2D konvolúciós neurális hálózatok

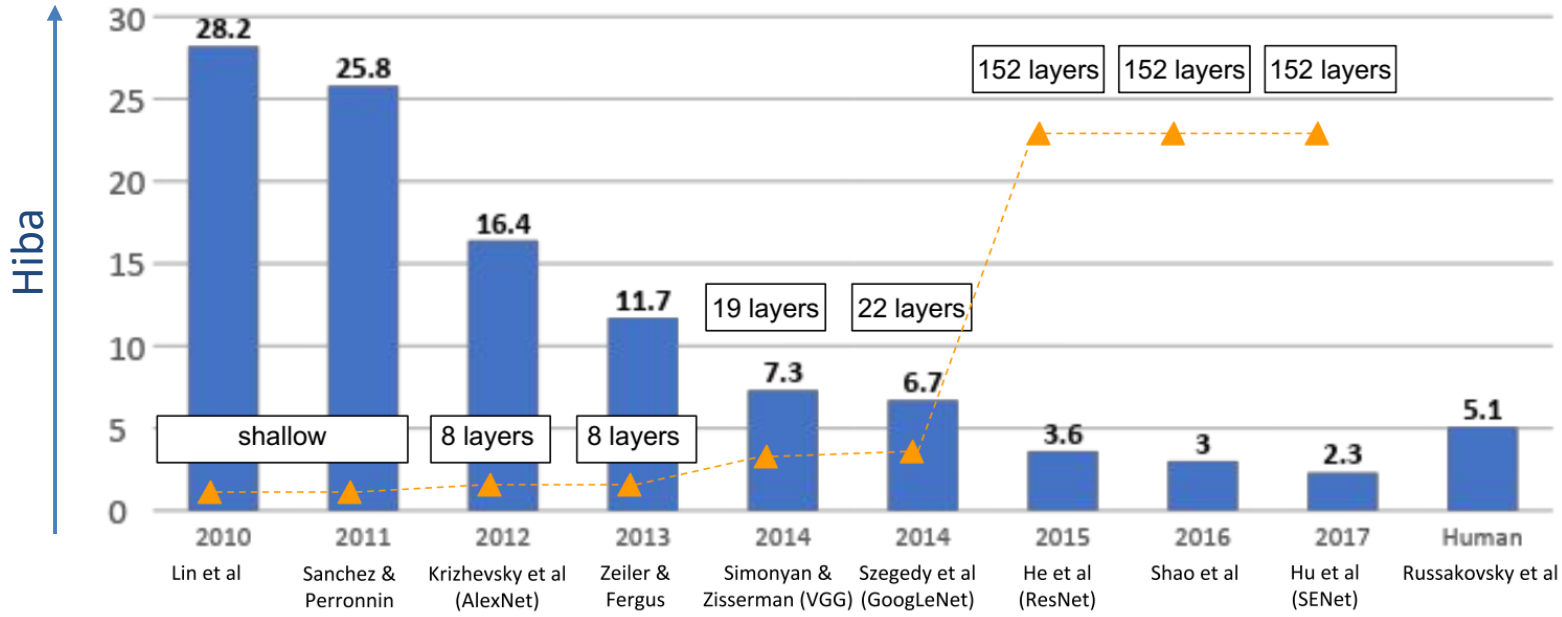
Neurális hálózatok – 21-22/2

Motiváció

- Alkalmazási feladatai:
  - Pixel szintű szegmentáció
  - Kép osztályozása, rajta szereplő objektum lokalizációja
  - Objektumok detektálása
  - Objektumok pixel szintű szegmentálása
- Klasszikus megközelítés (CNN-ek előtt):
  - Hibrid intelligens, szakértői rendszerekkel
  - Diszkriminatív szűrések (ROI / osztályozáshoz szükséges jellemzők kiemeléséhez) tervezése nehéz, esetleges
- Jelentősen jobb pontosság, pár buktatóval

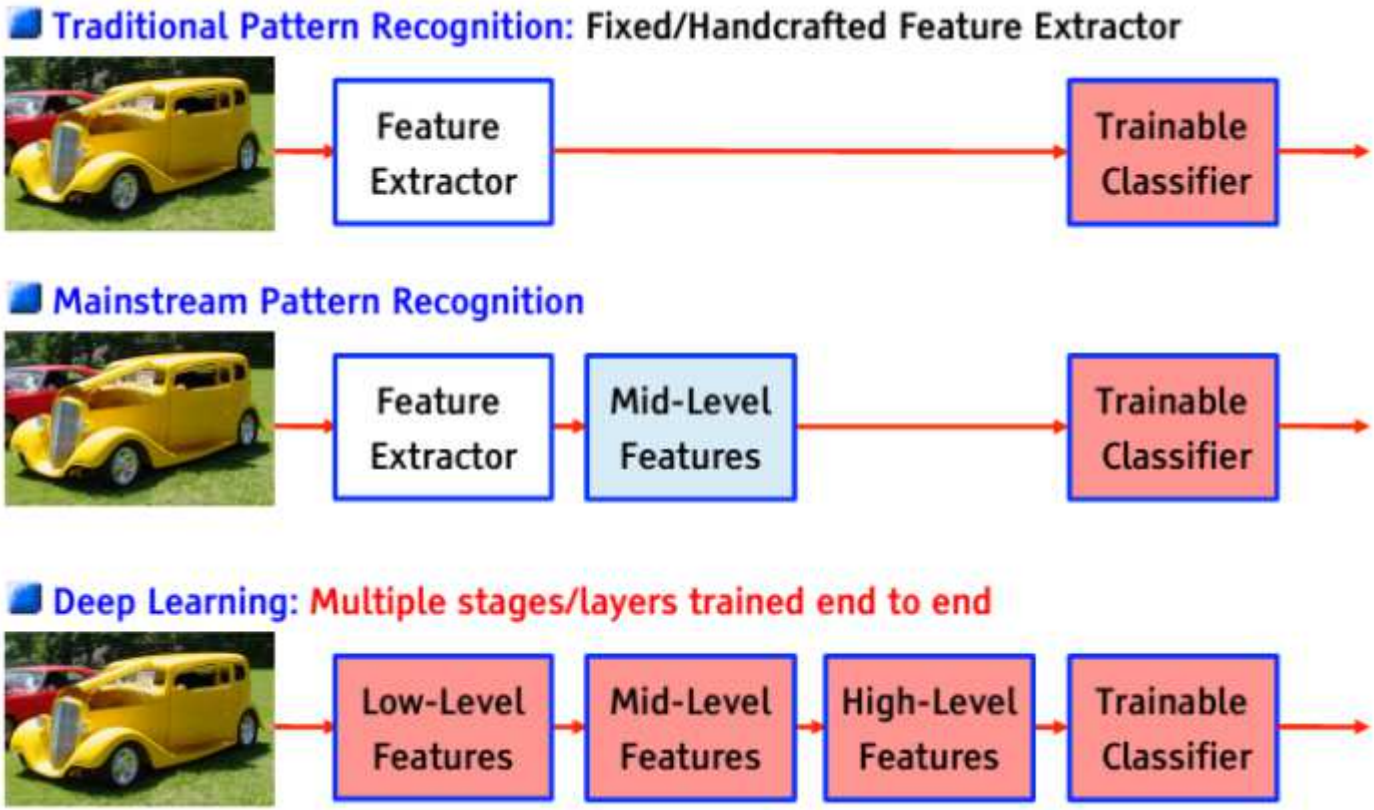
Motiváció

- State of the art eredmények:
  - Képfeldolgozási feladatok nehezen algoritmizálhatóak, de emberek számára nem jelentenek érdemi kihívást
  - ImageNet versenyen elért eredmények – 2012-nél jelent meg az első 2D CNN



Motiváció

- Képi objektumdetektáló megközelítések:



ÚJ ARCHITEKTURÁLIS ELEMEK

Konvolúciós réteg

- Motiváció:
  - Klasszikus képfeldolgozásnál alapművelet a konvolúció:
    - Zajszűrésre
    - Alacsony képi jellemzők kiemelésére (pl. élek, sarokpontok)
    - Összetett objektumok kiemelése (pl. illesztett szűrés)
  - Konvolúció ≡ eltolás invariáns, lineáris művelet:
    - Egy objektum képi megjelenése független a helyzetétől
    - Ezért egy objektumot mindenhol u.ú. keresünk a képen
  - Teljesen összekötött hálókhoz képest jóval kevesebb szabad paraméter

Konvolúciós réteg

- Definíció:

$$o_{(z)}^{(l)}(x,y) = \sum_{c} \sum_{(a,b)} y'^{(l-1)}_{(c)}(x \cdot s - a \cdot d, y \cdot s - b \cdot d) \cdot w^{(l)}(a,b,c,z) + bias_{(z)}^{(l)}$$

  - $o_{(z)}^{(l)}$: *l*-edik réteg *z*-edik neuronjának súlyozott összegképe (rövidebben *l*-edik réteg *z*-edik csatornája), pixelenként erre jön a nemlinearitás
  - $y'^{(l-1)}_{(c)}$: *l-1.* réteg *c.* csatornájának paddelt változata (szokás aktivációs térképnek is hívni)
  - s : lépésköz, d: dilatáció
  - Tanult paraméterek:
    - $w^{(l)}(a,b,c,z)$: *l.* réteg súlya a *c.* és a *z.* csatorna között
    - $bias_{(z)}^{(l)}$: *l.* réteg *z.* csatornájának eltolása

Konvolúciós réteg

- Definíció:

$$o_{(z)}^{(l)}(x,y) = \sum_{c} \sum_{(a,b)} y'^{(l-1)}_{(c)}(x \cdot s - a \cdot d, y \cdot s - b \cdot d) \cdot w^{(l)}(a,b,c,z) + bias_{(z)}^{(l)}$$

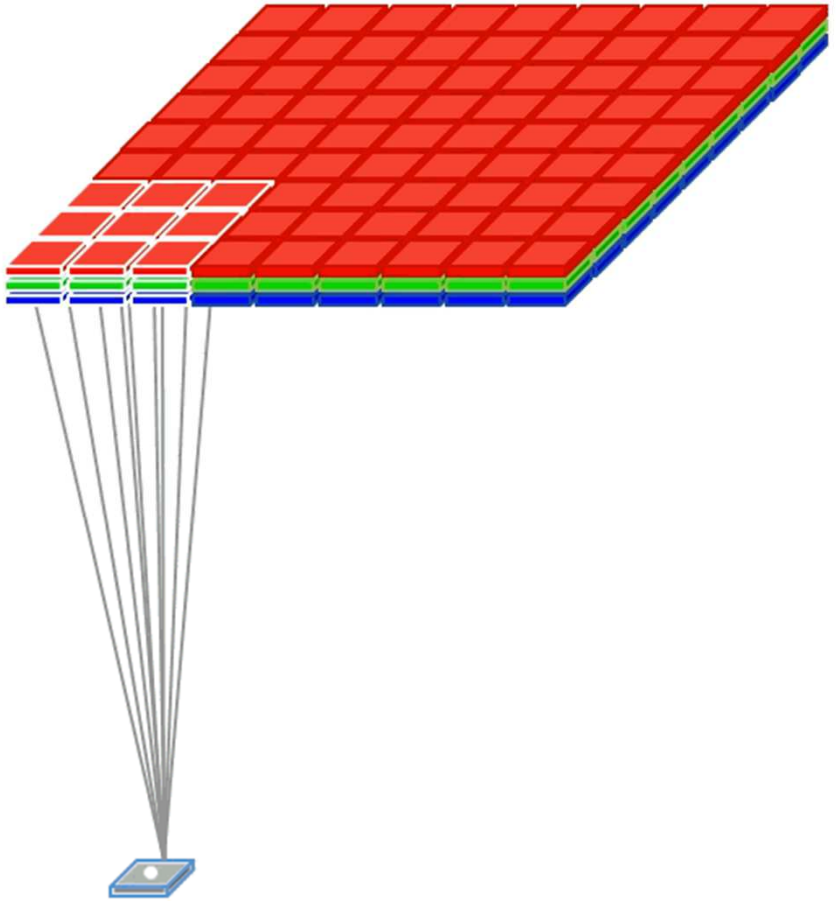
Helyett gyakorlatilag mindig korreláció történik:

$$o_{(z)}^{(l)}(x,y) = \sum_{c} \sum_{(a,b)} y'^{(l-1)}_{(c)}(x \cdot s + a \cdot d, y \cdot s + b \cdot d) \cdot w^{(l)}(a,b,c,z) + bias_{(z)}^{(l)}$$

- hibás elnevezés – képfeldolgozásban mindkét műveletet a szűrés névvel illetjük.
- Kedvez viszont a módosítás a művelet vizuális értelmezésének (csúszó ablakos skalárszorzat)

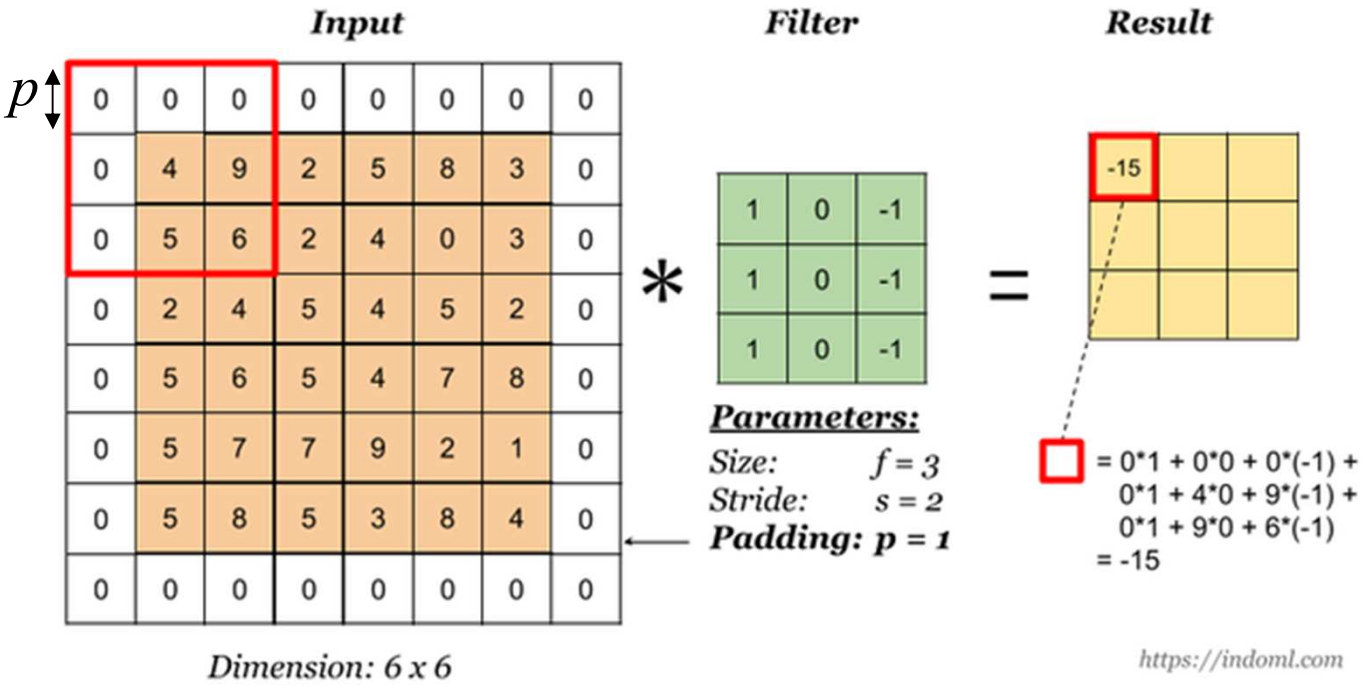
Konvolúciós réteg

- Szemléltetés:



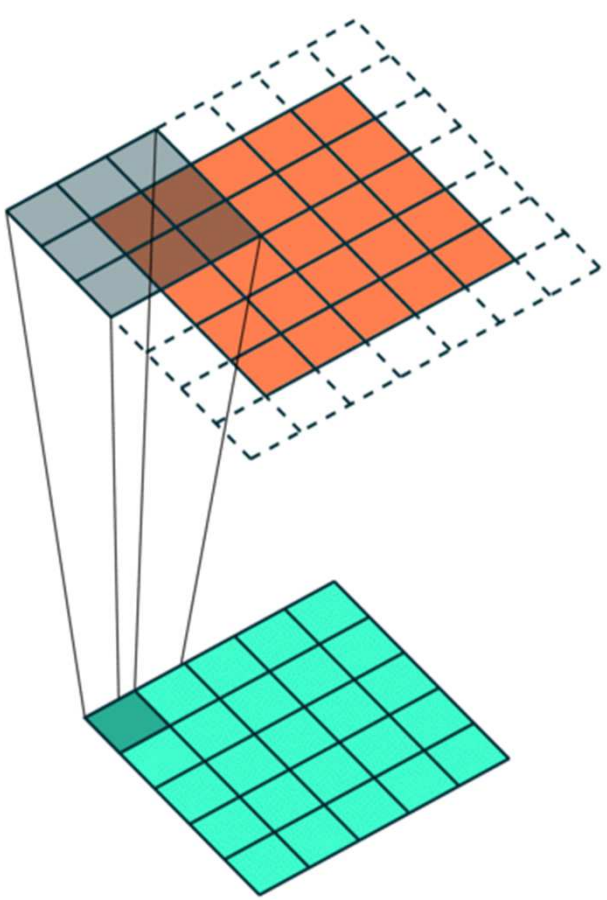
Konvolúciós réteg

- Padding (p) hatása:

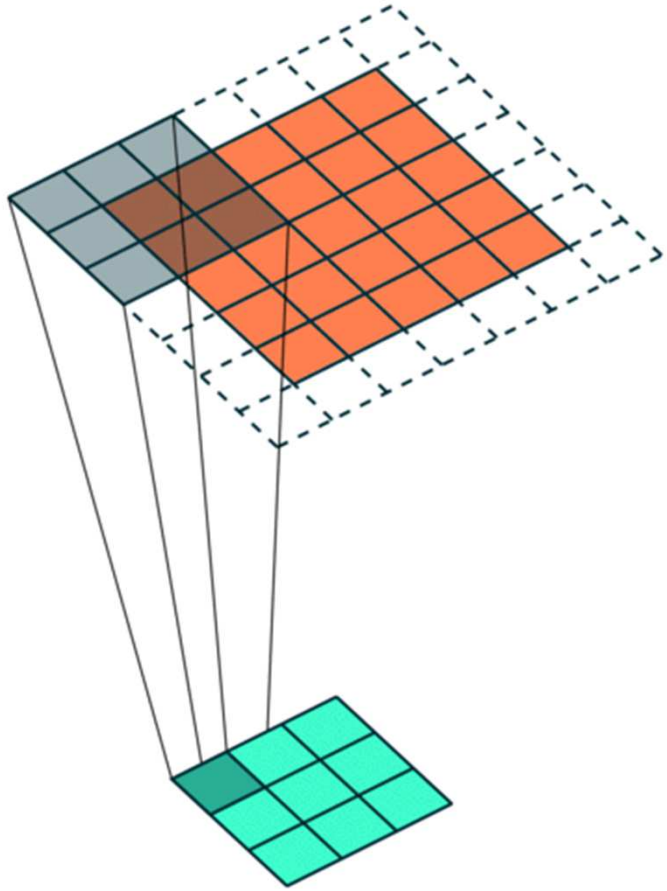


Konvolúciós réteg

- Lépésköz paraméter (s) hatása:



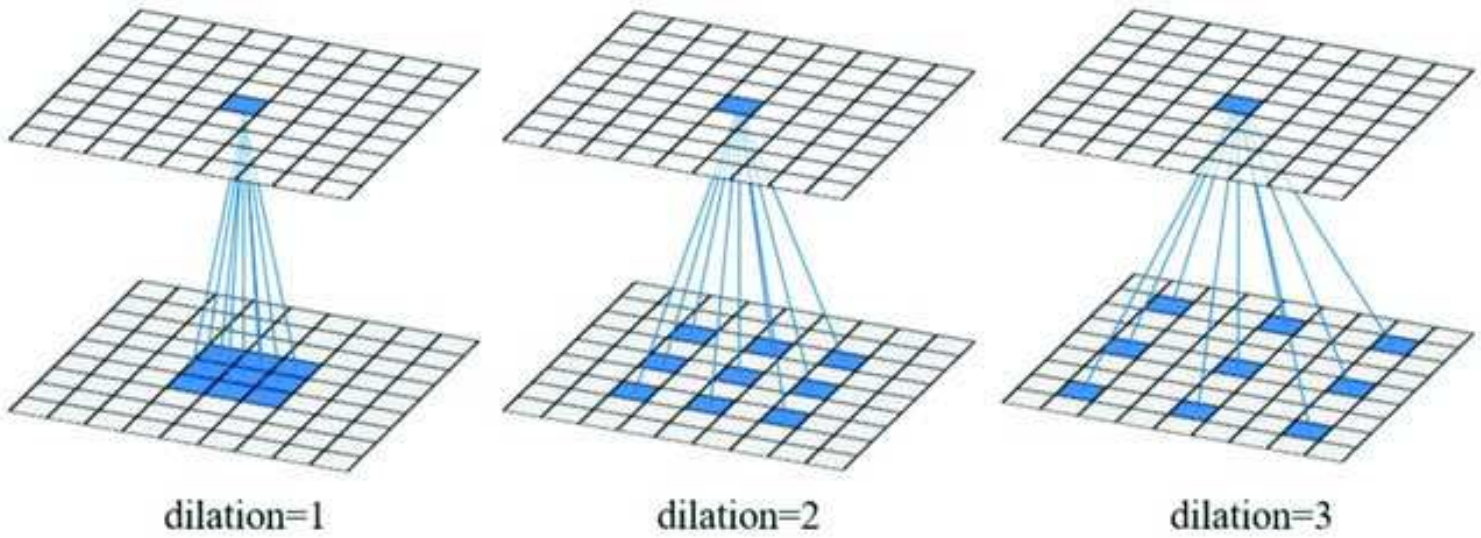
$s = 1$



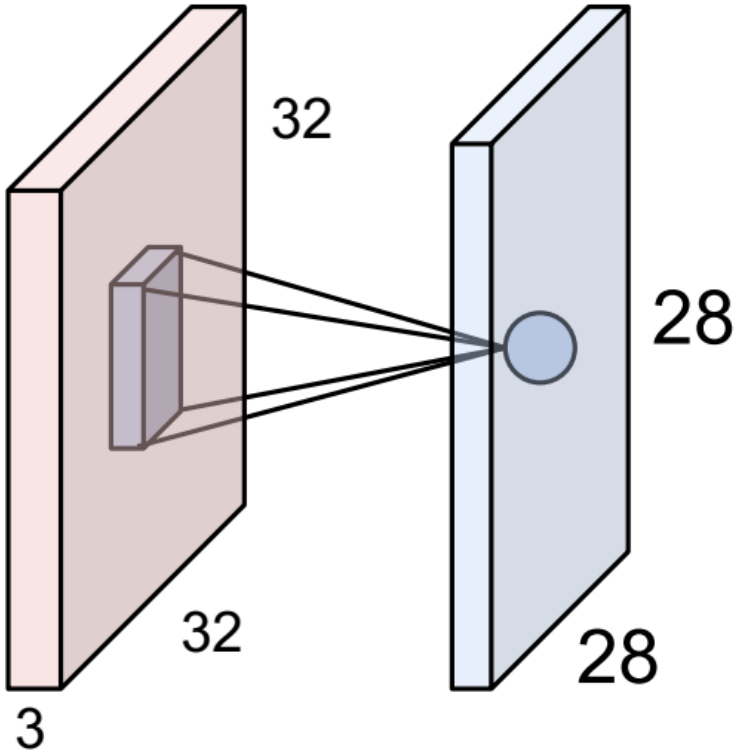
$s = 2$

Konvolúciós réteg

- Dilatáció hatása (d):

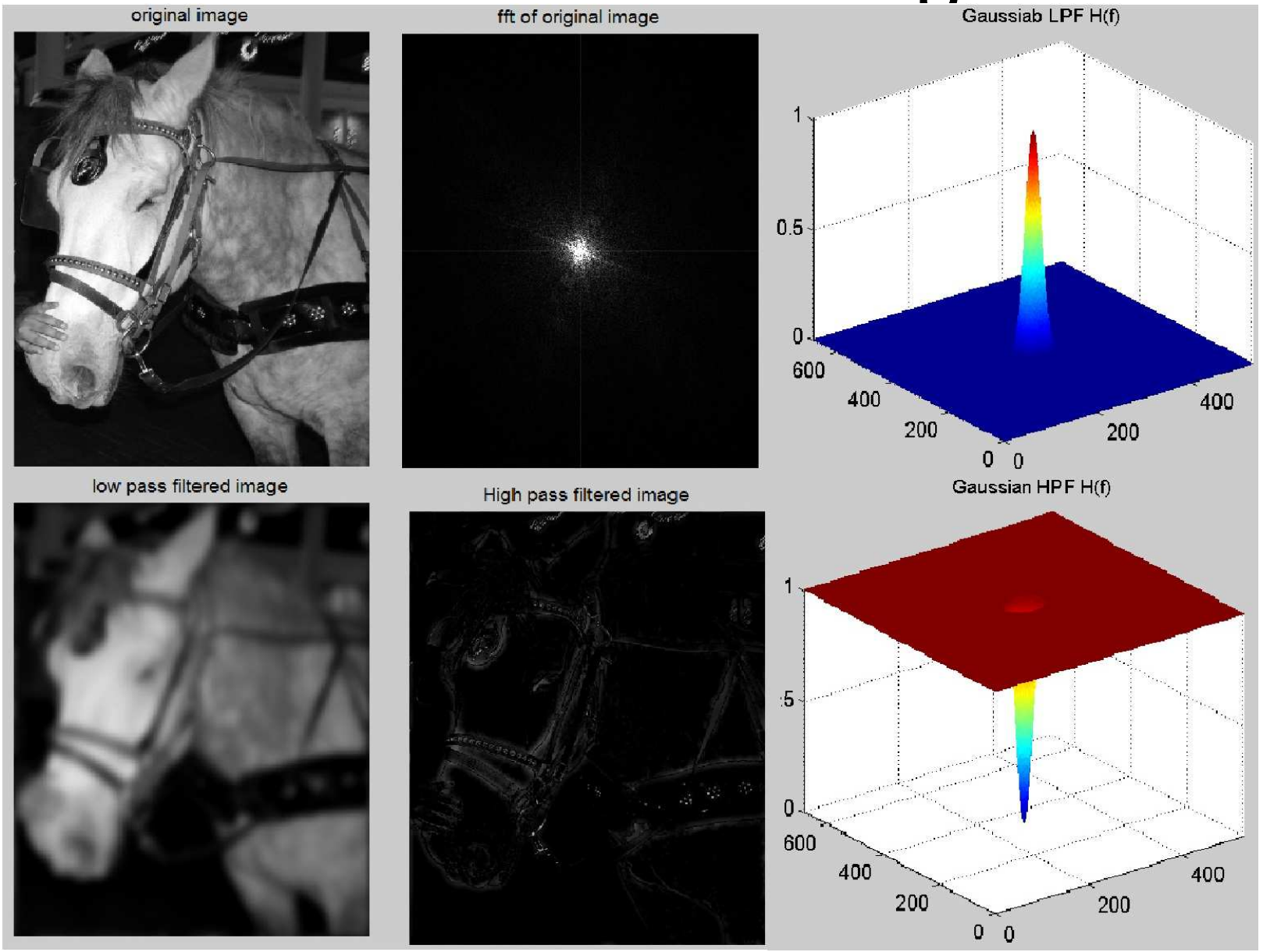


Konvolúciós réteg



- Neuron érzékenységi mezője: a bemeneti kép azon része, melytől függ a kimeneti értéke
- A példában az aktivációs térkép minden pixele az adott pixel kp.-ú 5×5-ös képrészlettől függ

Konvolúciós réteg



Szeparábilis konvolúció

- Csatornák felett szeparált (ún. depthwise):

$$o_{(z)}^{(l)}(x,y) = \sum_{c}\sum_{(a,b)} y'^{(l-1)}_{(c)}\left(x\cdot s - a\cdot d, y\cdot s - b\cdot d\right)\cdot w^{(l)}(a,b,z)\cdot \upsilon^{(l)}(c,z) + bias_{(z)}^{(l)}$$

  - Drasztikusan kisebb paraméterszám

    $size(w^{(l)}) = M_x \cdot M_y \cdot N_l$ és $size(\upsilon^{(l)}) = N_l \cdot N_{l-1}$

  - cserében korlátozottabb a leképzések halmaza

- Csatornákon belül szeparált (ún. diadikus):

$$o_{(z)}^{(l)}(x,y) = \sum_{c}\sum_{(a,b)} y'^{(l-1)}_{(c)}\left(x\cdot s - a\cdot d, y\cdot s - b\cdot d\right)\cdot w_x^{(l)}(a,c,z)\cdot w_y^{(l)}(b,c,z) + bias_{(z)}^{(l)}$$

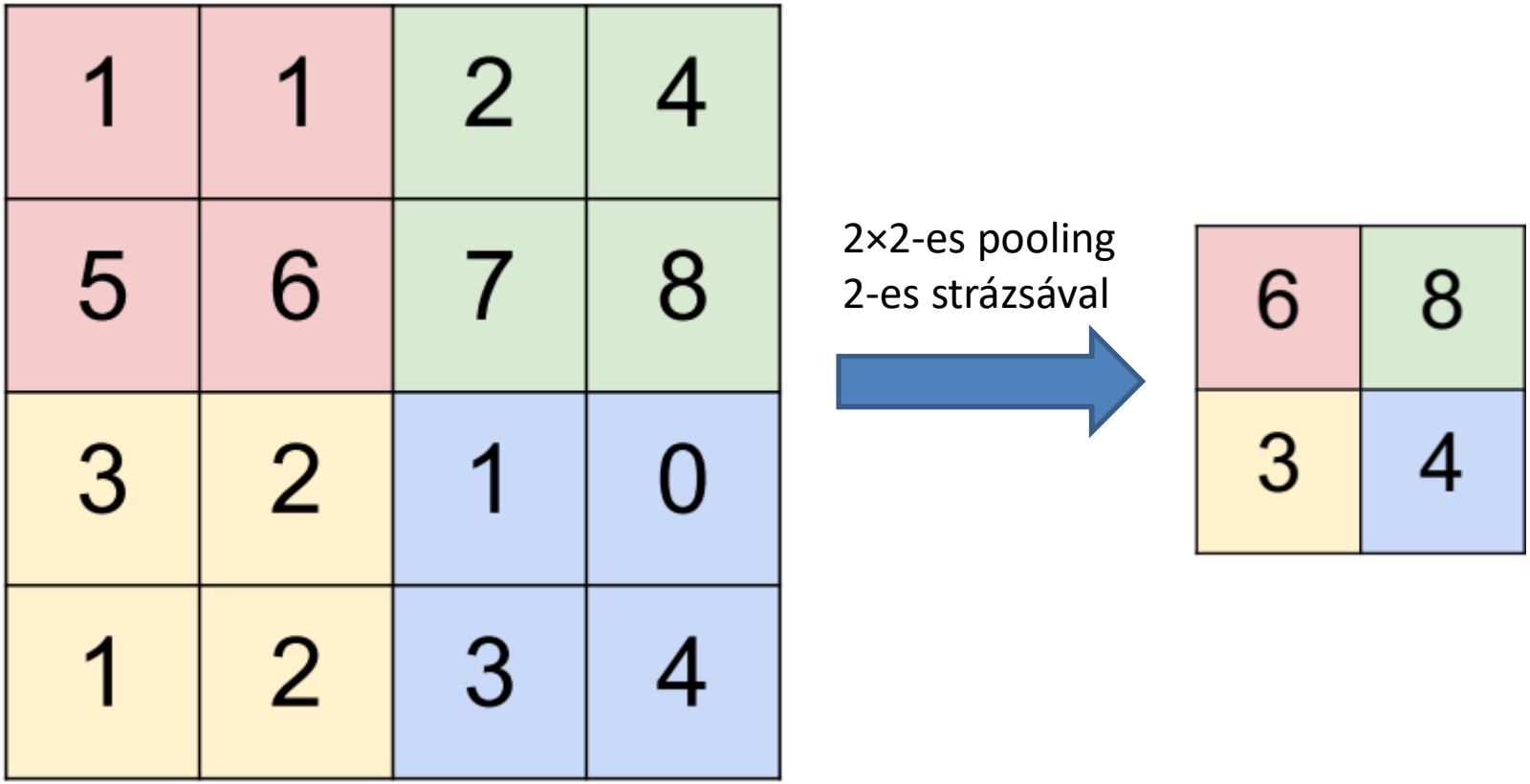
  - Kis kernelméretek miatt ritkán használjuk (sok paramétert így nem eliminálunk)

