Computer Vision and Deep Learning

Lecture 4

Today’s agenda

- Short history of of neural networks
- Convolutional neural networks
- How to evaluate a classifier

Classifier evaluation

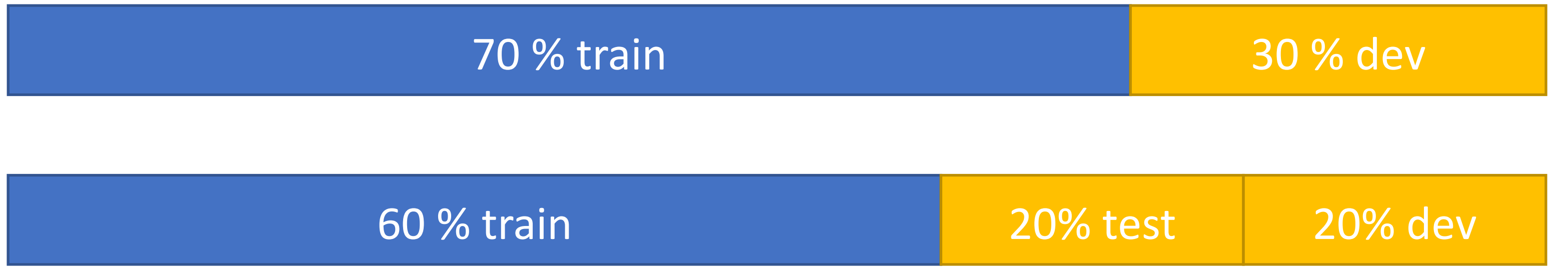
Train, dev and test sets

- Training set
  - Used to train the model, determines what the network learns
- Development (*dev*) set or validation set
  - Used to evaluate the performance of your models and determine which ones work best
- Test set
  - Used to get an unbiased estimate of the final performance of the model

Sometimes is might be ok to not have a test set (only train and dev sets)

How to split your data into train/dev/test set?

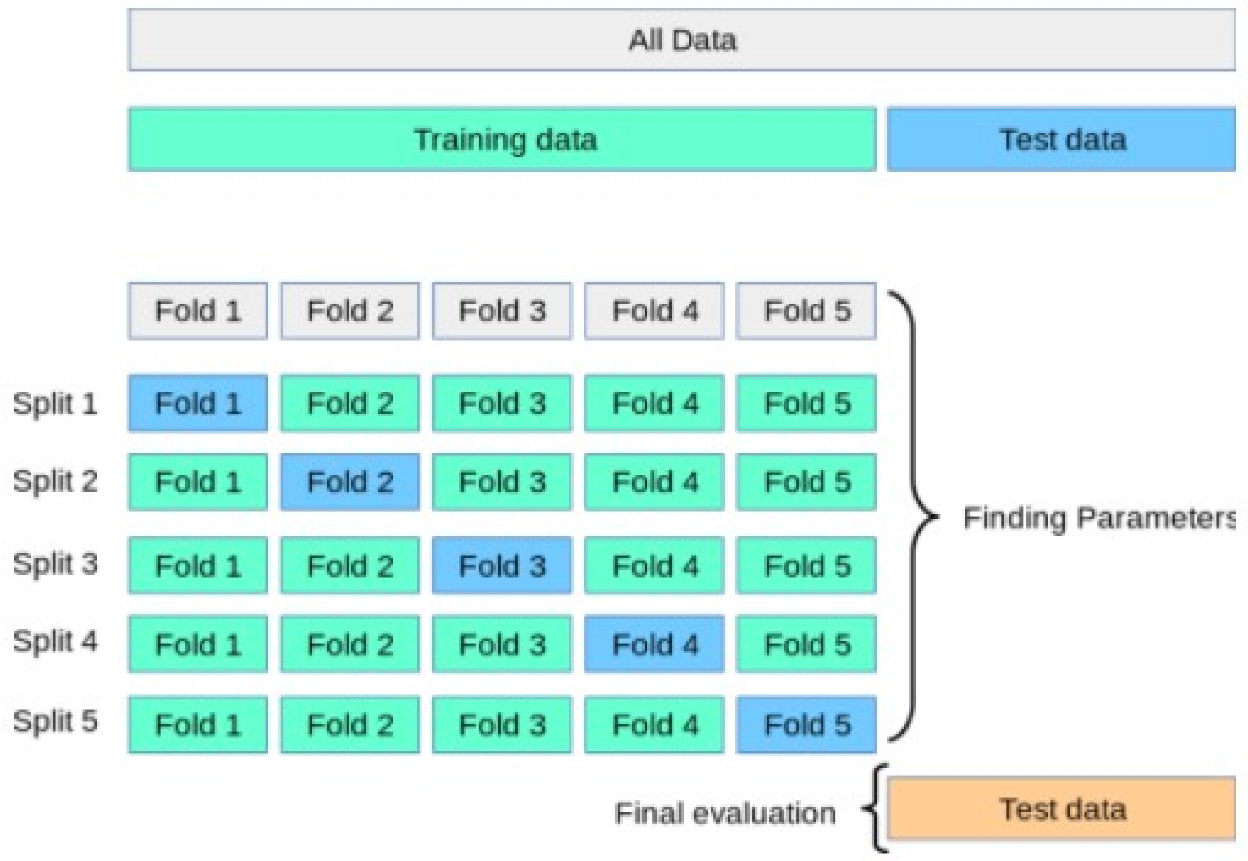
- Before deep learning, when available data was relatively limited (e.g. 10000 images)



