- Trénovanie hlbokej siete je problematické
- Čím hlbšia je sieť, tým viac hrozí, že sa gradient pri spätnom šírení stratí a váhy v úvodných vrstvách sa vôbec nebudú meniť (vanishing gradient problem)
- Práve riešenie tohto problému predstavuje hlavné rozdiely medzi backbonami (okrem počtu vrstiev, počtu parametrov, rozmeru vstupu a podobne)

Riešenia

- Dropout
- Batch normalization
- Reziduály

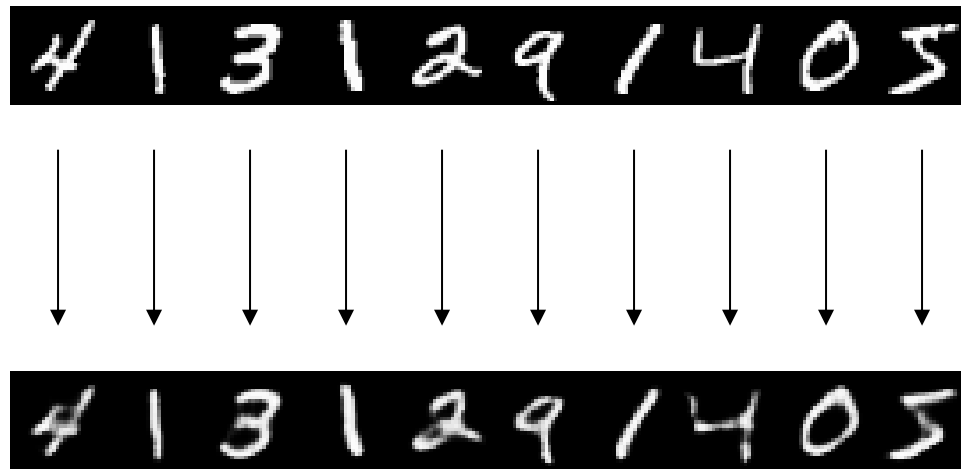
Dropout

$$\textit{drop}(x) = \begin{cases} 0 & \text{ak } rnd < p \\ x \frac{1}{1-p} & \text{inak} \end{cases}$$

ak $p = 0.2$ 20% vynulujeme

80% zosilníme

- Tento problém a jeho riešenia dobre vidno na trénoch identity
- Na rozdiel od autoencodera však neredukujeme dimenziu, nemáme latentný priestor



