

Szoftver mély neuronhálók alkalmazásához

10. előadás

Kovács Bálint, Varga Viktor
ELTE IK Mesterséges Intelligencia Tanszék

Előző órán - Felügyelt tanulás

Adott: A tanítóminta (training set), input-címke párok halmaza

$$\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$$

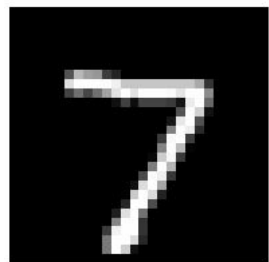
$$x \in X \subset \mathbb{R}^n, y \in Y \subset \mathbb{R}^k$$

Feladat: A címke (az elvárt output) minél jobb becslése az inputból.

Azaz, keresünk olyan h_θ függvényt (hipotézisfüggvényt), melyre:

$$h_\theta(x) = \hat{y} \approx y$$

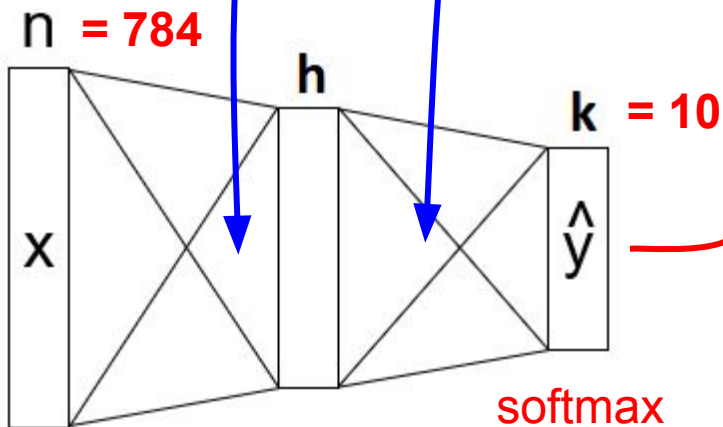
Előző órán - MLP kézírás felismerésére (MNIST)



x

súlyok
hangolása

J (categorical
crossentropy)



0.01
0.11
0.04
0.01
0.02
0.01
0.01
0.78
0.0
0.01

argmax

7

y[^]

y

0
0
0
0
0
0
0
1
0
0

Előző órán - MLP kézírás felismerésére (MNIST)

Eddig:

Pl. logisztikus regresszió koleszterinszint becslésére

→ 3 feature, 4 paraméter, több száz mintaelem
<

Most:

MLP egyetlen rejtett réteggel számjegyek klasszifikálására

→ 784 feature, 636 ezer paraméter, 60 ezer mintaelem
>

Mi történhet, ha túl sok paraméterünk van?

→ **Túltanulás**

Előző órán - Túltanulás elkerülése

Mit tehetünk a túltanulás elkerülése érdekében?

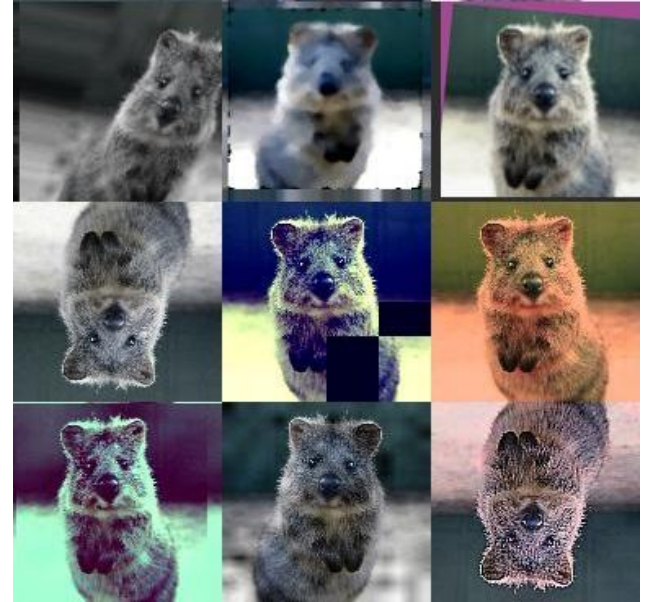
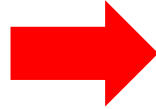
- Használjunk egyszerűbb modellt (pl. kevesebb paraméter)!
- Szerezzünk be több tanítóadatot!
- Regularizáció (pl. $\|W\|_2^2$ tag a költségben - L2 reg.)
- Early stopping

Továbbá:

- Adat-augmentáció
- Zajosítás
- Dropout

Előző órán - Adat augmentáció

Adat-augmentáció (Data augmentation)



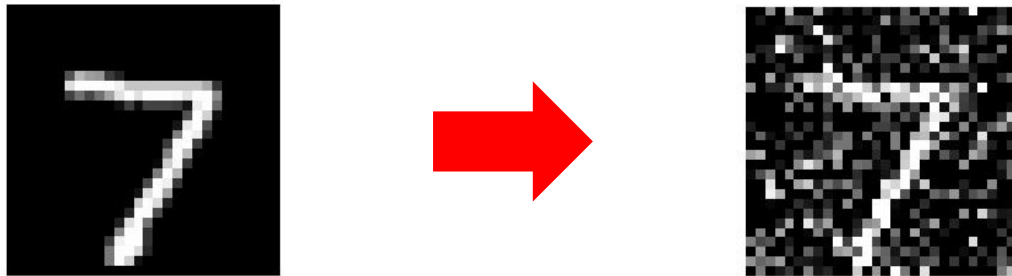
További tanítóadat beszerzése sokszor
nem lehetséges.

→ **Készítsünk “új” elemeket a meglevőkből,
különböző transzformációkkal**

Előző órán - Zajosítás

Zajosítás (az adat-augmentáció egy fajtája)

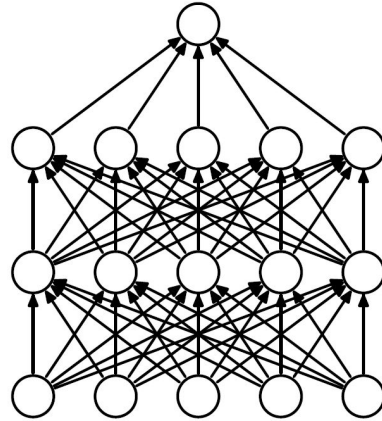
Normál- (Gauss-) eloszlású zajt az input változókhoz adva csökkenthető a túltanulás mértéke.



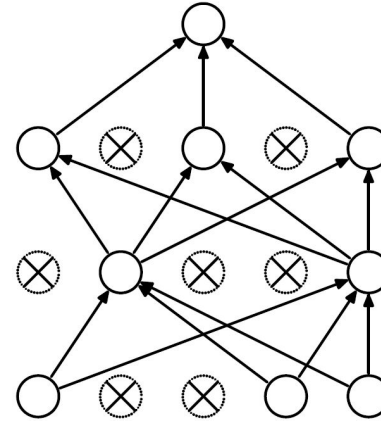
A zaj megakadályozza a pontos értékek “memorizálását”.

Előző órán - Dropout

Dropout (a zajosítás egy fajtája)



(a) Standard Neural Net



(b) After applying dropout.

**Exponenciálisan sok kisebb háló együttese →
kevésbé érzékeny a túltanulásra**

Előző órán - Az MLP nem ideális képfeldolgozásra

Multilayer Perceptron:

- **Ideális ha egy input változó “fix jelentést” hordoz**
(képek esetén ez tipikusan nem igaz)
- **Transzláció invariancia hiánya**
(mintázatok felismerése a képen pozícióhoz kötött)
- **Figyelman kívül hagyja input változók szomszédsági viszonyát**



Előző órán - 2D (diszkrét) konvolúció

$$\hat{y}[i, j] = (x * w)[i, j] = \langle x[i..i+U, j..j+V], w \rangle =$$
$$= \sum_{u=1}^U \sum_{v=1}^V x[i+u, j+v] \cdot w[u, v]$$

1	2	2	-2
1	0	2	4
2	-1	-1	3
0	0	-3	-2

x

*

1	0	3
2	-1	-1
0	1	-3

w

=

9	-20
22	11

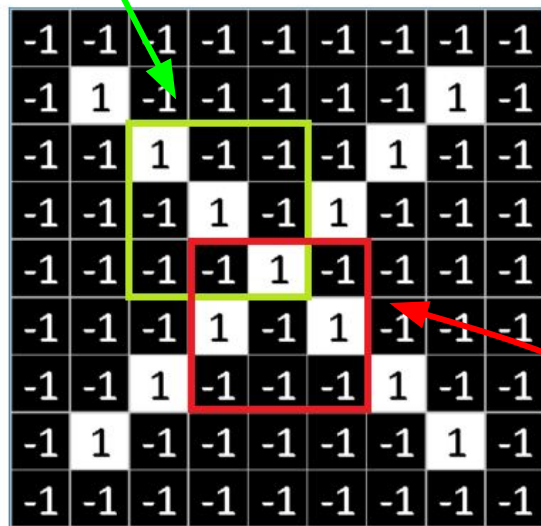
y

$$0 \cdot 1 + 2 \cdot 0 + 4 \cdot 3 + (-1) \cdot 2 + (-1) \cdot (-1) + 3 \cdot (-1) + 0 \cdot 0 + (-3) \cdot 1 + (-2) \cdot (-3) = 11$$

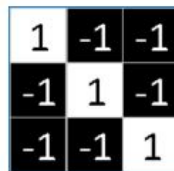
Előző órán - Konvolúció textúra felismeréséhez

jó illeszkedés: magas output érték

input



filter



=

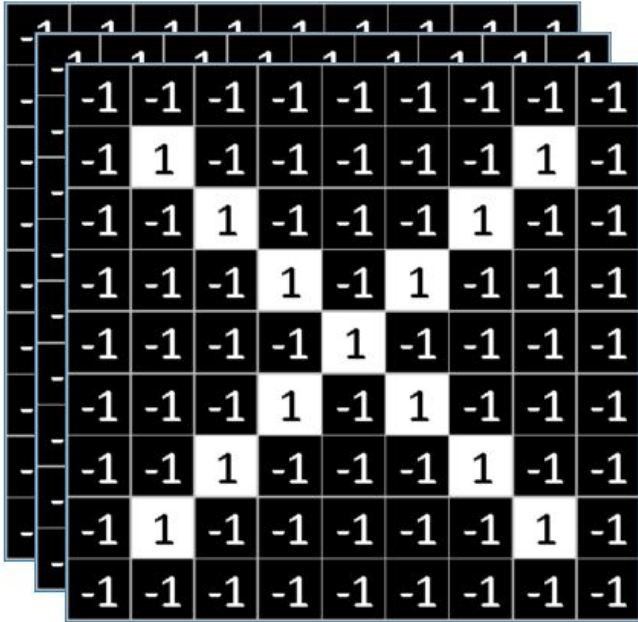
output
(heatmap, feature map, hőtérkép)



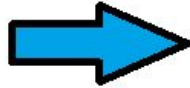
rossz illeszkedés: alacsony output érték

Előző órán - Konvolúciós réteg csatornák

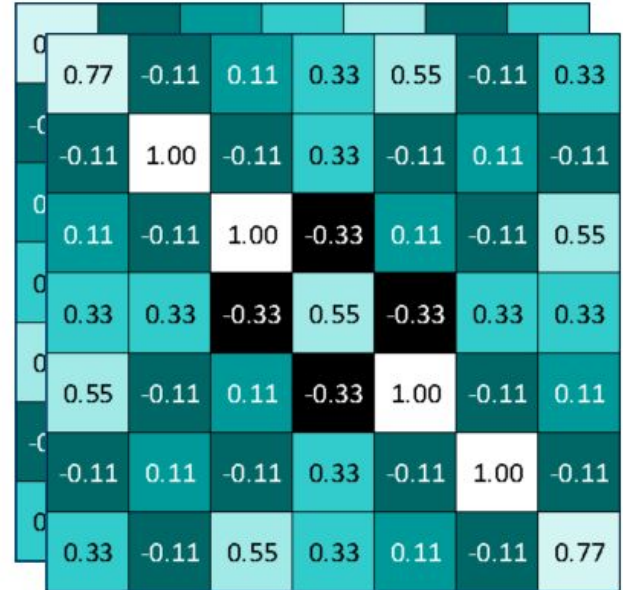
P darab input csatorna



Konvolúciós réteg
Q darab (**P** csatornás)
filterrel.