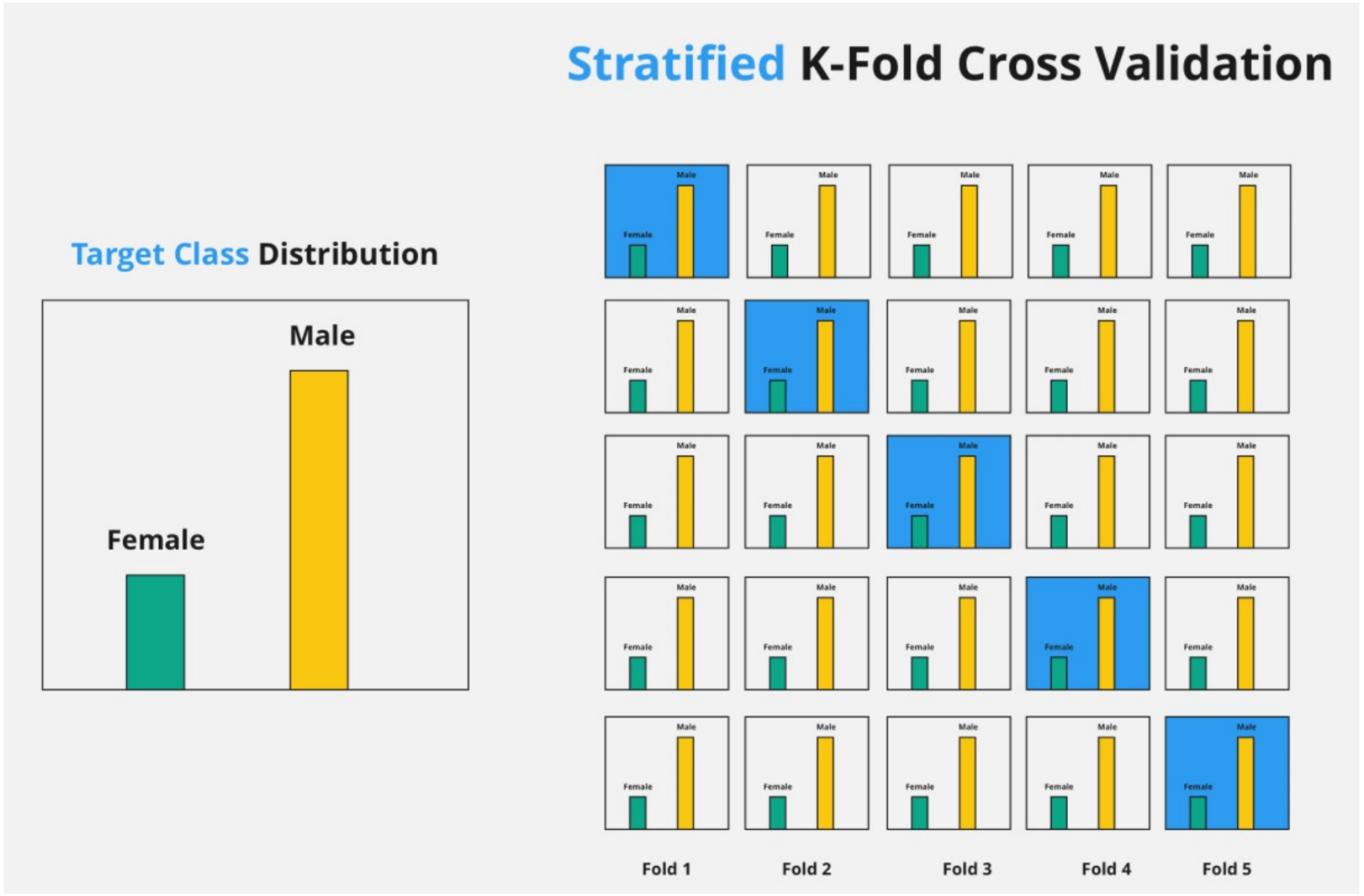
7000 train, 3000 dev

6000 train, 2000 dev, 2000 test

K-fold validation



Stratified k-fold validation



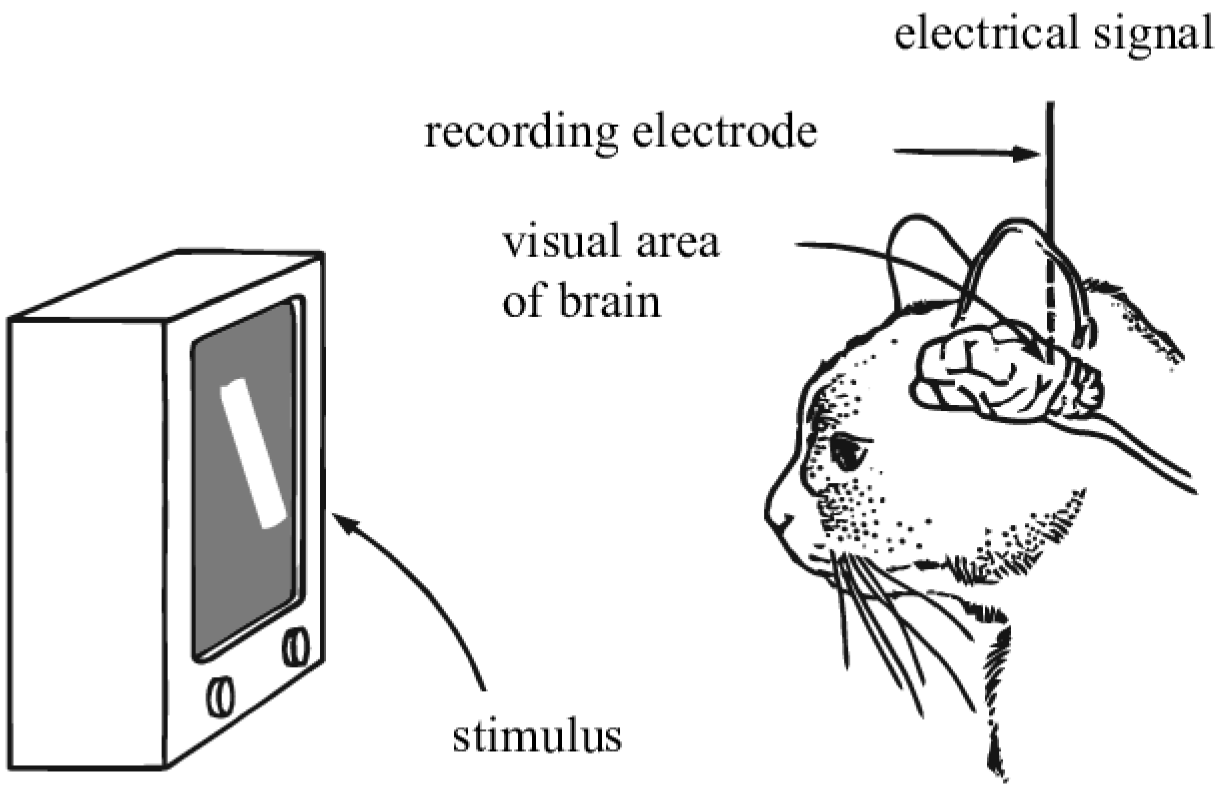
How to split your data into train/dev/test set?

- Large scale datasets
  - ImageNet > 14 million images
  - VGGFace 2 > 3.3 million images
  - JFT > 300 million images

- E.g. 1 million images
  - 980000 train set, 10000 dev set, 10000 test set -> 98% train, 1% dev, 1% test
- E.g. 4 million images
  - 3980000 train set, 10000 dev set, 10000 test set -> 99.5% train, 0.25% dev, 0.25% test

Convolutional neural networks

Understanding the visual cortex



Hubel and Wiesel, 1959

https://www.youtube.com/watch?v=IOHayh06LJ4&ab_channel=PaulLester

Nobel Prize for Physiology or Medicine in 1981:
David Hubel and Torsten Wiesel

Simple cells:
- orientation, position

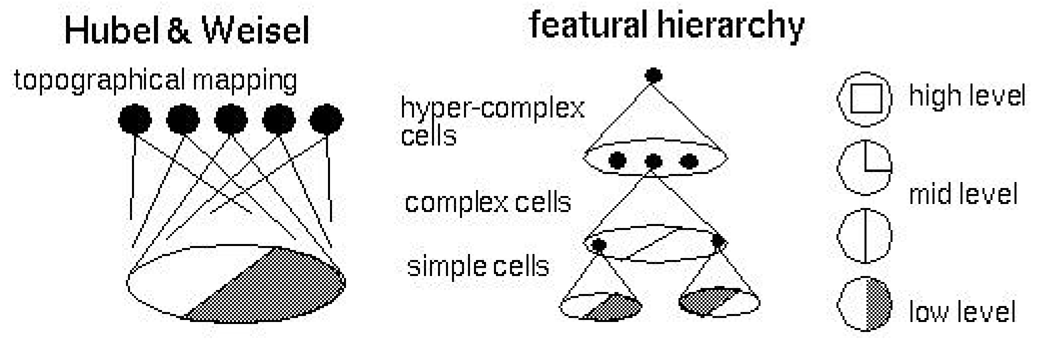
Complex cells:
- orientation, motion, direction

“Hypercomplex” cells:
- orientation, motion, direction, length

mage source: https://www.researchgate.net/publication/335707980_Understanding_Neural_Networks_via_Feature_Visualization_A_Survey

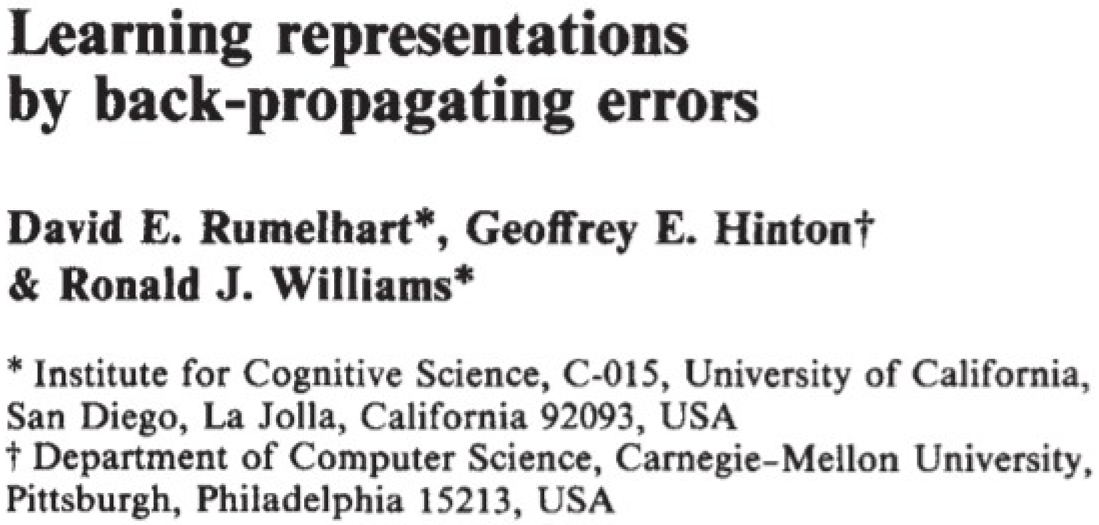
Understanding the visual cortex

- Nearby cells in the cortex represented and processed nearby regions in the visual field
- Hubel and Wiesel hypothesized that the visual cortex can be described by a hierarchical organization of simple cells that fed into complex cells which have more complicated activations and can form higher level representations



https://peterkim95.github.io/jekyll/update/2017/08/29/introspectiveness-of-nn/

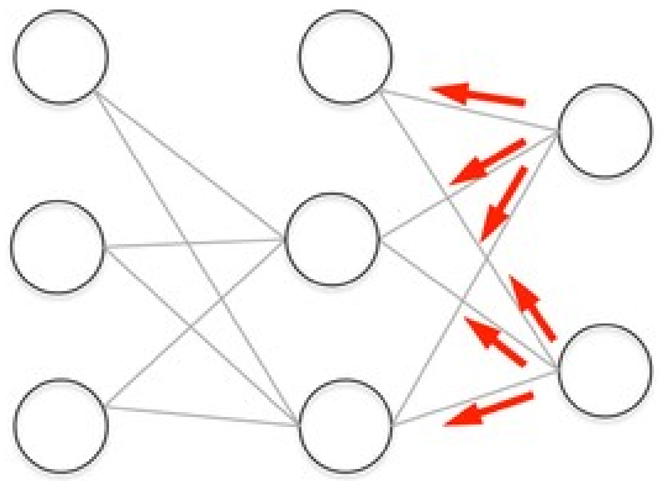
Backpropagation, 1986



Learning representations by back-propagating errors

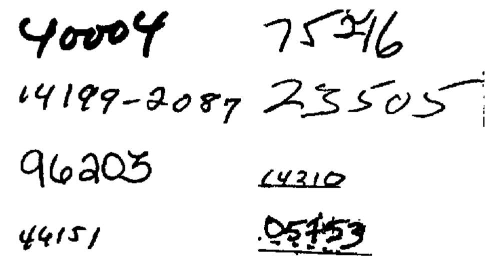
David E. Rumelhart*, Geoffrey E. Hinton† & Ronald J. Williams*

