GEORG-AUGUST-UNIVERSITÄT
GÖTTINGEN

Deep Learning

Lecture 6: Convolutional neural networks (CNN)

Alexander Ecker
Institut für Informatik, Uni Göttingen

alxecker

https://alexanderecker.wordpress.com

– Credit: some of the slides based on Fei Fei Li, Justin Johnson and Serena Yeung's slides –

Agenda for this week

Convolutional neural networks for image processing

Batch normalization

Modern CNN architectures

Transfer learning

Timeseries (1D) & volumetric/video (3D)

Background + history

FROM PRIMARY VISUAL CORTEX TO CONVOLUTIONAL NETWORKS

Hubel & Wiesel: Simple cells

Selective to orientation and location

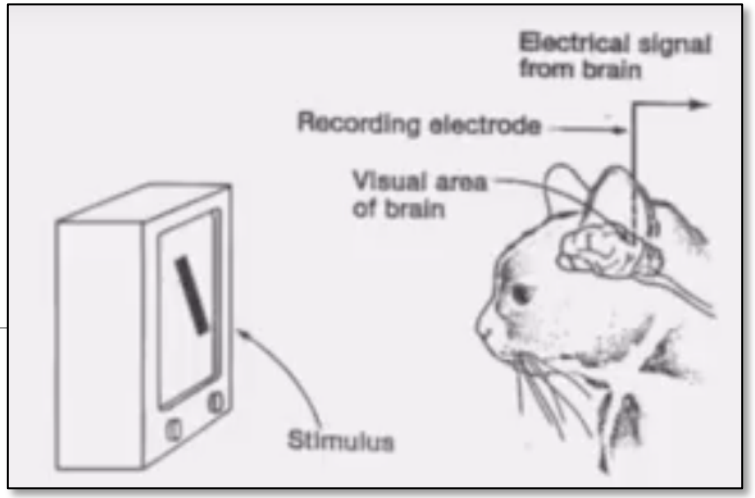
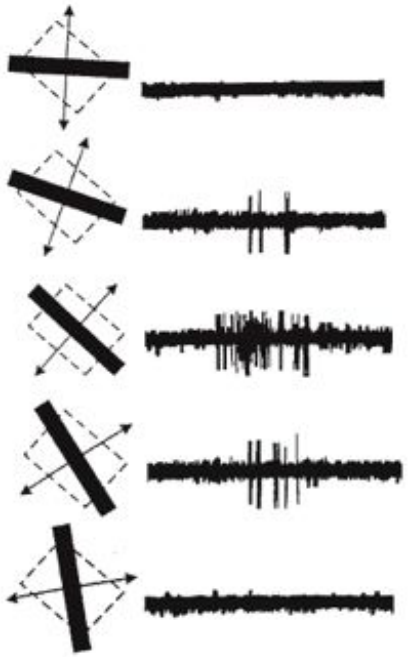
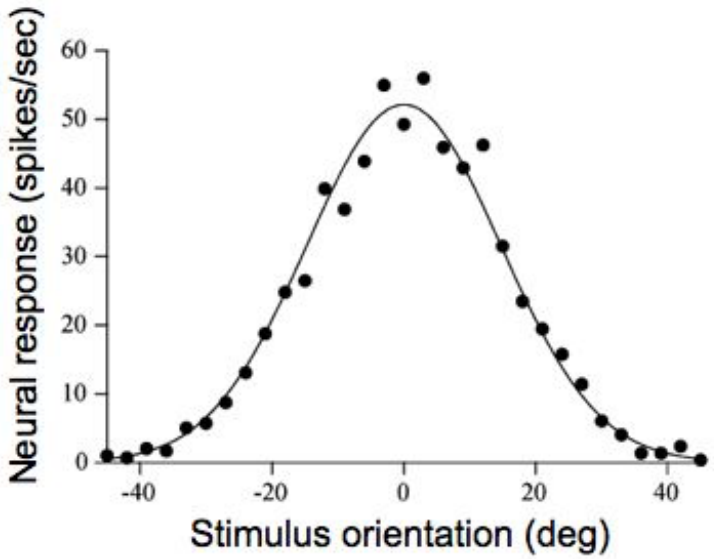
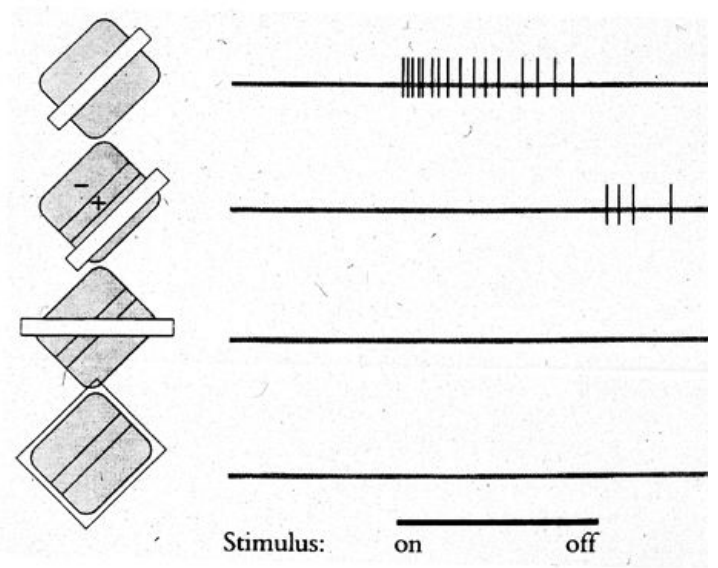


