

Mély tanulás

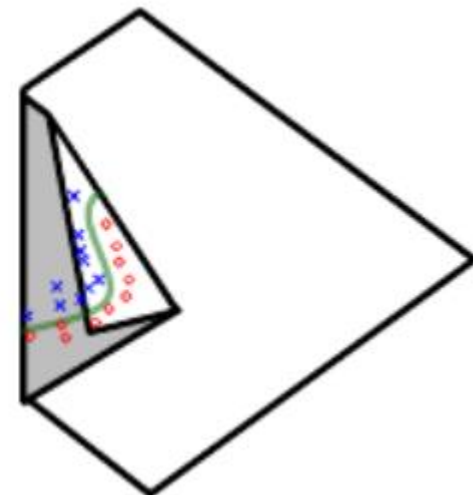
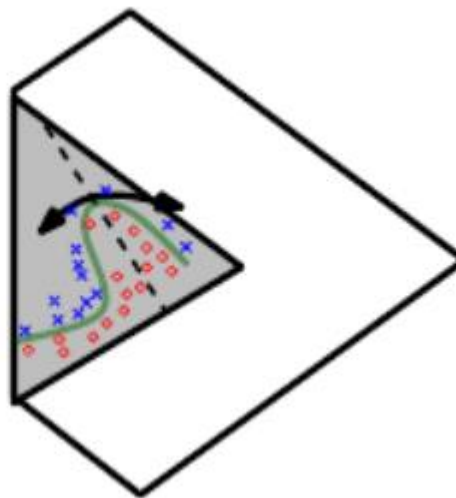
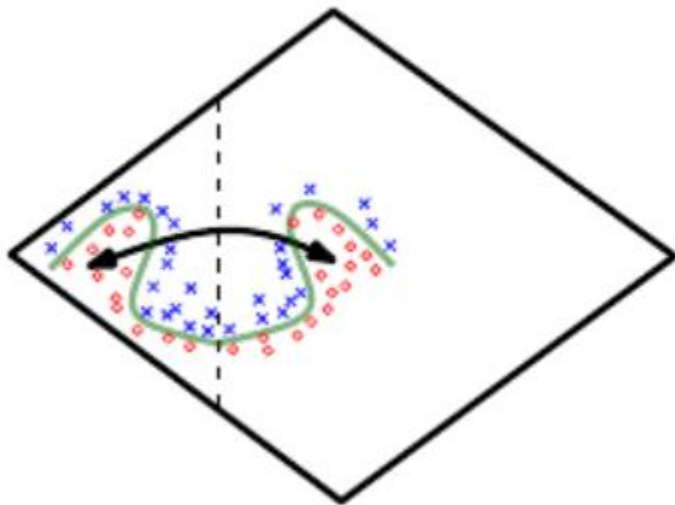
Hadházi Dániel

BME IE – 338

hadhazi@mit.bme.hu

Motivációk

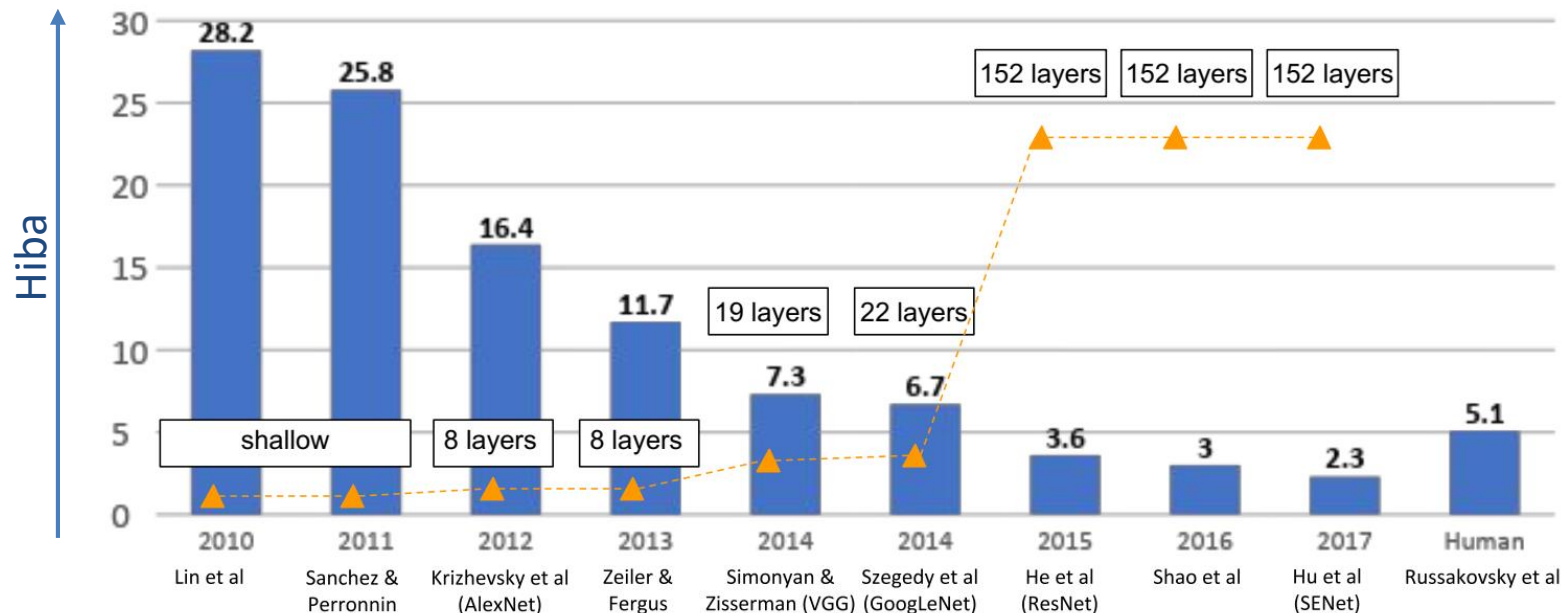
- Feladat dekomponálás elve:
 - Pl. Abs nemlinearitás alkalmazásának esetén:



- Részproblémák megoldása egyszerűbb, ember is így működik (pl. arc felismerése)
- Strukturális bias természetesebb, döntési felület szimmetriái, több reprezentáció transzformáció...

Motivációk

- Feladat dekomponálás elve:
 - A bemeneti domént a háló mélységének exponenciális számú részére partícionálja (tipikusan lin. leképzés)
 - Általánosan minél több réteg a preferált, **amíg az egyes rétegek értelmes feladatot képesek végezni**



KONVOLÚCIÓS NEURÁLIS HÁLÓZAT (CNN)

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

- Képi objektumdetektáló megközelítések:

Traditional Pattern Recognition: Fixed/Handcrafted Feature Extractor



Mainstream Pattern Recognition



Deep Learning: Multiple stages/layers trained end to end



Tipikus felépítés

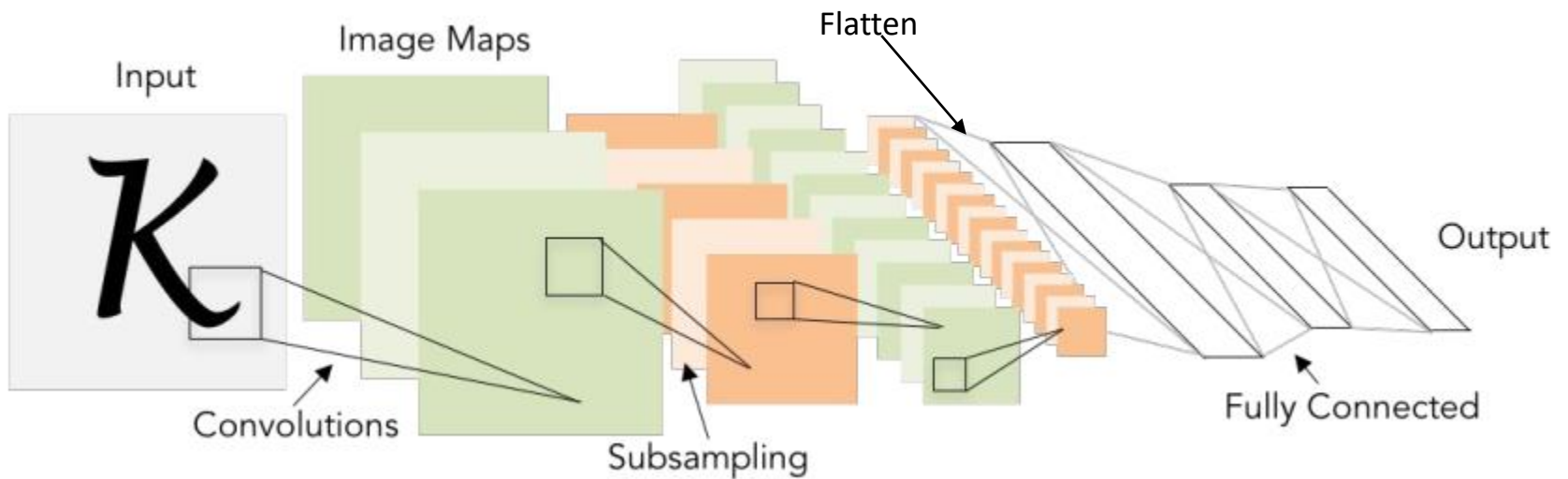


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

Konvolúciós réteg

- Motiváció:
 - Klasszikus képfeldolgozásnál alpművelet a konvolúció:
 - Zajszűrésre
 - Alacsony képi jellemzők kiemelésére (pl. élek, sarokpontok)
 - Összetett objektumok kiemelése (pl. illesztett szűrés)
 - Konvolúció $\equiv$ 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:

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

- $o_{(z)}^{(l)}$: l -edik réteg z -edik neuronjának súlyozott összegképe (rövidebben l -edik réteg z -edik csatornája), pixelenként erre hívódik majd meg a nemlinearitás
- $y_{(c)}^{(l-1)}$: $l-1$. réteg c . csatornájának paddelt változata (szokás aktivációs térképnek is hívni)
- 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:

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

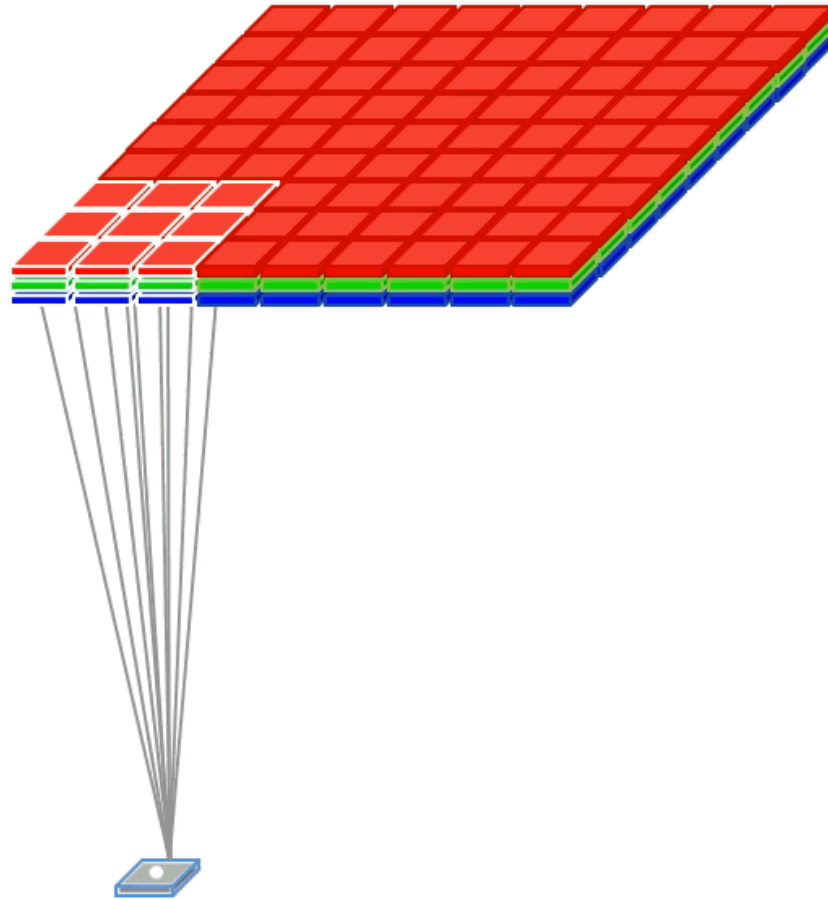
Helyett gyakorlatilag mindig korreláció történik:

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

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

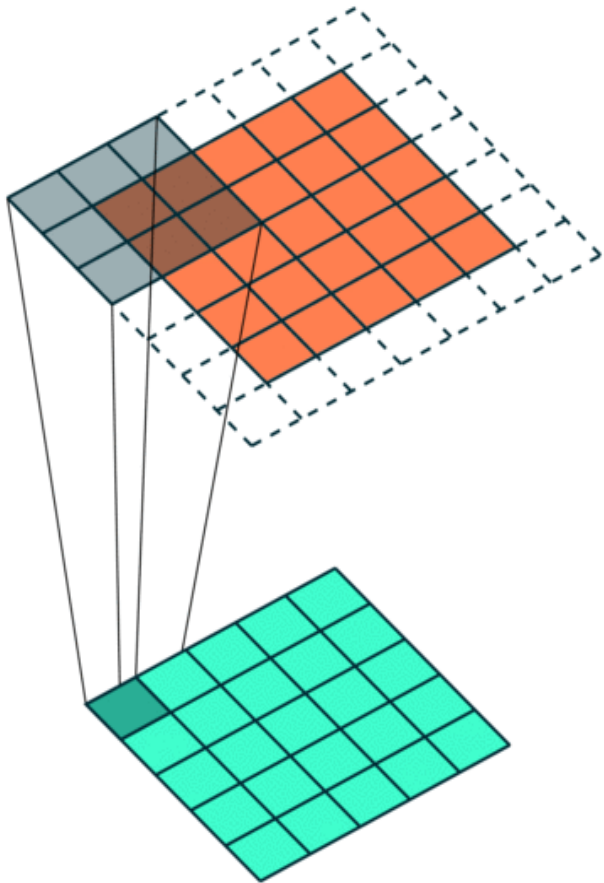
Konvolúciós réteg

- Szemléltetés:

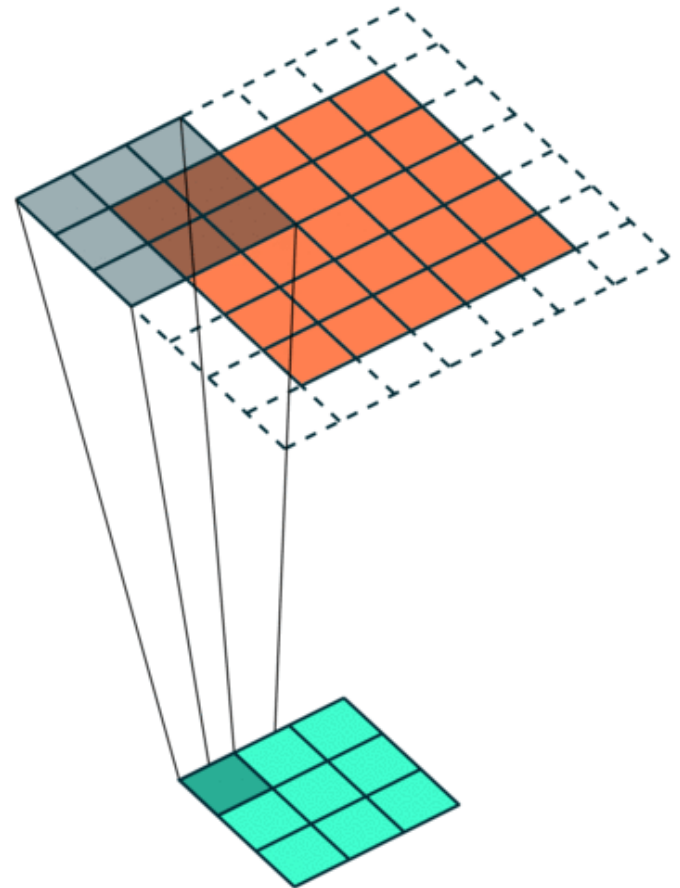


Konvolúciós réteg

- Lépésköz(stride) paraméter hatása:

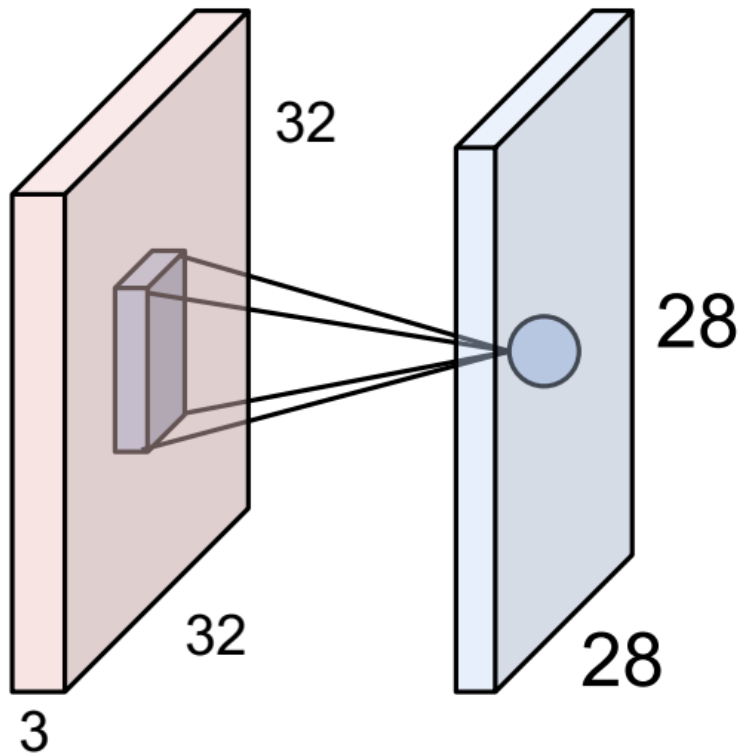


$s = 1$



$s = 2$

Konvolúciós réteg



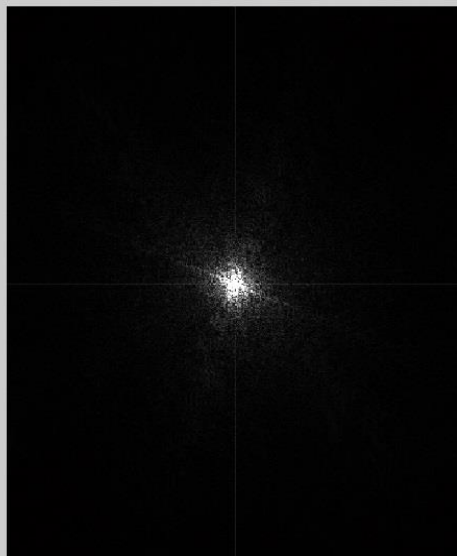
- Neuron érzékenységi mezője: a bemeneti kép azon része, melytől függ a kimeneti értéke
- A példában az aktivációs térkép minden pixele az adott pixel kp.-ú 5×5-ös képrészlettől függ

Konvolúciós réteg

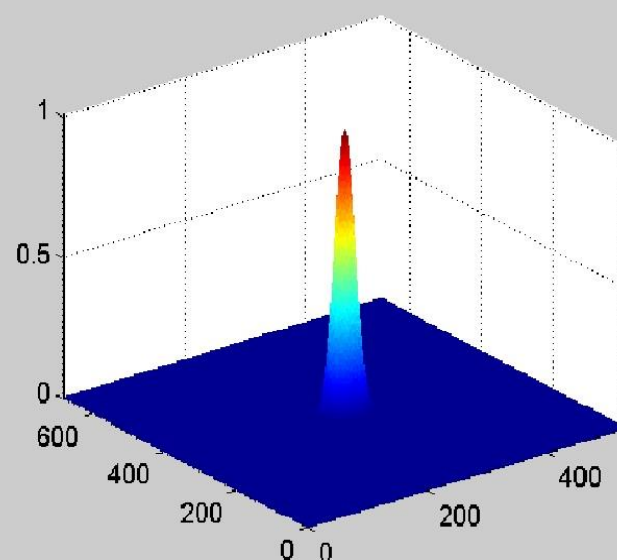
original image



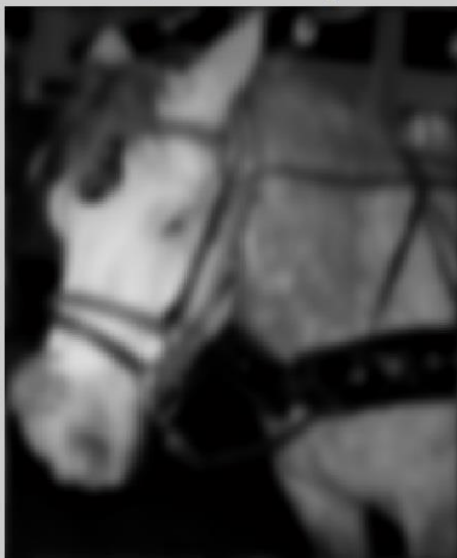
fft of original image



Gaussian LPF $H(f)$



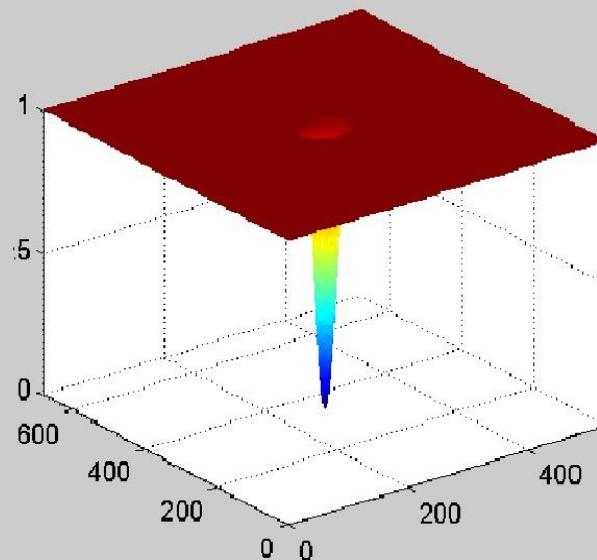
low pass filtered image



High pass filtered image



Gaussian HPF $H(f)$



Pooling réteg

- Motiváció:
 - Csatornák felbontásának csökkentése
 - Utána következő réteg szűréseinek érzékenységi területének növelése („kisebb szűrők is eleget látnak”)
- Lényegében mintavételezi a képet:
 - Average 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:

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

2×2-es pooling
2-es strázsával



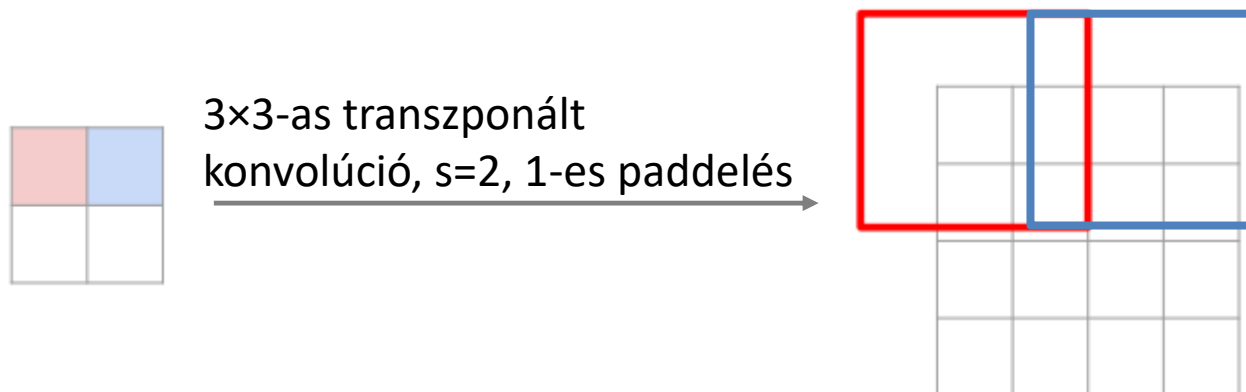
6	8
3	4

Transzponált konvolúciós réteg

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

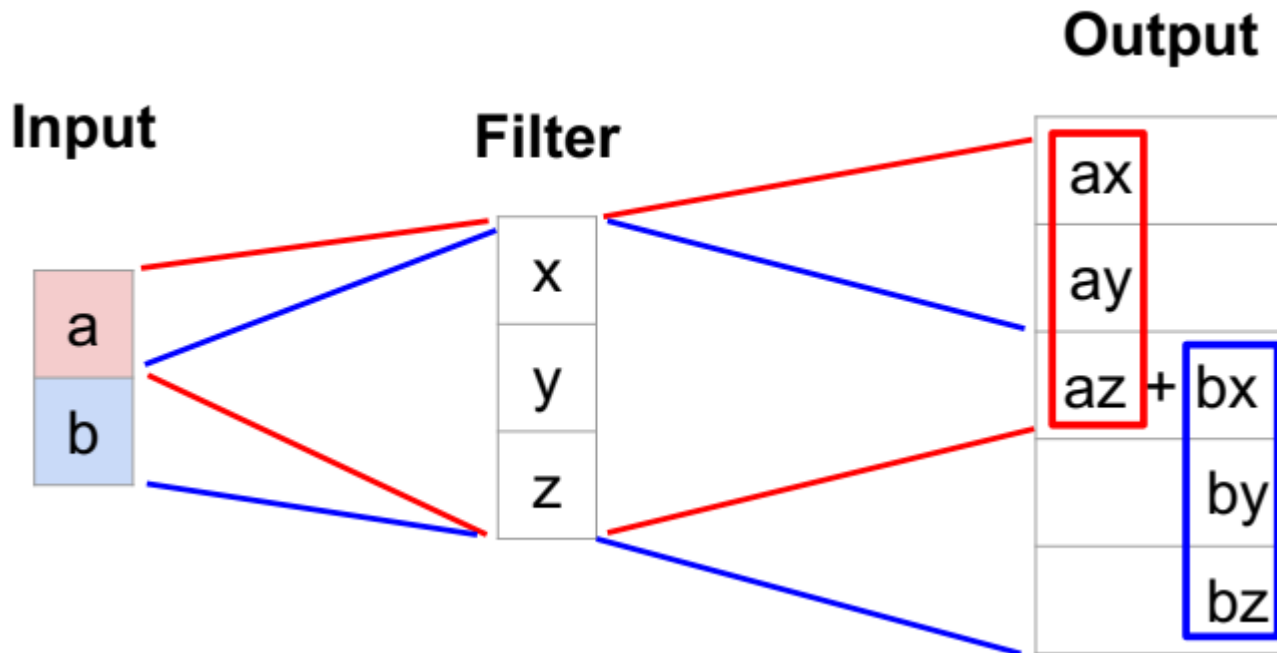
$$o_{(z)}^{(l)}(x, y) = \sum_c \sum_{(u, v)} y_{(c)}^{(l-1)}(u, v) \cdot w^{(l)}(x - u \cdot s, y - u \cdot s, c, z) + bias_{(z)}^{(l)}$$

- Szemléltetése:



Transzponált konvolúció

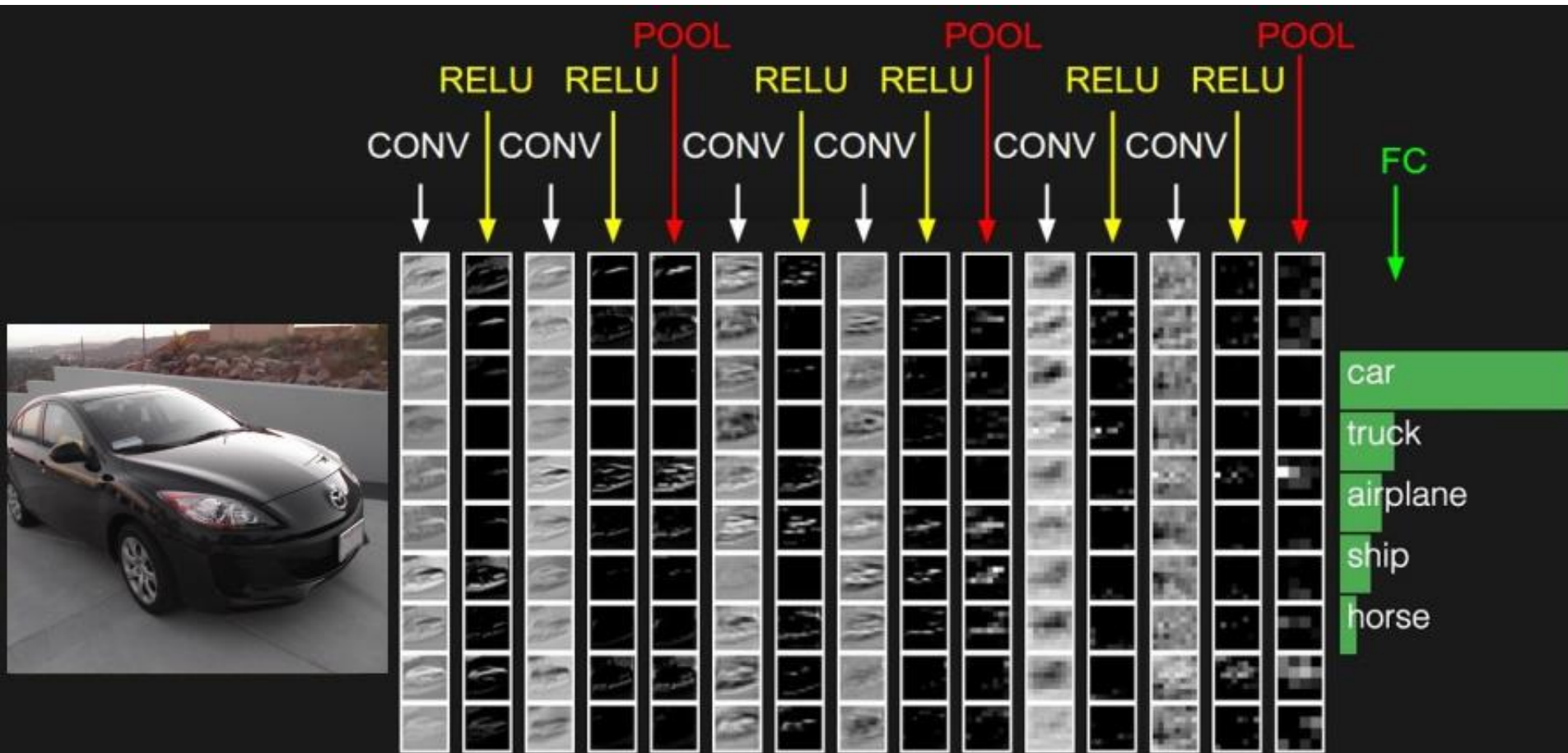
- 1D példa:



Sorosító réteg

- Angol elnevezése: Flatten
- Feladata többdimenziós jelek 1D-sé alakítása:
 - Fully Connected rétegek bemenetén szokták alkalmazni
- Nem tartalmaz tanítható paramétert
- Alkalmazása veszélyes lehet

Működés szemléltetése

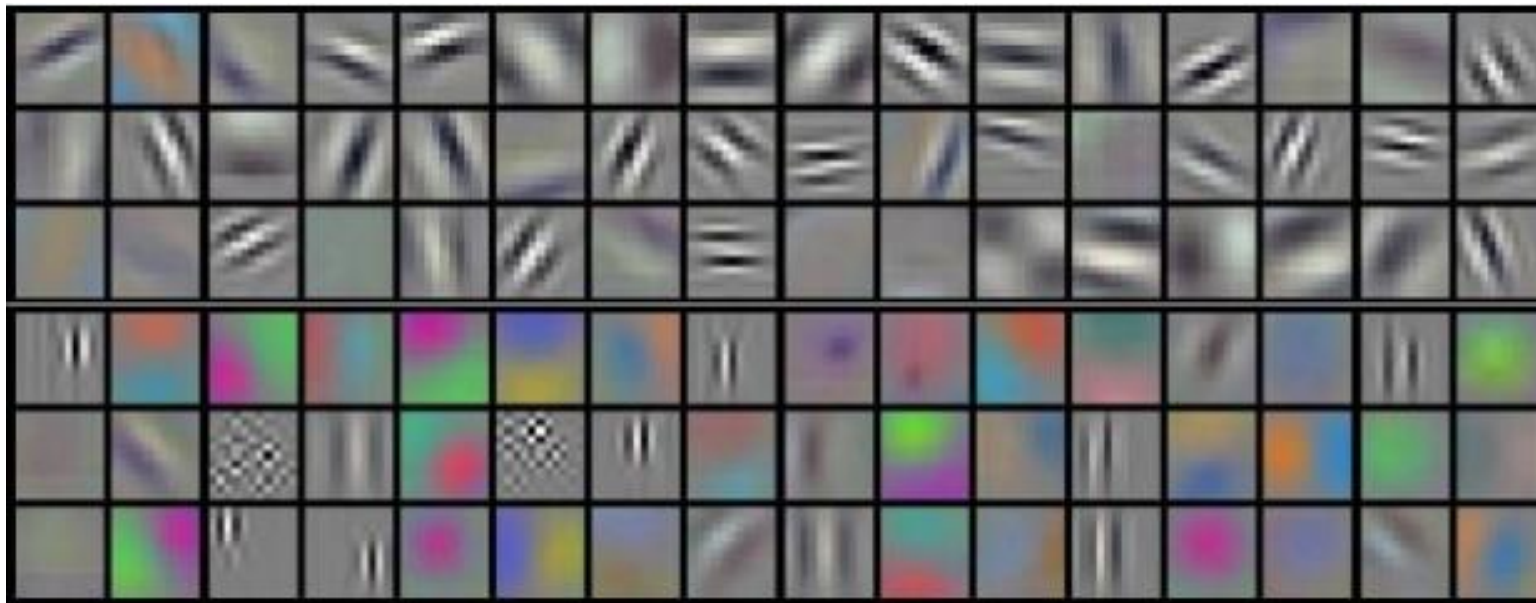


Működés értelmezése

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



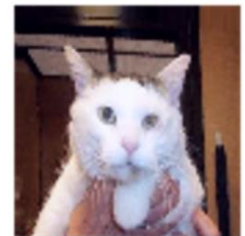
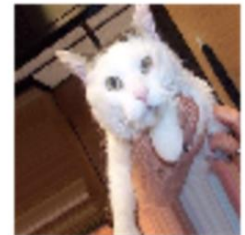
- Kimenethez közelebbi rétegek már összetett objektumokra érzékenyek

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:

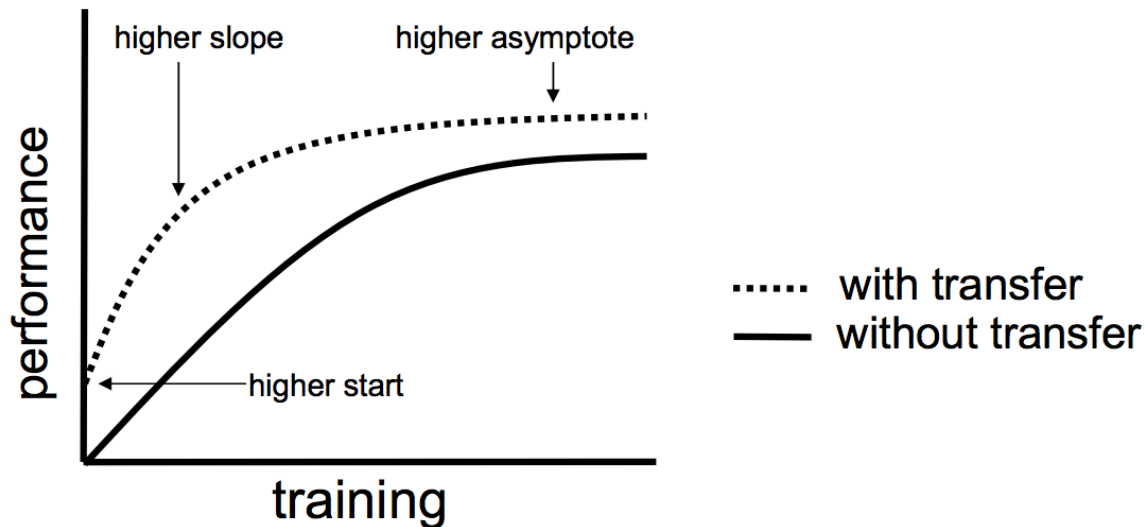


Tudás transzfer

- Konvolúciós hálók bemenethez közel olyan jellemzőket emelnek ki, melyek nem feladat specifikusak:
 - 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



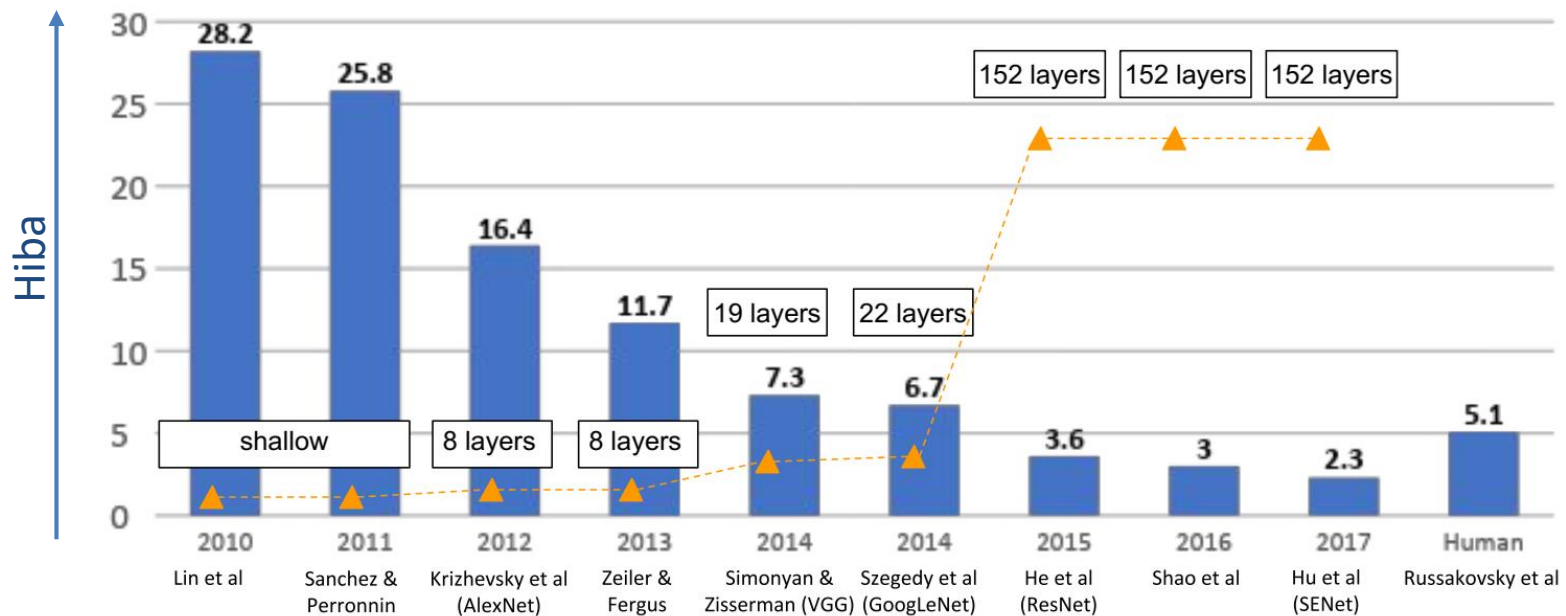
Transfer learning

- Általános ajánlás:

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

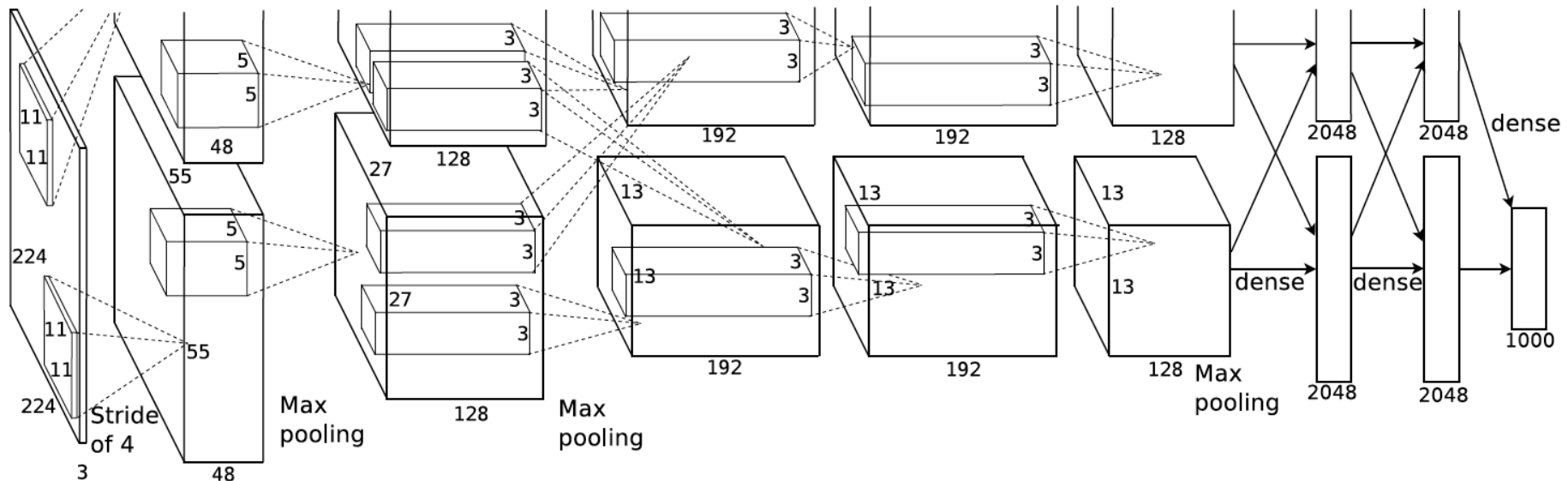
CNN architektúrák

- ImageNet Large Scale Visual Recognition Challenge:



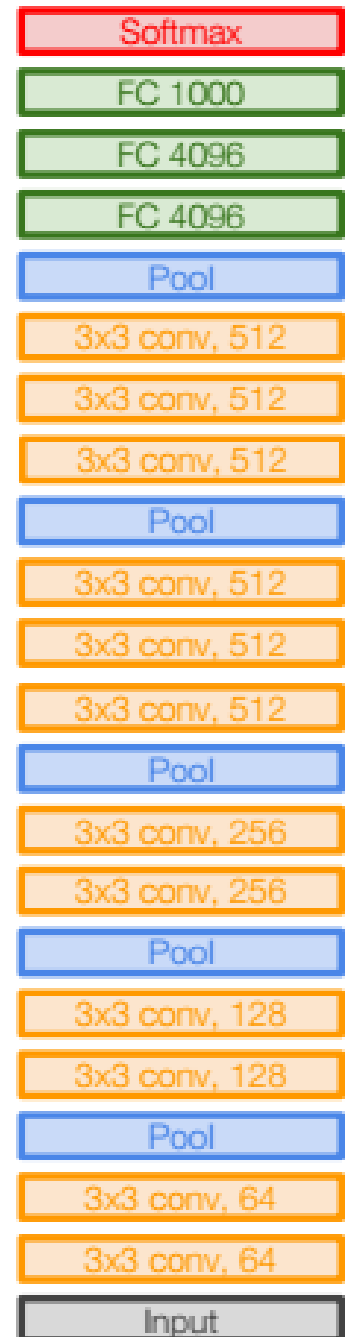
AlexNet (2012) (*)

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



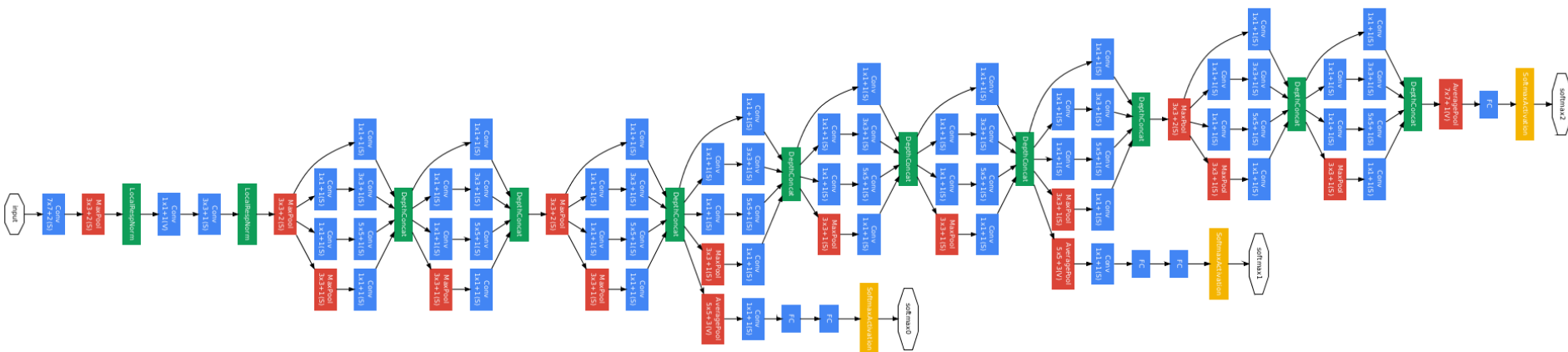
VGG (2014) (*)

- 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



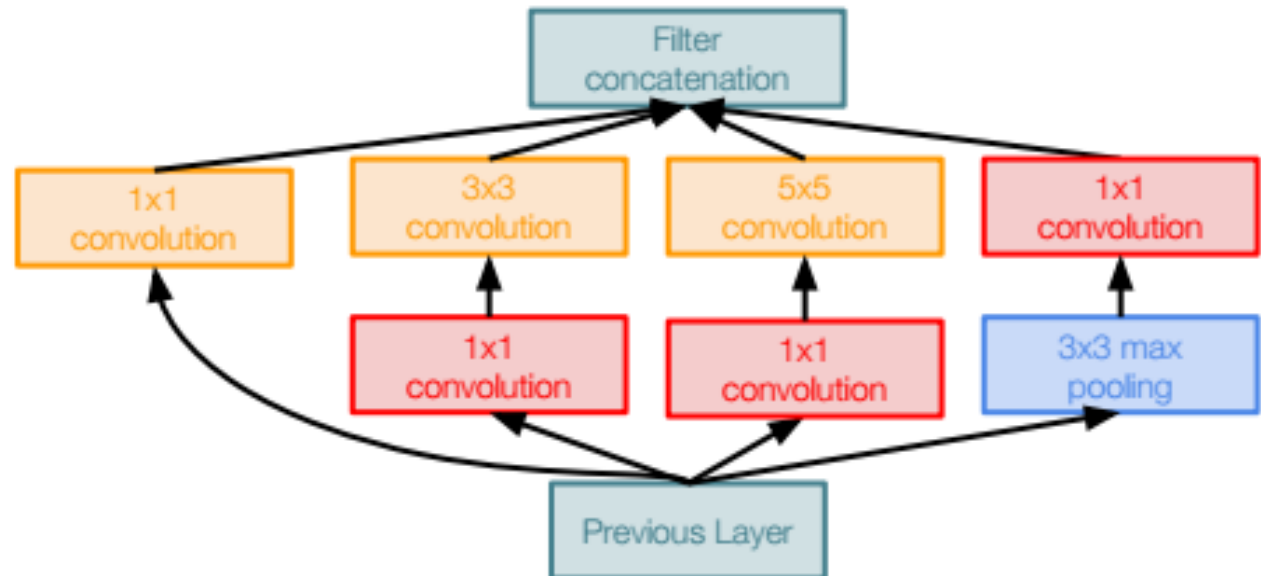
GoogleNet – Inception (2014) (*)

