Seminár Robotika.SK

# Hlboké učenie s Keras a OpenCV3

***Andrej Lúčny***

***Katedra aplikovanej informatiky FMFI UK***

***lucny@fmph.uniba.sk***

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

- Front-end pre Deep learning nad backendom TensorFlow (alternatívne: Theano) v Python-e
- Deep learning – Alex Krizhevsky, 2012
- Keras – Francois Chollet, projekt ONEIROS, 2015
- MIT Licencia
- GPU - CUDA, numphy, OpenCV, TensorFlow, Python

`www.robotika.sk/seminar/2018/cviko6-keras.zip`

# Úvodný príklad

Chceme na obraze určiť zvislé hrany

a

b

Môžeme na to použiť:

# Sobelov operátor

| | | |
|---|---|---|
| $a_{i-1,j-1}$ | $a_{i-1,j}$ | $a_{i-1,j+1}$ |
| $a_{i,j-1}$ | $a_{i,j}$ | $a_{i,j+1}$ |
| $a_{i+1,j-1}$ | $a_{i+1,j}$ | $a_{i+1,j+1}$ |

**o**

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

=

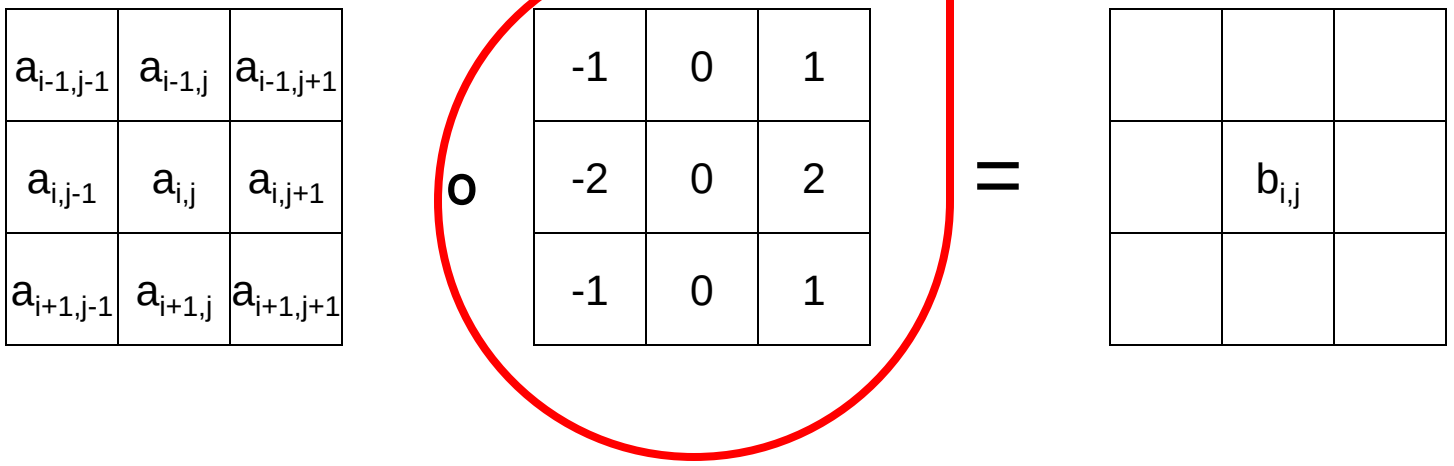
| | | |
|---|---|---|
| | | |
| | $b_{i,j}$ | |
| | | |

$$b_{i,j} = | a_{i-1,j+1} + 2a_{i,j+1} + a_{i+1,j+1} - a_{i-1,j-1} - 2a_{i,j-1} - a_{i+1,j-1} |$$

Môžeme na to použiť:

kernel 3x3

# Sobelov operátor




$$b_{i,j} = | a_{i-1,j+1} + 2a_{i,j+1} + a_{i+1,j+1} - a_{i-1,j-1} - 2a_{i,j-1} - a_{i+1,j-1} |$$

# Klasická implementácia v OpenCV

```
import cv2

kernel = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])

output_image = cv2.filter2D(input_image,-1,kernel)
```

***ker1.py***

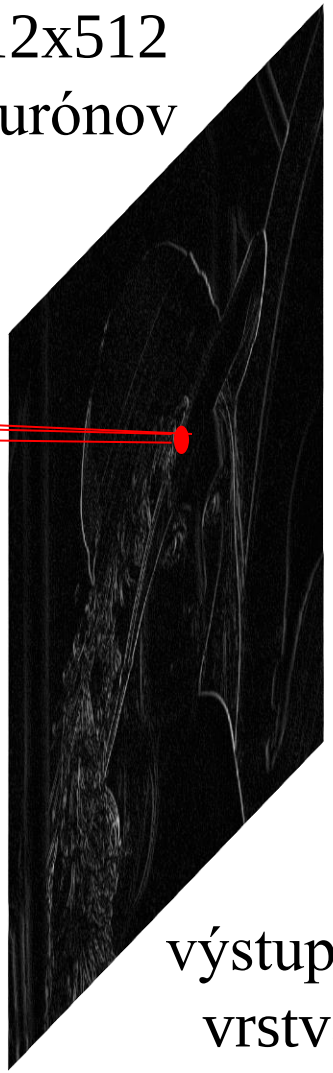
# Konvolučná neurónová sieť

512x512
neurónov

vstupná
vrstva

kernel 3x3

512x512
neurónov



výstupná
vrstva

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

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

# Implementácia v Keras-e

```
import keras

inp = Input(shape=(None,None,1))
out = Conv2D(1, (3, 3), kernel_initializer="normal",
       use_bias=False, padding="same")(inp)
model = Model(inputs=inp, outputs=out)

w = np.array([[ [[-1]],[[0]],[[1]]], [[-2]],[[0]],[[2]]],
       [[[-1]],[[0]],[[1]]] ])
model.layers[1].set_weights(w)

input_images = np.array([input_image.reshape(...)])
output_images = model.predict(input_images)
output_image = output_images[0].reshape(...)
```

***ker2.py***

# V čom je rozdiel?

Predstavme si, že som zabudol aký kernel Sobelov operátor používa, mám len želaný výstupný obrázok.

Skúsim do kernelu zadať náhodné hodnoty a vyrátam čo dá. Dá sa usúdiť z takto vyrátaného a želaného výstupného obrázku, ako by som mal zvolený kernel opraviť? Opraviť tak, aby som dostal menšiu chybu?

Dá sa potom opakovať oprava, až kým nezrekonštruujeme „zabudnutý“ kernel?

Dá. A Keras to vie urobiť.