Dataset MNIST



conv



conv

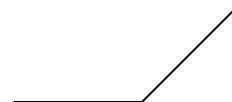


conv



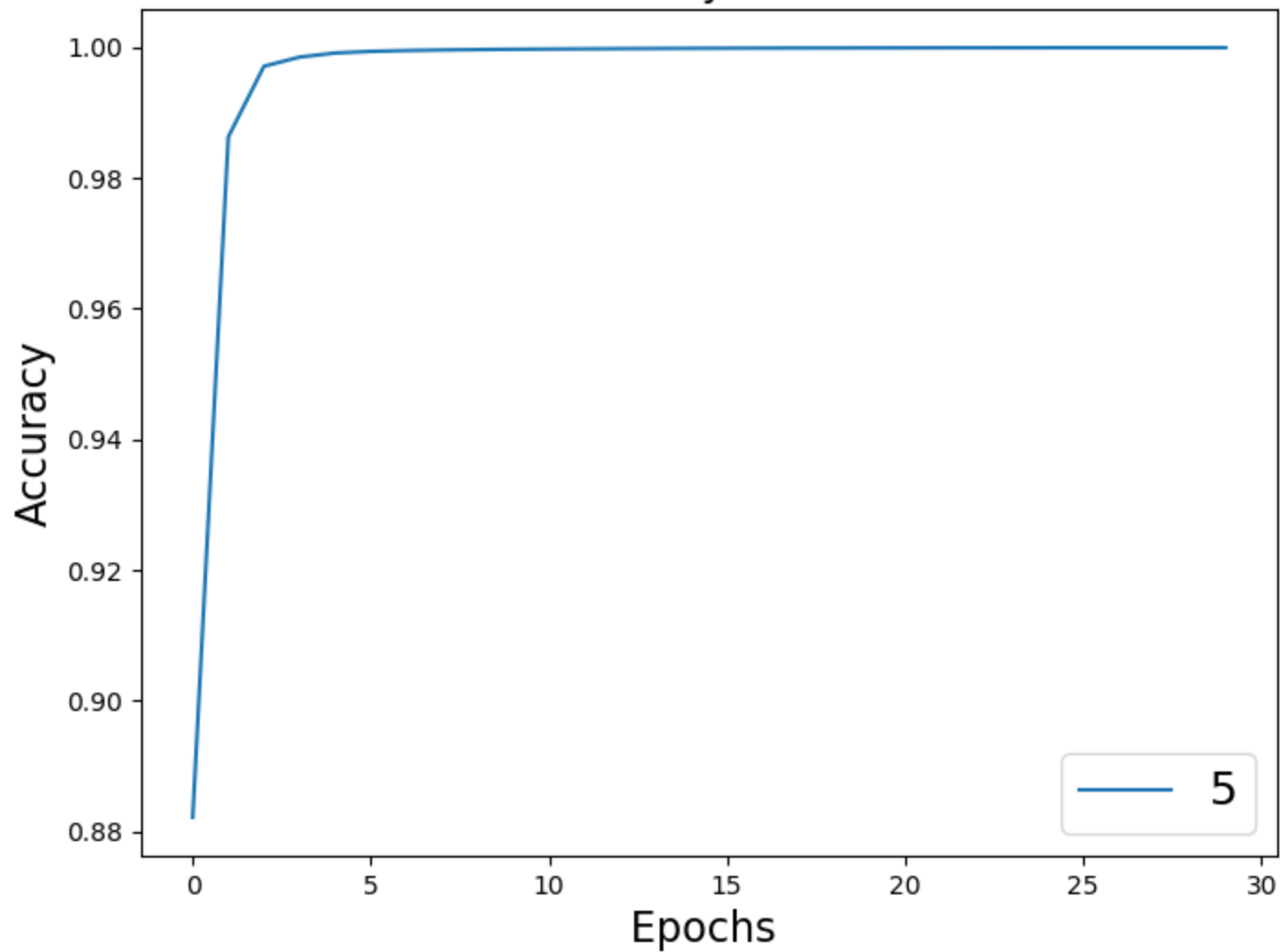
conv

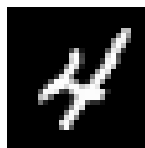
⋮



všade: ReLU

Accuracy Curves





conv



dropout



conv

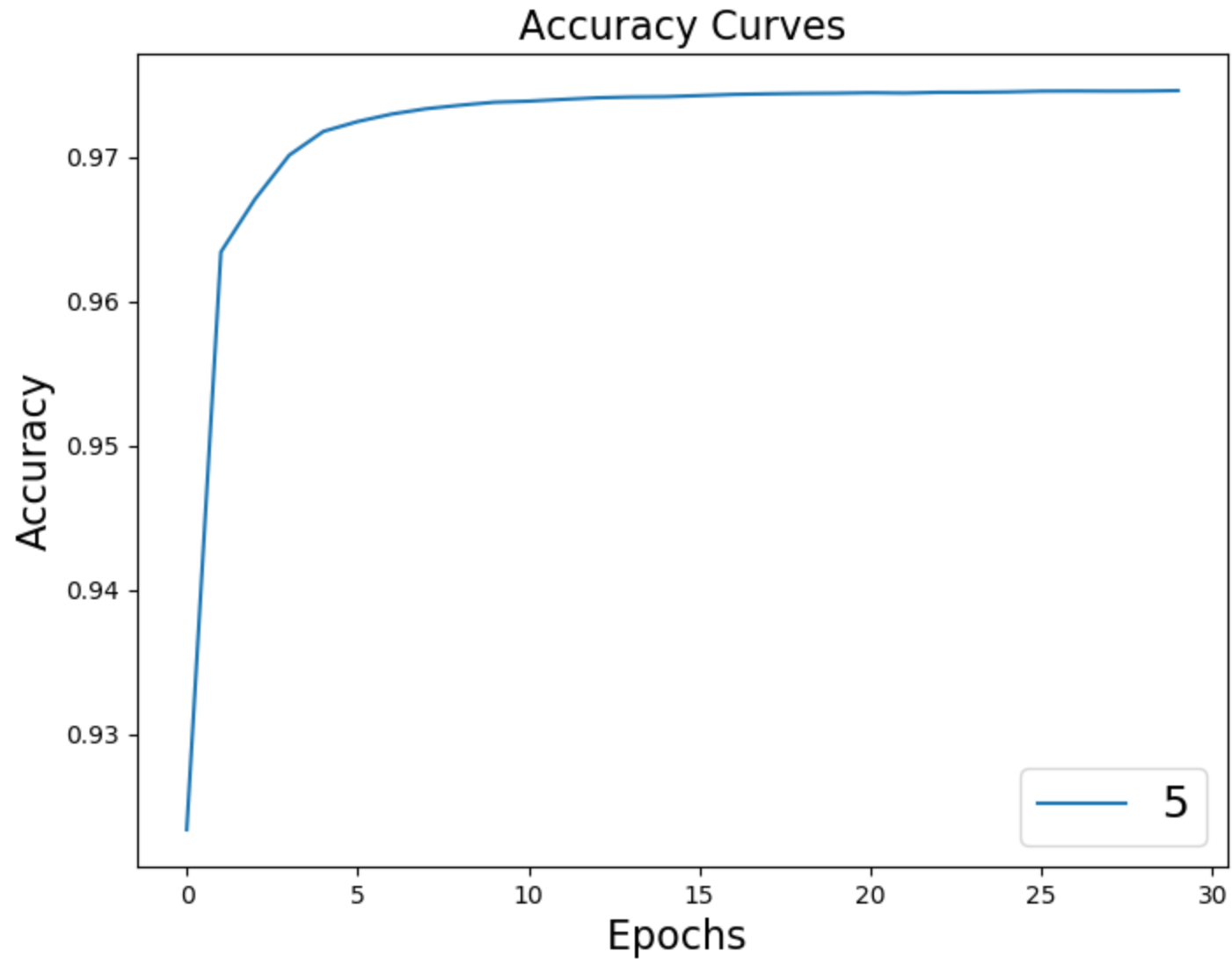


dropout

⋮



- Dropout



Batch normalization

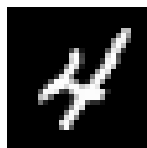
trénovanie

$$bn(x) = \gamma \frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

inferencia

$$bn(x) = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

$$\mu = \sum_{i=1}^t (1 - \alpha)^{t-i} \mu_{B_i} \qquad \sigma^2 = \sum_{i=1}^t (1 - \alpha)^{t-i} \sigma_{B_i}^2$$



