

Deep Learning

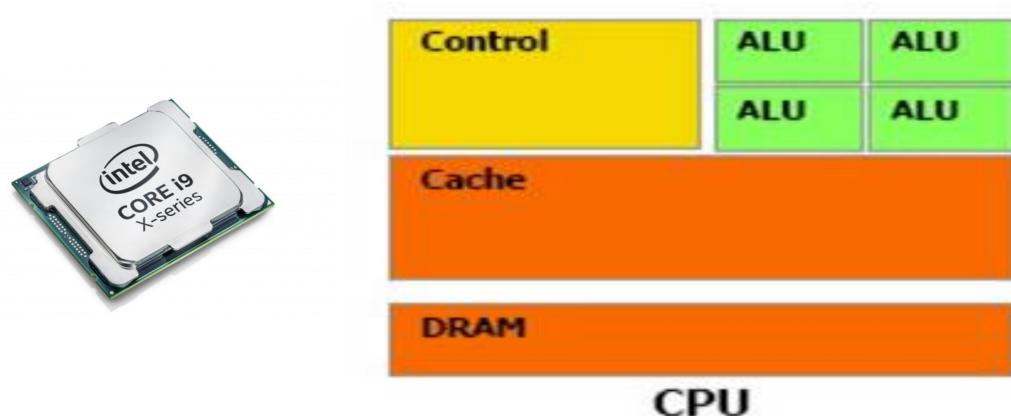
TSM-DeLearn

8. Keras - Convolutional Neural Networks

Jean Hennebert
Martin Melchior



RECAP 1 - CPU vs GPU



CPU - Central Processing Unit

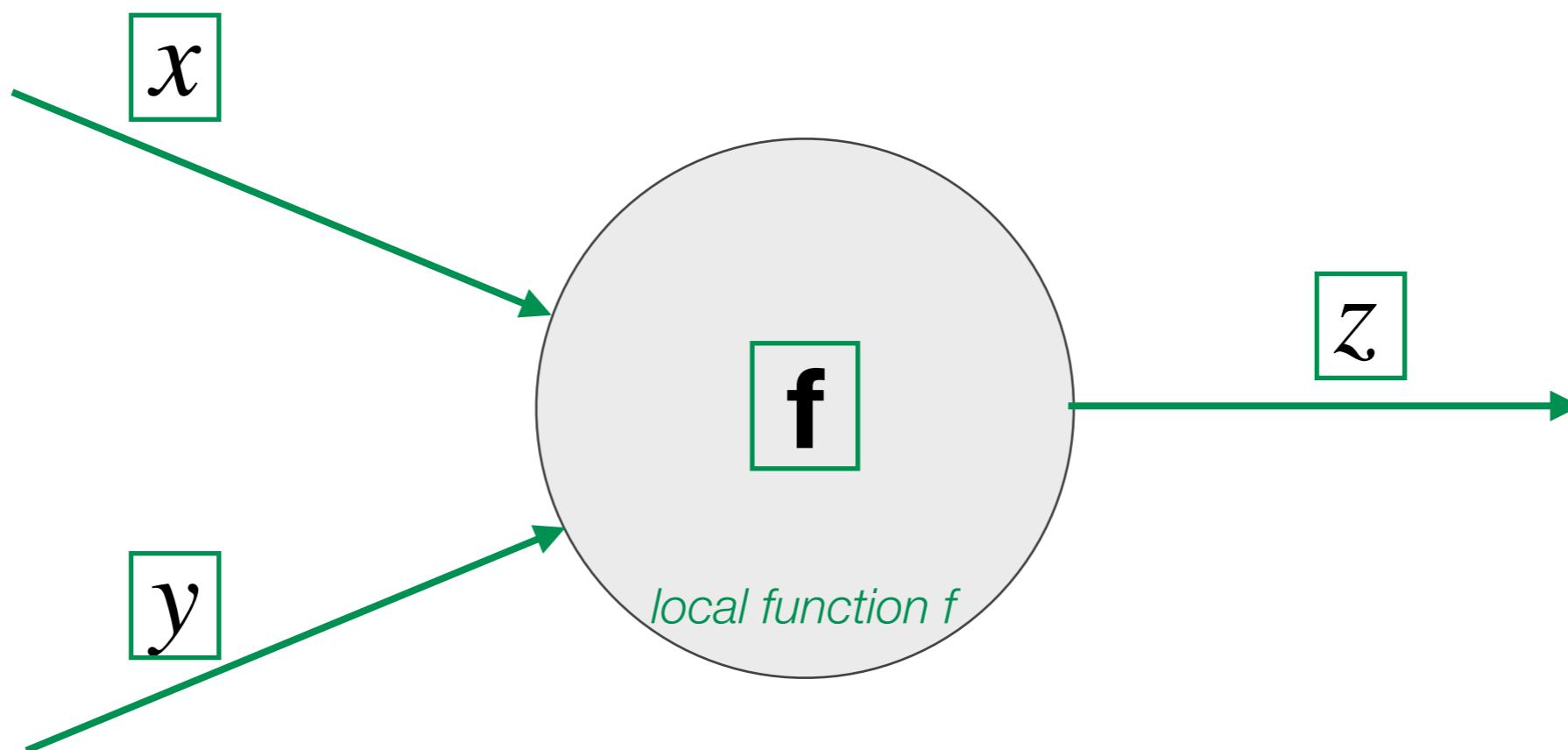
- Few cores (~10), fast (~4 GHz), lots of cache, few parallel processes
- Great at sequential tasks
- Providers: Intel, AMD

GPU - Graphical Processing Unit

- Many cores (~1'000), slow (~1.5 GHz), few cache, many parallel processes
- Great at parallel tasks
- Providers: **NVIDIA**, AMD

RECAP 2 - Computational Graphs

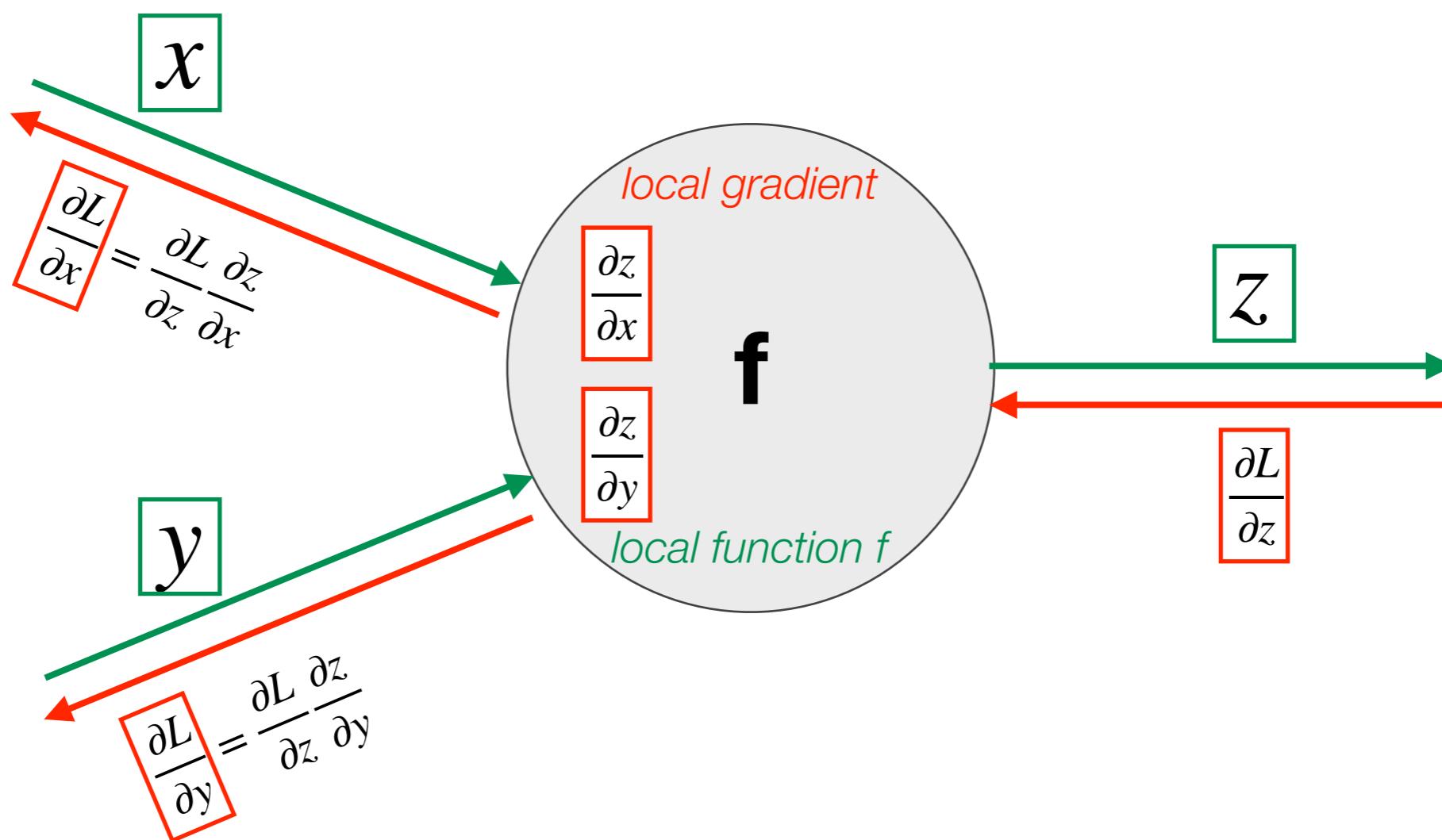
Node - forward / backward principles



Forward = propagate the signal in the graph

RECAP 2 - Computational Graphs

Node - forward / backward principles

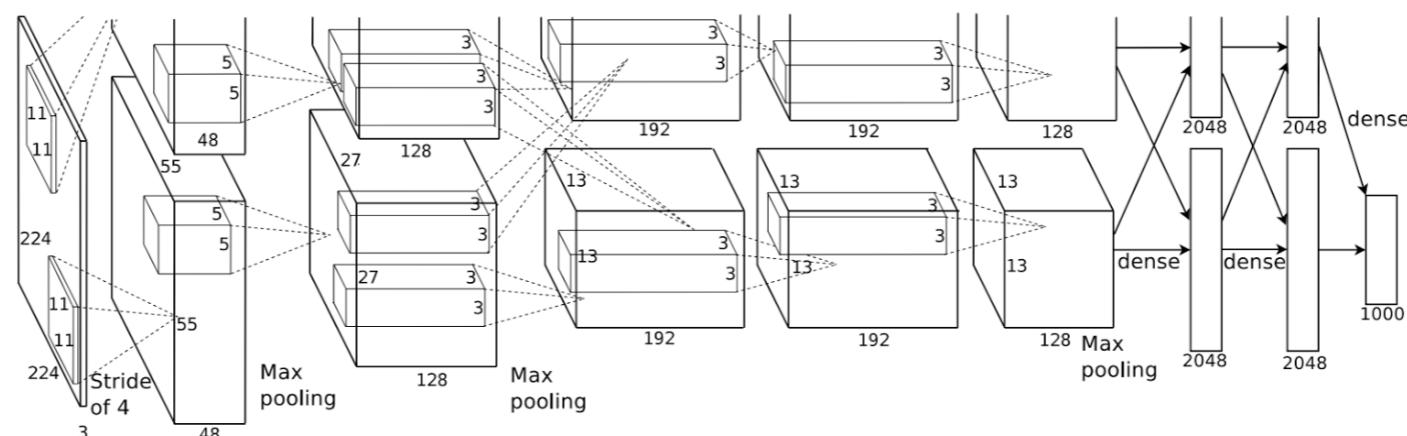
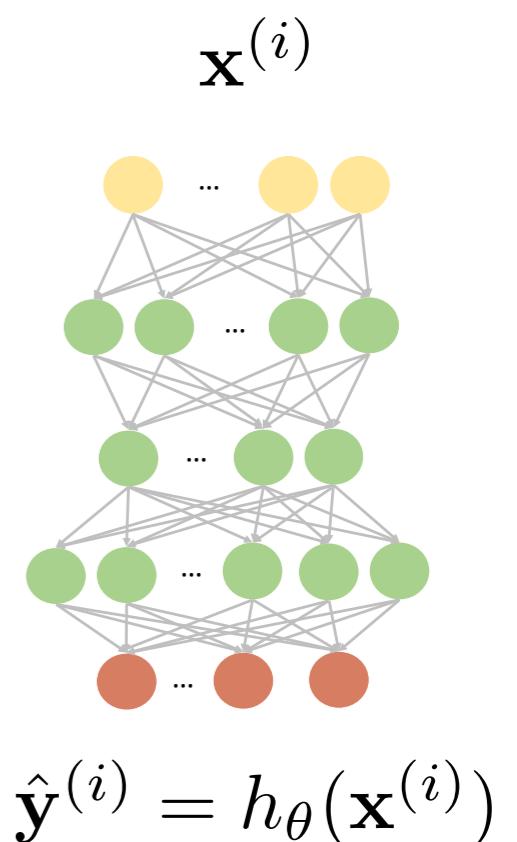


Backward = back-propagate the gradient in the graph with the chain rule

RECAP 3

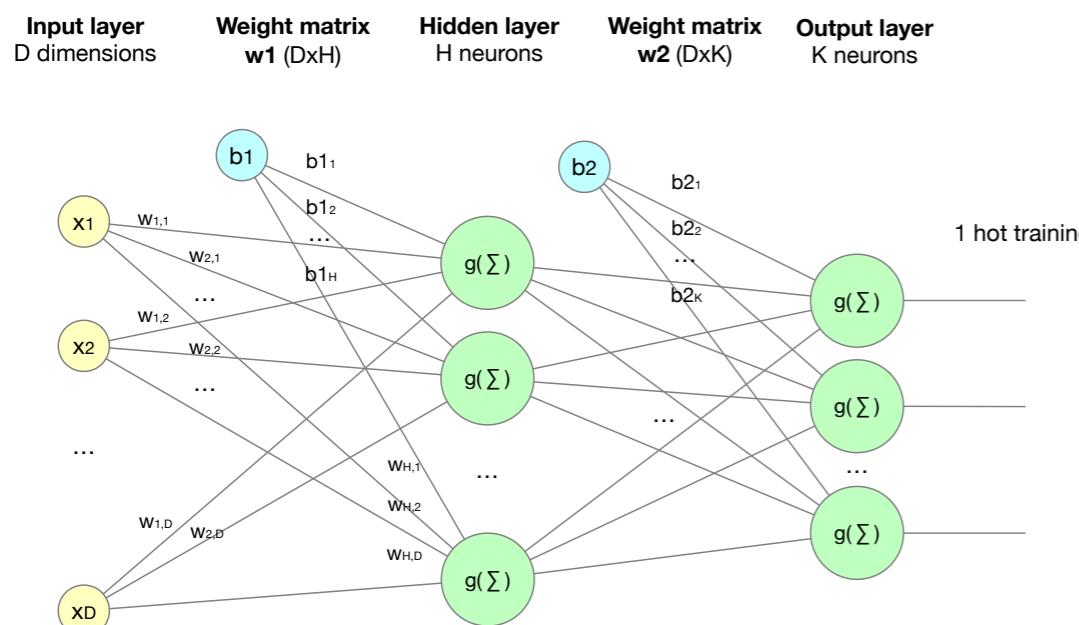
4 main reasons to use DL frameworks

1. Easily build big computational graphs
2. Easily compute losses $L(\hat{\mathbf{y}}^{(i)}, \mathbf{y}^{(i)})$ and gradients in computational graphs for update rules $\text{param} \leftarrow \text{param} - \alpha \frac{\partial L}{\partial \text{param}}$
3. Have at hand all the state-of-the-art strategies for regularisations and optimisations
4. Switch easily from cpu to gpu when needed



RECAP 4 - TensorFlow - a more complex example

First define the computational graph.
No computation, just building the graph.