Figure credit: Andrej Karpathy

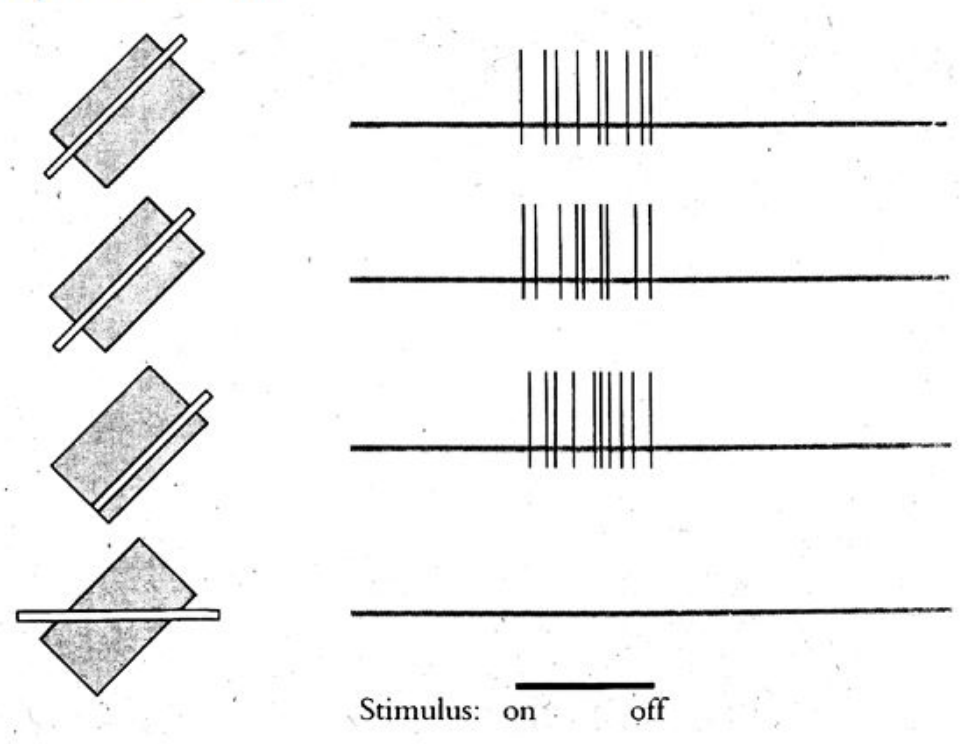






Hubel & Wiesel 1968

Hubel & Wiesel: Complex cells



Some tolerance to changes in location

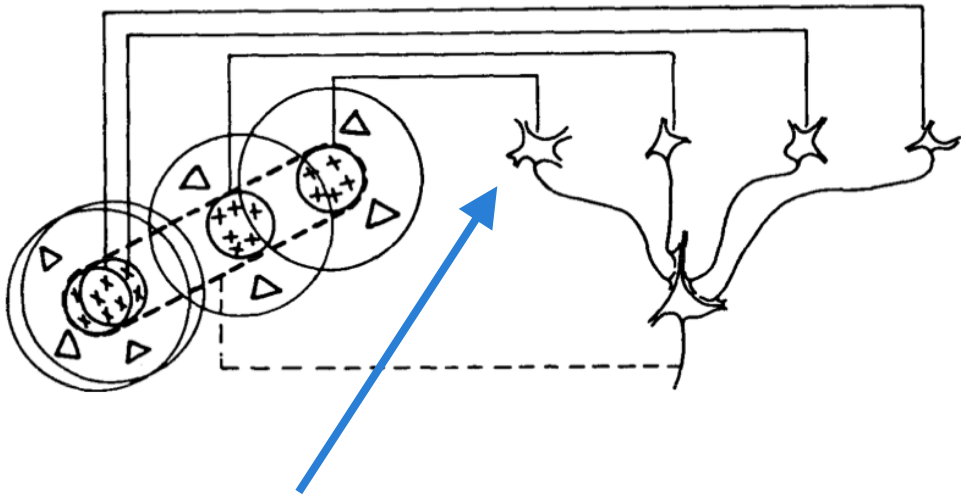
Selective to orientation

https://www.cns.nyu.edu/~david/courses/perception/lecturenotes/V1/lgn-V1.html

Hubel & Wiesel: Summary

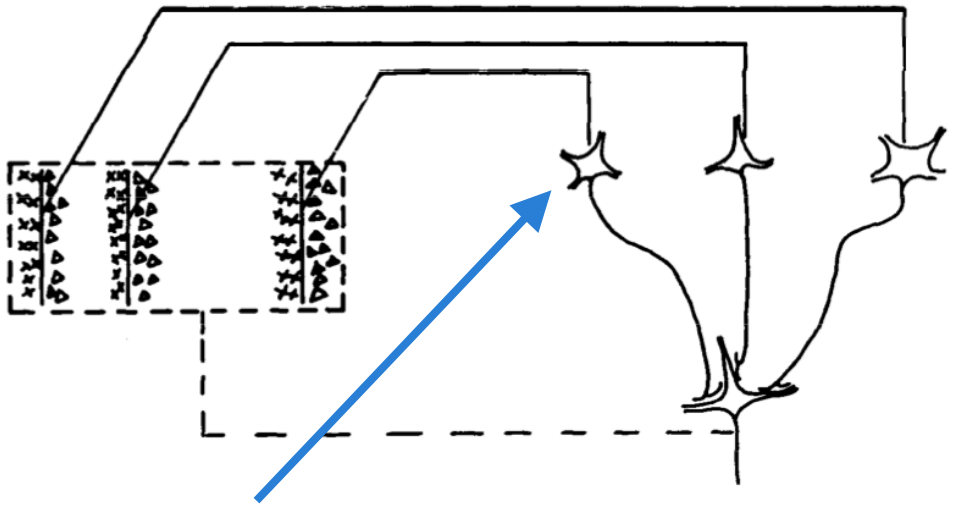
Simple cells

Edge detection / linear filter + rectification



Complex cells

Local shift invariance / pooling

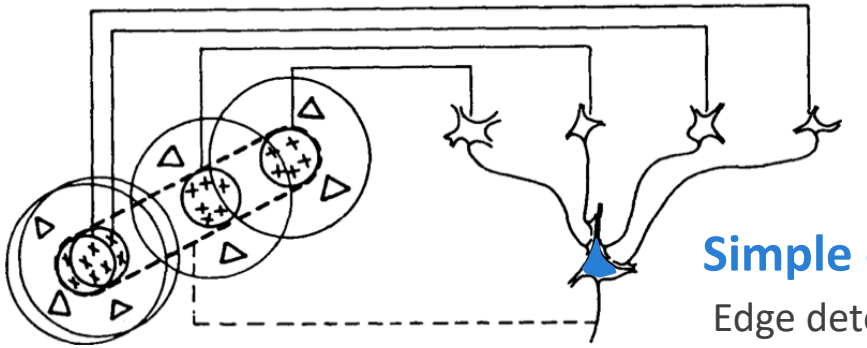


Center-surround → **Simple cells** → Simple cells → **Complex cells**

Retina/LGN → **Primary visual cortex (V1)**

Hubel & Wiesel 1959, 1962, 1968, …

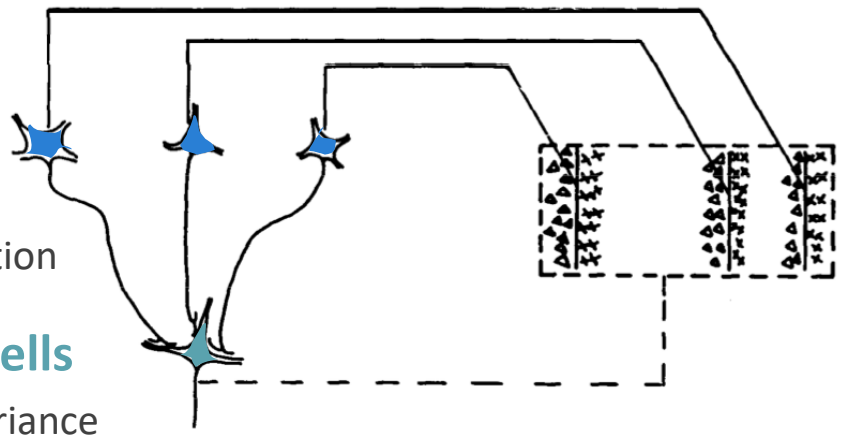
Hubel & Wiesel: Summary



LGN

Simple cells

Edge detection
linear filter + rectification



Complex cells

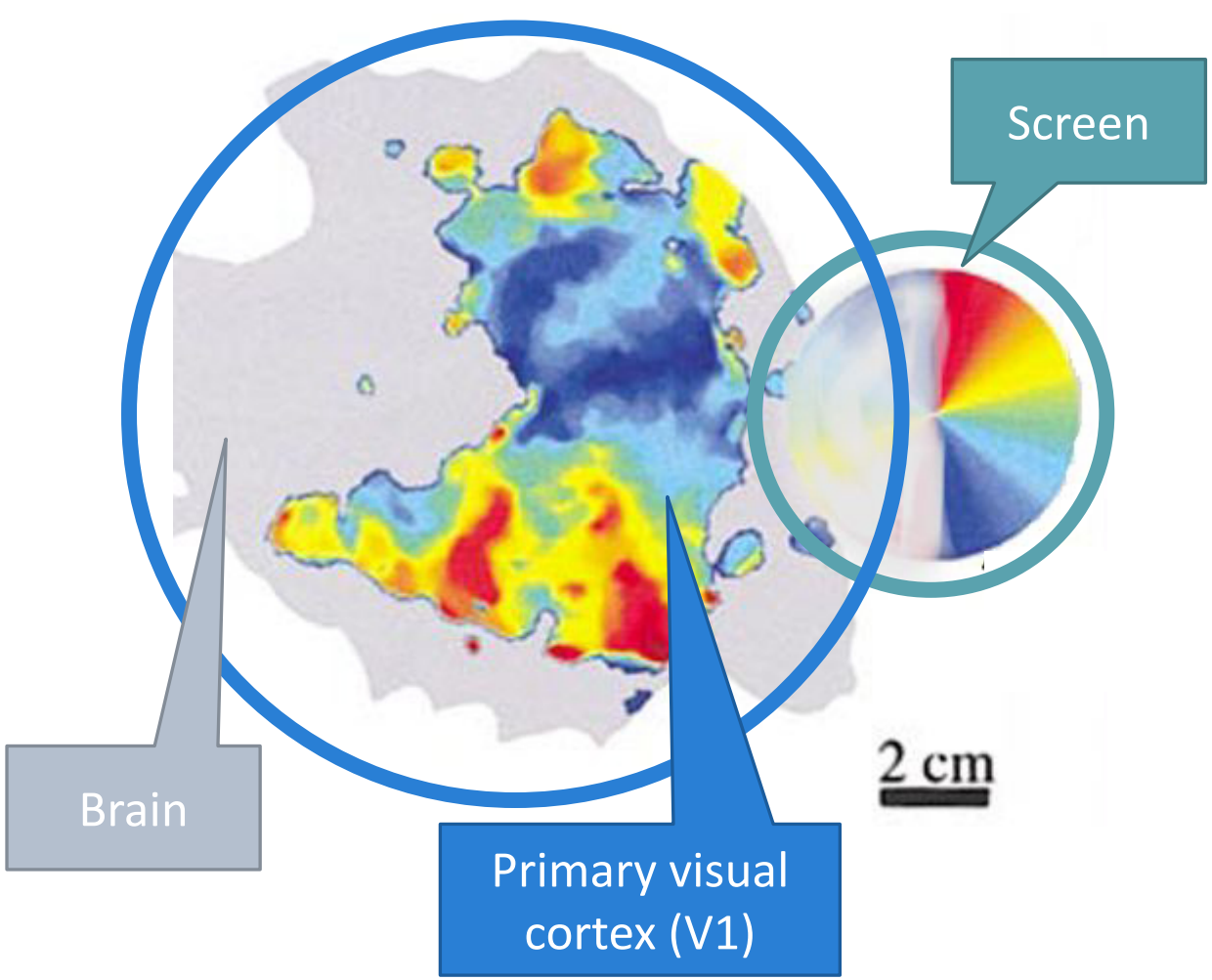
Local shift invariance
pooling

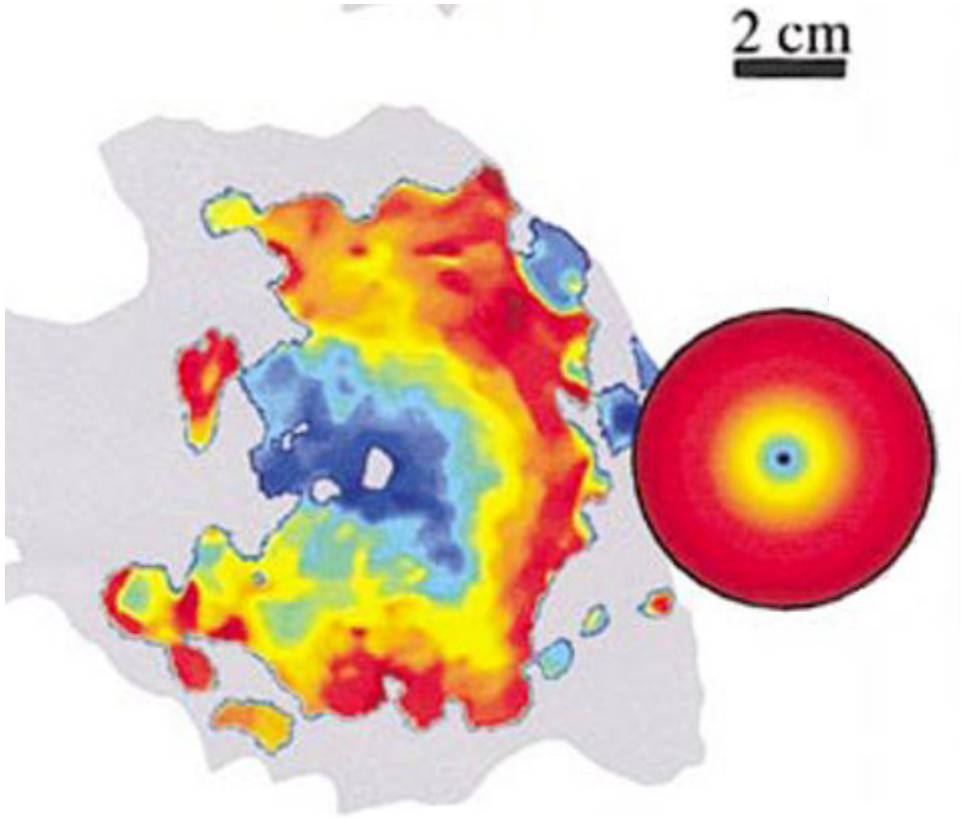
Retina + LGN

Primary visual cortex (V1)

Hubel & Wiesel 1959, 1962, 1968, …

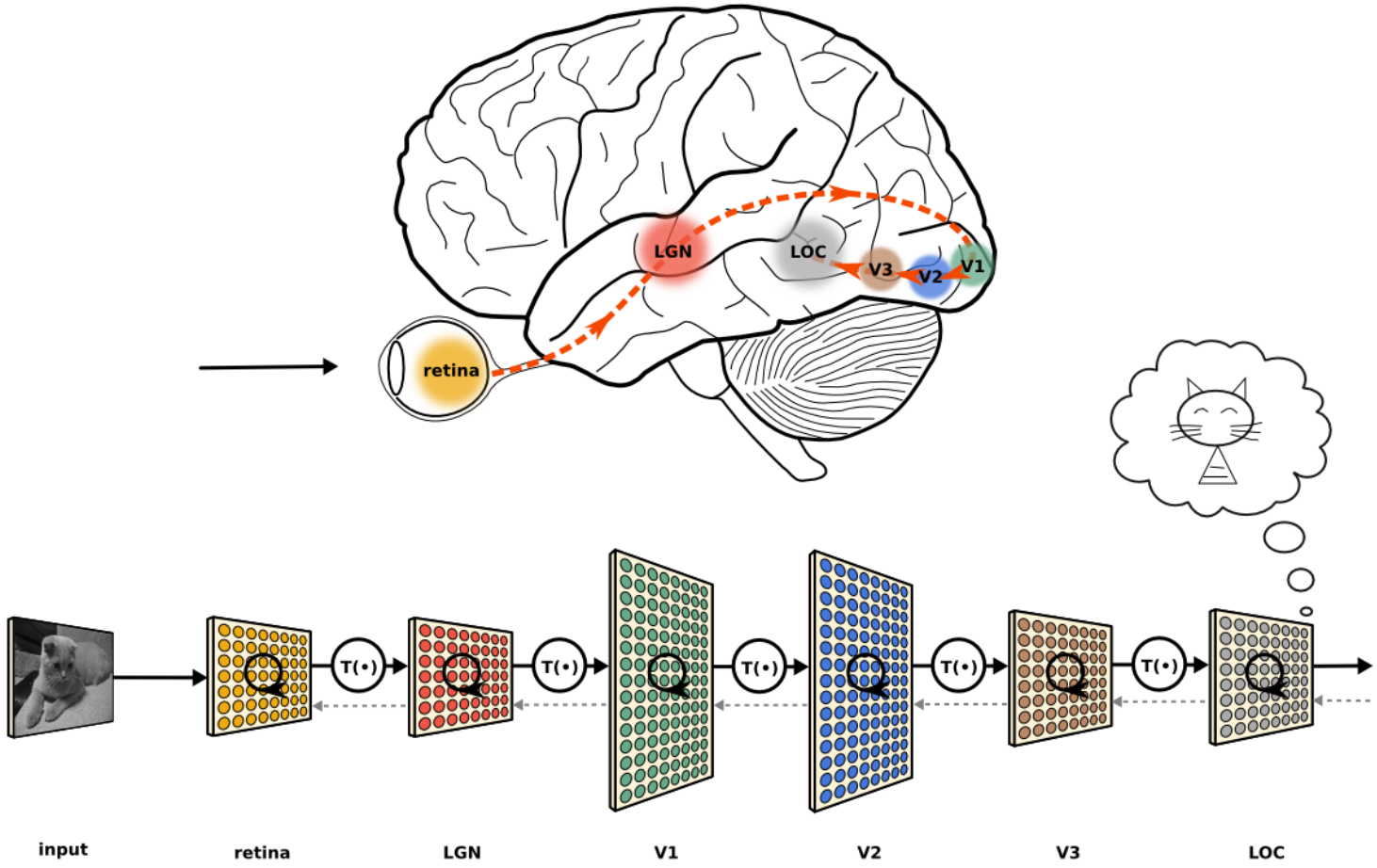
Retinotopic organization of visual cortex





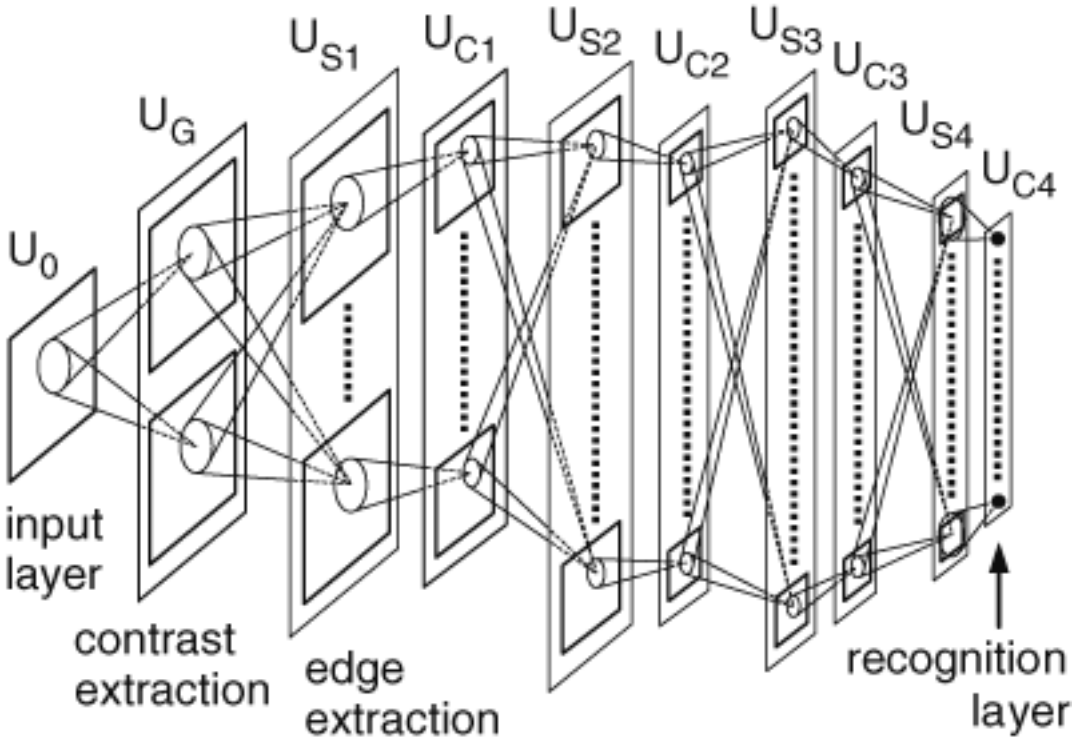
Source: Wikipedia

Hierarchical data processing



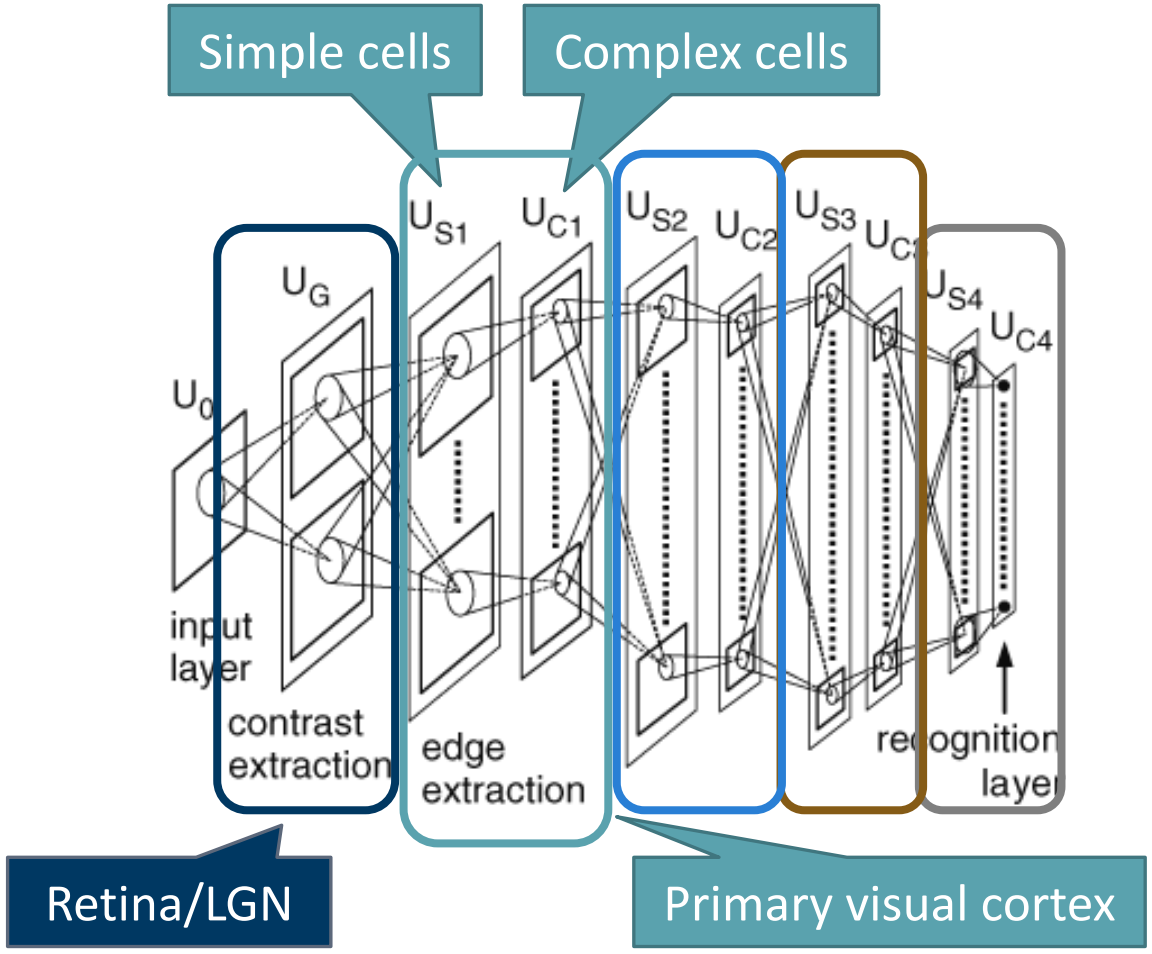
https://neuwritesd.org/2015/10/22/deep-neural-networks-help-us-read-your-mind/

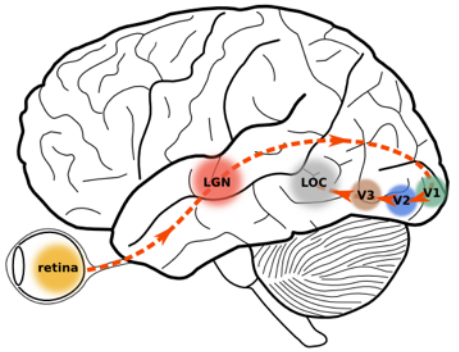
The Neocognitron (Fukushima 1980)



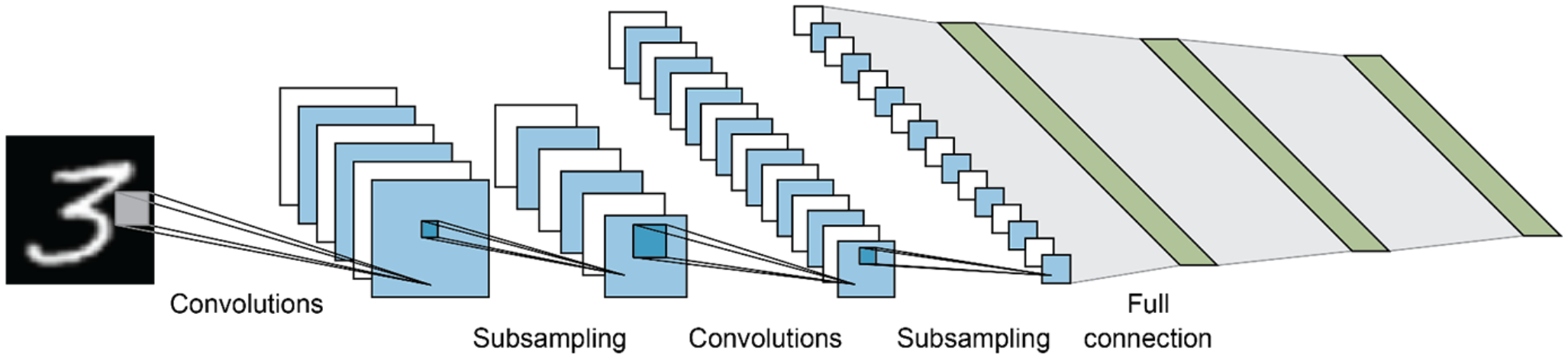
K. Fukushima (1980): "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position", *Biological Cybernetics*.