conv



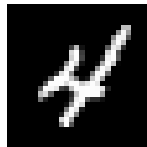
Batch normalization



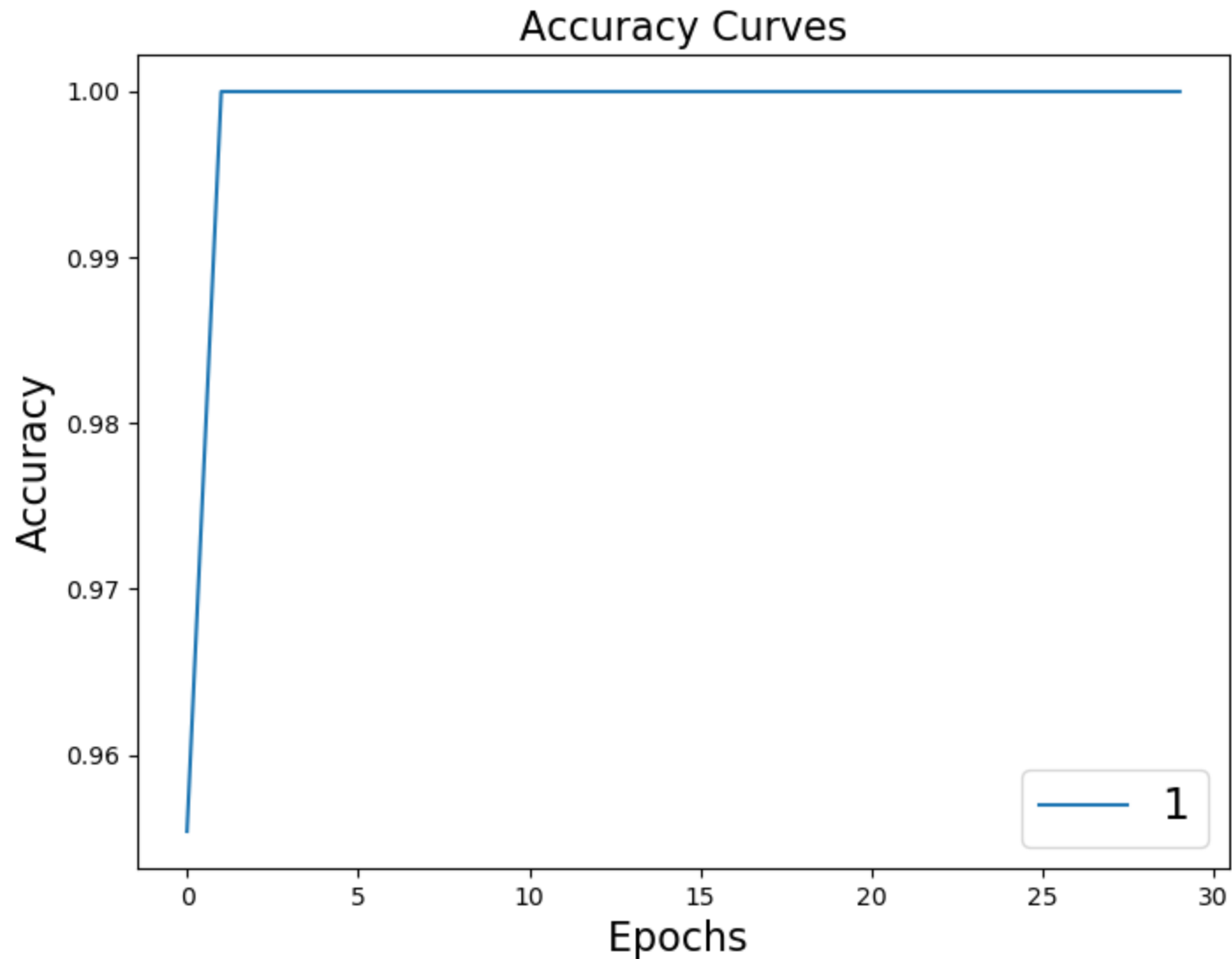
conv



Batch normalization

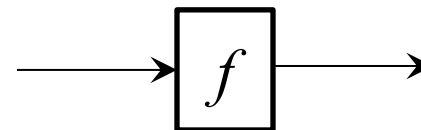


- Batch normalization

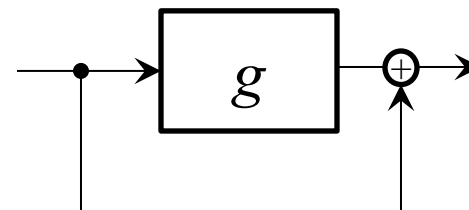


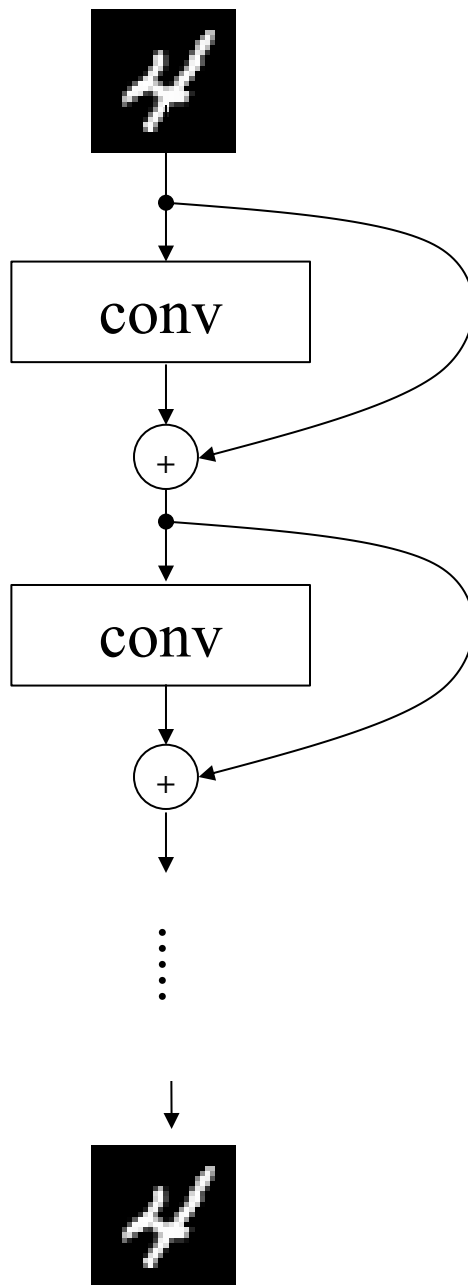
Reziduálne spojenie

Nehľadáme $f(x)$

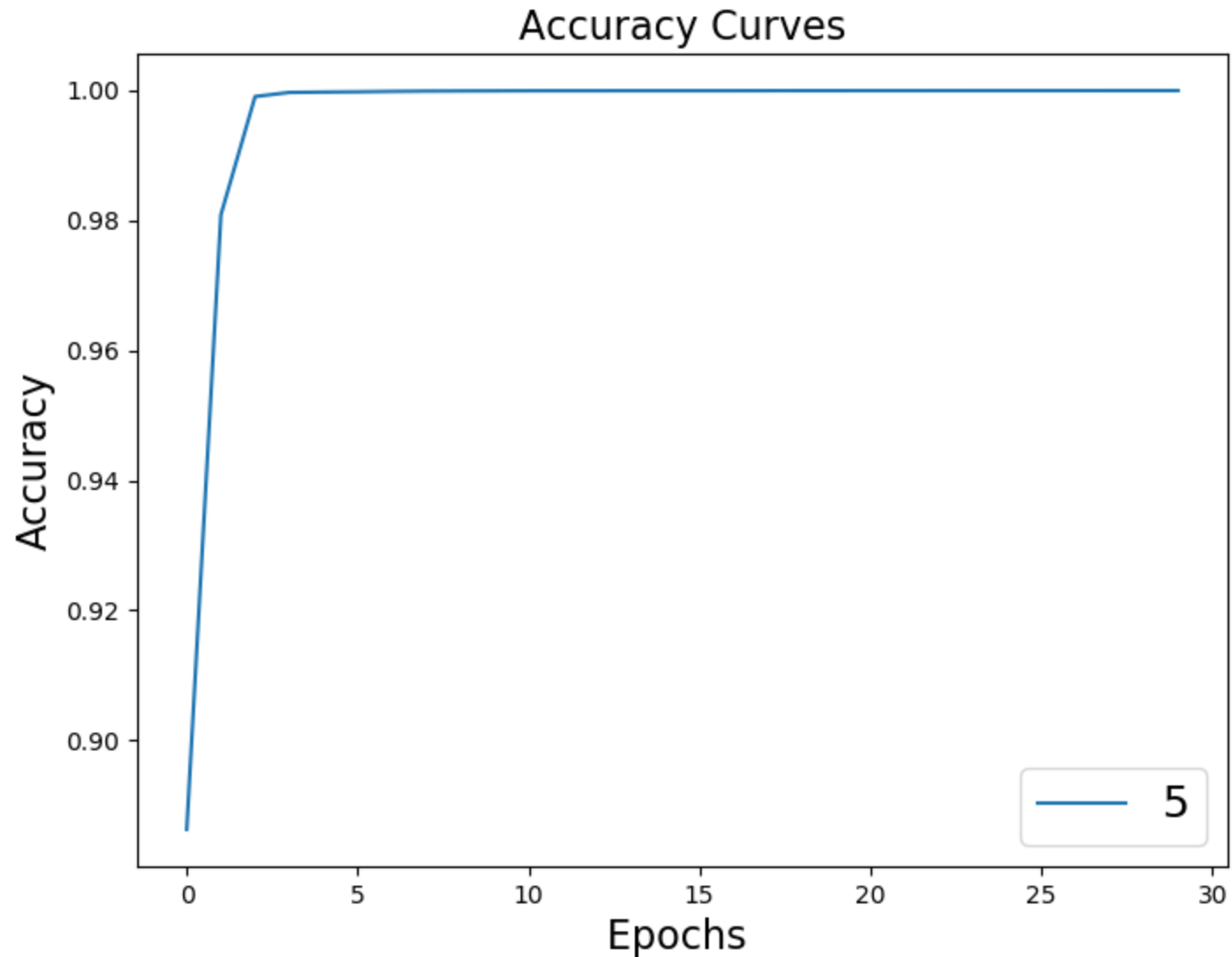


Ale doplnok $g := f(x) - x$





- Reziduály



Augmentation

- Anotovať dataset je namáhavé, takže hľadáme cesty ako príklady z datasety rozmnožiť.
- Napríklad v datasete tváří, nám vertikálny flip de facto zdvojnásobí dataset, lebo tváre nie sú úplne súmerné