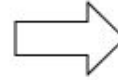
Q darab output csatorna



Előző órán - Downsampling / pooling réteg

Max pooling (2x2)

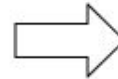
5	2	-3	0
2	-1	4	-1
-4	-4	-3	0
3	5	0	-1



5	4
5	0

Average pooling (2x2)

5	2	-3	0
2	-1	4	-1
-4	-4	-3	0
3	5	0	-1

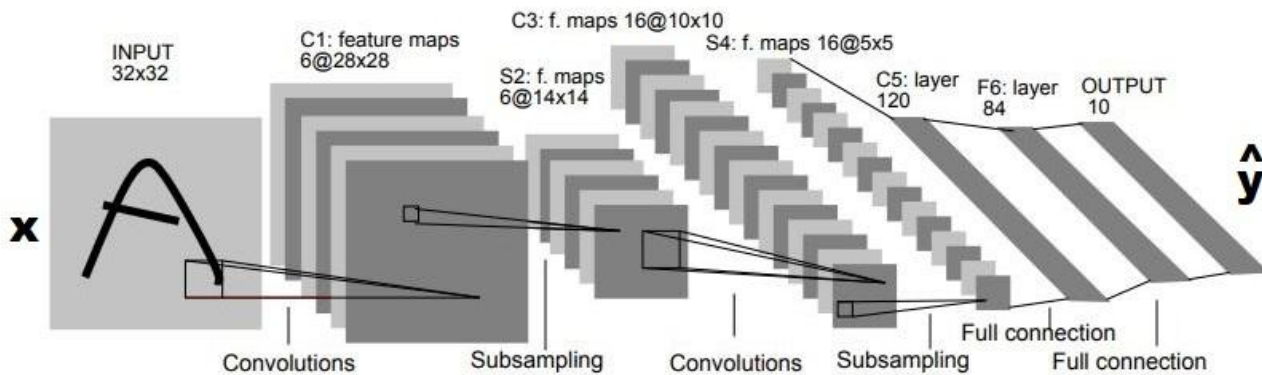


2	0
0	-1

Előző órán - Konvolúciós háló (LeNet-5)

**Felváltva alkalmazott
konvolúciós és pooling
rétegek.**

**Végül,
teljesen összekötött
rétegek.**



Előző órán - Konvolúciós háló felépítése

A váltakozó konvolúciós és pooling rétegek a **hierarchikus mintázatfelismeréshez** ideálisak.

Nagyobb, magasabb szintű mintázatokat több kisebb, egyszerűbb mintázat kombinációjaként tanulja a háló felismerni.

Előző órán - Keras, LeNet-5 (egyszerűsített)

```
model = Sequential()  
model.add(Conv2D(6, (5, 5), activation="tanh", \n                 input_shape=(32, 32, 1)))  
model.add(AveragePooling2D(pool_size=(2, 2)))  
model.add(Conv2D(16, (5, 5), activation="tanh"))  
model.add(AveragePooling2D(pool_size=(2, 2)))  
model.add(Flatten())  
model.add(Dense(120, activation='tanh',))  
model.add(Dense(84, activation='tanh'))  
model.add(Dense(10, activation='softmax'))  
model.compile(loss='categorical_crossentropy', optimizer=sgd)
```

(n_mintaelem, 32, 32, 1)

(n_mintaelem, 28, 28, 6)

(n_mintaelem, 14, 14, 6)

(n_mintaelem, 10, 10, 16)

(n_mintaelem, 5, 5, 16)

(n_mintaelem, 400)

(n_mintaelem, 120)

(n_mintaelem, 84)

(n_mintaelem, 10)

Konvolúciós háló hiperparaméterei

Padding:

0	0	0	0	0	0	0	0
0	3	3	4	4	7	0	0
0	9	7	6	5	8	2	0
0	6	5	5	6	9	2	0
0	7	1	3	2	7	8	0
0	0	3	7	1	8	3	0
0	4	0	4	3	2	2	0
0	0	0	0	0	0	0	0

$6 \times 6 \rightarrow 8 \times 8$

*

1	0	-1
1	0	-1
1	0	-1

3×3

=

-10	-13	1			
-9	3	0			

6×6

Az input kiegészítése a széleken (például nullákkal), hogy az output méretét megváltoztassuk (például ugyanakkorra, mint az input)

Konvolúciós háló hiperparaméterei

Padding:

`Conv2D(..., padding="same", ...)`

0	0	0	0	0	0	0	0
0	3	3	4	4	7	0	0
0	9	7	6	5	8	2	0
0	6	5	5	6	9	2	0
0	7	1	3	2	7	8	0
0	0	3	7	1	8	3	0
0	4	0	4	3	2	2	0
0	0	0	0	0	0	0	0

$6 \times 6 \rightarrow 8 \times 8$

*

1	0	-1
1	0	-1
1	0	-1

3×3

=

-10	-13	1			
-9	3	0			

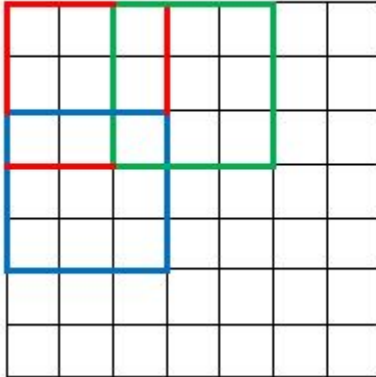
6×6

Az input kiegészítése a széleken (például nullákkal), hogy az output méretét megváltoztassuk (például ugyanakkorra, mint az input)

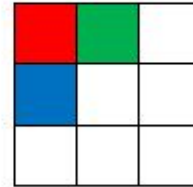
Konvolúciós háló hiperparaméterei

Stride (lépésköz):

7 x 7 Input Volume



3 x 3 Output Volume



A konvolúciót nagyobb lépésközzel is alkalmazhatjuk az inputon, ami kisebb képet eredményez... Pooling rétegek stride-ja is változhat.

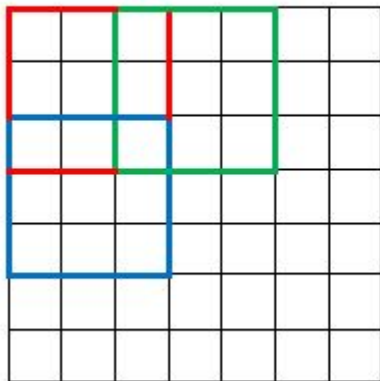
Konvolúciós háló hiperparaméterei

Stride (lépésköz):

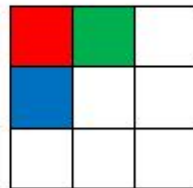
```
Conv2D(..., strides=(2, 2), ...)
```

```
MaxPooling2D(..., strides=(2, 2), ...)
```

7 x 7 Input Volume



3 x 3 Output Volume



A konvolúciót nagyobb lépésközzel is alkalmazhatjuk az inputon, ami kisebb képet eredményez... Pooling rétegek stride-ja is változhat.

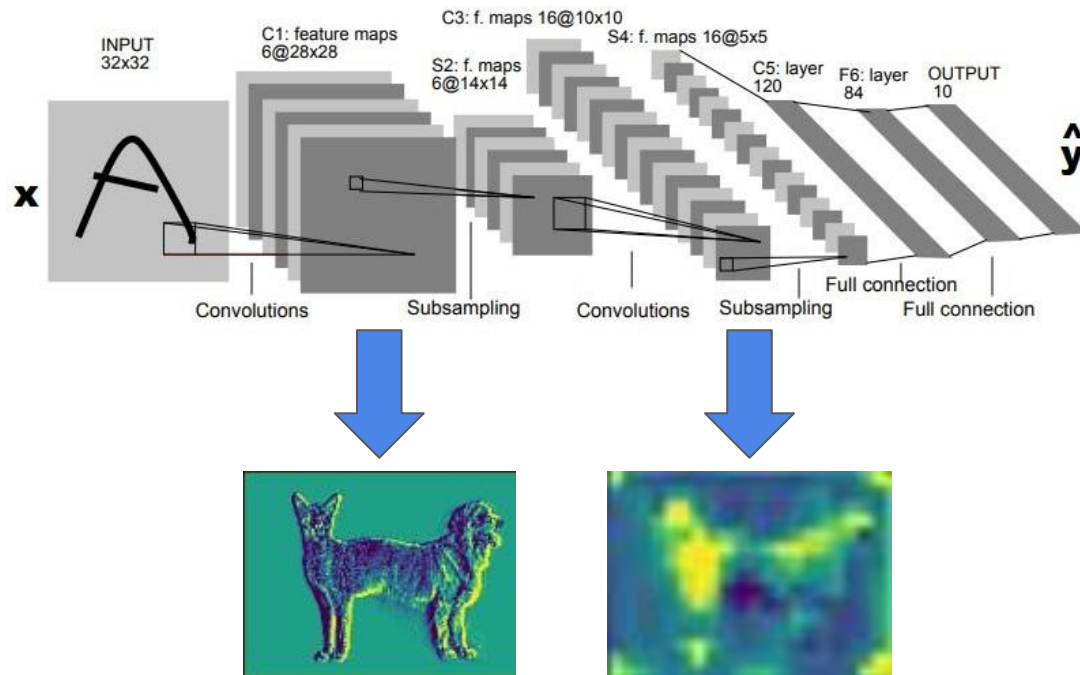
Mit tanul egy konvolúciós háló?

Heatmap vizualizáció

https://colab.research.google.com/drive/1I4K0x-r2f_ipKUA_BtgfkcXNywYw9dY-?usp=sharing

Mit tanul egy konvolúciós háló?

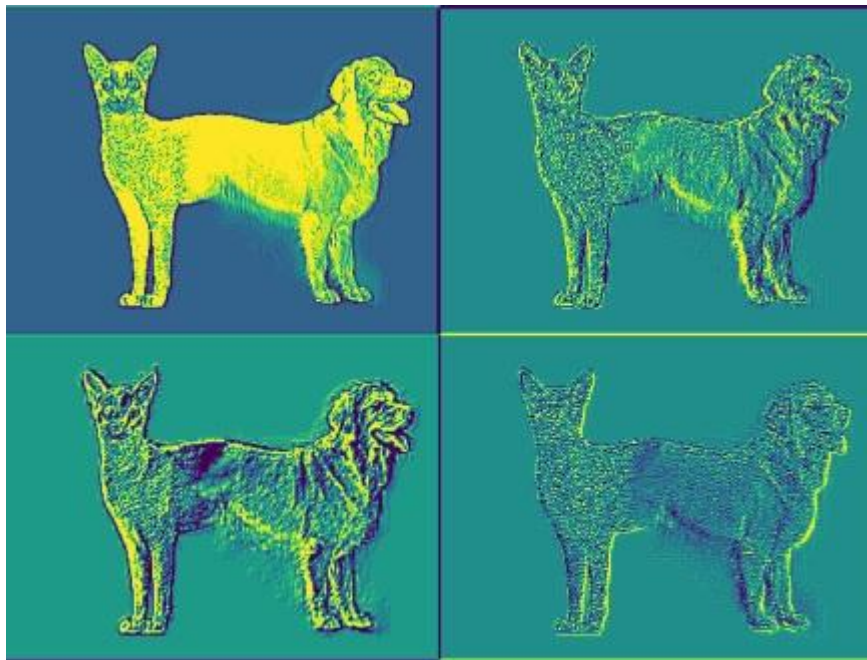
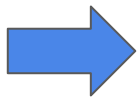
Heatmap vizualizáció



Mit tanul egy konvolúciós háló?