# Skúsme „zabudnutý“ kernel rekonštruovať

```
input_images = np.array([input_image.reshape(...)])
output_images = np.array([output_image.reshape(...)])

inp = Input(shape=(None,None,1))
out = Conv2D(1, (3, 3), kernel_initializer="normal",
       use_bias=False, padding="same")(inp)
model = Model(inputs=inp, outputs=out)

model.compile(optimizer='rmsprop', loss='mse',
       metrics=['mse', 'mae'])
model.fit(samples_inp, samples_out, batch_size=1,
       epochs = 15000)
model.layers[1].get_weights()
```

***ker3.py***

# Funguje to!

```
>>> model.layers[1].get_weights()

[array([[[[ -1.00157452e+00]],
         [[ -2.61067878e-03]],
         [[  1.00584388e+00]]],


        [[[ -1.99438238e+00]],
         [[  3.66915925e-03]],
         [[  1.99175704e+00]]],


        [[[ -1.00293767e+00]],
         [[ -4.22276789e-05]],
         [[  1.00477922e+00]]]],    dtype=float32)]
>>>
```

***ker3.py***

# Funguje to!

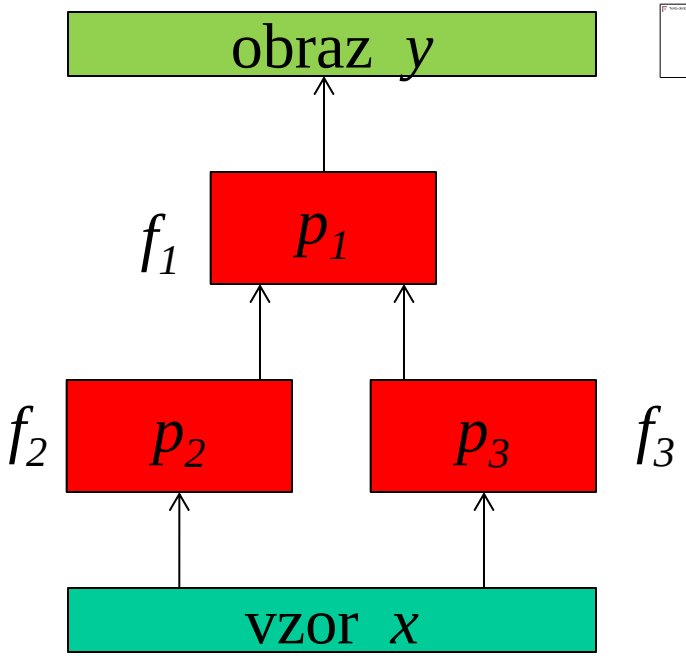
```
>>> model.layers[1].get_weights()

[array([[[[ -1.00157452e+00]],   -1
         [[ -2.61067878e-03]],    0
         [[  1.00584388e+00]]],   1

        [[[ -1.99438238e+00]],   -2
         [[  3.66915925e-03]],    0
         [[  1.99175704e+00]]],   2

        [[[ -1.00293767e+00]],   -1
         [[ -4.22276789e-05]],    0
         [[  1.00477922e+00]]]], 1  dtype=float32)]
>>>
```

***ker3.py***

A prečo ?

# Čo je vlastne Keras?

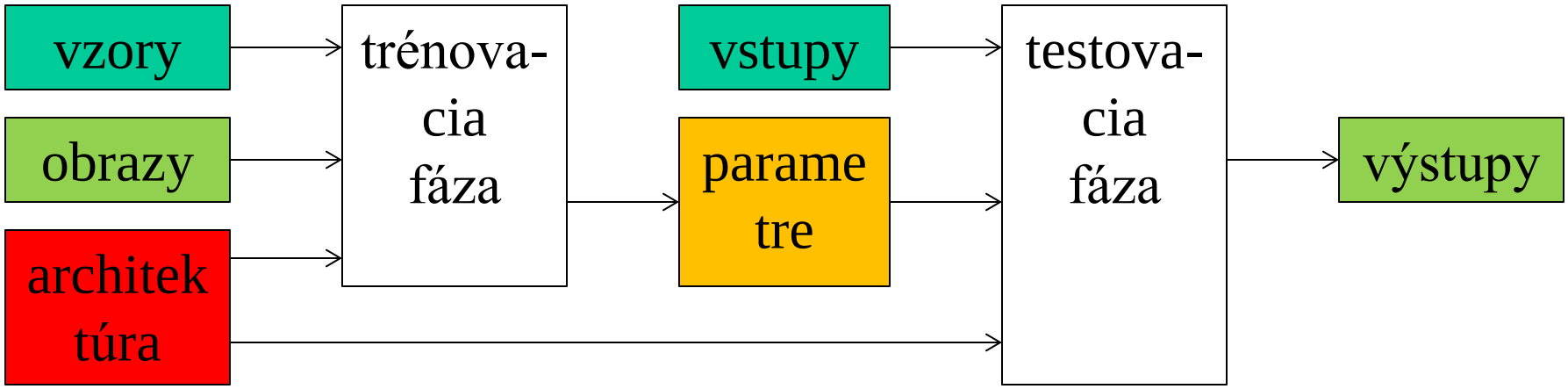
- Keras je tool, ktorý umožní výpočet poskladať z takých modulov (konvolučná neurónová sieť je jedným z nich), že ich výpočtovú činnosť vie vyjadriť funkciou, ktorú vie symbolicky derivovať.




$$\frac{\partial f_1(p_1, f_2(p_2, x), f_3(p_3, x))}{\partial p_1}$$

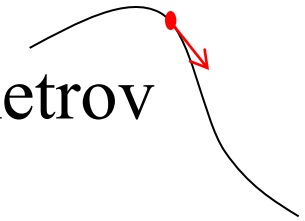
# Čo je vlastne Keras?

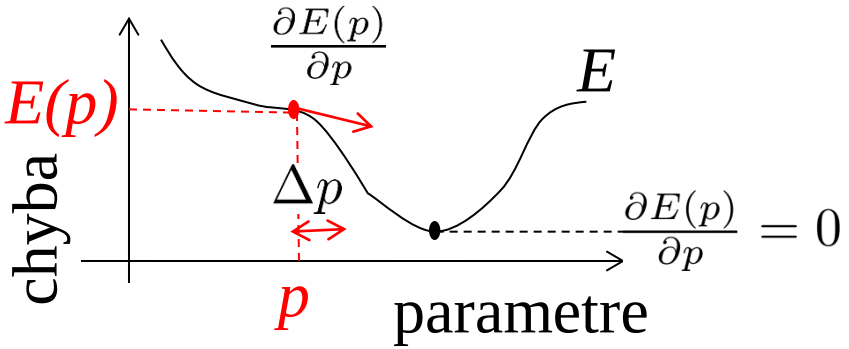
- Vďaka tomu dokáže realizovať proces učenia s učiteľom, t.j. ak dostane sadu vzorov, sadu želaných obrazov a architektúru výpočtu s množstvom neznámych parametrov, tak po množstve iterácii vypočíta také hodnoty týchto parametrov, že obrazy vypočítané zo vzorov sa čo najmenej líšia od obrazov želaných.



# Ako to robí?

- Keras sa opiera o zadefinovanú chybovú funkciu, spravidla sumu druhých mocnín rozdielov obrazov a želaných obrazov.
- Chybovú funkciu vníma ako funkciu parametrov architektúry výpočtu a vyráta jej deriváciu (gradient), t.j. ako zmeniť každý jeden parameter, aby sa hodnota chybovej funkcie zmenšila.
- Keďže používa iba také moduly, ktorých činnosť možno popísať funkciou, ktorú vieme symbolicky zderivovať, tento gradient vyráta dosadením do správnych vzorcov.