- Operácie ktoré vyrábajú z jednej vzorky ďalšie (zväčšením, zmenšením, pootočením, posunutím, prevrátením, ...) nazývame augmentation

Sliding window algorithm

Každý klasifikátor sa dá premeniť hrubou silou na detektor pomocou algoritmu používajúceho plávajúce okno:

- otestujeme všetky možné výskyty
- zosumarizujeme výskyty cez Non-Maximum Suppression

Detektor

Urobíme ho ako klasifikátor ktorý sa bude
bude pozerat' na rôzne miesta na obraze a
klasifikovat' tam kúsky hľadaných objektov
(zameraním sa na kúsky si zároveň zväčšujeme
dataset)

Implementujeme ho ako perceptron

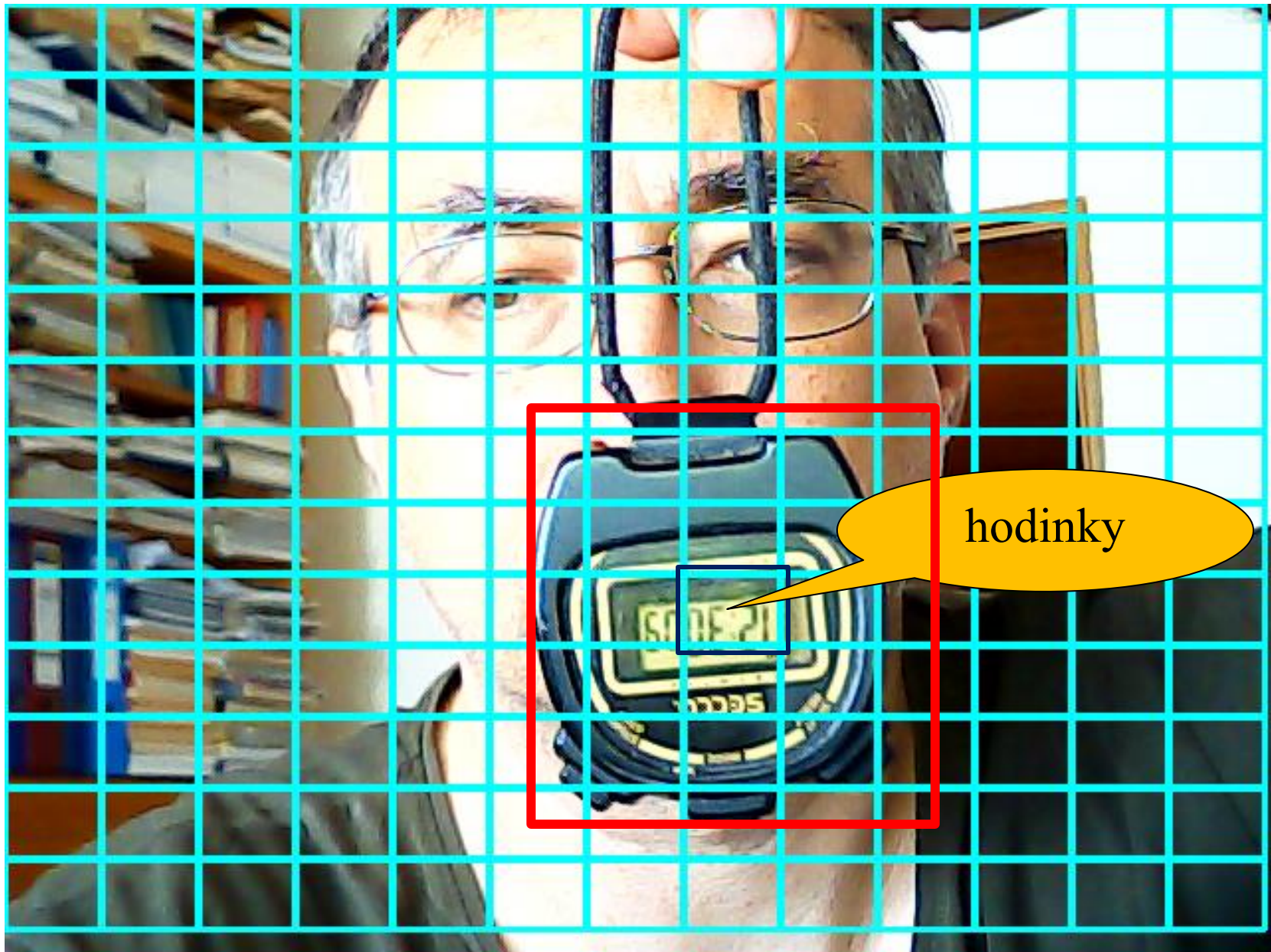
To vieme urobiť pomocou dvoch či viacerých
blokov konvolučných vrstiev