- 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



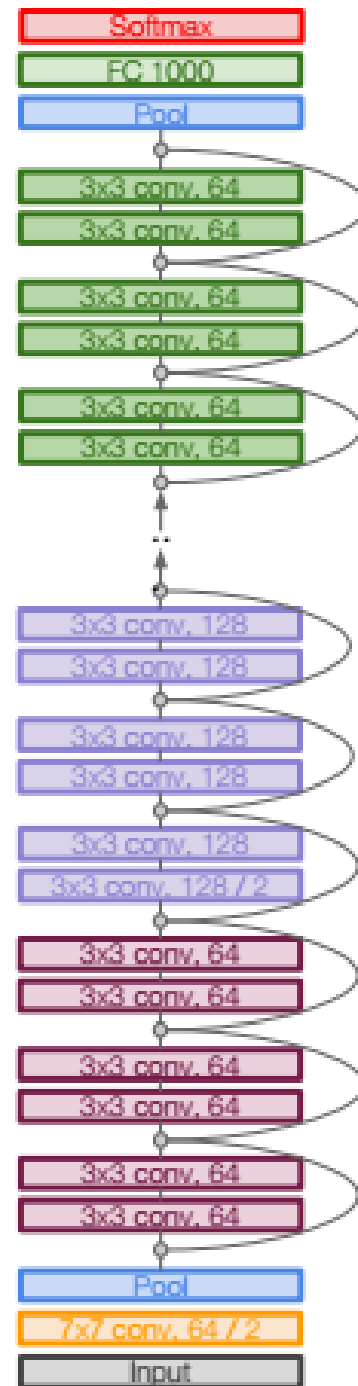
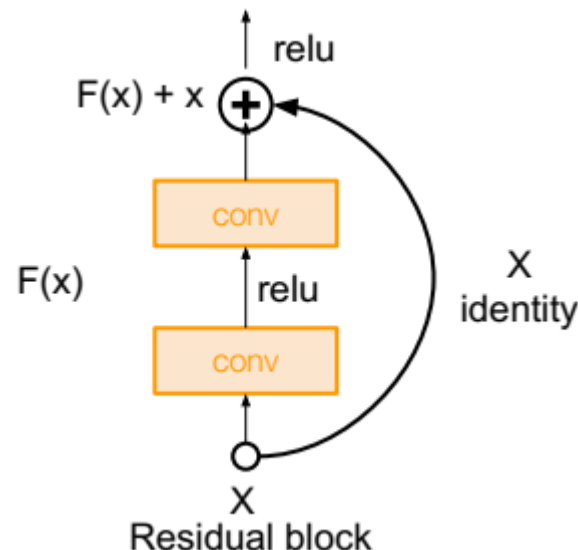
Googlenet – Inception modul (*)

- 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



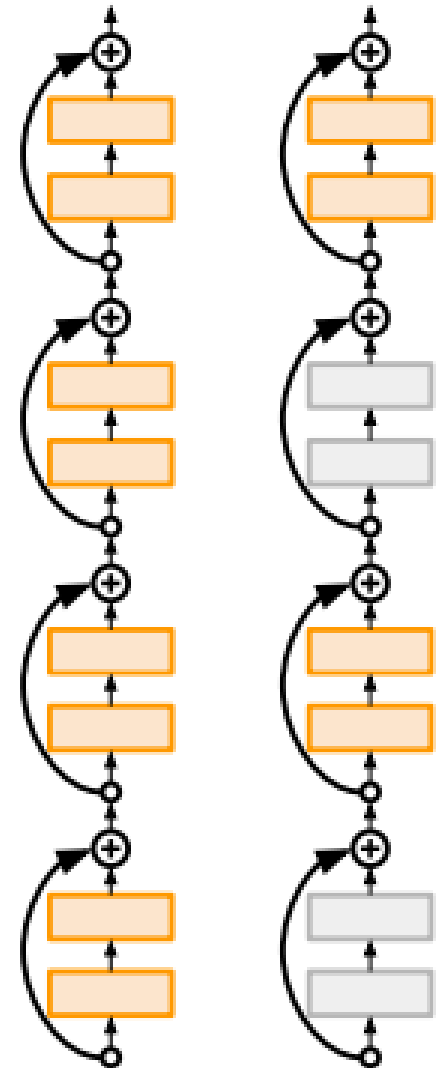
Resnet (2015) (*)

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



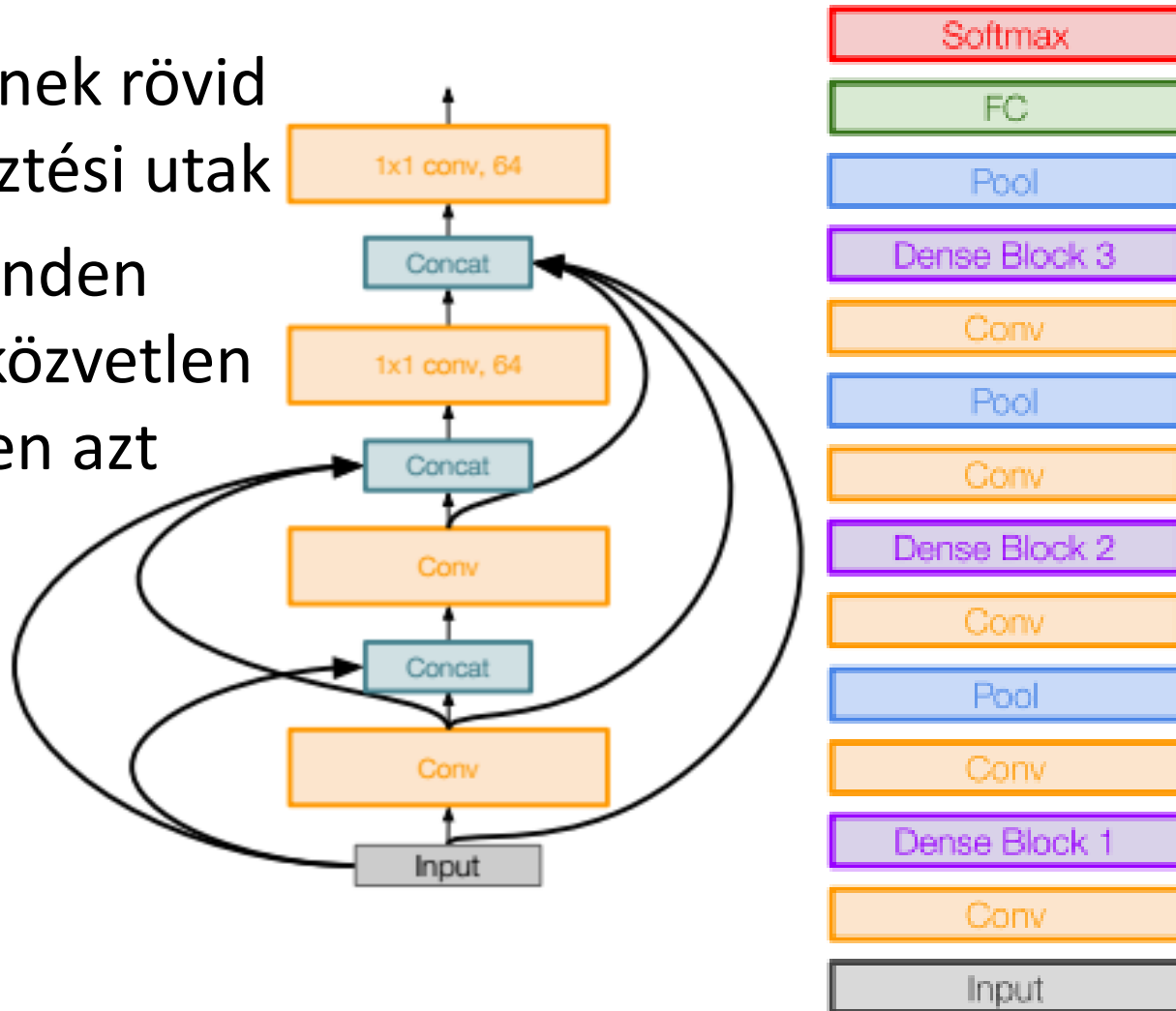
Stochastic Depth (*)

- Motiváció: egyszerűbb a hiba visszatérjesztése, 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)



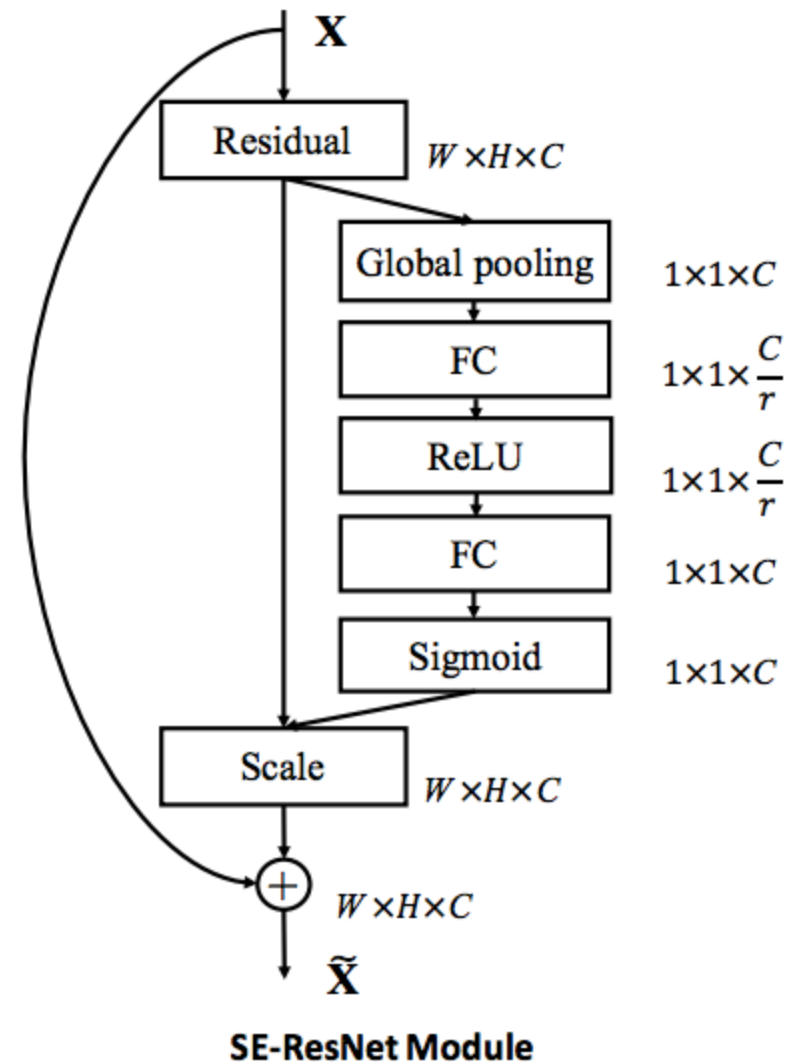
DenseNet (2017) (*)

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

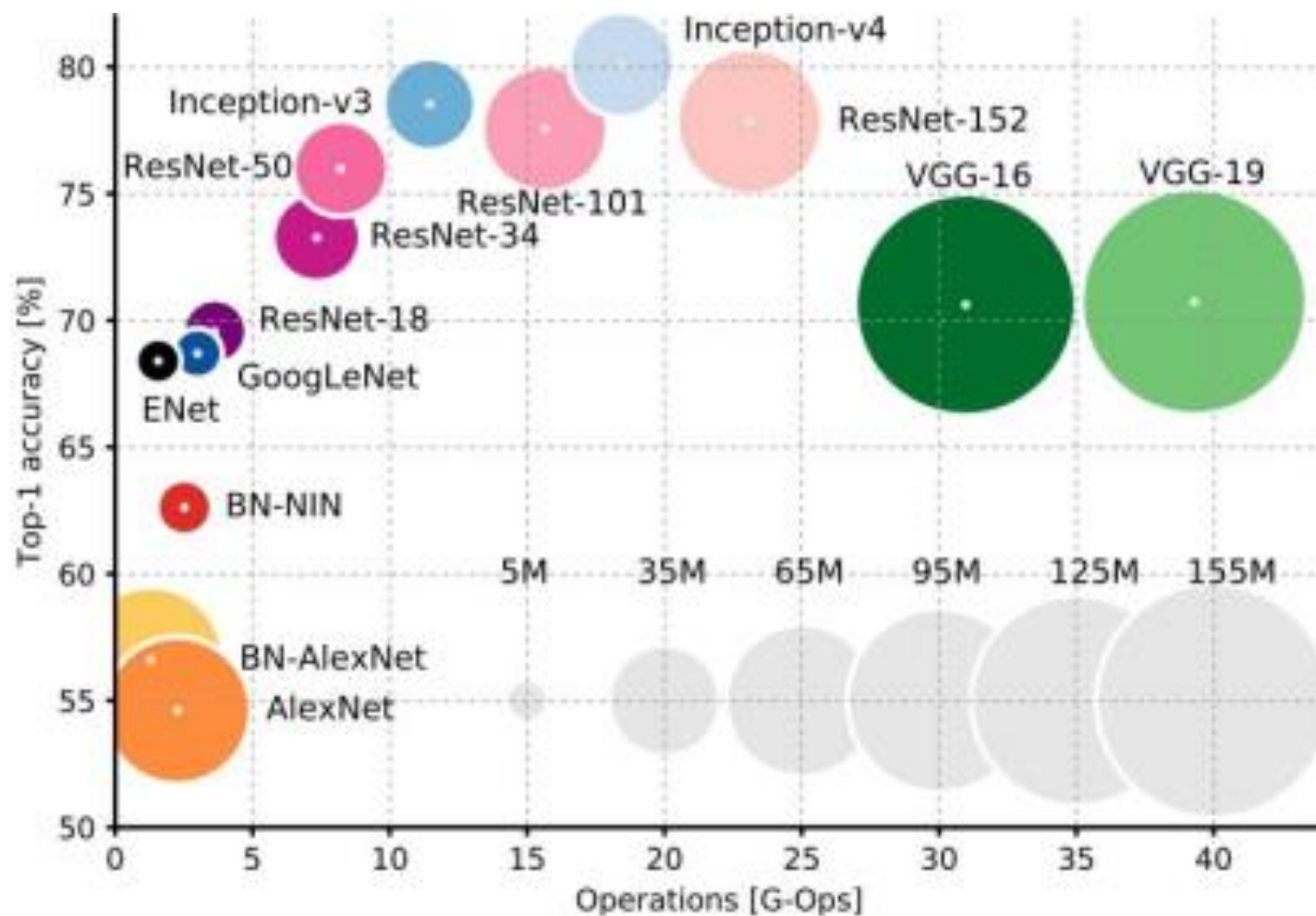


Squeeze and Excitation Networks (SENet) (2017) (*)

- Aktivációktól függően újrásúlyozza a konvolúciós csatornákat.
- Az újrásúlyozást egy két rétegű MLP végzi
- Látványos javulás:
SEResNet-50 $\approx$
ResNet 152



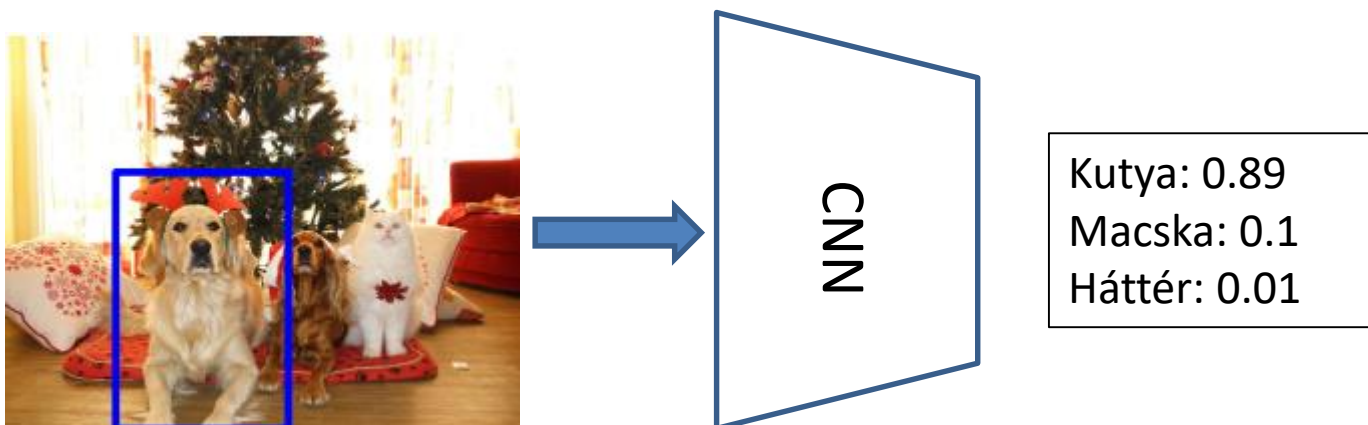
Háló komplexitása és pontossága



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

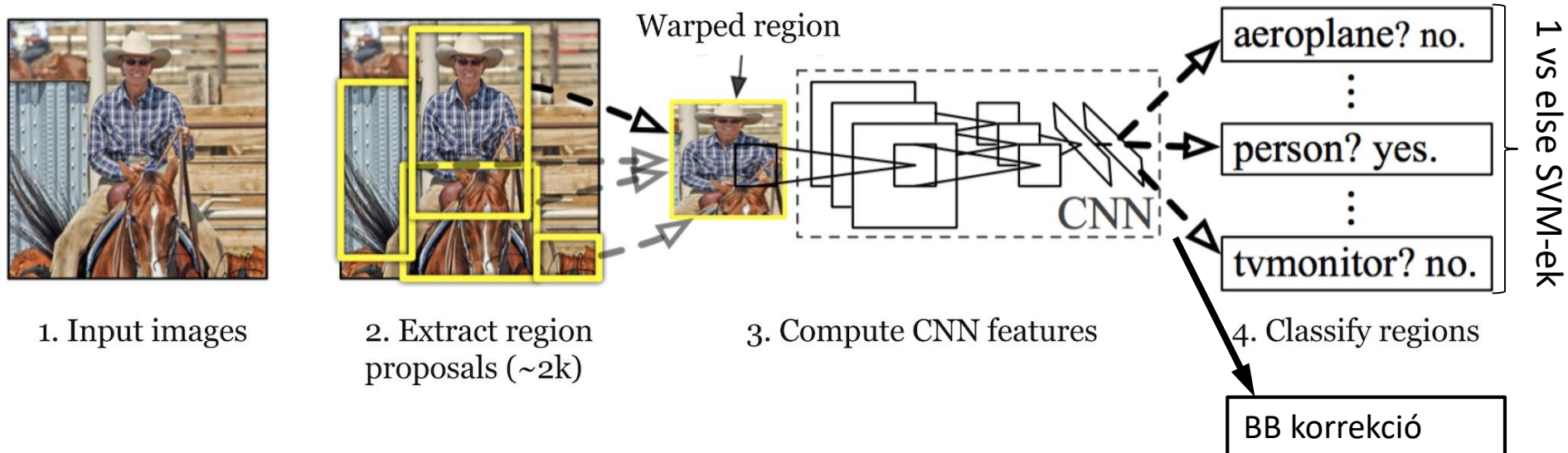


Régió alapú CNN-ek

- 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

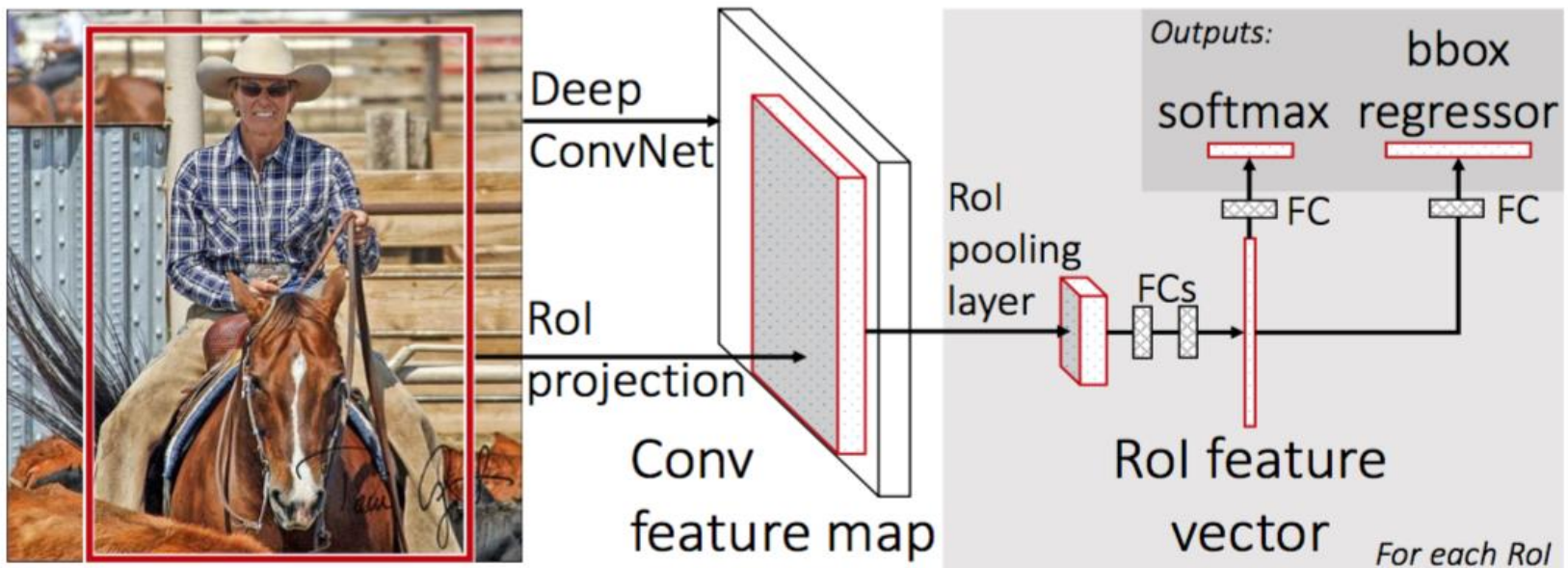


R-CNN

- 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 képrészletek 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