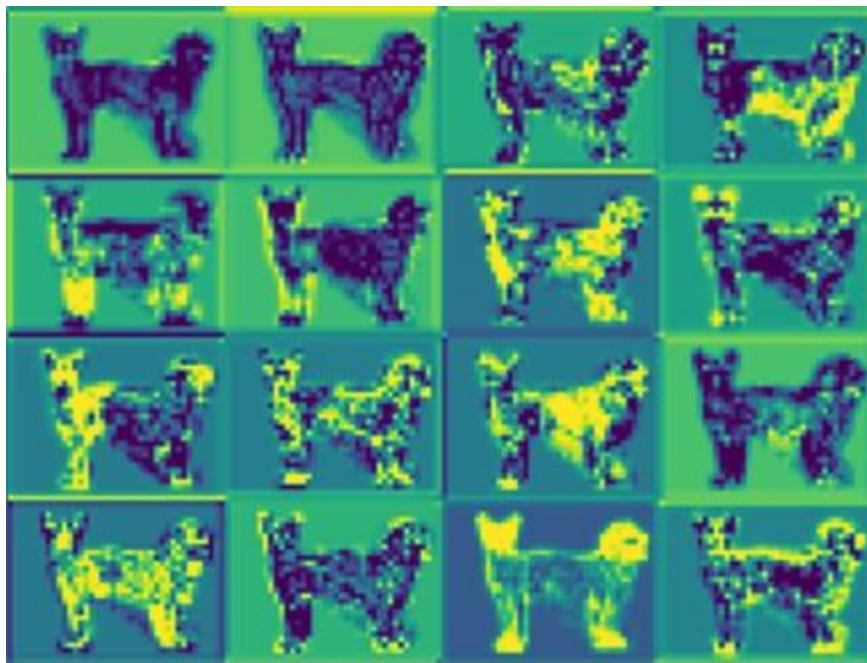
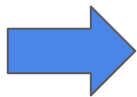
Heatmap vizualizáció - első réteg

(ImageNet-en tanult mélyháló)



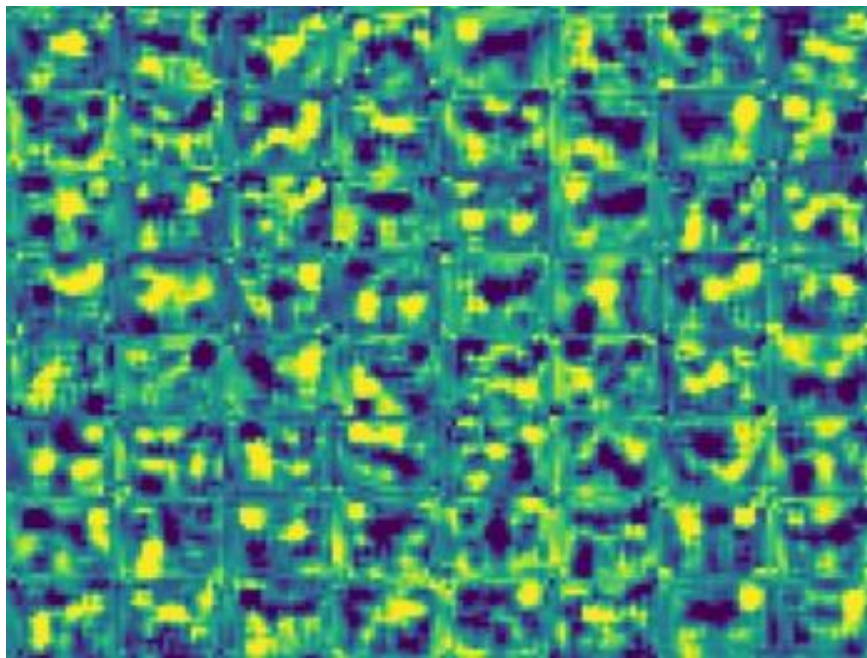
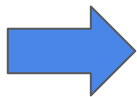
Mit tanul egy konvolúciós háló?

Heatmap vizualizáció - középső réteg



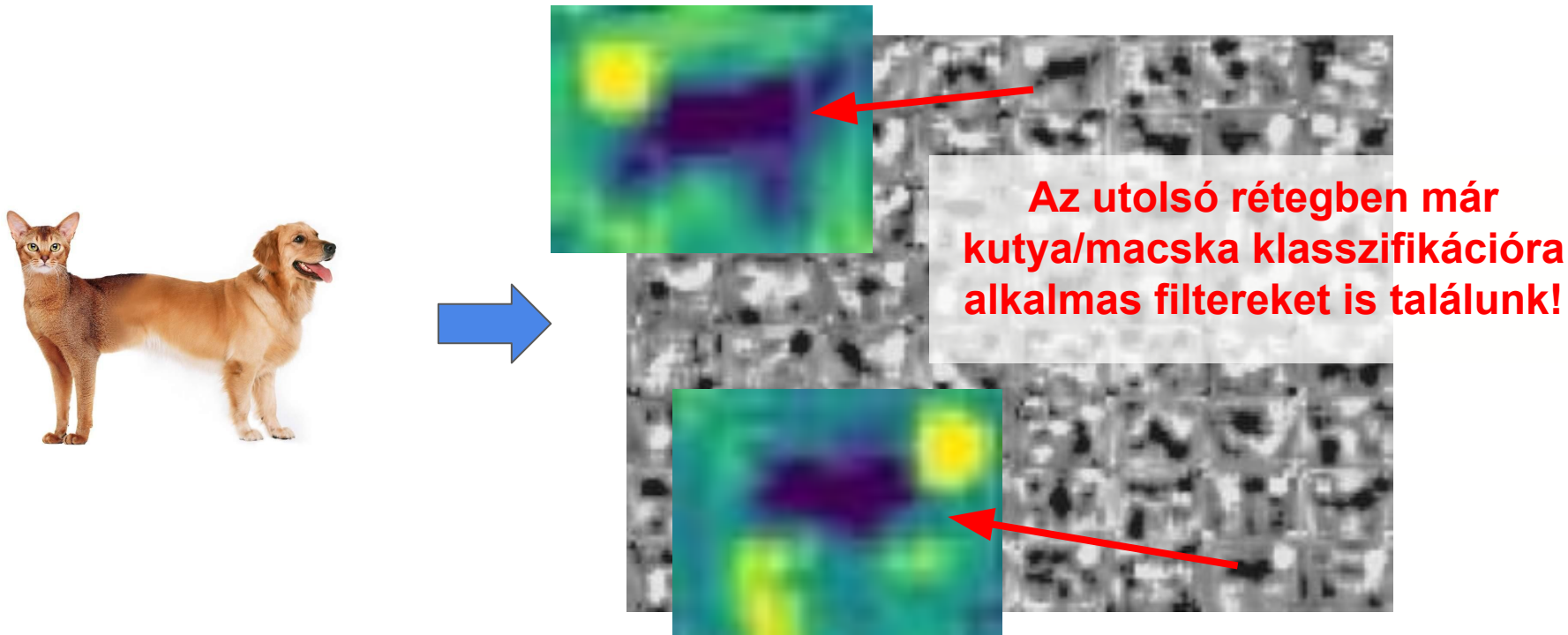
Mit tanul egy konvolúciós háló?

Heatmap vizualizáció - utolsó réteg



Mit tanul egy konvolúciós háló?

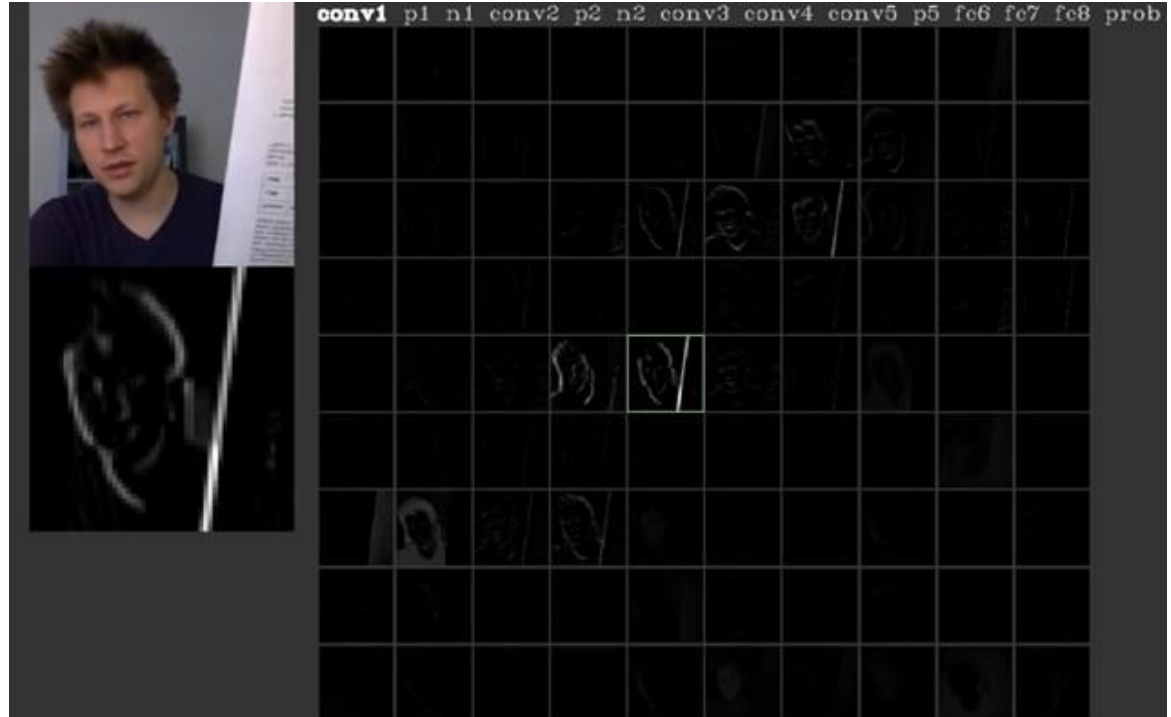
Heatmap vizualizáció - utolsó réteg



Mit tanul egy konvolúciós háló?

Első konv. réteg output

Heatmap vizualizáció



Mit tanul egy konvolúciós háló?

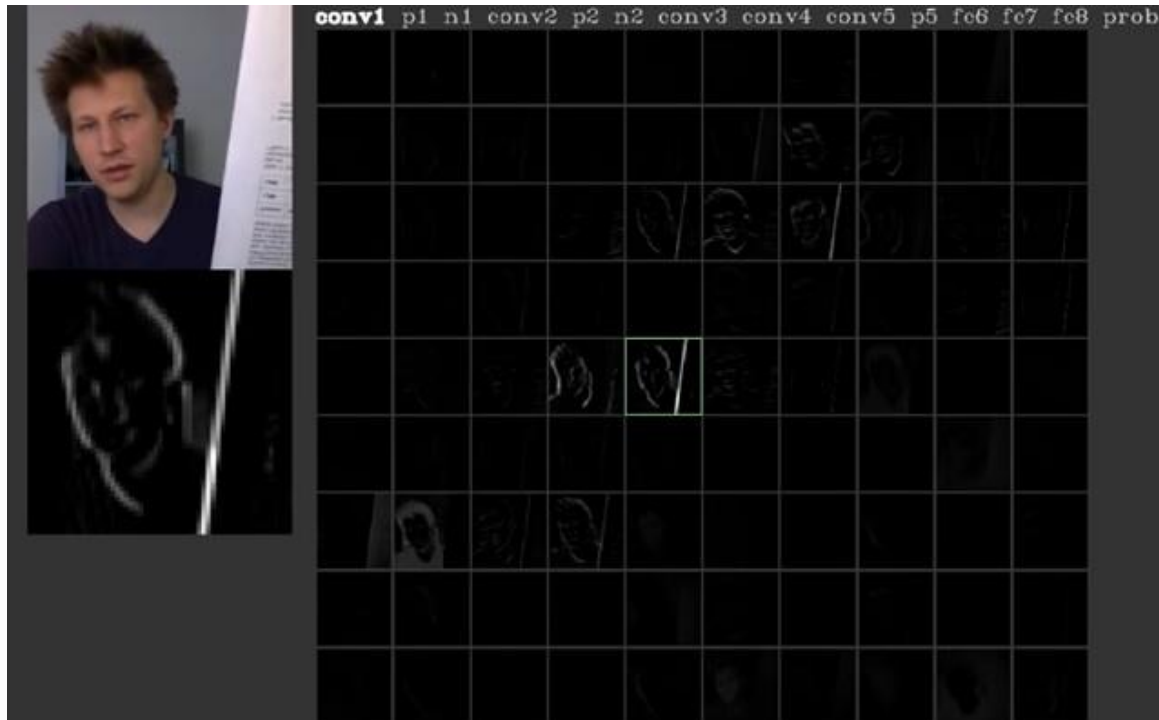
Első konv. réteg output

Heatmap vizualizáció

**ImageNet-en tanult
mélyháló (AlexNet, 2012)**

**Az első réteg az alacsony
szintű mintázatokért felel:**

az egyik filter a függőleges,
sötét → világos átmenetet
tanulta meg



Mit tanul egy konvolúciós háló?