Then run the graph.

```
# Build the computational graph
x = tf.placeholder(tf.float32, shape=(B, D))
y = tf.placeholder(tf.float32, shape=(B, n_classes))
alpha = tf.placeholder(tf.float32, shape=())
w1 = tf.Variable(tf.truncated_normal((D, H), stddev = 0.1))
b1 = tf.Variable(tf.constant(0.0, shape=[H]))
w2 = tf.Variable(tf.truncated_normal((H, n_classes), stddev = 0.1))
b2 = tf.Variable(tf.constant(0.0, shape=[n_classes]))

h = tf.maximum(tf.matmul(x, w1) + b1, 0.0)
y_pred = tf.sigmoid(tf.matmul(h, w2) + b2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
grad_w1, grad_b1, grad_w2, grad_b2 = tf.gradients(loss, [w1, b1, w2, b2])

new_w1 = w1.assign(w1 - alpha * grad_w1)
new_b1 = b1.assign(b1 - alpha * grad_b1)
new_w2 = w2.assign(w2 - alpha * grad_w2)
new_b2 = b2.assign(b2 - alpha * grad_b2)
updates = tf.group(new_w1, new_b1, new_w2, new_b2)

# Run the computational graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    J = []
    for epoch in range(E):
        J_epoch = 0.0
        for _ in range(int(N/B)): # number of batches to visit for 1 epoch
            x_train_batch, y_train_batch = next_batch(B, x_train, y_train)
            values = { x: x_train_batch, y: y_train_batch, alpha: A}
            loss_val = sess.run([loss, updates], feed_dict=values)
            J_epoch += loss_val[0]
        J.append(J_epoch)
        print("epoch", epoch, J_epoch)

    # now retrieve the weights and bias out of the computational graph
    w1_trained, b1_trained, w2_trained, b2_trained = sess.run([w1, b1, w2, b2])
```

Plan

1. Keras
2. CNN - Convolutional Neural Networks
3. PW on CNN with Keras

Keras

Overview

Cheat Sheet

Example

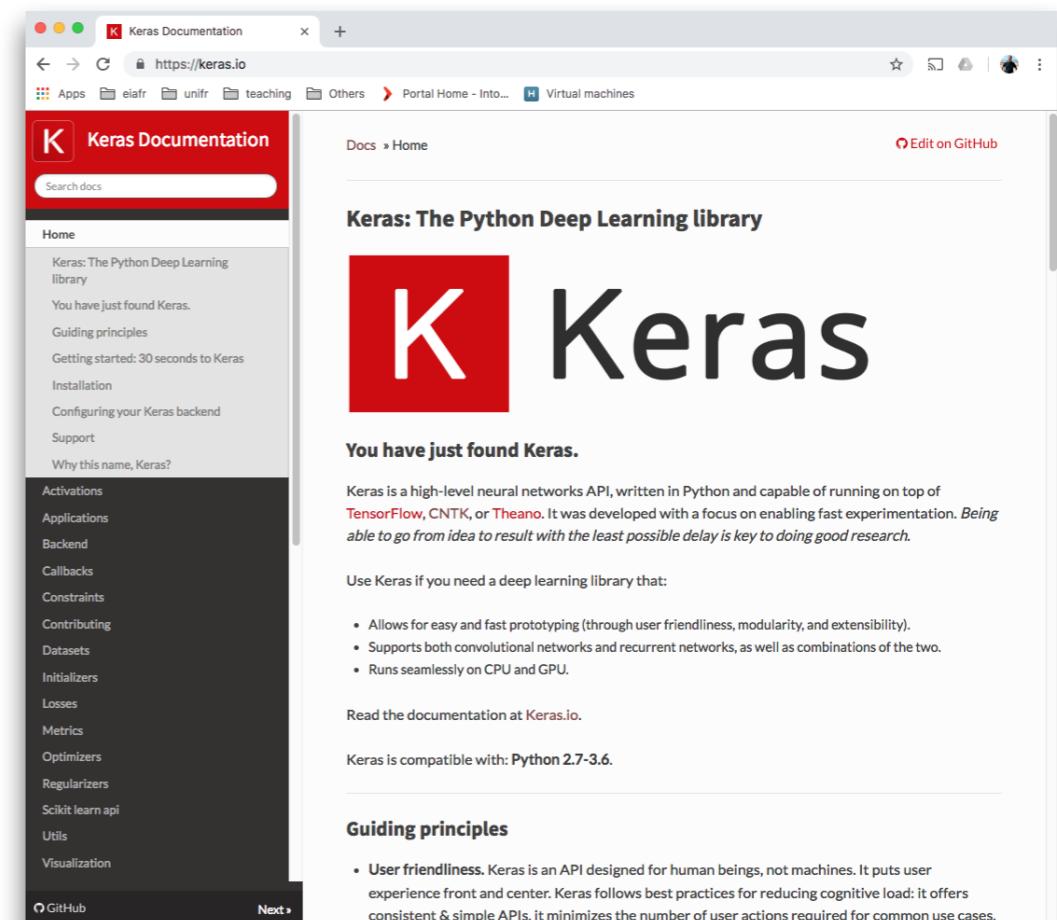


fusquo

Keras - what is it?

Keras is a high-level open-source neural networks API, written in Python and capable of running on top of **TensorFlow**, **CNTK**, or **Theano**. It was developed with a focus on enabling fast experimentation.

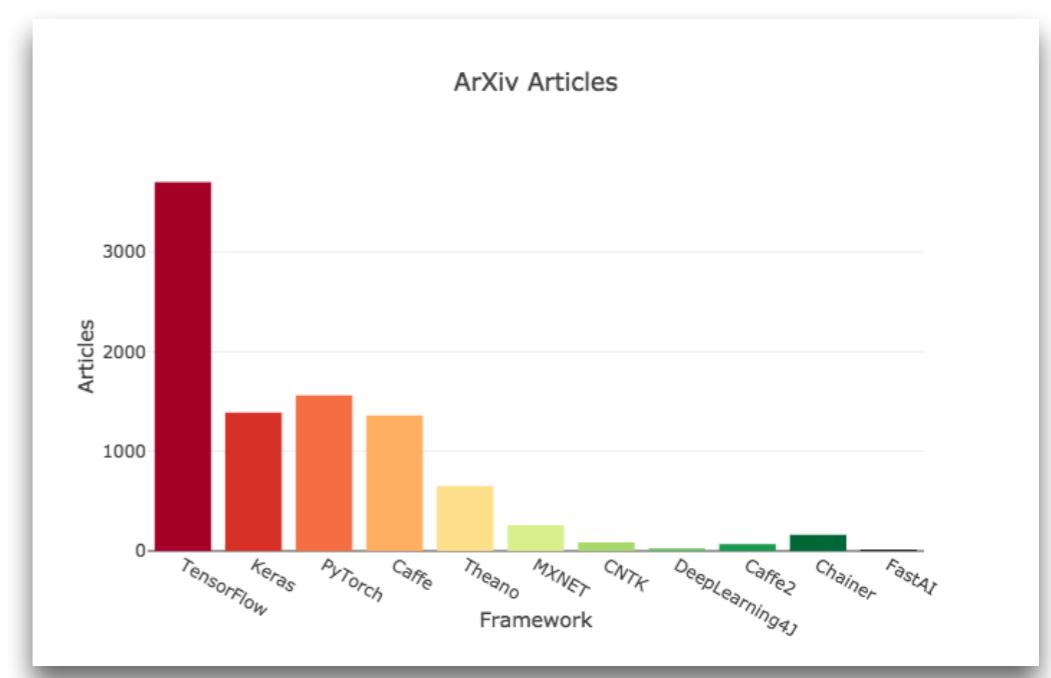
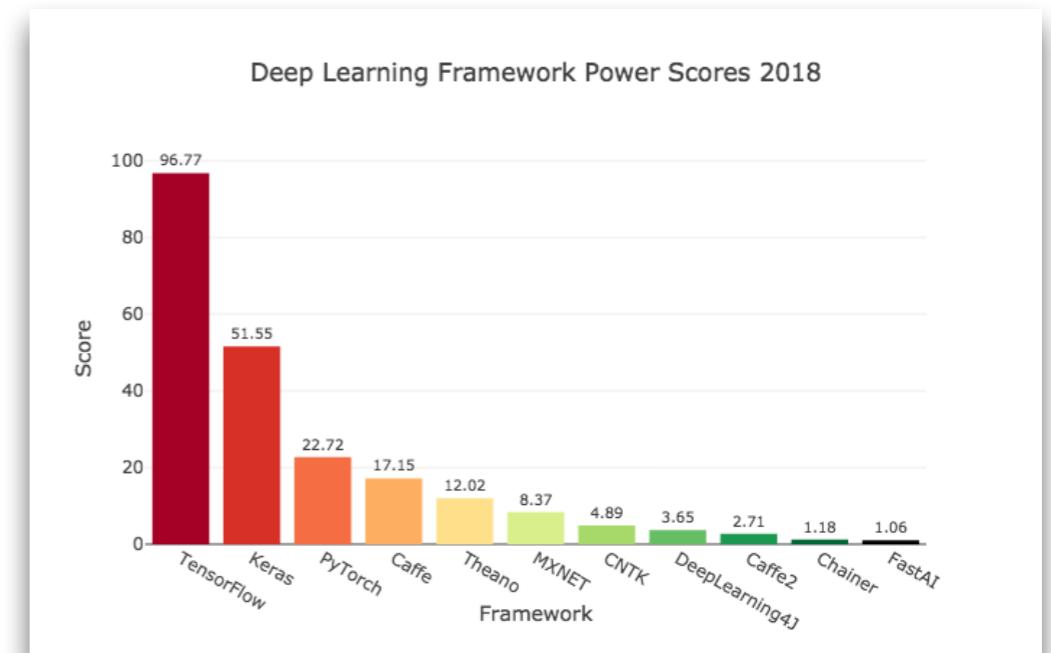
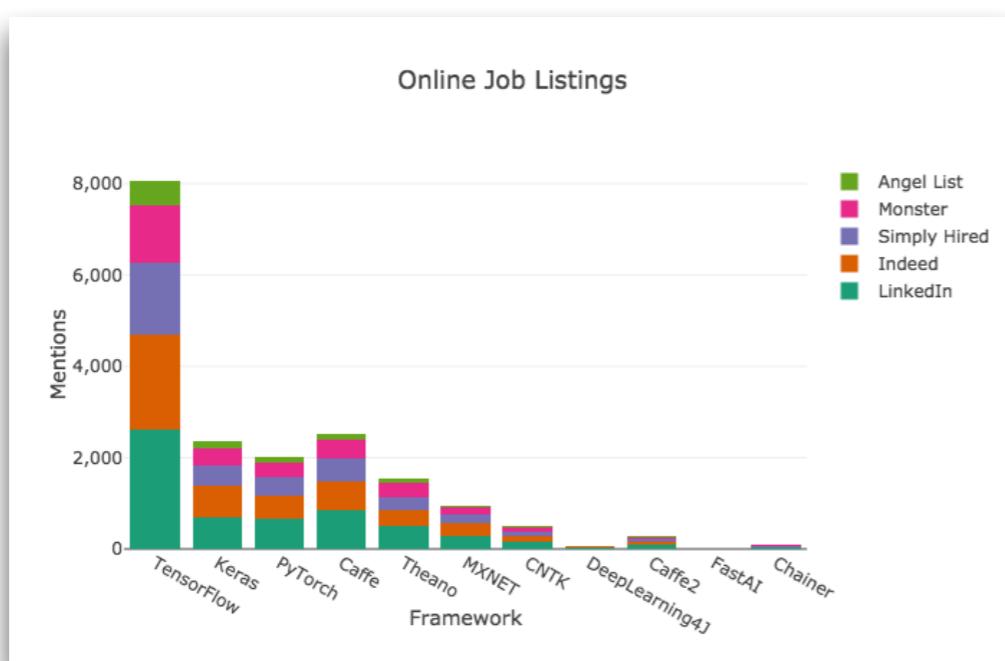
- Designed to be minimalistic & straight forward yet extensive
- “Deep” enough to build serious models
 - Support Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN), combination of both.
- Wrapper on top of TF / CNTK / Theano backend
 - But not only a wrapper - see next slides



Source: <https://keras.io>

Keras - why using?

- Simple to use, well done documentation
- Not only wrapper, it “integrates” with TensorFlow
 - K can call low-level TF methods
 - TF can call high-level Keras methods
- Not only TF, also CNTK (from Microsoft) and Theano
 - Code and model portability
 - For ex train on TF and test on CNTK
 - MXNet coming soon (officially not supported yet)

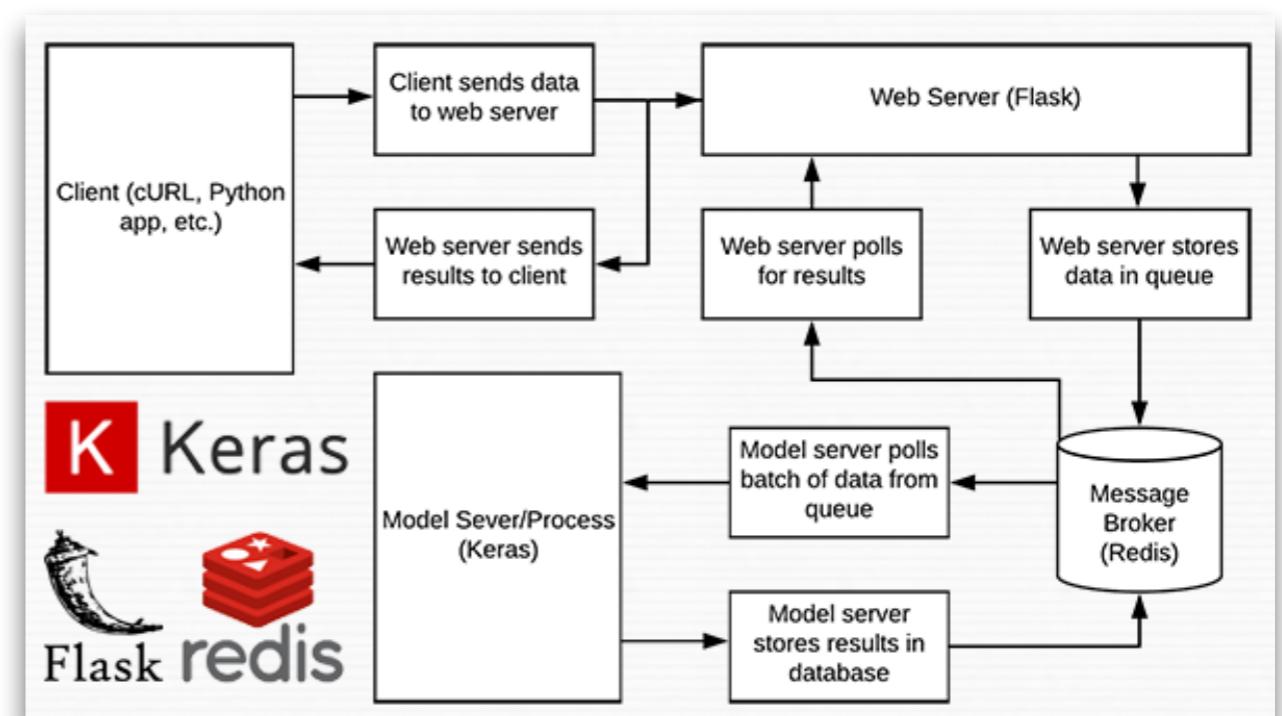


Source: <https://towardsdatascience.com/deep-learning-framework-power-scores-2018-23607ddf297a>

Keras - why using?

- Not only for research, in production at Netflix, Uber, Yelp, Instacart, Zocdoc, Square, and many others.
- Going into production:
 - On iOS, via [Apple's CoreML](#)
 - On Android, via TensorFlow Android runtime.
 - Example: [Not Hotdog app](#). See the hilarious Silicon Valley episode <https://youtu.be/mrk95jFVKqY>
 - In the browser, via GPU-accelerated JavaScript runtimes such as [Keras.js](#) and [WebDNN](#)
 - On Google Cloud, via [TensorFlow-Serving](#)
 - In an R or Python webapp backend (such as a Shiny or [Flask app](#))

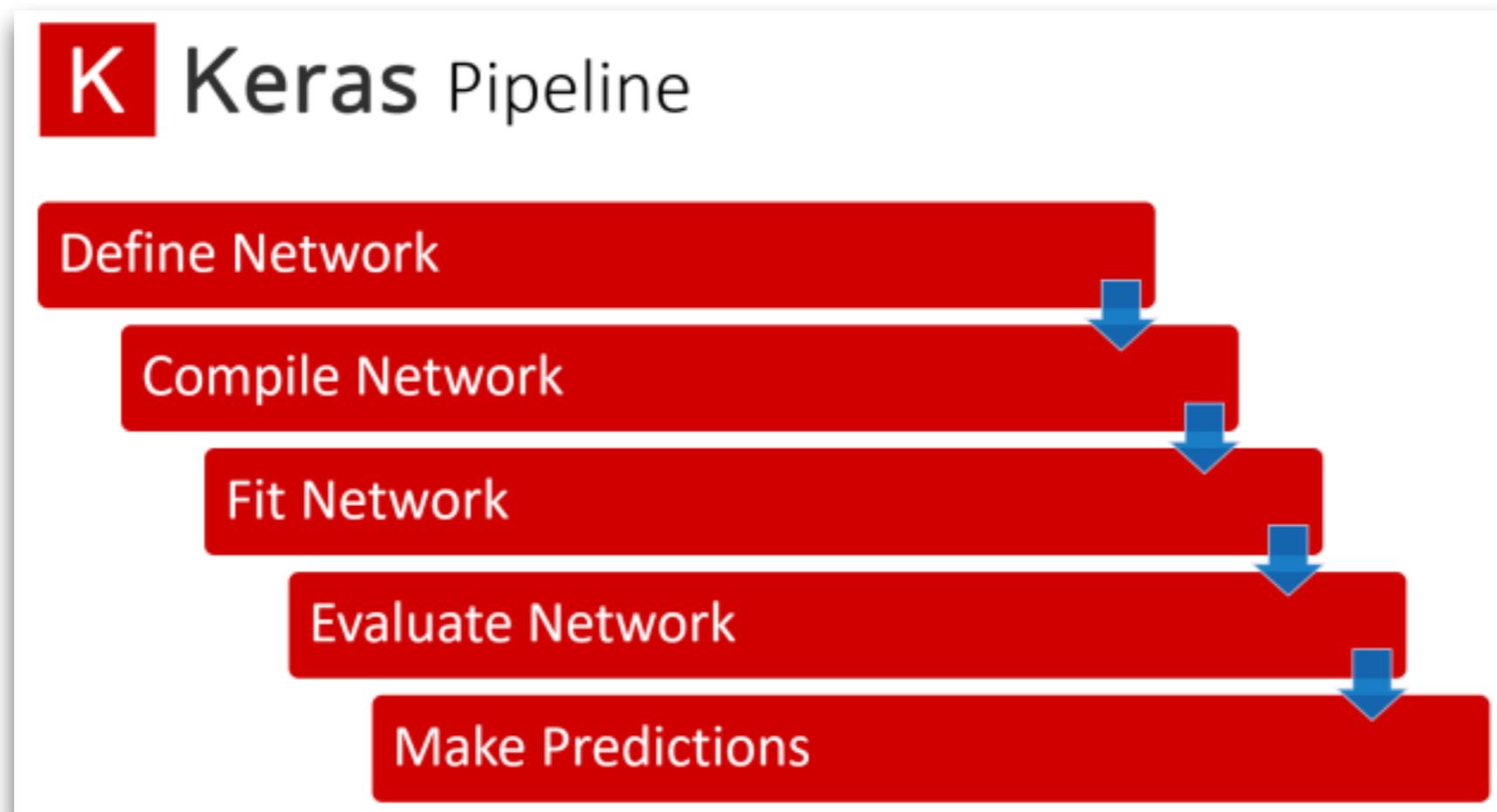
Source: https://keras.rstudio.com/articles/why_use_keras.html



<https://www.pyimagesearch.com/2018/01/29/scalable-keras-deep-learning-rest-api/>

Keras pipeline

- Simple unified neural network pipeline



Source: <https://naadispeaks.wordpress.com/2018/06/26/keras-the-api-for-human-beings/>

In-class exercise - MNIST-MLP with Keras



Activity

- Get the 'MNIST-MLP_from_raw_data-to-complete.ipynb'
- Complete the missing parts to implement a simple MLP taking the pixel data as input.
- Use the cheatsheet as help.