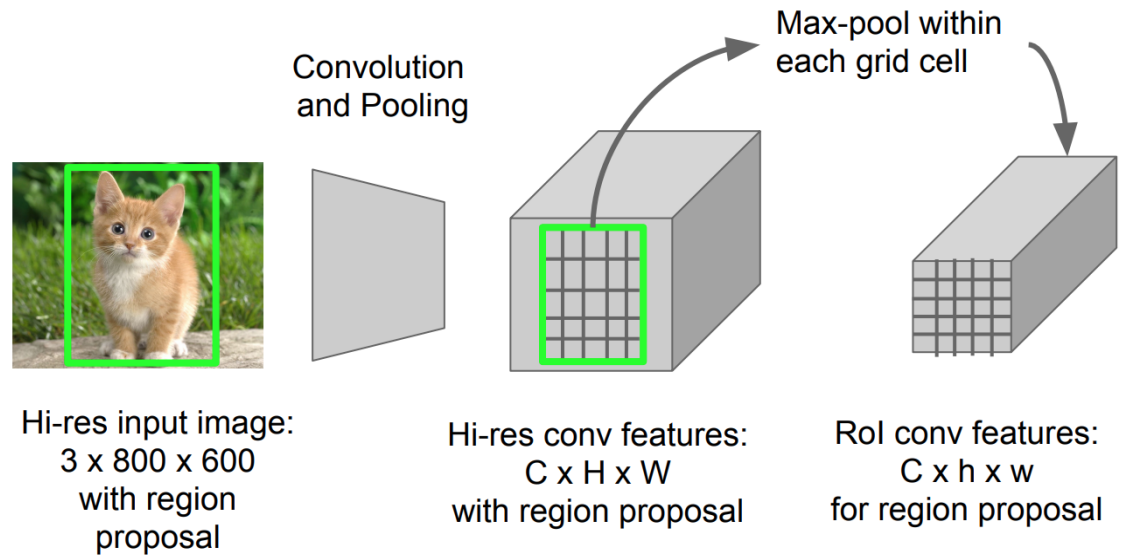
Fast R-CNN

- 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

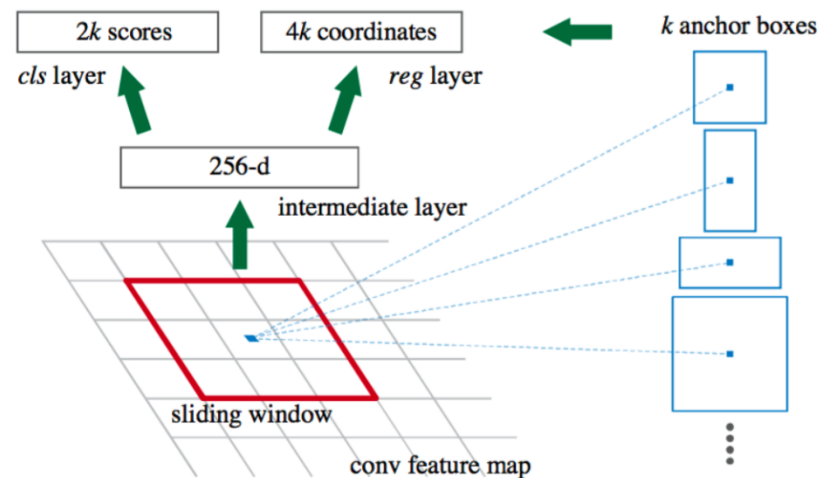
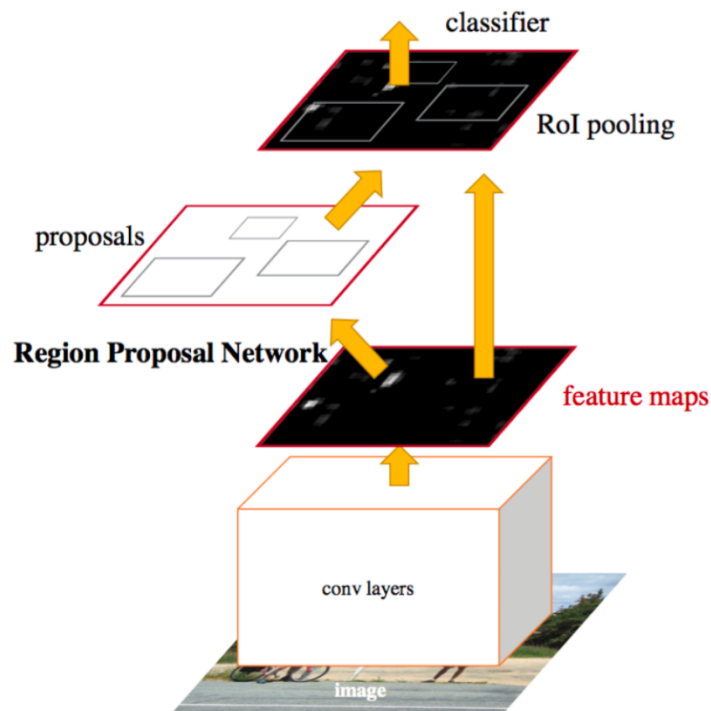
- ROI pooling háló:



- 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

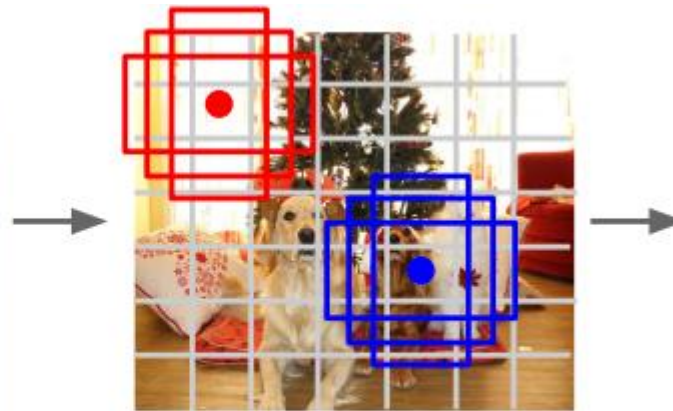
- Tanítása:
 1. ROI-kat előállító hálón képezünk csak hibát
 2. ROI kiemelő kimenete alapján tanítjuk a ROI pooling utáni rétegeket
 3. Fine tuning a ROI kiemelő rétegekre (és az alatta lévő konvolúciós részhálóra)
 4. Fine tuning csak a Fast R-CNN rétegekre
 5. 3.-4. lépés ismétlése
- Legpontosabb meta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



Input image
 $3 \times H \times W$



Divide image into grid
 7×7

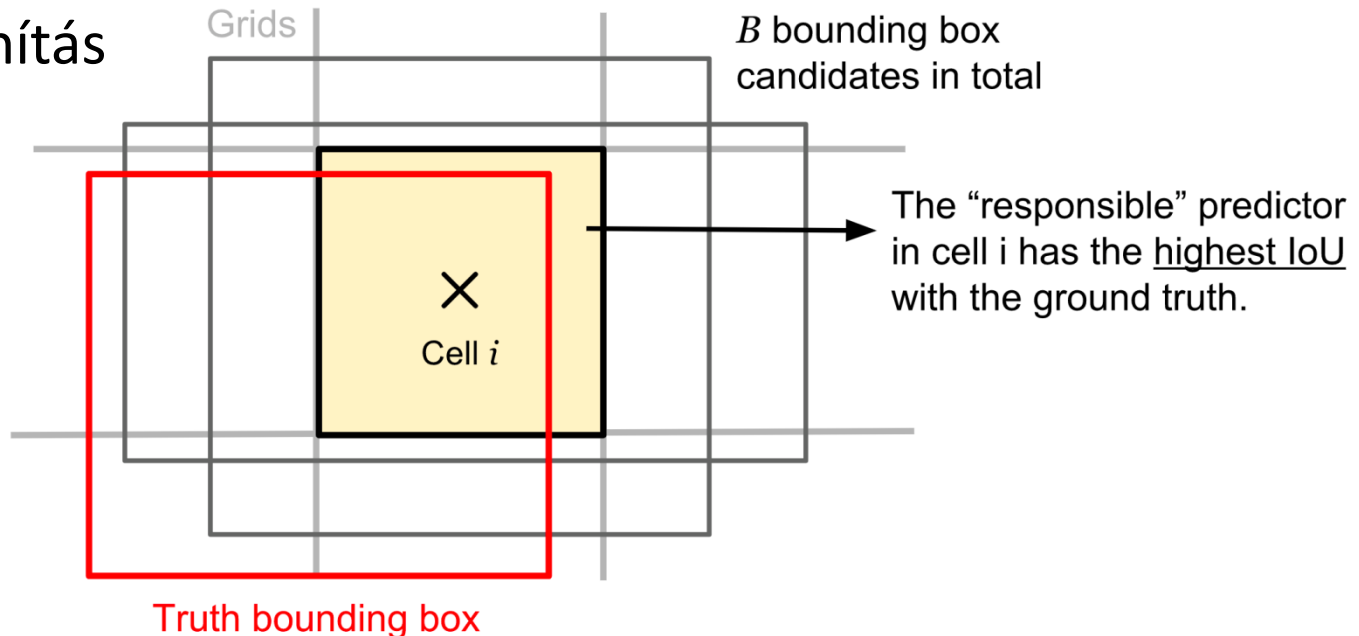
- Cellánként B box pozíciója: (dx, dy, dh, dw, obj. konfidencia)
- Cellánként egy osztályba sorolás (C-s softmax)

Kimenet: $7 \times 7 \times (5B + C)$

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

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

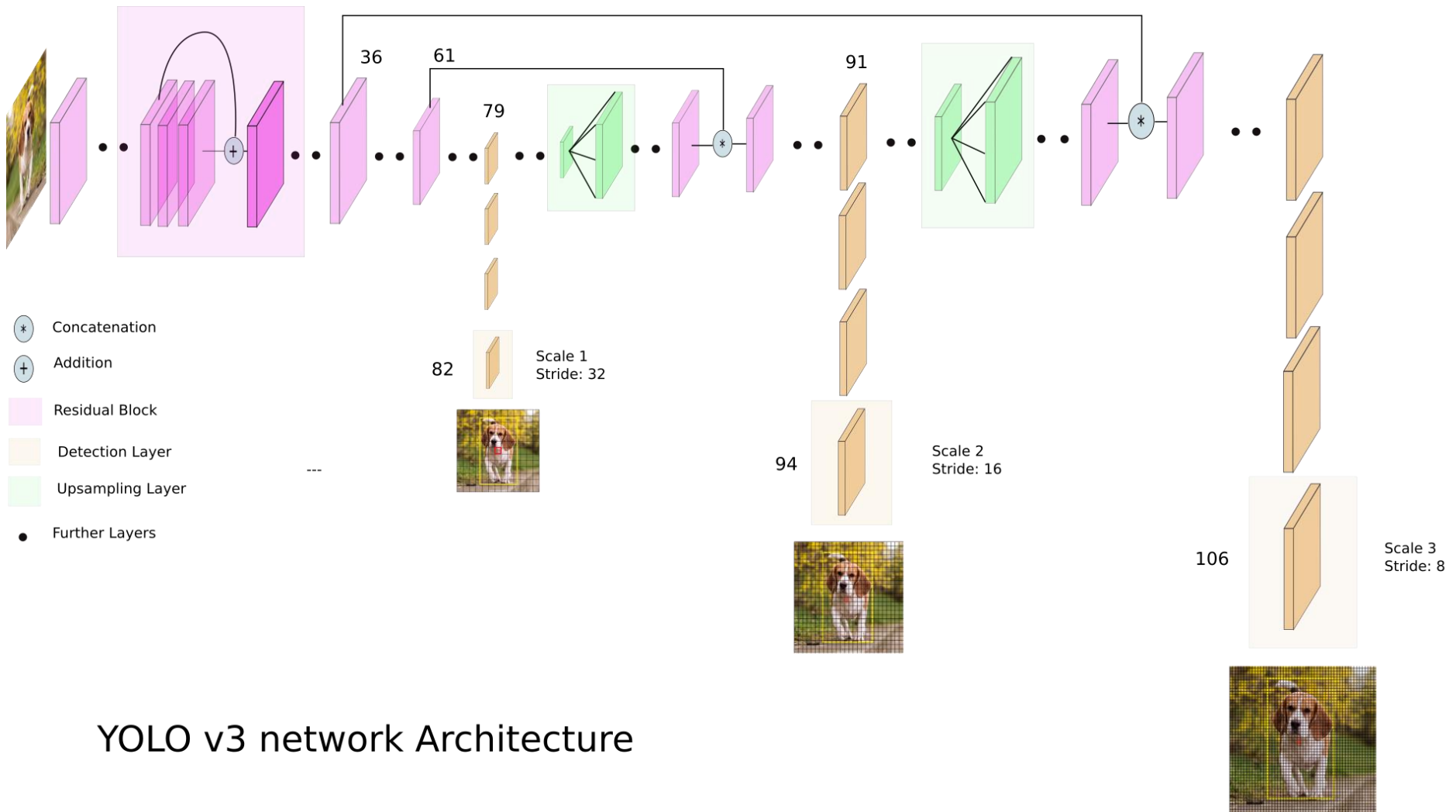
YOLOv2

- 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 $p_i(c)$ tanítása bináris kereszt entrópiával
 - $p_i(c)$ logisztikus szigmoid aktiváció kimenete (nincs már rajta softmax, tehát több osztályba is tartozhat egy-egy anchor)

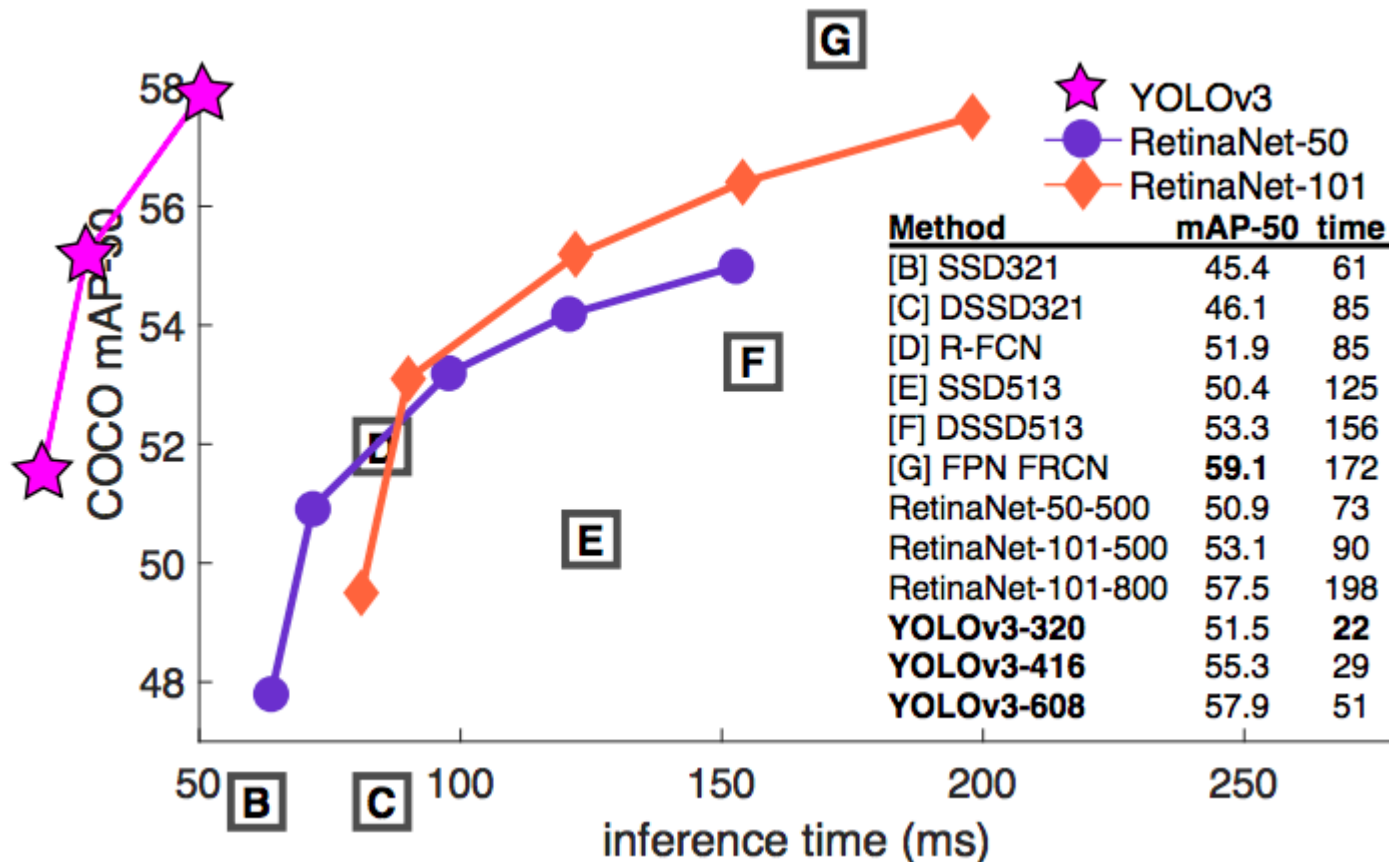
YOLOv3



YOLO v3 network Architecture

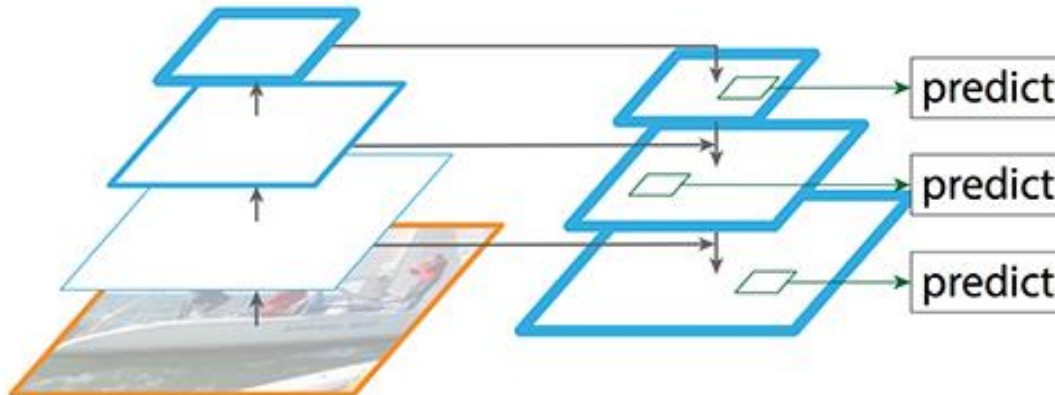
YOLOv3

- Szórakozott teljesítménykiértékelés:



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 keresztentropia:

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

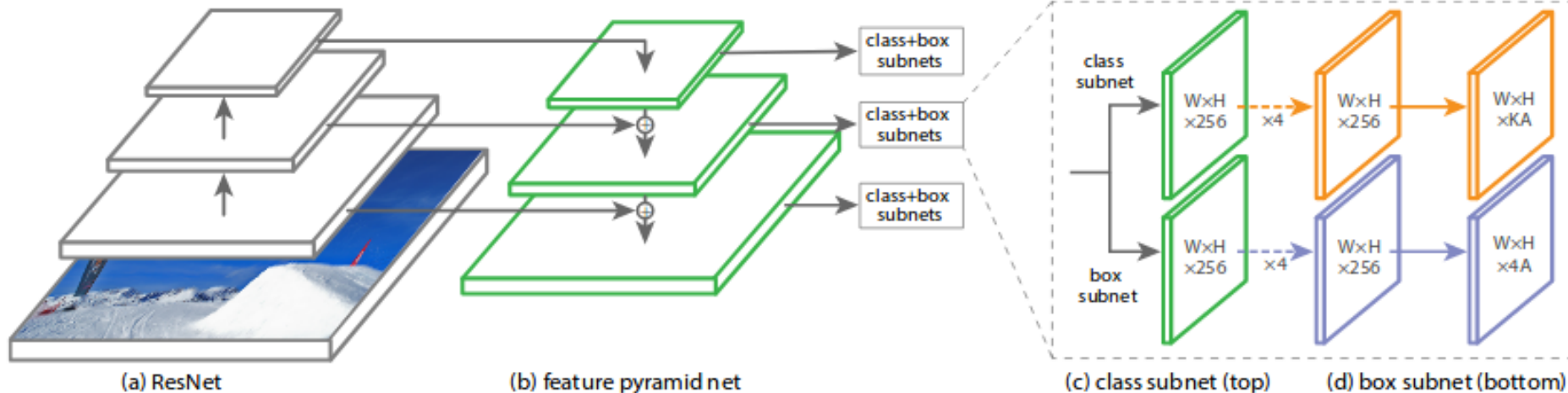
- Új loss függvény (focal loss):

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

- γ : hiperparaméter – konfidencia szerinti súlyozást definiálja

RetinaNET (*)

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

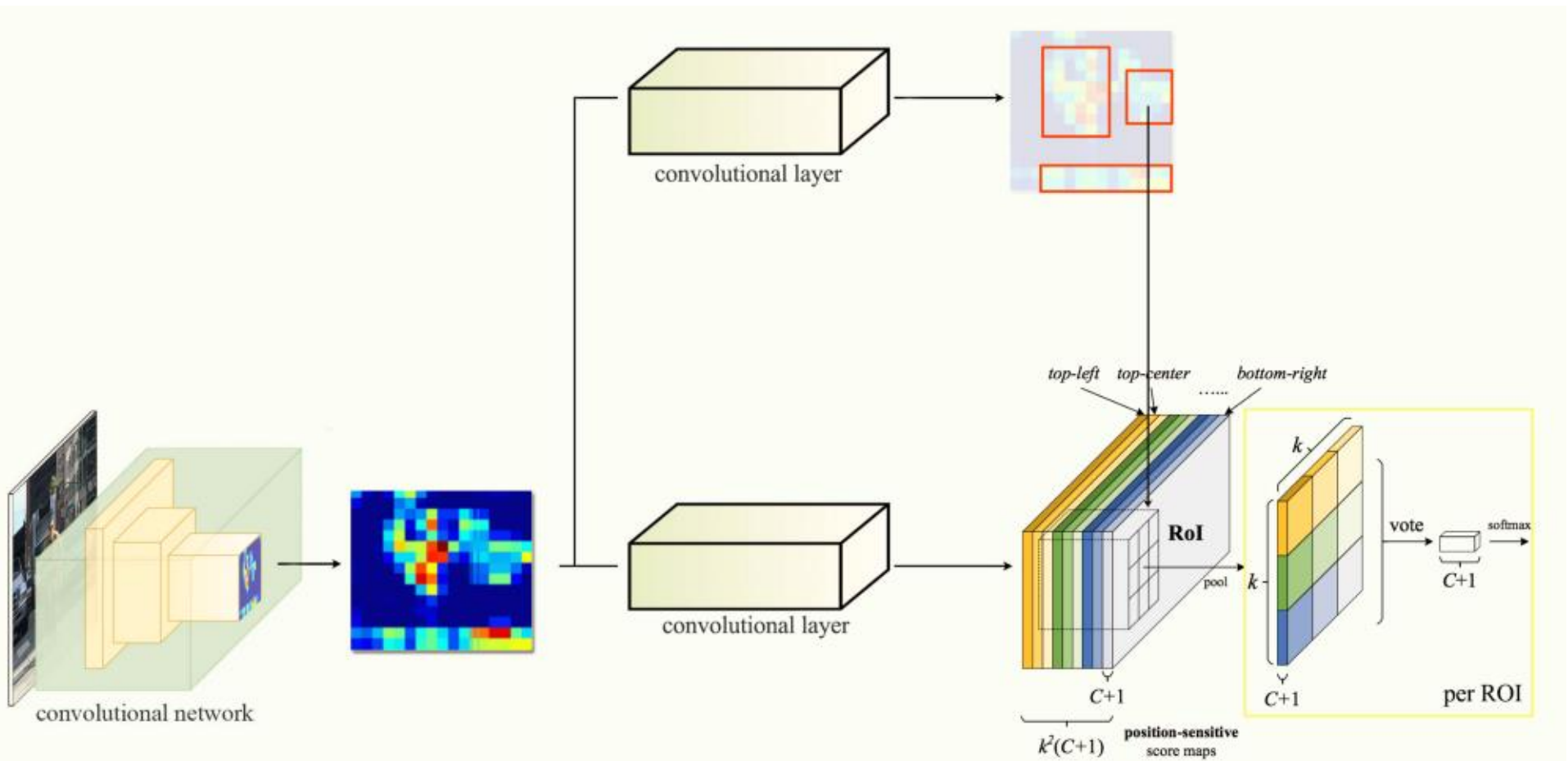


R-FCN (*)