- vzory $x_i$
- želané obrazy $y_i$
- obrazy pri daných parametroch $f(p, x_i)$

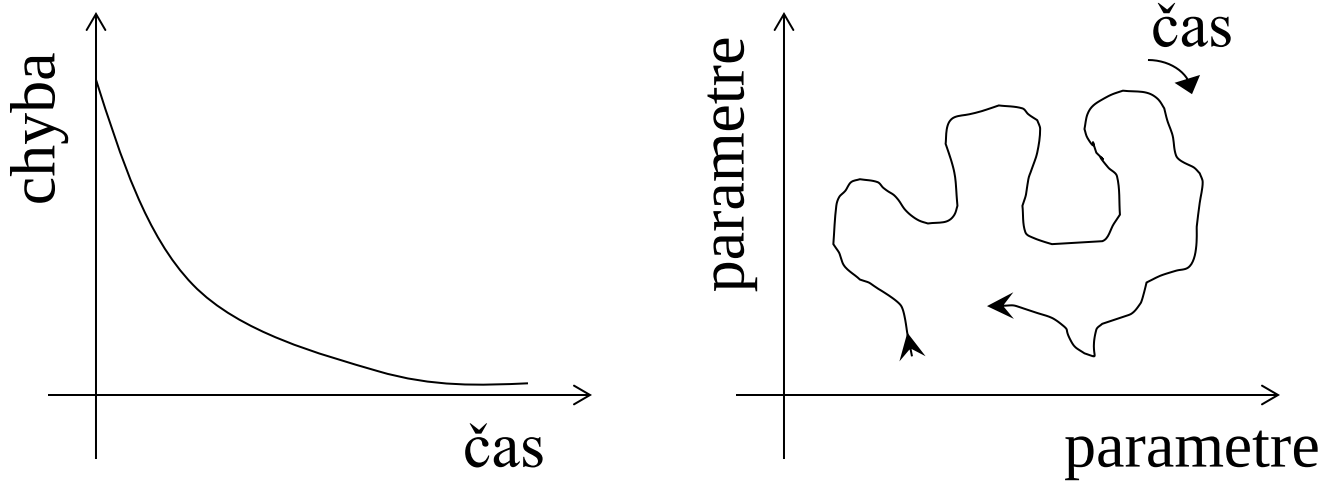
- chybová funkcia: $E(p) = \sum (f(p, x_i) - y_i)^2$

- želáme si, aby bola minimálna: $\frac{\partial E(p)}{\partial p} = (\nabla E)(p) = 0$
  t.j. aby bol gradient nulový
- úprava parametrov
  ktorá zmenší chybu

$$\Delta p \propto \frac{\partial E(p)}{\partial p}$$

# Ako to robí?

- Keras potom iteratívne (cyklicky) upravuje parametre architektúry, až kým nedostane minimálnu chybu.




- Tento proces má mnoho záludností, takže Keras ponúka množstvo možností ako ho presne realizovať

# Ako to robí?

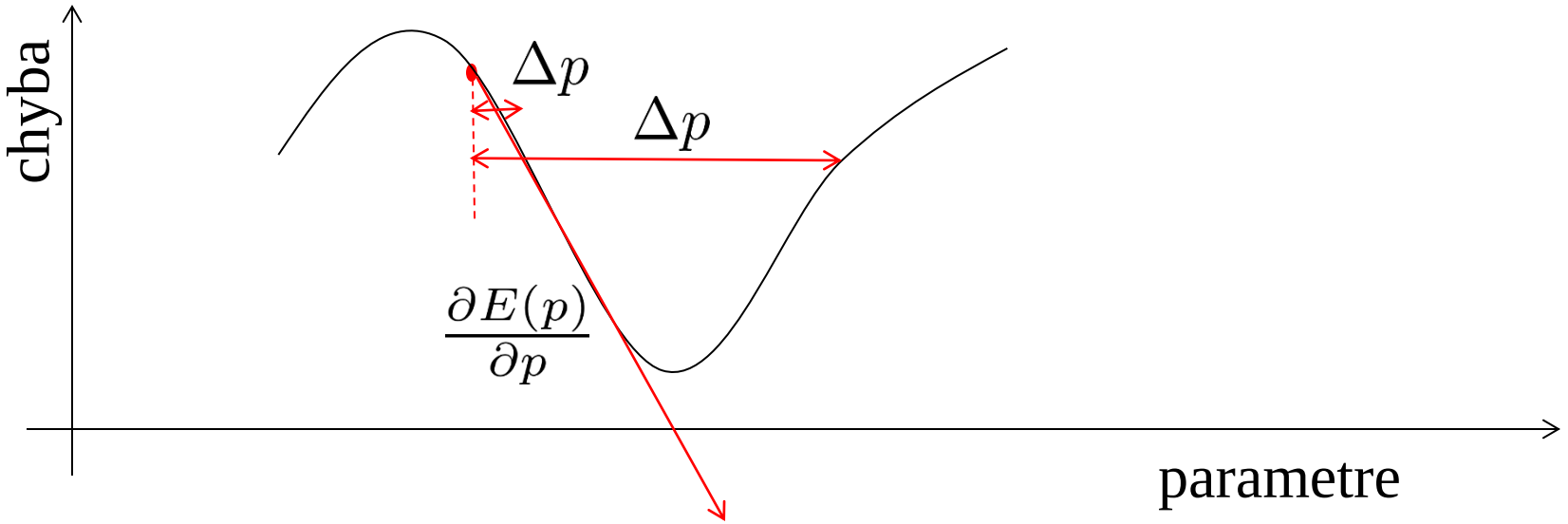
Algoritmus (optimizer)

1. vezmi náhodné parametre p
2. Vypočítaj zmenu parametrov $\Delta p$
3. Uprav $p = p + \Delta p$
4. Ak je E(p) dostatočne malá a p dostatočne dobré skonči, inak pokračuj na 2.

# Záludnosti

Presnosť:

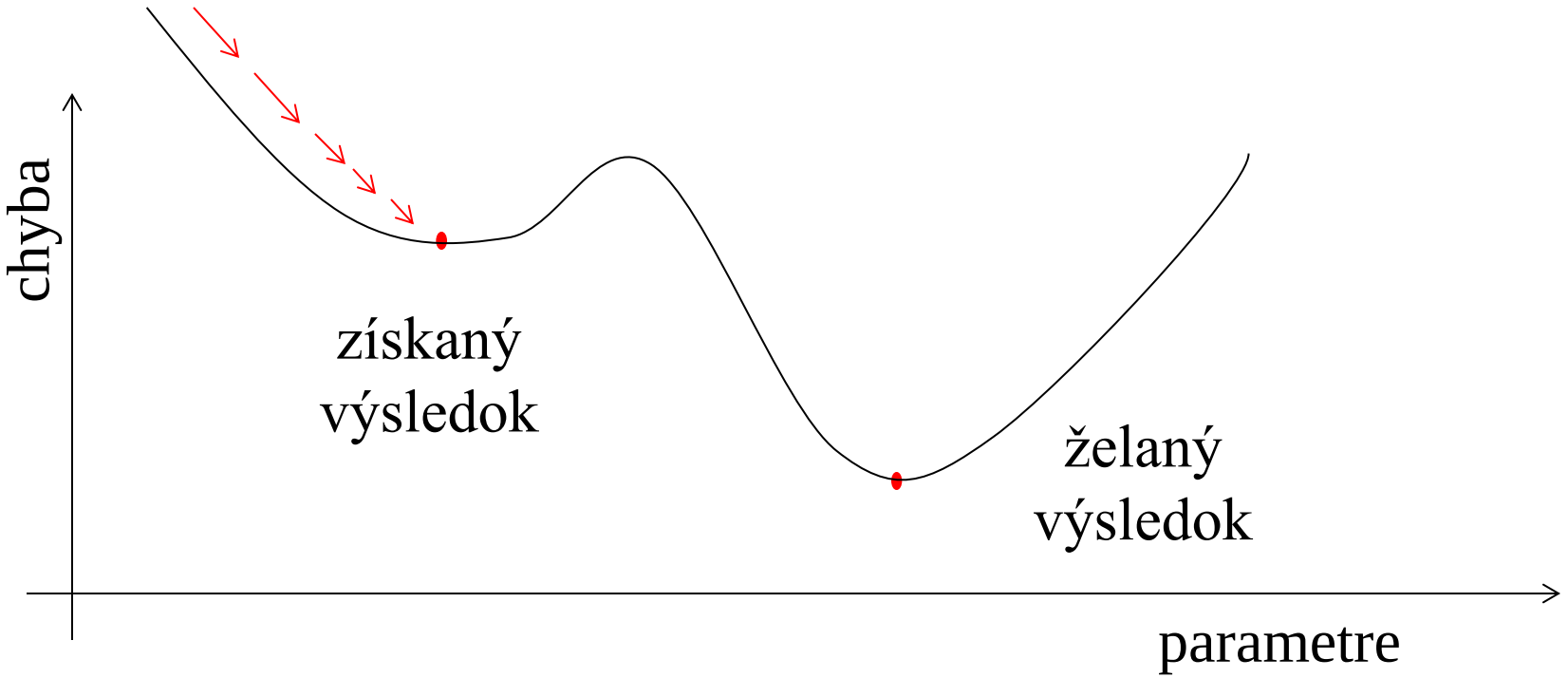
- Vieme správny smer, ale nevieme správnu konštantu úmery medzi $\Delta p$ a $\frac{\partial E(p)}{\partial p}$
- Príliš malá → pomalé učenie
- Príliš veľká → priveľká chyba