Python For Data Science Cheat Sheet

Keras

Learn Python for data science **Interactively** at www.DataCamp.com



Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
        mnist,
        cifar10,
        imdb

>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/ml/machine-learning-databases/pima-indians-diabetes/pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

Preprocessing

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))
>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.layers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Also see NumPy & Scikit-Learn

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> x_train5,x_test5,y_train5,y_test5 = train_test_split(x,
        y,
        test_size=0.33,
        random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

```
>>> model.output_shape
>>> model.summary()
>>> model.get_config()
>>> model.get_weights()
```

Model output shape
Model summary representation
Model configuration
List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
        y_train4,
        batch_size=32,
        epochs=15,
        verbose=1,
        validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4, batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                  optimizer=opt,
                  metrics=['accuracy'])
```

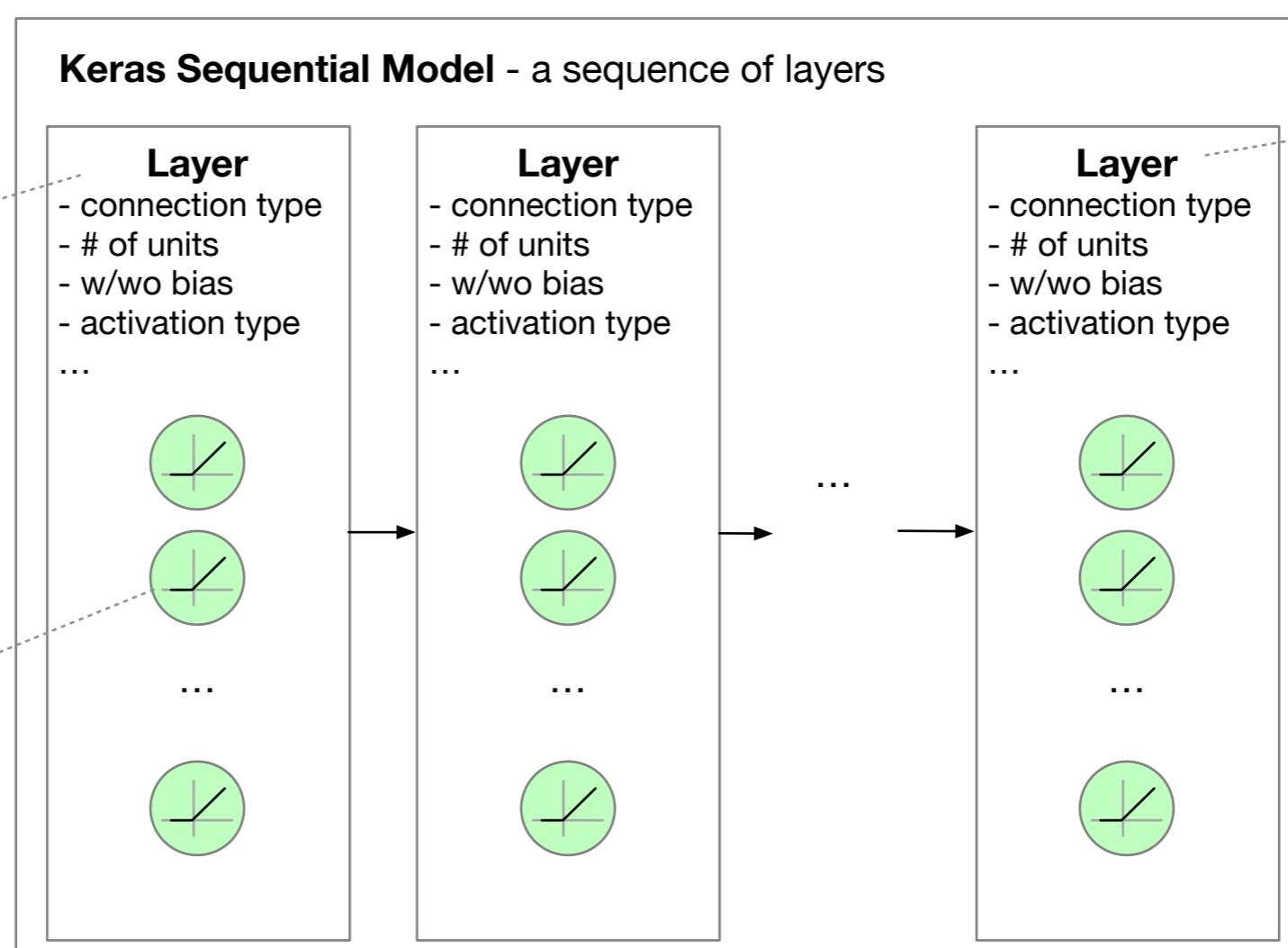
Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
        y_train4,
        batch_size=32,
        epochs=15,
        validation_data=(x_test4,y_test4),
        callbacks=[early_stopping_monitor])
```



Keras Models

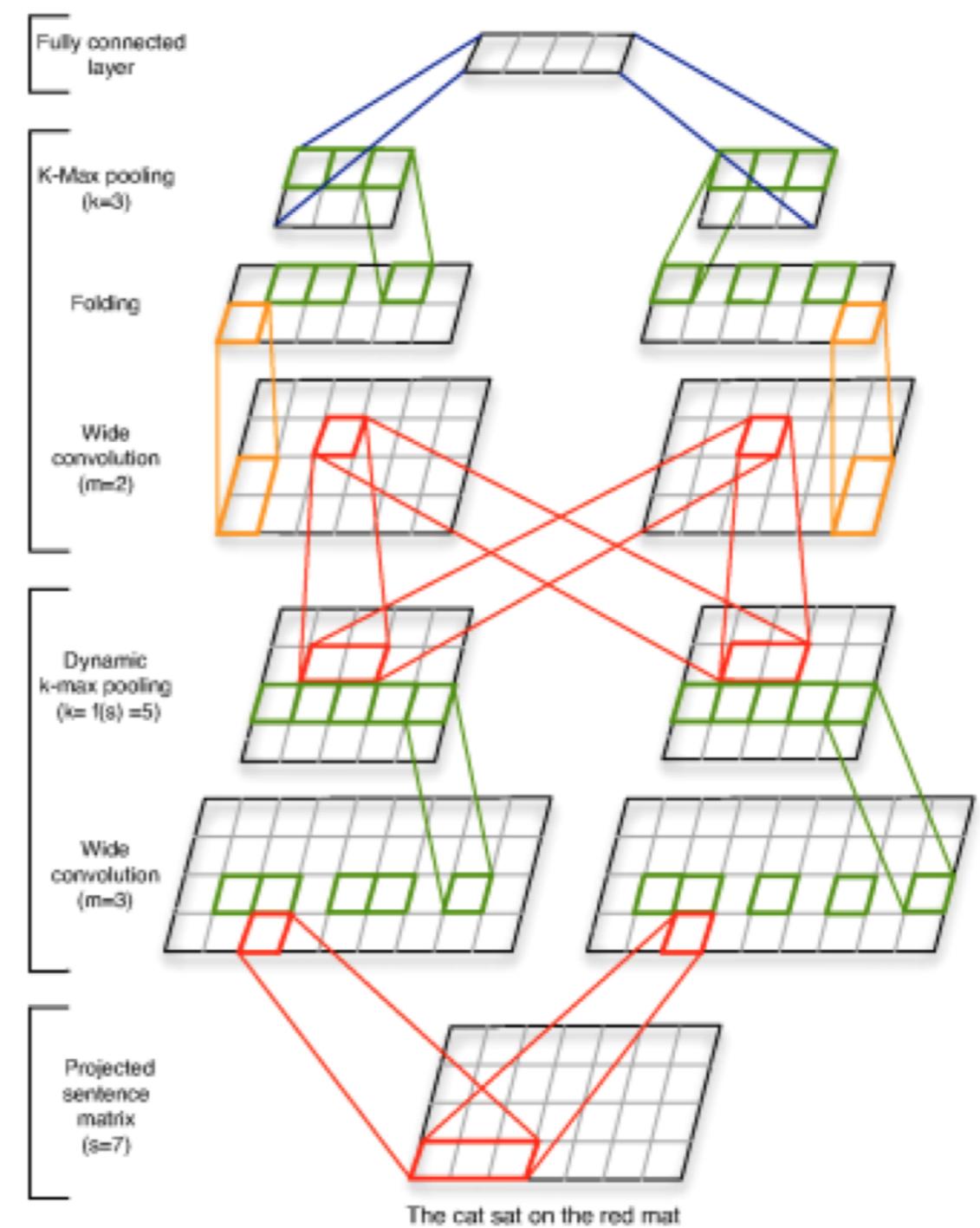
- A **model** in Keras is the way to organise the layers of neurons: sequential or graph
- The **sequential** model corresponds to a regular stack of layers
 - 1 layer = 1 object that feeds to the next



A layer is defined by the type of connections to the previous layer, e.g. Dense, by the number of neurons, by the activation type, e.g. ReLu, sigmoid, ... by the use or not of bias terms, etc

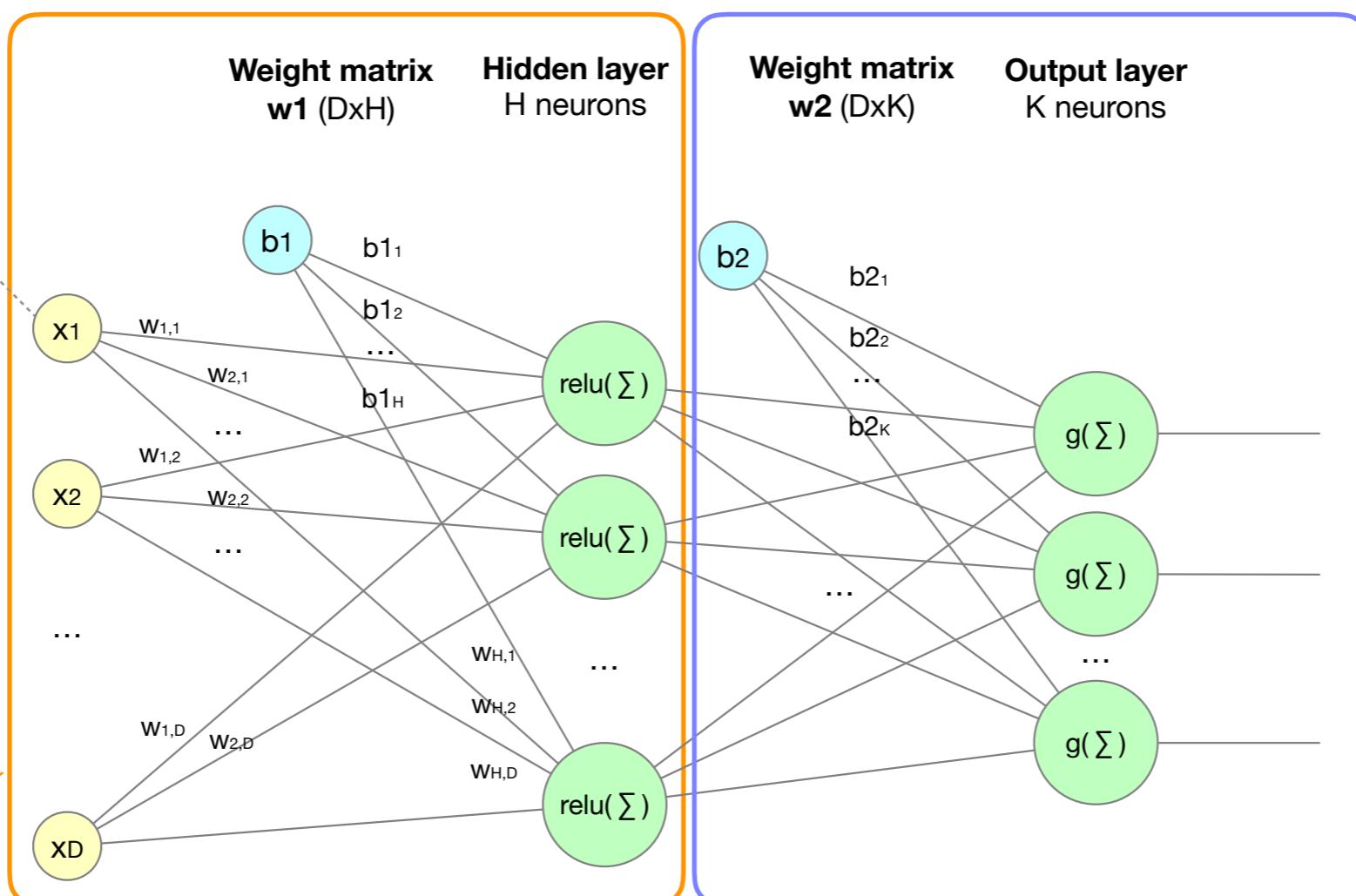
Keras Models

- The **graph** model is used for non sequential architectures
 - Typically allowing for independent networks to diverge or merge



Layers in Keras - Dense

- Dense layers represent fully connected layers of neurons



Note: the “input layer” usually defined in books is actually not a really a layer.

```
model.add(Dense(H, input_shape=(D,), activation='relu'))
```

```
model.add(Dense(n_classes, activation='sigmoid'))
```

Convolutional Neural Networks - CNN

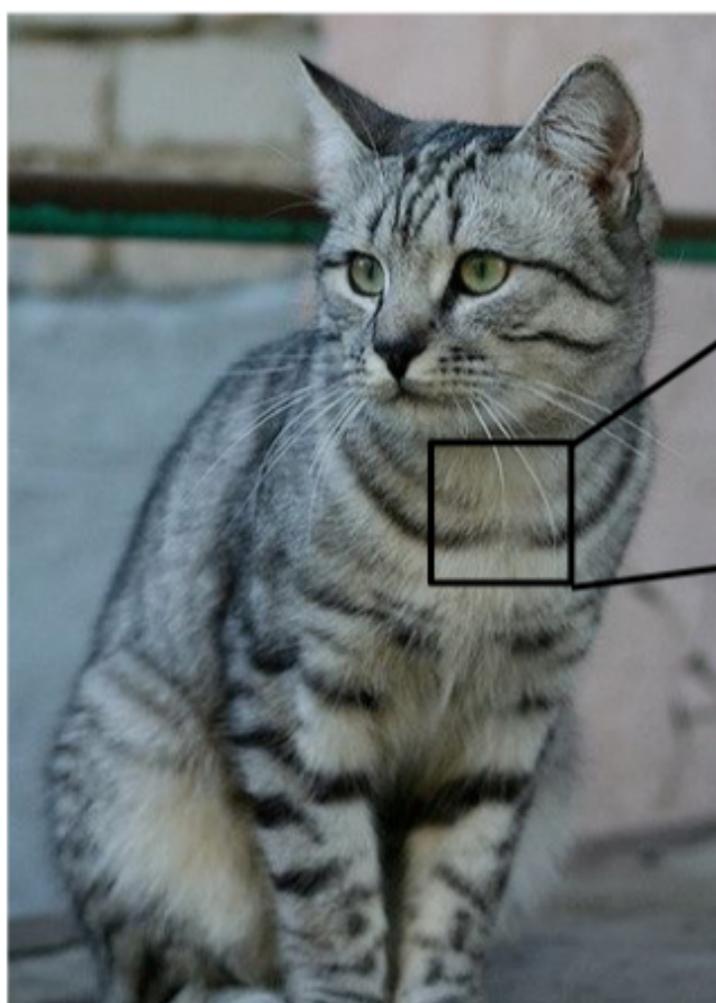
Motivations - image
classification

Intuition behind a one-
layer classifier

CNN Motivations

- Initially motivated to face the challenges of object recognition in images
 - viewpoint, zoom
 - illumination variability
 - object deformation
 - object occlusion
 - background noise
 - intra-class variability
- General idea: let's define new type of layers and connections that will bring
 - preservation of the spatial structure
 - hierarchical feature detection - objects are composed of features that are themselves composed of other features
 - robustness to object variabilities such as viewpoint, occlusion, etc

Image representation



This image by [Nikita](#) is
licensed under [CC-BY 2.0](#).