YOLO Detektor (You Look Only Once)

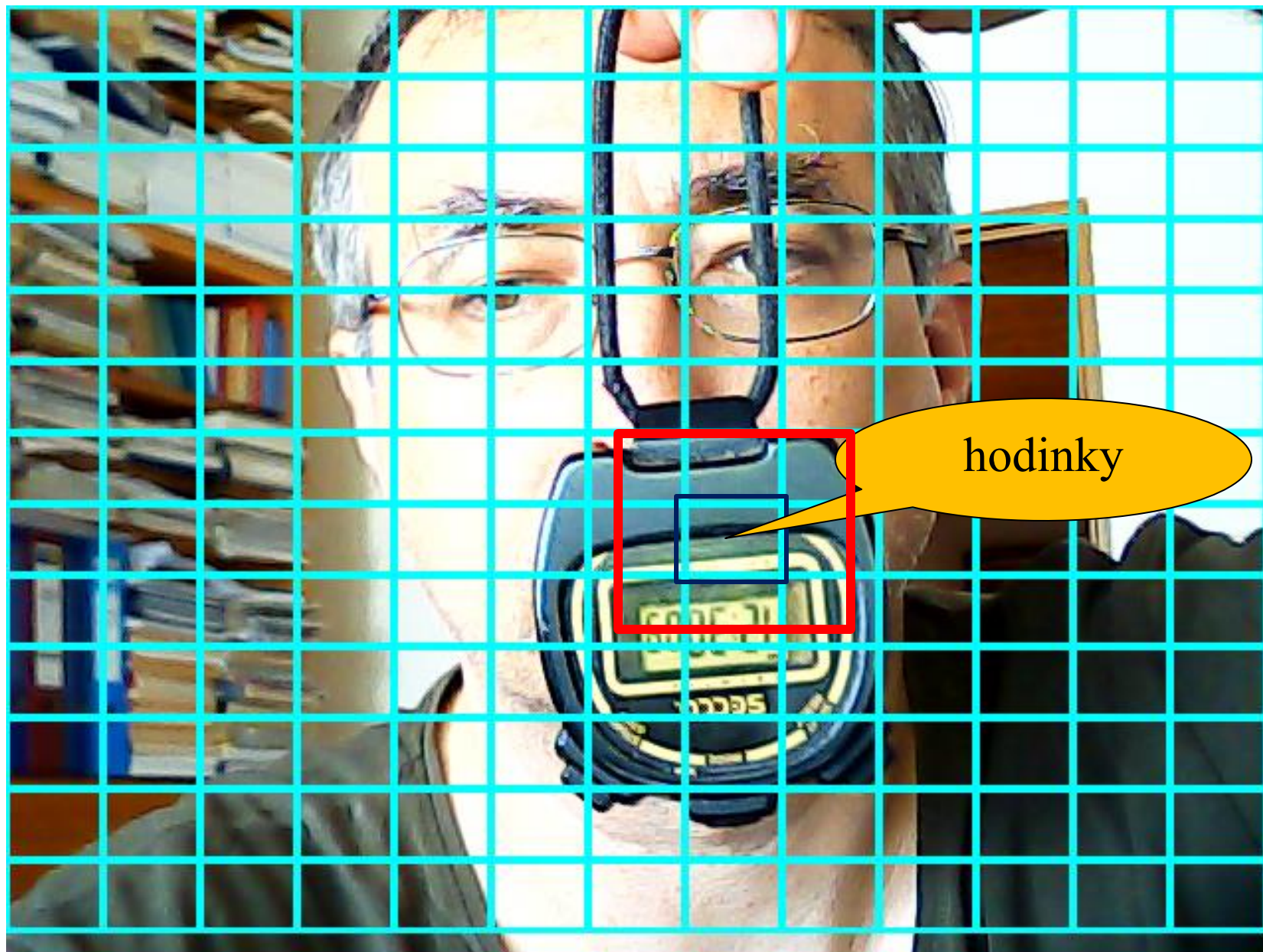
- obraz pokryjeme neprekrývajúcimi sa regiónmi
- nad každým regiónom vykonáme klasifikáciu a regresiu a zosumarizujeme výsledky



- **Klasifikátor** dáva pravdepodobnosť, že región patrí k detekovanému objektu
- **Regressor** dáva obdĺžnik obsahujúci objekt, ktorého je región súčasťou