Ötödik konv. réteg output

Heatmap vizualizáció



Mit tanul egy konvolúciós háló?

Ötödik konv. réteg output

Heatmap vizualizáció

ImageNet-en tanult
mélyháló (AlexNet, 2012)

Az utolsó rétegek a magas
szintű mintázatokért
felelnek:

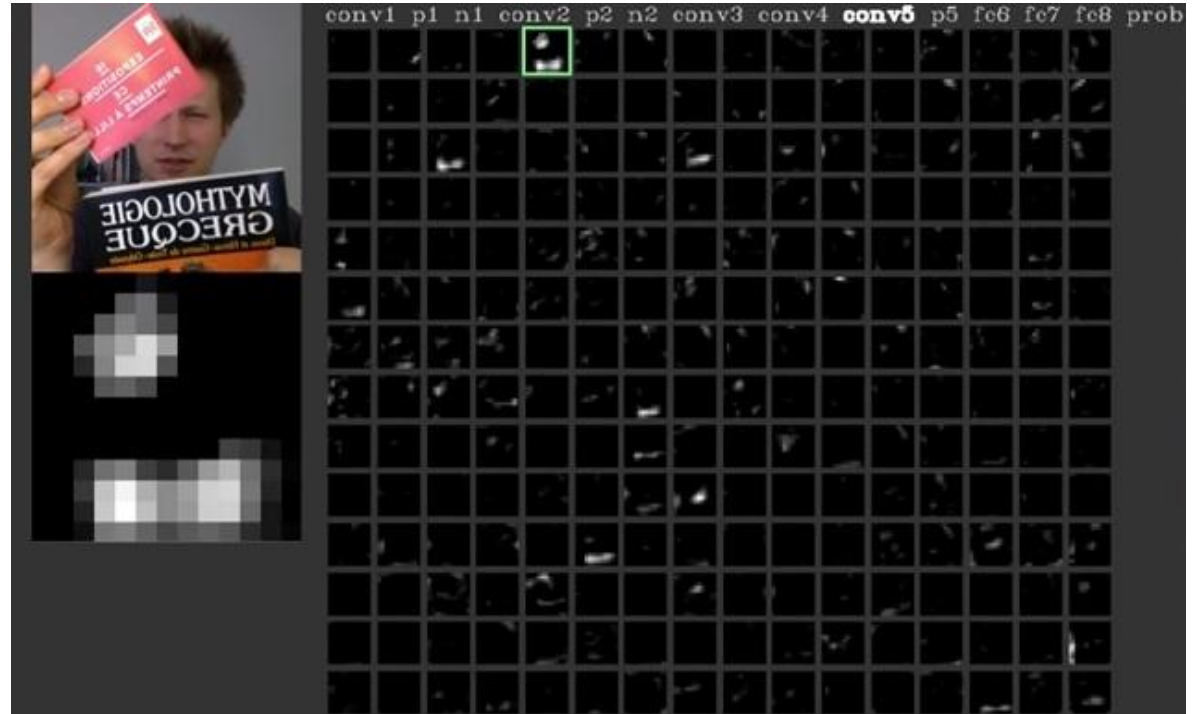
egy filter
az arcokat ismeri fel



Mit tanul egy konvolúciós háló?

Ötödik konv. réteg output

Heatmap vizualizáció



Mit tanul egy konvolúciós háló?

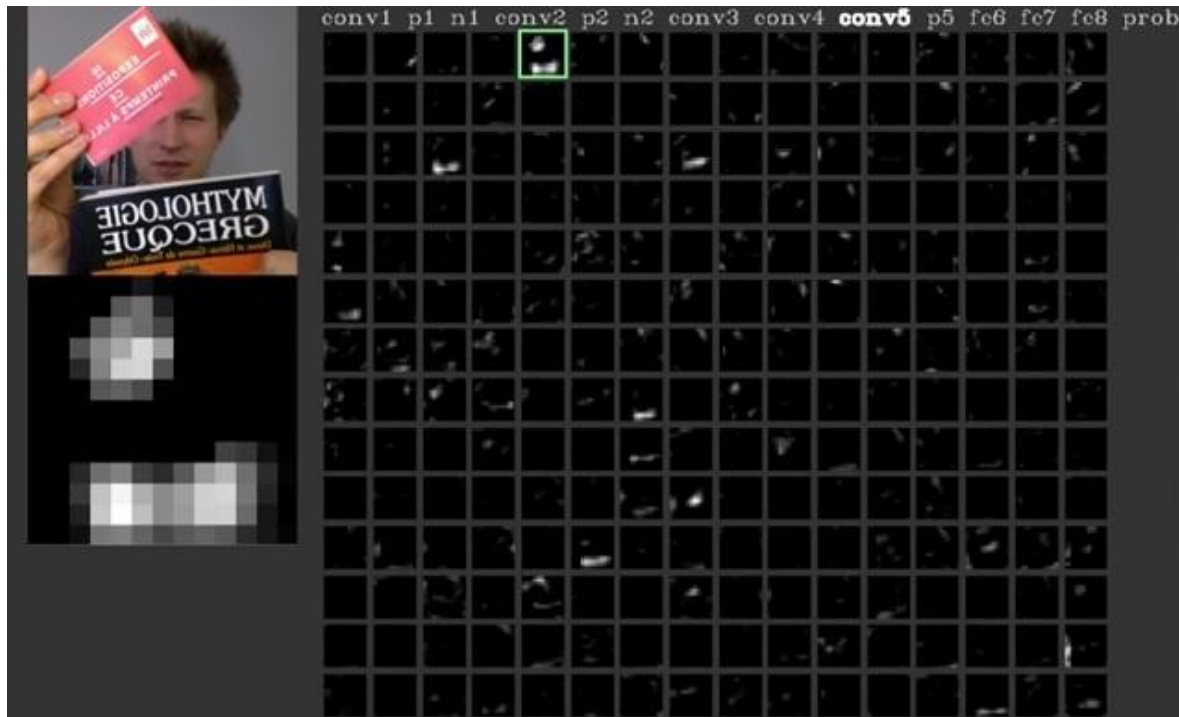
Ötödik konv. réteg output

Heatmap vizualizáció

**ImageNet-en tanult
mélyháló (AlexNet, 2012)**

**Az utolsó rétegek a magas
szintű mintázatokért
felelnek:**

egy filter
a nyomtatott szöveget
ismeri fel



Mit tanul egy konvolúciós háló?

Megfordítva: Keressünk olyan inputot, ami magas értékeket ad egy-egy heatmap-en!

Mit tanul egy konvolúciós háló?

Megfordítva: Keressünk olyan inputot, ami magas értékeket ad egy-egy heatmap-en!



első konv. réteg



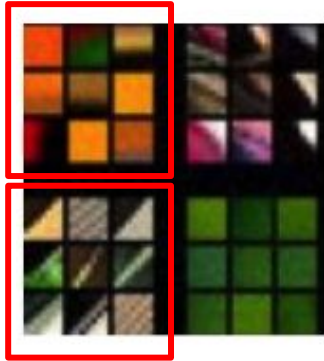
középső konv. réteg



utolsó konv. réteg

Mit tanul egy konvolúciós háló?

Az első réteg 1. filtere ilyen inputra a legaktívabb



... 2. filter

első konv. réteg



középső konv. réteg



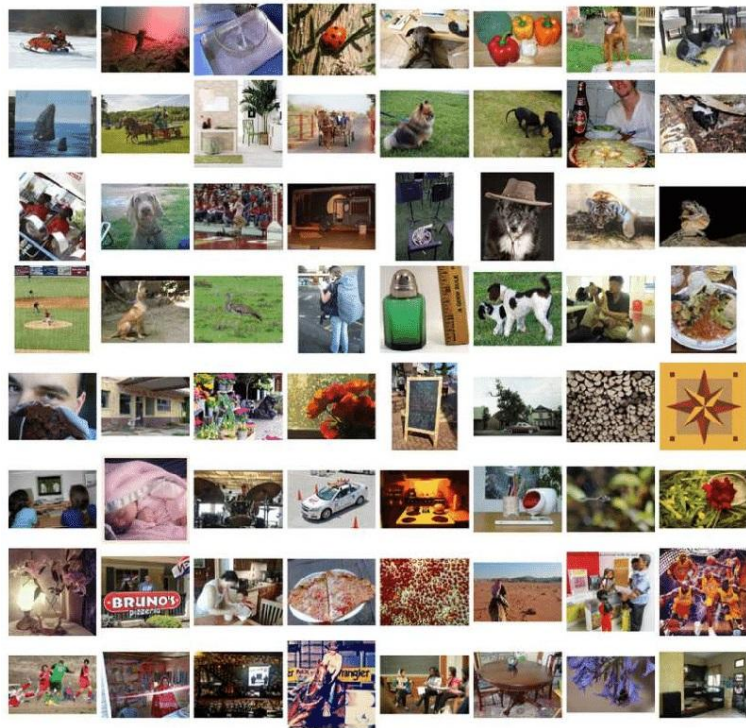
utolsó konv. réteg

Hierarchia: Az egymásra épülő rétegek egyre magasabb szintű alakzatokat tanulnak.

Emlékeztető: ImageNet adatbázis

Fényképek különböző objektum-kategóriákkal.