# Záludnosti

Uviaznutie v lokálnom minime:

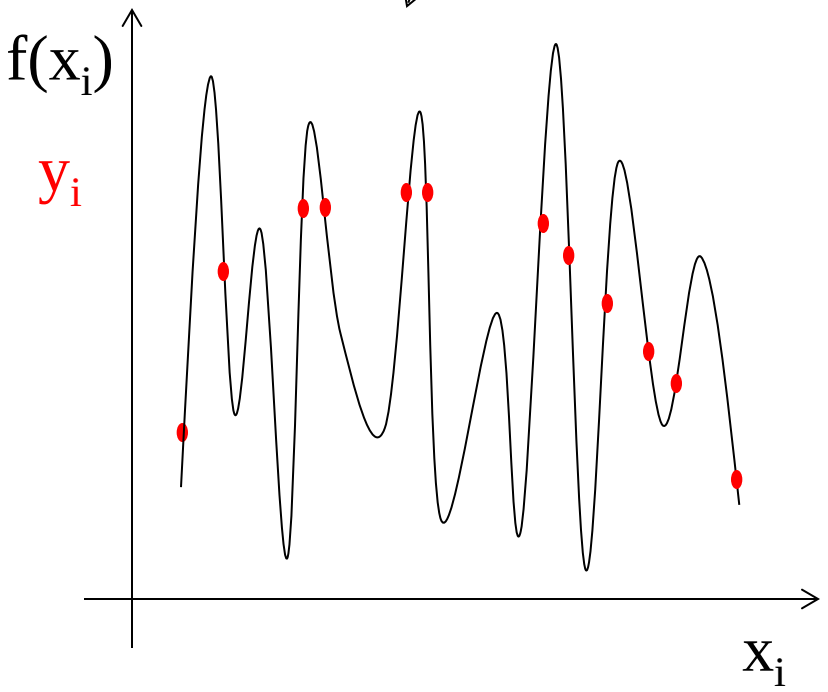
- Zmena parametrov nemôže byť len v smere gradientu

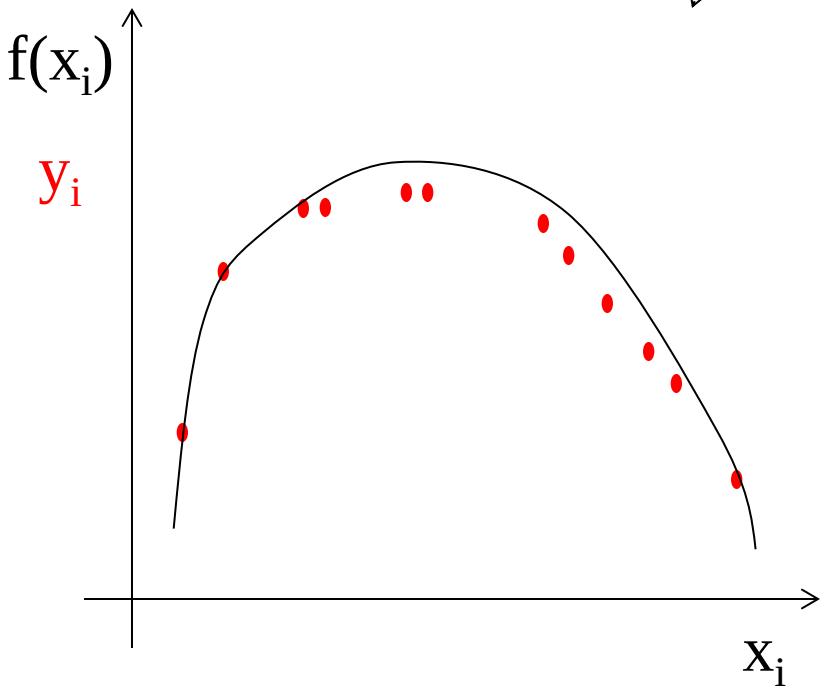


# Záludnosti

Preučenie (overfitting):

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



# Zložitejší príklad

Pri Sobelovom operátore sme vystačili s jedinou konvolučnou vrstvou. To je ale veľmi zriedkavý prípad.