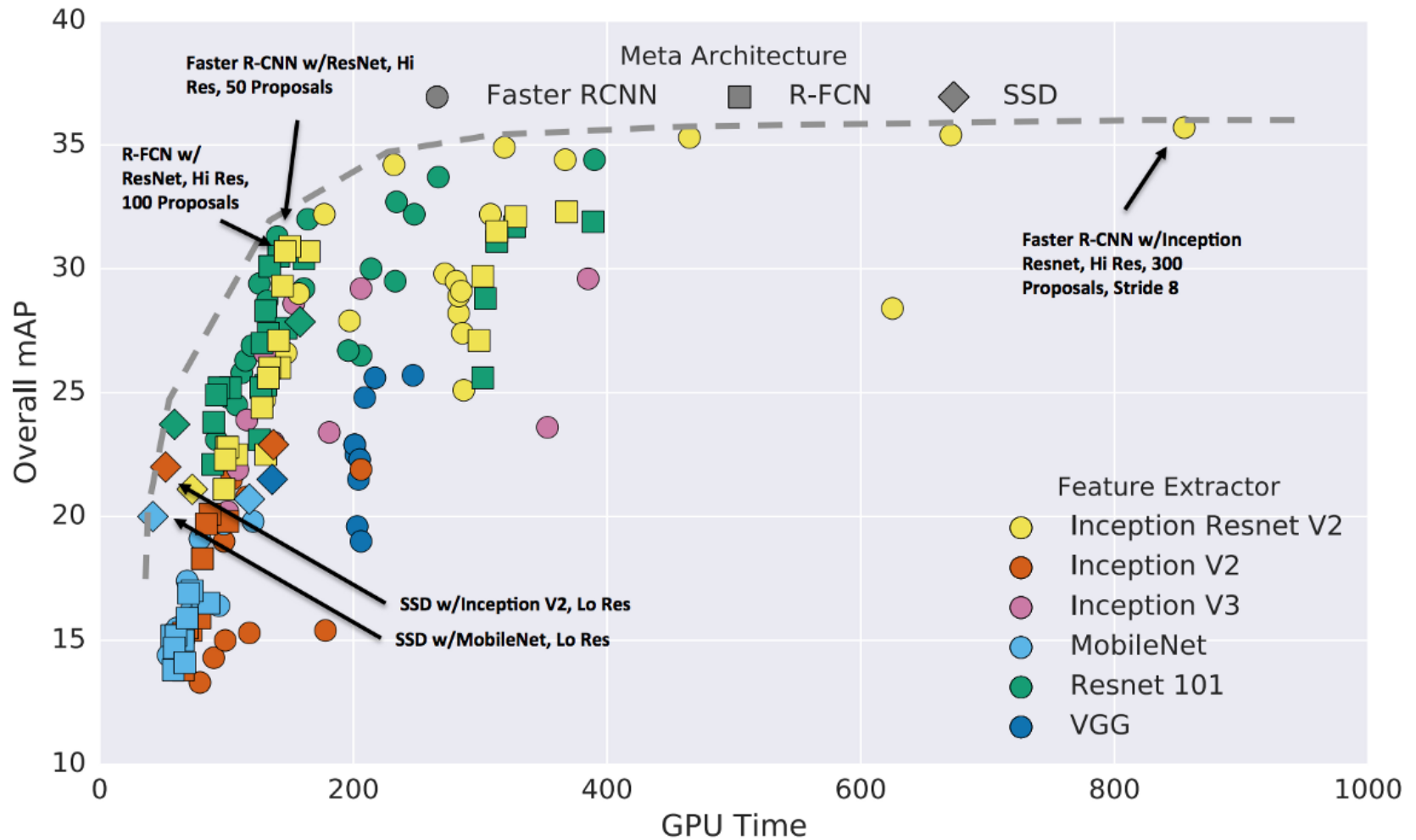
- Pozíció érzékeny ROI pooling:
 - ROI-kat egy Faster R-CNN alapú ROI jelölő háló állítja elő
 - $sc^{(r)}(c) = \sum_k \sum_{(x,y) \in R^{(r)}(k)} scm^{(c,k)}(x,y)$
 - Score-ok felett régióként softmax...
 - Motiváció: így meg tudja tanulni a háló, hogy egy összetett objektum mely részén milyen jellemzőket 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 (*)

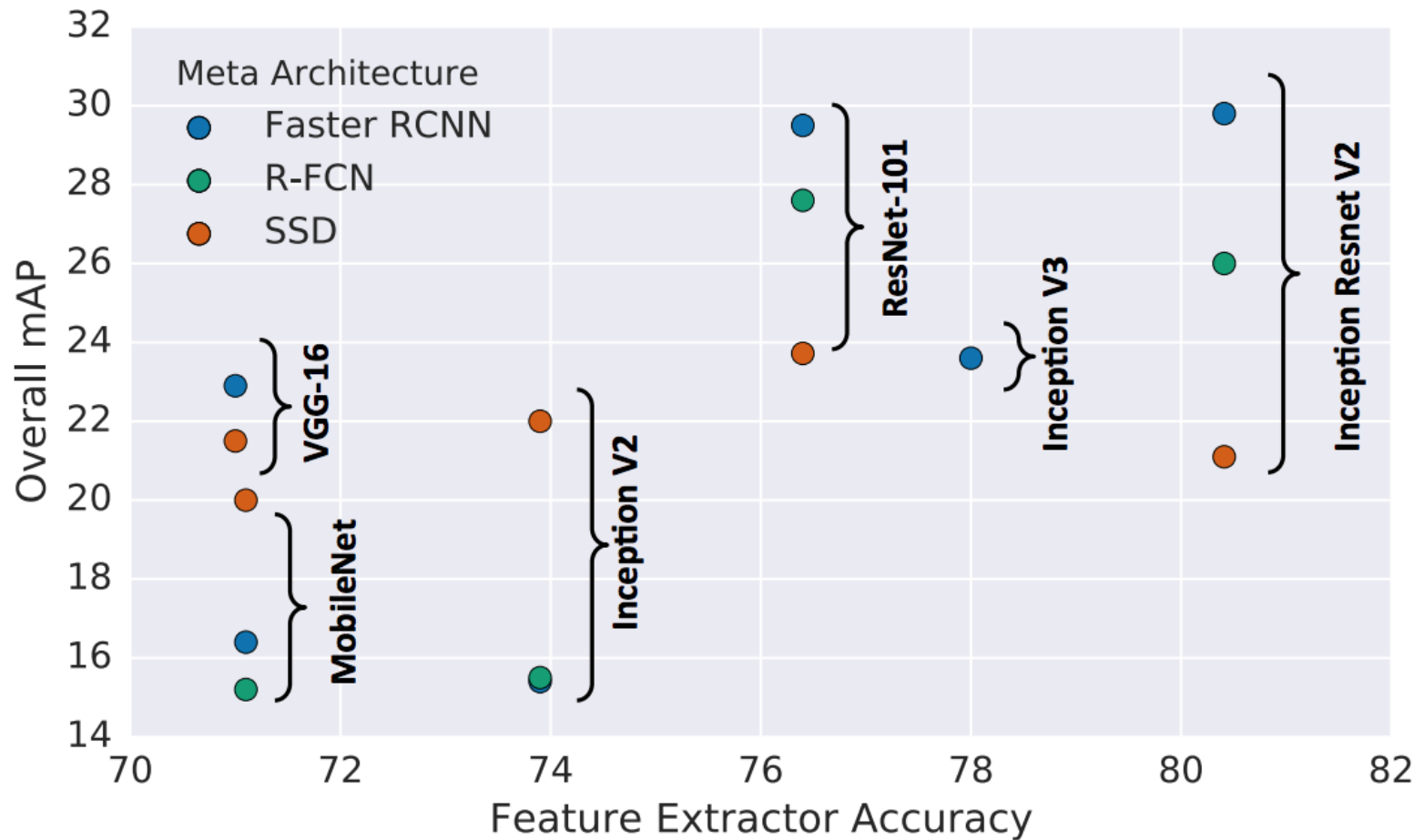
- Architektúra:



Meta architektúrák összehasonlítása



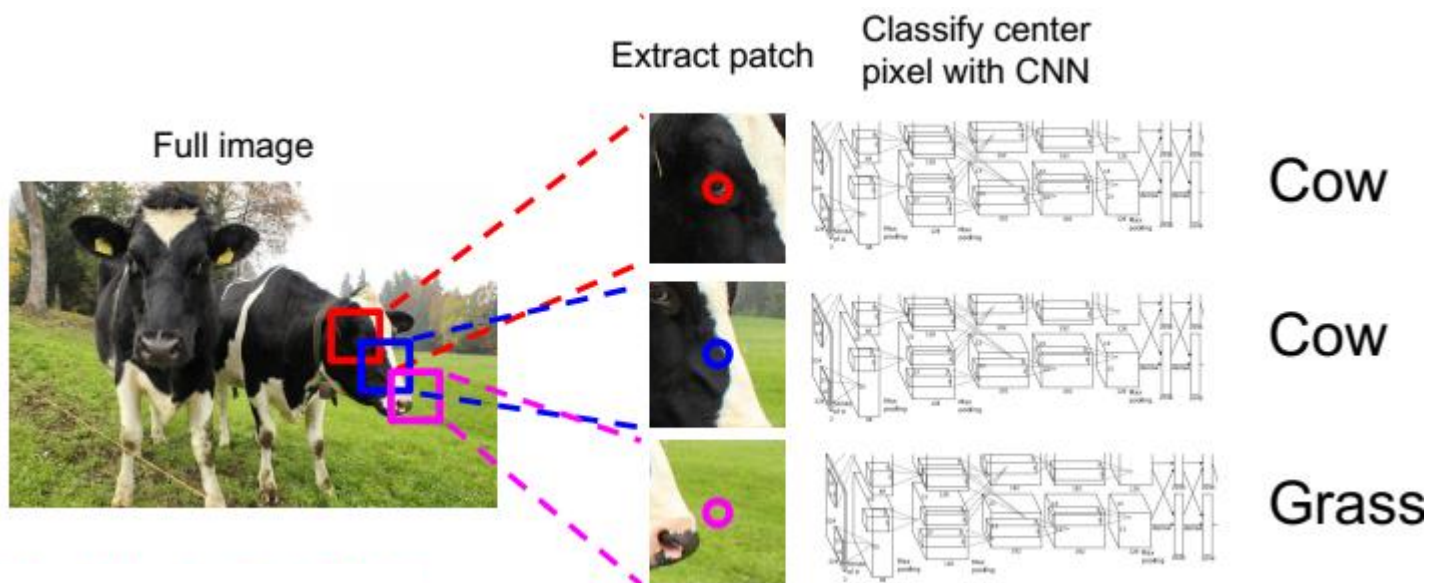
Meta architektúrák összehasonlítása



PIXEL SZINTŰ SZEGMENTÁLÁS CNN-EL

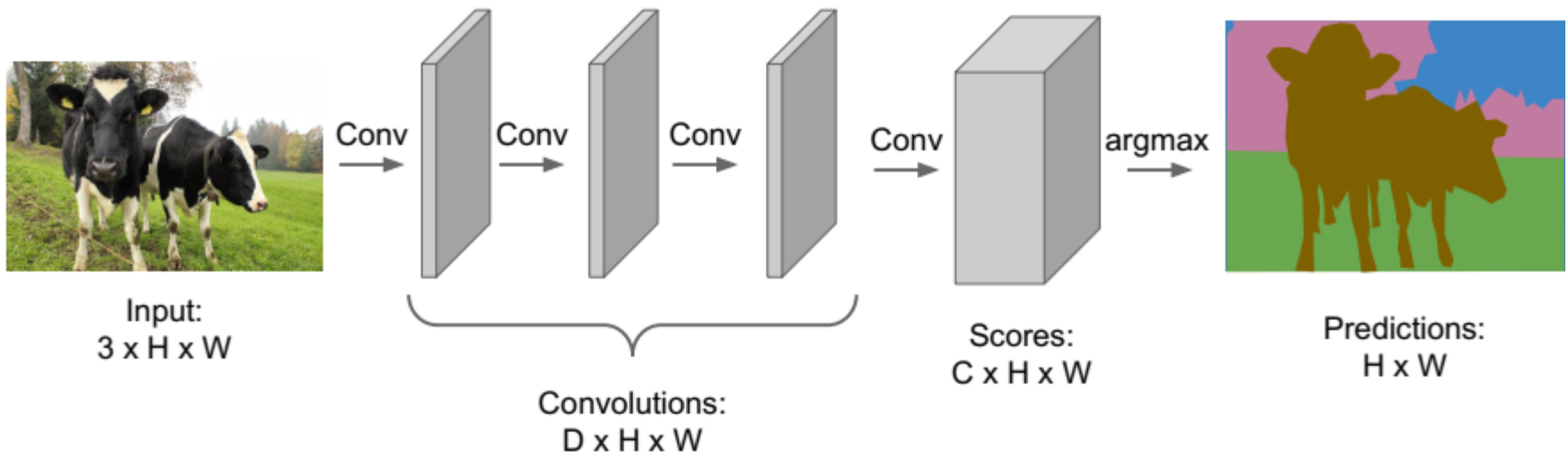
Csúszóablakos szegmentálás

- Szegmentálás direkt osztályozással
 - Kisméretű ablakkal kivágott kép alapján megítéli az adott pixel környezetének a típusát
 - Nagyon lassú, nehezen tanítható, pontatlan



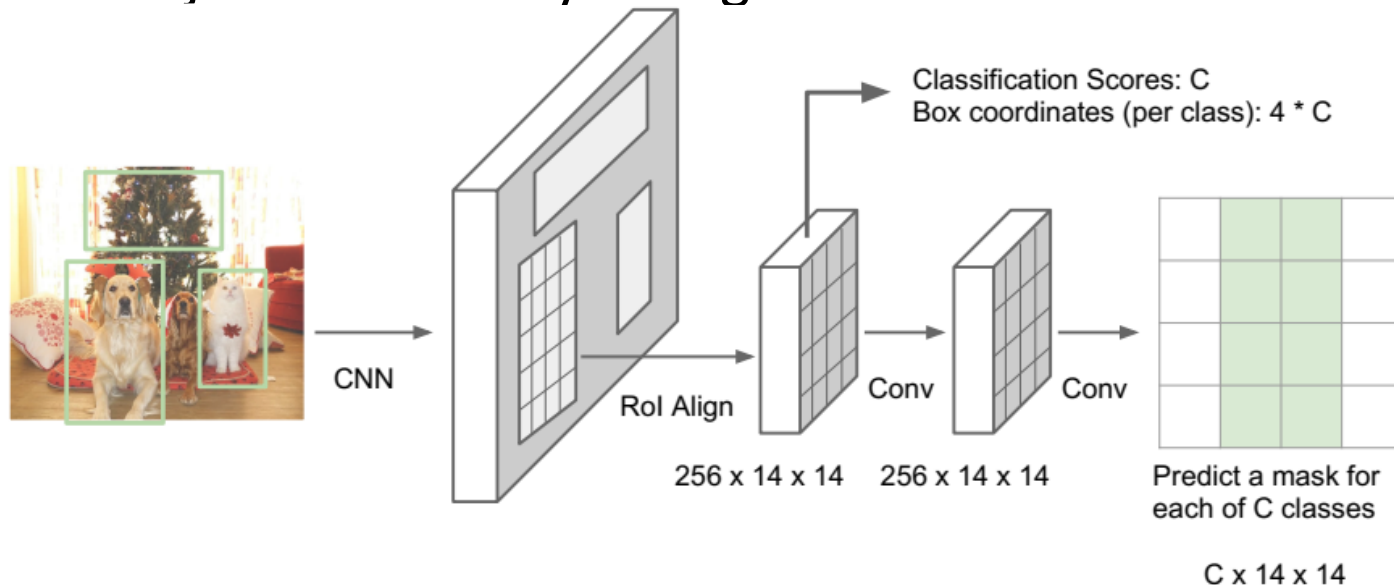
Teljesen konvolúciós hálóval

- Teljesen összekötött réteg nélküli hálóval:
 - Kimenete: minden képponthoz kategória számnyi konfiden-
ciát rendel (softmax nemlin. után) / regresszió pixelenként
 - Általában az U-net-el valósítják meg:
 1. szakasz: felbontás csökkentése, csatornák számának növelésével
 2. szakasz: felbontás növelése transzponált konvolúciókkal, csatornák számának csökkentésével



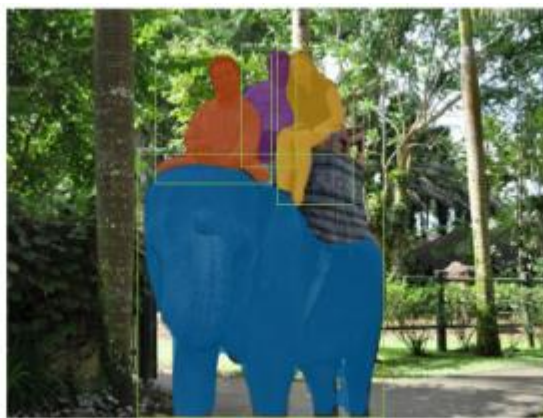
Mask R-CNN (*)

- Faster R-CNN kimeneti régióin teljesen konvolúciós hálóval szegmentál:
 - a) Régiókat uniform méretűvé mintavételezve, majd az eredményt vissza méretezve
 - b) Régiókat invalid pixelekkal uniform méretűvé kiegészítve, majd az eredményt kivágva



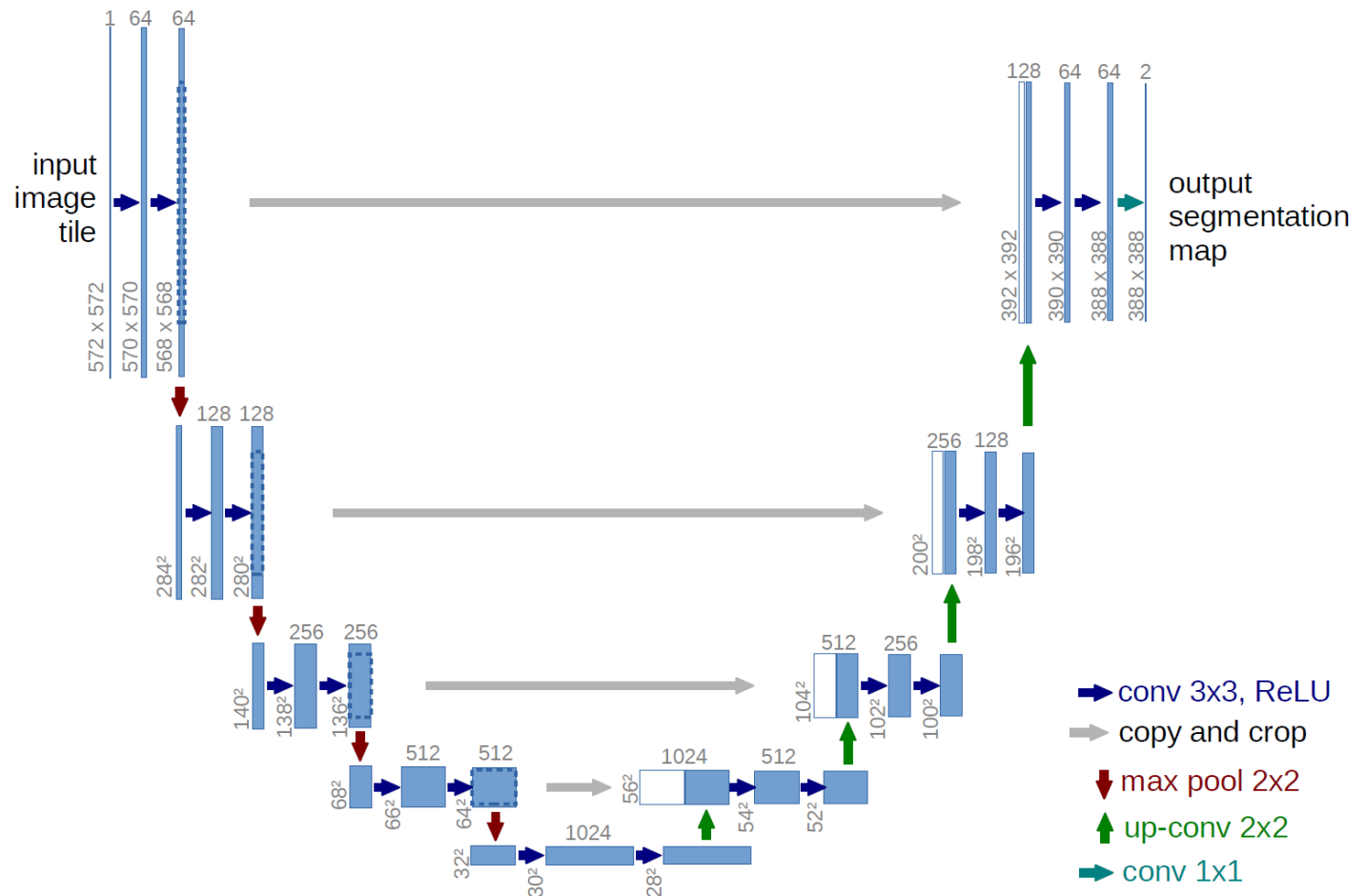
Mask R-CNN (*)

- Jó eredmények – pontos szegmentáció
- Nagy hatékonyság



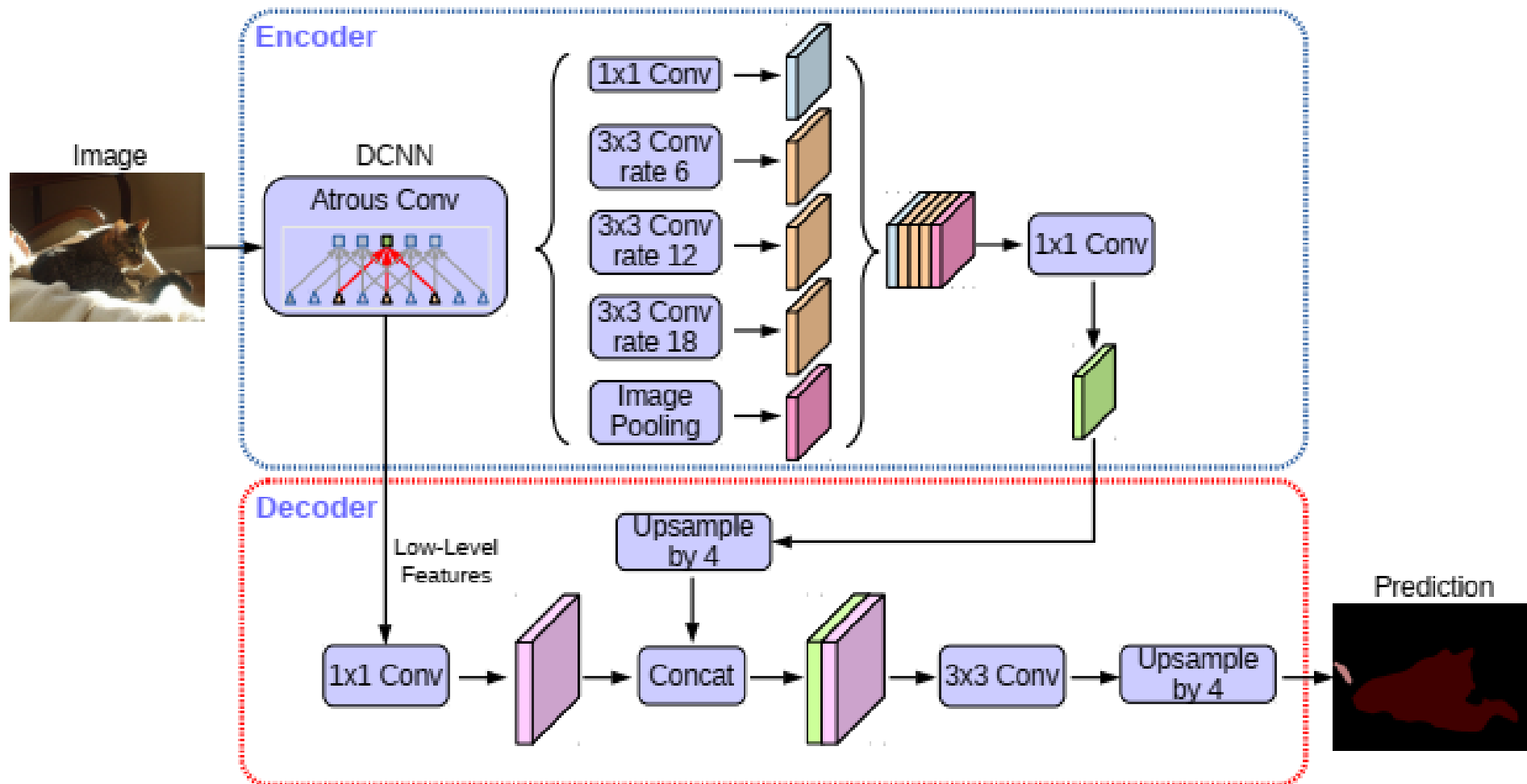
U-Net (2016)