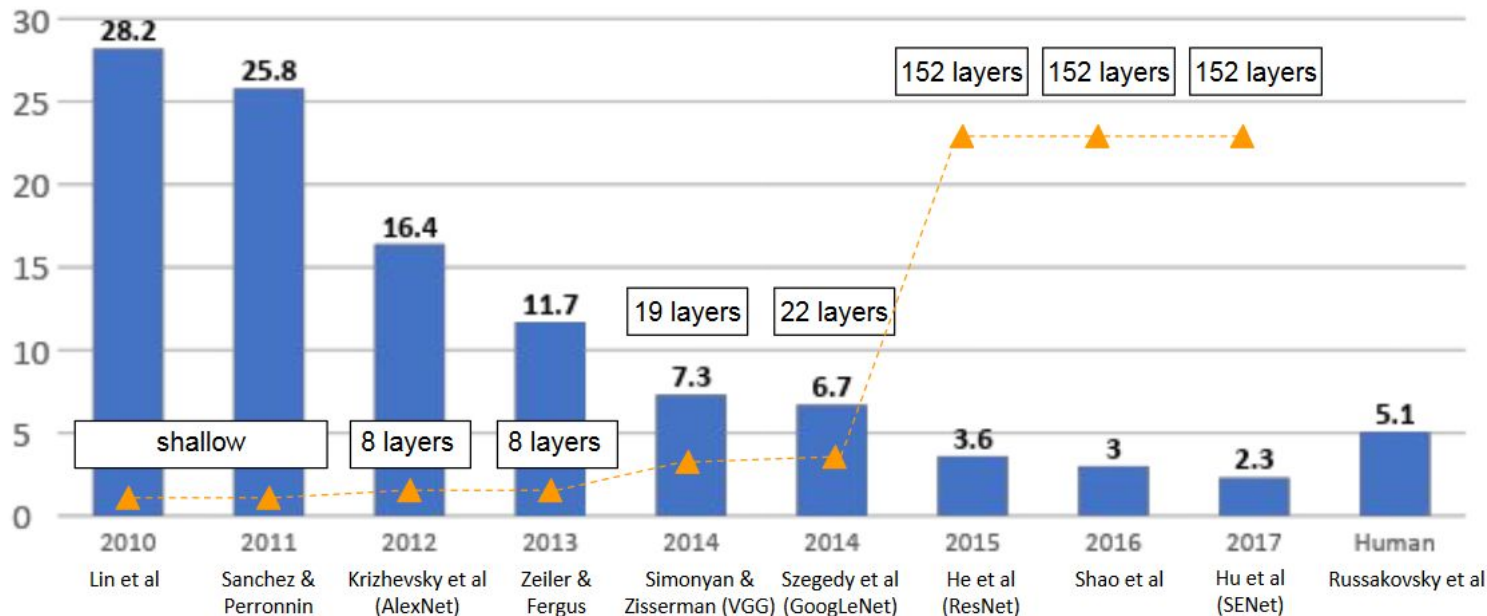
- 1000+ kategória, például:
 - Mocsári teknős
 - Gofrisütő
 - Norfolk terrier
 - Viadukt
- 1 megapixel körüli felbontás, színes
→ **kb. 3 millió input feature**



Emlékeztető: ImageNet adatbázis

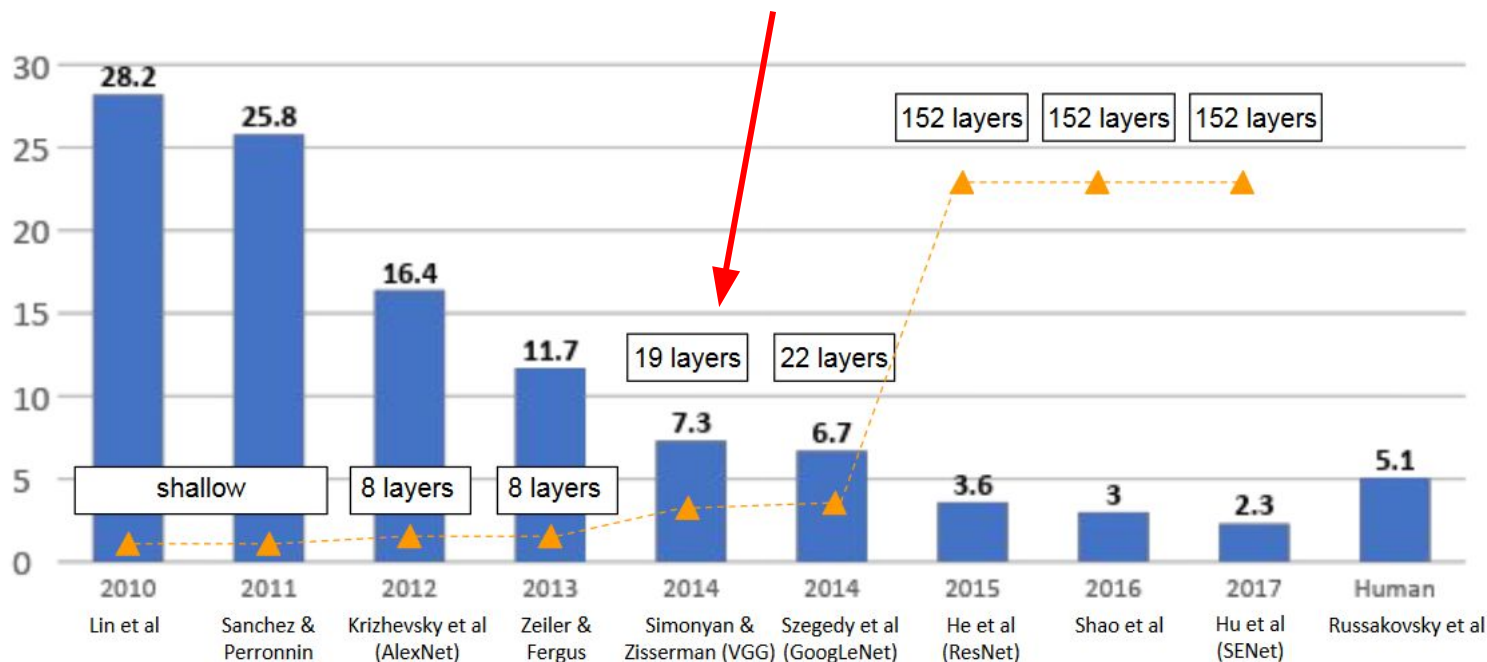
ImageNet ILSVRC challenge nyertesek

(1000 kat. kép klasszifikáció, 1-accuracy 5 próbálkozás esetén)

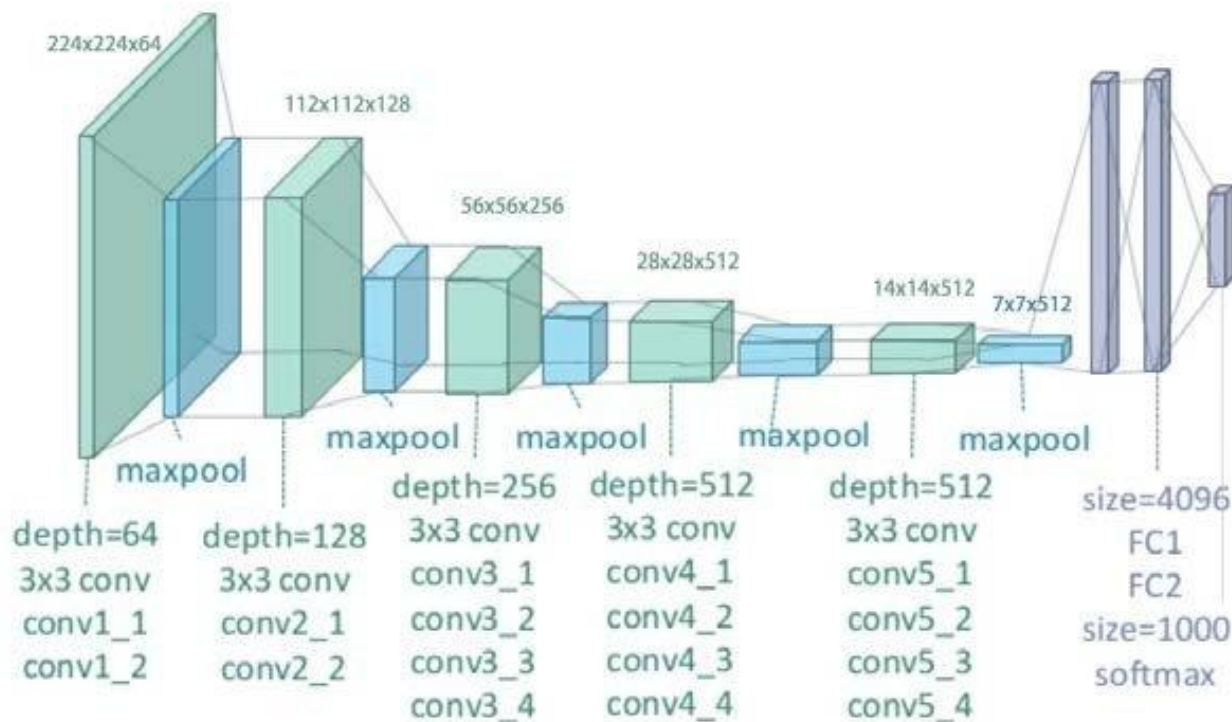


Emlékeztető: ImageNet adatbázis

Egy rendkívül népszerű
konvolúciós mélyháló: VGG-19

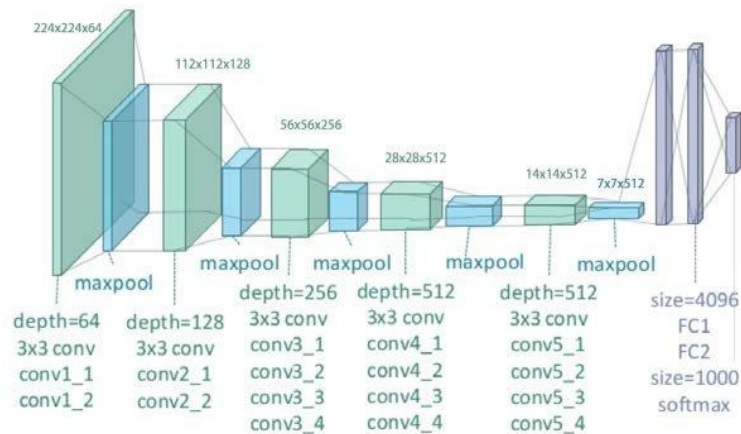


VGG-19



VGG-19

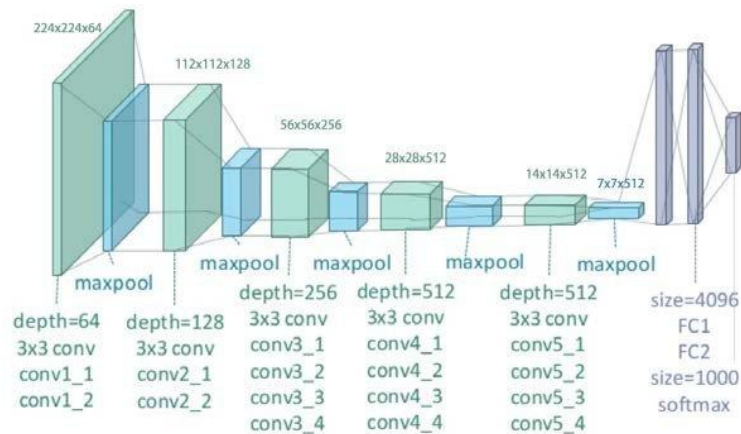
- 92.7% top-5 accuracy (ILSVRC)
- 19 paraméteres réteg
- 143 millió paraméter
- Egy iteráció: 20 GFLOPS
- Megjelenésekor **4 csúcskategóriás GPU-n 2-3 hétig tartott betanítani.**



Ez azt jelenti, hogy semmi értelme csúcskategóriás hardver nélkül nekiállni érdemi feladatokat megtanulni?

VGG-19

- 92.7% top-5 accuracy (ILSVRC)
- 19 paraméteres réteg
- 143 millió paraméter
- Egy iteráció: 20 GFLOPS
- Megjelenésekor **4 csúcskategóriás GPU-n 2-3 hétig tartott betanítani.**



Ez azt jelenti, hogy semmi értelme csúcskategóriás hardver nélkül nekiállni érdemi feladatokat megtanulni?

Nem!

Transfer learning

Fel kell ismernünk, hogy **nem szükséges minden egyes feladathoz előlről kezdeni a betanítást.**

Transfer learning: Egyfajta feladat tanulása közben szerzett tudás hasznosítása egy másik feladat megoldására.

Transfer learning - példa

Példa:



Nagyfelbontású fényképekről kutya/macska klasszifikáció. Rendelkezésünkre áll 500-500 példa a betanításhoz.

Hogyan oldjuk meg?

- Tanítsunk be egy igazán mély konvolúciós hálót!

Eredmény:

- Hamar túltanul. A sokmillió paramétert néhány száz képpel, még adat-augmentálással sem igazán lehet jól betanítani.

Transfer learning - példa

Példa:



Nagyfelbontású fényképekről kutya/macska klasszifikáció. Rendelkezésünkre áll 500-500 példa a betanításhoz.

Hogyan oldjuk meg?

- Tanítsunk be egy egyszerű konvolúciós hálót!