hodinky

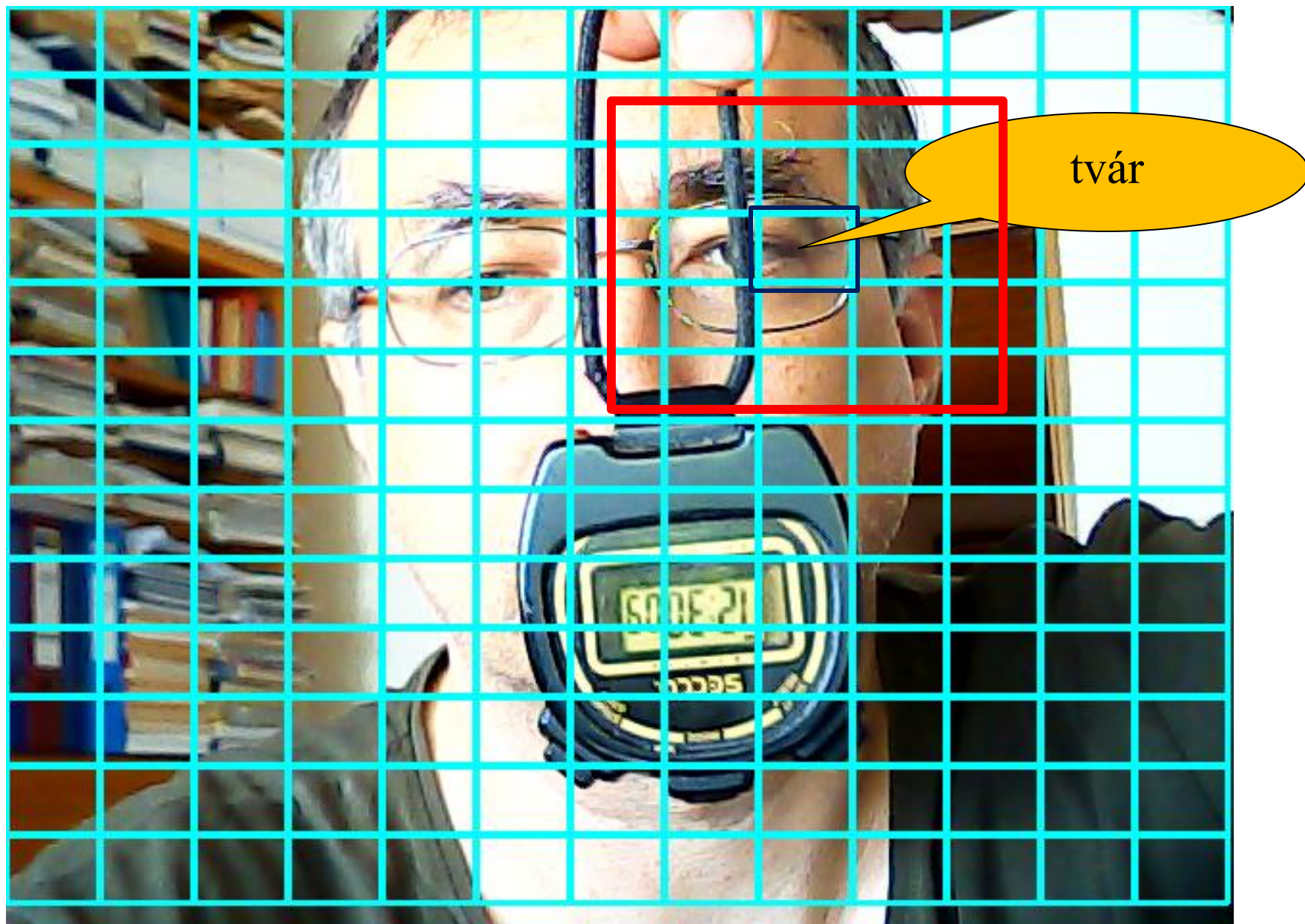




hodinky

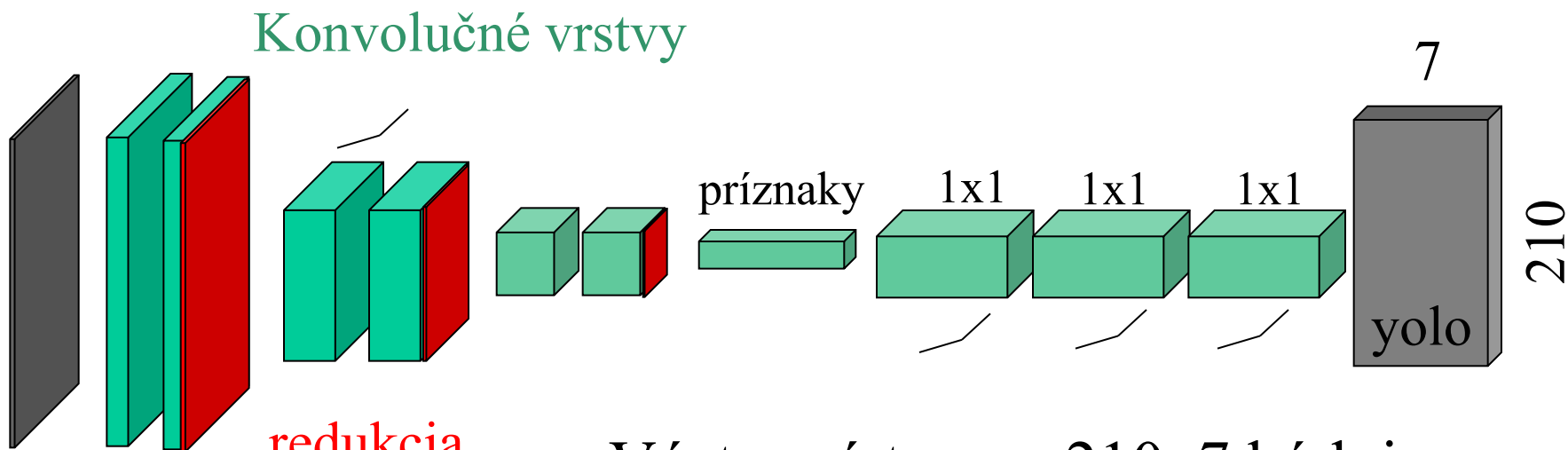


tvár



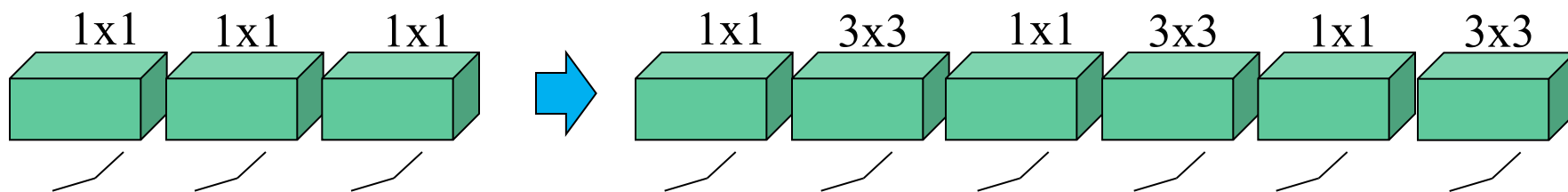
tvár

YOLO v1

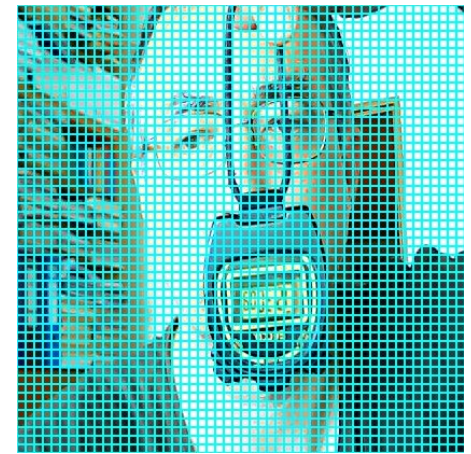
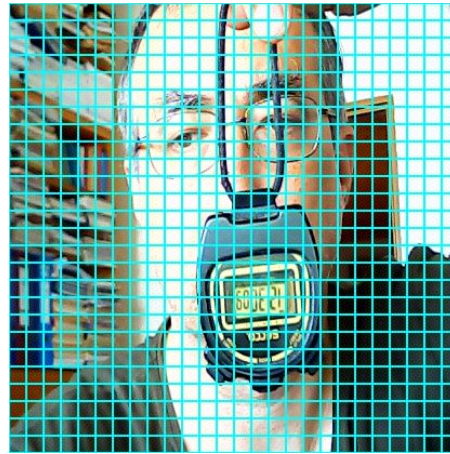


Výstupný tenzor 210x7 kóduje maximálne 210 detekcií, pričom každá obsahuje: relevantnosť, kategóriu, spoľahlivosť, x, y, w, a h (sumarizáciu vykonáva špeciálny stavebný prvok zvaný yolo)