Pooling réteg

- Motiváció:
  - Csatornák felbontásának csökkentése
  - Utána következő réteg szűréseinek érzékenységi területének növelése („kisebb szűrők is eleget látnak”)
- Lényegében mintavételezi a képet:
  - *Avarage pooling*: lineáris interpolációval (textúra, stb. a régióra jellemző, gyakori minták kiemelését segíti elő).
  - *Max pooling*: olyan jellemzők kiemelését segíti, melyek csak kis számban fordulnak elő (pl. élek, sarokpontok, stb.)
- „Katasztrófa, hogy ilyen hatásosan működik”

Pooling réteg

- Max Pooling példa:

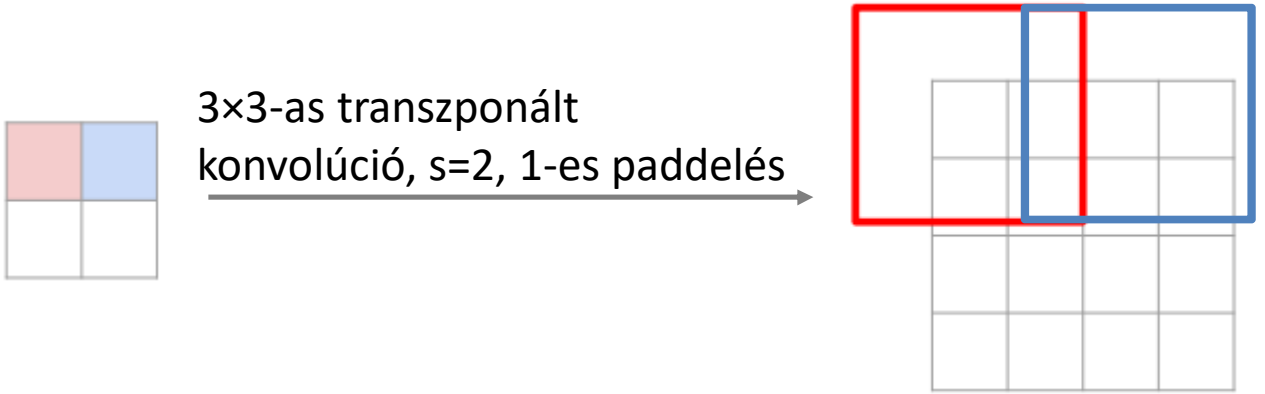


Transzponált konvolúciós réteg

- Transzponált konvolúció:
  - Konvolúció leírható Toeplitz blokkokból felépülő mátrixszal szorzásként
  - Transzp. konv.: Ezen mátrixok transzponáltjával való szorzás

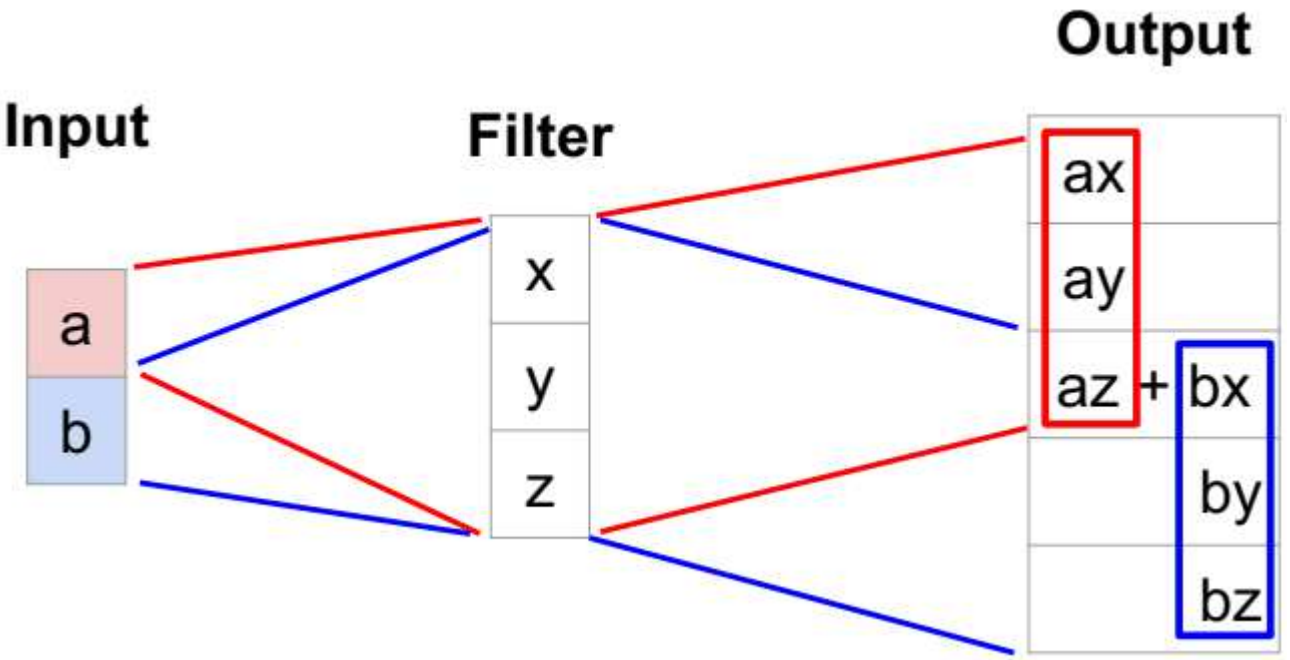
$$o_{(z)}^{(l)}(x,y) = \sum_{c} \sum_{(u,v)} y_{(c)}^{(l-1)}(u,v) \cdot w^{(l)}(\mathrm{x}-\mathrm{u}\cdot\mathrm{s}, y-u\cdot s, c, z) + bias_{(z)}^{(l)}$$

- Szemléltetése:



Transzponált konvolúció

- 1D példa:

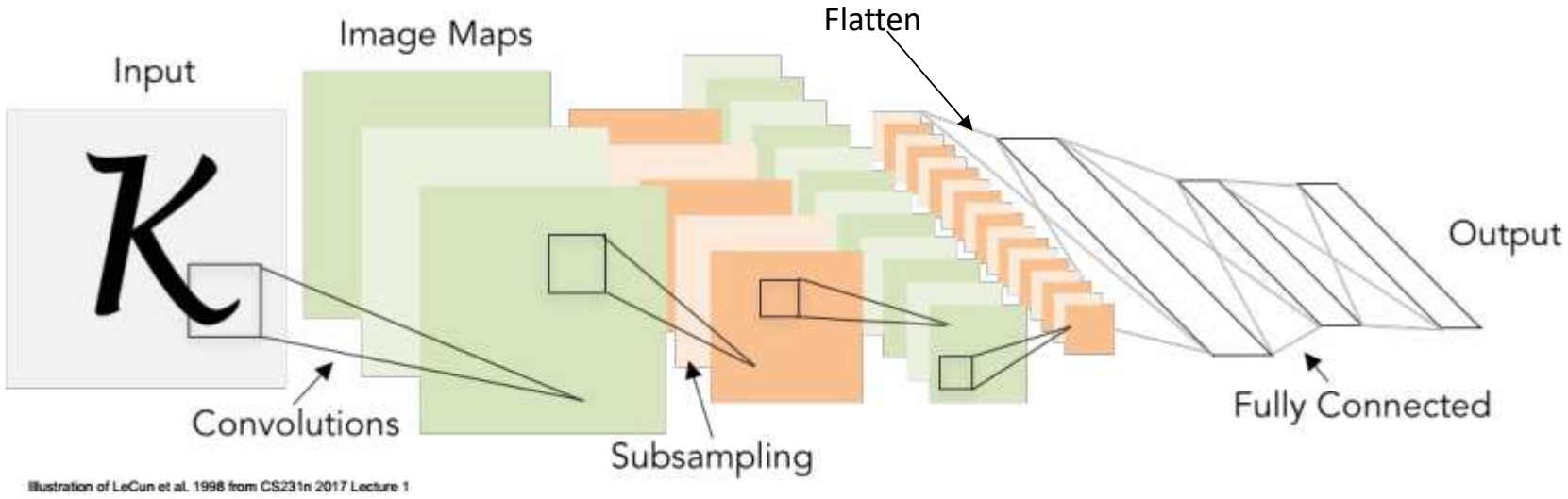


Sorosító réteg

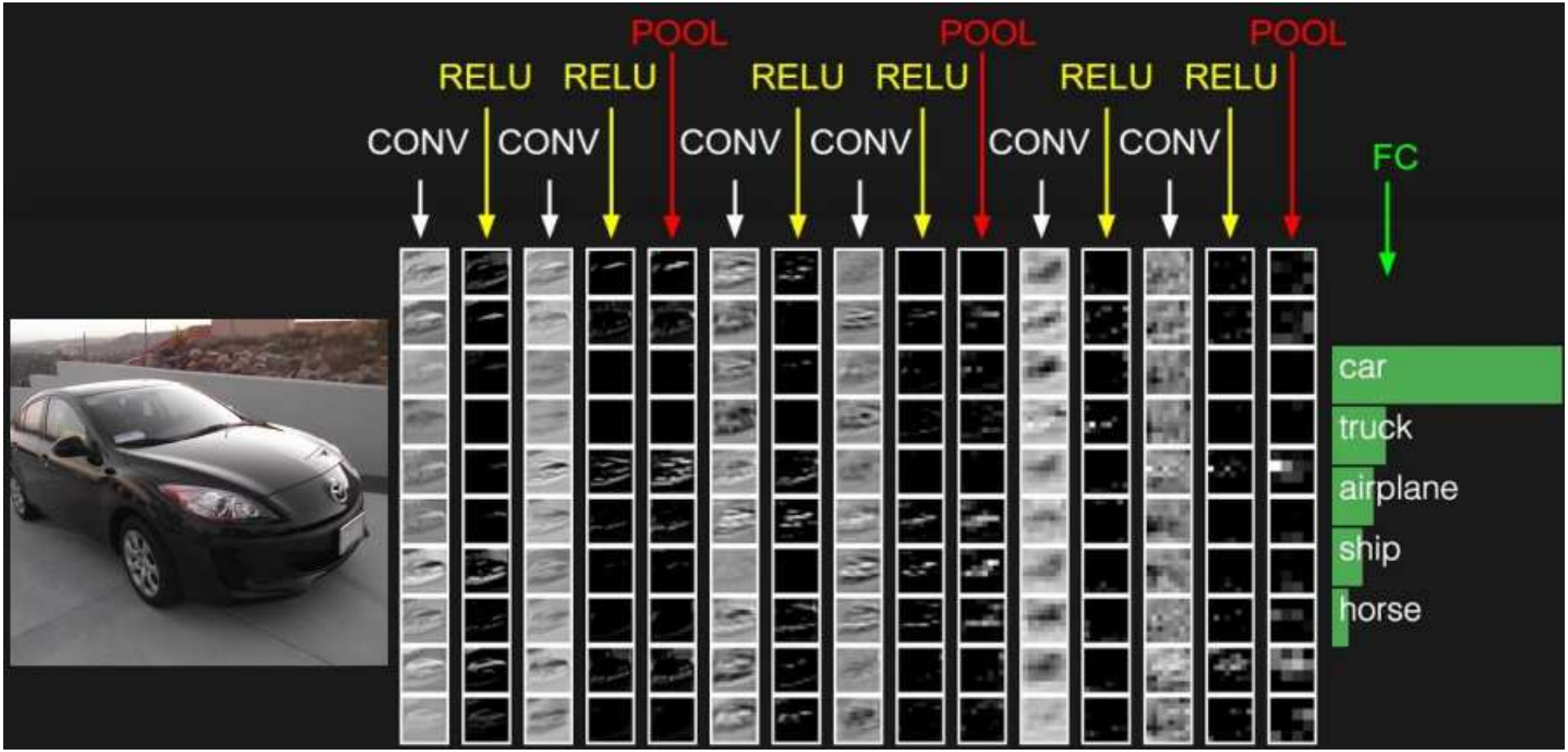
- Angol elnevezése: Flatten
- Feladata többdimenziós jelek 1D-sé alakítása:
  - Fully Connected rétegek bemenetén szokták alkalmazni
- Nem tartalmaz tanítható paramétert
- Osztályozási feladatoknál használjuk
  - Ezt szokta követni egy teljesen összekötött NN-es rész

KONVOLÚCIÓS NEURÁLIS HÁLÓZATOK A GYAKORLATBAN

Tipikus felépítés



Működés szemléltetése



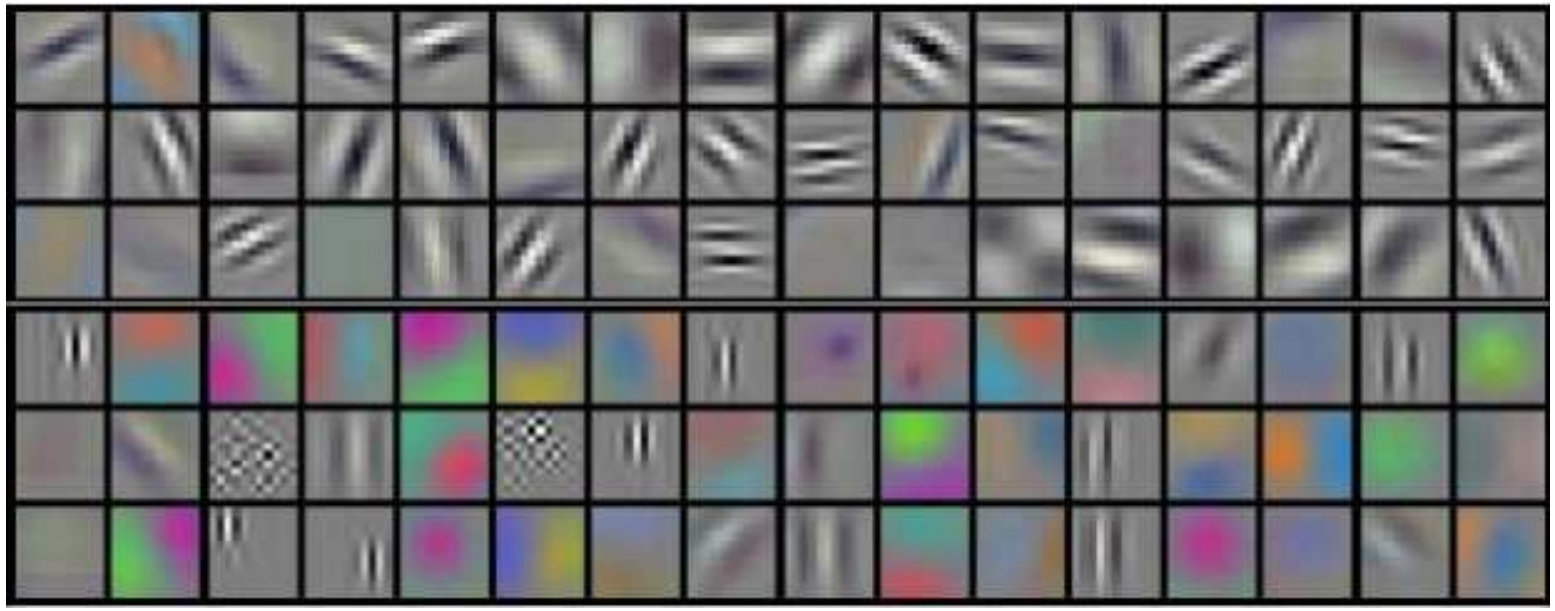
Működés értelmezése

- Sok réteg miatt nagyjából lehetetlen:
  - Pedig a konvolúciók kompozíciója is egy konvolúció
  - Nem linearitások, valamint a Maxpool-ok gyakorlatilag követhetetlenné teszik a hálók működését
- Interpretáció lehetőségei:
  - Bemenet utáni első konvolúciós szűrőinek vizualizációja
  - Olyan absztrakt részdomének keresése, melyen belül a háló válasza konstans
  - Aktiváció térképek vizsgálata

    (esetleg olyan bemenetek generálása, melyek egy ilyen térkép valamelyik normáját maximalizálják – Deep Dreaming)

Működés értelmezése

- Bemenethez közeli rétegek alacsony absztrakciós objektumokat keresnek (pl. AlexNet szűrői):



- Kimenethez közelebbi rétegek már összetett objektumokra érzékenyek

Tanítási módszerek

- Konvolúció ellenére is túl sok a paraméter:
  - Konkrét feladat esetén sosincs elég minta
  - Két új módszer: tudás transzfer, és minta generálás
- **Minták generálása – Data augmentation**:
  - Meglévő képekből *újakat generálunk* olyan *véletlenszerű torzítások alkalmazásával*, *melyekre invariáns viselkedést várunk el* a hálótól:
    - Pl. Eltolás, forgatás, tükrözés, nyírás, perspektív transzf.
    - Pl. megvilágítás változásának szimulálása
    - Pl. képzaj hozzáadása
  - Mivel a hálót invariáns viselkedésre tanítja, ezért *javul az általánosító képesség* is!