Sobel

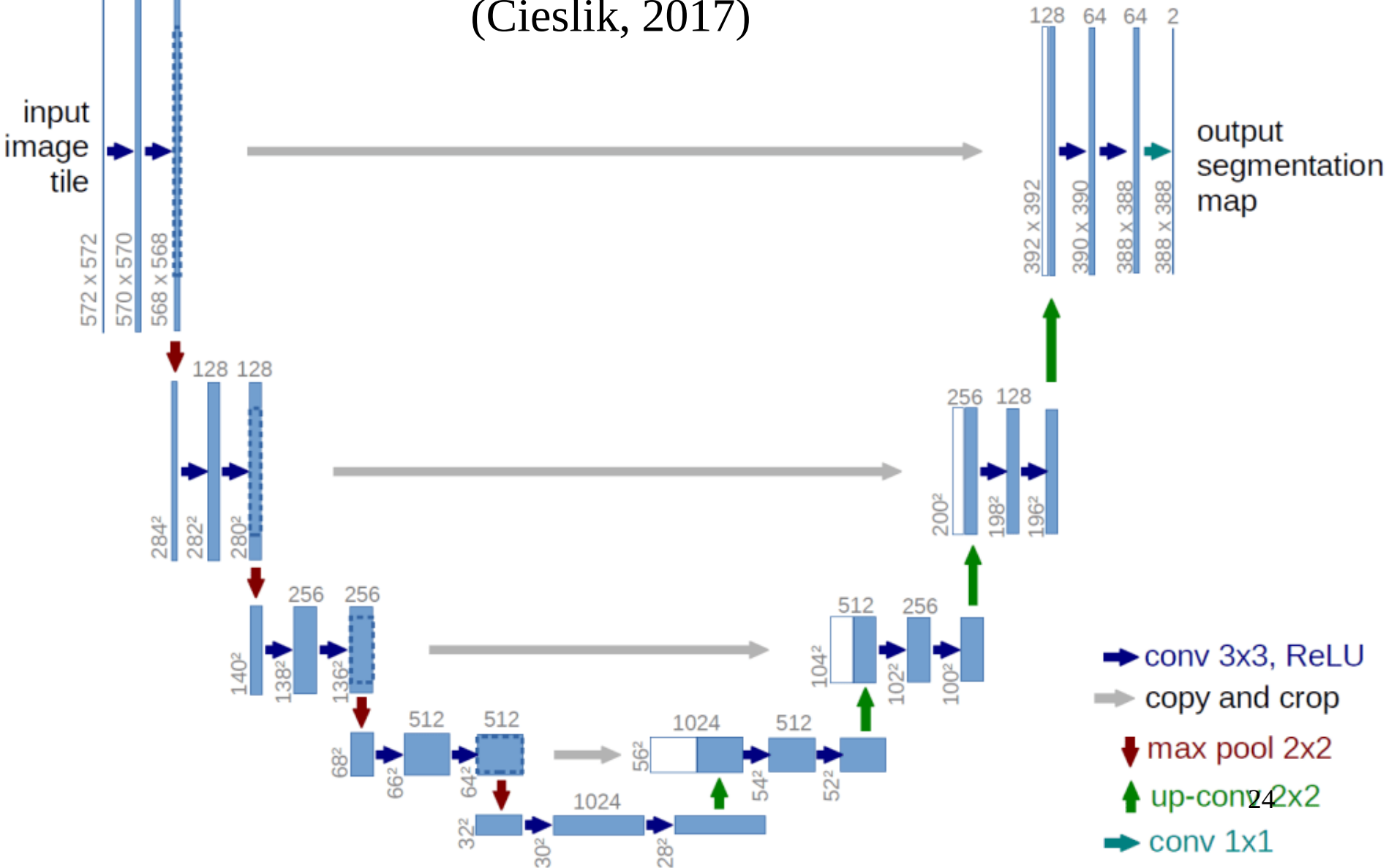
# Zložitejší príklad

Pokiaľ sa pokúsime naučiť počítač naučiť niečo zložitejšie, napríklad operátor Canny, hĺbka riešenia sa môže rádovo zvýšiť...

Canny

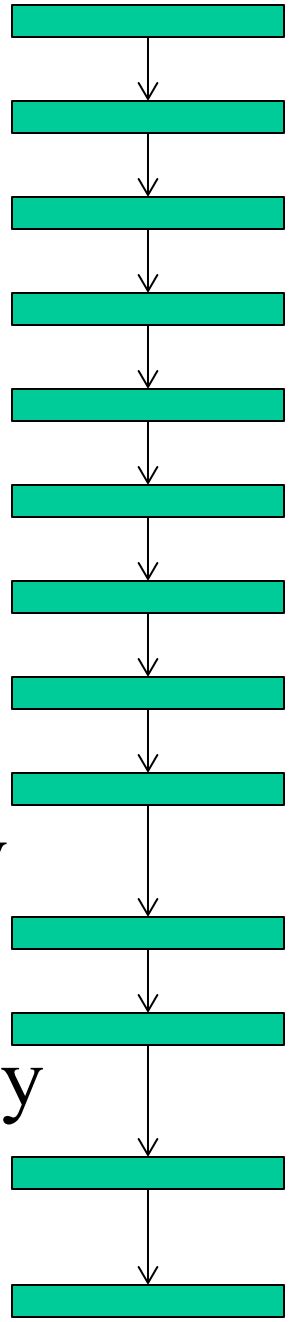
# Canny cez DL

(Cieslik, 2017)



# Deep Learning

- V takejto úlohe s jednou vrstvou nevystačíme
- Čím zložitejšia úloha tým zložitejšia architektúra, tým viac vrstiev a tým sa to učenie stáva „hlbokým“
- V minulosti sa predpokladalo, že hlboké učenie nie je použiteľné pre vysoké nároky na výkon a presnosť
- Vedelo sa tiež, že dve plne prepojené vrstvy neurónov sú univerzálnym aproximátorom
- Dnešná prax: DL už vykonať dá a oplatí sa



# Aké súčiastky Keras poskytuje?

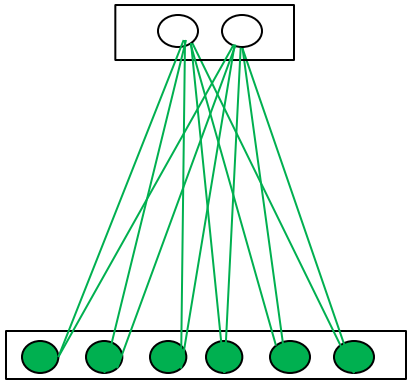
- Core Layers: Dense, Activation, Flatten, Dropout, ...
- Convolutional Layers: Conv2D, ...
- Locally-connected Layers: LocallyConnected2D, ...
- Pooling Layers: MaxPooling2D, AveragePooling2D, ...
- Merge Layers: Add, Subtract, ...
- Recurrent Layers: RNN, LSTN, ...
- Embedding Layers: Embedding
- Advanced Activation Layers: Softmax, LeakyReLU, ...
- Normalization Layers: Batch Normalization, ...
- Noise Layers: GaussianNoise, GaussianDropout, ...
- Layer Wrappers: TimeDistributed, ...

# S akými typmi dát Keras pracuje?

- Vrstvy si medzi sebou odovzdávajú **Tenzory**, čo v tomto prípade treba chápať ako viacrozmerné polia, ktorých súradnice sú určitého druhu:
  - **batch**
  - **data**
  - **channel**
- Keď máme na vstupe 10 bežných farebných obrázkov 32x32 pixelov, tak je to tenzor **10** x **32** x **32** x **3** celých čísel 0-255

# Dense Layer

- Vrstva neurónov rovnakého charakteru ako predošlá.
- Neuróny sú prepojené každý s každým
- Každý má svoje váhy a prípadne aj bias a aktivačnú funkciu



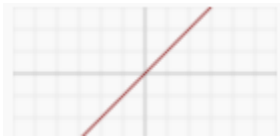
*Bias sa odpočíta od výstupnej hodnoty*
*Pri Dense layeri ho má každý neurón rôzny*

```
>>> Dense(6)
```

# Activation Layer

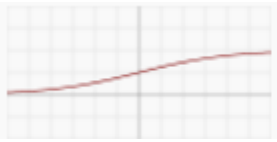
- Na každý vstup aplikuje určitú funkciu a dá výsledok na výstup

`linear`



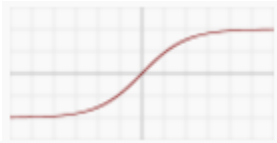
$$f(x) = x$$

`sigmoid`



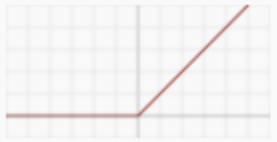
$$f(x) = \frac{1}{1 + e^{-x}}$$

`tanh`



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

`relu`



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

`softmax`

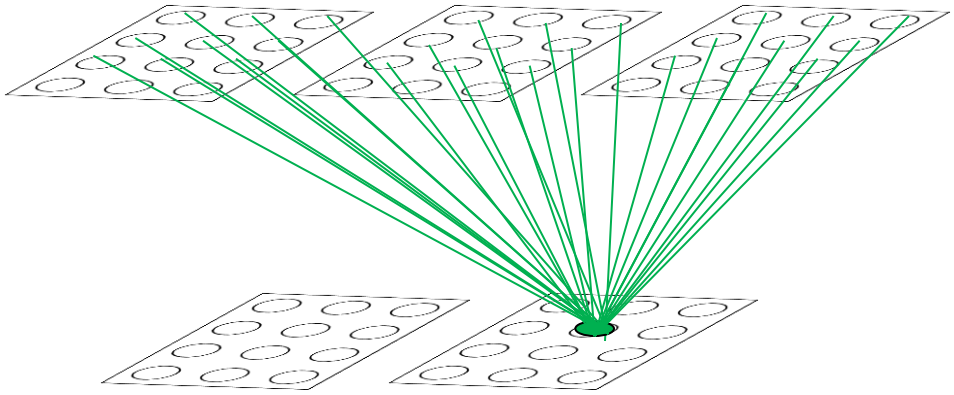

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$$

- Rovnakú funkcionalitu možno zadefinovať priamo vo väčšine ostatných vrstiev parametrom *activation*

```
>>> Activation('relu')
```