- Ak máme kúsok hodínok v určitom regióne, zvyšuje ho pravdepodobnosť, že je aj v susedných regiónoch.
- Ako by sme mohli dosiahnuť, že paralelne pustené perceptrony budú kooperovať?
- Riešenie: Kernely 1×1 preložíme kernelmi 3×3

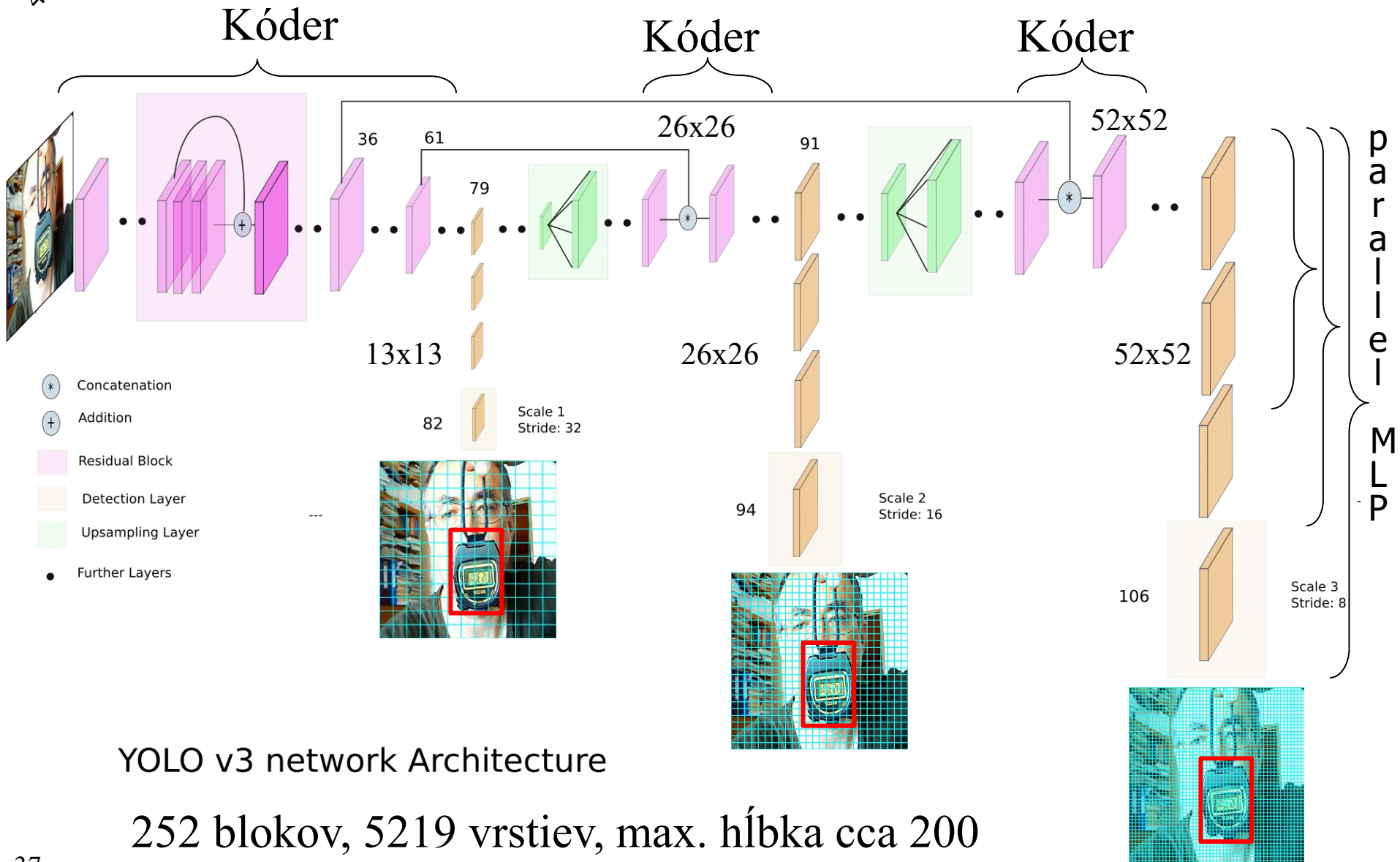


- Ako by sme mohli zohľadniť rôznu veľkosť objektov?
- Pustíme celé spracovanie paralelne viac krát pre 3×13 , 26×26 , a 52×52 regiónov