Eredmény:

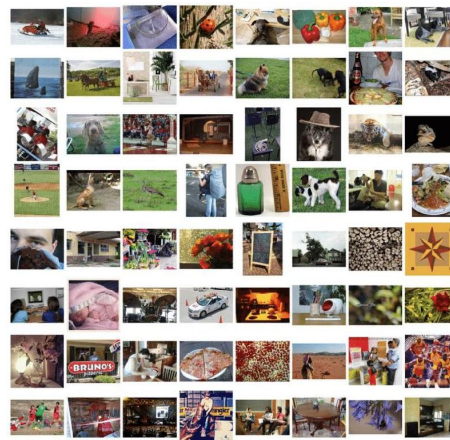
- Gyenge teljesítmény. Kevés paraméterrel rendelkező háló nem fog tudni bonyolult feladatokat jól megoldani.

Transfer learning - példa

Ehelyett:

1. Keressünk egy jóval nagyobb adatbázist, ami valamelyest hasonlít a céladatbázisunkra!

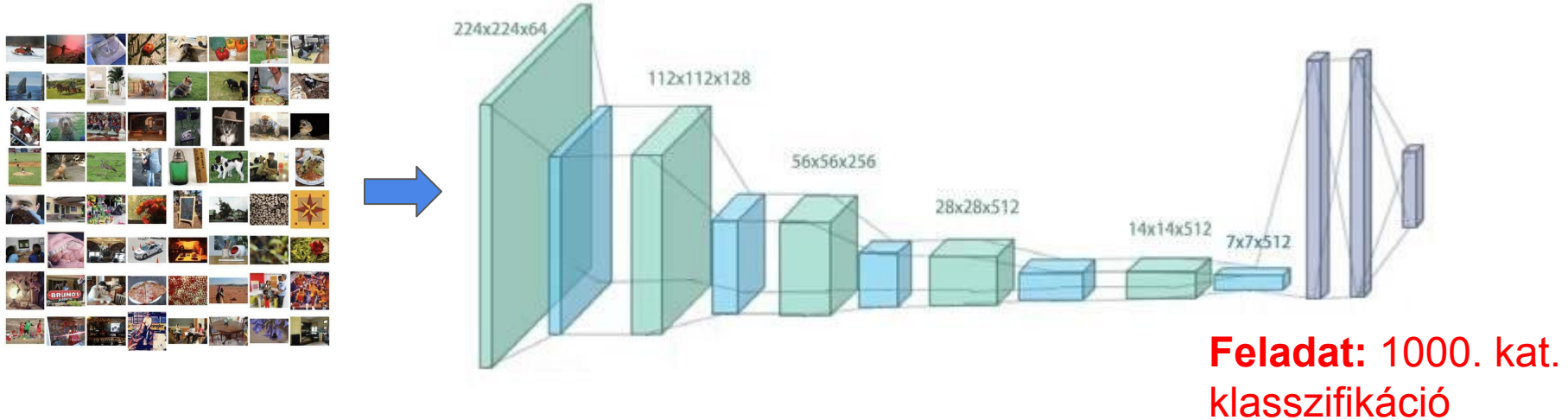
IMAGENET



Transfer learning - példa

Ehelyett:

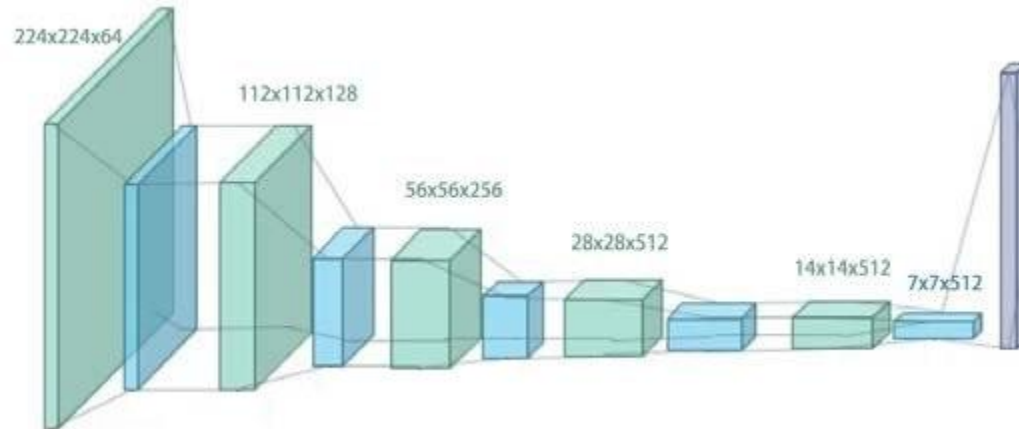
2. Tanítsunk be egy igazán mély neuronhálót a nagy adatbázison!



Transfer learning - példa

Ehelyett:

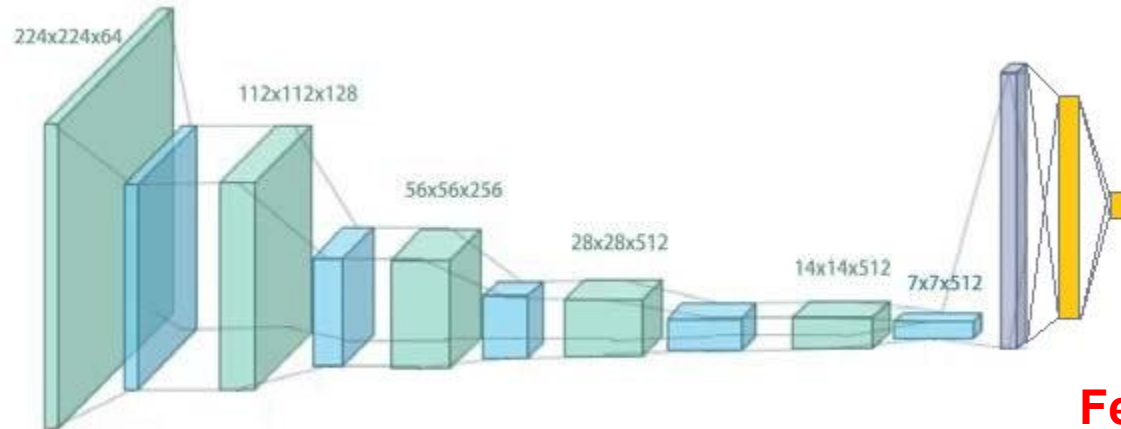
3. Dobjuk el az **előtanított (pretrained)** háló utolsó rétegeit!



Transfer learning - példa

Ehelyett:

- Adjunk a csonka előtanított hálóhoz új, a célfeladathoz igazodó rétegeket!

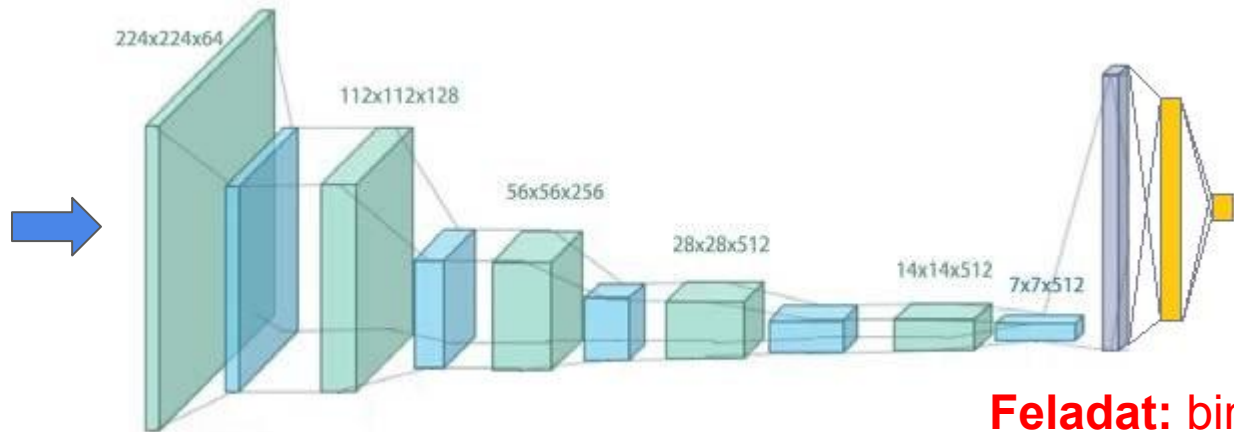


Feladat: bináris
klasszifikáció

Transfer learning - példa

Ehelyett:

5. Finomhangoljuk (fine-tune) a hálót a célfeladaton!



Feladat: bináris
klasszifikáció


Transfer learning - példa

Transfer learning (előtanítás - finomhangolás):


1. Keressünk egy jóval nagyobb adatbázist, ami valamelyest hasonlít a céladatbázisunkra!
2. Tanítsunk be egy igazán mély neuronhálót a nagy adatbázison!
3. Dobjuk el az **előtanított (pretrained)** háló utolsó rétegeit!
4. Adjunk a csonka előtanított háléhoz új, a célfeladathoz igazodó rétegeket!
5. **Finomhangoljuk (fine-tune)** a hálót a célfeladaton!

Transfer learning - példa

**Sok millió tanítandó paraméter,
nagy adatbázis szükséges (100k+
mintaelem), hosszú betanítás**



Transfer learning (előtanítás - finomhangolás):

1. Keressünk egy jóval nagyobb adatbázist, ami valamelyest hasonlít a céladatbázisunkra!
 2. Tanítsunk be egy igazán mély neuronhálót a nagy adatbázison!
 3. Dobjuk el az **előtanított (pretrained)** háló utolsó rétegeit!
 4. Adjunk a csonka előtanított háléhoz új, a célfeladathoz igazodó rétegeket!
 5. **Finomhangoljuk (fine-tune)** a hálót a célfeladaton!
- 

**Néhány ezer tanítandó paraméter, kis adatbázis elégséges
(néhány száz, vagy ezer mintaelem), gyors betanítás**

Transfer learning - példa

~~Sok millió tanítandó paraméter,
nagy adatbázis szükséges (100k+
mintaelem), hosszú betanítás~~

Transfer learning (előtanítás - finomhangolás):

1. **Keressünk egy, a célfeladatunkhoz közel álló feladaton/adatbázison előtanított mélyhálót!**
2. Dobjuk el az **előtanított (pretrained)** háló utolsó rétegeit!
3. Adjunk a csonka előtanított hálózathoz új, a célfeladathoz igazodó rétegeket!
4. **Finomhangoljuk (fine-tune) a hálót a célfeladaton!**

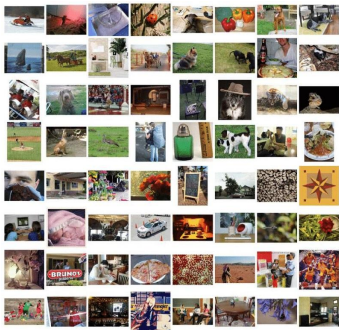
**Néhány ezer tanítandó paraméter, kis adatbázis elégséges
(néhány száz, vagy ezer mintaelem), gyors betanítás**



Transfer learning - példa

Nem minden célfeladathoz könnyű előtanító adatbázist, vagy előtanított modellt találni...

Előtanítás



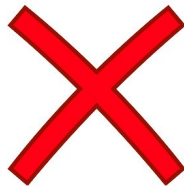
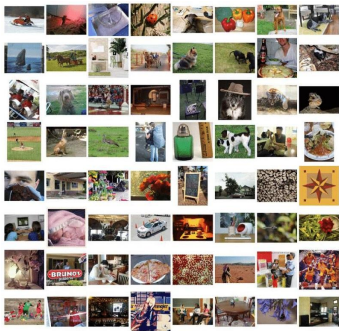
Célfeladat



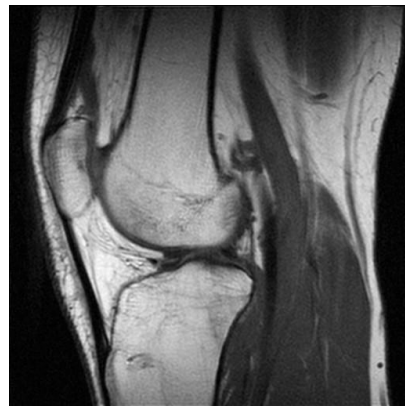
Transfer learning - példa

Nem minden célfeladathoz könnyű előtanító adatbázist, vagy előtanított modellt találni...