Minta generálás példa

- Bemenet: 













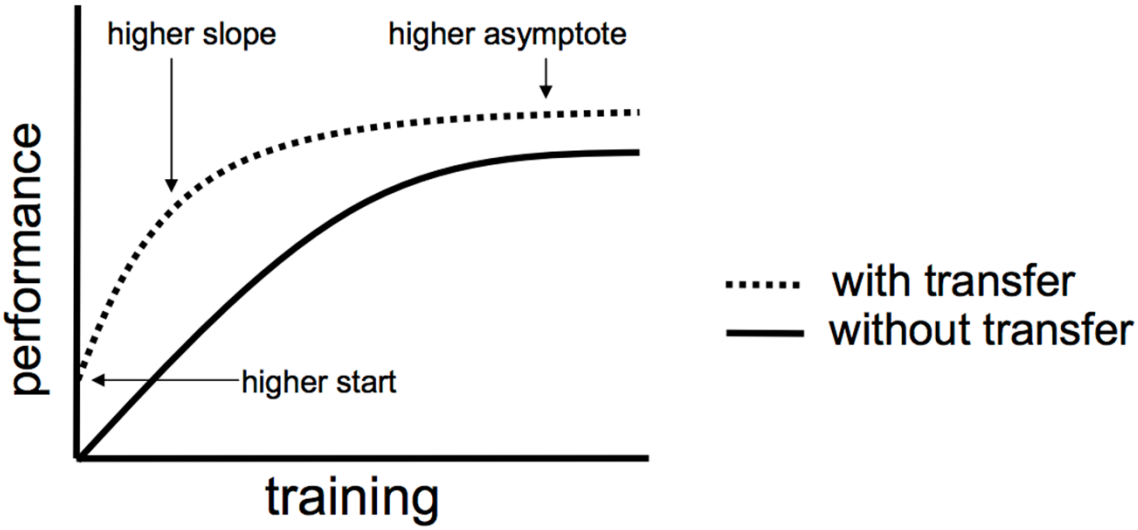


Tudás transzfer

- Konvolúciós hálók *bemenethez közel* olyan *jellemzőket* emelnek ki, melyek *nem feladat specifikus*ak:
  - Tipikusan élek, sarokpontok, általános textúra (pl. Gábor wavelet szűrések), magas frekvenciás kiemelés, stb.
- Csak a szükséges rétegeket tanítsuk újra:
  - *Transfer learning*: egy más (jóval nagyobb mintaszámú) példán betanított háló első N. rétegét befagyasztjuk, majd arra ültetett hálót tanítjuk csak
  - *Fine tuning*: iteratívan az alsóbb rétegek befagyasztásával növelve a kimenethez közeli, tanított rétegek számát tanítunk

Transfer Learning

- Előre tanított hálót jellemzők kiemelésére használjuk, majd annak a kimenetét kezeljük bemenetként:
  - Nagy hangsúly jut ezeknek, pár dia múlva nézünk párat
- Jelentősen jobb tanulás:
  - Főleg ha elegendően nagy a hálónk, és komplex a probléma



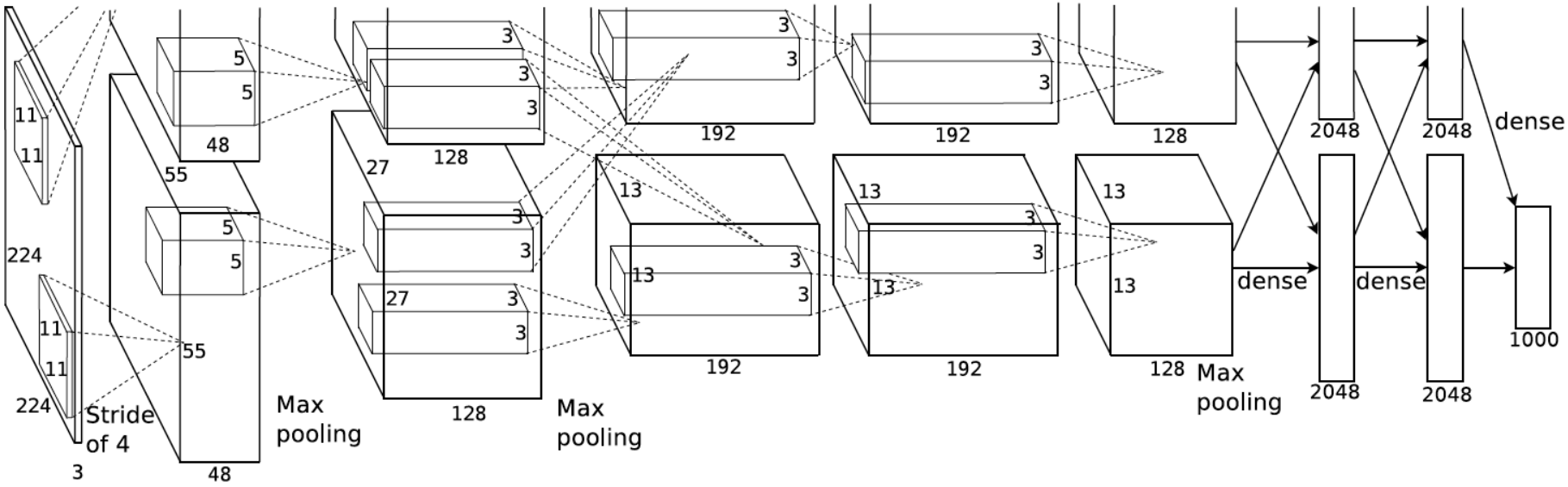
Transfer learning

- Általános ajánlás:

	Hasonló mintakészlet	Erősen eltérő minták
Kevés minta	*Transfer Learning*: kimenethez közeli aktivációk osztályozása	*Transfer Learning*: Bemenet közeli aktivációk osztályozása
Sok minta	*Fine tuning*: Elég csak a kimenet közeli rétegeket finomhangolni	*Fine tuning:* Finomhangolás a bemenet közelében is szükséges

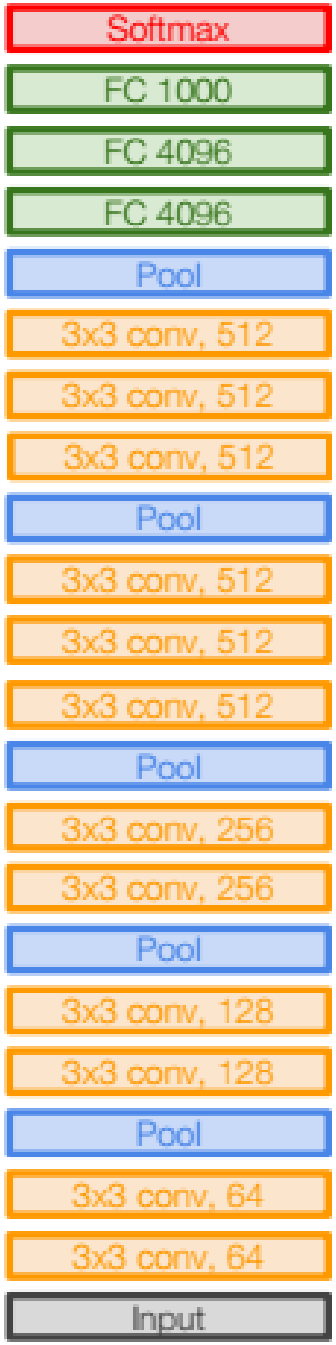
AlexNet (2012)

- Kevés réteg, ezért nagy kernelek (pl. 11 × 11)
- ~60 millió súly
- Sok augmentációs módszert alkalmazott
- Két GPU-s végrehajtásra bazírozott arch.



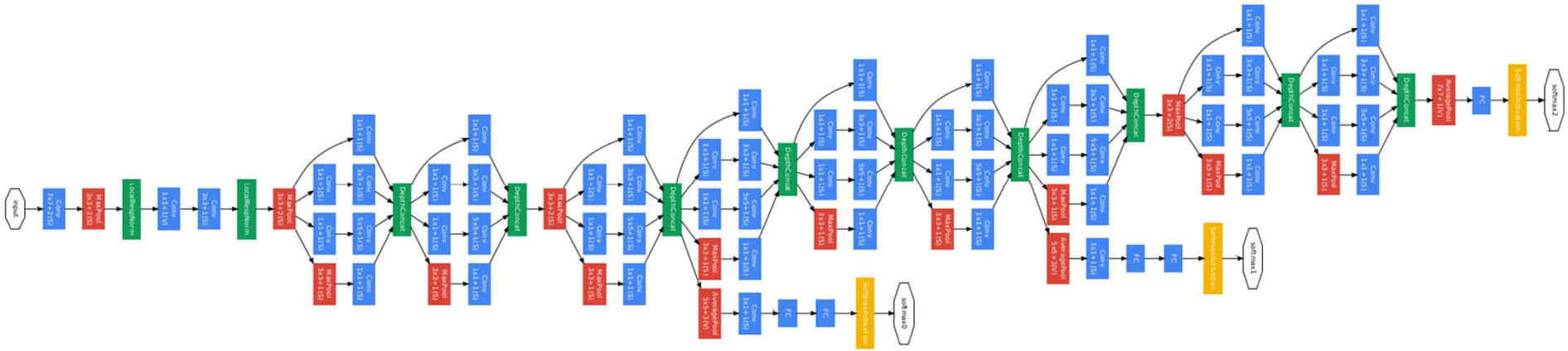
VGG (2014)

- 16-19 réteg, 3×3-as kernelek
- 2×2-es MaxPooling
- ~140 millió súly
- 256 × 256-os képekre 100 MB RAM
- Transfer learning: ált. utolsó előtti FC kimenetén



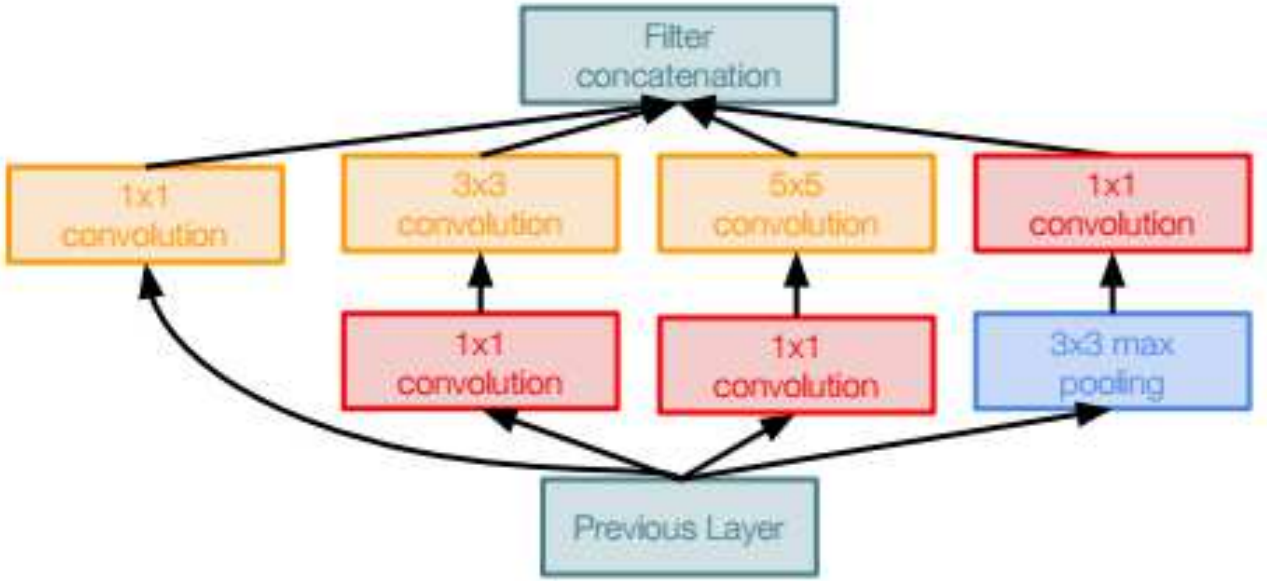
GoogleNet – Inception (2014)

- 22 réteg, de ~5 millió súly
- Kimenet több mélységből számítva, ezeken u.ú. képződik a hiba (cél a hiba visszaterj. rövidítése)
  - Éles használatban csak az utolsó kimenetet szokták figyelni (esetleg átlagolják a kimeneteket – Ensemble)
- Új strukturális elem – inception modul



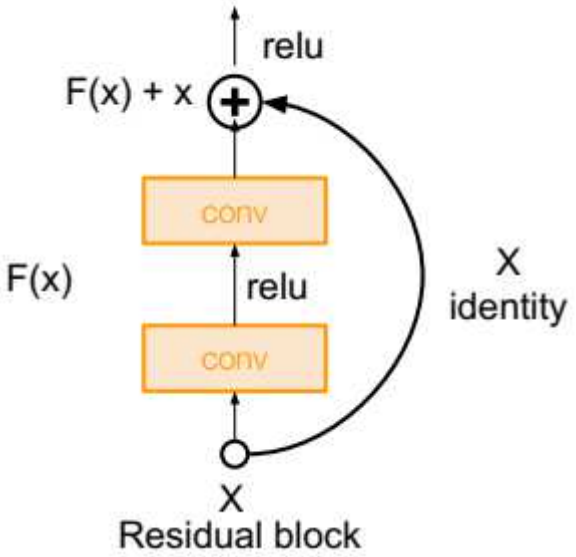
Googlenet – Inception modul

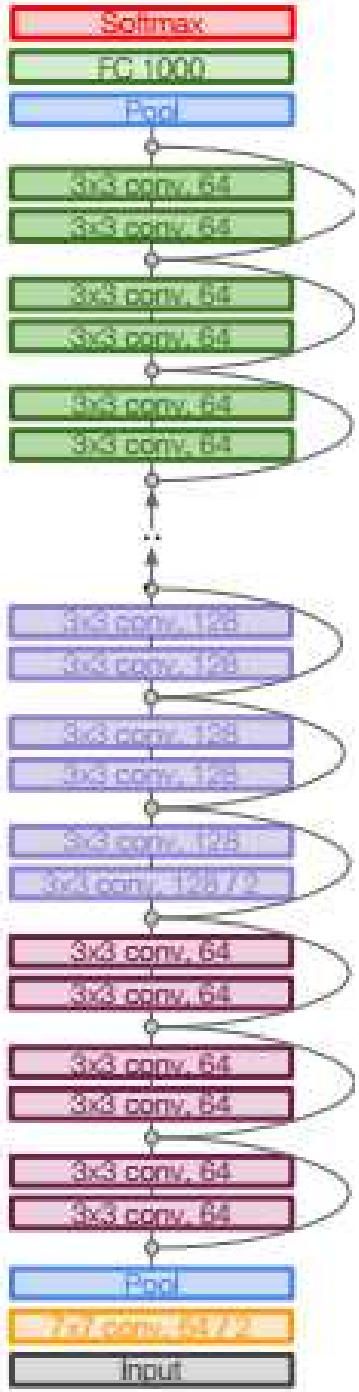
- Motiváció : előre nem tudjuk, hogy mekkora kernel lesz jó, ezért legyen egy szinten több, különböző méretű.
- 1 × 1-es konvolúciók célja a csatornák számának (így a RAM, PU igény) csökkentése (kivéve a narancssárga elemet)
- Konkatenáció, mint új elem