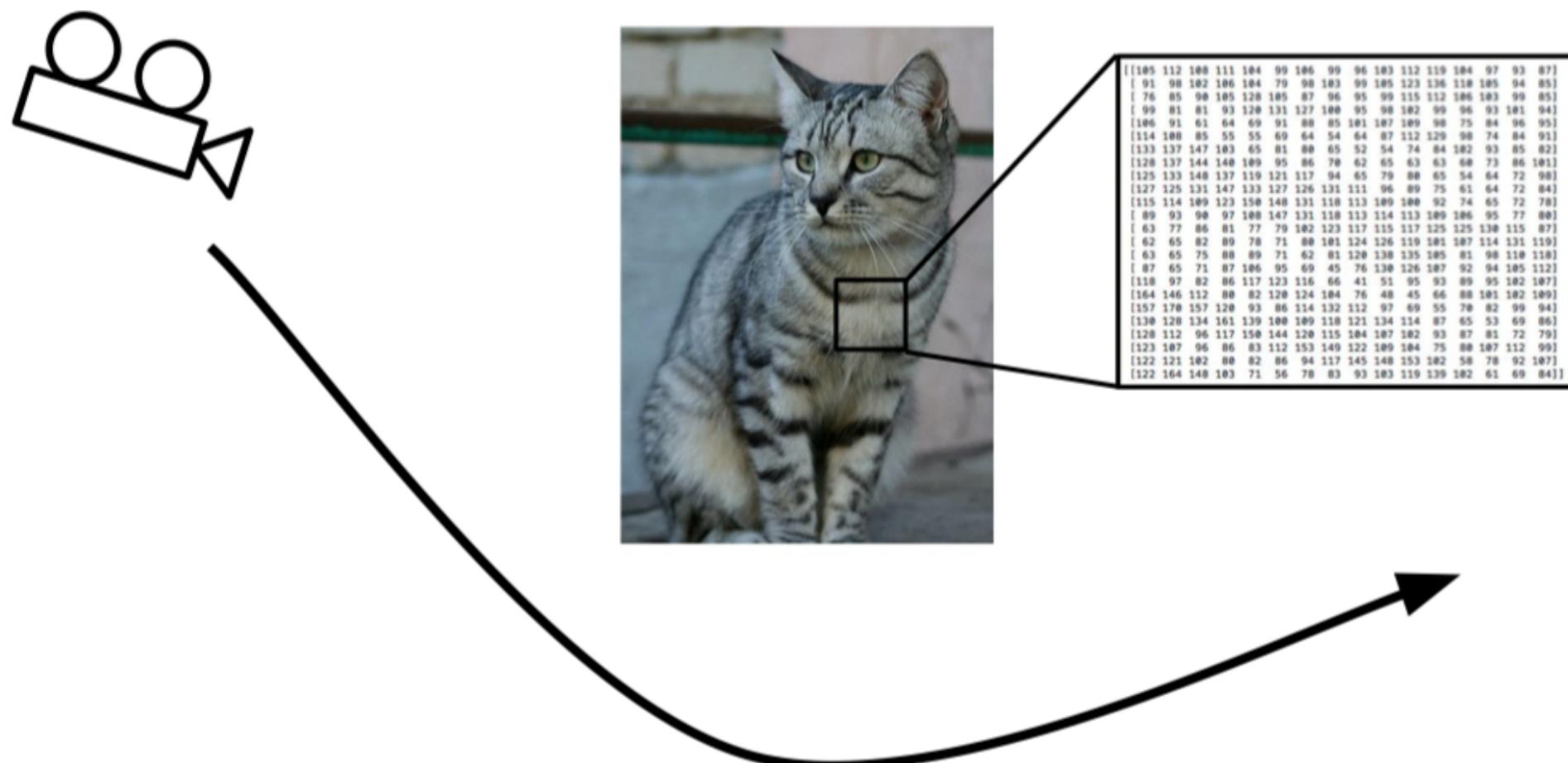
[105 112 108 111 104 99 106 99 96 103 112 119 104 97 93 87]
[91 98 102 106 104 79 98 103 99 105 123 136 118 105 94 85]
[76 85 90 105 128 105 87 96 95 99 115 112 106 103 99 85]
[99 81 81 93 128 131 127 100 95 98 102 99 96 93 101 94]
[106 91 61 64 69 91 88 85 101 107 109 98 75 84 96 95]
[114 108 85 55 55 69 64 54 64 87 112 129 98 74 84 91]
[133 137 147 103 65 81 80 65 52 54 74 84 102 93 85 82]
[128 137 144 140 109 95 86 70 62 65 63 63 68 73 86 101]
[125 133 148 137 119 121 117 94 65 79 80 65 54 64 72 98]
[127 125 131 147 133 127 126 131 111 96 89 75 61 64 72 84]
[115 114 109 123 150 148 131 118 113 109 100 92 74 65 72 78]
[89 93 90 97 108 147 131 118 113 114 113 109 106 95 77 80]
[63 77 86 81 77 79 102 123 117 115 117 125 125 130 115 87]
[62 65 82 89 78 71 80 101 124 126 119 101 107 114 131 119]
[63 65 75 88 89 71 62 81 120 138 135 105 81 98 110 118]
[87 65 71 87 106 95 69 45 76 130 126 107 92 94 105 112]
[118 97 82 86 117 123 116 66 41 51 95 93 89 95 102 107]
[164 146 112 80 82 120 124 104 76 48 45 66 88 101 102 109]
[157 170 157 120 93 86 114 132 112 97 69 55 70 82 99 94]
[130 128 134 161 139 100 109 118 121 134 114 87 65 53 69 86]
[128 112 96 117 150 144 120 115 104 107 102 93 87 81 72 79]
[123 107 96 86 83 112 153 149 122 109 104 75 80 107 112 99]
[122 121 102 80 82 86 94 117 145 148 153 102 58 78 92 107]
[122 164 148 103 71 56 78 83 93 103 119 139 102 61 69 84]]

What the computer sees

An image is just a big grid of numbers between [0, 255]:

e.g. 800 x 600 x 3
(3 channels RGB)

Challenges: viewpoint

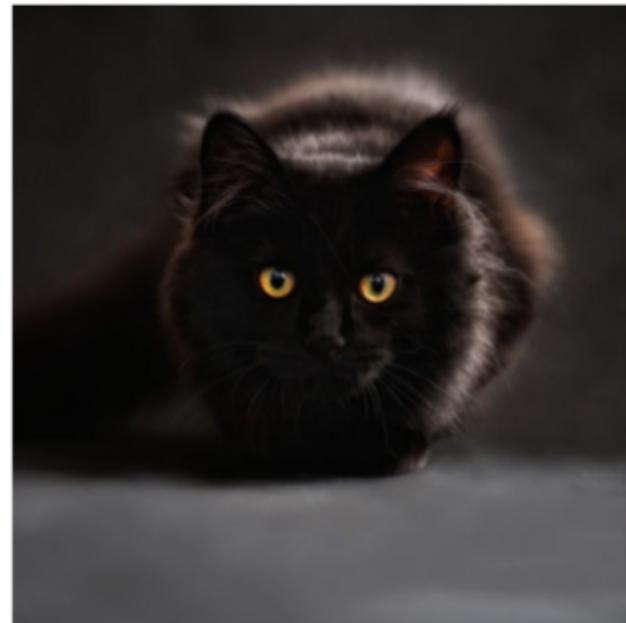


From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

Challenges: illumination / deformations



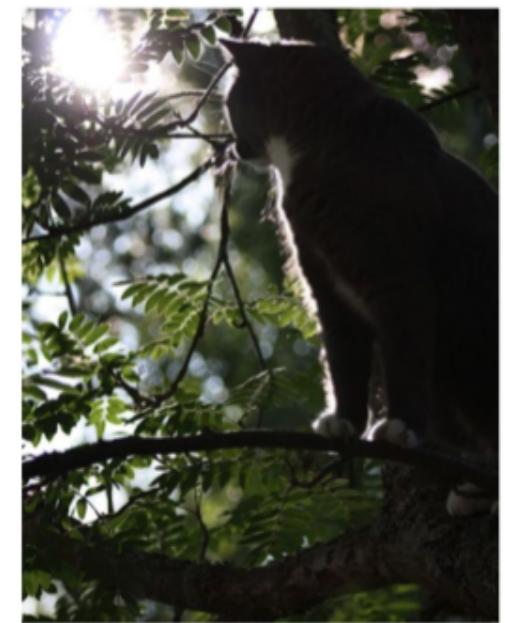
This image is CC0 1.0 public domain



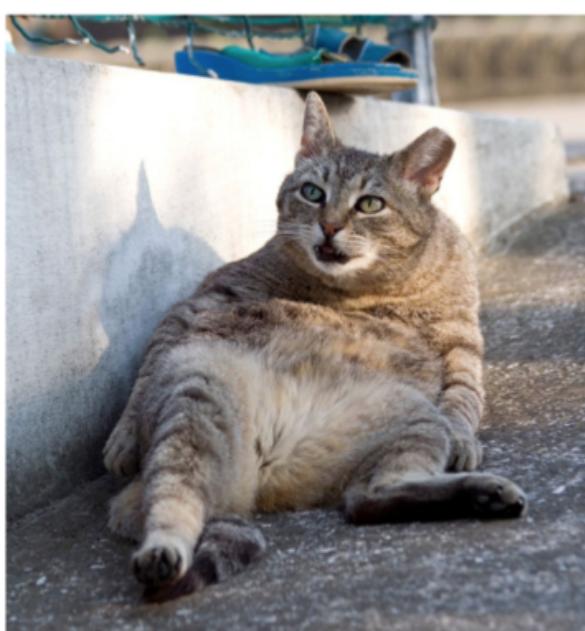
This image is CC0 1.0 public domain



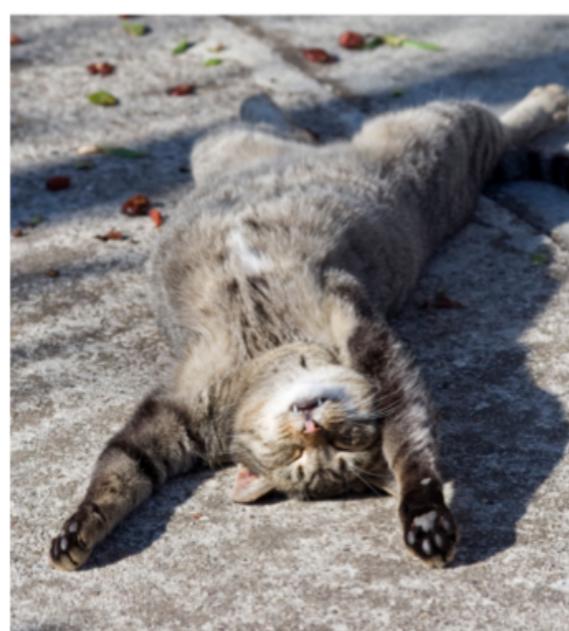
This image is CC0 1.0 public domain



This image is CC0 1.0 public domain



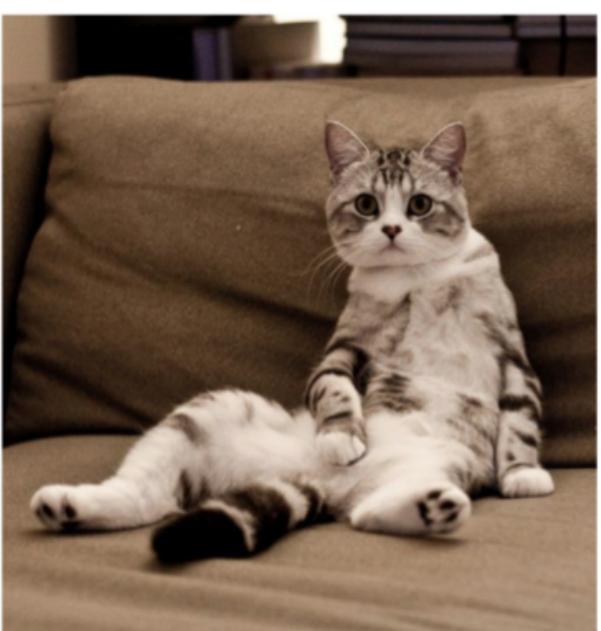
This image by Umberto Salvagnin
is licensed under CC-BY 2.0



This image by Umberto Salvagnin
is licensed under CC-BY 2.0



This image by sare bear is
licensed under CC-BY 2.0



This image by Tom Thajis
licensed under CC-BY 2.0

Challenges: occlusions / background



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)



[This image by ionsson is licensed under CC-BY 2.0](#)



[This image is CC0 1.0 public domain](#)



[This image is CC0 1.0 public domain](#)

Challenges: intra-class variabilities

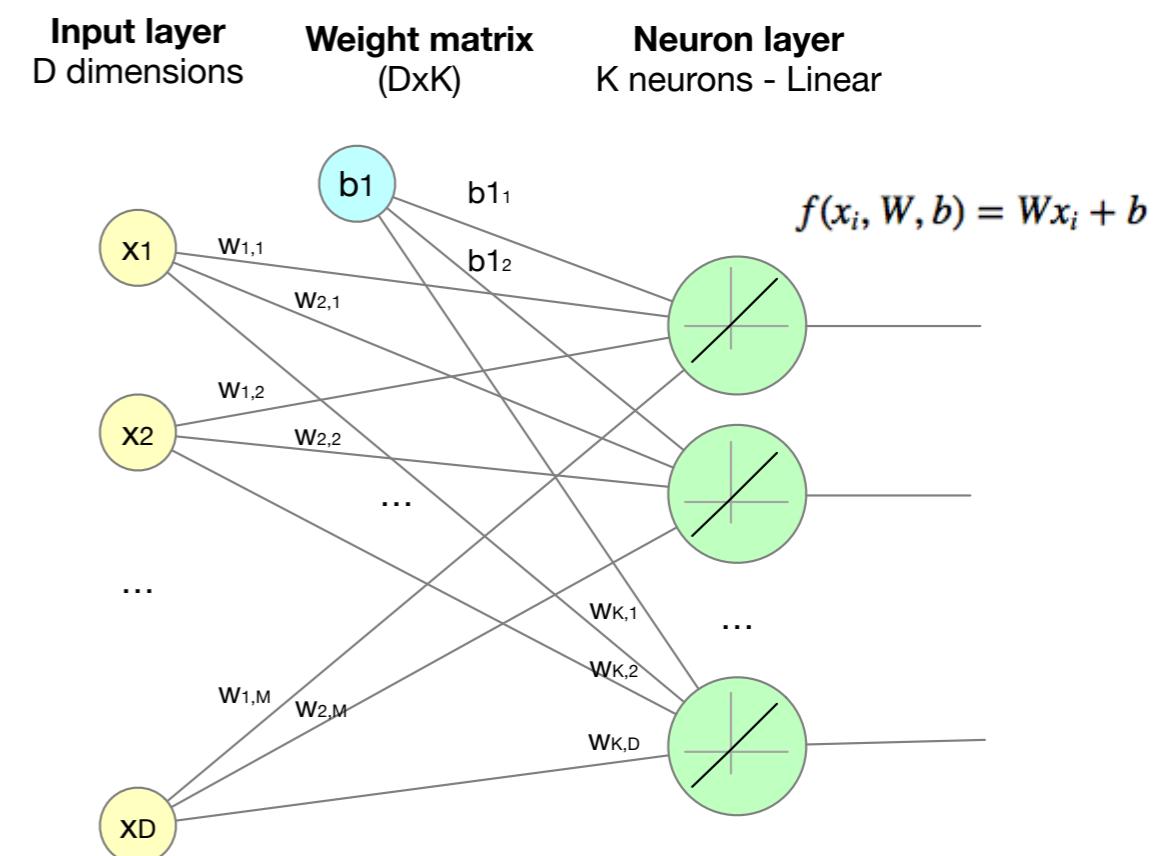
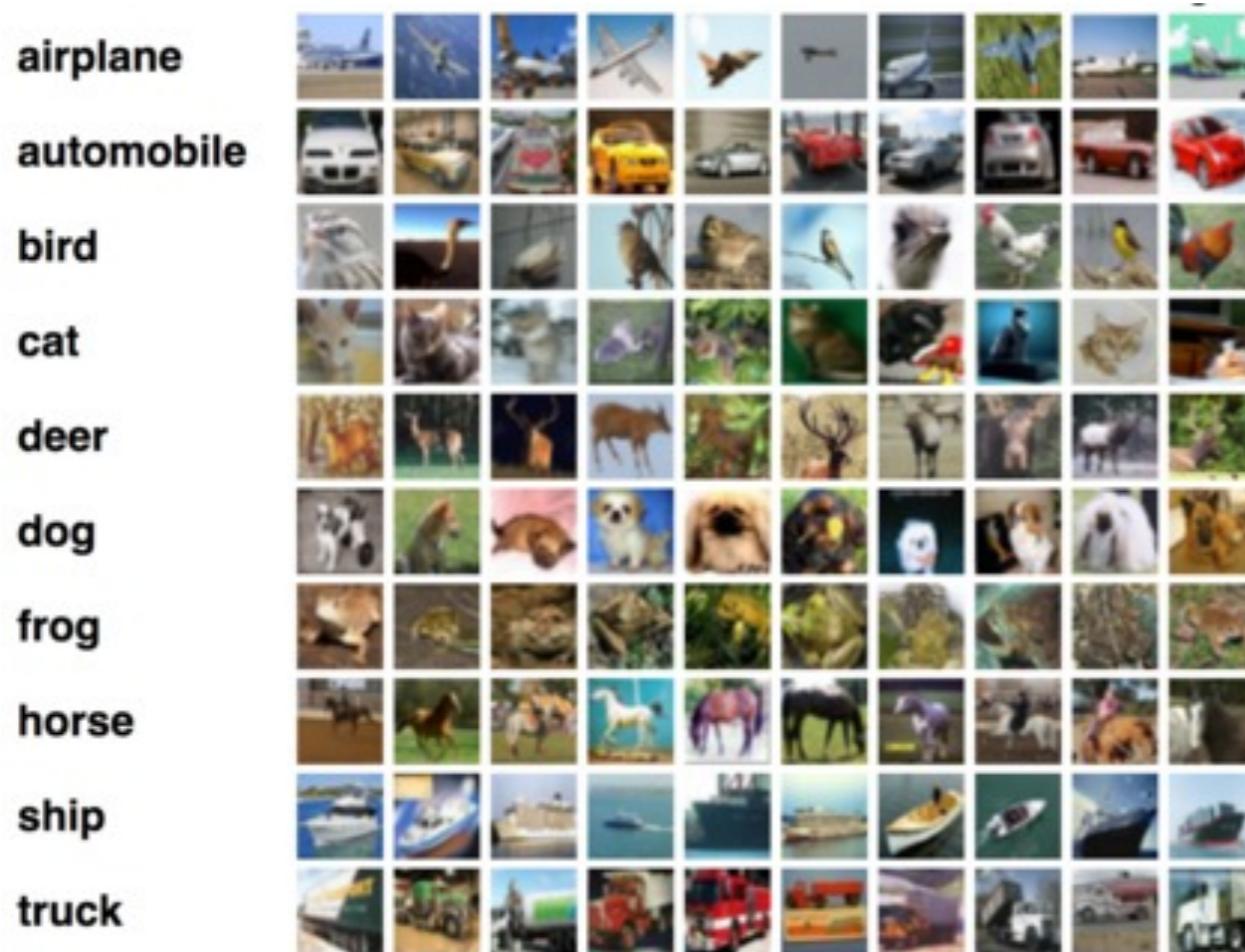