The Neocognitron (Fukushima 1980)





Convolutional neural networks (CNN)

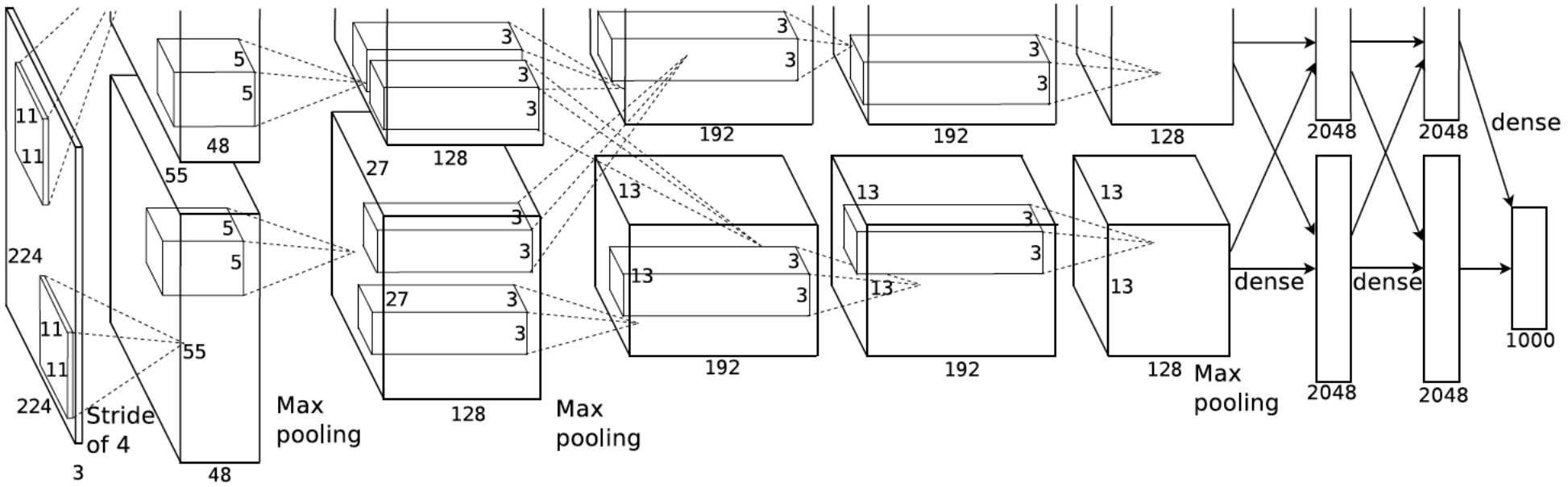


Denker et al. 1989, LeCun et al. 1989, 1998

Image source: https://livebook.manning.com/book/grokking-deep-learning-for-computer-vision/chapter-5/v-3/13

ImageNet Classification with Deep Convolutional Neural Networks

Krizhevsky, Sutskever, Hinton 2012



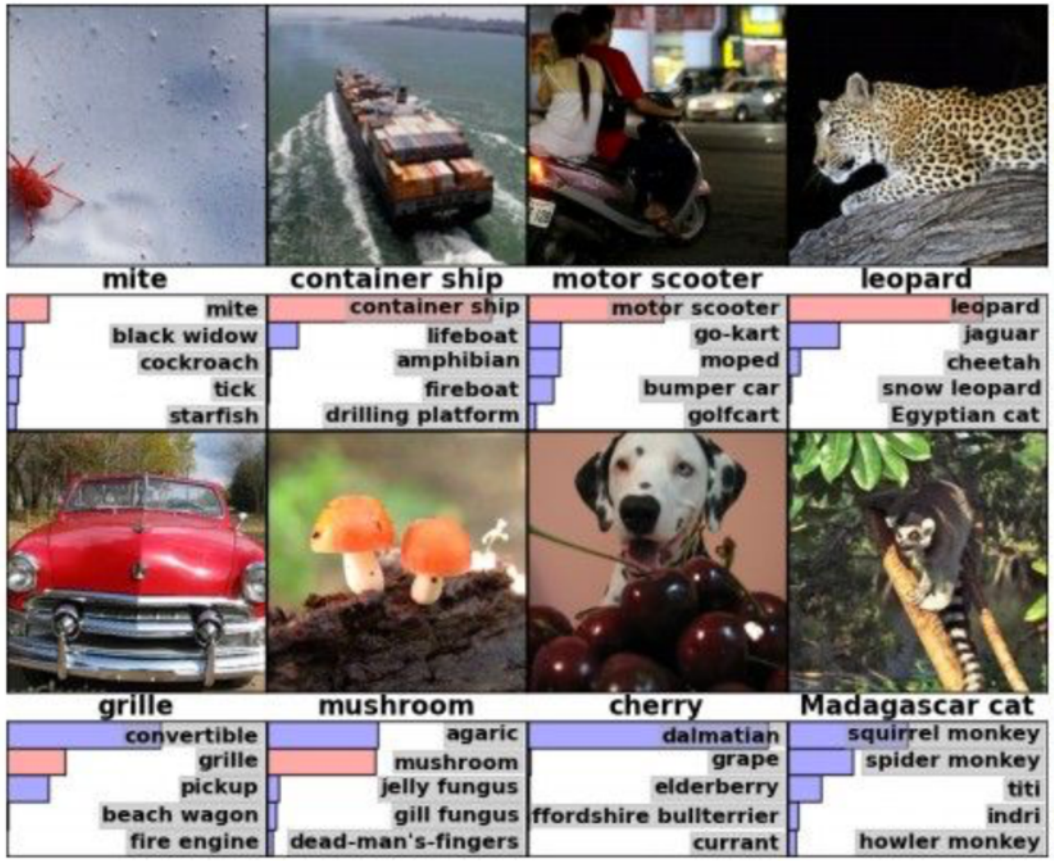
Today usually referred to as "**AlexNet**"

Fast-forward to today: ConvNets are everywhere

Always working with images, convNets are used

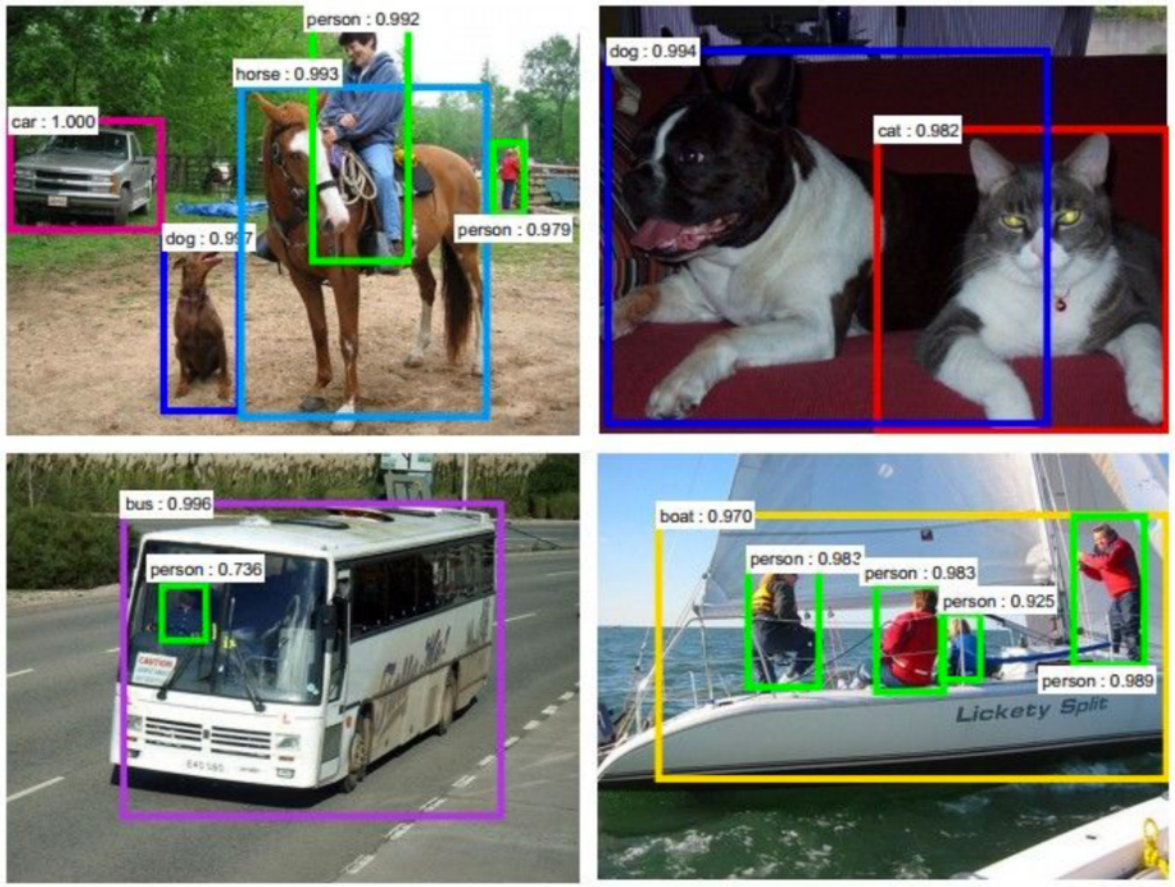
Classification

Retrieval



Fast-forward to today: ConvNets are everywhere

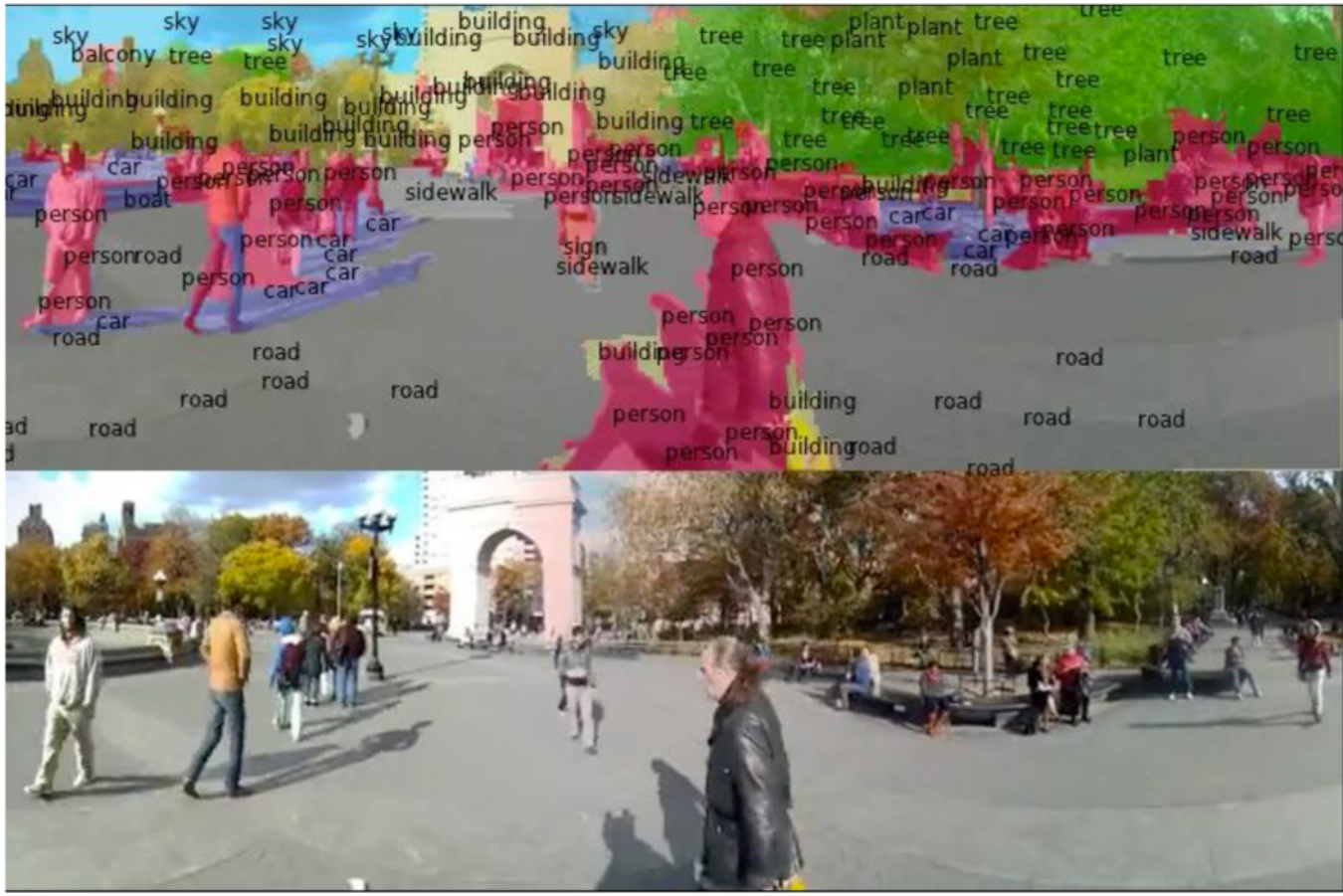
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere

Photo by Lane McIntosh. Copyright CS231n 2017.

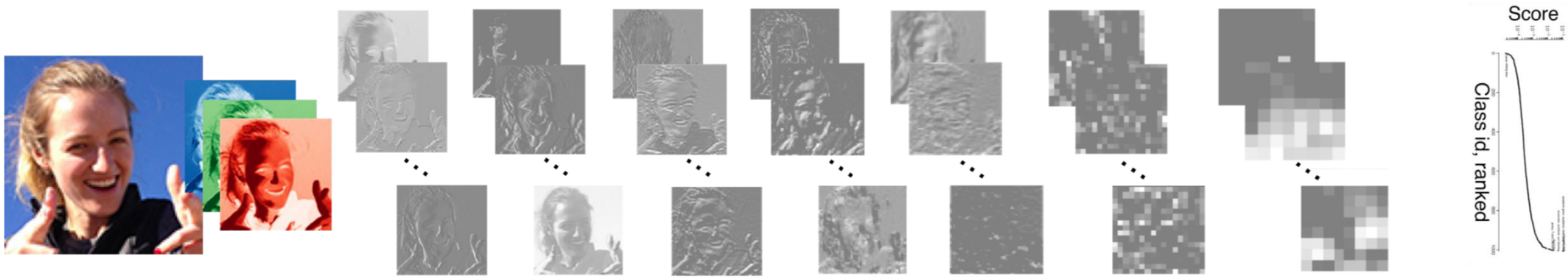
self-driving cars

NVIDIA Tesla line

(these are the GPUs on rye01.stanford.edu)

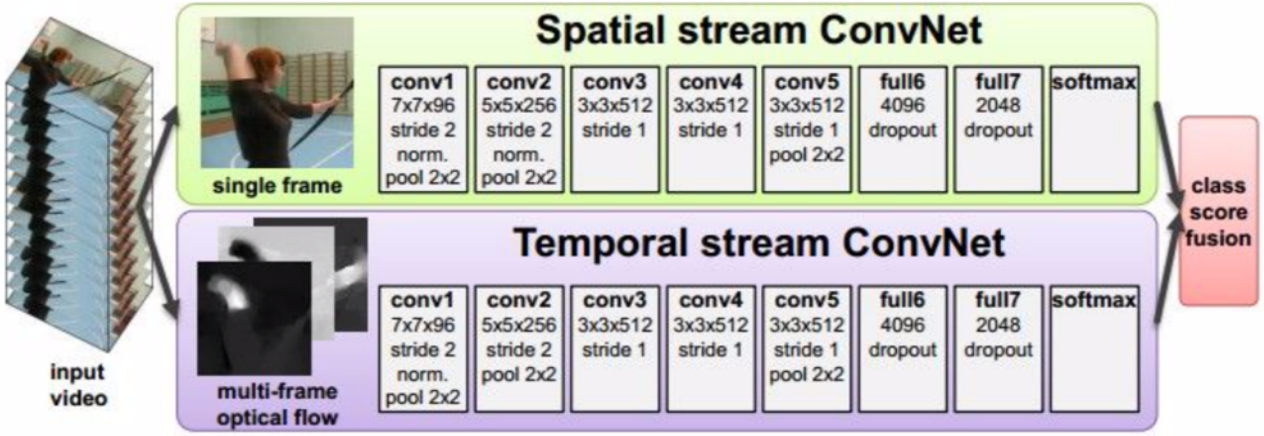
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