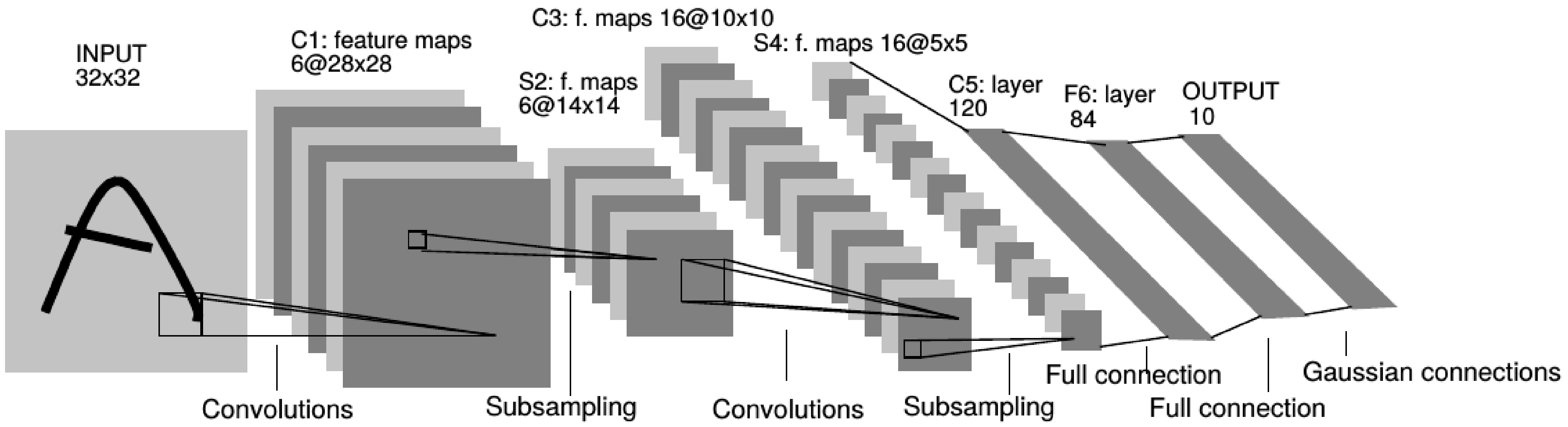
* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA
† Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Philadelphia 15213, USA



Gradient-Based Learning Applied to Document Recognition, *Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner*, ***1998***

1989 – the original form of LeNet

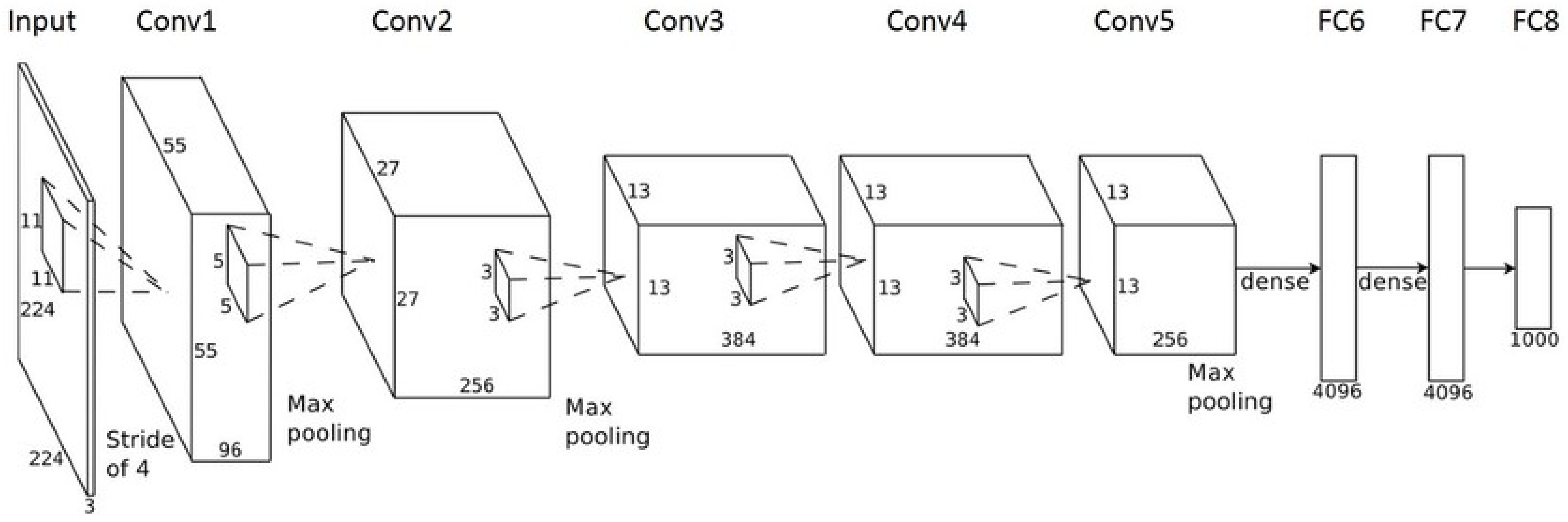




Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition
Dan Ciresan, 2010

- One of the very fist implementations of GPU Neural nets
  - forward and backward pass implemented of an artificial neural network (up to 9 layers) implemented on an NVIDIA GTX 280 graphic processor

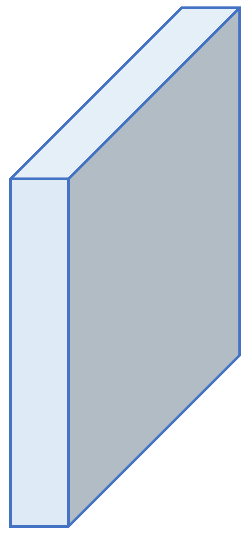
2012 ImageNet Classification with Deep Convolutional Neural Networks, *Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton*

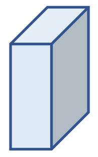


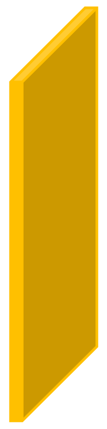
Convolutional neural networks

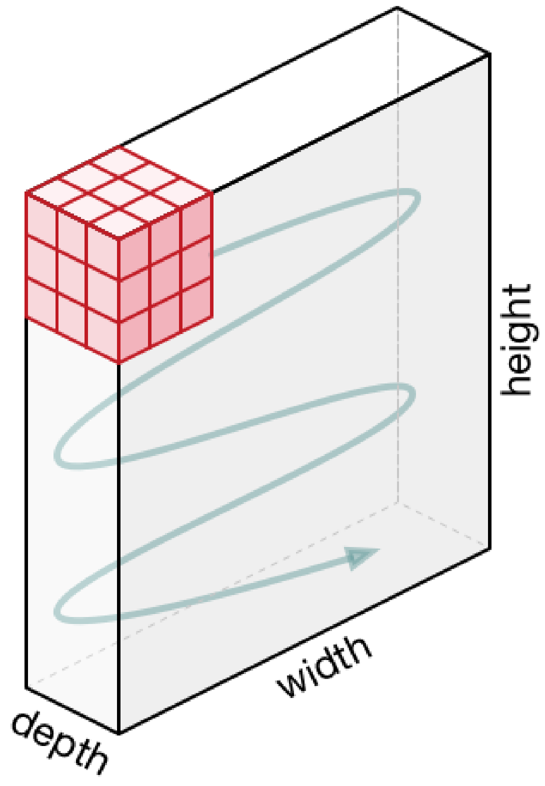
Convolutional layers

- Preserve the spatial information
- Convolve the filter over the entire input volume
    - The filter has the **same depth** as the input