[This image](#) is CC0 1.0 public domain

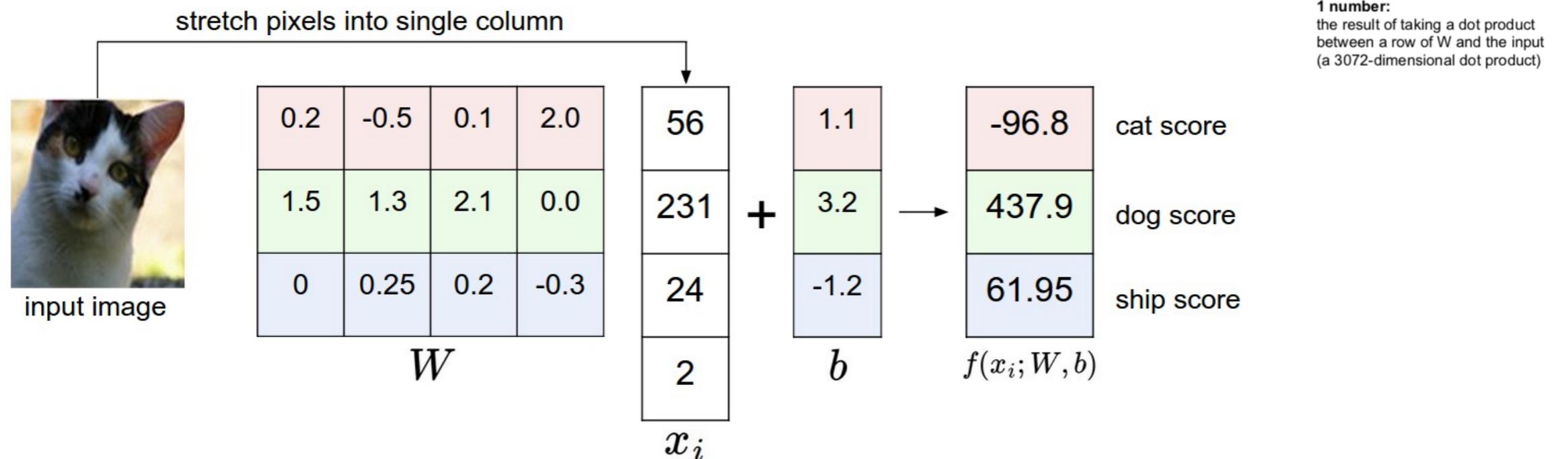
From Fei Fei Li - Stanford University School of Engineering - Class #2 on Image Classification

Example on CIFAR10 - let's assume a one-layer classifier

- 60'000 images : 10 classes X 6'000 images
- input dim = 32 X 32 X 3
- 50'000 for training and 10'000 for test



Example on CIFAR10 - let's assume a one-layer classifier

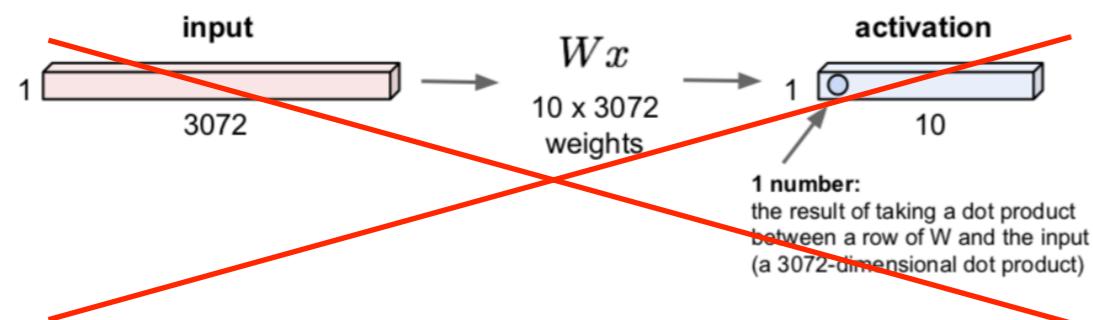


- The weights on a given row are connected to a given output neuron and then contributes to a given class “score”
- Intuitively, we can feel that if the weights are “aligned” to the pixel values, then the score for the given class will be high
- For this reason, the weights for a given class can be interpreted as “templates” or “filters” for that class
- As these weights have the same dimension as the input image, they can actually be associated to the pixel positions and compose images (after re-scaling)

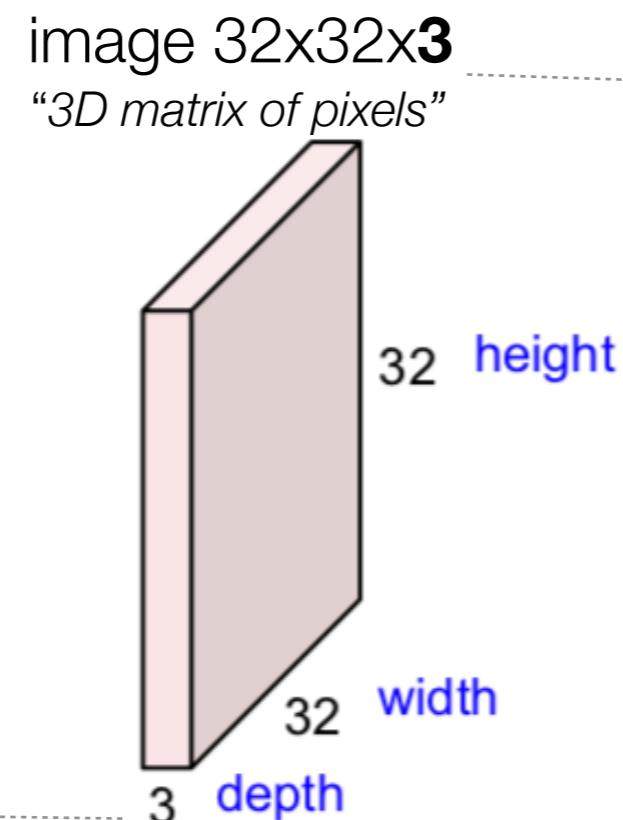


Convolution layer - 2D signals

- The first idea is to **preserve the spatial structure**

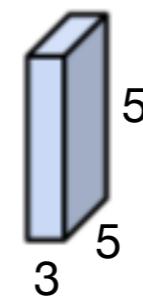


In dense layers, we completely lose the information on the spatial structure as the output neurons are fully connected. E.g. we could shuffle the pixels randomly (but equally for all images) and the network would perform the same.



Images in colour have 3 “channels”, Red-Green-Blue RGB

filter 5x5x3
“3D matrix of weights”

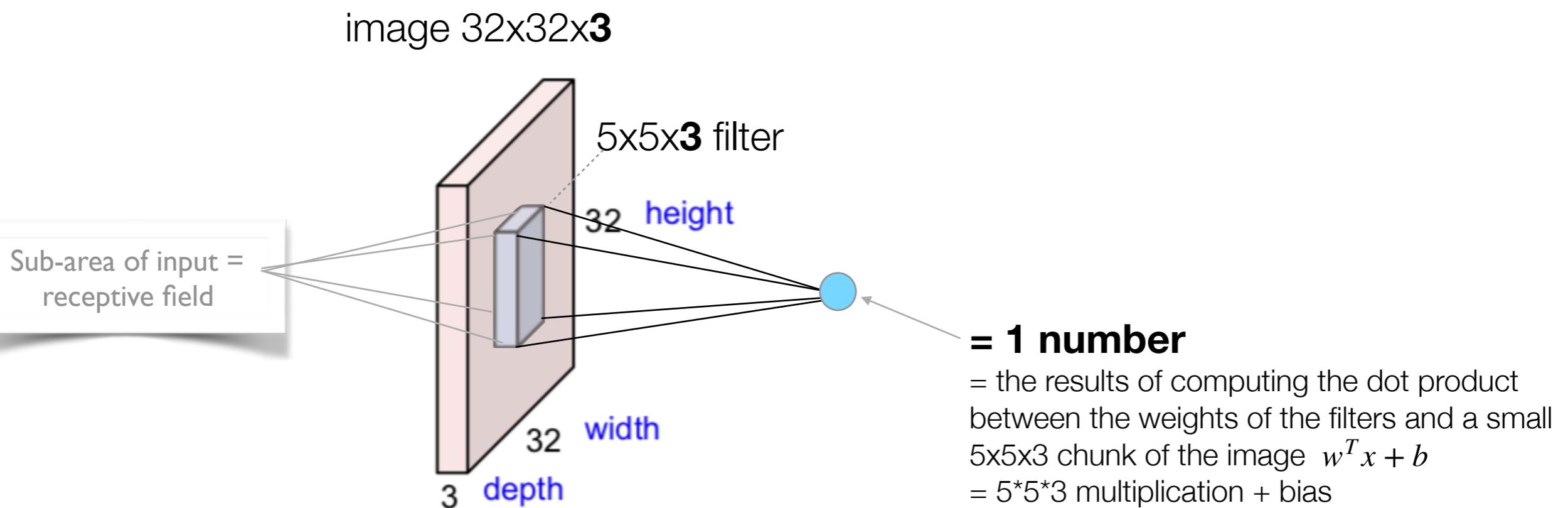


Filters have the same depth as the input “volume”

Filters are also called “kernels”

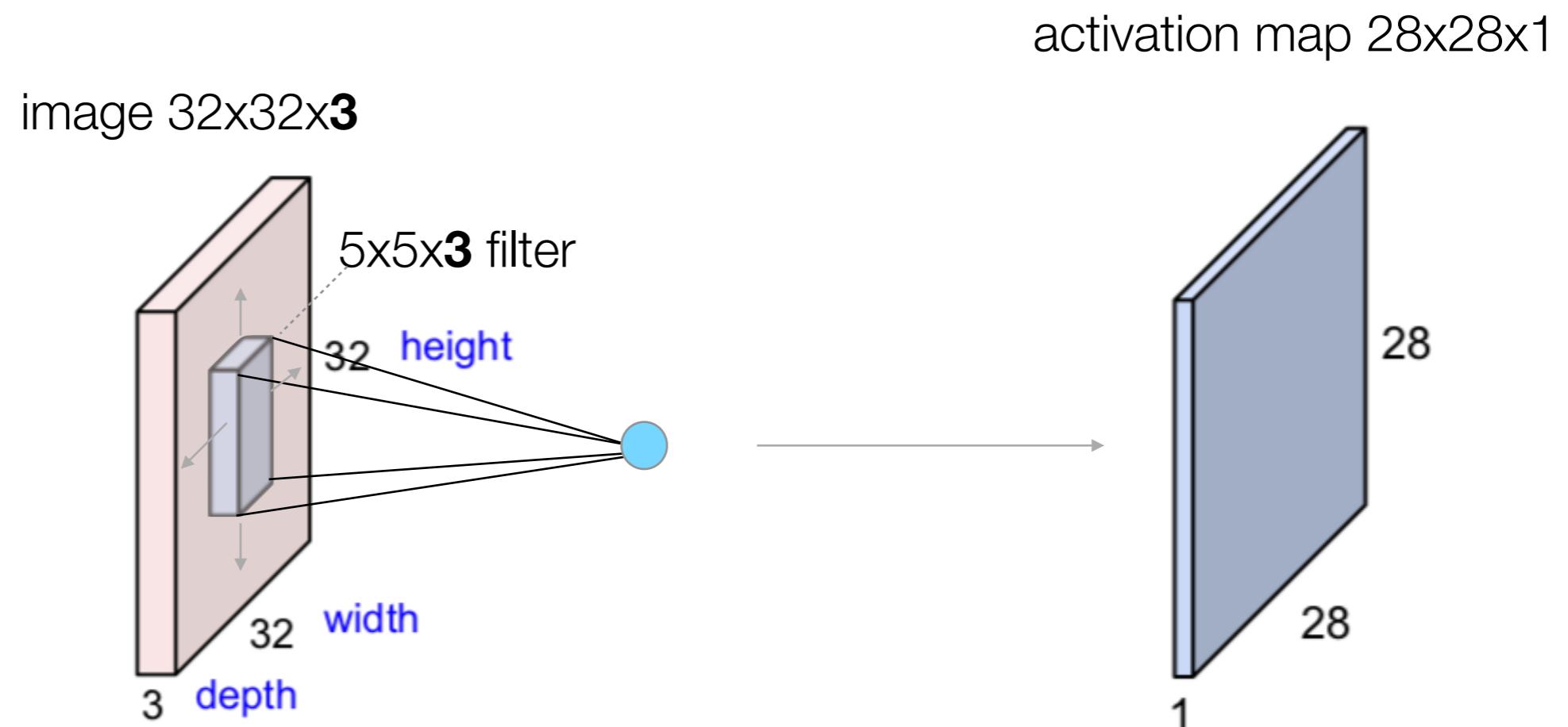
Convolution layer - 2D signals

- The second idea is to **convolve the filter with the image**, in our case “slide over the image spatially, computing dot products of the weights of the filter with the window of pixels where we slide the filter.
- The sub-area of an input that influences a component of the output is sometimes called the **receptive field** of the latter.