Fast-forward to today: ConvNets are everywhere



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

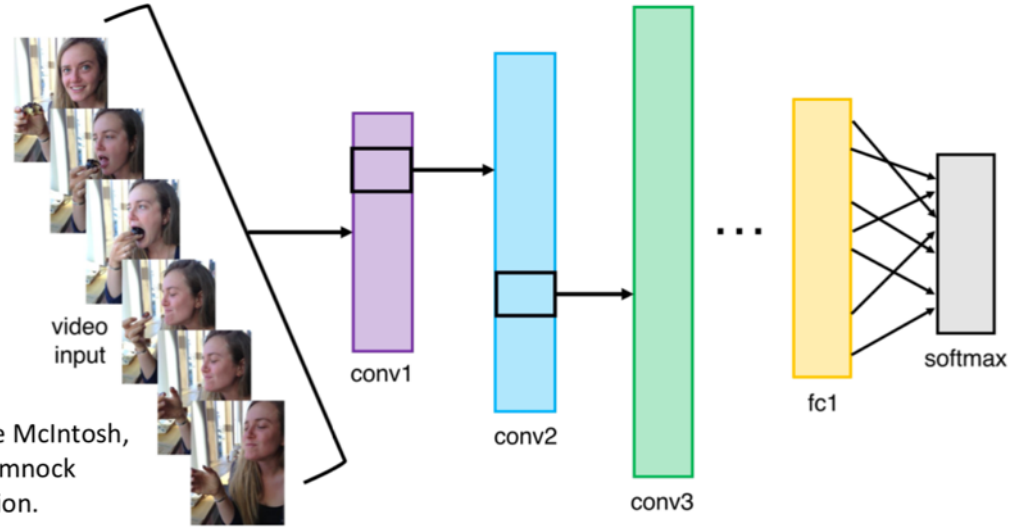
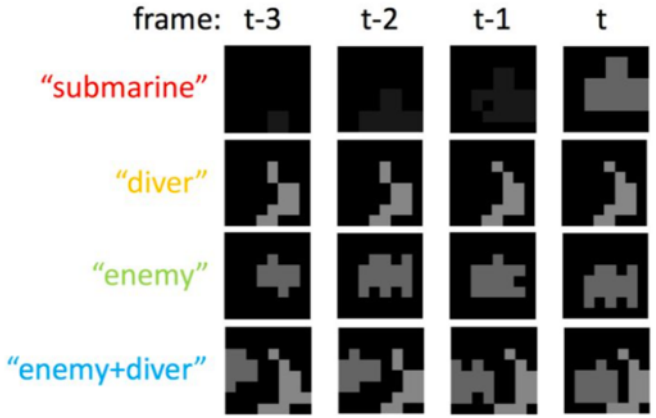


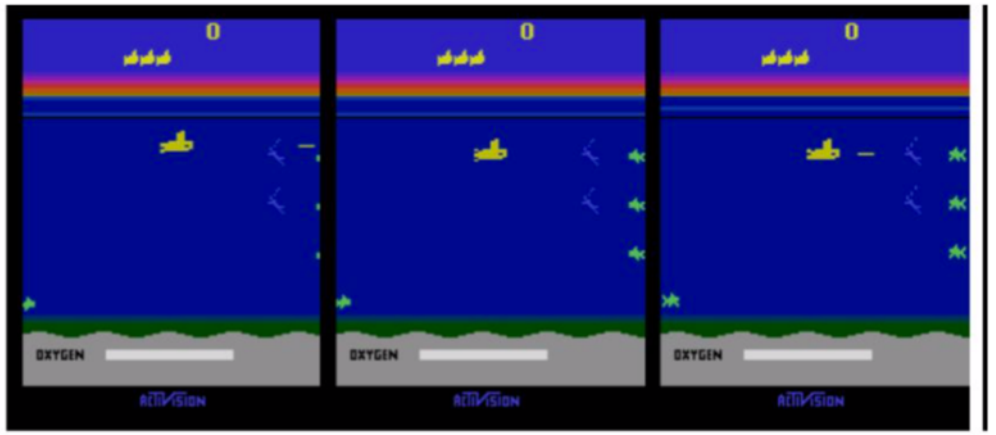
Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

Fast-forward to today: ConvNets are everywhere

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]

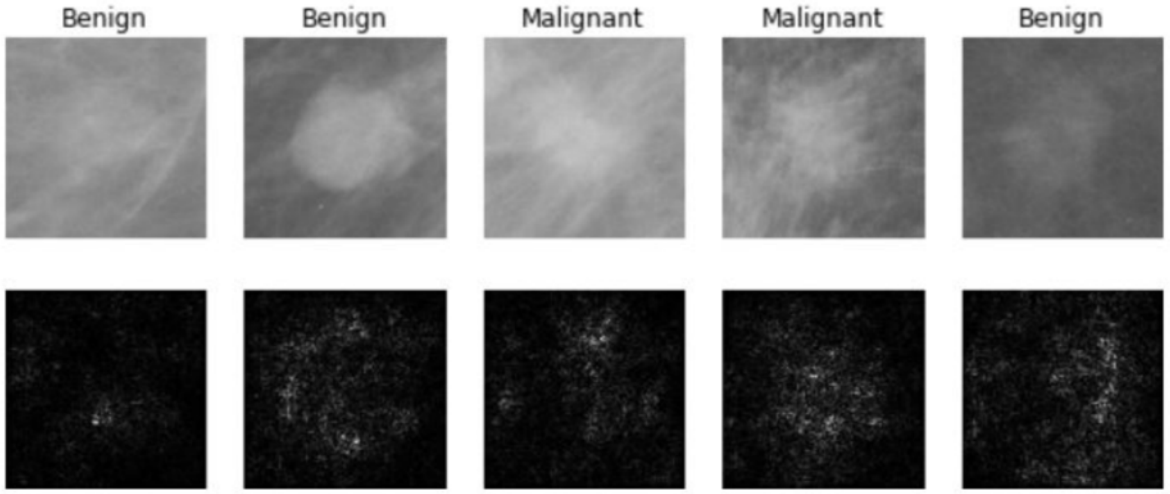




[Guo et al. 2014]

Figures copyright Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, and Xiaoshi Wang, 2014. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere



[Levy et al. 2016]

[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

[Sermanet et al. 2011]
[Ciresan et al.]

Photos by Lane McIntosh.
Copyright CS231n 2017.

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Whale recognition, Kaggle Challenge

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.

Mnih and Hinton, 2010

No errors

A white teddy bear sitting in the grass

Minor errors

A man in a baseball uniform throwing a ball

Somewhat related

A woman is holding a cat in her hand

A man riding a wave on top of a surfboard

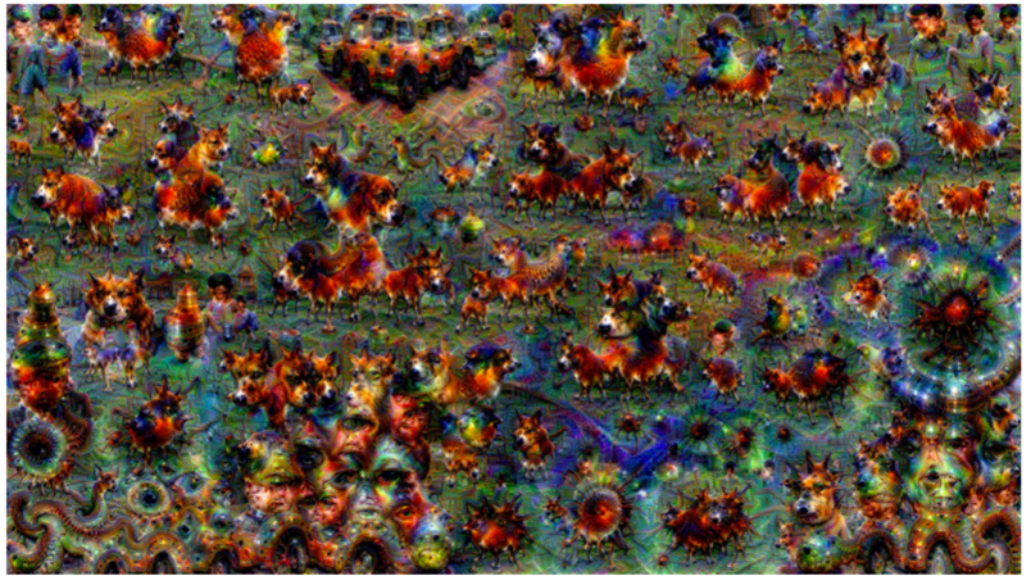
A cat sitting on a suitcase on the floor

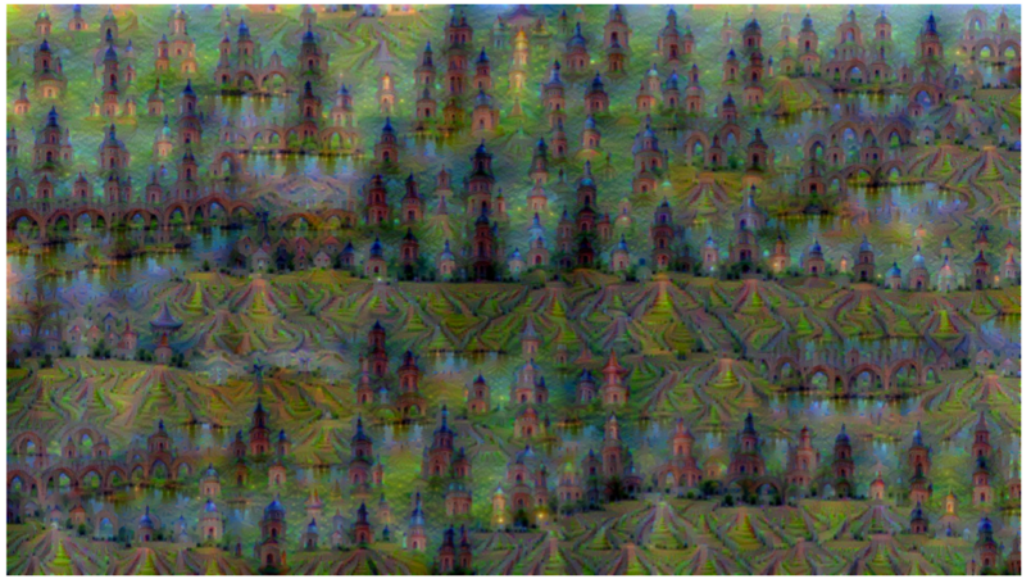
A woman standing on a beach holding a surfboard

Image Captioning

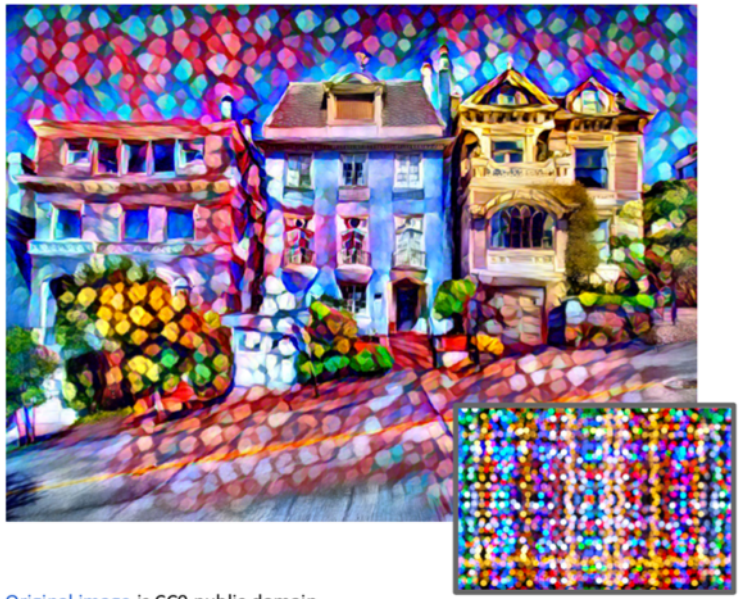
[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

Captions generated by Justin Johnson using Neuraltalk2





Figures copyright Justin Johnson, 2015. Reproduced with permission. Generated using the Inceptionism approach from a blog post by Google Research.



Original image is CC0 public domain
Starry Night and Tree Roots by Van Gogh are in the public domain
Bokeh image is in the public domain
Stylized images copyright Justin Johnson, 2017;
reproduced with permission

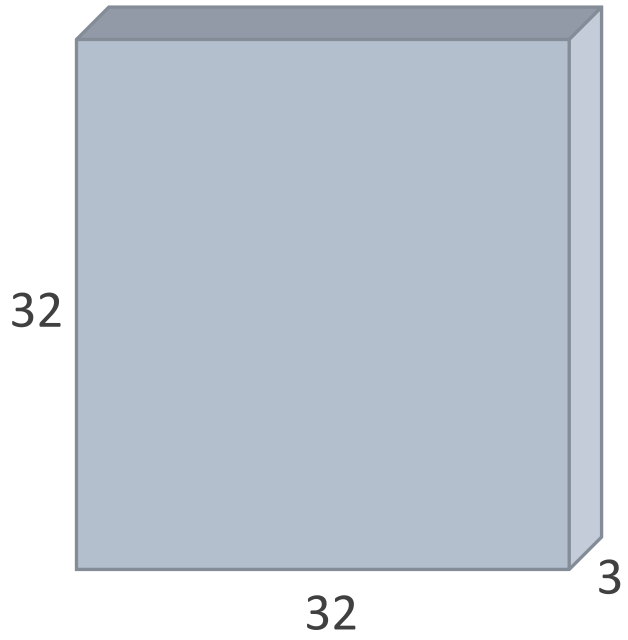
Gatys et al, "Image Style Transfer using Convolutional Neural Networks", CVPR 2016
Gatys et al, "Controlling Perceptual Factors in Neural Style Transfer", CVPR 2017