# Conv2D Layer

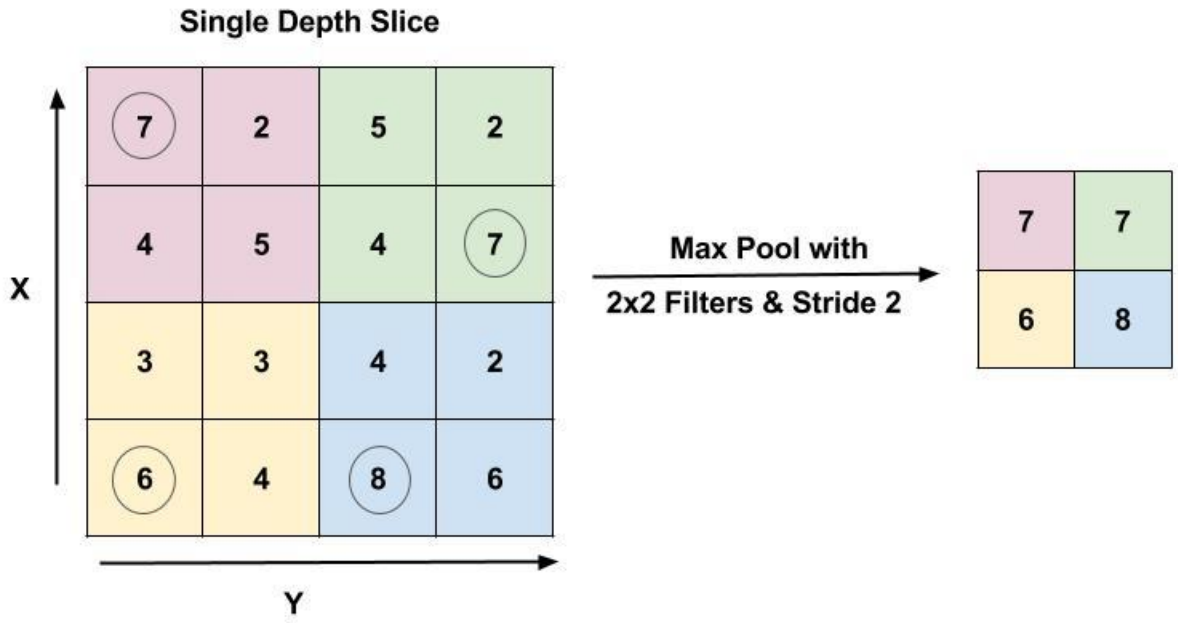
- Konvolučná vrstva obsahuje jednu alebo viacero konvolučných sietí



- Všetky neuróny konvolučnej siete sú rovnaké t.j. majú rovnaké váhy a bias
- Každý neurón je spojený do všetkých batch častí vstupu kde na svoje miesto v dátovej zložke aplikuje kernel danej veľkosti

```
>>> Conv2D(2, (3, 3))
```

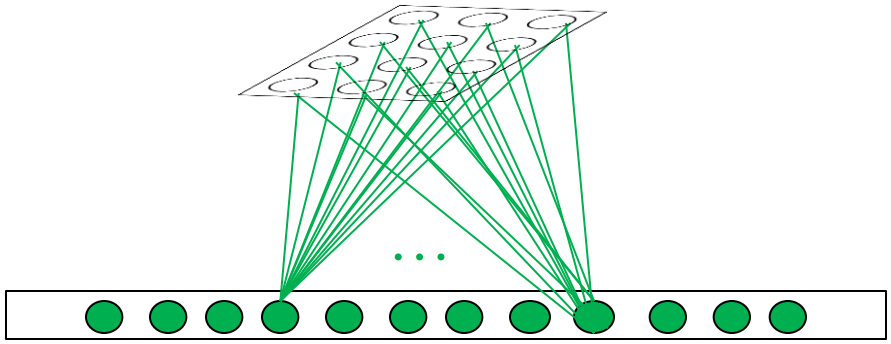
# MaxPooling2D Layer




- Redukcia dát, aplikuje sa často po Conv2D
- pool_size = (sx, sy),  strides = pool_size (krok)

```
>>> MaxPooling2D(pool_size=(2, 2))
```

# Flatten Layer



- Preusporiada tvar vstupu na 1D výstup

```
>>> Flatten()
```

# Dropout Layer

- Slúži na odolávanie preučeniu

```
>>> Dropout()
```

# Add (Merge) Layer

- Spája rozvetvujúcu sa štruktúru vrstiev

```
>>> x1 = …
>>> x2 = …
>>> x = Add([x1,x2])
```

# Embeded Layer

- Premení (spravidla vstupné) celé číslo na aktiváciu viacerých neurónov reálnym číslom od 0.0 po 1.0

```
>>> Embeded()
```

# Batch Normalization Layer

- Normalizuje vstupné dáta aby spĺňali Gaussove normálne rozdelenie s priemerom 0 a smerodajnou odchýlkou 1