Input volume
32x32x3

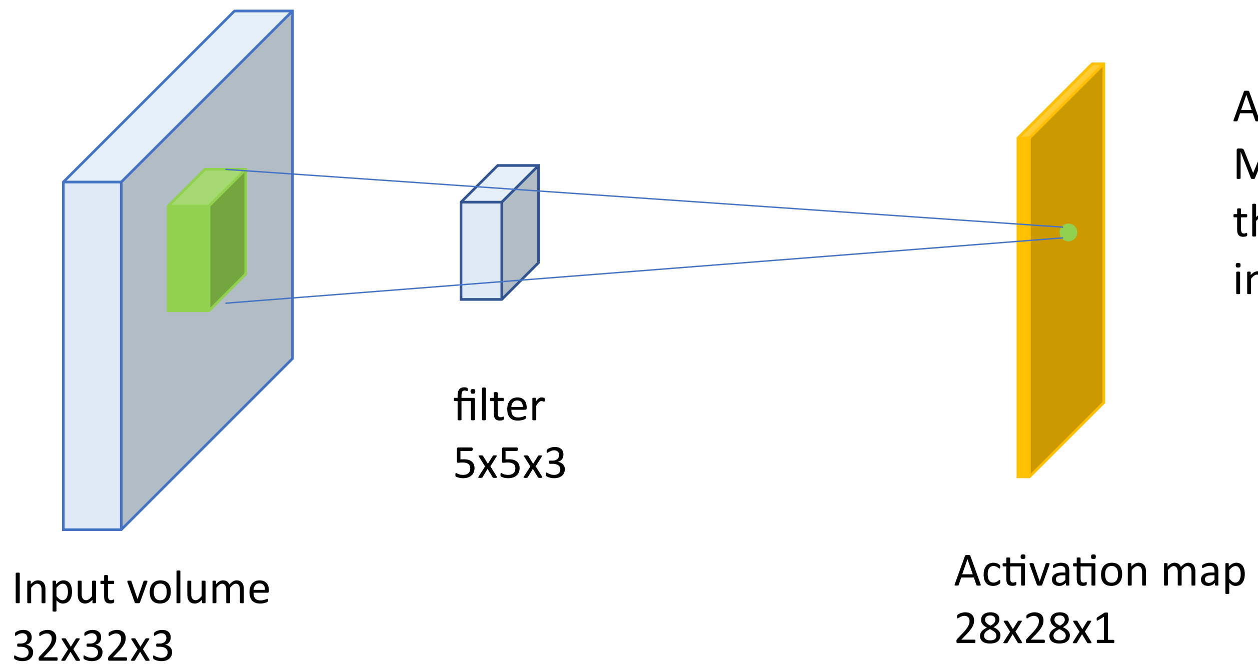

filter
5x5x3


Activation map
28x28x1



Convolutional layers

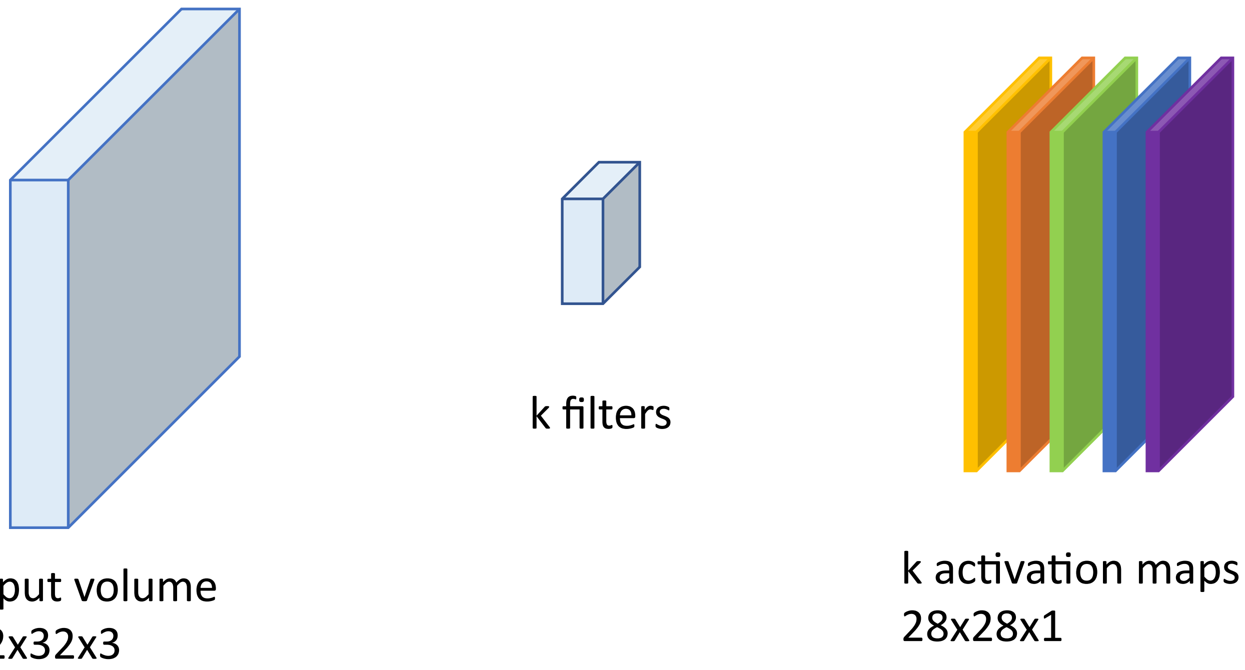
- Preserve the spatial information
- Convolve the filter over the entire input volume
  - The filter has the **same depth** as the input



At each position in the input volume:
Multiply (element-wise, across all channels) the filter and a small patch (5x5x3) in this input volume and add a bias term

Convolutional layers

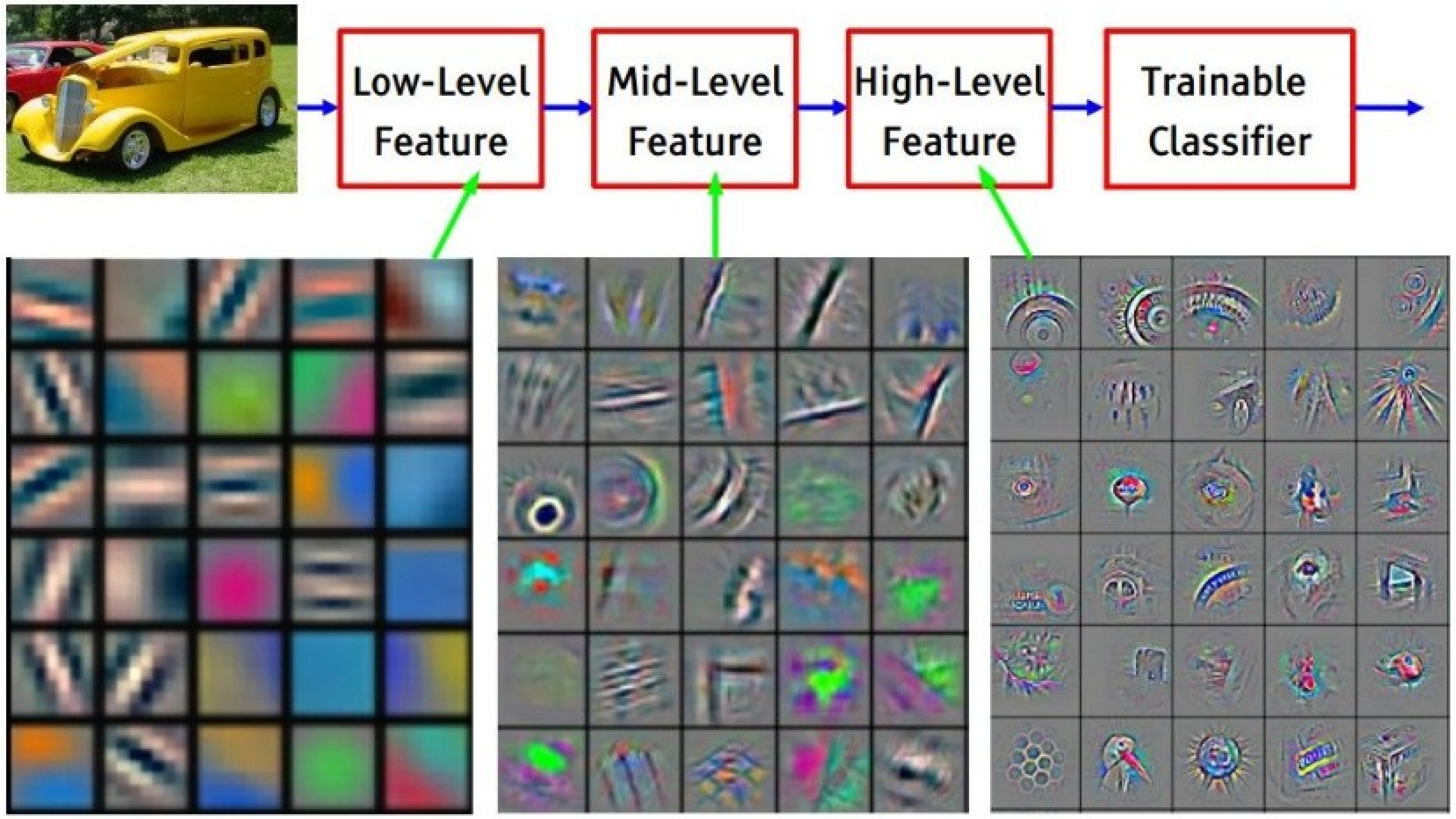
- Preserve the spatial information
- Convolve the filter over the entire input volume
  - The filter has the **same depth** as the input



Convolutional layers

- Neurons in an activation map:
    - Each neuron is connected to a small region in the input
    - All of neurons in the activation map share parameters
- Receptive field of a neuron
- k filters → k different neurons all looking at the same region in the input volume

k activation maps
28x28x1



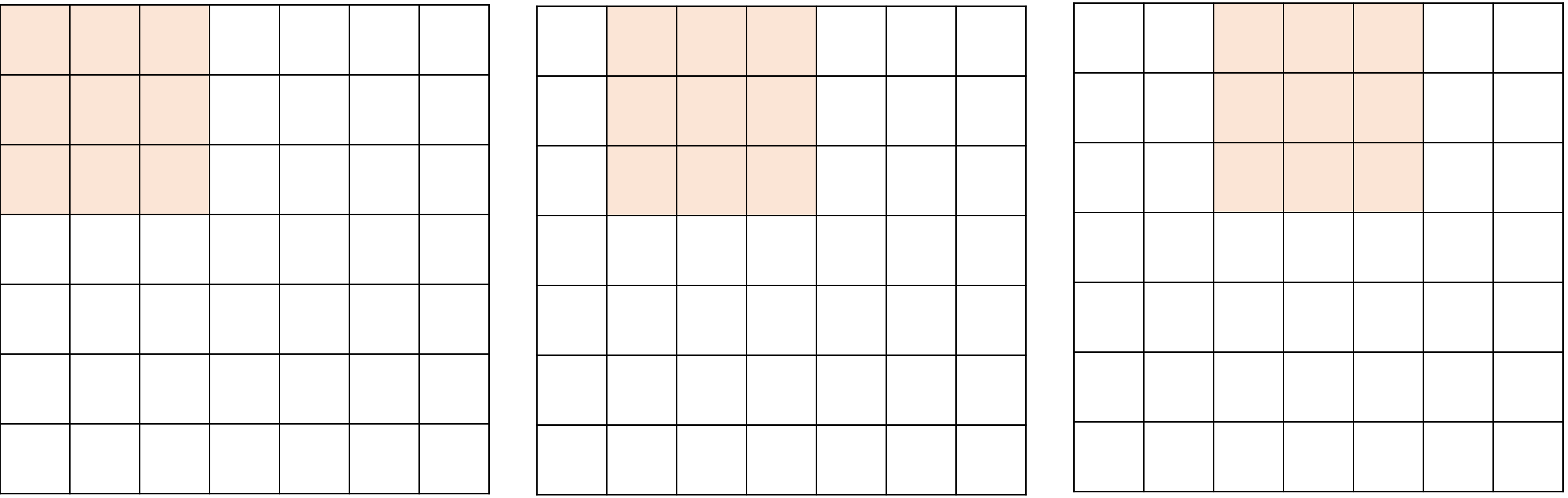
https://medium.com/@chriskevin_80184/feature-maps-ee8e11a71f9e