Resnet (2015)

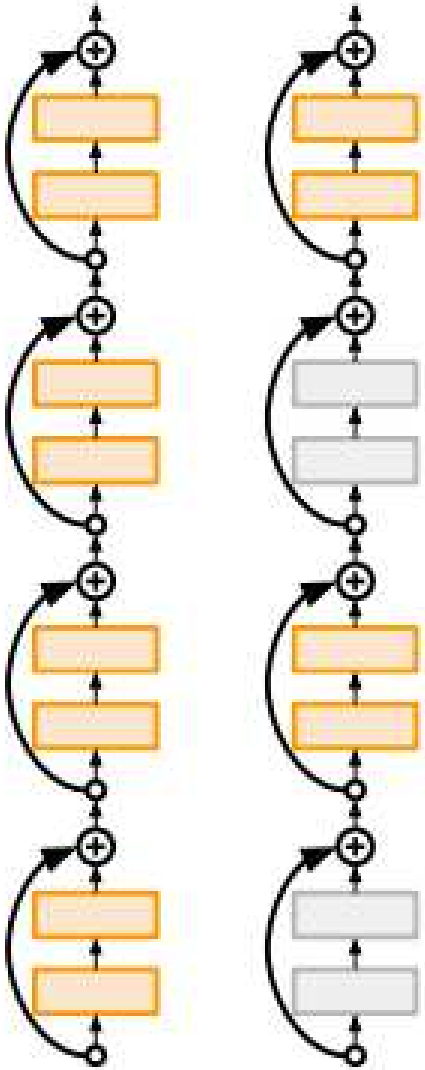
- 152 réteg, mindegyik konvolúció 3×3-as
- Skipp connection, mint új elem
  - Cél itt is az optimalizációs problémák megkerülése
  - Identikus leképzés + különbség dekompozíció
  - *Nem kell az identikus leképzést (ID) külön megtanulni !*
  - *Amúgy is minél több réteg van, az ID annál közelebb van az optimumhoz*





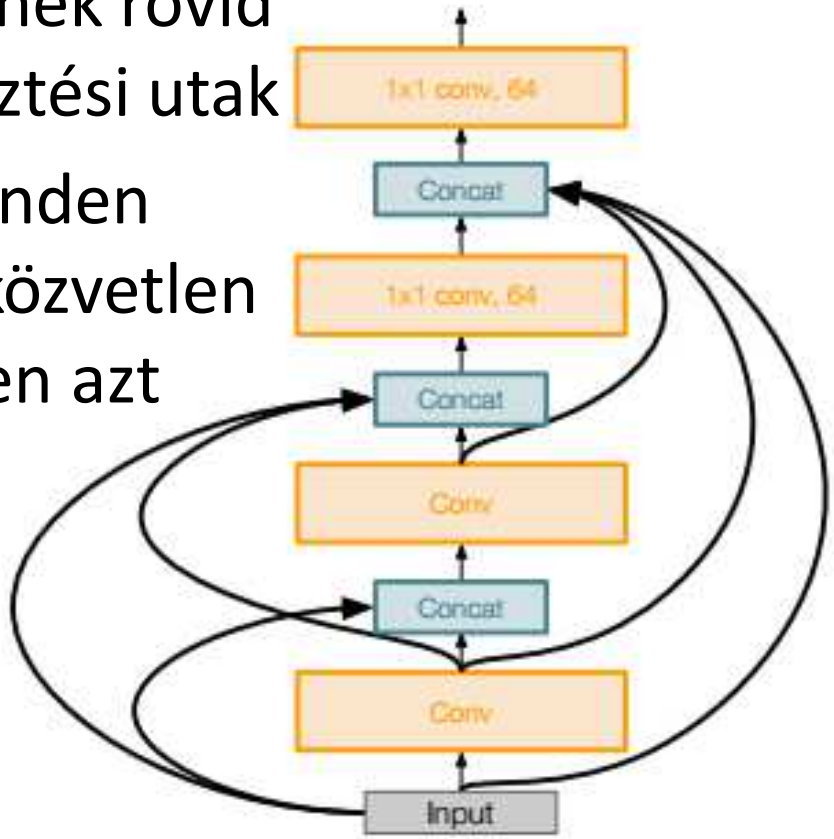
Stochastic Depth

- Motiváció: egyszerűbb a hiba visszaterjesztése, ha sekélyek a hálók
- *Dropout adaptálása*: tanítás során teljes blokkokat hagyigálunk ki:
  - Kidobott blokk helyett ID leképzés
  - Teszt időben teljes hálót nézzük / MC Dropout mintájára MC SD (bizonytalanság minősítés)



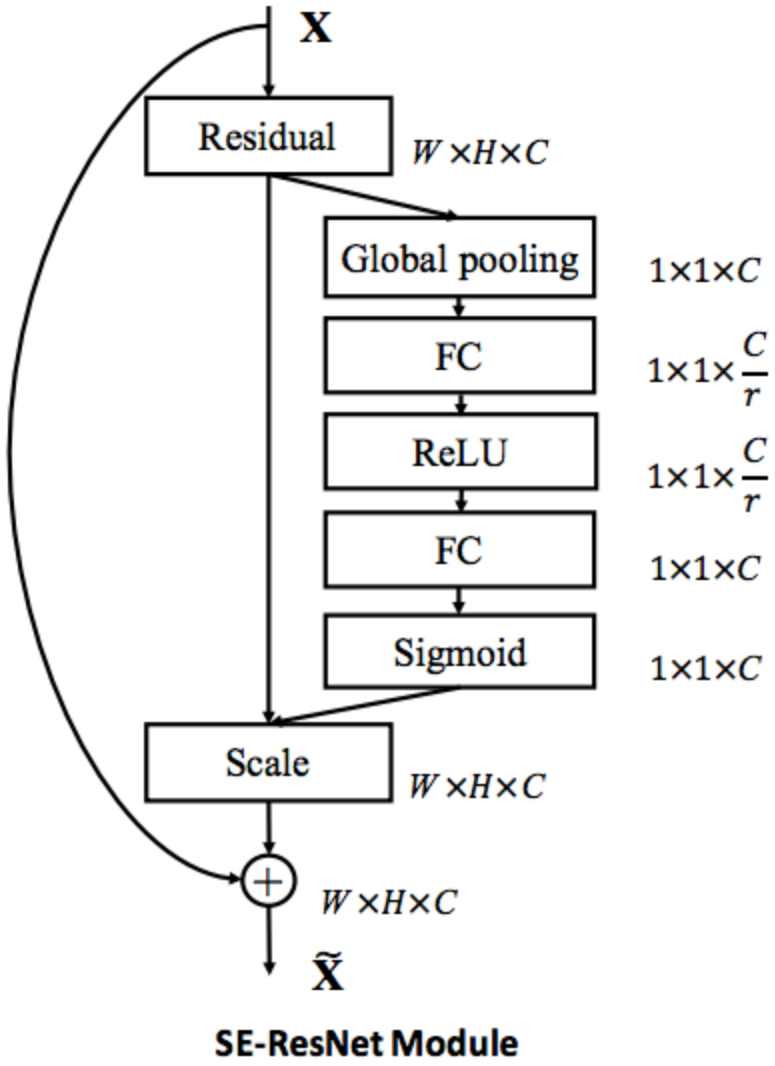
DenseNet (2017)

- Motiváció: legyenek rövid hiba-visszaterjesztési utak
- Blokkon belül minden réteg kimenete közvetlen bemenete minden azt követő rétegnek
- „Feature reuse”



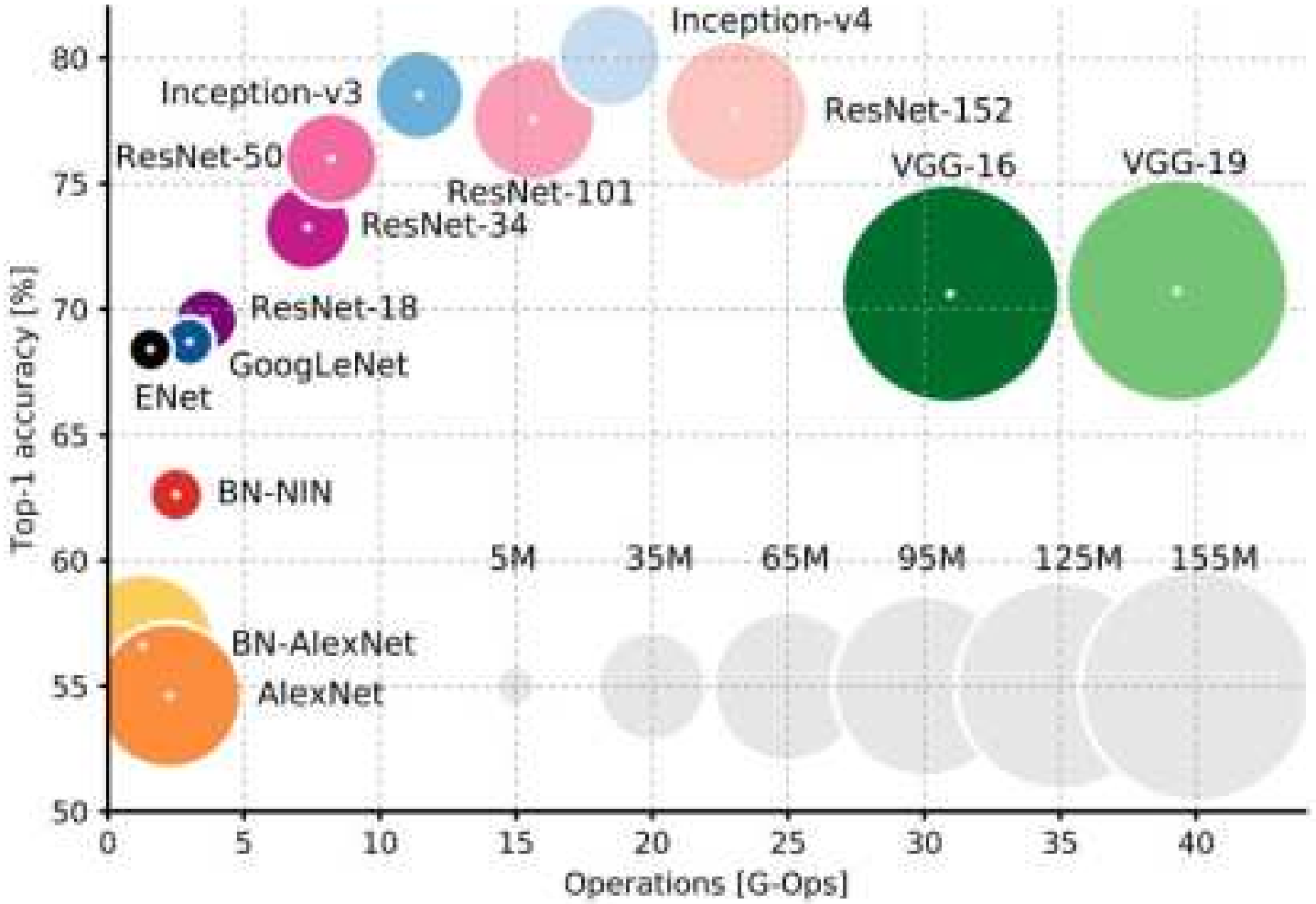
Squeeze and Excitation Networks (SENet) (2017)

- Aktivációktól függően újrasúlyozza a konvolúciós csatornákat.
- Az újrasúlyozást egy két rétegű MLP végzi
- Látványos javulás: SEResNet-50 ≈ ResNet 152