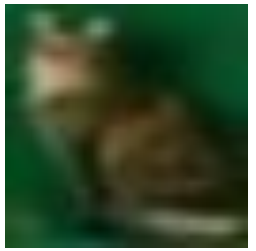
# TimeDistributed Wrapper

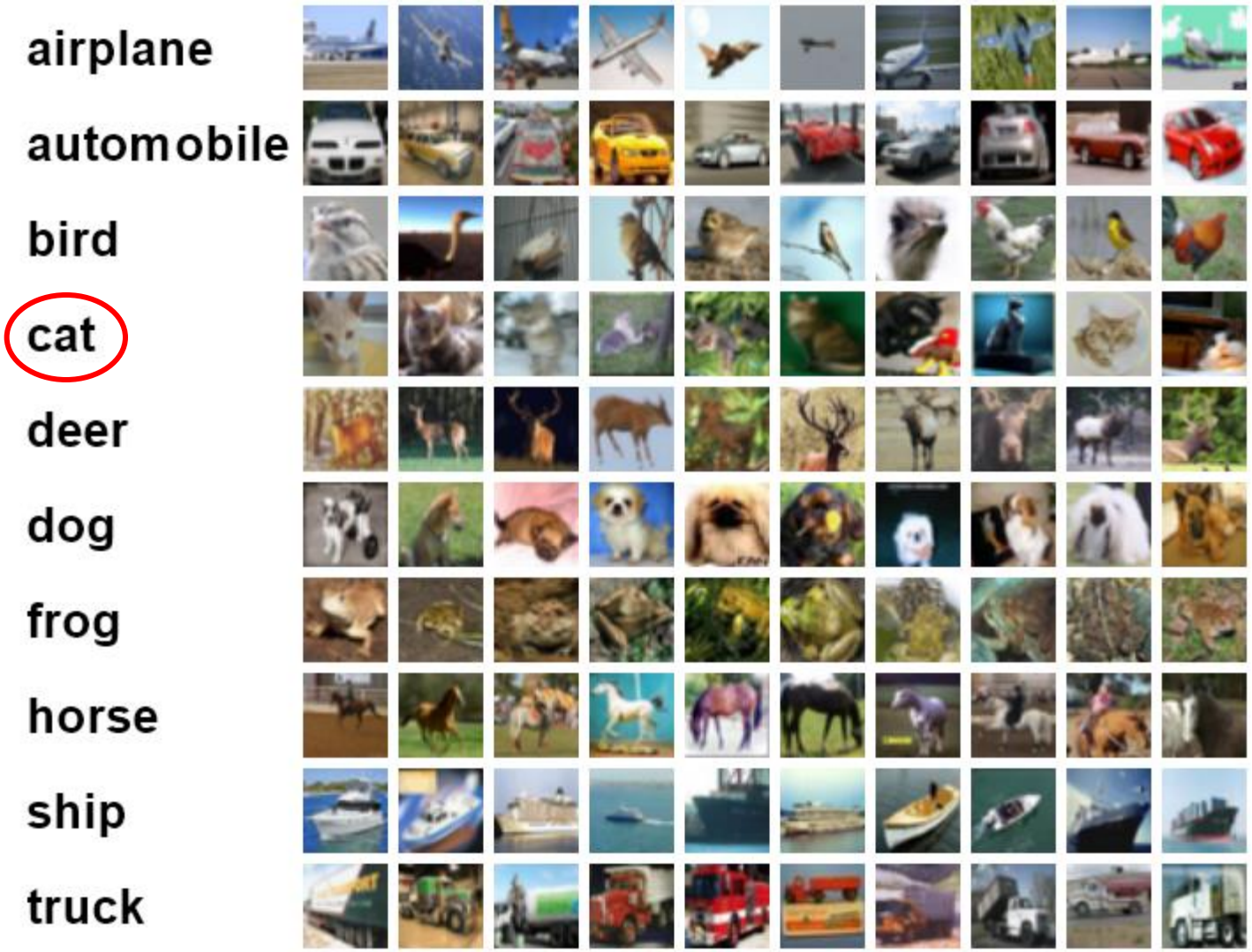
- Postupne púšťa jednotlivé batch zložky vstupu do výstupu
- Používa sa pre rekurentné neurónové siete

# Ako využiť Keras?

- Doteraz sme riešili úlohy, pri ktorých sme mali aspoň určitú predstavu o ich riešení
- Keras je však nápomocný práve vtedy, keď nemáme ani šajn, ako úlohu riešiť, napríklad: rozpoznať, či je na obrázku mačka
- Ako na to? Architektúru riešenie budeme proste hádať a budeme sa pozerať či sa nepodarí systém z dát niečo naučiť

# Ako rozpoznať mačku?






**cifar10_cnn.py** https://www.learnopencv.com/image-classification-using-convolutional-neural-networks-in-keras/ **ker5.py**

# Kde vziať dáta?

Sú voľne dostupné anotované dáta

- CIFAR10 (obrázky 10 kategórii z WordNet)

  je súčasťou keras.datasets

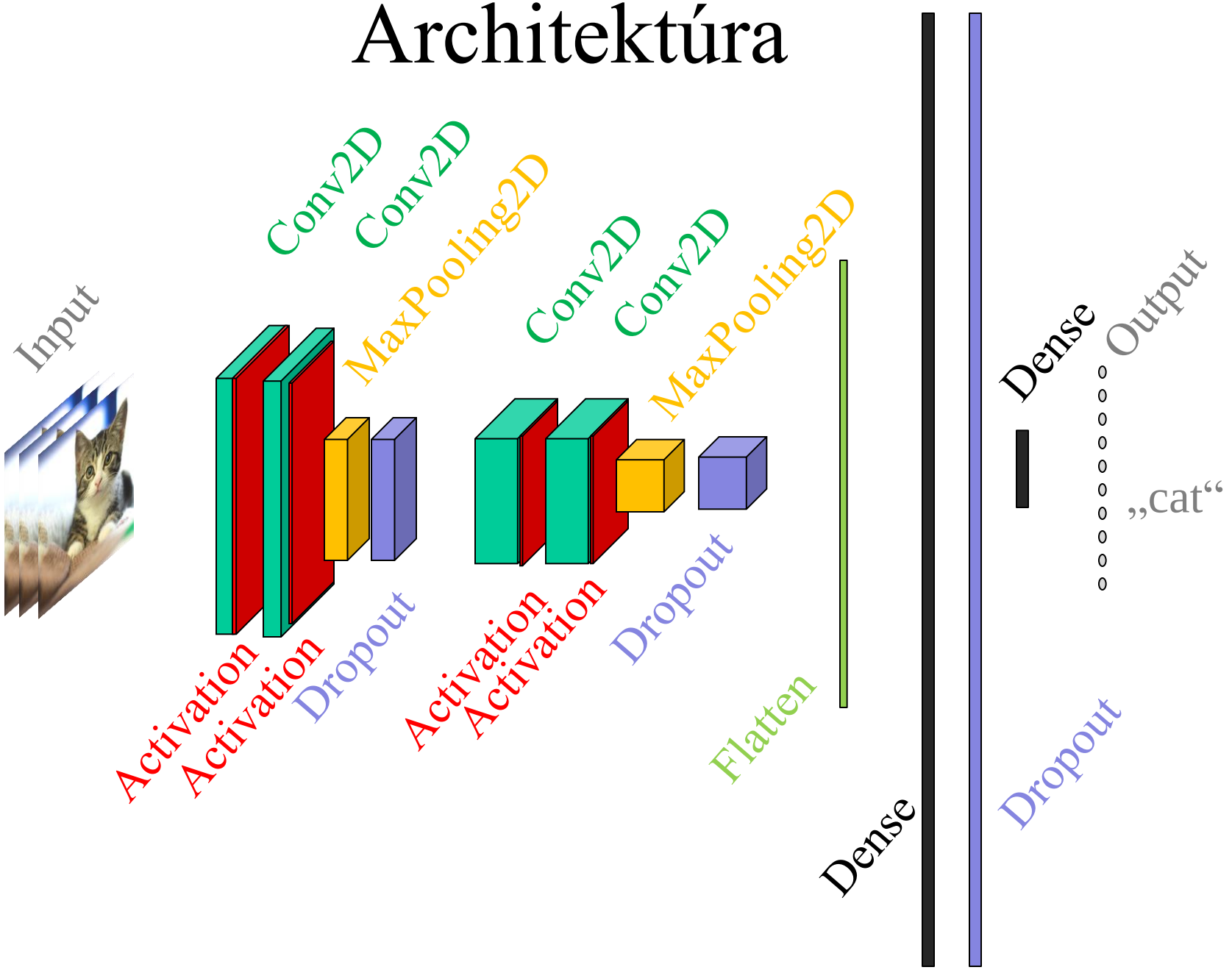
  50000 trénovacích a 10000 testovacích obrázkov

  sú to farebné obrázky 32 x 32
  10 kategórii
- potrebujeme aj negatívne príklady, systém vlastne učíme rozlišovať jedny dáta od druhých