Convolution

Consider a 32×32 RGB image

Image

$32 \times 32 \times 3$

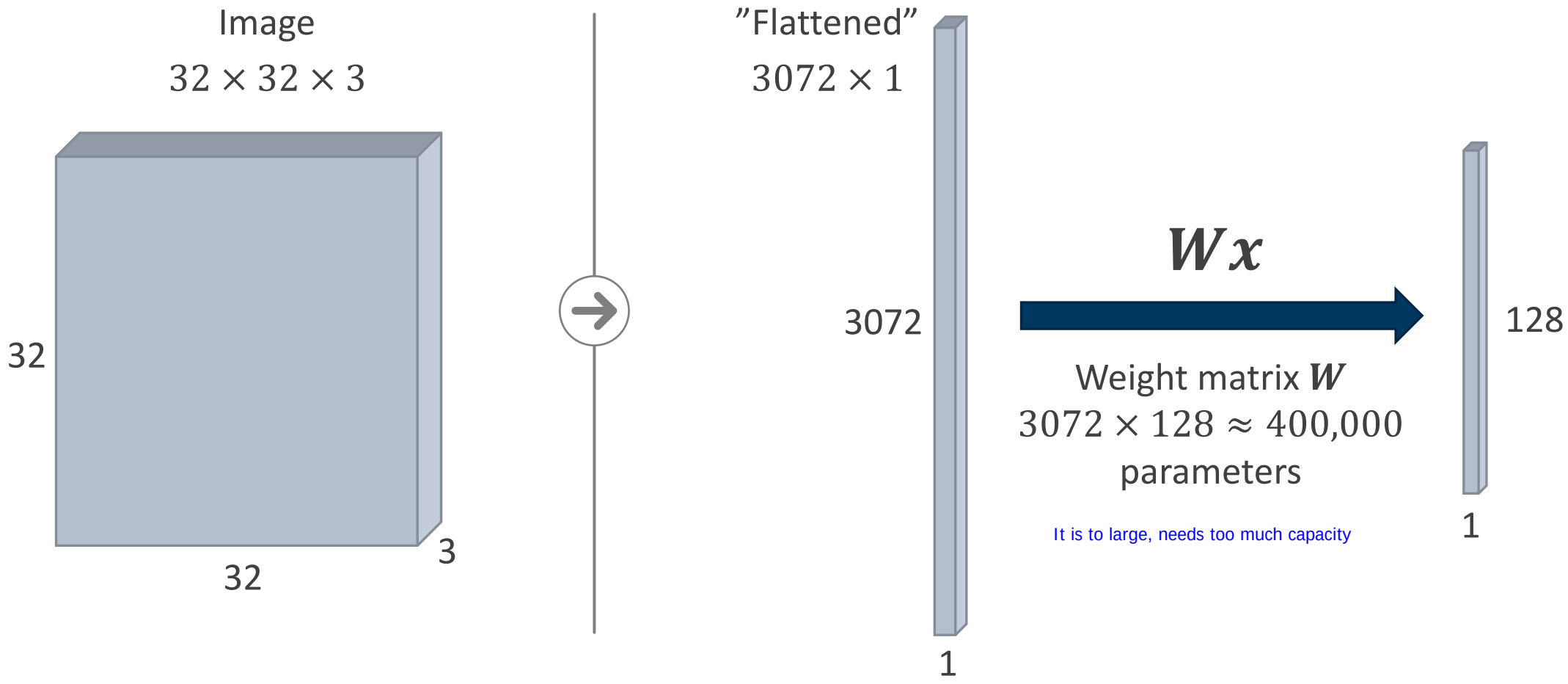


Red

Green

Blue

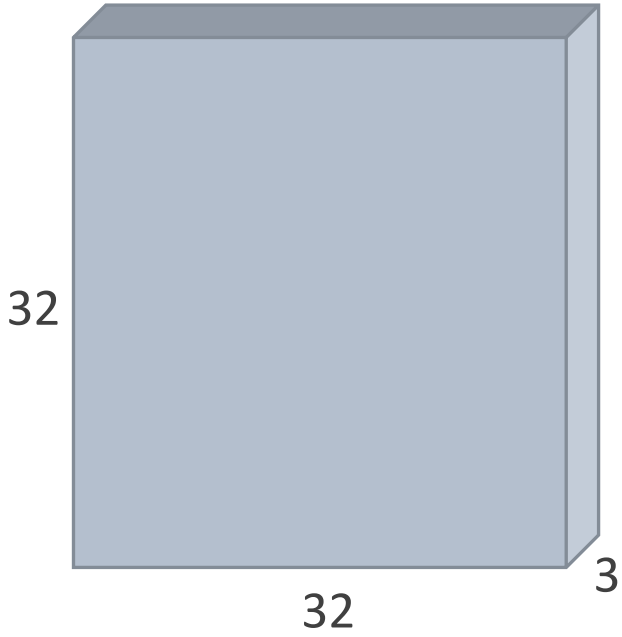
Fully-connected layer



Convolution

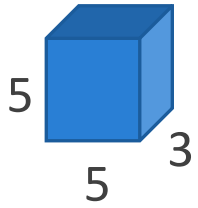
Image

$32 \times 32 \times 3$



Filter

$5 \times 5 \times 3$

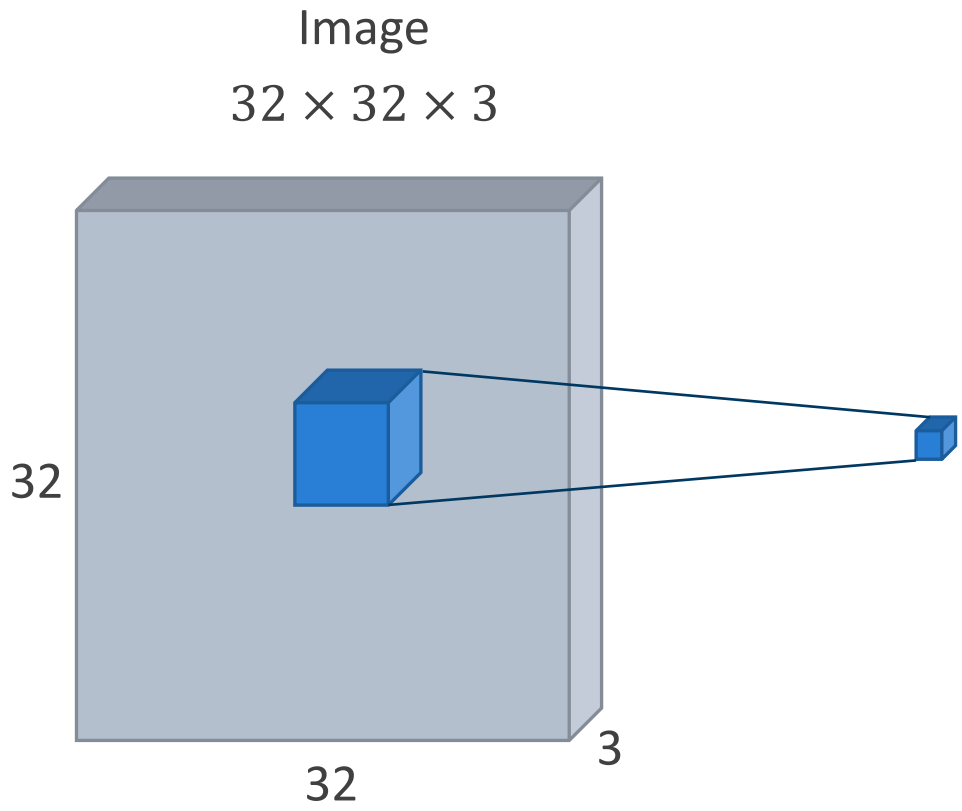


Convolve the filter with the image

=

Slide over the image, computing dot products

Convolution

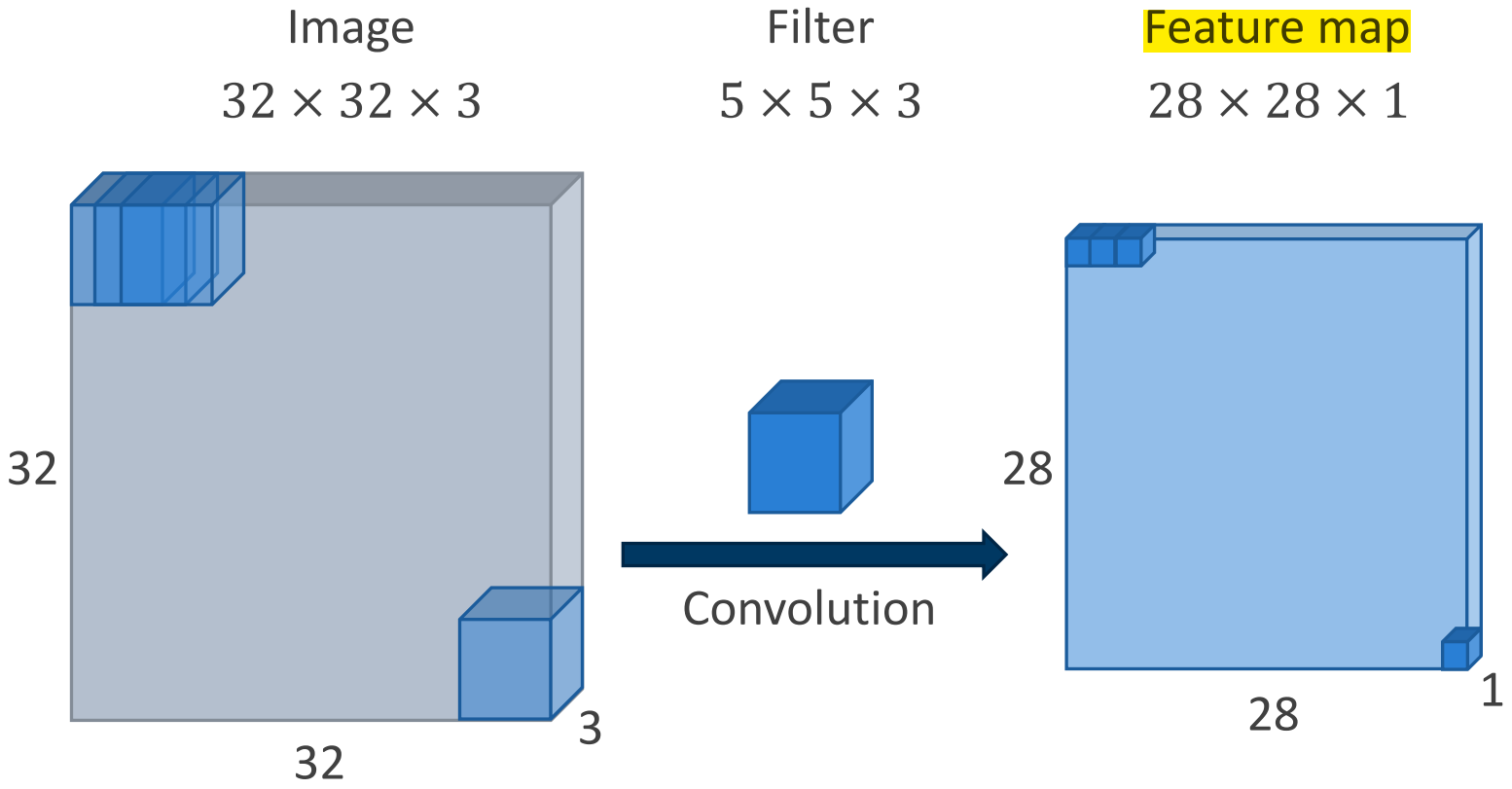


1 number: $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + b$

The result is the inner product of

- a local $5 \times 5 \times 3$ chunk $\boldsymbol{x}$ of the image
- a $5 \times 5 \times 3$ filter $\boldsymbol{w}$

Convolution



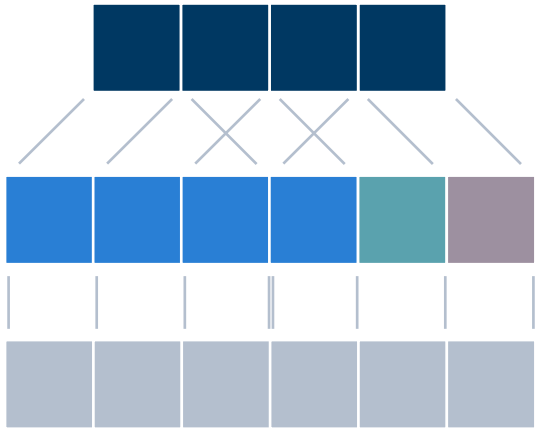
Because of the sliding the result of the filter is an image again!

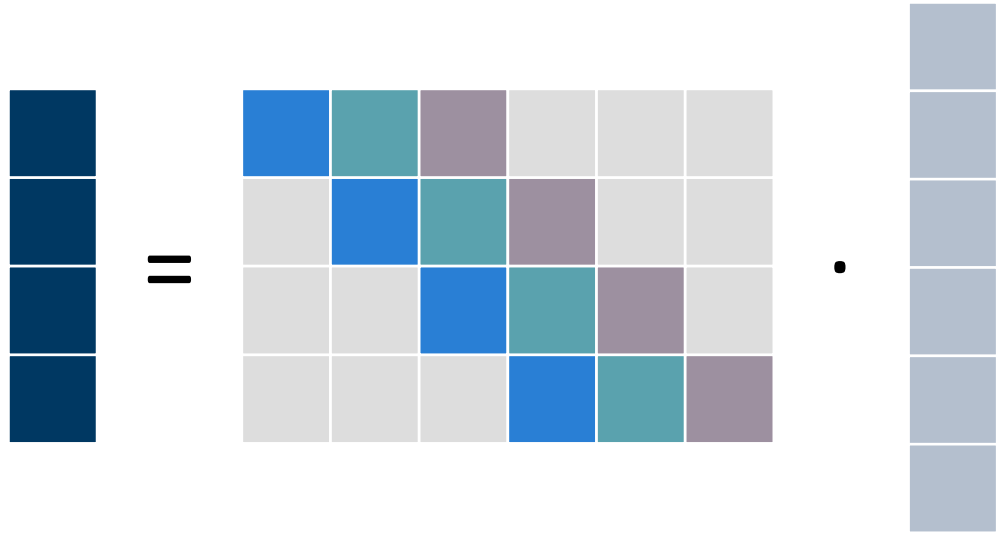
Advantage:
- can express a convolution as a matrix
- can rewrite as an inner product
- easier to calc

Each row is a copy of another row or so
- at the end just get it through the dot product