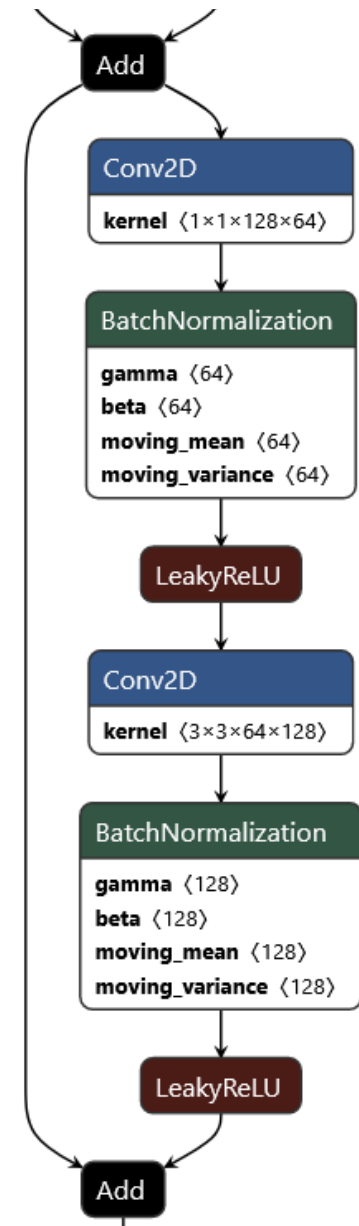
- V takom prípade aj menej detailné spracovanie môže radiť detailnejšiemu

YOLO v3 [Redmon, Farhadi 2018]

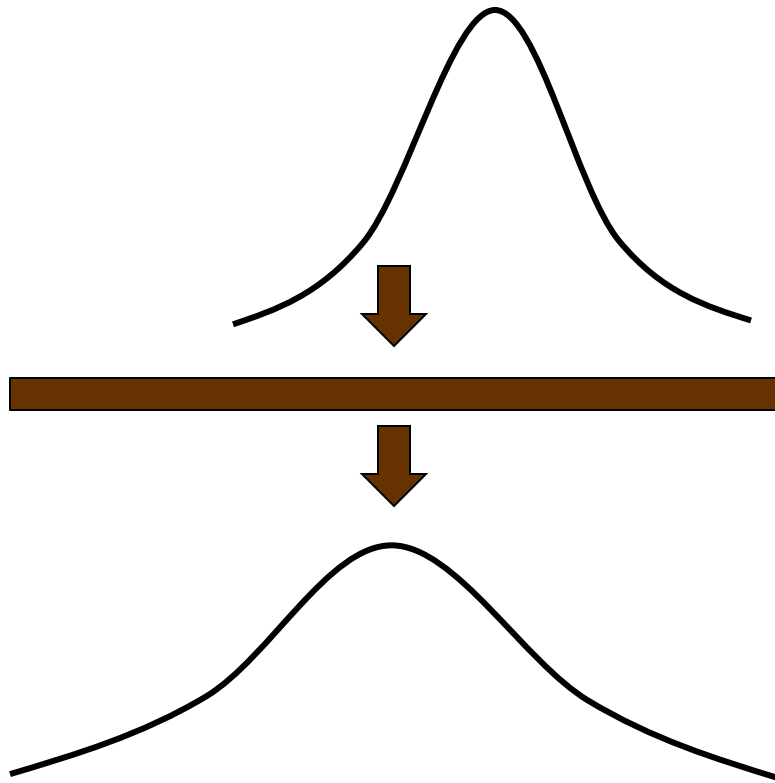


Trénovanie hlbokých sietí

- Hlboké siete dokážu mať sofistikovanú architektúru, ale trénujú sa oveľa ťažšie (problém miznúceho gradientu pri spätnom šírení)
- Preto do takýchto sietí vkladáme stavebné prvky, ktoré trénovanie podporujú. YOLO v3 využíva reziduálne spojenia



Normalizácia dávky



- Aby bolo tréňovanie dostatočne rýchle, hlboké siete používajú sigmoidálne aktivácie len na výstupe a vnútri používajú ReLU
- ReLU ale nie je sigmoidálna a nedá sa s ňou implementovať aproximátor
- Preto pred (alebo za) ReLU vkladáme normalizáciu dávky

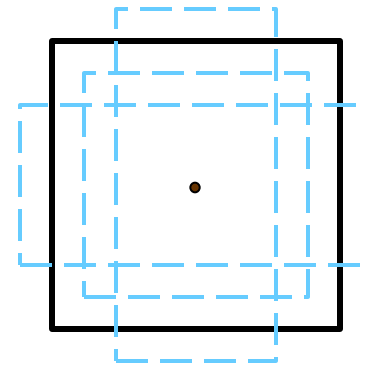
YOLO v3 schéma



YOLO v3 - výstup

- $13 \times 13 \times 255$, $26 \times 26 \times 255$, $52 \times 52 \times 255$
- Každý perceptron dáva tri detekcie,
 $255 \text{ hodnôt} = 3 \times 85$
- Každá detekcia dáva $85 \text{ hodnôt} = 4 + 1 + 80$
obsahujúcich:
 - 4 ... x, y, w, h
 - 1 ... spoľahlivosť
 - 80 ... pravdepodobnosť príslušnosti
k jednej z 80 kategórii

Ukotvenia



- Každá detekcia vyjadruje x, y, w , a h relatívne k určitému ukotveniu. YOLO v3 využíva tri ukotvenia pre každú z troch veľkostí objektov (dohromady deväť), každé s rôznym pomerom strán
- Ukotvenia v YOLO v3:
[116x90, 156x198, 373x326] (výstup 52x52)
[30x61, 62x45, 59x119] (výstup 26x26)
[10x13, 16x30, 33x23] (výstup 13x13)

Transférové učenie

YOLO v3 poskytuje predtrénovaný model pre 80 kategórii z dátovej vzorky COCO:

person, bicycle, car, motorbike, aeroplane, bus, train, truck, boat, traffic light, fire hydrant, stop sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear, zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports, ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donut, cake, chair, sofa, pottedplant, bed, diningtable, toilet, tvmonitor, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair drier, toothbrush

Pri tréňovaní pre vlastné dátové vzorky, nezačínáme s náhodnými parametrami, ale parametrami predtrénovaného modelu.

Trénovanie YOLO v3 <https://youtu.be/CPGR26Fhcro>



<https://youtu.be/ENYEUUbqjIE>

Spracovanie videa