# Architektúra

```
inp = Input(shape=x_train.shape[1:])
x = Conv2D(32, (3, 3), padding='same')(inp)
x = Activation('relu')(x)
x = Conv2D(32, (3, 3))(x)
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Dropout(0.25)(x)
x = Conv2D(64, (3, 3), padding='same')(x)
x = Activation('relu')(x)
x = Conv2D(64, (3, 3))(x)
x = Activation('relu')(x)
x = MaxPooling2D(pool_size=(2, 2))(x)
x = Dropout(0.25)(x)
x = Flatten()(x)
x = Dense(512)(x)
x = Activation('relu')(x)
x = Dropout(0.5)(x)
x = Dense(num_classes)(x)
out = Activation('softmax')(x)
model = Model(inputs=inp, outputs=out)
```

# Architektúra



| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_1 (Conv2D) | (None, 32, 32, 32) | 896 |
| conv2d_2 (Conv2D) | (None, 30, 30, 32) | 9248 |
| max_pooling2d_1 (MaxPooling2 | (None, 15, 15, 32) | 0 |
| dropout_1 (Dropout) | (None, 15, 15, 32) | 0 |
| conv2d_3 (Conv2D) | (None, 15, 15, 64) | 18496 |
| conv2d_4 (Conv2D) | (None, 13, 13, 64) | 36928 |
| max_pooling2d_2 (MaxPooling2 | (None, 6, 6, 64) | 0 |
| dropout_2 (Dropout) | (None, 6, 6, 64) | 0 |
| conv2d_5 (Conv2D) | (None, 6, 6, 64) | 36928 |
| conv2d_6 (Conv2D) | (None, 4, 4, 64) | 36928 |
| max_pooling2d_3 (MaxPooling2 | (None, 2, 2, 64) | 0 |
| dropout_3 (Dropout) | (None, 2, 2, 64) | 0 |
| flatten_1 (Flatten) | (None, 256) | 0 |
| dense_1 (Dense) | (None, 512) | 131584 |
| dropout_4 (Dropout) | (None, 512) | 0 |
| dense_2 (Dense) | (None, 10) | 5130 |

Total params: 276,138
Trainable params: 276,138
Non-trainable params: 0

počet farieb

= 3x(3x3)x32 + 32

= (32x(3x3))x32 + 32

počet sietí vo vrstve

= (32x(3x3))x64 + 64

= (64x(3x3))x64 + 64

veľkosť kernelu

= (64x(3x3))x64 + 64

= (64x(3x3))x64 + 64

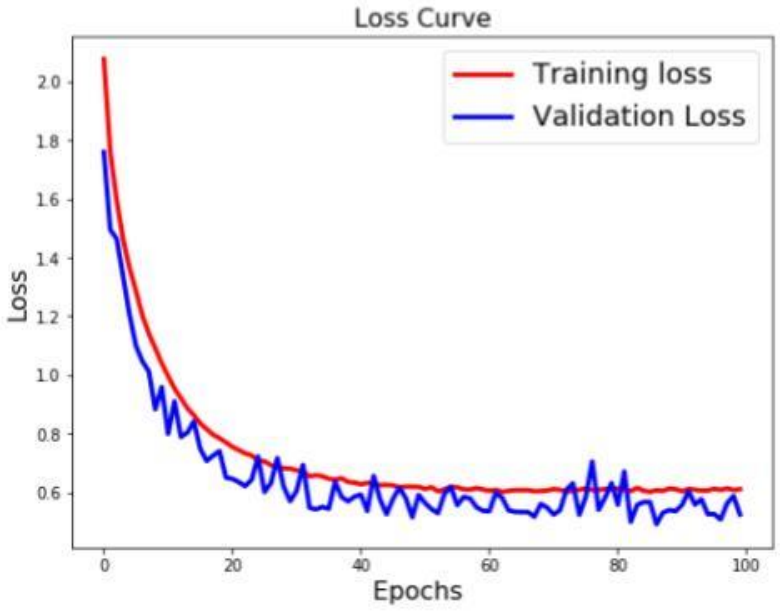
= 512 x 256 + 512

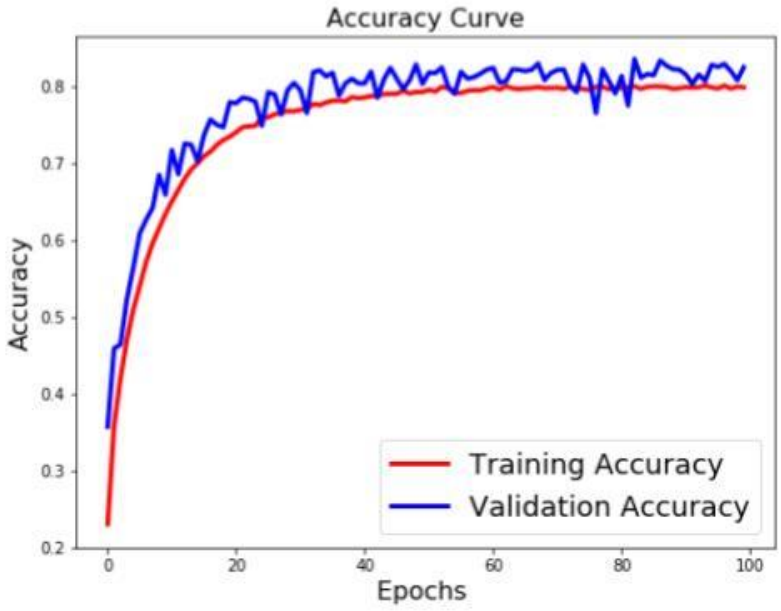
= 10 x 512 + 10

počet kategórii

# Trénovanie

- Data augmentation (obohatenie dát) zmnoží vzorky tým, že ich pootáča a poposúva. Bez toho nastane overfitting (preučenie)

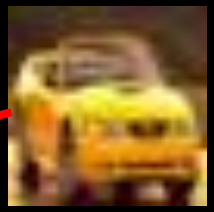


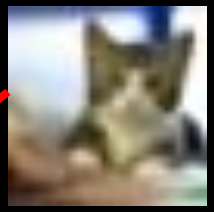



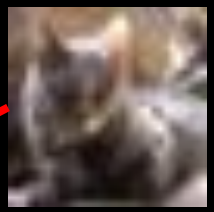
# Použitie

Command Prompt - python

```
>>> import numpy as np
>>> import cv2
>>> from keras.layers import Input
>>> from keras.layers.convolutional import Conv2D
>>> from keras.models import Model
>>> from keras.models import load_model
>>>
>>> model_name = 'keras_cifar10_trained_model.h5'
>>> model = load_model(model_name)
>>>
>>> image1 = cv2.imread('car.png',cv2.IMREAD_COLOR)
>>> image2 = cv2.imread('cat.png',cv2.IMREAD_COLOR)
>>> image3 = cv2.imread('cat2.png',cv2.IMREAD_COLOR)
>>> input_images = np.array([image1,image2,image3])
>>>
>>> categories = model.predict(input_images)
>>>
>>> categories
array([[ 0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.],
       [ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.]], dtype=float32)
>>>
```







# Čo s naučeným modelom?

- V prvom rade ho musíme uložiť do .h5 (keras) prípadne konvertovať do .pb (TensorFlow)
- Môžeme ho potom používať v Pythone (`ker6.py`) aj na platformách, kde samotný proces učenia nie je podporovaný, alebo aspoň nie je pohodlný
- C++ ... OpenCV contrib cv::ml::
- Java ... Deeplearning4j

# Hotové modely

- VGG16 (500MB)
- VGG-BUDDY (50MB)

Ďakujeme za pozornosť

Seminár Robotika.SK

# Hlboké učenie s Keras a OpenCV3

**Andrej Lúčny**

**Katedra aplikovanej informatiky FMFI UK**

**lucny@fmph.uniba.sk**

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