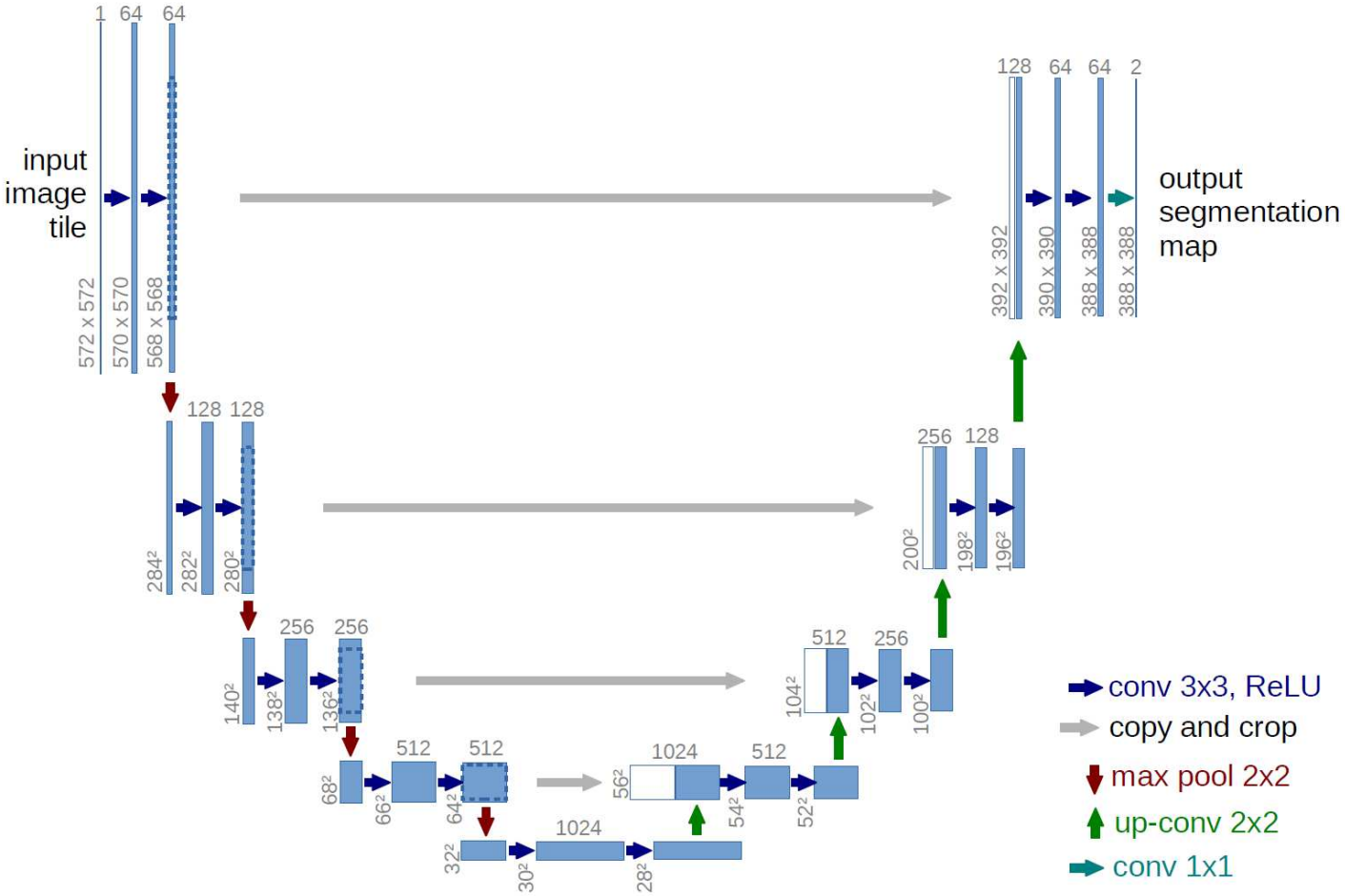
SE-ResNet Module

Hálók komplexitása és pontossága



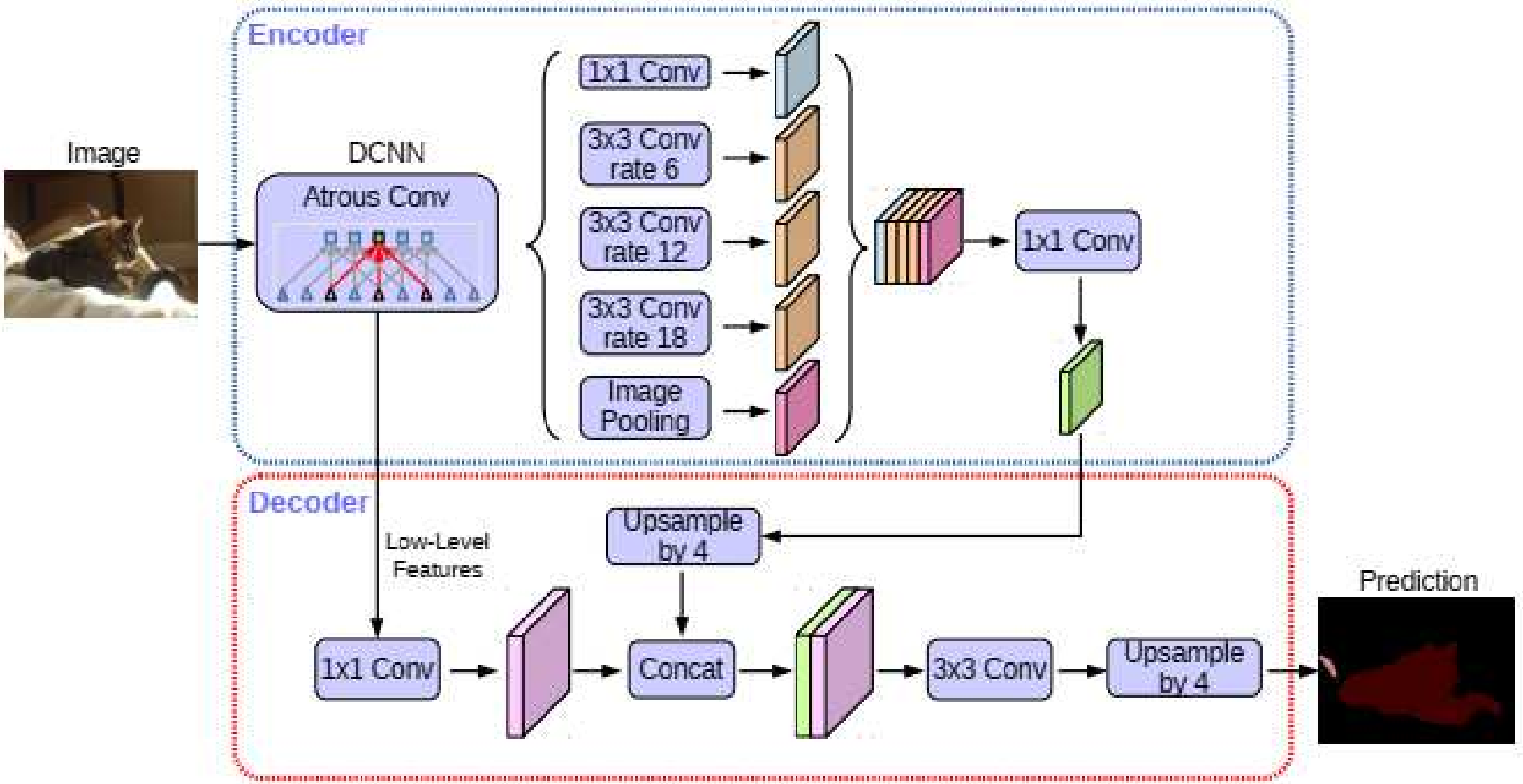
U-Net (2016)

- Teljesen konvolúciós (Fully Convolutional) háló



DeepLab –v3+ (2018)

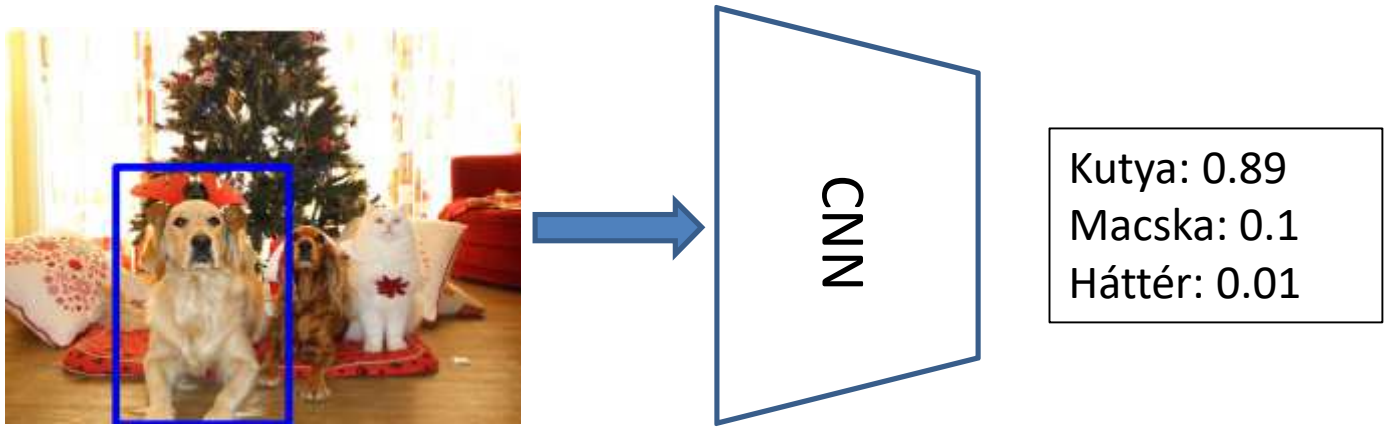
- Enkóder – Dekóder architektúrák továbbfejlesztése



OBJEKTUMOK KIEMELÉSE CNN-EL

Csúszó ablakos detektálás

- Obj. Detektálás direkt redukálása osztályozásra:
  - Különböző méretű (skálájú, alakú) téglalapokat tol végig a vizsgálandó képen, és az így kivágott részeket osztályozza
  - Osztályok: felismerendő obj. típusok + háttér
  - Előnye: kis munka, hátránya: nagyon lassú, vagy pontatlan

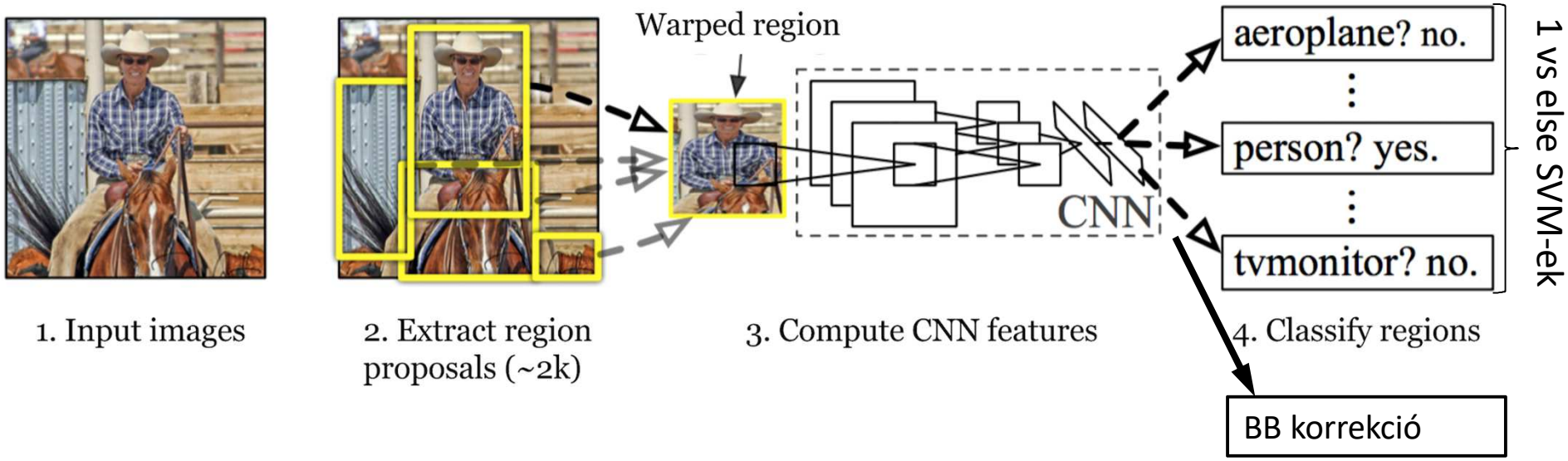


Régió alapú CNN-ek

- Klasszikus objektum detektálási séma:
  1. ROI-k kiemelése (olyan képrész, mely „fontos”)
  2. Kiemelt ROI-k taksálása
- Megvalósításai (meta architektúrák):
  - ROI detektálását kívülről váró eljárások:
    - R-CNN, Fast R-CNN
  - ROI detektálását is elvégző eljárások:
    - Faster R-CNN
    - Region based fully convolutional NN (R-FCN)
  - ROI-kat nem kereső (és bemenetén sem kérő) eljárások:
    - You Only Look Once (Yolo), Single Shot Detection (SSD), RetinaNet

R-CNN

- Bemenete: kivágott és átméretezett képrészlet
  - Szakértői rendszerrel végzendő, épp ezért sokszor nehéz feladat (pl. Selective Search – glob. szegm.)
- Kimenete: képrészlet osztályozása + pozíciójának korrekciója

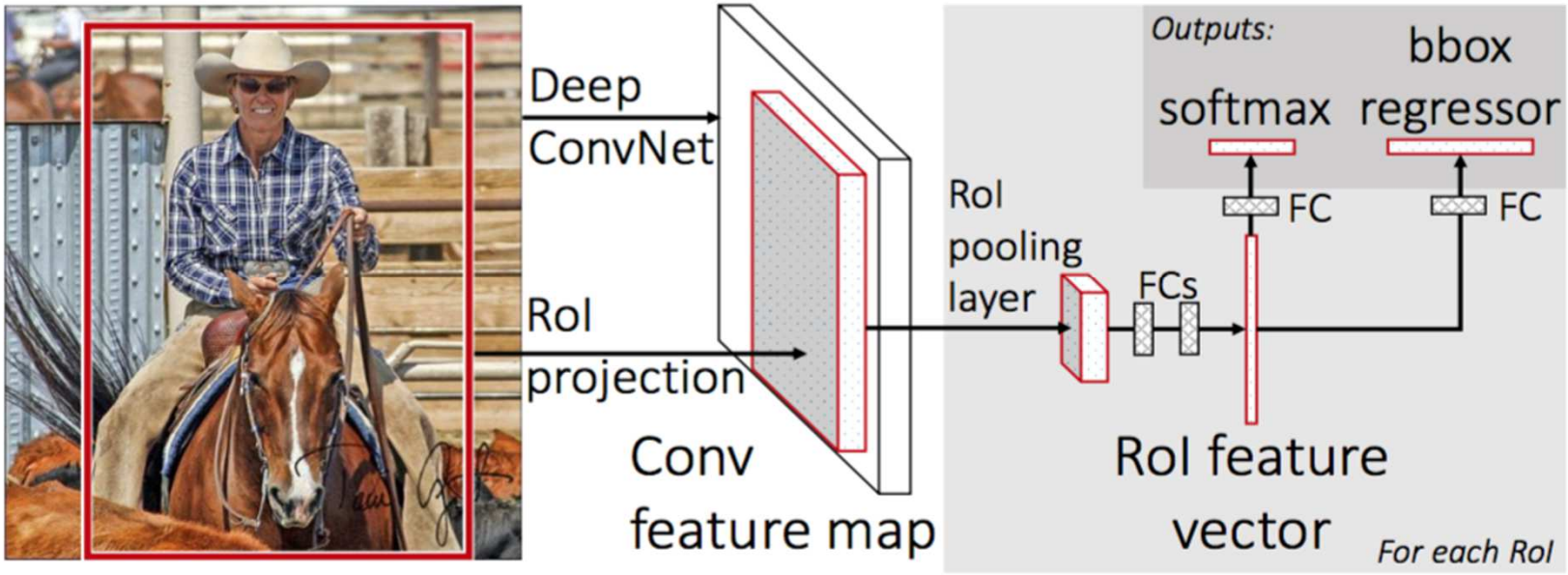


R-CNN

- Gyakorlati alkalmazást tekintve túl lassú:
  - Elegendően nagy pontossághoz sok ROI kell
  - Mindegyik ROI-t végig kell pofozni a teljes hálón
    - Miközben ezek gyakran átlapolódnak, ezért u.a. a képrészletetek alapján többször is számolni kényszerülünk
- Ötlet:
  - *A teljes képet együtt vizsgáljuk egy mély FC-CNN-el, és az abból kinyert jellemzőket vizsgáljuk egy sekélyebb NN-el*
  - Ez a Fast R-CNN alapelve

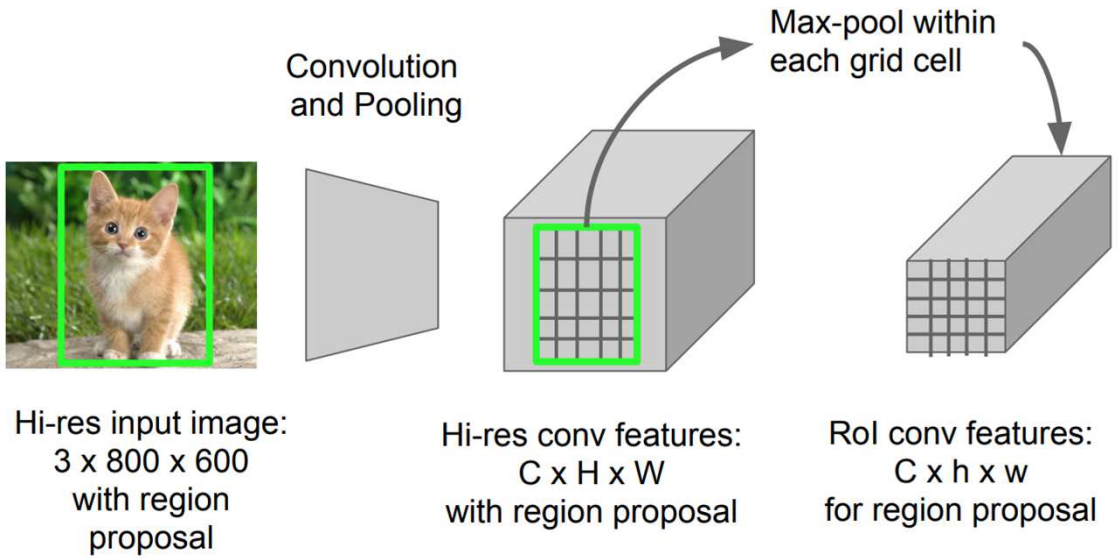
Fast R-CNN

- Új réteg: RoI Pooling:
  - Bemenetei aktivációs térképek, valamint a ROI koordinátái
  - Kimenet: az aktivációs térképek ROI-hoz tartozó részének 7×7 pixelesre átméretezett változata (Max-pool)