Weight sharing as the key advantage of CNNs

A convolution can be written as a matrix multiplication

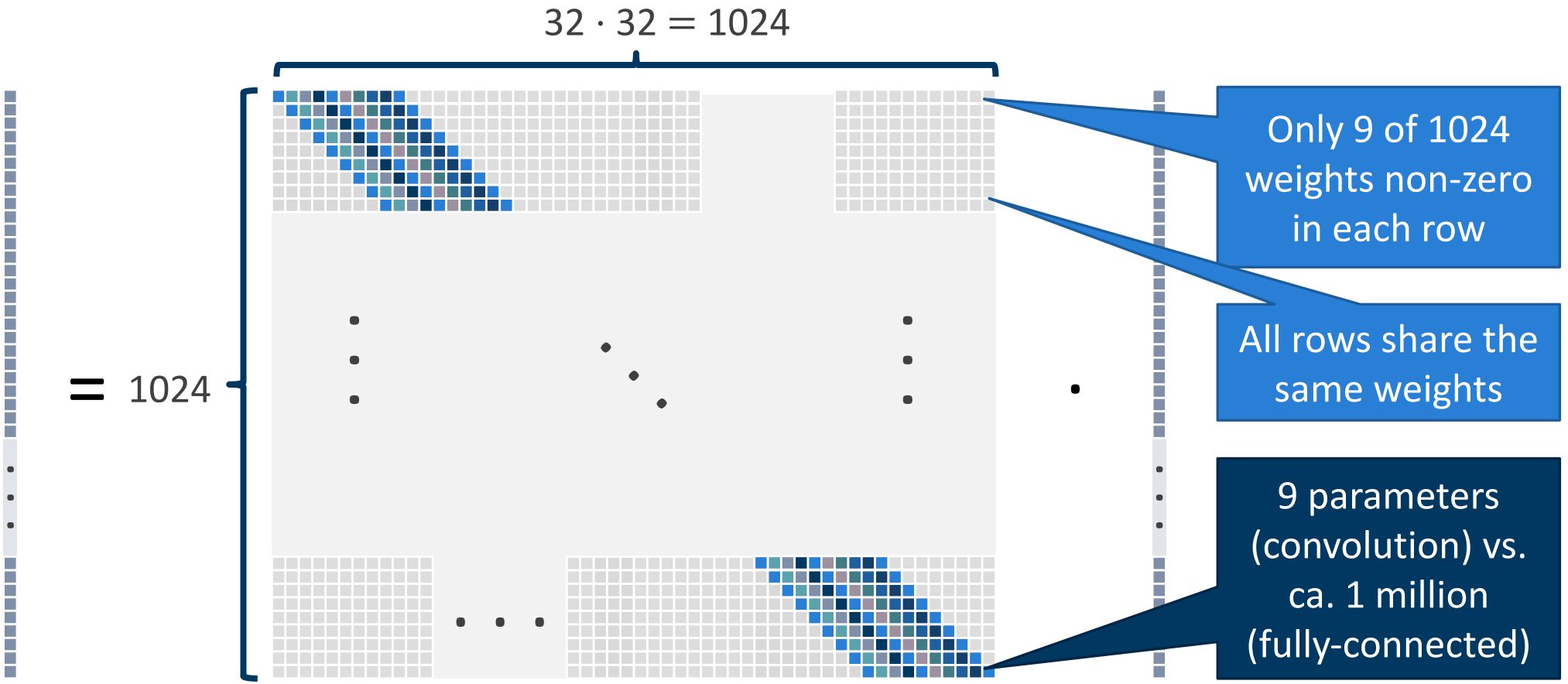




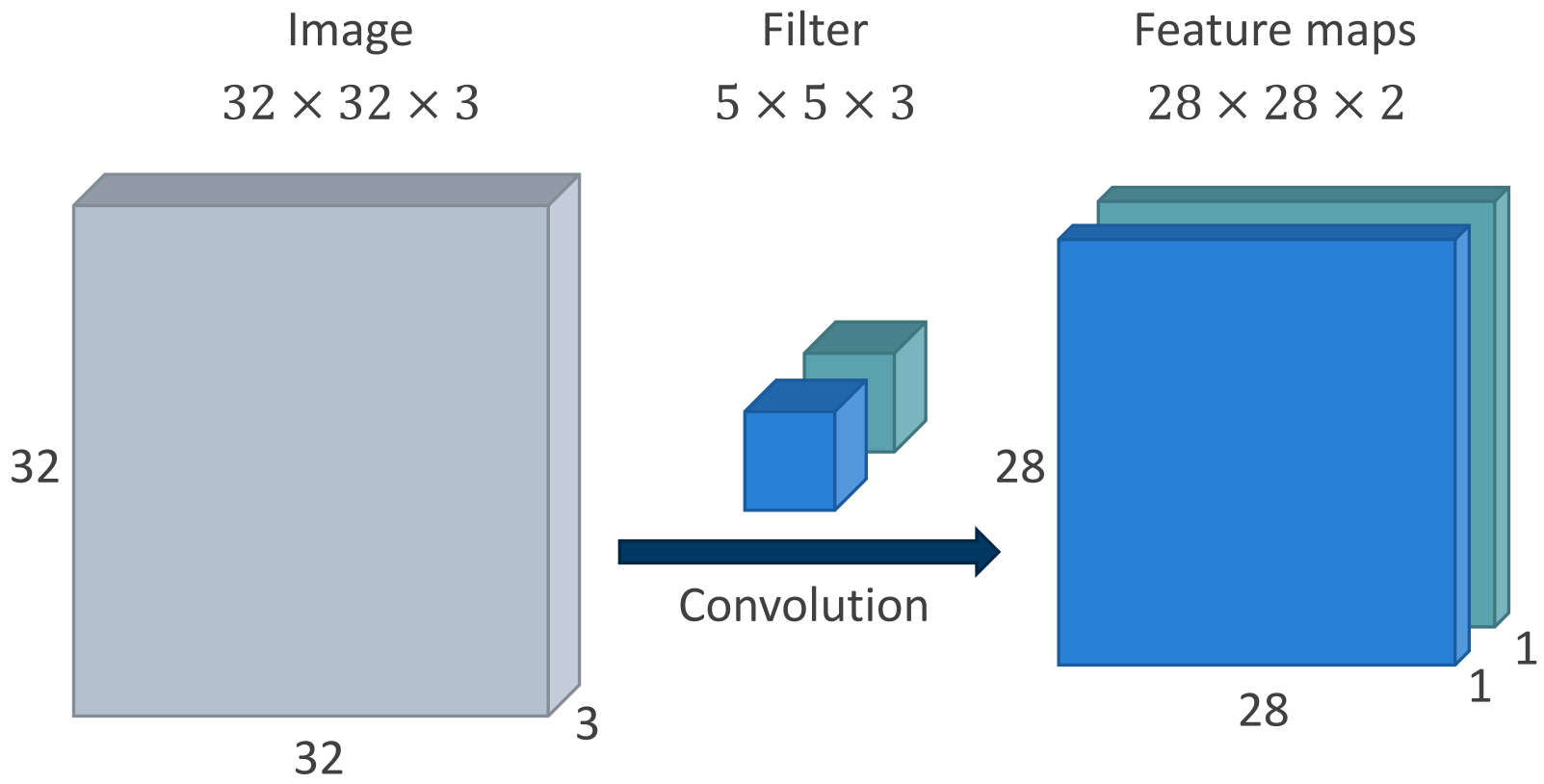
Weights are shared (same color)

Connections are localized (gray = 0)

Example: 32 × 32 feature map / 3 × 3 kernel



Convolution

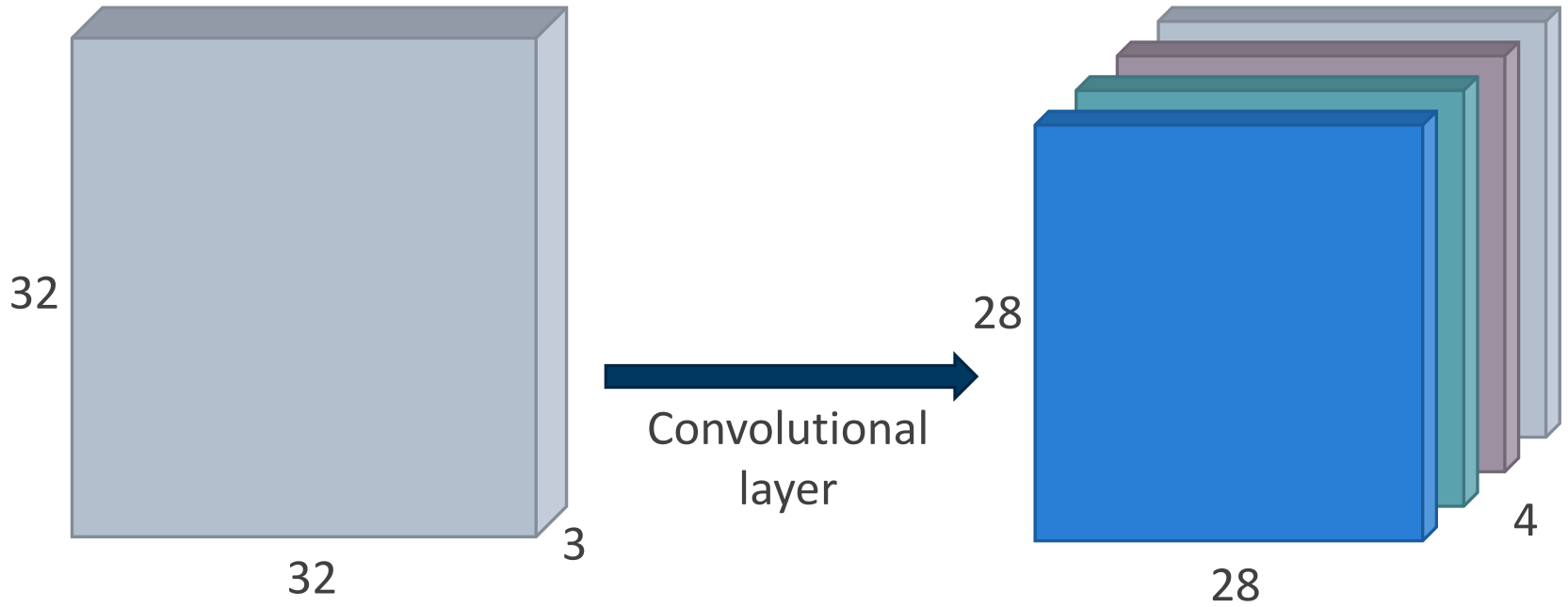


Convolution

Image
$32 \times 32 \times 3$

Filters
$5 \times 5 \times 3 \times 4$

Feature maps
$28 \times 28 \times 4$



"Stack up" feature maps to get a new $28 \times 28 \times 4$ image

one filter =>

one activation map

example 5x5 filters
(32 total)

Activations:

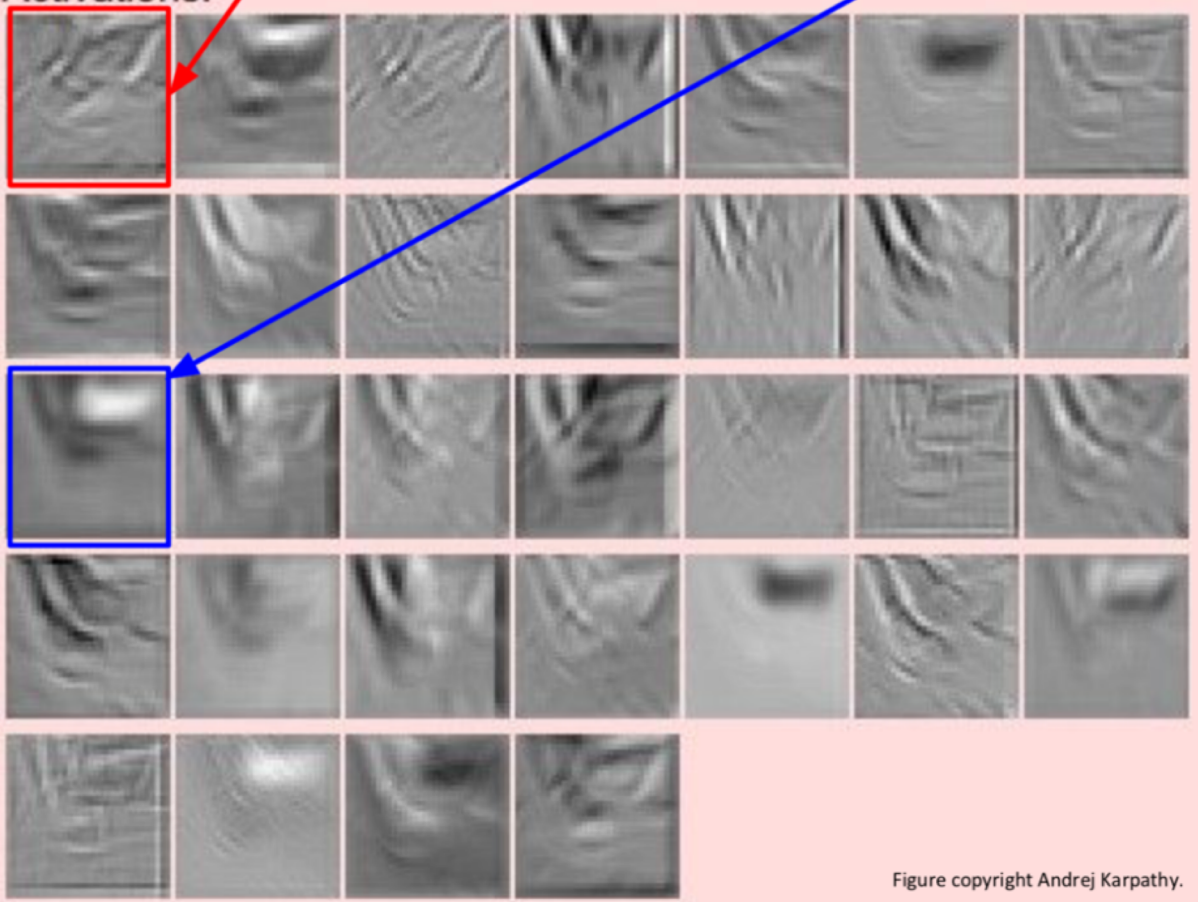


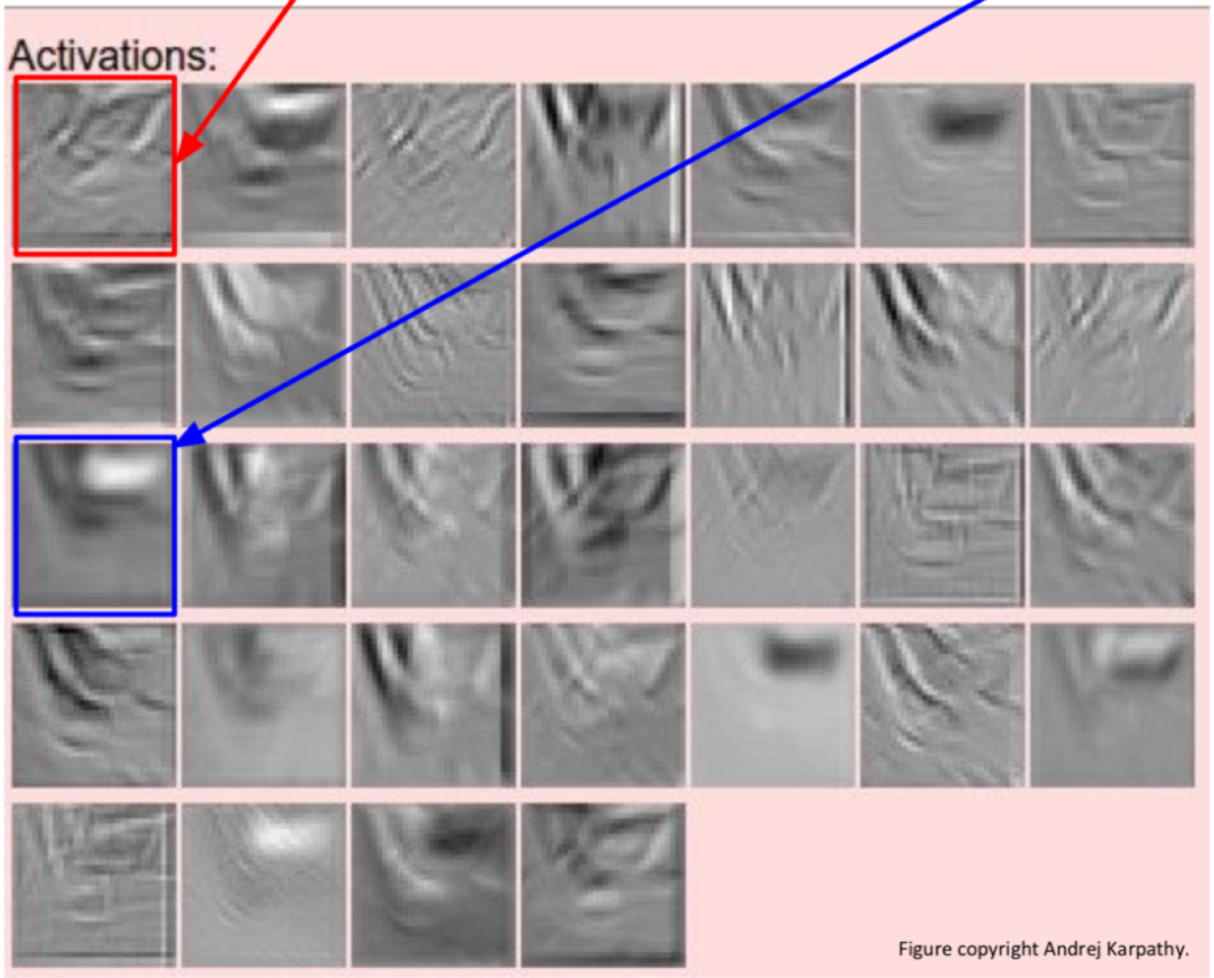
Figure copyright Andrej Karpathy.

one filter =>

one activation map

example 5x5 filters
(32 total)

even by grey-scalling, the color informattion isn't lost



We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

one filter =>
one activation map

example 5x5 filters
(32 total)

Normally known: cross-validation, in deep learning: convolution

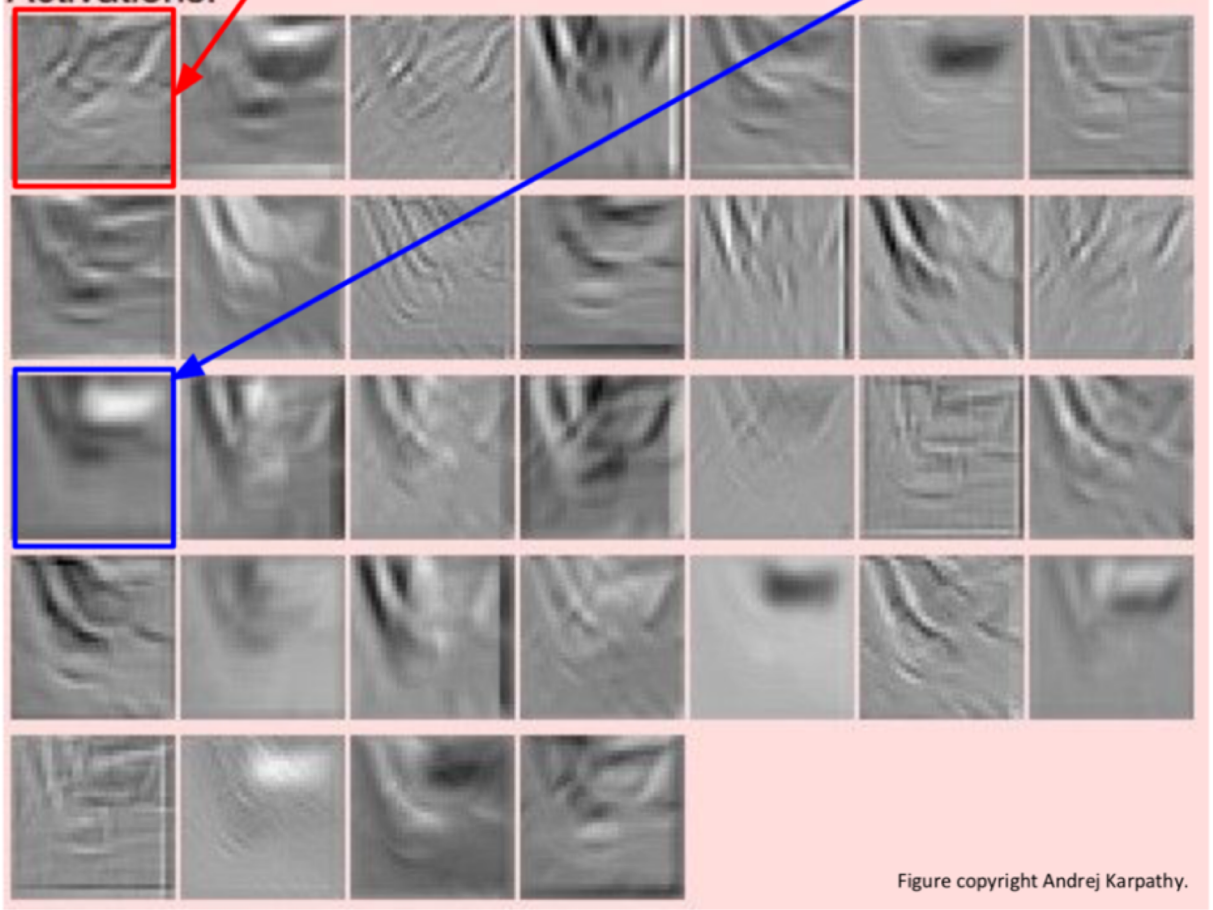


Figure copyright Andrej Karpathy.