- Teljesen konvolúciós (Fully Convolutional) háló



DeepLab –v3+ (2018) (*)

- Enkóder – Dekóder architektúrák továbbfejlesztése

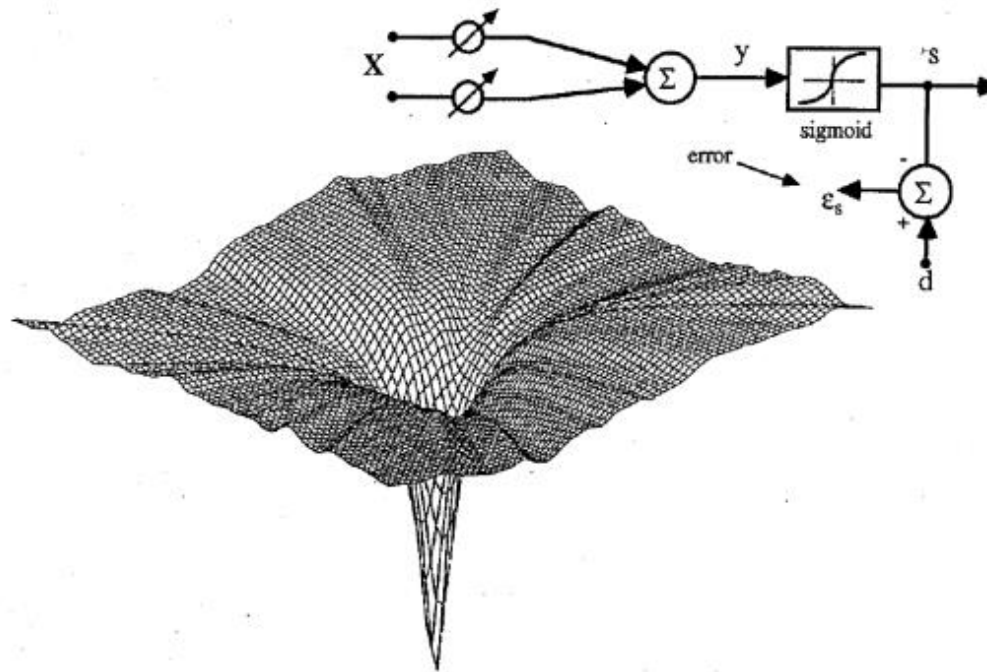


NEURÁLIS HÁLÓK TANÍTÁSA SORÁN ALKALMAZOTT OPTIMALIZÁCIÓ

Hibafelületek

- Hálók által megvalósított leképezés:
 - Bemenetet exponenciálisan sok partícióra bontja
 - Egy-egy ilyen területen lineárishoz közeli leképezés
- Hibafelületek általános jellemzője:
 - Sok szabad paraméter (1E9-es nagyságrend)
 - Ezek közül sok nem befolyásolja érdemben a kimenetet (pl. neuron ReLU-ja 0-ra vág)
 - Erősen nem konvex:
 - Hosszan elnyúló platók, hirtelen letörések
 - Tele van nyeregponttal, lokális optimumokkal

Hibafelületek - példák

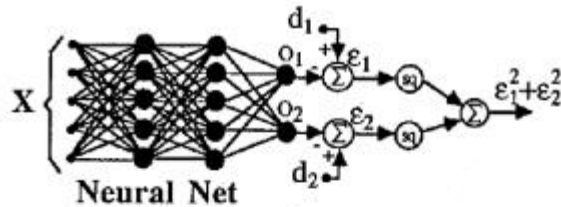


EXPERIMENT 2b

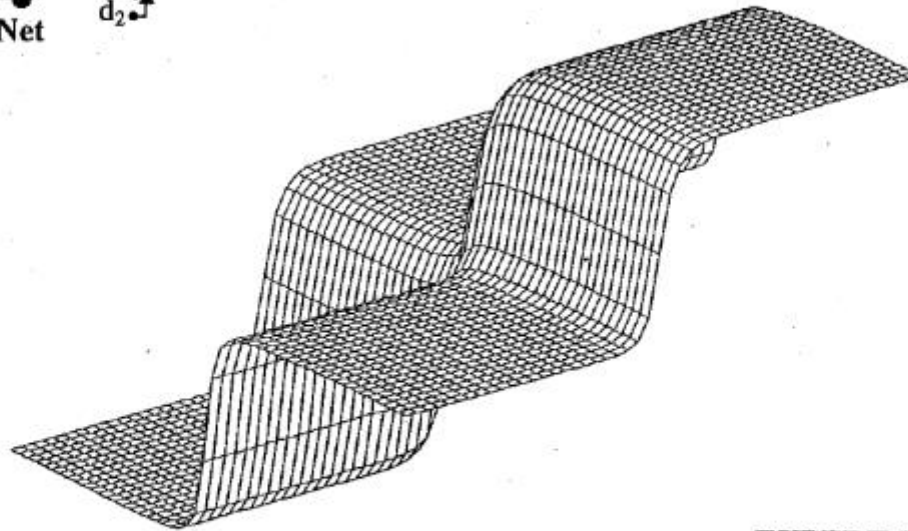
Same input as EXPT. 1b,
but different desired response.

Mean Squared Error Surface (sigmoid)

Hibafelületek - példák



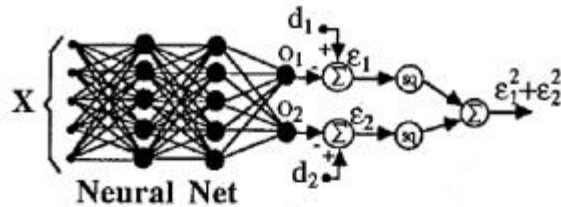
- 4 inputs
- 3-layer network: 5 feed 8 feed 2
- sigmoids



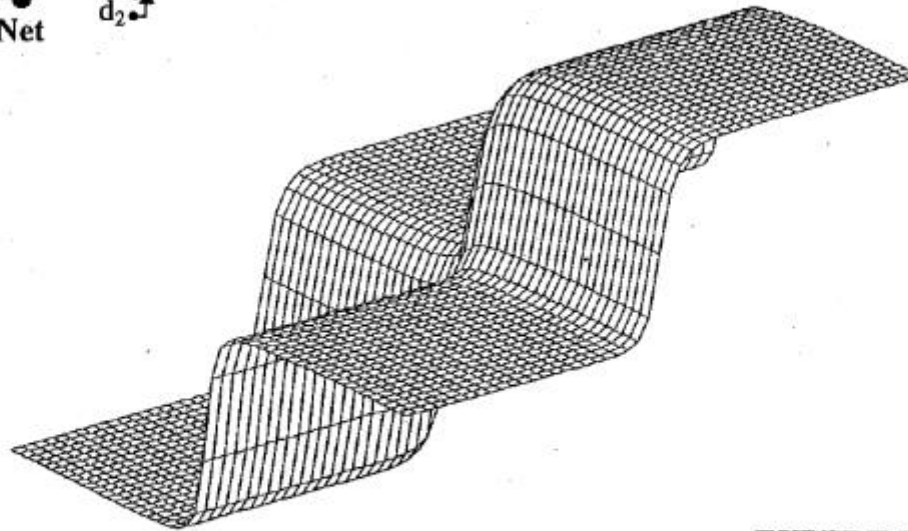
EXPERIMENT 3a.
Randomize weights,
then vary two
first-layer weights.

Mean Square Error Surface

Hibafelületek - példák



- 4 inputs
- 3-layer network: 5 feed 8 feed 2
- sigmoids

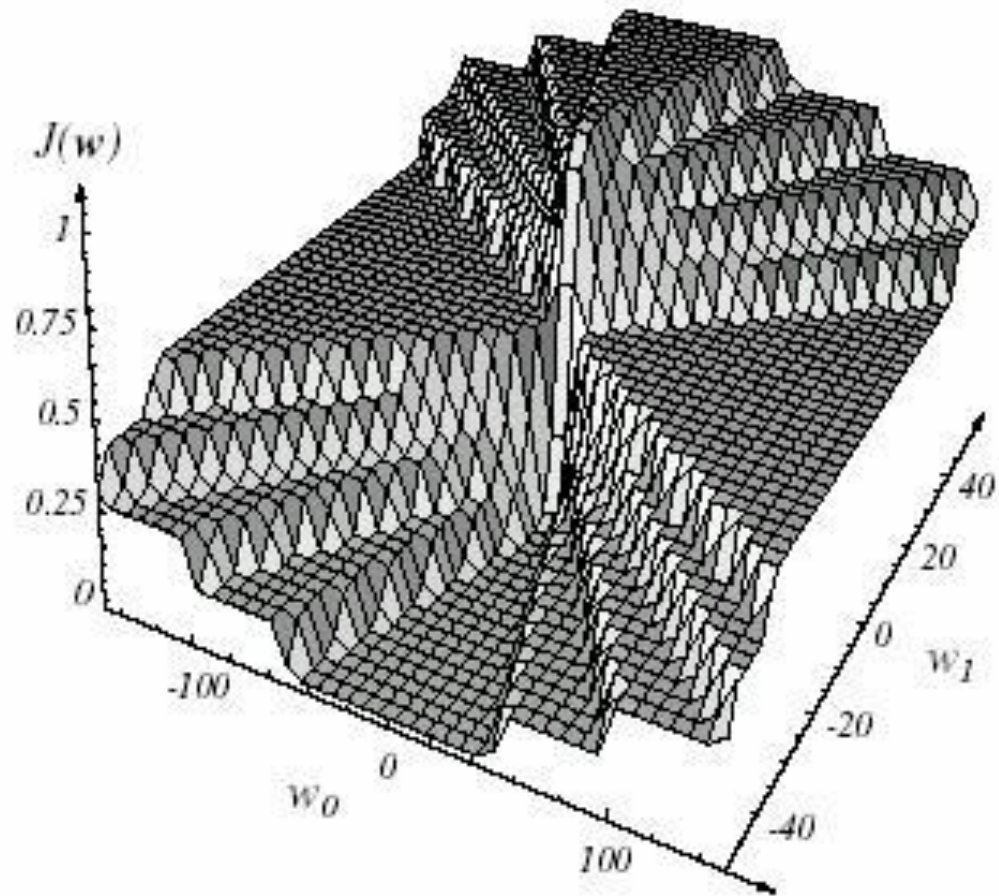


EXPERIMENT 3a.
Randomize weights,
then vary two
first-layer weights.

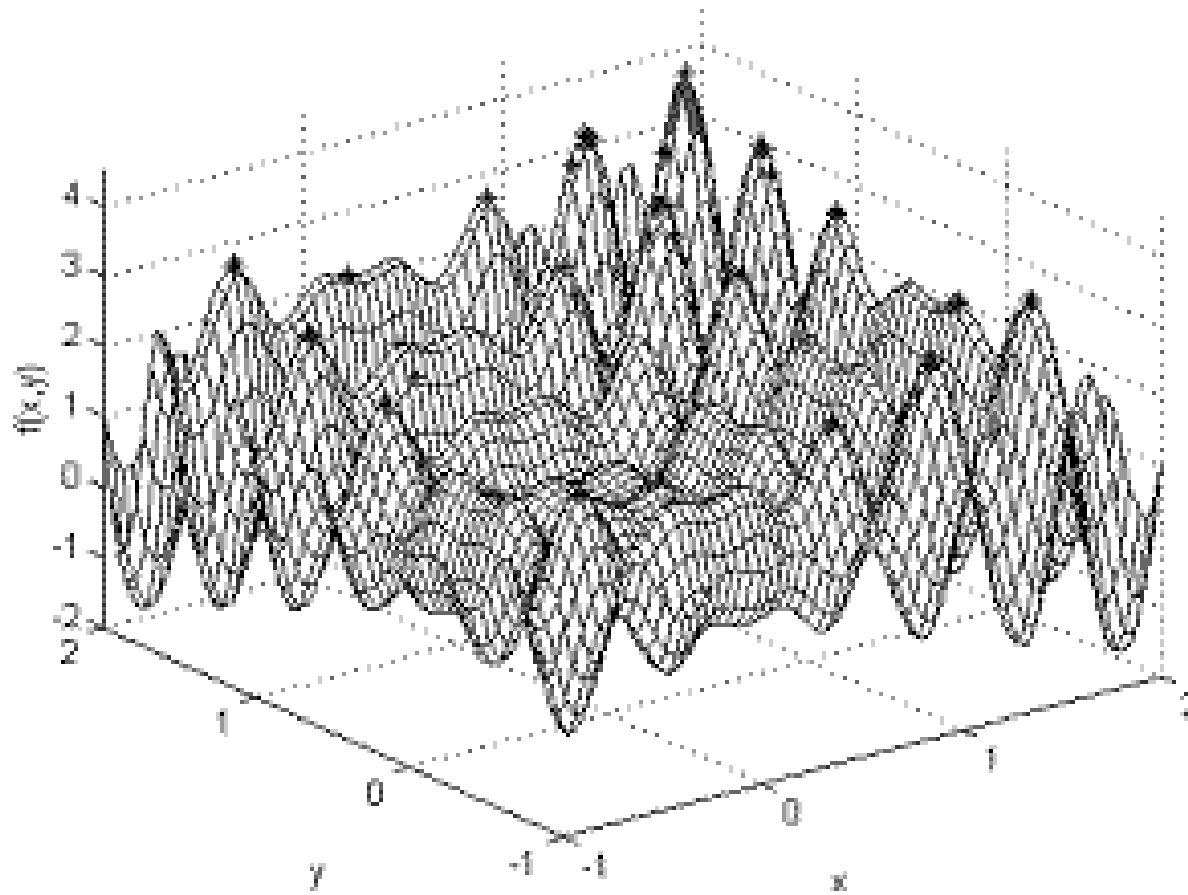
Mean Square Error Surface

Hibafelületek - példák

2 rejtett rétegű MLP



Hibafelületek - példák



Neurális hálók tanítása

- Erősen nem konvex hibafelületek:
 - Ezzel praktikusán nem foglalkozunk
- Numerikus optimalizáció módszerei:
 - Másodrendű eljárások:
 - hibafelület másodrendű közelítését minimalizálják
 - Newton iteráció, Levenberg Marquadt, BFGS, L-BFGS
 - Elsőrendű eljárások:
 - első fokú Taylor polinomot minimalizálják
 - SGD, Adaptív módszerek, Polyak GD, Nesterov GD

Newton módszer

- $C_{\theta_0}^{(2)} \{ \theta + \theta_0 \} = C \{ \theta_0 \} + \theta^T \cdot \nabla C \{ \theta_0 \} + (1/2) \cdot \theta^T \cdot \mathbf{H} \{ \theta_0 \} \cdot \theta$

- $\mathbf{H} \{ \theta_0 \}_{(i,j)} = \frac{\partial^2 C \{ \theta_0 \}}{\partial \theta_{(i)} \partial \theta_{(j)}}$ jelöli C Hesse-mátrixát

- $\nabla C \{ \theta_0 \}_{(i)} = \frac{\partial C \{ \theta_0 \}}{\partial \theta_{(i)}}$ jelöli a gradienst θ_0 -ban

- Ha $\mathbf{H} \{ \theta_0 \}$ pozitív definit, akkor

- $\theta^* = \arg \max_{\theta} \{ C_{\theta_0}^{(2)} \{ \theta \} \} \Leftrightarrow \nabla C_{\theta_0}^{(2)} \{ \theta^* \} = \mathbf{0}$

- Tehát $\theta^* = -\mathbf{H} \{ \theta_0 \}^{-1} \cdot \nabla C \{ \theta_0 \}$

Newton módszer

- $C_{\theta_0}^{(2)} \{\theta\}$ csak approximálja C -t:
 - Viszont C θ_0 pontbeli kvadratikus közelítésének t.f.h. a minimumhelye: $\theta_0 + \theta^*$
 - Tehát $\Delta\theta = \alpha \cdot \theta^*$
 - α meghatározása line-search algoritmusokkal. Az alábbi trade-off problémát oldják meg:
 - $\left| C_{\theta_0}^{(2)} \{\theta_0 + \alpha \cdot \Delta\theta\} - C \{\theta_0 + \alpha \cdot \Delta\theta\} \right| < \varepsilon$
 - $C_{\theta_0}^{(2)} \{\theta_0\} - C_{\theta_0}^{(2)} \{\theta_0 + \alpha \cdot \Delta\theta\} > \delta$

Newton módszer

- Interpretációja:
 1. Aktuális paramétervektor környezetében egy kvadratikus felülettel közelítjük $C\{\theta\}$ -t.
 2. Megkeressük a közelítés minimumhelyét, majd annak irányába lépünk „elegendően” nagyot.
 3. Ha nem teljesül a leállási feltétel, akkor GOTO 1.
- Előnyei:
 - Pár 10 / 100 iteráció gyakorlatban elég
 - $C\{\theta\}$ affin transzformációjára érzéketlen
 - Nyeregpontnál viszont gond van - instabil

Newton módszer

- Hátrányai:
 - Minden iterációban újra kell számolni, és invertálni a Hesse mtx.-ot (mely akár $10^8 \times 10^8$ méretű):
 - Kvázi Newton módszerek ezt próbálják kikerülni
 - Batch-elt tanításhoz illeszkedik jól – túl nagy lépéseket tesz ahhoz, hogy sztochasztikus (pl. mini batch) optimalizációra alkalmas legyen:
 - Így nem is lehet jól párhuzamosítani
- Klasszikus Neurális hálóknál alkalmazható inkább (ld. Levenberg-Marquadt)

Kvázi Newton módszerek – BFGS

- A másodfokú közelítésben
- BFGS algoritmus ú.n. secant feltétele:
 - $-\nabla C_{\boldsymbol{\theta}_k}^{(2)} \{ \boldsymbol{\theta}_k \} = \nabla C \{ \boldsymbol{\theta}_k \}$ és $-\nabla C_{\boldsymbol{\theta}_k}^{(2)} \{ \boldsymbol{\theta}_{k-1} \} = \nabla C \{ \boldsymbol{\theta}_{k-1} \}$
 - Első feltétel $C_{\boldsymbol{\theta}_k}^{(2)} \{ \cdot \}$ definíciója miatt teljesül
 - Második feltétel kifejtését követően:
$$\nabla C \{ \boldsymbol{\theta}_{k-1} \} = \nabla C \{ \boldsymbol{\theta}_k \} + \mathbf{H} \{ \boldsymbol{\theta}_k \} \cdot (\boldsymbol{\theta}_{k-1} - \boldsymbol{\theta}_k)$$
 - Tehát a secant feltétel igaz, ha
$$\mathbf{H} \{ \boldsymbol{\theta}_k \}^{-1} \cdot (\nabla C \{ \boldsymbol{\theta}_{k-1} \} - \nabla C \{ \boldsymbol{\theta}_k \}) = (\boldsymbol{\theta}_{k-1} - \boldsymbol{\theta}_k)$$

Kvázi Newton módszerek – BFGS

- Mit tudunk eddig $\mathbf{H}\{\boldsymbol{\theta}_k\}^{-1}$ -ről:
 - PSD, hiszen egy Hesse mtx.-ről lenne szó
 - $\mathbf{H}\{\boldsymbol{\theta}_k\}^{-1} \cdot (\nabla C\{\boldsymbol{\theta}_{k-1}\} - \nabla C\{\boldsymbol{\theta}_k\}) = (\boldsymbol{\theta}_{k-1} - \boldsymbol{\theta}_k)$
 - De ilyenből végtelen van – „regularizáció” kell:
 - Élünk a simasági feltételezéssel ($\mathbf{H}\{\cdot\}$ folytonos)
 - $\mathbf{H}\{\boldsymbol{\theta}_k\}^{-1} = \arg \min_{\mathbf{H}^{-1}} \left\{ \left\| \mathbf{H}^{-1} - \mathbf{H}\{\boldsymbol{\theta}_{k-1}\}^{-1} \right\|_F^2 \mid \dots \right\}$
- Szerencsére analitikusan megoldható

Kvázi Newton módszerek – BFGS

- BFGS frissítés tulajdonságai:
 - Ha $\mathbf{H}\{\boldsymbol{\theta}_0\}$ -t diagonálisnak választjuk, akkor $\Delta\boldsymbol{\theta}_k$ számítása $O(k \cdot \text{size}(\boldsymbol{\theta}))$ komplexitású.
 - Ehhez viszont szükség van $\nabla C\{\boldsymbol{\theta}_k\}$ -re és $\boldsymbol{\theta}_k$ -ra, minden $i = 0, \dots, k$ esetén
 - Nagyméretű problémák esetén ez már problémát jelent (memória).
 - Memória használata túl nagy:
 - Pl. 1E5 iteráció, és 1E7 paraméter esetén ~ 1 TB

Kvázi Newton módszerek – L-BFGS

- Belátható, hogy konvergál az optimumhoz (konvex)
 - Feltétele: $\mathbf{H}\{\boldsymbol{\theta}_0\}$ approximációja pd.
 - BFGS: superlineáris, Newton: négyzetes
 - Skálázásra érzékeny (sok numerikus művelet)
- L(imitált memóriahasználatú)-BFGS:
 - Bár a közelített inverz Hesse kiszámításához szükség lenne az összes addigi gradiensre és paraméterre, ezt csak az utolsó m db alapján becsli.
 - Matematikai bizonyítása a konvergenciának nincs, de praktikusan konvergens (itt már jelentősen függ a szükséges iterációk száma a változók számától).
 - Mini Batch-es tanítást ez sem igazán kedveli

Kvázi Newton módszerek – Konjugált Gradiens módszer

- Lineáris egyenletek megoldására találták ki, most $\boldsymbol{\theta}^* = -\mathbf{H}^{-1} \cdot \nabla C \{\boldsymbol{\theta}_0\}$ -t keressük iteratívan.