Előtanítás



Célfeladat



keras.models.Model: Sequential vs. Functional API

keras.models.Sequential:
listaszerű defíció

```
model = Sequential()  
model.add(Dense(h, activation='relu', input_dim=n))  
model.add(Dense(1, activation='sigmoid'))  
model.compile(loss='binary_crossentropy', optimizer=sgd)
```

```
inp_t = Input(shape=(n,))  
t = Dense(h, activation='relu')(inp_t)  
t = Dense(1, activation='sigmoid')(t)  
model = Model(inputs=[inp_t], outputs=[t])  
model.compile(loss='binary_crossentropy', optimizer='sgd')
```

keras.models.Model (functional API):
gráfszerű defíció

keras.models.Model: Functional API

```
inp_t = Input(shape=(n,))  
t = Dense(h, activation='relu')(inp_t)  
t = Dense(1, activation='sigmoid')(t)  
model = Model(inputs=[inp_t], outputs=[t])  
model.compile(loss='binary_crossentropy', optimizer='sgd')
```

Első réteg input tenzor

Első réteg output tenzor
== második réteg input
tenzor

inp_t, t: Tensorflow
Tensor típusú változók

Példa: heatmap vizualizáció

```
inp_t = Input(shape=(32, 32, 1))
t1 = Conv2D(6, (5, 5), activation="tanh")(inp_t)
t = AveragePooling2D(pool_size=(2, 2))(t1)
...
t = Flatten()(t)
t2 = Dense(10, activation='softmax')(t)
model1 = Model(inputs=[inp_t], outputs=[t2])
model2 = Model(inputs=[inp_t], outputs=[t2, t1])
model1.compile(loss='categorical_crossentropy', optimizer='sgd')
model2.compile(loss='categorical_crossentropy', optimizer='sgd')
```

Loss az utolsó layer
kimenetén számolva

Betranítás: model1.fit()

Példa: heatmap vizualizáció

```
inp_t = Input(shape=(32, 32, 1))
t1 = Conv2D(6, (5, 5), activation="tanh")(inp_t)
t = AveragePooling2D(pool_size=(2, 2))(t1)
...
t = Flatten()(t)
t2 = Dense(10, activation='softmax')(t)
model1 = Model(inputs=[inp_t], outputs=[t2])
model2 = Model(inputs=[inp_t], outputs=[t2, t1])
model1.compile(loss='categorical_crossentropy', optimizer='sgd')
model2.compile(loss='categorical_crossentropy', optimizer='sgd')
```

**Heatmap-ek visszaadása
a kimeneten: erre viszont
nem tanítunk...**

Predikció: model2.predict()

Példa: pretraining

```
inp_t = Input(shape=(32, 32, 1))
t = Conv2D(6, (5, 5), activation="tanh")(inp_t)
t = AveragePooling2D(pool_size=(2, 2))(t)
...
t1 = Flatten()(t)
t = Dense(10, activation='softmax')(t1)
model_pre = Model(inputs=[inp_t], outputs=[t])
model_pre.compile(loss='categorical_crossentropy', optimizer='sgd')
```

Betanítás az előtanító feladaton: model_pre.fit()

Példa: pretraining

```
inp_t = Input(shape=(32, 32, 1))
t = Conv2D(6, (5, 5), activation="tanh")(inp_t)
t = AveragePooling2D(pool_size=(2, 2))(t)
...
t1 = Flatten()(t)
t = Dense(10, activation='softmax')(t1)
model_pre = Model(inputs=[inp_t], outputs=[t])
model_pre.compile(loss='categorical_crossentropy', optimizer='sgd')
```

ide kötjük be az új rétegeket a finomhangoláshoz

Betanítás az előtanító feladaton: `model_pre.fit()`

Példa: fine-tuning

```
t = Dense(1, activation='sigmoid')(t1)
model_ft = Model(inputs=[model_pre.input], outputs=[t])
model_ft.compile(loss='binary_crossentropy', optimizer='sgd')
```

Az előtanított modell utolsó
felhasznált rétegének kimeneti
tenzora



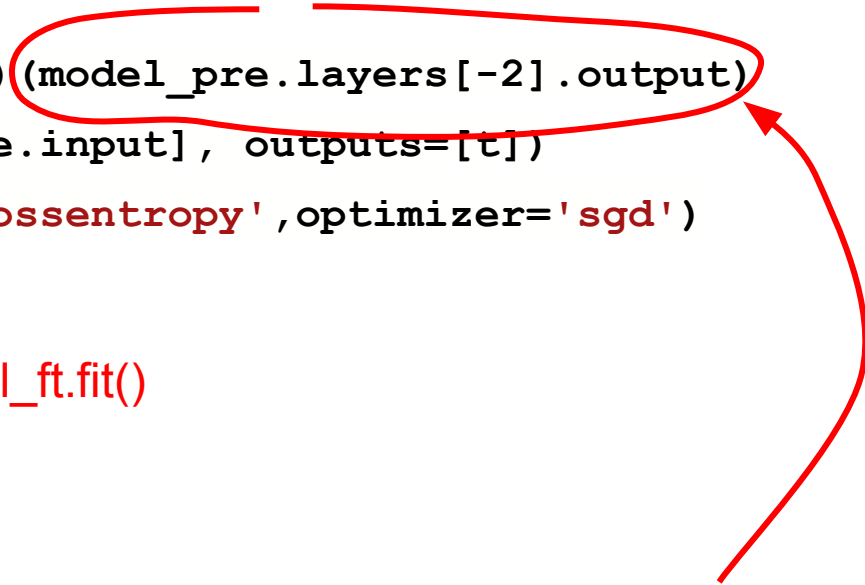
Finomhangolás a célfeladaton: `model_ft.fit()`

Az input marad az eddigi...



Példa: fine-tuning

```
t = Dense(1, activation='sigmoid')(model_pre.layers[-2].output)
model_ft = Model(inputs=[model_pre.input], outputs=[t])
model_ft.compile(loss='binary_crossentropy', optimizer='sgd')
```



Finomhangolás a célfeladaton: model_ft.fit()

Ha az előtanított modellünk (model_pre) Sequential osztállyal lett volna definiálva: ki kellene keresni a megfelelő réteget és kikérni annak az output tenzorát.

Példa: ImageNet-en tanított mélyhálók Keras-ban

```
inp_t = Input(shape=(128,128,3))
model_pre = VGG16(input_shape=(128,128,3), include_top=False,
weights='imagenet', input_tensor=inp_t)
t = Conv2D(16, (3,3), activation='relu', padding='same')
                                (model_pre.layers[9].output)

t = Flatten()(t)
t = Dense(10, activation='softmax')(t)
model_ft = Model(inputs=[inp_t], outputs=[t])
model_ft.compile(loss='categorical_crossentropy', optimizer='sgd')
```

Finomhangolás a célfeladaton: model_ft.fit()

Példa: ImageNet-en tanított mélyhálók Keras-ban

```
inp_t = Input(shape=(128,128,3))
model_pre = VGG16(input_shape=(128,128,3), include_top=False,
weights='imagenet', input_tensor=inp_t)
t = Conv2D(16, (3,3), activation='relu', padding='same')
                                (model_pre.layers[9].output)

t = Flatten()(t)
t = Dense(10, activation='softmax')(t)
model_ft = Model(inputs=[inp_t], outputs=[t])
model_ft.compile(loss='categorical_crossentropy', optimizer='sgd')
```

Finomhangolás a célfeladaton: model_ft.fit()

Transfer learning - weight freezing

Előtanítás után befagyaszthatjuk az előtanított modell összes súlyát, vagy egy részüket.

- Ez csökkenti a valószínűségét, hogy túltanuljunk a célfeladaton, hiszen csökken a tanítható paraméterek száma...

Transfer learning - weight freezing

Előtanítás után befagyaszthatjuk az előtanított modell összes súlyát, vagy egy részüket.

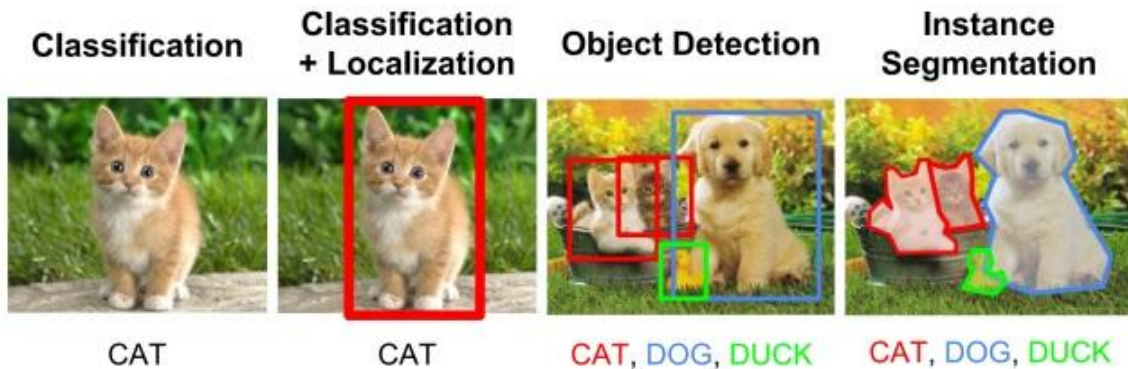
- Ez csökkenti a valószínűségét, hogy túltanuljunk a célfeladaton, hiszen csökken a tanítható paraméterek száma...

PI:

```
for layer in model_pre.layers[:6]:  
    layer.trainable = False
```

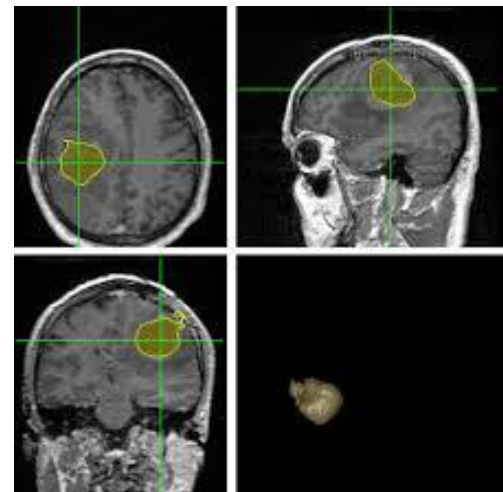
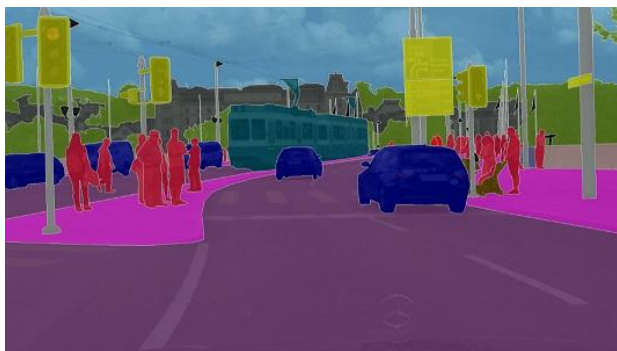
További feladatok a képfeldolgozásban

A címke a különböző feladatoktól függően, például egy kép is lehet.