https://www.youtube.com/watch?v=AgkfIQ4IGaM

Convolutional layers

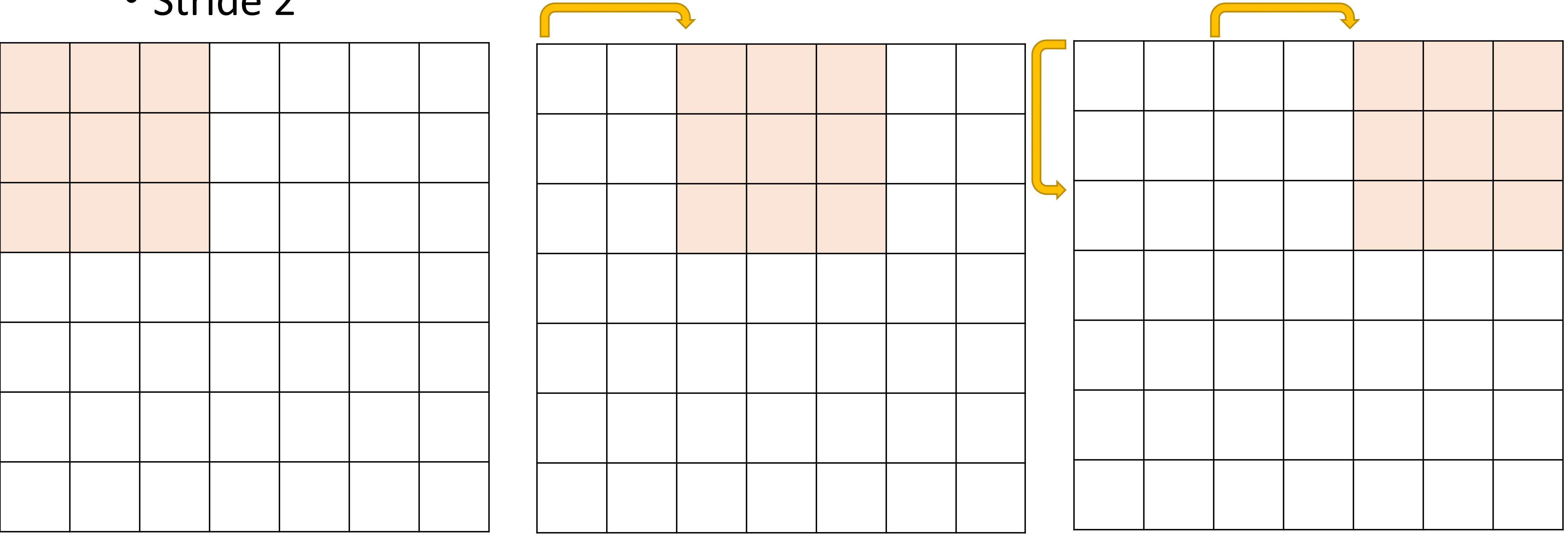
Stride

- Stride – the amount by which the filter shifts
- Stride 1



Convolutional layers
Stride

- Stride – the amount by which the filter shifts
- Stride 2



Convolutional layers
Padding

0	0	0	0	0	0	0
0						0
0						0
0						0
0						0
0						0
0	0	0	0	0	0	0

padding: 1

Convolve an input of spatial size 7x7 with a 3x3 filter : output spatial size?
Convolve an input of spatial size 7x7 with a 3x3 filter, and applying 0 padding to the input: output spatial size?

To preserve size spatially:
CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2

Why convolutions

- Parameter sharing
  - A feature detector that's useful in one part of an image is probably useful in other parts of the image
  - Translation invariance
- Sparsity of connections
  - The output volume depends only on a small (filter size) subset of the input volume

Convolutional layers

(, ,) (, ,)