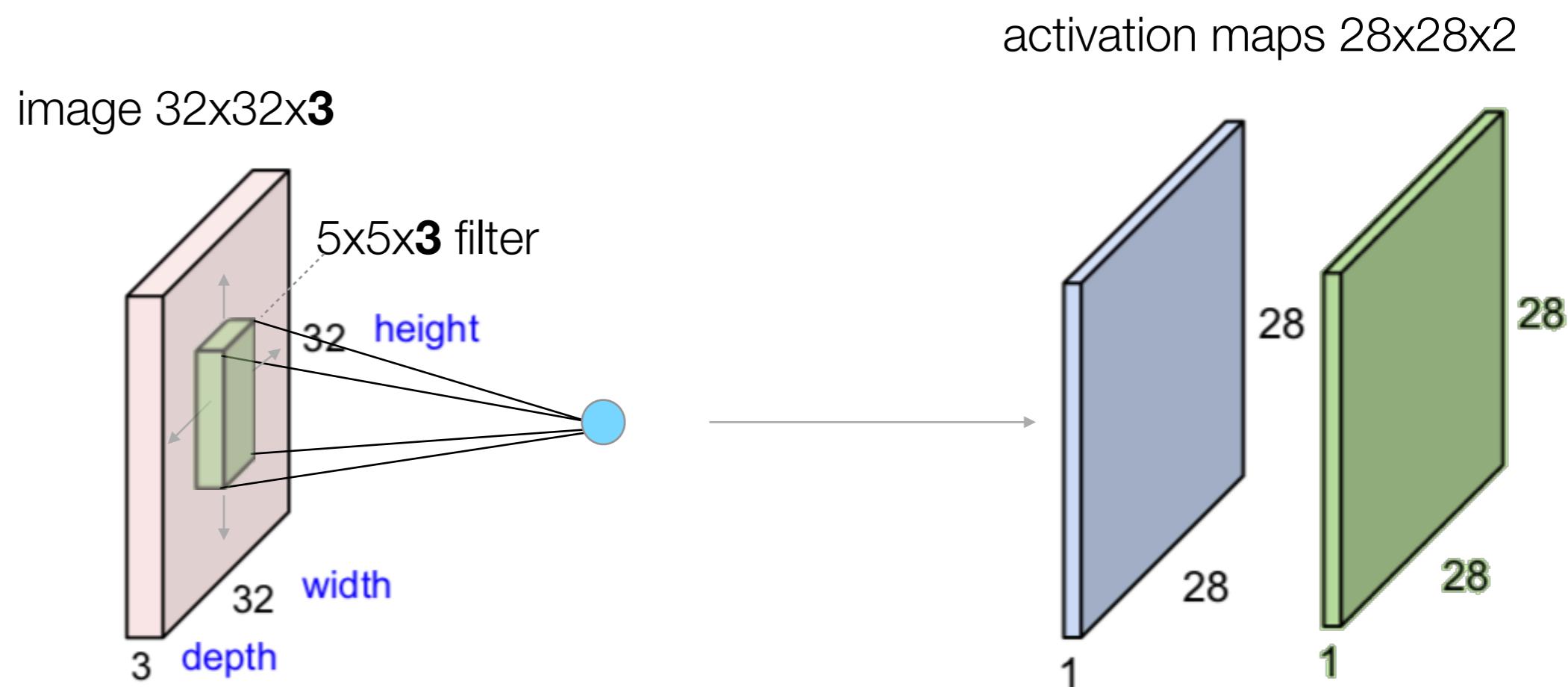
Convolution layer - 2D signals

- Notion of **activation map** that is the result of the convolution of the filter, i.e. the sliding over all possible spatial locations of the image.
- This sliding of the filter gives us **translation invariance** regarding to what the filter is sensitive for



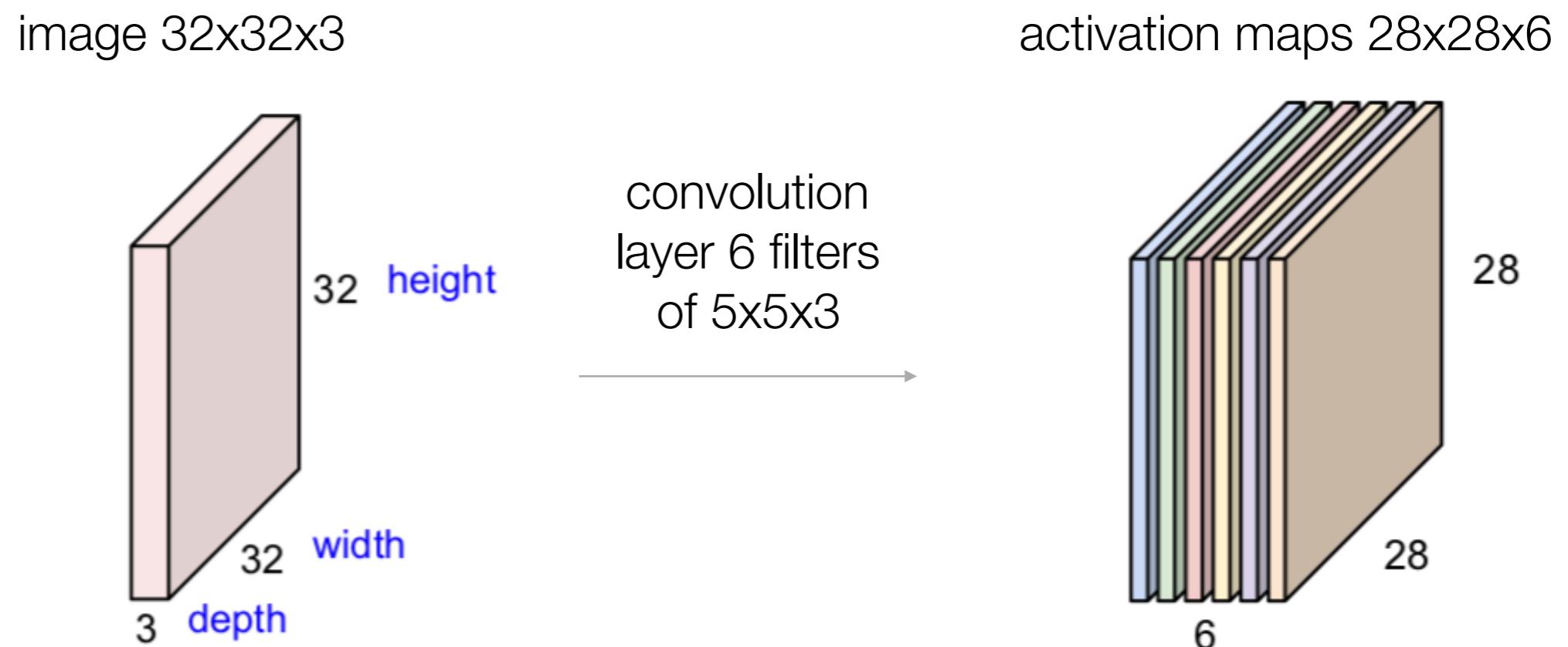
Convolution layer - 2D signals

- Notion of **several filters** producing **several activation maps**
- Now with another filter



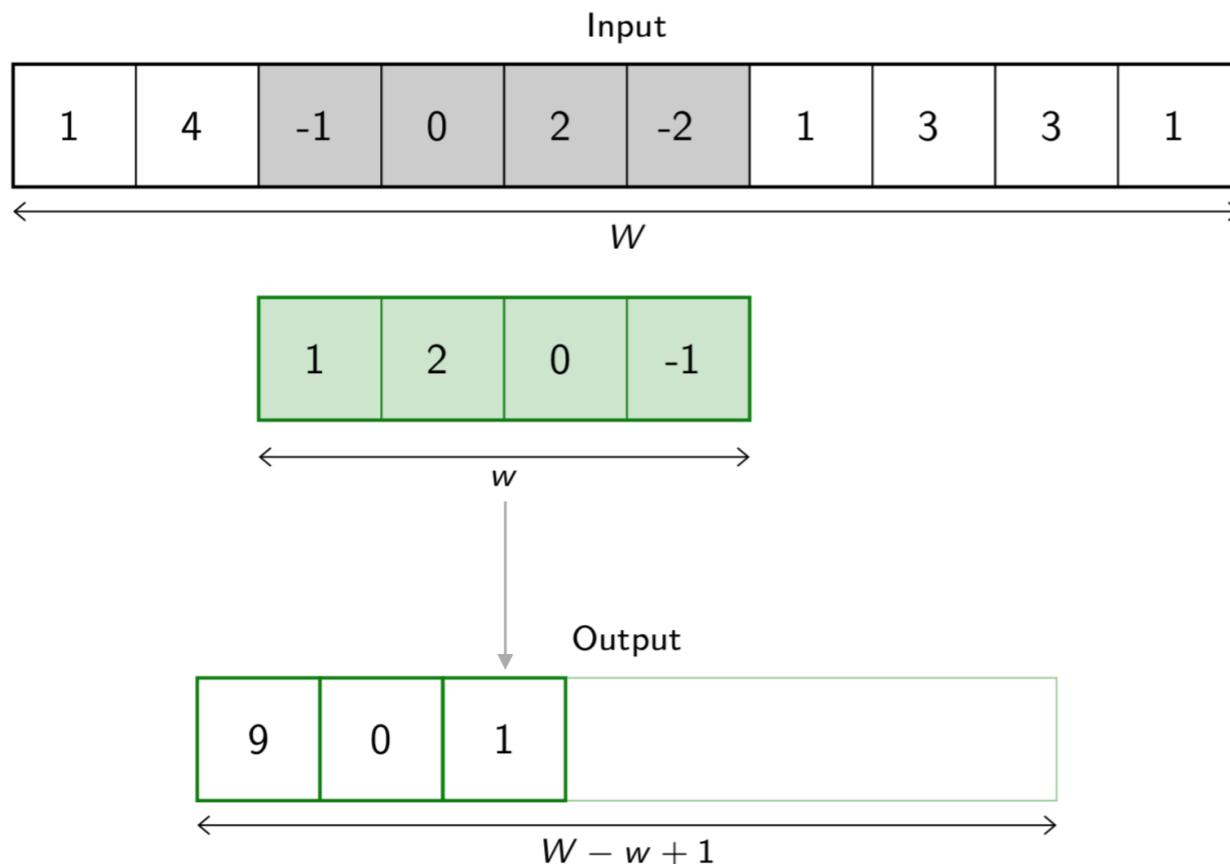
Convolution layer - 2D signals

- Notion of **several filters** producing **several activation maps**
- Let's say we have 6 $5 \times 5 \times 3$ filters, we'll get 6 maps



Convolution layer - 1D signals

- The same principle holds for 1D signals such as time series.



Formally, in 1d, given

$$x = (x_1, \dots, x_W)$$

and a “convolution kernel” (or “filter”) of width w

$$u = (u_1, \dots, u_w)$$

the convolution $x \circledast u$ is a vector of size $W - w + 1$, with

$$\begin{aligned} (x \circledast u)_i &= \sum_{j=1}^w x_{i-1+j} u_j \\ &= (x_i, \dots, x_{i+w-1}) \cdot u \end{aligned}$$

for instance

$$(1, 2, 3, 4) \circledast (3, 2) = (3 + 4, 6 + 6, 9 + 8) = (7, 12, 17).$$



The convolution here is a bit different to the “classical” convolution as indices of filter weights and signal are visited in increasing order

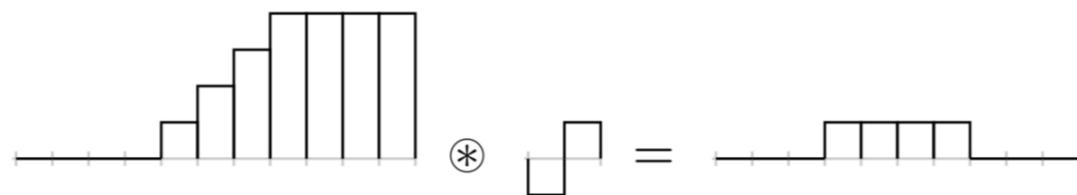
Source: François Fleuret , CAS Deep Learning, Idiap 2018

Convolution layer - 1D signals

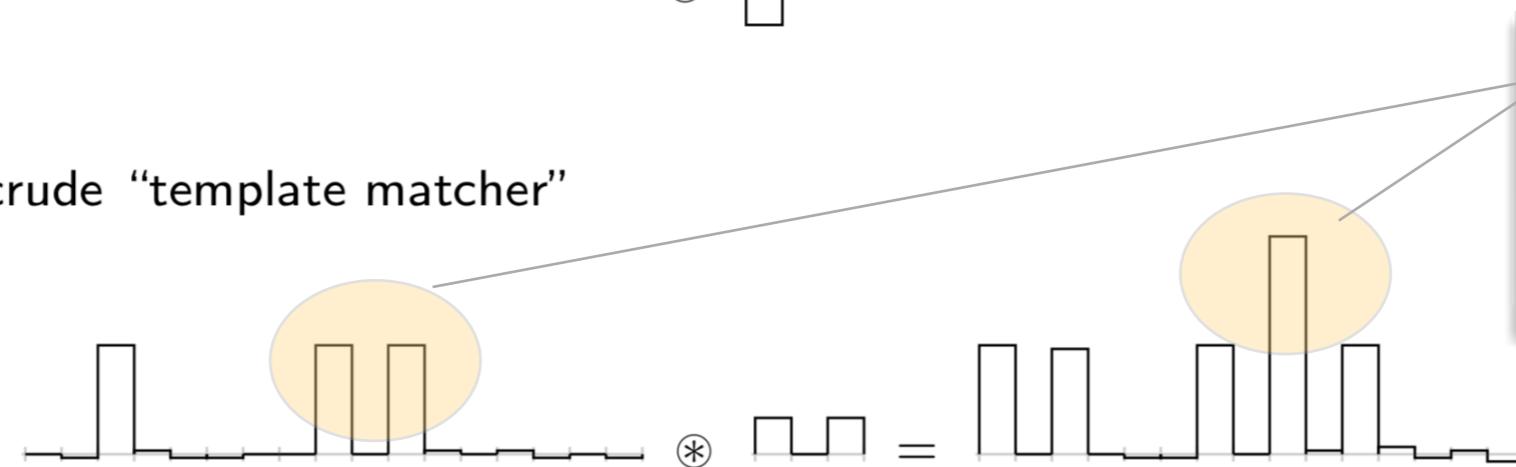
- What are doing such filters?

Convolution can implement a differential operator

$$(0, 0, 0, 0, 1, 2, 3, 4, 4, 4, 4) \circledast (-1, 1) = (0, 0, 0, 1, 1, 1, 1, 0, 0, 0).$$



or a crude “template matcher”

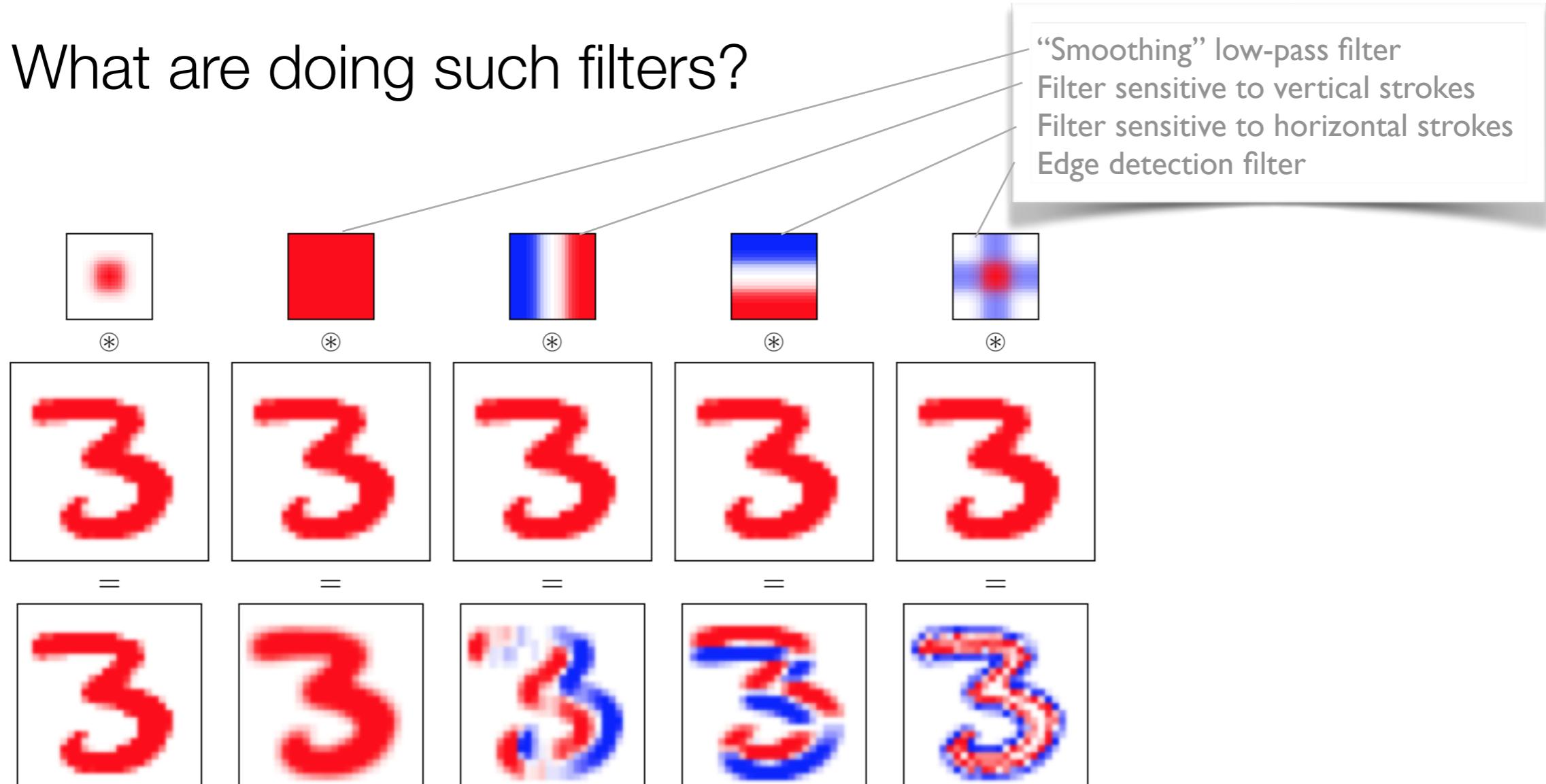


The sub-area of the input that “matches” the shape of the filter produces the highest output.