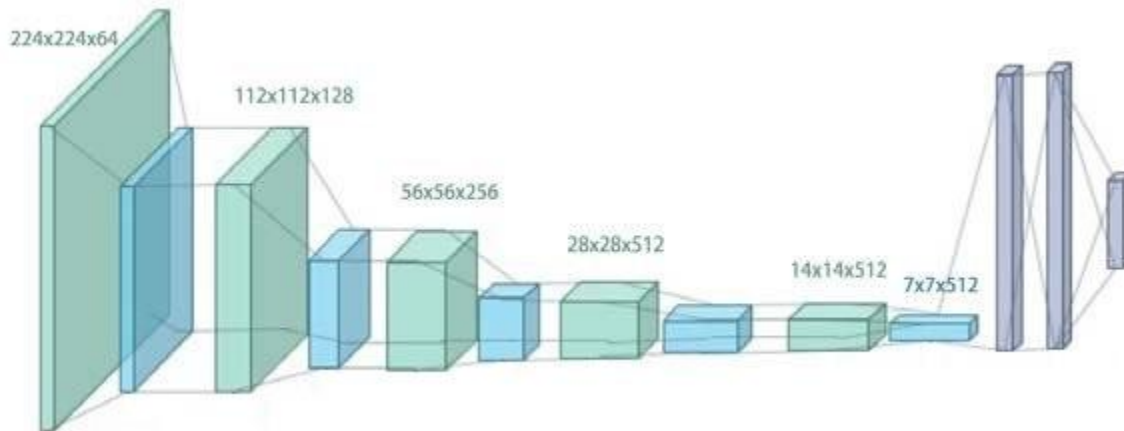
Képszegmentálás - alkalmazások

- **ADAS** (Advanced driver-assistance systems)
- **Térképészet, földmérés**
- **Orvosi képalkotás**



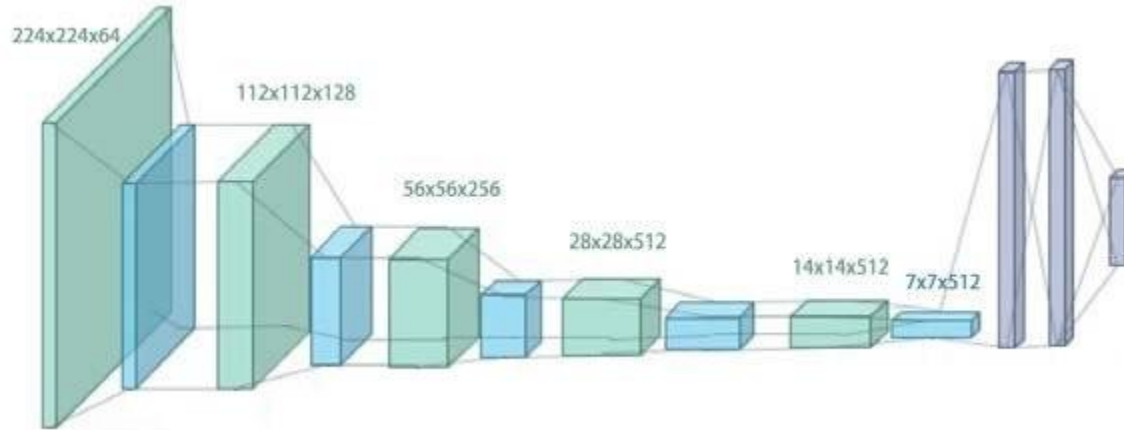
Képszegmentálás konvolúciós hálóval

Alkalmas-e egy hagyományos konvolúciós háló képszegmentálásra?



Képszegmentálás konvolúciós hálóval

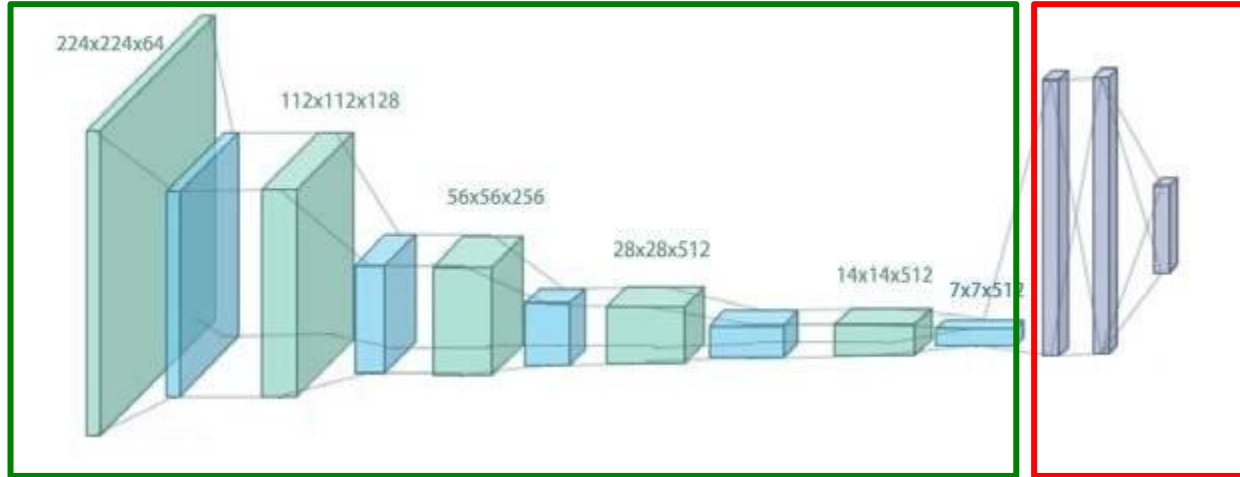
Alkalmas-e egy hagyományos konvolúciós háló képszegmentálásra?



A címke itt egy 2 dimenziós (1, vagy többcsatornás) kép kell, hogy legyen. Bár az utolsó, teljesen összekötött rétegeket átalakíthatjuk ilyen formájúra, viszont **elveszítjük az eltolás invarianciát!**

Képszegmentálás konvolúciós hálóval

Alkalmas-e egy hagyományos konvolúciós háló képszegmentálásra?

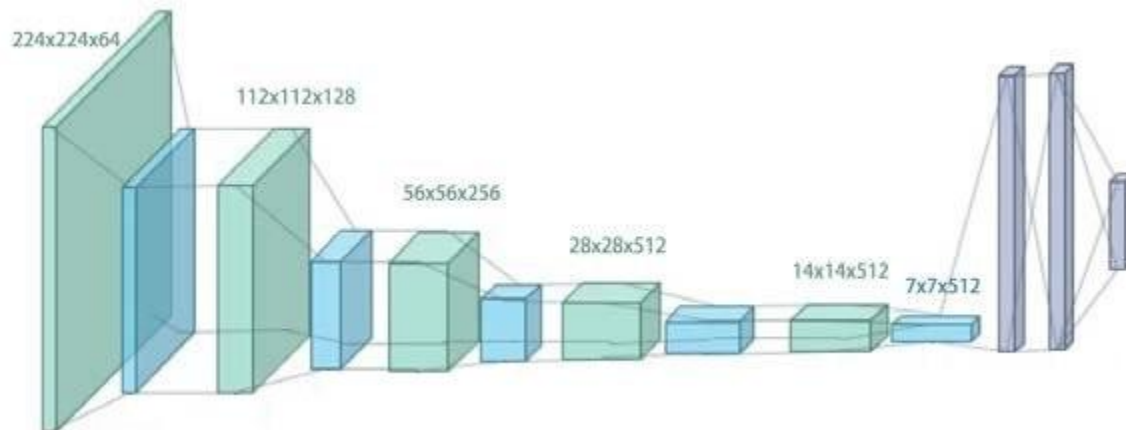


Konvolúciós és pooling rétegek:
megközelítőleg eltolás-invariáns

MLP / teljesen összekötött rétegek:
eltolás invariancia hiánya

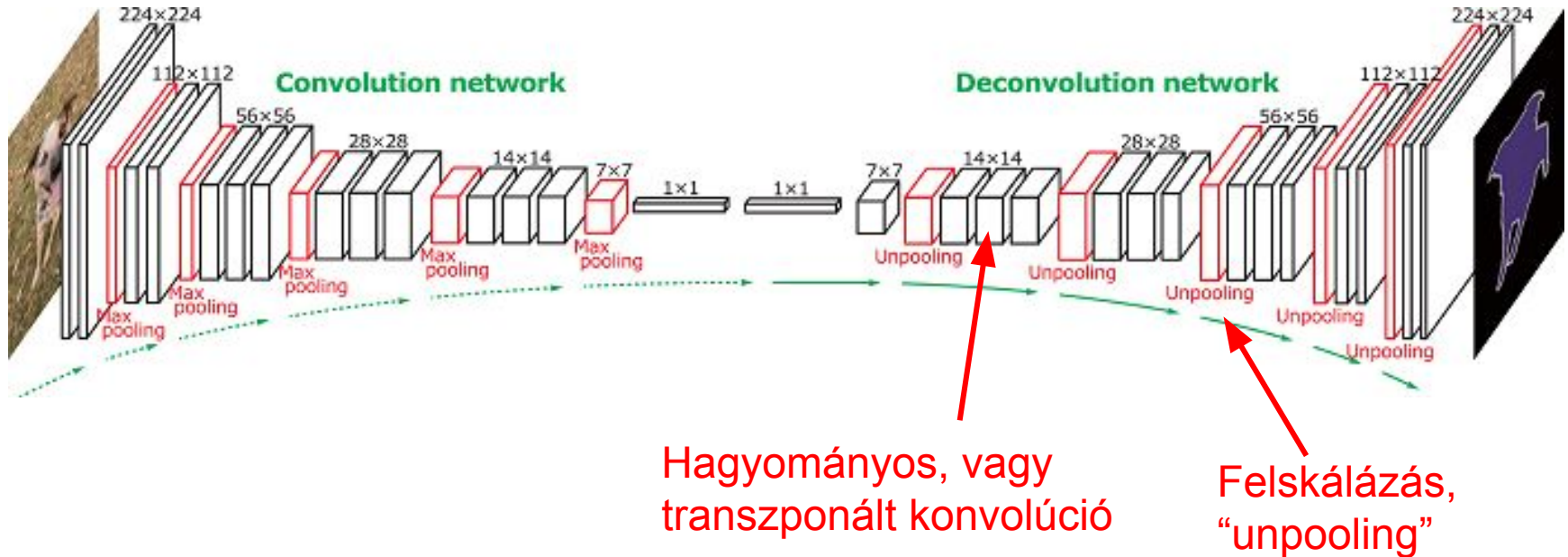
Képszegmentálás konvolúciós hálóval

Képszegmentálás során **fontos** lenne **az eltolás invariancia fenntartása a teljes hálóban**. Hiszen, ha valamit megtanulunk szegmentálni a kép egyik részén, szükséges, hogy a kép más részein is meg tudjuk azt tenni.



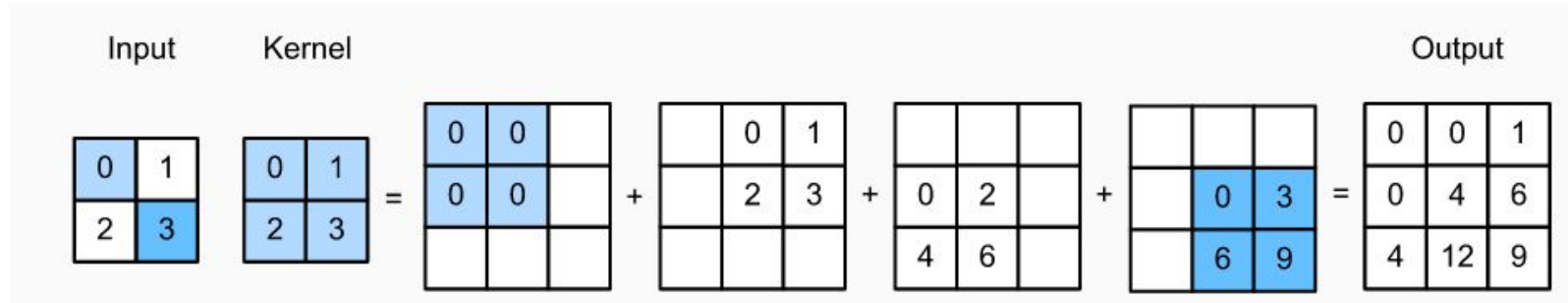
Fully-convolutional network (FCN)

Megoldás: a hálónk álljon kizárólag konvolúciós- és skálázás rétegekből!



Fully-convolutional network (FCN)

Transzponált konvolúció (transposed convolution, “deconvolution”):



Fully-convolutional network (FCN)

Felskálázás (unpooling, upscaling):