Fast R-CNN

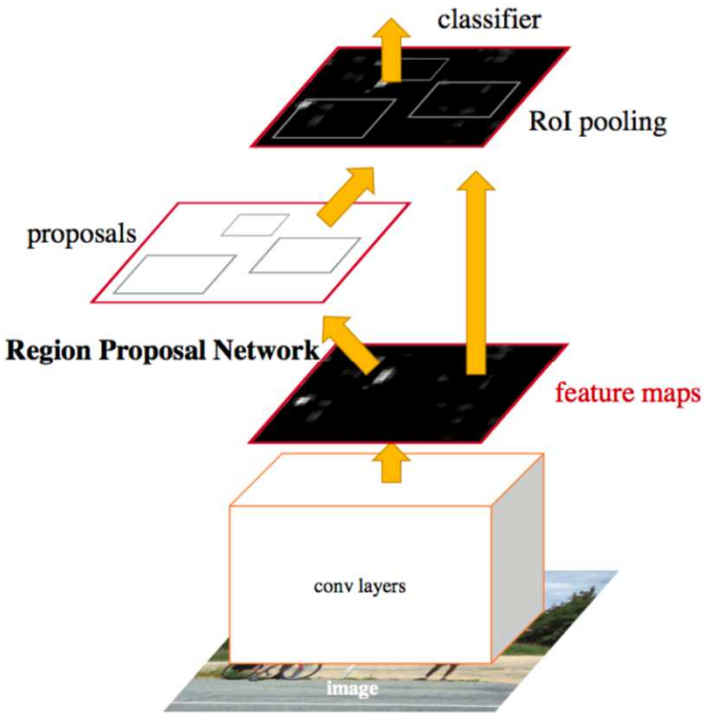
- ROI pooling háló:

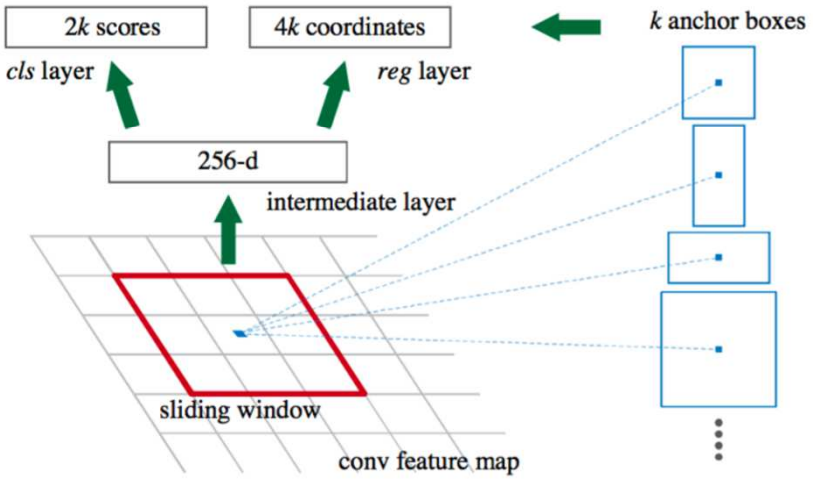


- Gyorsabb a számított jellemzők újra használása miatt
  - Vizsgált ROI-k számának korlátozásával munkapont jelölhető ki
- De még mindig kell bele szakértői ROI kiemelő
  - Ezt fogja átvállalni a Faster R-CNN

Faster R-CNN

- Új elem: Region Proposal network
  - Előre definiált régiókat minősít a szerint, hogy ROI-e
  - U.a. részháló válasz alapján dönt, mint a kimeneti osztályozó



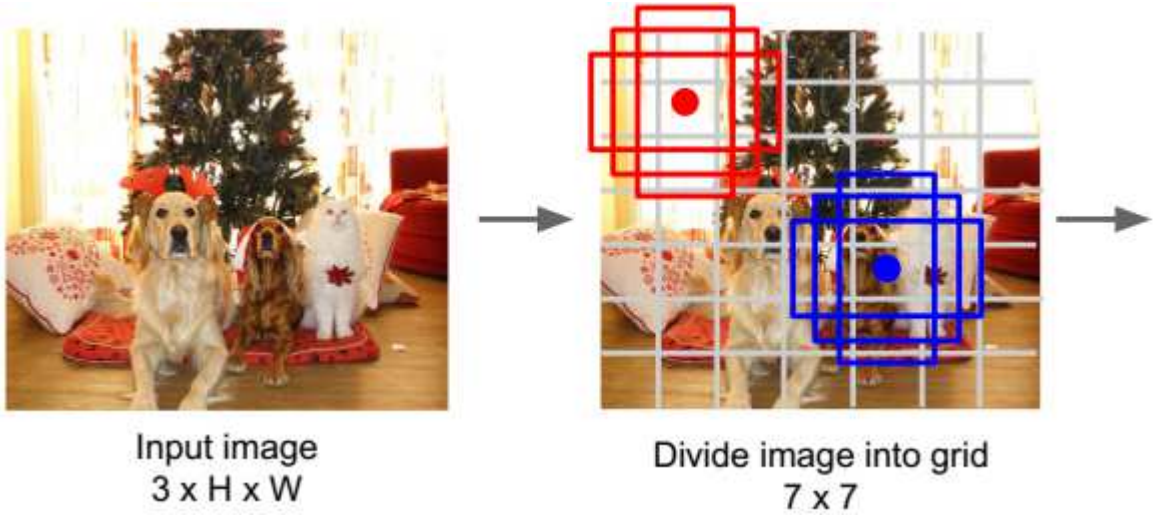


Faster R-CNN

- Tanítása:
  1. ROI-kat előállító hálón képezünk csak hibát
  2. ROI kiemelő kimenete alapján tanítjuk a ROI pooling utáni rétegeket
  3. Fine tuning a ROI kiemelő rétegekre (és az alatta lévő konvolúciós részhálóra)
  4. Fine tuning csak a Fast R-CNN rétegekre
  5. 3.-4. lépés ismétlése
- Legpontosabb meta architektúra volt sokáig:
  - Az eddigiek közül a leggyorsabb is

YOLOv2

- Eltűnik ROI pooling:
  - Képet nem átlapolódó régiókra osztja (7×7), melyekbe előre meghatározott boxokat illeszt
  - Nincs benne FC réteg (jóval kevesebb paraméter)

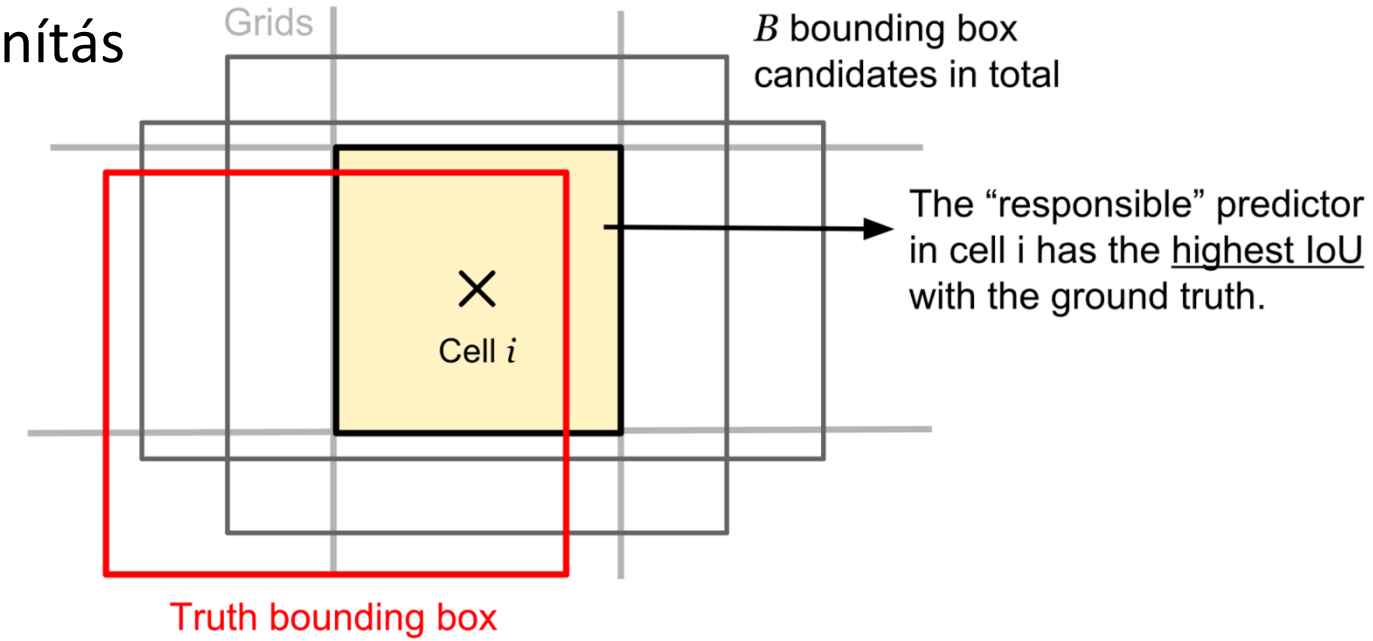


- Cellánként B box pozíciója: (dx, dy, dh, dw, obj. konfidencia)
- Cellánként egy osztályba sorolás (C-s softmax)

Kimenet: 7×7×(5B+C)

YOLOv2

- Tanítás:
  1. Boxok konfidenciájára hiba képzés, az alapján tanítás
  2. Objektumokat tartalmazó Boxok regressziós kimenetei (dx,dy,dh,dw) alapján történő tanítás
  3. Objektumokat tartalmazó cellákra a C kategória szerinti tanítás



YOLOv2

$$
\begin{aligned}
&\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2 \qquad (3)
\end{aligned}
$$

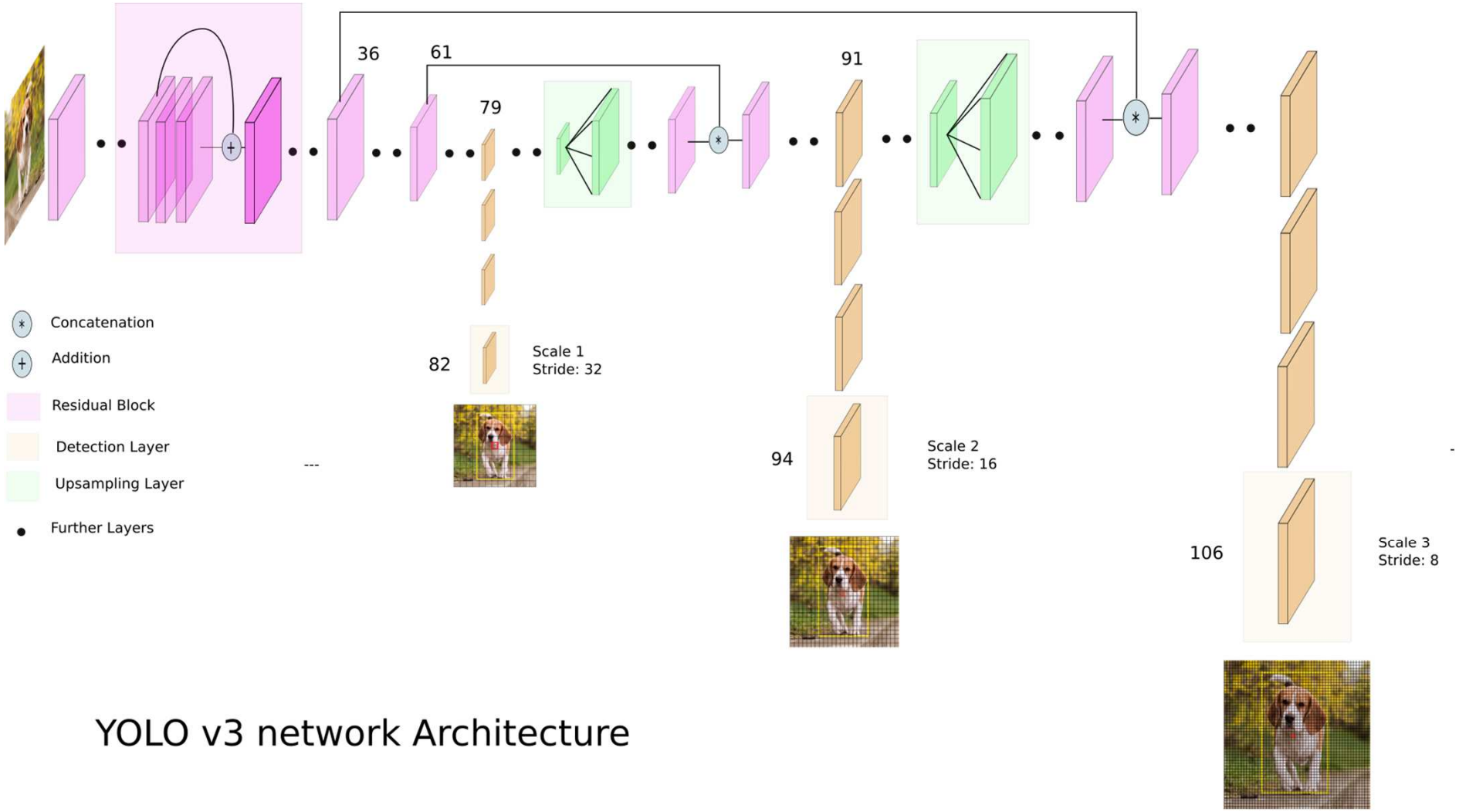
YOLOv2

- Zárt architektúra:
  - Sokáig csak a saját fejlesztő környezetében volt elérhető
- Gyors futás:
  - 300 × 300-as képekre ~45 fps
- Pontossága korlátozott:
  - Érzékeny az előre meghatározott ROI-kra (melyek száma viszonylag alacsony)
  - Skála és pozíció érzékeny (legalábbis a kelleténél jobban)
    - Kis objektumoknál csapnivaló eredmény
    - Bár ez szabályozható a boxok számának megválasztásával

YOLOv3

- 3 skálán keres objektumokat
  - Lassabb, mint elődjei, de kevésbé skála érzékeny
  - Mindegyiken 3-3 anchor, melyek meghatározására a tanító mintákon lévő objektumok bb-jének klaszterezését ajánlják
  - Lassabb lett: ~30 FPS
- Architektúrális módosítások
  - Res. blokk, skipp conn., lin. interpoláció, batch normalizáció
- Hibafüggvény módosítása
  - C_i és $p_i\left(c\right)$tanítása bináris kereszt entrópiával
  - $p_i\left(c\right)$ logisztikus szigmoid aktiváció kimenete (nincs már rajta softmax, tehát több osztályba is tartozhat egy-egy anchor)