We call the layer convolutional because it is related to convolution of two si

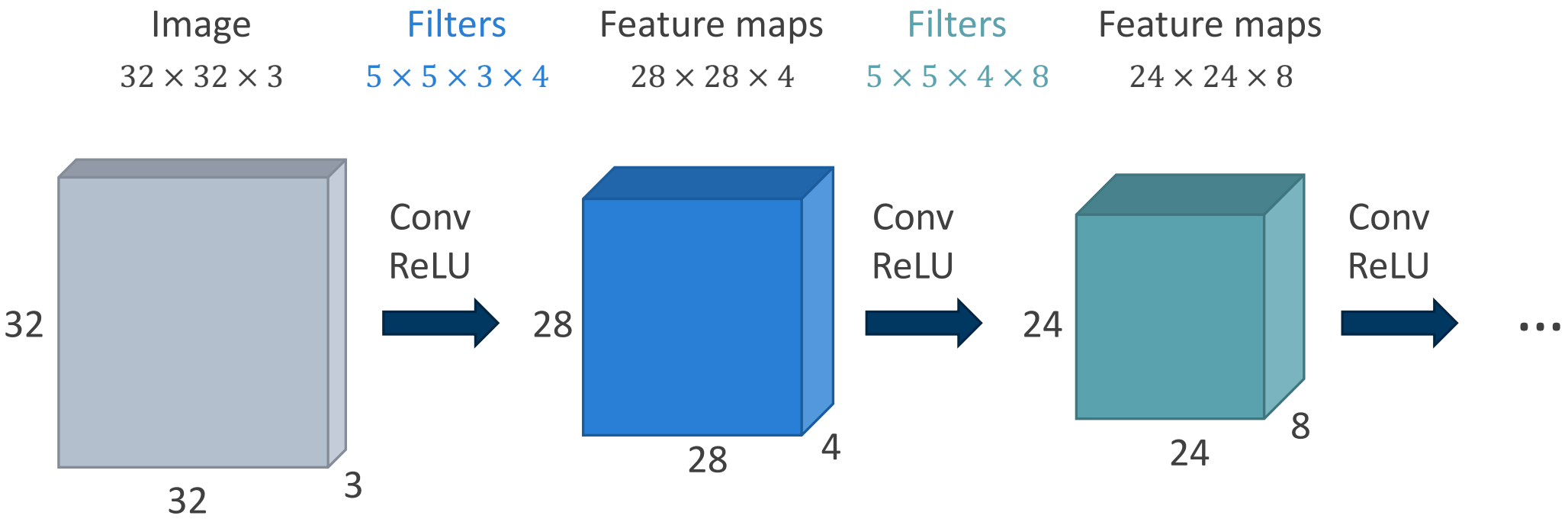
Strictly speaking, it's a correlation

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x+n_1, y+n_2]$$

elementwise multiplication and sum of a filter and the signal (image)

Convolutional neural network

means a sequence of convolutions

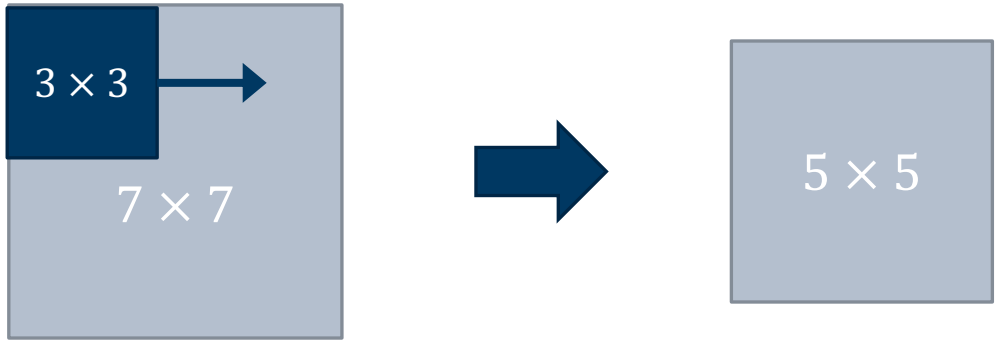


A convnet is a sequence of convolution layers and activation functions

Zero padding

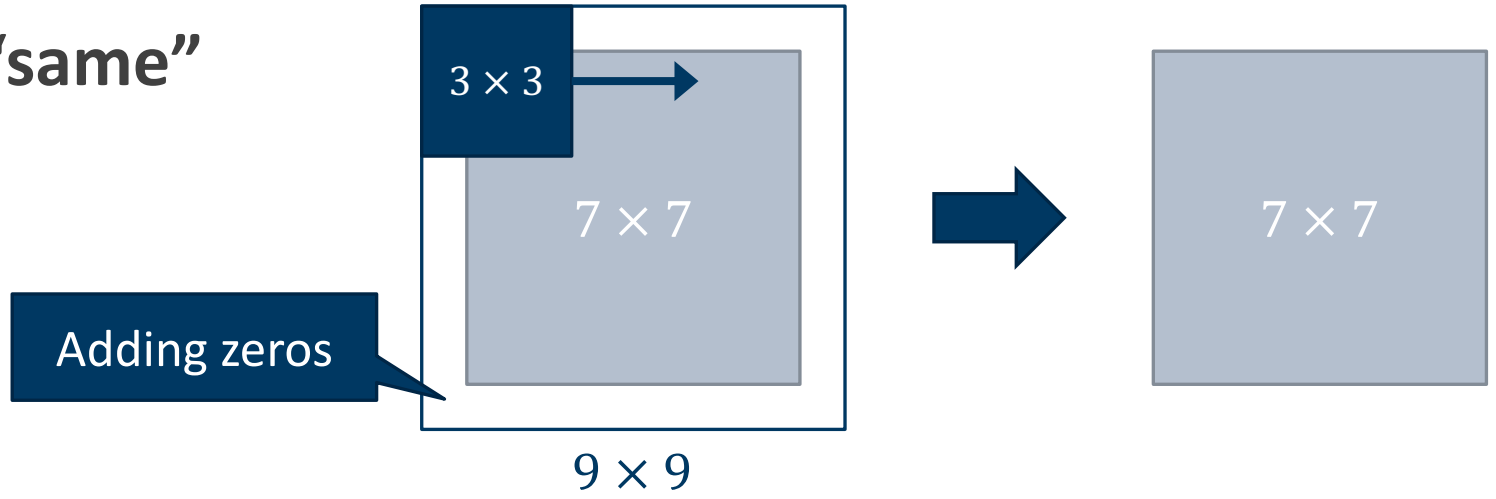
feature maps get alwasy smaller and smaller, all the time when using a filter

“valid”



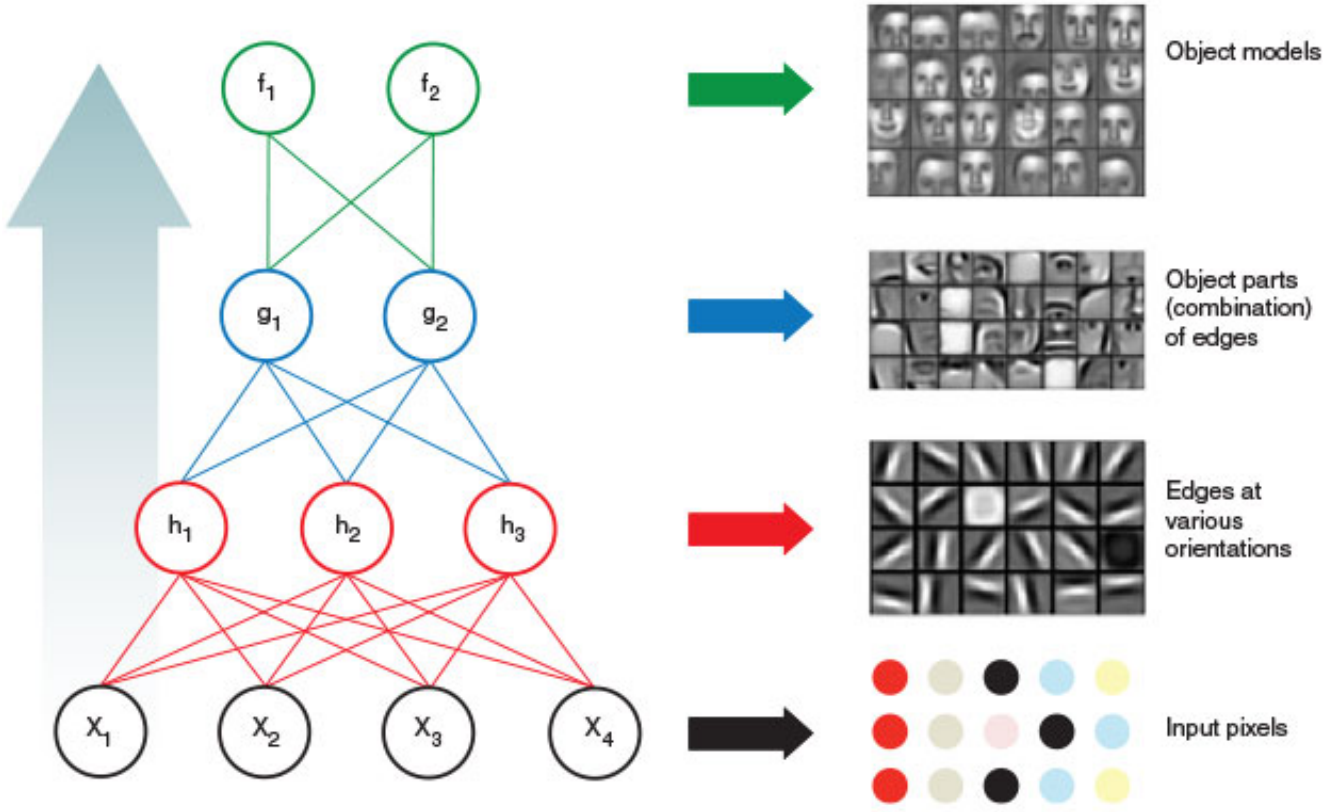
Feature maps “shrink” by $k - 1$ pixels after each convolution with $k \times k$ kernel

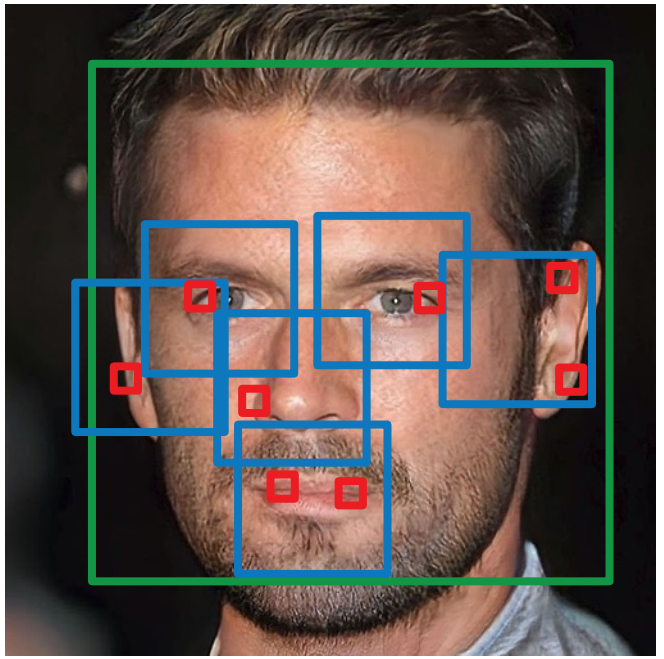
“same”



Feature maps keep the same size

Recall goal of NNs: repeated composition



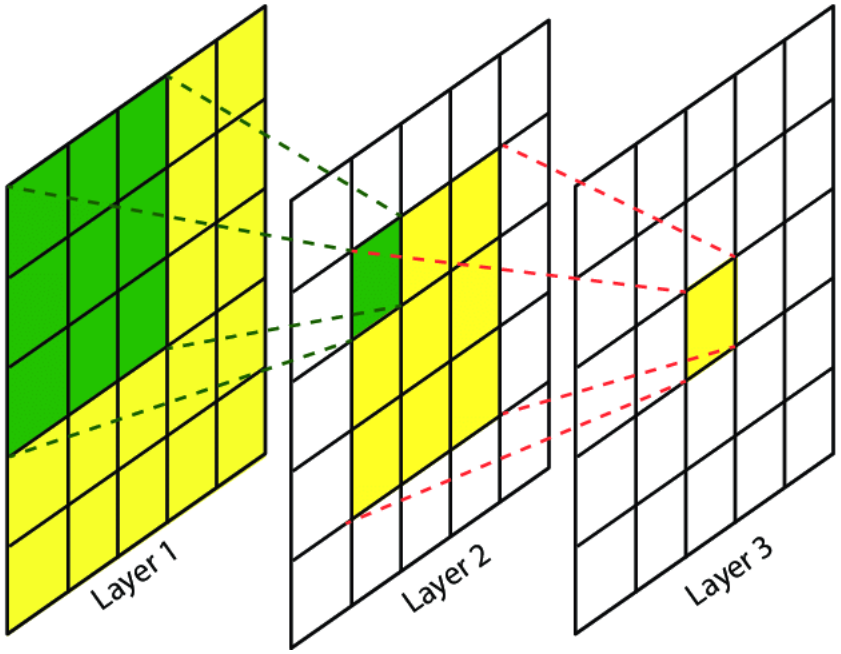


256

256

Source: https://str.llnl.gov/june-2016/chen | BigGAN

Receptive fields



3×3 kernel 3×3 kernel

Receptive field sizes of a vanilla CNN:

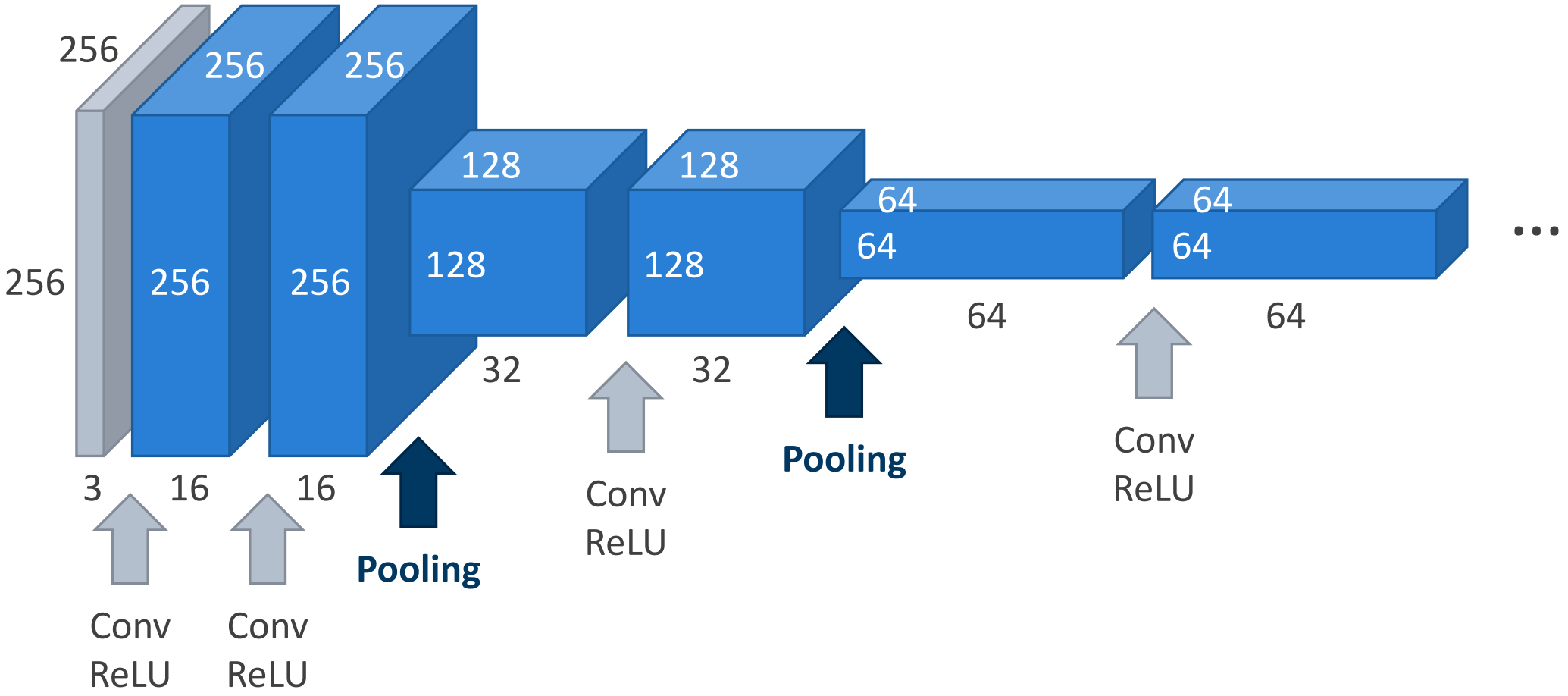
$$\text{RF size} = 1 + \sum_{l=1}^{L} (k_l - 1)$$