= filter depth

– input width, – output width

– input height, – input height

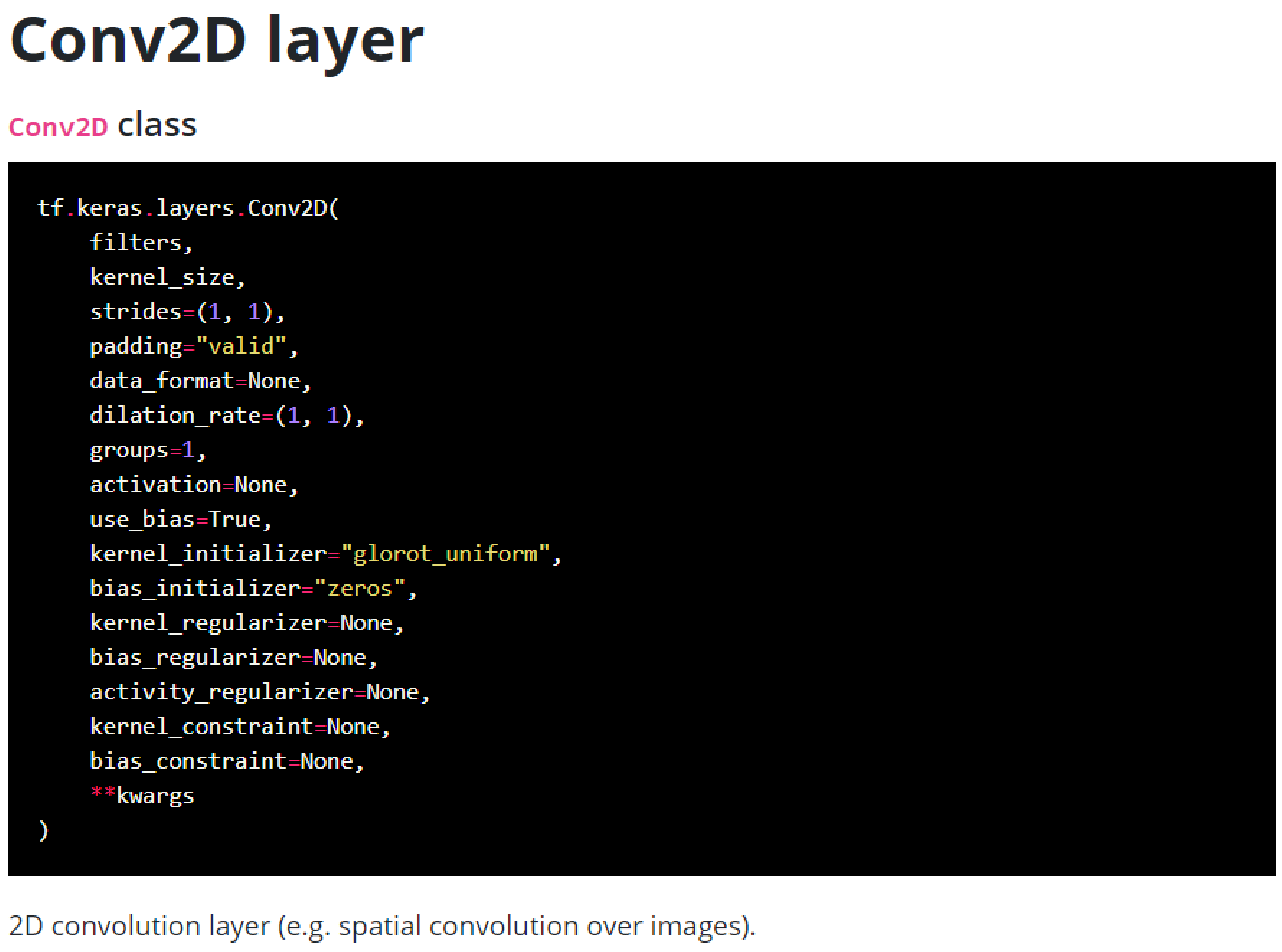
F – filter size

P – padding

S - stride

$$W_o = \frac{W_I - F + 2P}{S} + 1$$

$$H_o = \frac{H_I - F + 2P}{S} + 1$$



Conv2D layer

Conv2D class

```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

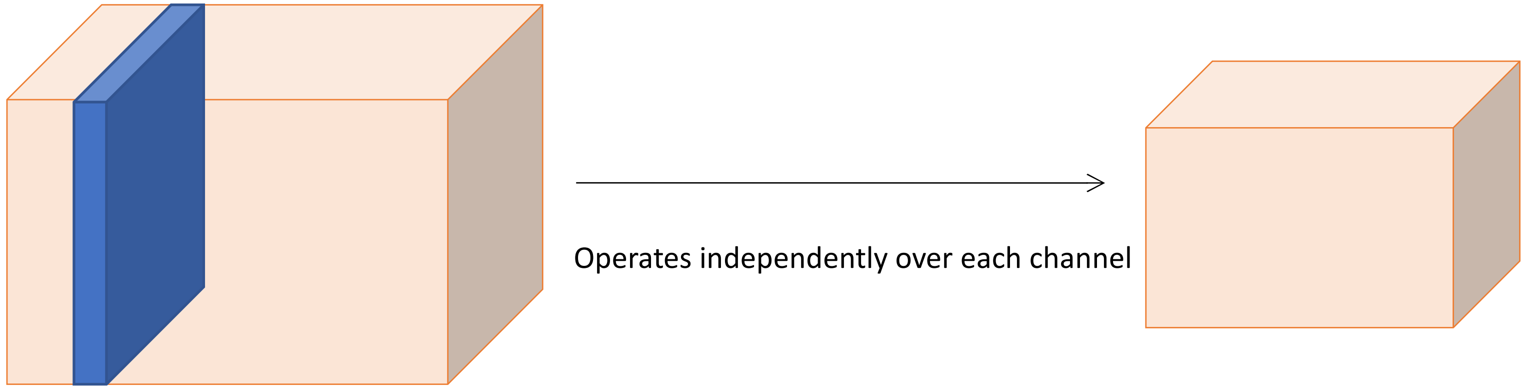
2D convolution layer (e.g. spatial convolution over images).

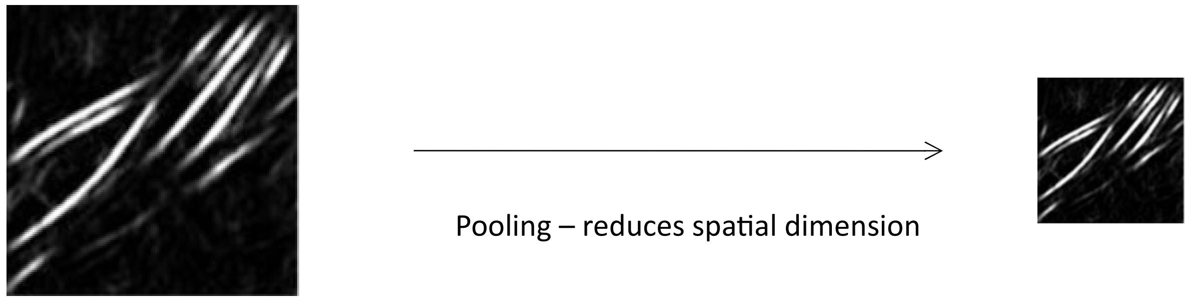
Padding: “same” or “valid”

Pooling layers

- Operate **independently** over each channel in the input
- Reduces the input size, creating smaller (and more manageable) representations
- Common pooling layers:
  - max pooling: takes the maximum value within each “patch” in the feature map
  - average pooling: takes the average value of each “patch” in the feature map
- It **does not contain** any **learnable weights**

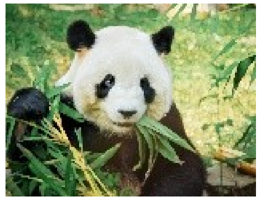
Pooling layers



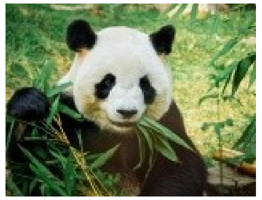


Pooling layers

Examples



Max pooling



Avg pooling

Pooling layers

Example: max pooling layer, F 2, stride 1

1	3	6	6
20	9	8	4
2	1	4	5
1	12	13	10

20	9	8
20	9	8
12	13	13

Pooling layers

Example: max pooling layer, F 2, stride 2

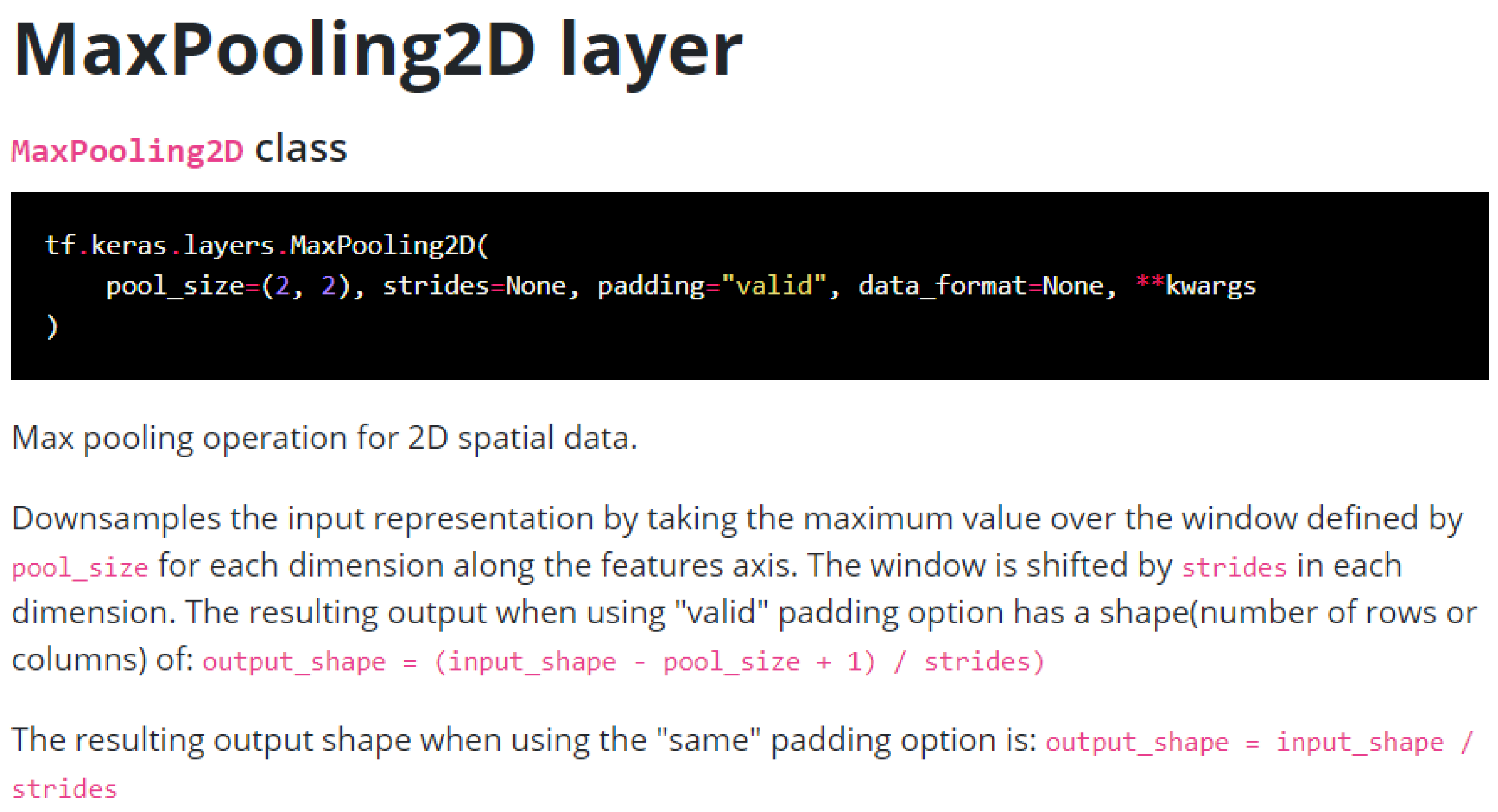
1	3	6	6
20	9	8	4
2	1	4	5
1	12	13	10

20	8
12	13

Pooling layer

- Parameters
  - Filter size (spatial extent): F
  - Stride: S
- Input: $W_I \times H_I \times D$
- Output: $W_O \times H_i \times D$
- It has no learnable parameters

$$W_o = \frac{W_I - F}{S} + 1 \qquad H_o = \frac{H_I - F}{S} + 1$$



MaxPooling2D layer

MaxPooling2D class

```
tf.keras.layers.MaxPooling2D(
    pool_size=(2, 2), strides=None, padding="valid", data_format=None, **kwargs
)
```

Max pooling operation for 2D spatial data.

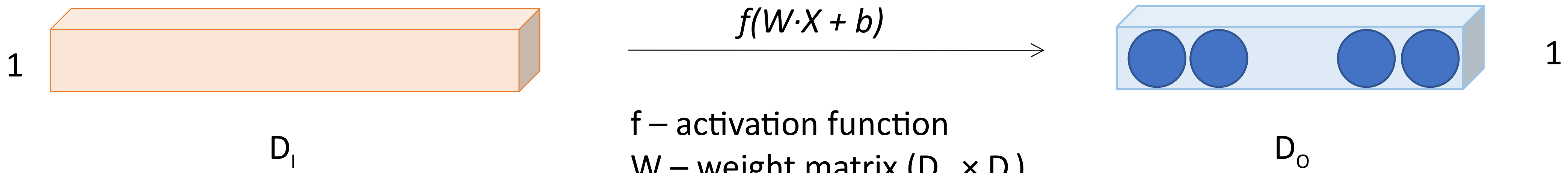
Downsamples the input representation by taking the maximum value over the window defined by `pool_size` for each dimension along the features axis. The window is shifted by `strides` in each dimension. The resulting output when using "valid" padding option has a shape(number of rows or columns) of: `output_shape = (input_shape - pool_size + 1) / strides)`

The resulting output shape when using the "same" padding option is: `output_shape = input_shape / strides`

AveragePooling2D

Fully connected layers

- They don’t preserve spatial information
  - Linear unit followed by a linearity
- Just in regular NN, contain several neurons that are connected to the entire input volume
  - Each neuron “sees” the entire input volume



f – activation function
W – weight matrix ($D_O \times D_I$)
b – bias vector

- If we have an input volume of 27x27x5, what will be size of this volume if we apply a padding of 2?
- How many parameters (including the bias) does a convolutional layer with 10 filters of size 5x5 have?
- How many parameters (including the bias) does a convolutional layer with 10 filters of size 5x5 have if the input size is 32x32x3? What if we use a stride of 2?
- Given an input volume that is 63x63x16 that is convolved with 32 filters that are each 7x7, using a stride of 2 and no padding. What is the output volume?
- Given a RGB image of size 300x300, and you use a classical neural network with the first hidden layer of 100 neurons (each one fully connected to the input). How many parameters does this hidden layer have?

Dense layer

Dense class

```
tf.keras.layers.Dense(
    units,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

Just your regular densely-connected NN layer.