We can here really observe the **invariance in translation** and the fact that we preserve **spatial structure**, i.e. the output is also a 1D signal

Convolution layer - 2D signals

- What are doing such filters?



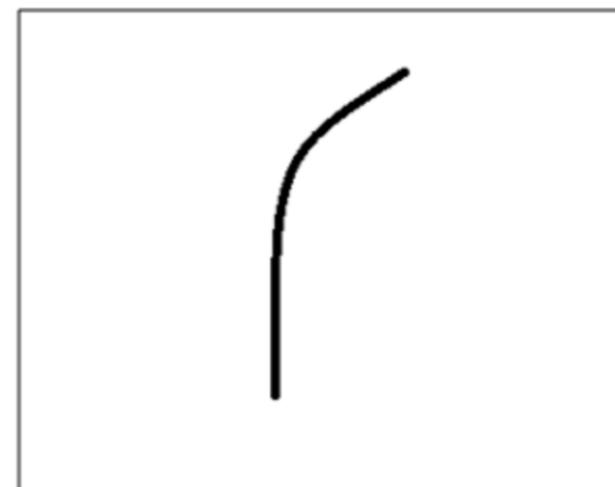
For a given problem, the “best” filters will be discovered through the training process, i.e. weights of the filters will be learnt

Convolution layer - 2D signals

- What are doing such filters?

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

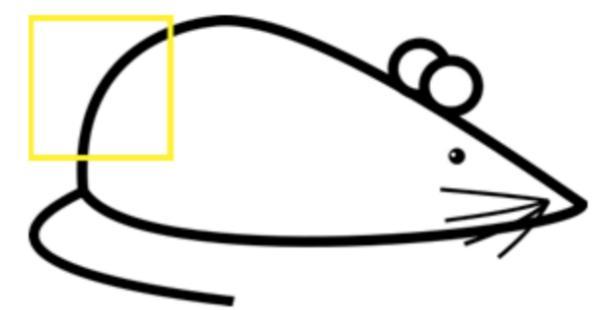
Pixel representation of filter



Visualization of a curve detector filter



Original image



Visualization of the filter on the image



Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	0	30	0
0	0	0	0	30	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter

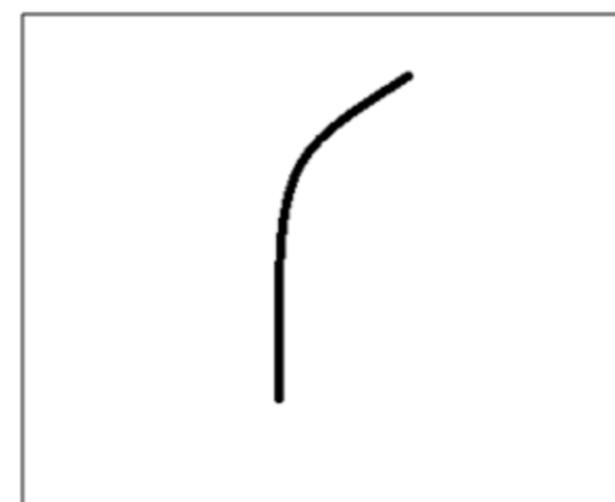
$$\text{Multiplication and Summation} = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 \text{ (A large number!)}$$

Convolution layer - 2D signals

- What are doing such filters?

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter



Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

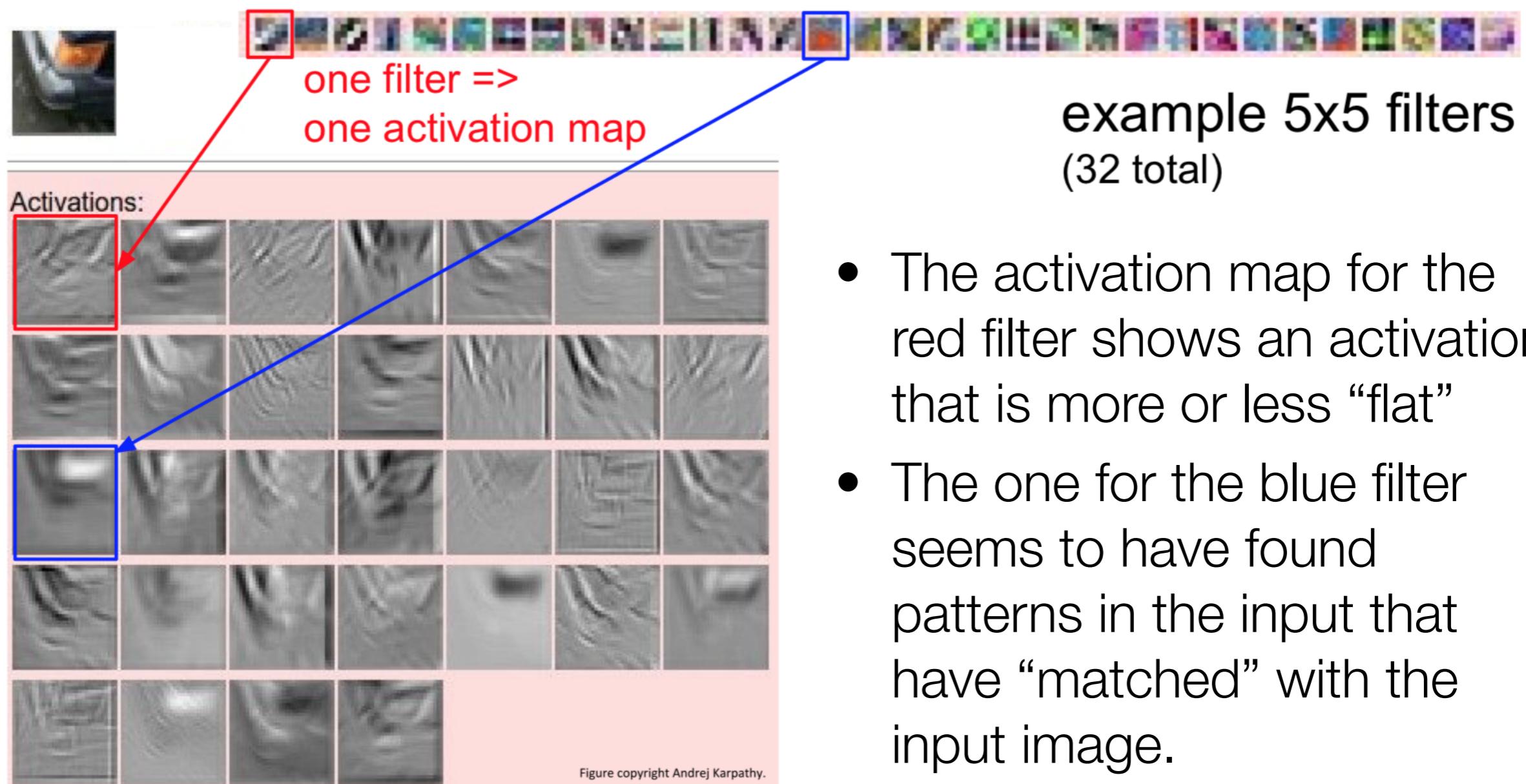
*

0	0	0	0	0	0	30	0
0	0	0	0	0	30	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	30	0	0	0	0
0	0	0	0	0	0	0	0

Pixel representation of filter

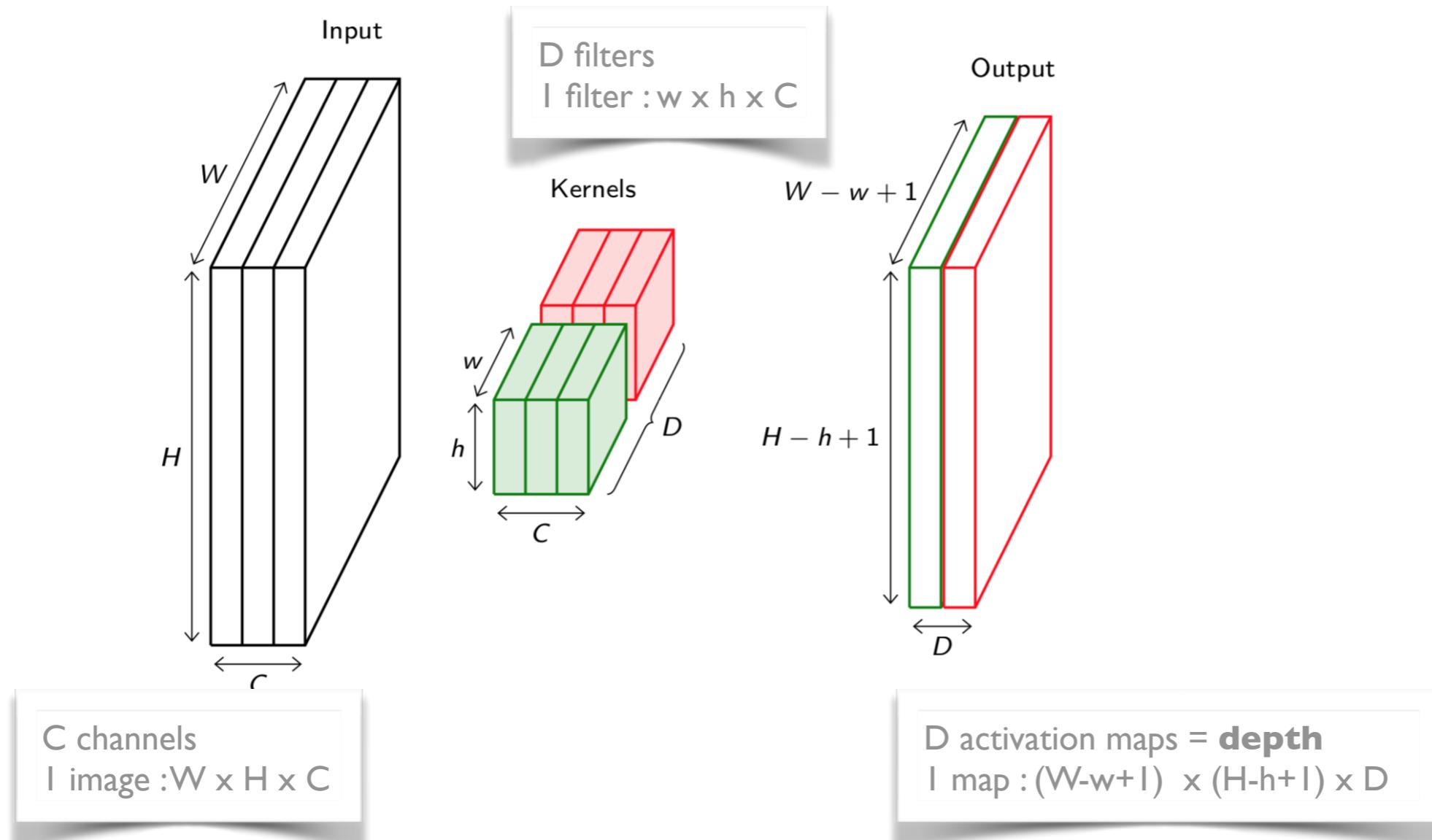
Multiplication and Summation = 0

Convolution layer - 2D signals - CIFAR10



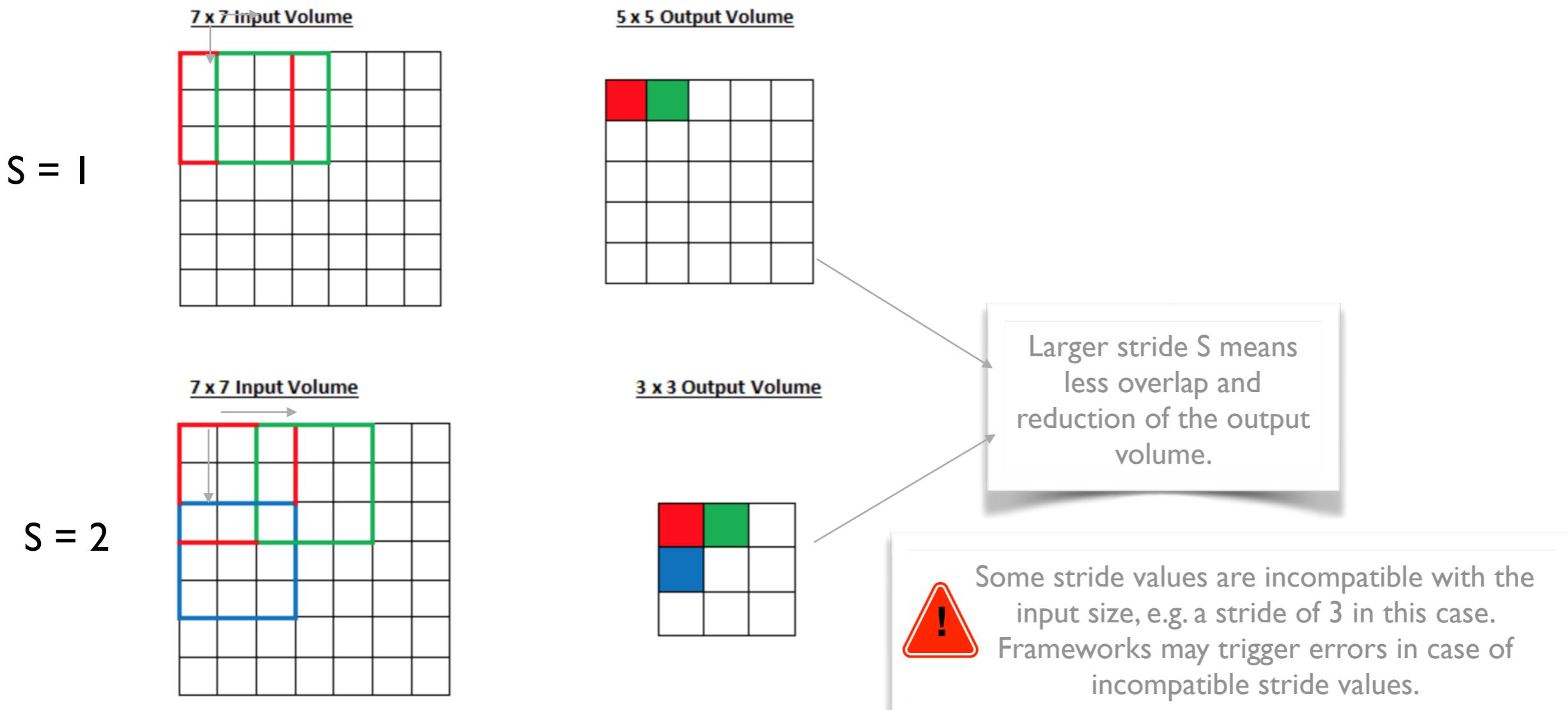
Convolution layer - 2D signals - channels

- The input images have usually 3 channels, e.g. RGB
- The filters have then the same depth as the input “volume”
- The activation maps have a depth equal to the number of filters



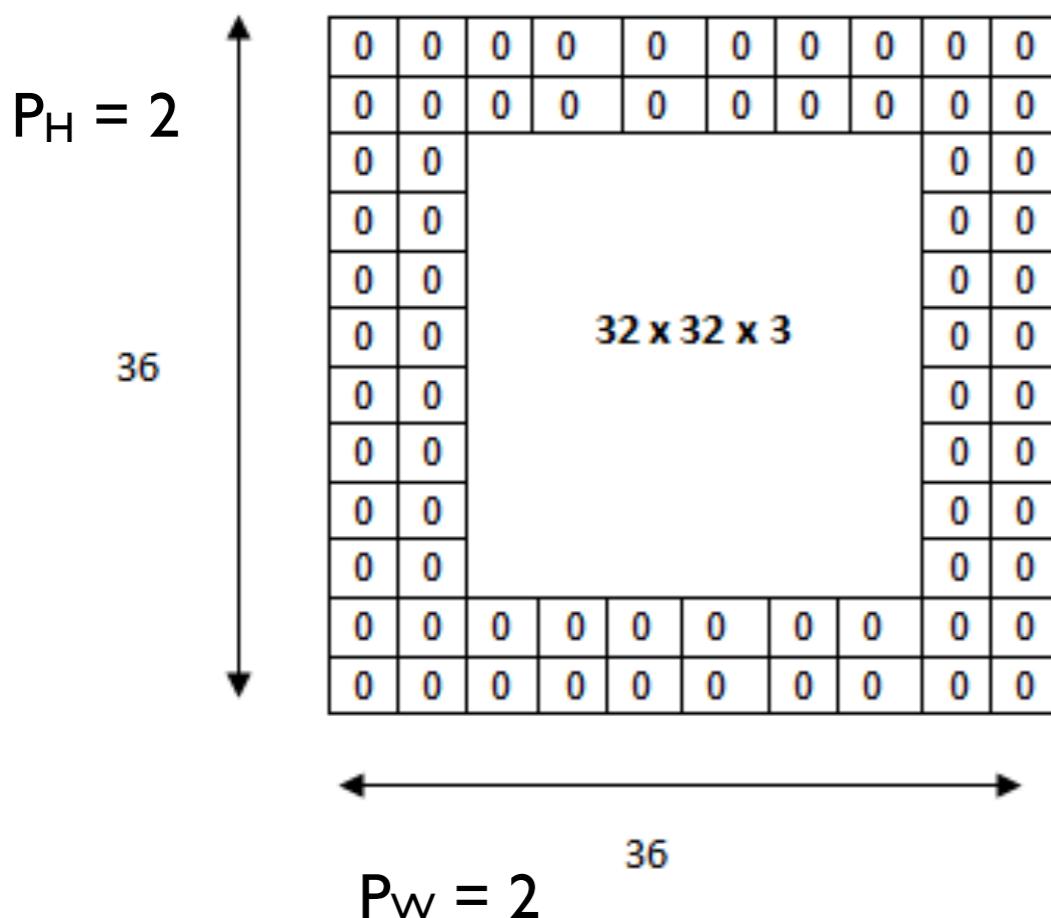
Convolution layer - stride

- The **stride S** specifies a step size when moving the filter across the signal.
- Rationale: reduce computation, reduce output size, no need of strong overlap



Convolution layer - padding

- The **padding P** specifies the size of a zeroed frame added around the input.
- Rationale:
 - Apply filter at the border of the input
 - Avoid reducing the dimension from conv layer to conv layer



The input volume is $32 \times 32 \times 3$. If we imagine two borders of zeros around the volume, this gives us a $36 \times 36 \times 3$ volume. Then, when we apply our conv layer with our three $5 \times 5 \times 3$ filters and a stride of 1, then we will also get a $32 \times 32 \times 3$ output volume.