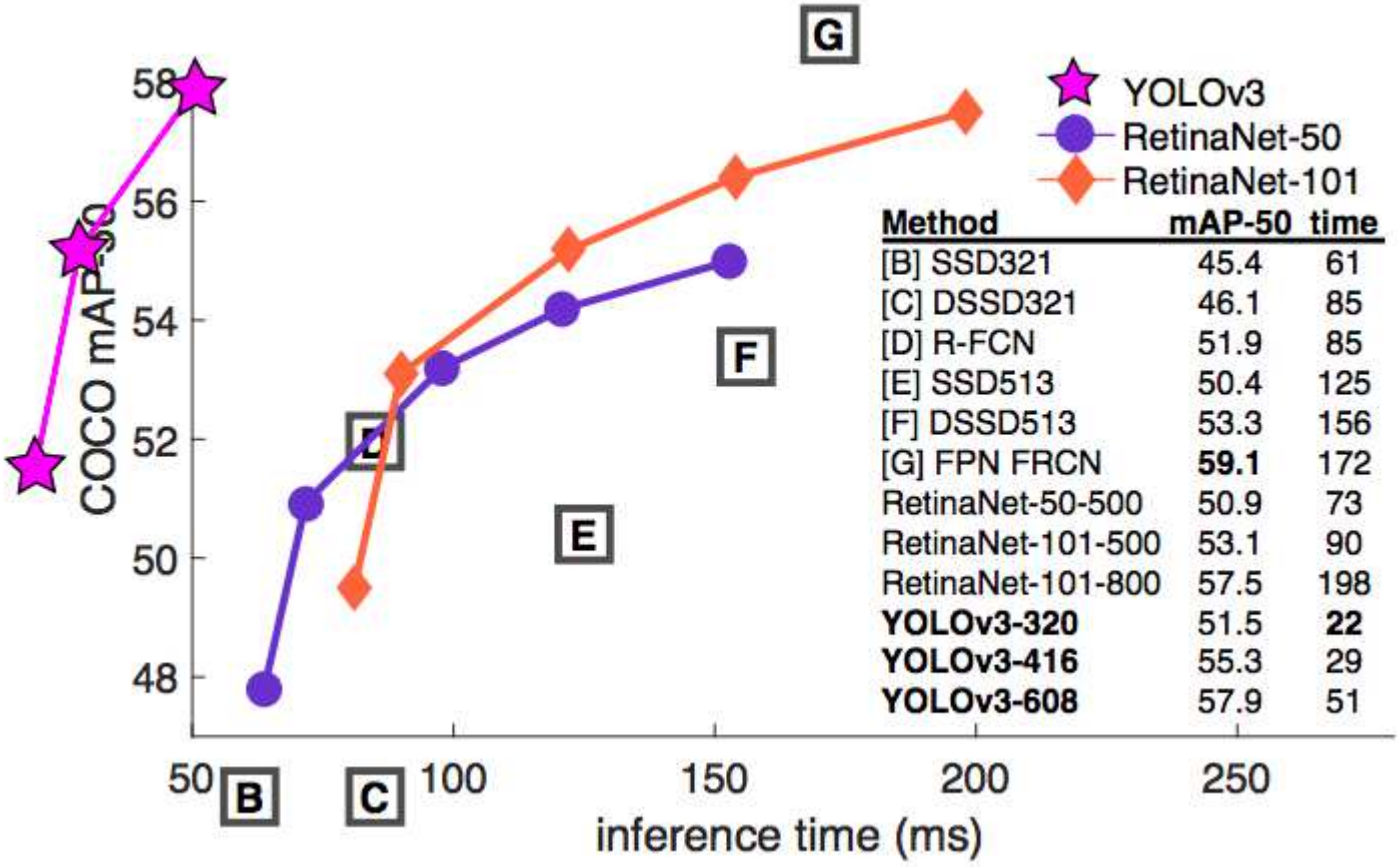
YOLOv3



YOLO v3 network Architecture

YOLOv3

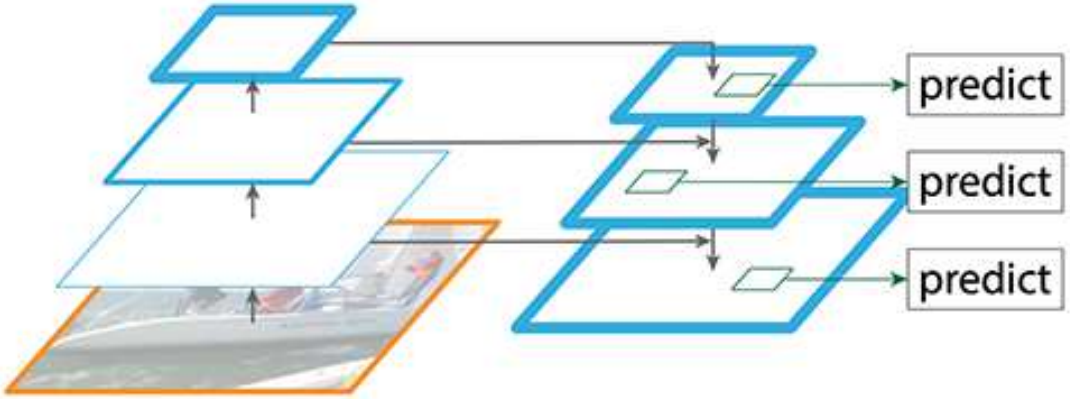
- Szórakozott teljesítménykiértékelés:



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

RetinaNET

- Feature Pyramid Network:
  - Cél az RPN-t több skálára futtatni (ezáltal jobb pontosság)
  - A klasszikus CNN-ek nagyobb felbontású jellemző térképei erre alkalmatlanok (bementhez közeliek, ezért csak alacsony absztrakciójú objektumokat emelnek ki (pl. élek))
  - Ötlet: top-down irányba residual connection-ökkel állítsunk elő egy *jellemzőtérkép piramist* (más szinten más felbontás)



RetinaNET

- Focal Loss:
  - Sokkal több negatív anchor, mint pozitív, ami problémás
  - Viszont a nagy konfidenciával negatívnak taksált mintákat le lehetne súlyozni (ezáltal redukáljuk a minták nagymértékű kiegyensúlyozatlanságát)
  - Bináris keresztentrópia:

$$CE(pt) = -\log(pt) \;:\; pt = d \cdot p + (1-d)\cdot(1-p)$$

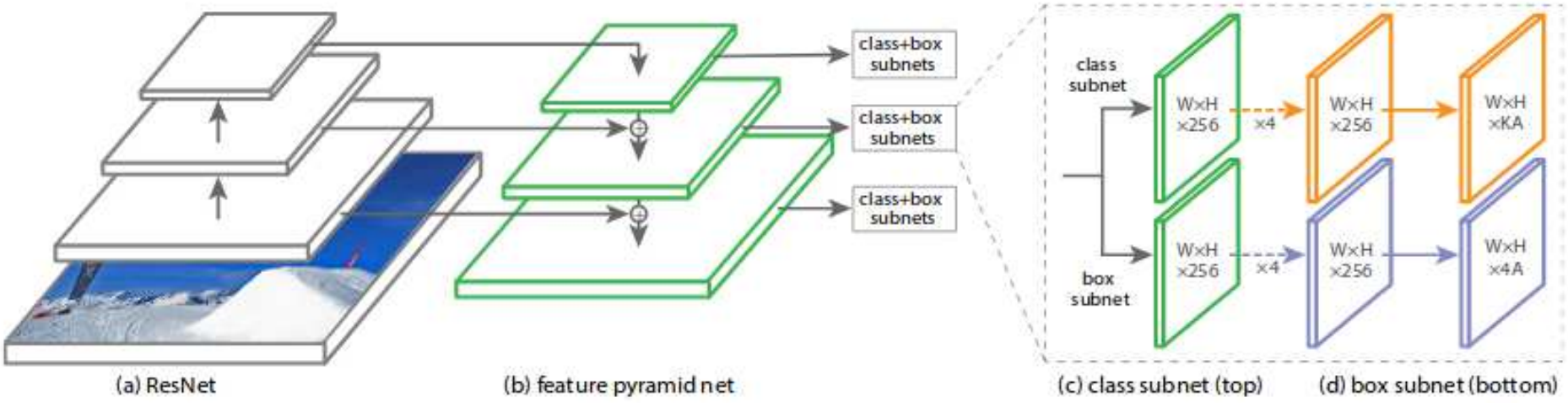
  - Új loss függvény (focal loss):

$$FL(pt) = -(1-pt)^{\gamma}\log(pt)$$

    - γ : hiperparaméter – konfidencia szerinti súlyozást definiálja

RetinaNET

- Teljesen konvolúciós háló
- Osztályozásnál Focal Loss-t alkalmaz
- Regressziónál Huber büntetőfüggvény
- Teszt időben non maxima supression-el fésüli össze a különböző szintek átlapolódó jelöltjeit (szintenként A db anchor)

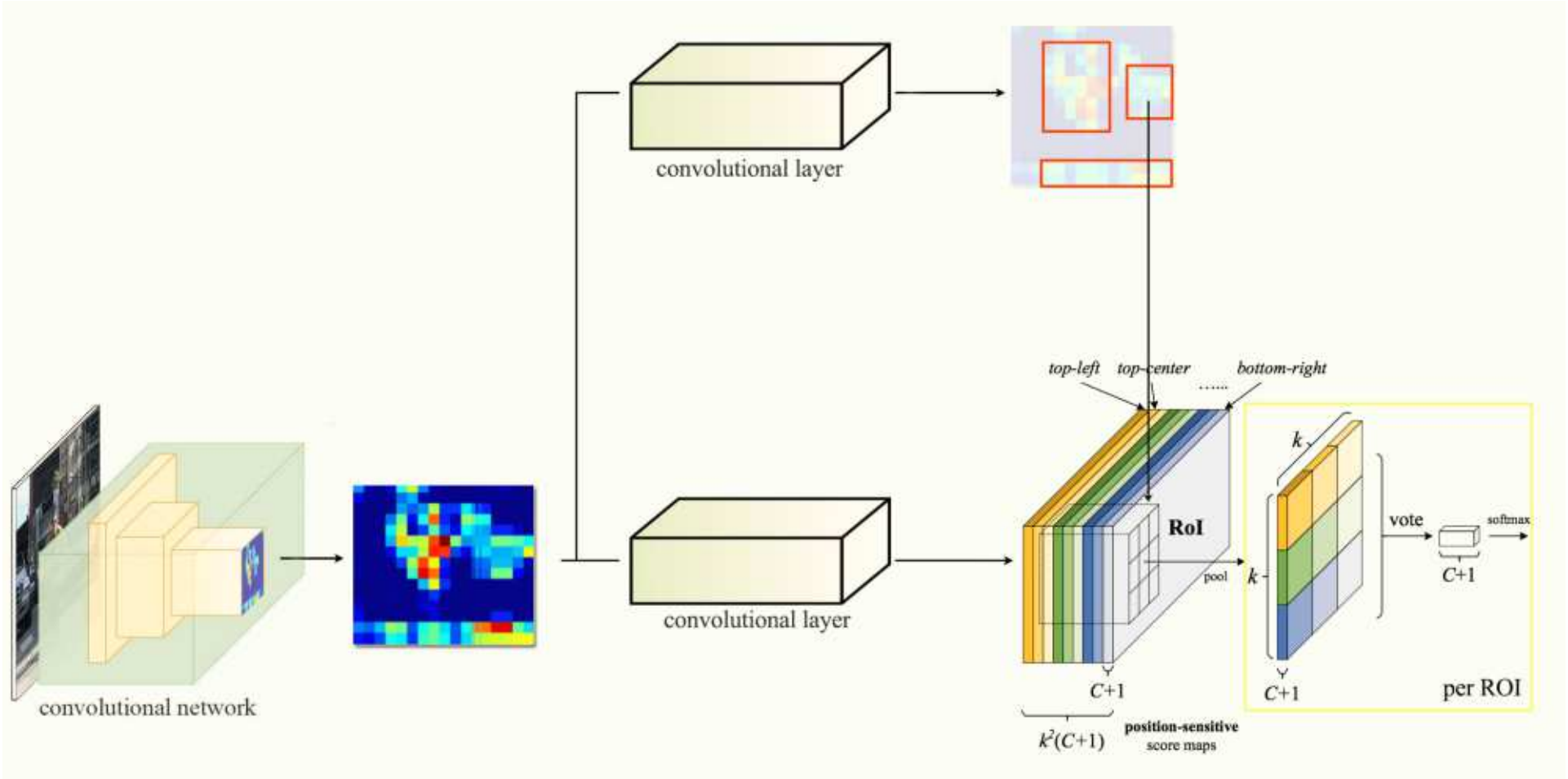


R-FCN

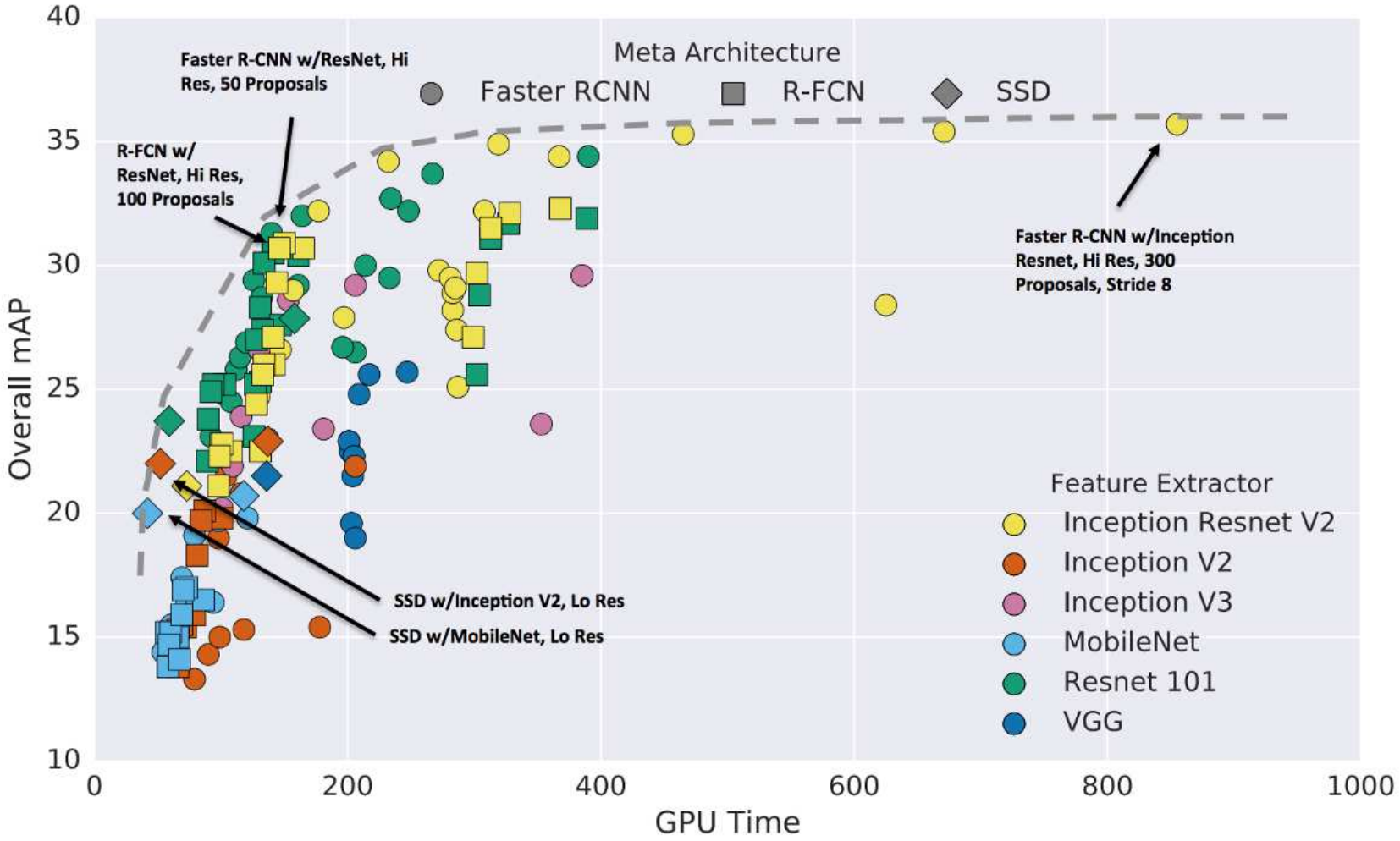
- Pozíció érzékeny ROI pooling:
  - ROI-kat egy Faster R-CNN alapú ROI jelölő háló állítja elő
  - $sc^{(r)}(c) = \sum_{k} \sum_{(x,y) \in R^{(r)}(k)} scm^{(c,k)}(x,y)$
  - Score-ok felett régiónként softmax…
  - Motiváció: így meg tudja tanulni a háló, hogy egy összetett objektum mely részén milyen jellemzőket kell keresnie (ezáltal jobb lok.)
- Ötvözi a régió alapú és FCN megközelítést
  - Gyorsabb a Faster R-CNN-nél ~10 FPS MS COCO-n
  - Pontosabb, mint a YOLO / SSD

R-FCN

- Architektúra:



Meta architektúrák összehasonlítása



Meta architektúrák összehasonlítása