Typical neural network architecture

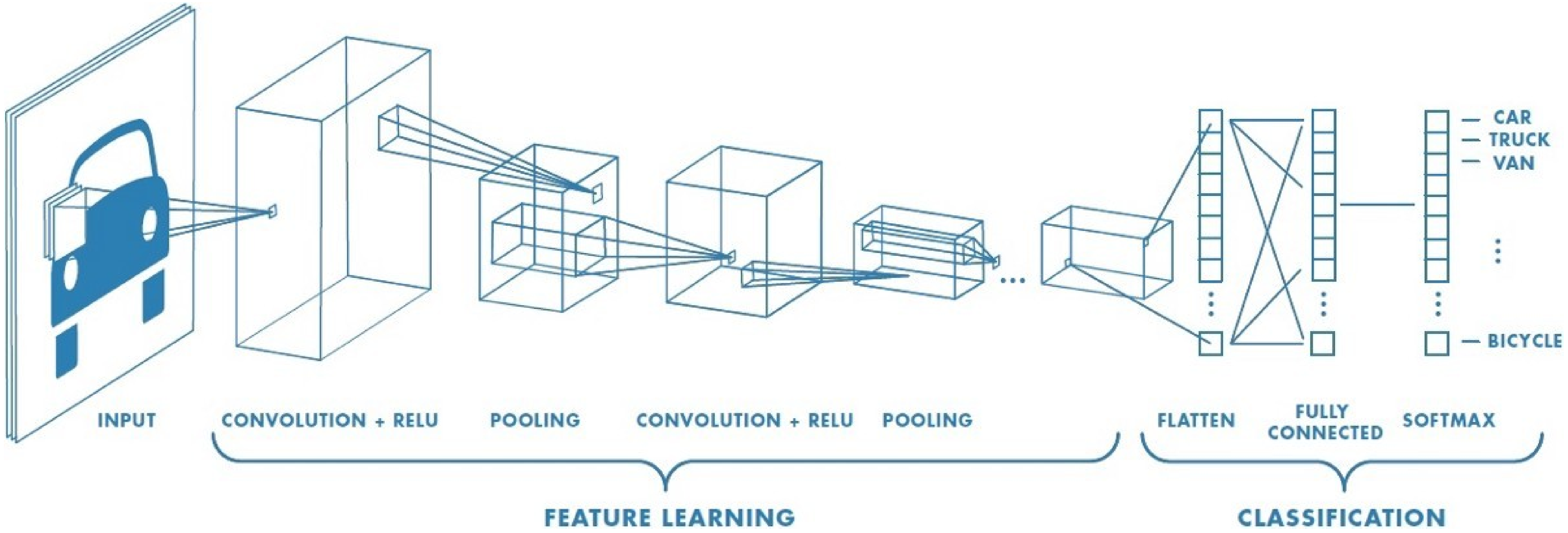
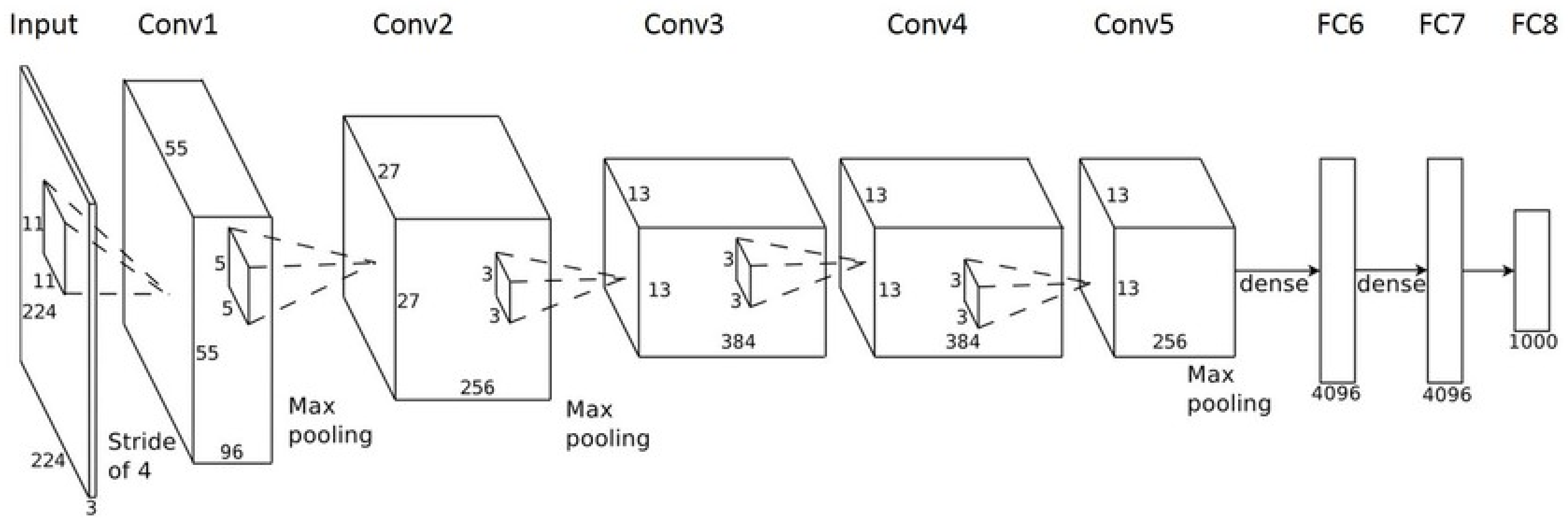


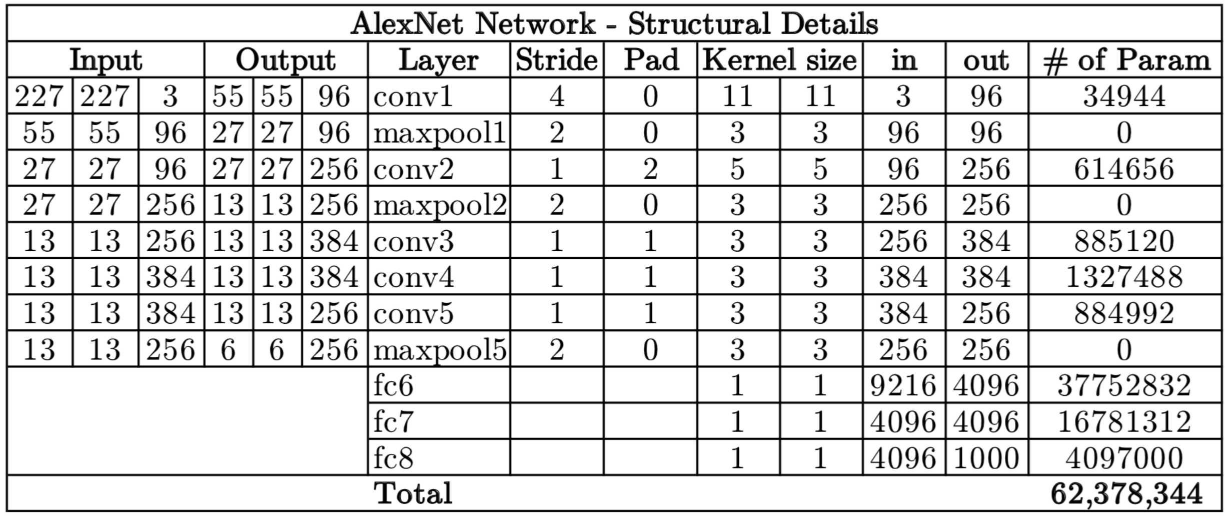
Image source: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

Typical neural network architecture

- Several CONV, POOL and FC layers stacked together
    - [CONV-ReLU-POOL]*N – [FC]*K – softmax
    - Recent neural networks change this paradigm
- The trend is to reduce the filter sizes and to increase the depth of the networks
- Another trend is to avoid using POOL and FC layers, and use only CONV layers

Alexnet example





AlexNet Network - Structural Details

Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
						Total							62,378,344

“AlexNet input starts with 227 by 227 by 3 images. And if you read the paper, the paper refers to 224 by 224 by 3 images. But if you look at the numbers, I think that the numbers make sense only of actually 227 by 227.”

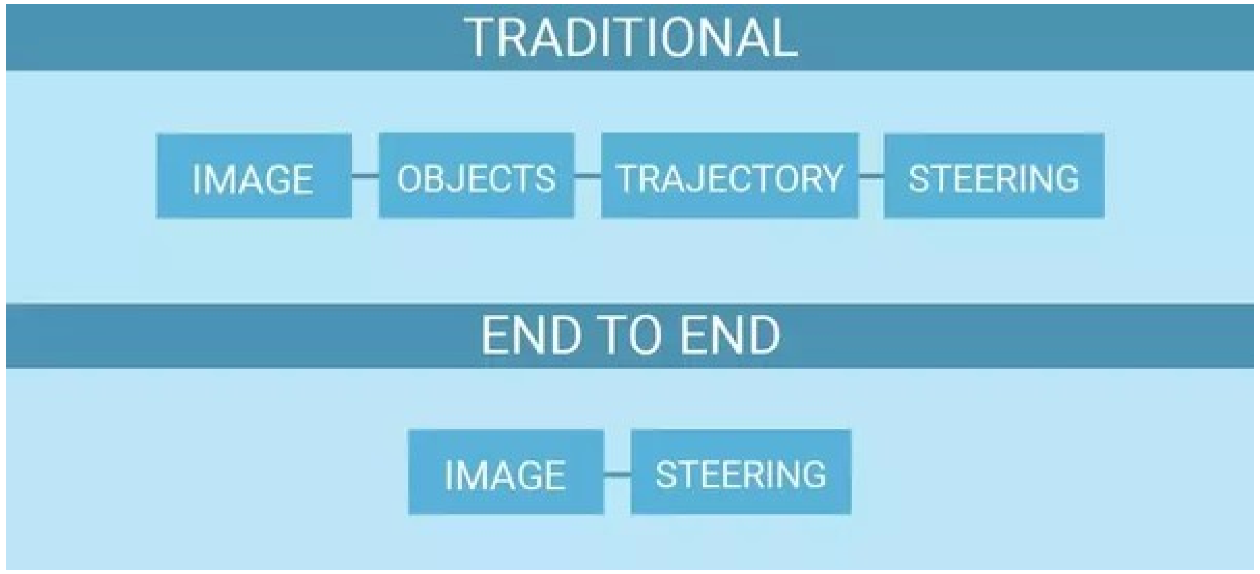
Playground

https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html

Learning from multiple tasks

- Transfer learning
  - Task A and task B have the same input
  - More data for task A than for task B
  - Low level features from task A could be helpful for learning task B
- Multi task learning
  - Training on a set of tasks that could benefit from having shared low level features
  - Amount of data for each task is quite similar. Can train a big enough network to do well on all the tasks.