Condition to obtain an output size equal to the input size:

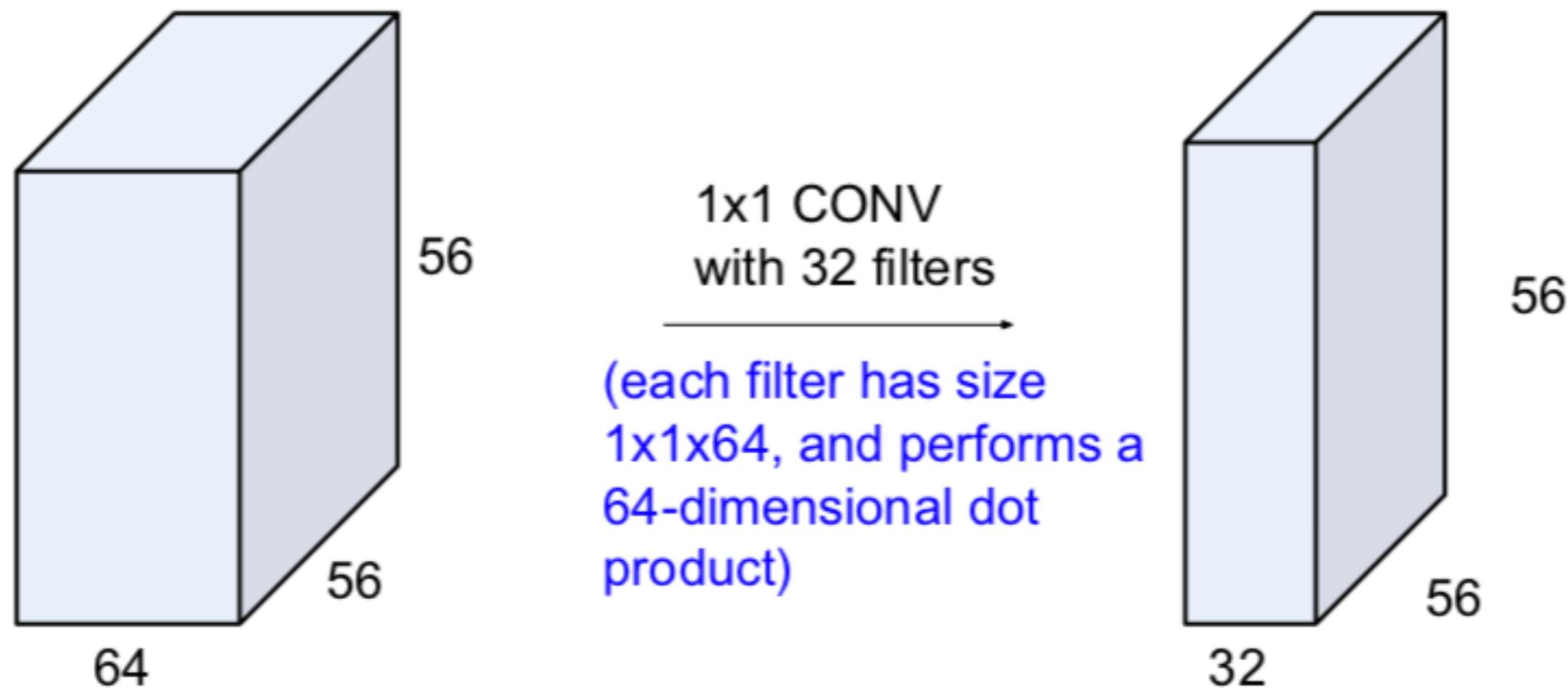
$$P_W = \frac{w-1}{2}$$
$$P_H = \frac{h-1}{2}$$

w = width of filter
h = height of filter

Common settings: K=32, 64, 128, 512
h=w=3, S=1, P=1 h=w=5, S=1, P=2
h=w=1, S=1, P=0

Convolution layer - 1x1 convolution

- 1x1 convolutions are sometimes used
 - filters apply along the depth of the input



Convolution layer - output size / # params

- The output size can be computed with :

$$O_W = \frac{W - w + 2P_W}{S_w} + 1 \quad O_H = \frac{H - h + 2P_H}{S_h} + 1$$

(W, H) = width and height of input
 (w, h) = width and height of filter
 P = padding, S = stride

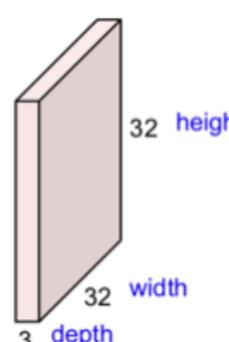
- The number of parameters can be computed with:

$$N_{param} = whCD + D$$

weights bias

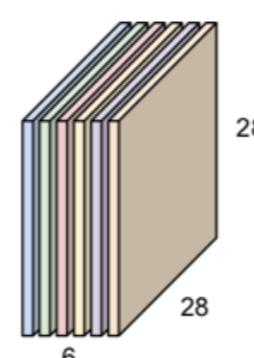
C = number of channels
D = number of filters

image 32x32x3



convolution
layer 6 filters
of 5x5x3

activation maps 28x28x6



$5 \times 5 \times 3 \times 6 + 6 =$
456 parameters

Red

Input Volume (+pad 1) (7x7x3)							Filter W0 (3x3x3)		
x[:, :, 0]							w0[:, :, 0]		
0	0	0	0	0	0	0	1	1	0
0	2	1	0	2	2	0	0	0	-1
0	2	0	1	2	1	0	1	1	-1
0	0	2	1	2	0	0	0	-1	1
0	0	0	1	0	2	0	0	-1	1
0	1	2	0	1	0	0	0	-1	1
0	0	0	0	0	0	0	0	0	1
x[:, :, 1]							w0[:, :, 1]		
0	0	0	0	0	0	0	0	0	-1
0	0	2	2	1	0	0	0	0	-1
0	0	2	2	1	1	0	0	0	-1
0	2	0	2	0	1	0	0	0	-1
0	1	1	0	2	0	0	0	1	1
0	0	0	1	2	0	0	0	1	1
0	0	0	0	0	0	0	1	1	1
x[:, :, 2]							Bias b0 (1x1x1) b0[:, :, 0]		
0	0	0	0	0	0	0	1	1	1
0	0	2	0	0	1	0	0	1	1
0	2	1	1	1	1	0	0	1	1
0	2	0	0	2	1	0	0	1	1
0	0	2	0	2	2	0	0	1	1
0	2	2	1	2	1	0	0	1	1
0	0	0	0	0	0	0	0	0	0

Green

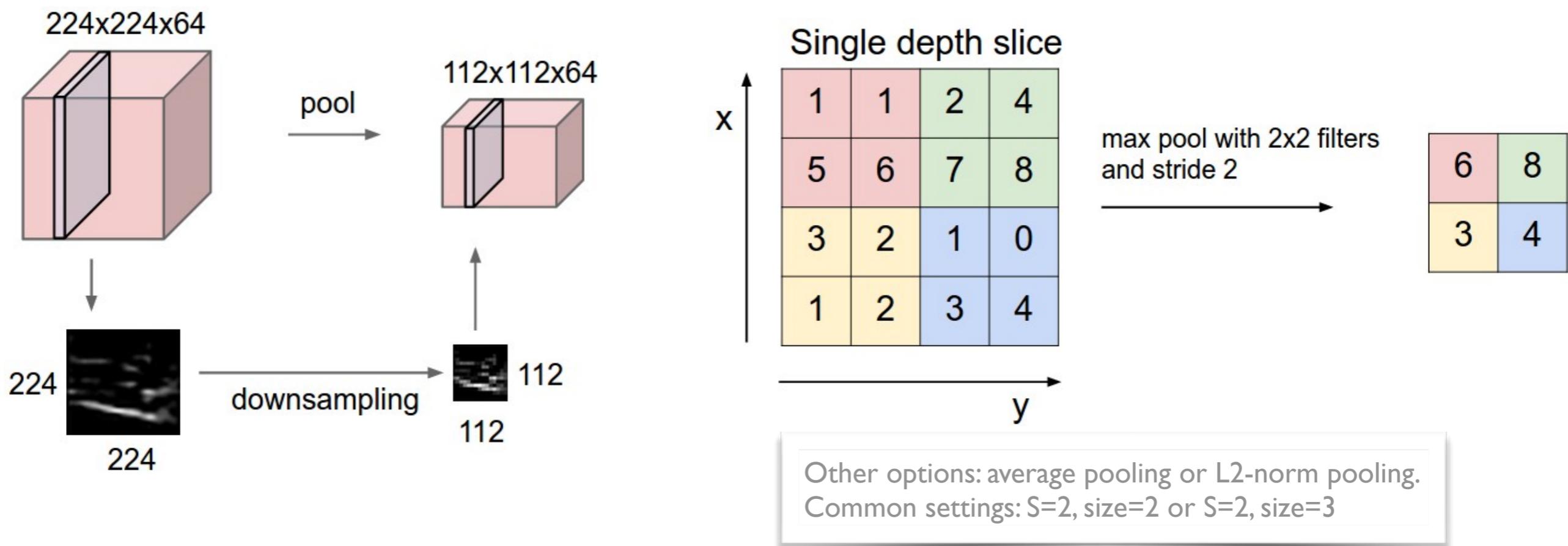
Blue

Filter W1 (3x3x3)			Output Volume (3x3x2)		
w1[:, :, 0]			o[:, :, 0]		
1	0	1	3	-3	4
1	1	1	-1	-5	4
0	1	-1	-5	0	3
w1[:, :, 1]			o[:, :, 1]		
0	-1	1	10	5	8
0	1	0	14	16	8
-1	1	0	9	9	2
w1[:, :, 2]			Bias b1 (1x1x1) b1[:, :, 0]		
0	1	1	0		
-1	1	1			
1	1	1			

toggle movement

Max pooling layer

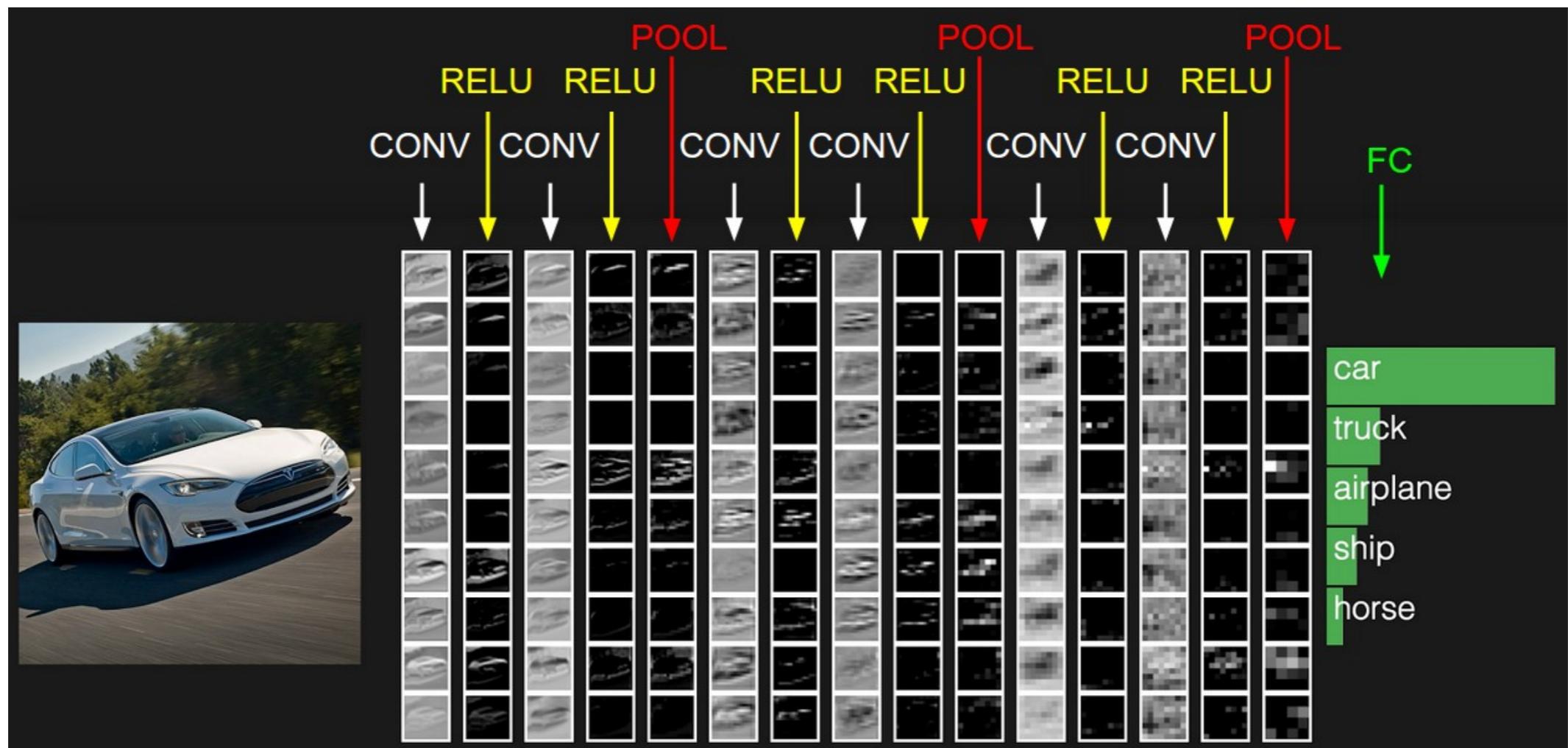
- Reduce the spatial size of the representation
 - Applies independently to every depth, defined by a stride and size
 - Rationale:
 - most significant activations are kept, brings hierarchical approach
 - reduce the amount of computation and control overfitting



Other layers

- Dropout layer
 - The layer drops out a random set of activations by setting them to zero
 - Rationale:
 - makes the network more robust, redundant
 - avoids overfitting
- Dense - fully connected layer
 - Can be viewed as a convolution layer with K filters spanning the full input space
 - Usually used at the output of the CNN architecture to “use” all the features provided by the previous layer in order to take a decision about a class label.

Full architecture for image recognition



- Typical configuration:
 - stack sequences of CONV-RELU-POOL or CONV-RELU-CONV-RELU-POOL
 - end with a fully connected “dense” layer to perform classification

Wrap-up

- **Keras** is a high-level open-source neural networks API, written in Python and capable of running on top of **TensorFlow**, **CNTK**, or **Theano**.
 - Large community in both research and industry
 - Simple unified neural network pipeline: **define**, **compile**, **fit**, **evaluate**, **use**
 - Concept of **sequential model** corresponding to a sequence of layers, one feeding the next
- **Convolutional neural networks - CNN** are deep neural network architectures composed of:
 - **Convolution layers**: principle is to convolve filters on the image with the effect to provide spatial preservation and translation invariance. A given filter is translated on **receptive fields** of the input and produces as output an **activation map**. The output of the layer is usually a “**volume**” corresponding to several activation maps.
 - **Max pooling layers**: layers that are applied to the activation maps of a previous layer in order to reduce its spatial dimension.
 - **Dense layers**: regular fully connected layers usually used as the last layers in the architecture to take classification decisions