- Alapötlet – lineáris egyenlet megoldása:

$$-\alpha^* = \arg \min_{\alpha} \left\{ \left\| \mathbf{H} \cdot (\boldsymbol{\theta}_{k-1} + \alpha \cdot \Delta \boldsymbol{\theta}_{k-1}) - \nabla C \{\boldsymbol{\theta}_0\} \right\|_2 \right\}$$

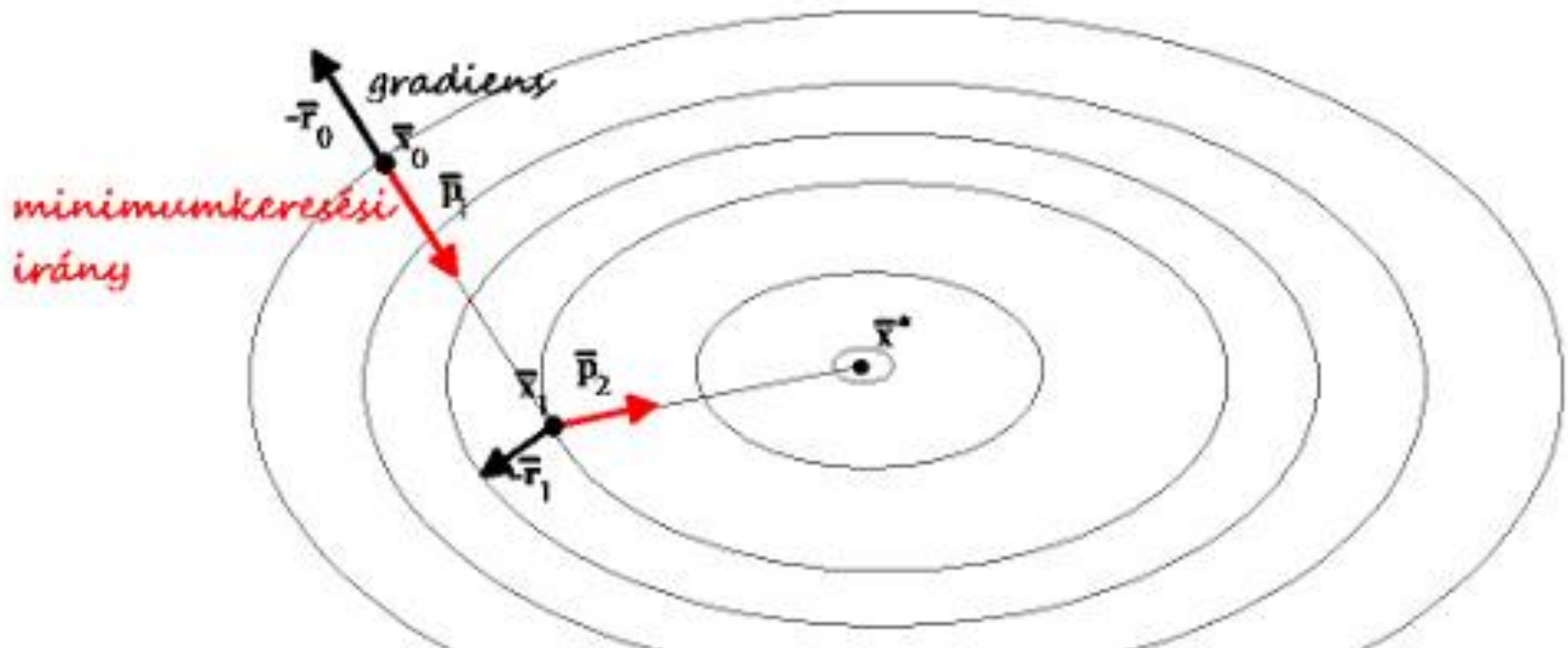
$$\Delta \boldsymbol{\theta}_{k-1}^T \cdot \left(\nabla C \{\boldsymbol{\theta}_0\} - \mathbf{H} \cdot (\boldsymbol{\theta}_{k-1} + \alpha^* \cdot \Delta \boldsymbol{\theta}_{k-1}) \right) =$$

$$= \Delta \boldsymbol{\theta}_{k-1}^T \cdot \mathbf{H} \cdot (\boldsymbol{\theta}^* - \boldsymbol{\theta}_k) = 0$$

Tehát az k -adik javítóirány $\mathbf{H}$ feletti konjugáltja a $(k-1)$ -ediknek, azaz $\Delta \boldsymbol{\theta}_{k-1}^T \cdot \mathbf{H} \cdot \Delta \boldsymbol{\theta}_k = 0$

Kvázi Newton módszerek – Konjugált Gradiens módszer

- Mint lineáris egyenlet megoldó: $\mathbf{x}^* = \mathbf{A}^{-1} \cdot \mathbf{b}$



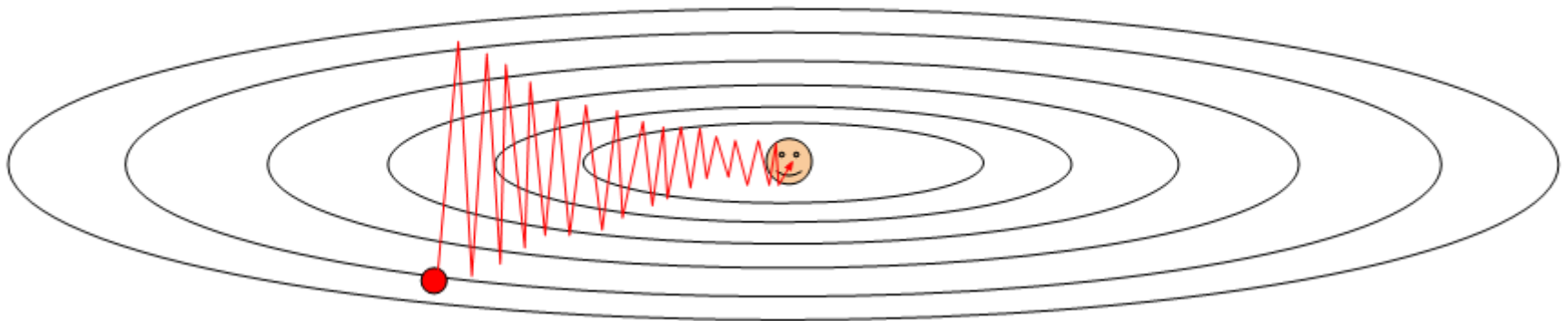
$\mathbf{p}_i$: i -edik keresési irány, $\mathbf{x}_i$: i -edik becslése $\mathbf{x}^*$ -nak

Kvázi Newton módszerek – Konjugált Gradiens módszer

- Mint veszteségfüggvény minimalizáló eljárás:
 - Aktuális pontbeli gradiensből az előző javítóiránnyal párhuzamos komponenszt levonva áll elő az új javítóirány: $\Delta\boldsymbol{\theta}_k = -\nabla C(\boldsymbol{\theta}_{k-1}) - \beta \cdot \Delta\boldsymbol{\theta}_{k-1}$
 - Javító lépés nagysága: $\arg \min_{\alpha} \{C(\boldsymbol{\theta}_{k-1} + \alpha \cdot \Delta\boldsymbol{\theta}_k)\}$
 - Nem igényli a Hesse mtx. kiszámítását
- Konvergencia:
 - Invertálandó mátrix sajátértékeinek sűrűsödési pontjainak a számától függ.

Mély hálók során alkalmazott optimalizáció

- Elsőrendű módszerek:
 - SGD (mini batch-es tanítás miatt)
 - Másodrendű módszerekkel sem nyernénk sokat (sok réteg miatt ennél magasabb fokú a hibafelület)
 - Rosszul kondicionált hibafelületeken viszont a Gradient Descent hajlamos a beragadásra, lassú mozgásra (vagy gyors divergálásra)



Adaptív módszerek

- Adaptív módszerek : közvetlenül megelőző iterációk lépéseiből becslik a hibafelület jellegét:
 - Ha egy adott súly keveset változik, akkor annak mentén elnyújtott a hibafelület => lehet bátrabban is lépni
 - Ha egy súly mentén oszcilláló mozgás van, akkor túl nagy a bátorsági tényező
- Momentum módszer:
 - Szemléletesen: GD által becsült javítóutak mozgó átlagával regularizálja azt
 - Nesterov momentum: bizonyítottan, asszimptotikusan optimális első rendű optimalizáció (négyzetes rátával)

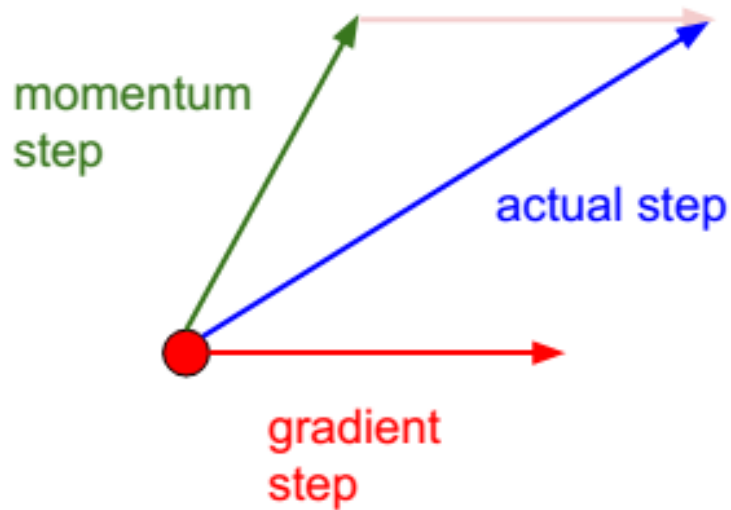
Momentum módszerek

- Első interpretáció – alul-áteresszük θ_k -at:
 - Lényegében mozgó átlaggal
 - $\Delta \theta_k' = \beta \cdot \Delta \theta_{k-1}' + (1 - \beta) \cdot \nabla C \{ \theta_{k-1} \} \quad \beta \in (0, 1)$
 - $\theta_k = \theta_{k-1} - \alpha \cdot \Delta \theta_k'$
- Másik interpretáció – fizikai modell:
 - $\mathbf{v}_k = \mu \cdot \mathbf{v}_{k-1} - \alpha \cdot \nabla C \{ \theta_{k-1} \} \quad \mu \in (0.5, 1)$
Integráljuk a gyorsulást ($-\alpha \cdot \nabla C \{ \theta_{k-1} \}$), μ súrlódással
 - $\theta_k = \theta_{k-1} + \mathbf{v}_k$
Integráljuk a sebességet

Nesterov Momentum

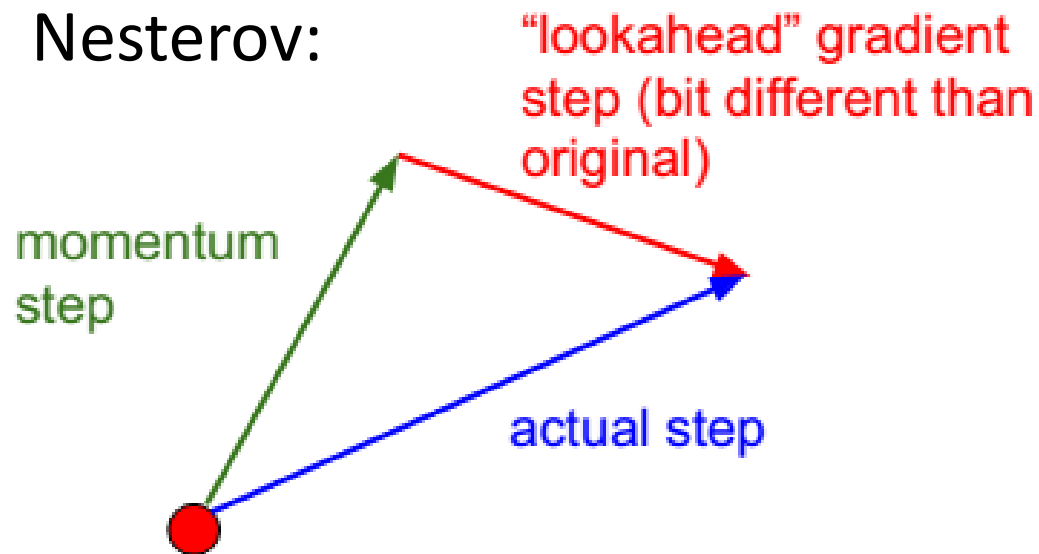
- Ötlet: Ha úgyis ellépünk $\mu \cdot \mathbf{v}_{k-1}$ irányba, akkor az ottani gradiens irányába lépünk tovább (extrapoláció):

Egyszerű momentum:



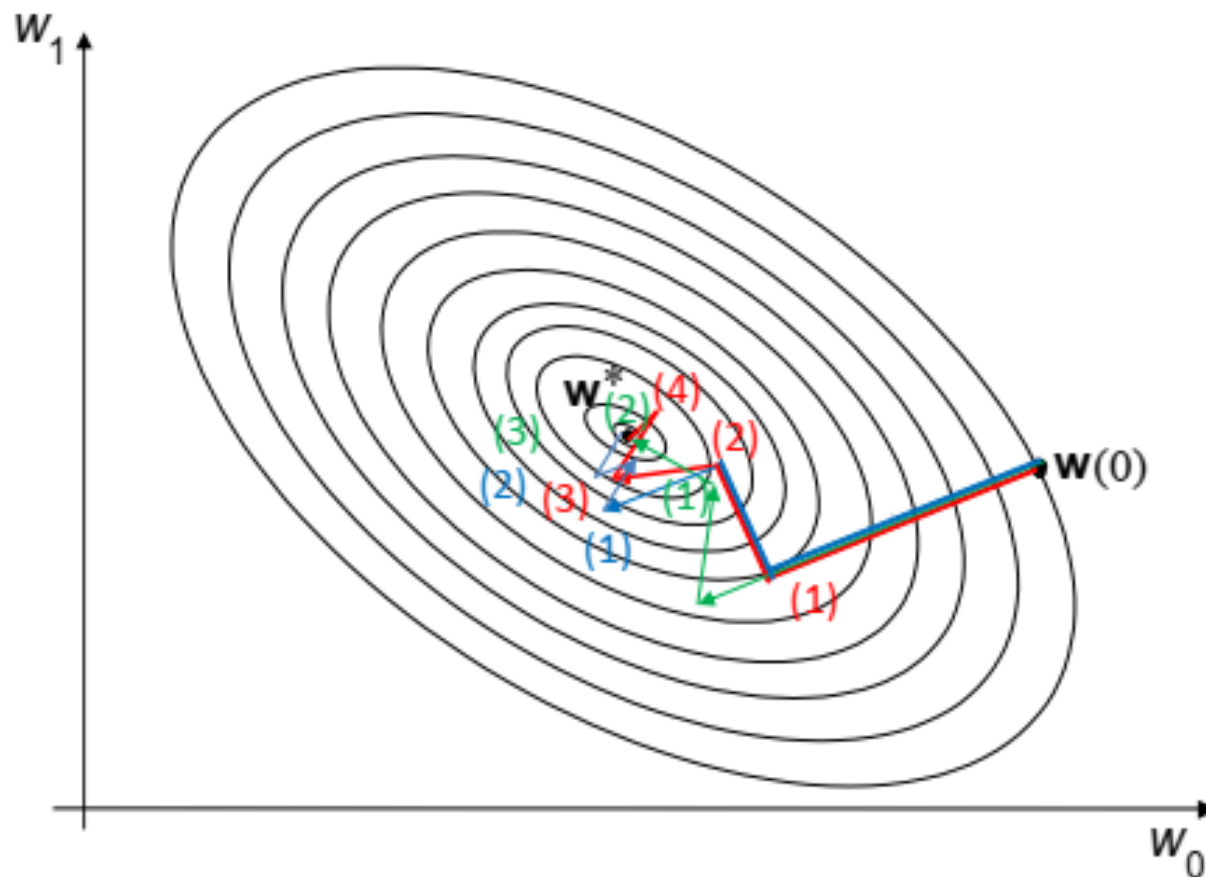
$$\mathbf{v}_k = \mu \cdot \mathbf{v}_{k-1} - \alpha \cdot \nabla C \{ \boldsymbol{\theta}_{k-1} \}$$

Nesterov:



$$\mathbf{v}_k = \mu \cdot \mathbf{v}_{k-1} - \alpha \cdot \nabla C \{ \boldsymbol{\theta}_{k-1} + \mu \cdot \mathbf{v}_{k-1} \}$$

Momentum módszerek



Gradiens (-1)-szerese irányában

Hagyományos momentum

Nyeszterov momentum

Adaptív gradiens (AdaGrad)

- Dimenzióként normalizálja a javító lépéseket végtelen memóriájú aggregációval:

$$(\boldsymbol{\tau}_k)_{(i)} = \sum_{j=1}^{k-1} \nabla C \{ \boldsymbol{\theta}_j \}_{(i)}^2$$

$$\Delta \boldsymbol{\theta}_{k(i)} = -\mu \cdot \nabla C \{ \boldsymbol{\theta}_{k-1} \}_{(i)} / \left(\sqrt{(\boldsymbol{\tau}_k)_{(i)}} + \varepsilon \right)$$

- Probléma: hajlamos a leállásra a nem felejtő memória miatt ($(\boldsymbol{\tau}_k)_{(i)}$ k függvényében monotonan nő)
- Összetett, sok részdoménre bontható, azokban jelentősen eltérő hibafelület miatt sem jó a túl hosszú emlékezet

RMSProp

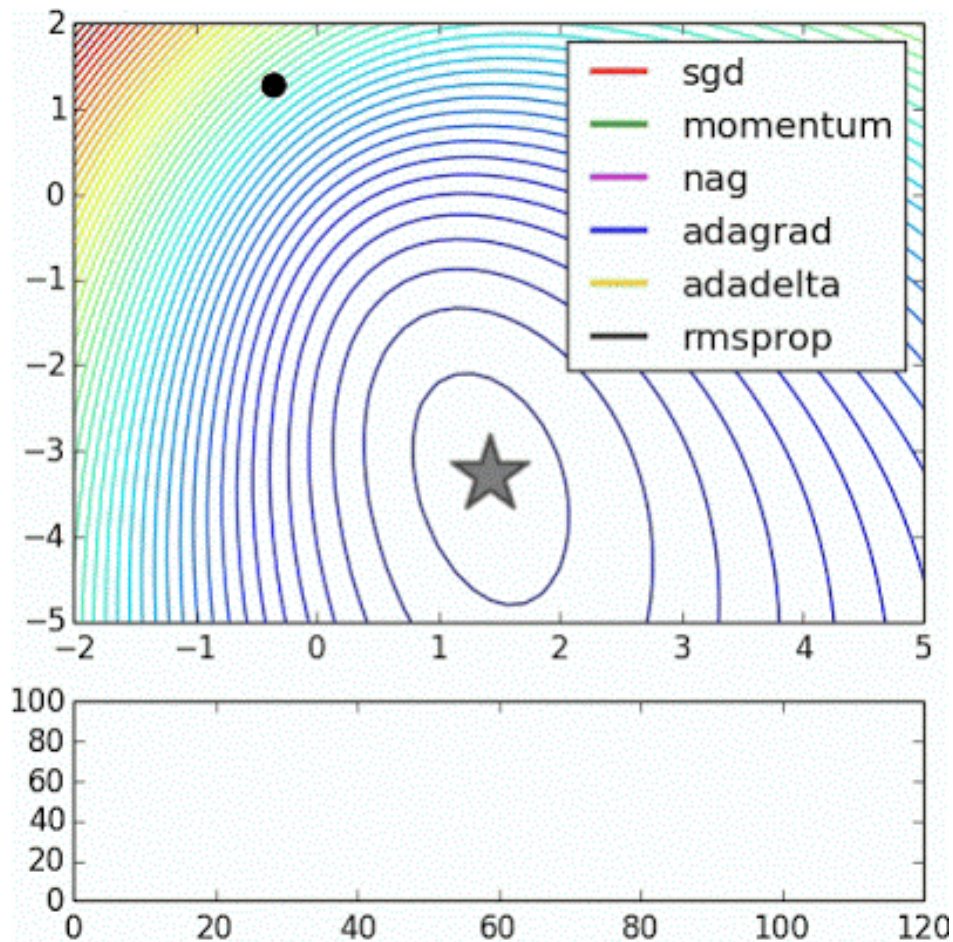
- Ötlet: végtelen összegzés helyett mozgó átlag alapján normáljunk
 - MS tag: $(\boldsymbol{\tau}_k)_{(i)} = \beta \cdot (\boldsymbol{\tau}_{k-1})_{(i)} + (1 - \beta) \cdot \nabla C \{ \boldsymbol{\theta}_k \}_{(i)}^2$
 - Gradiens tag: $(\Delta \boldsymbol{\theta}_k)_{(i)} = -\mu \cdot \nabla C \{ \boldsymbol{\theta}_{k-1} \}_{(i)} / \left(\sqrt{(\boldsymbol{\tau}_k)_{(i)}} + \varepsilon \right)$
 - Mozgó átlagolás miatt nem haragtartó (túl távoli dolgokra már nem emlékszik)
 - Nem nő minden iterációban az értéke

Adaptive Momentum (Adam)

- Kombináljuk az RMSProp-ot a Momentummal:
 1. $\mathbf{v}_k = \beta_1 \cdot \mathbf{v}_{k-1} + (1 - \beta_1) \cdot \nabla C \{ \boldsymbol{\theta}_k \} \quad \beta_1 \in (0, 1)$
 2. $(\boldsymbol{\tau}_k)_{(i)} = \beta_2 \cdot (\boldsymbol{\tau}_{k-1})_{(i)} + (1 - \beta_2) \cdot \nabla C \{ \boldsymbol{\theta}_k \}_{(i)}^2 \quad \beta_2 \in (0, 1)$
 3. $(\boldsymbol{\theta}_k)_{(i)} = (\boldsymbol{\theta}_{k-1})_{(i)} - \alpha \cdot (\mathbf{v}_k)_{(i)} / \left(\sqrt{(\boldsymbol{\tau}_k)_{(i)}} + \varepsilon \right) \quad \alpha \in \mathbb{R}_{++}$
- Ötvözi a két módszer előnyét:
 - Simítja a lépési irányt, mint a Momentum
 - Adaptálódik a hibafelülethez, mint az RMSprop
 - Mély hálók tanításának standard módszere

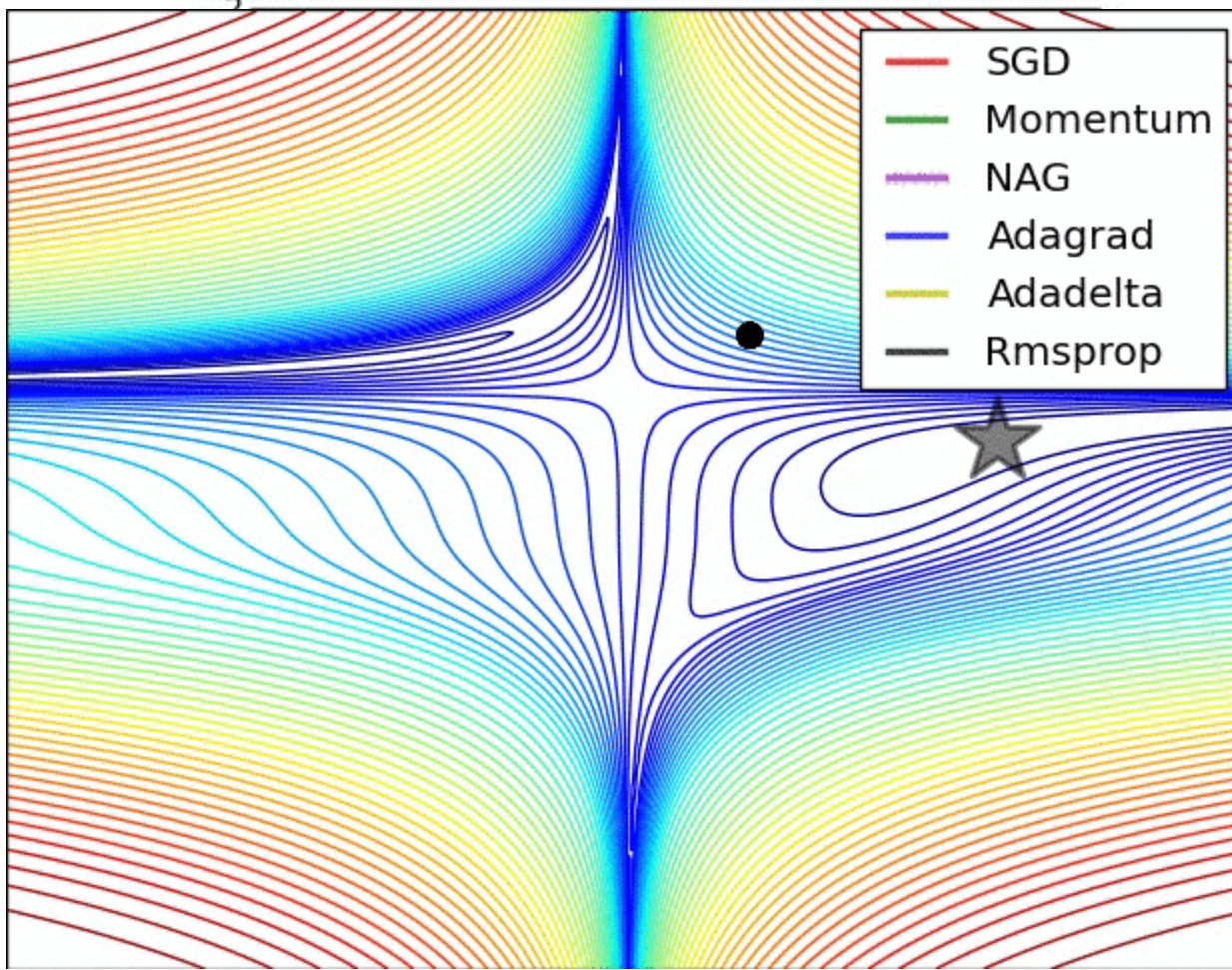
Elsőrendű optimalizációs példák

- Kvadratikus hibafelület:



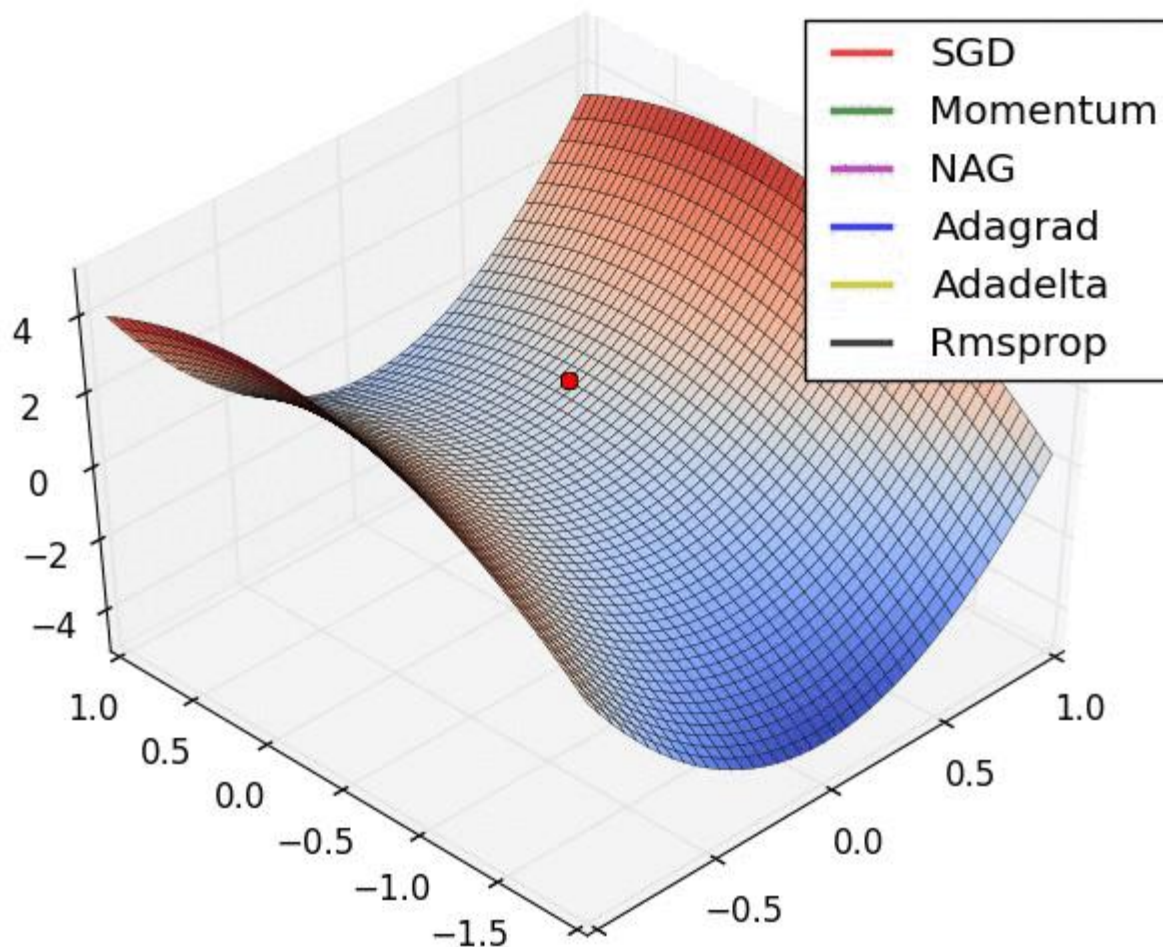
Elsőrendű optimalizációs példák

- Összetettebb hibafelület:



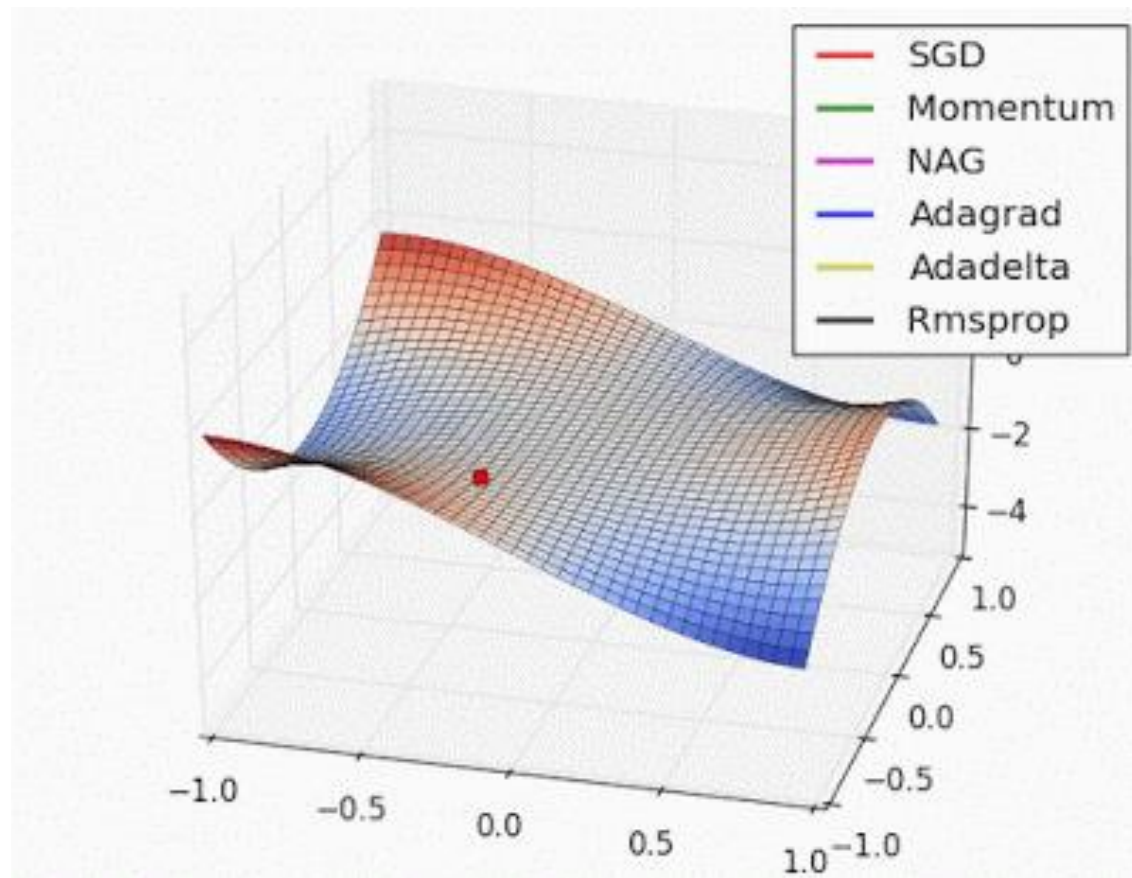
Elsőrendű optimalizációs példák

- Nyeregpont:



Elsőrendű optimalizációs példák

- Két irányból inflexiós pont:

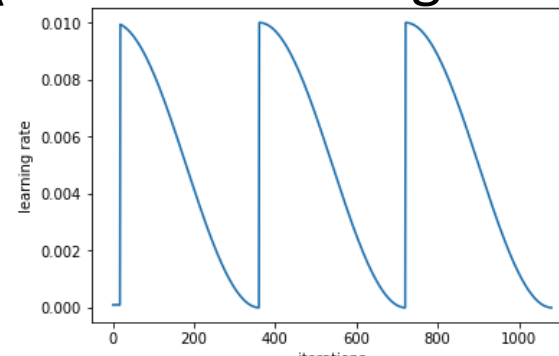


Adaptív módszerek értékelése

- Bazíroznak arra, hogy jelentősen eltérően fontosak az egyes paraméterek
 - Mély NH-knál ez tipikus, más területen viszont ritkán az
 - Láttunk már ilyen törekvéseket (SVM)
- Ha a koordináta tengellyel korrelálatlanok az autokorrelációs mtx. sajátértékei, akkor nem segítenek

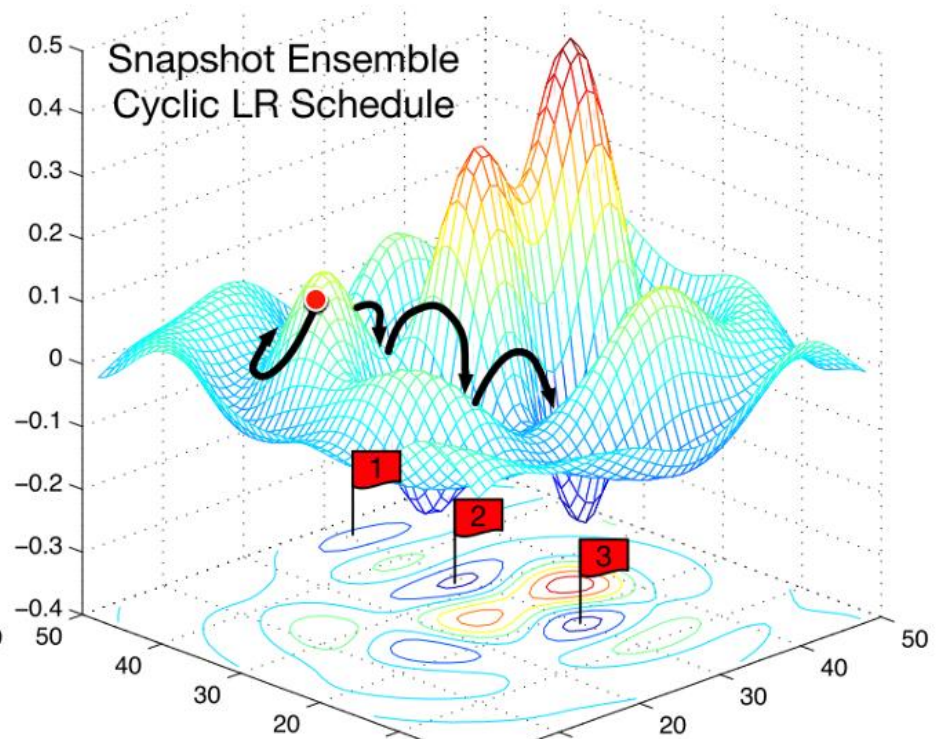
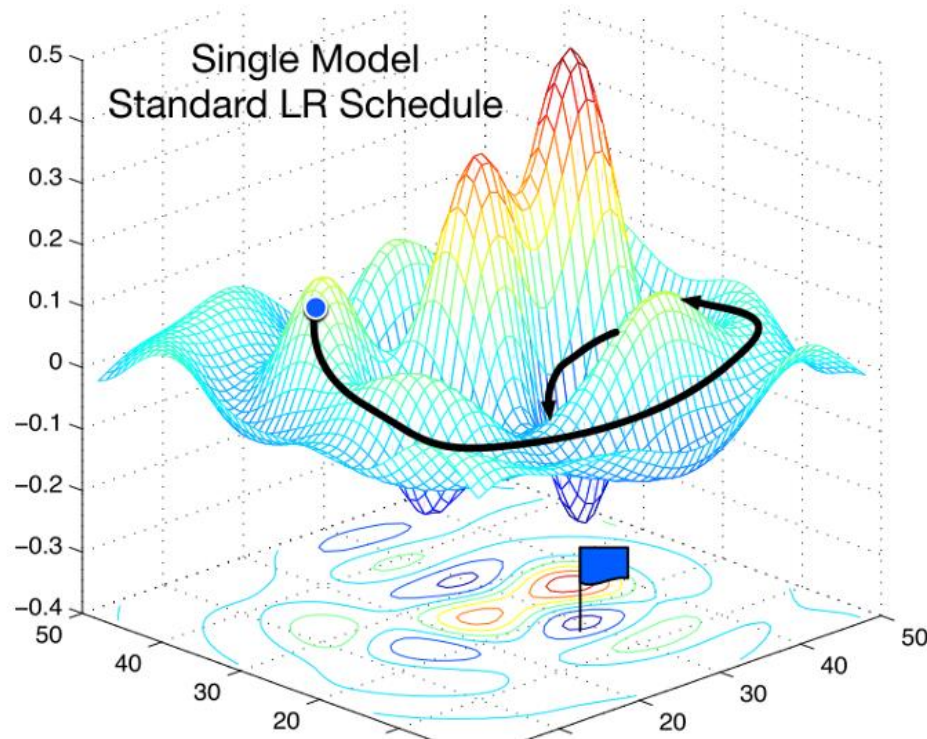
Optimalizáció hiper paramétere

- Bátorági tényező:
 - Konvergenciához szükséges értéke maximalizált
 - Négyzetes hibafüggvény esetén: $\mu \leq 1/\lambda_{\max} \{ \mathbf{R}_{xx} \}$
 - Általános esetben: $\mu_{\max} \sim 1/L \{ C(\cdot) \}$
 - Learning rate decay elve:
 - Kezdetben nagy bátorági tényező (minél hamarabb opt. közelébe jussunk)
 - Majd ennek értékét redukáljuk (kicsiket lépünk, hogy be is találjunk)
 - Learning rate decay restarttal (Ciklikus bátorági tényező):
 - Motiváció: rosszul általánosító optimumok elkerülése



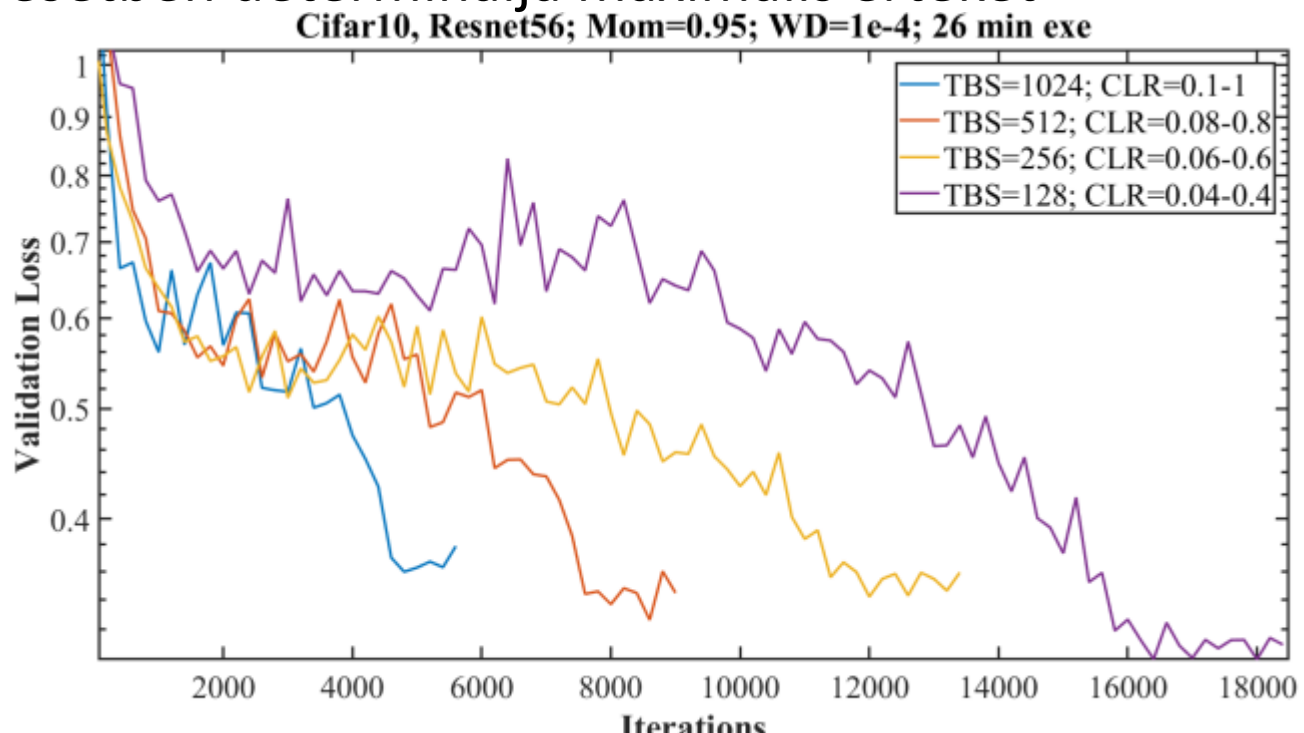
Optimalizáció hiper paramétereit (*)

- Ciklikus bátorsági tényezővel szakértő együttes:



Optimalizáció hiper paramétere

- Batch mérete:
 - Hibafelület közelítésének zaját szabályozza
 - Kisebb batch esetén nagyobb zaj (regularizálunk)
 - Hardver sok esetben determinálja maximális értékét
 - GPU VRam



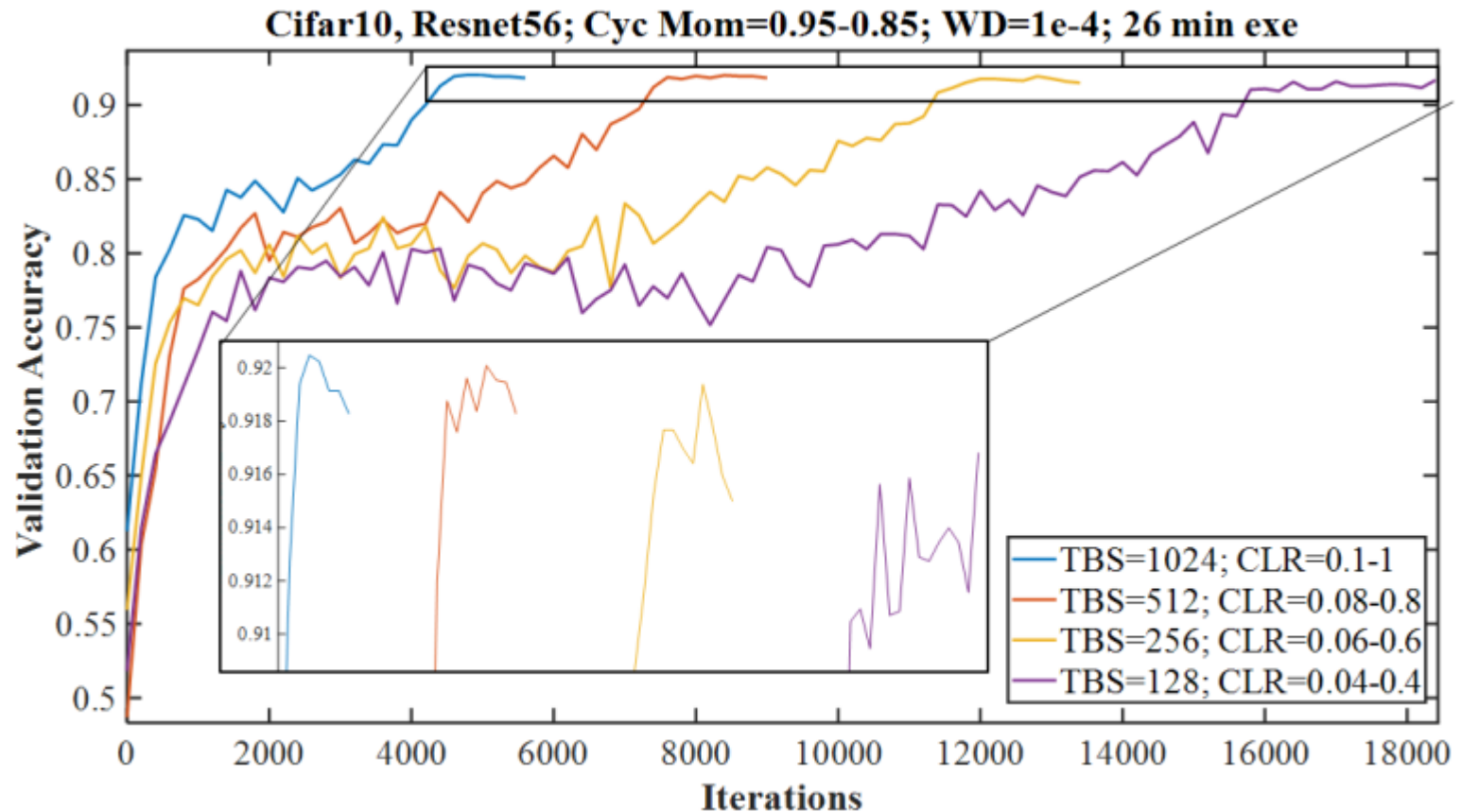
Optimalizáció hiper paramétere

- Momentum:

- $\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + (\mathbf{v}_k = \beta \cdot \mathbf{v}_{k-1} - \mu \cdot \nabla C \{ \boldsymbol{\theta}_{k-1} \}) \quad \beta \in (0.5, 1)$
- látható, hogy a bátorsági tényező és a momentum tag „hasonlóan” növeli a lépés nagyságát
 - Túl nagy momentum viszont lomhává teszi a rendszert (kevésbé tud konvergálni szűkebb tartományokba)
 - Ellenben segít az inflexiós / nyeregponatokon átlendülésében
- Elv: ellentétesen hangoljuk értékét a bátorsági tényezővel
 - Ciklikusan csökkenő bátorsági tényező
 - Ciklikusan növekvő momentum tag (u.a. ciklusidővel)

Optimalizáció hiper paramétere

- Hiper paraméterek együttes hangolása



Inicializáció

- Sekély hálóknál elég csak a szimmetria megtörése
- Mély hálóknál ez két lehetőséghez vezethet:
 - Aktivációk eltűnése a kimenet felé haladva (baj, mert a hiba visszaterjesztésnél 0-val szorzunk, nem tanul)
 - Aktivációk felrobbanása (végtelennel szorzunk, ez sem jó)
- Xavier Glorot inicializáció:
 - Az alkalmazott nemlinearitás függvényében úgy választjuk meg a kezdeti súlyokat, hogy az aktivációk (várható) eloszlása (a réteg sorszámától függetlenül) egyenletes maradjon.