End to end deep learning



Advantages:

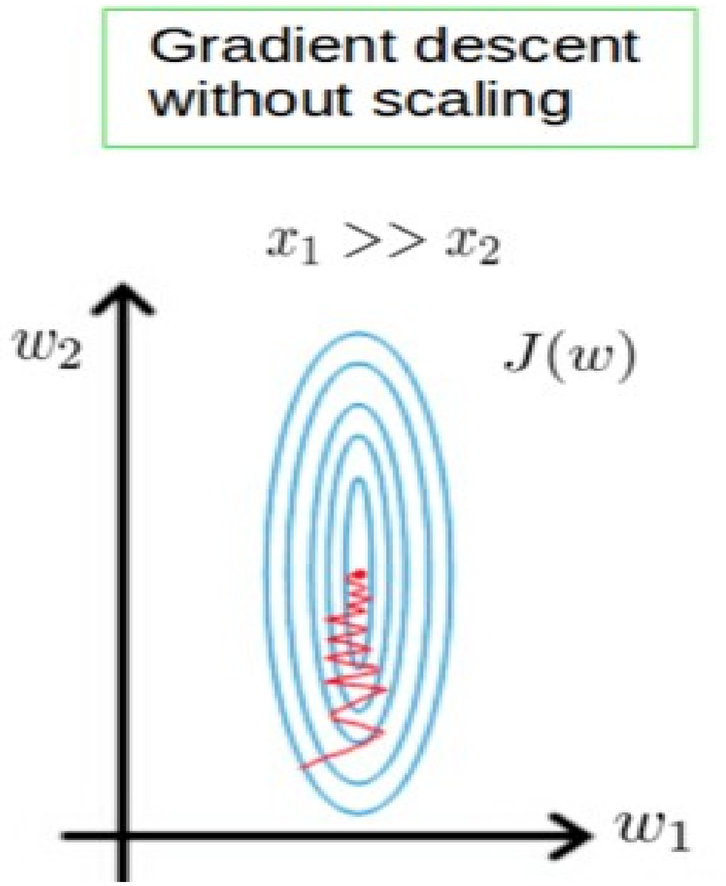
- Lets the data speak
- No handcrafted features

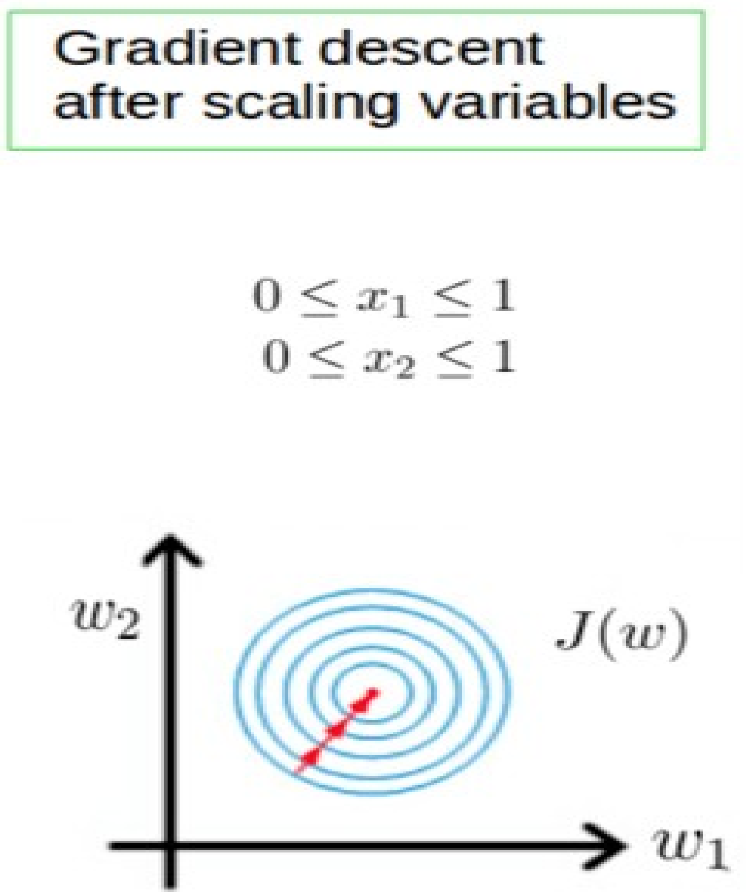
Disadvantages

- Needs huge amount of data
- Excludes potentially useful hand-designed components

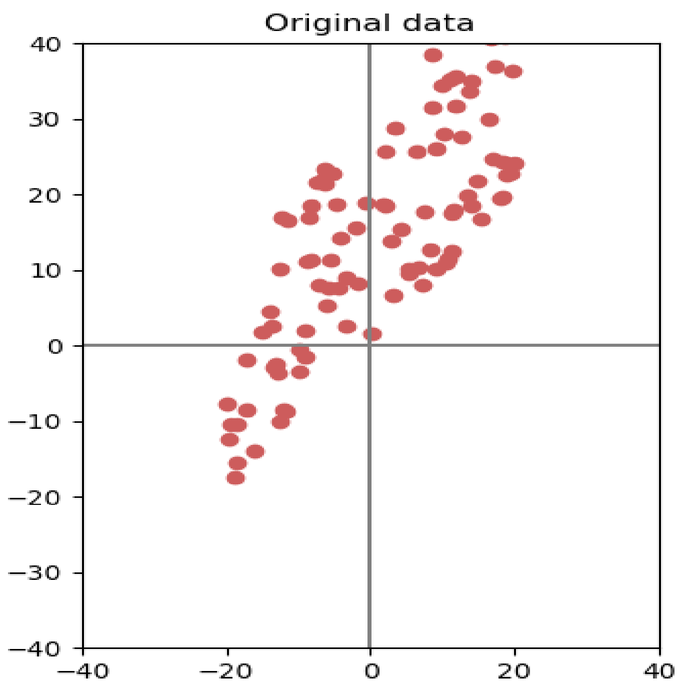
Data pre-processing

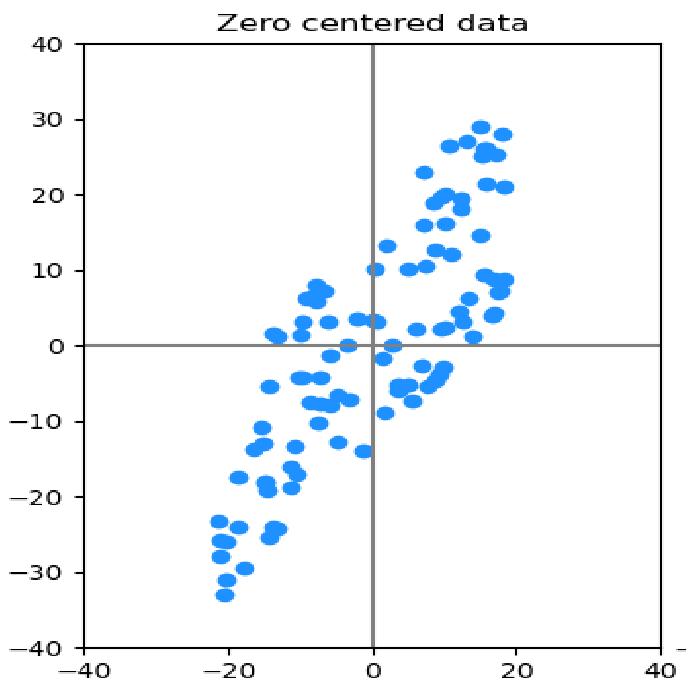
Importance of feature scaling

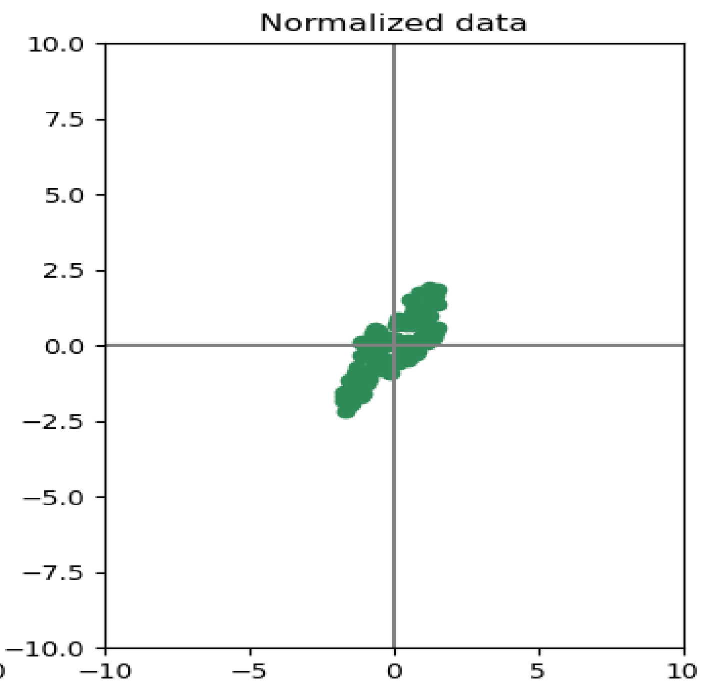




Mean subtraction, scaling







X

X -= np.mean(X, axis=0)

X /= np.std(X, axis=0)

Pre-processing for Images

- Zero center: subtract the mean across every individual feature in the data
  - Mean image
  - Mean across each channel
- Optional: normalize the data such that dimensions are approximately the same scale

https://github.com/keras-team/keras-applications/blob/master/keras_applications/imagenet_utils.py