Suppose you want to achieve a 200 × 200 px receptive field with 5 × 5 kernels

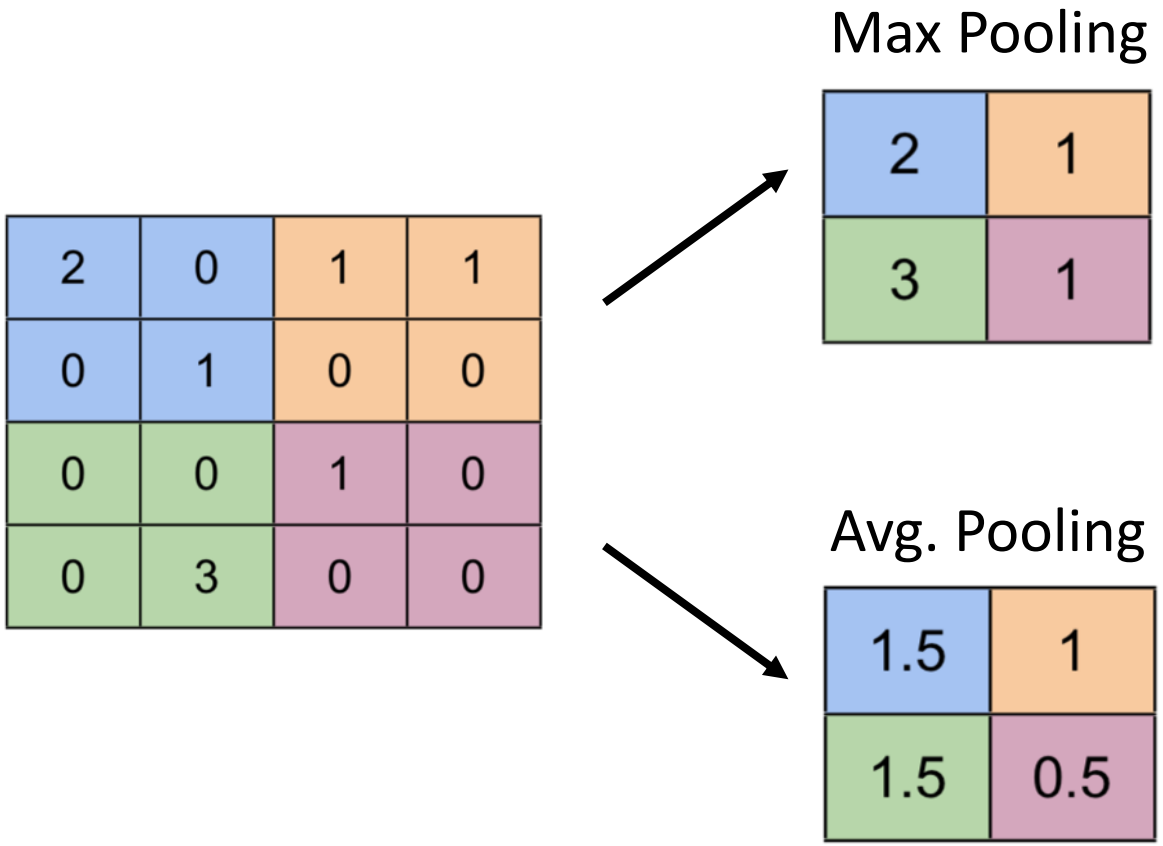
→ **Would require 50 layers!**

Downsammpling the featuresize

Convolutional neural network



Reducing spatial dimensionality by pooling

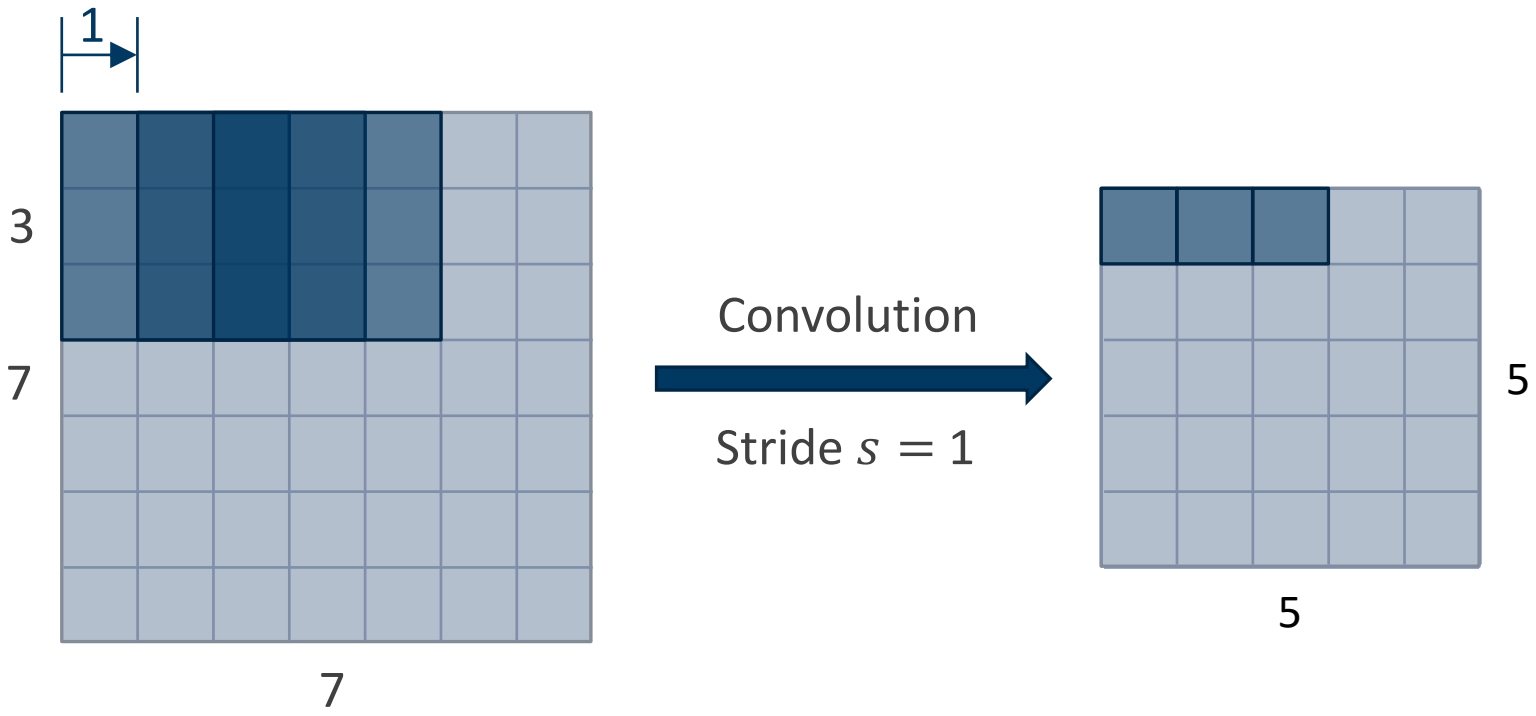


Stride usually equal to size of pooling region

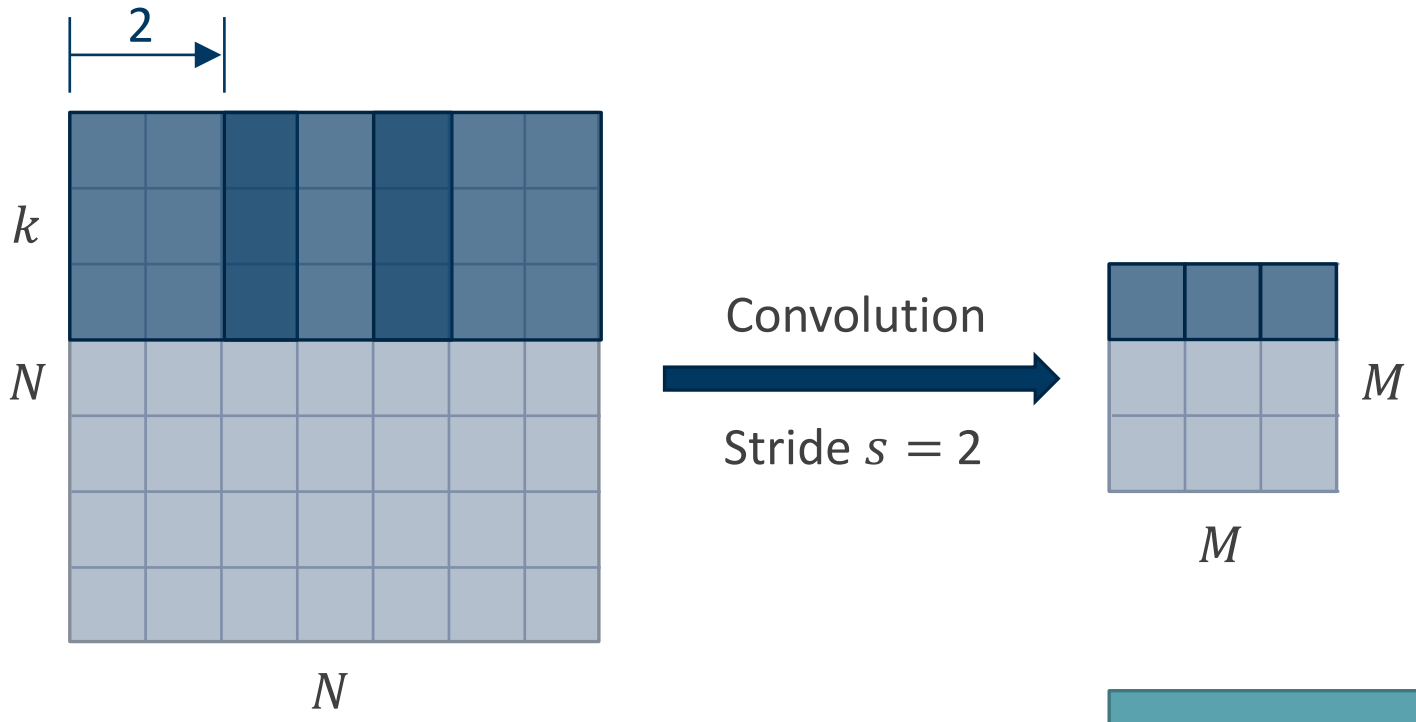
Operates on each feature map independently

Source:https://towardsdatascience.com/a-beginners-guide-to-convolutional-neural-networks-cnns-14649dbddce8

Reducing size by strided convolution



Reducing size by strided convolution

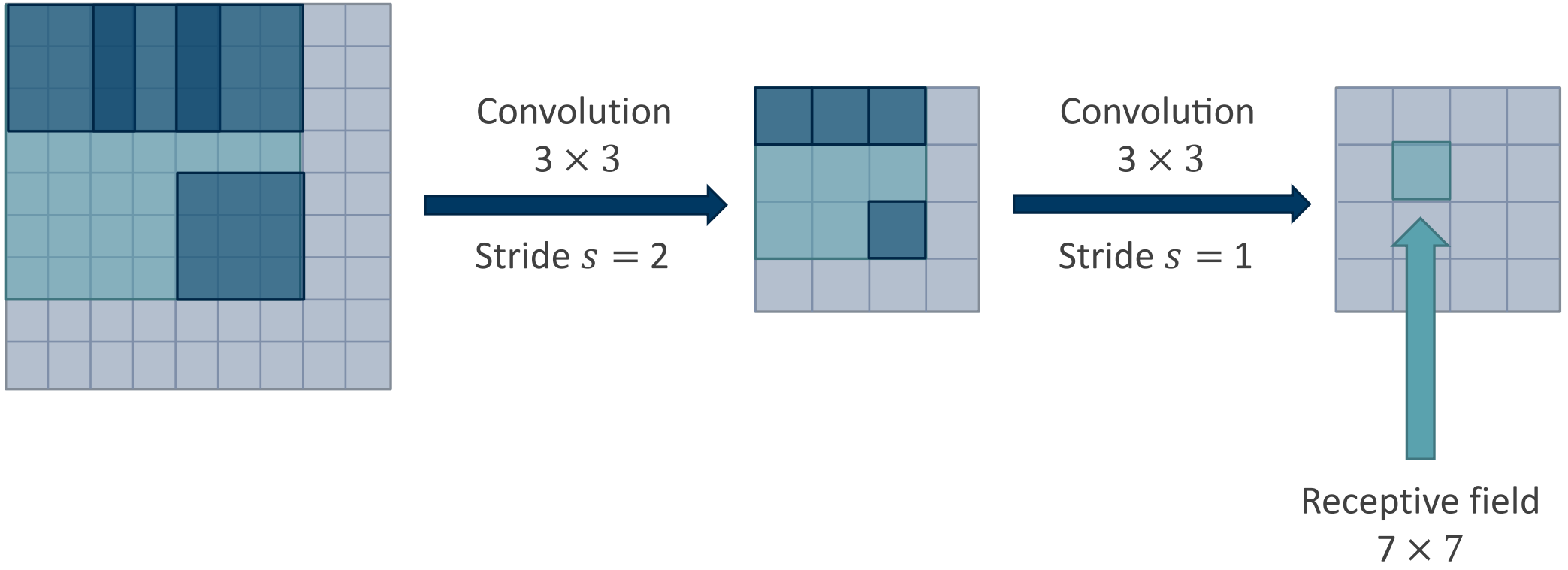


Output size:

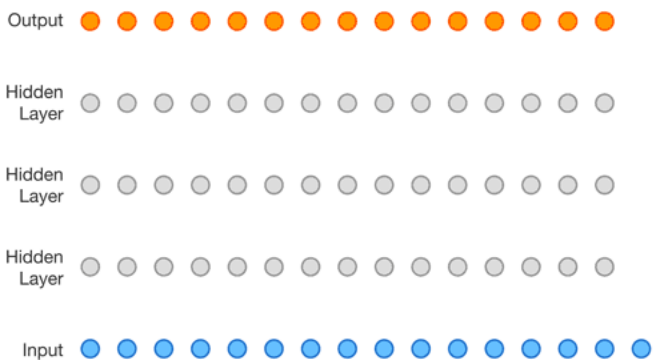
$$M = \frac{N - k}{s} + 1$$

Modern CNN architectures tend to use strided convolutions instead of max pooling

Receptive fields grow faster with pooling layers



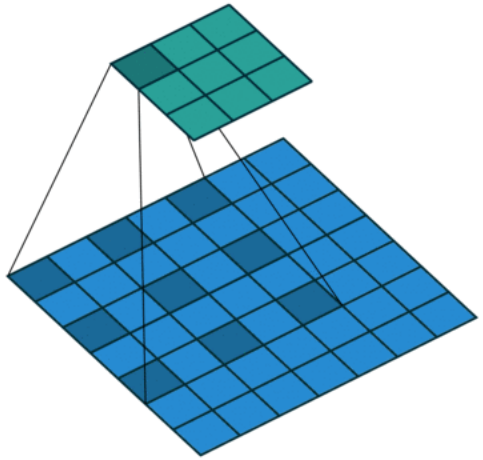
Dilated convolutions



We are skipping pixels

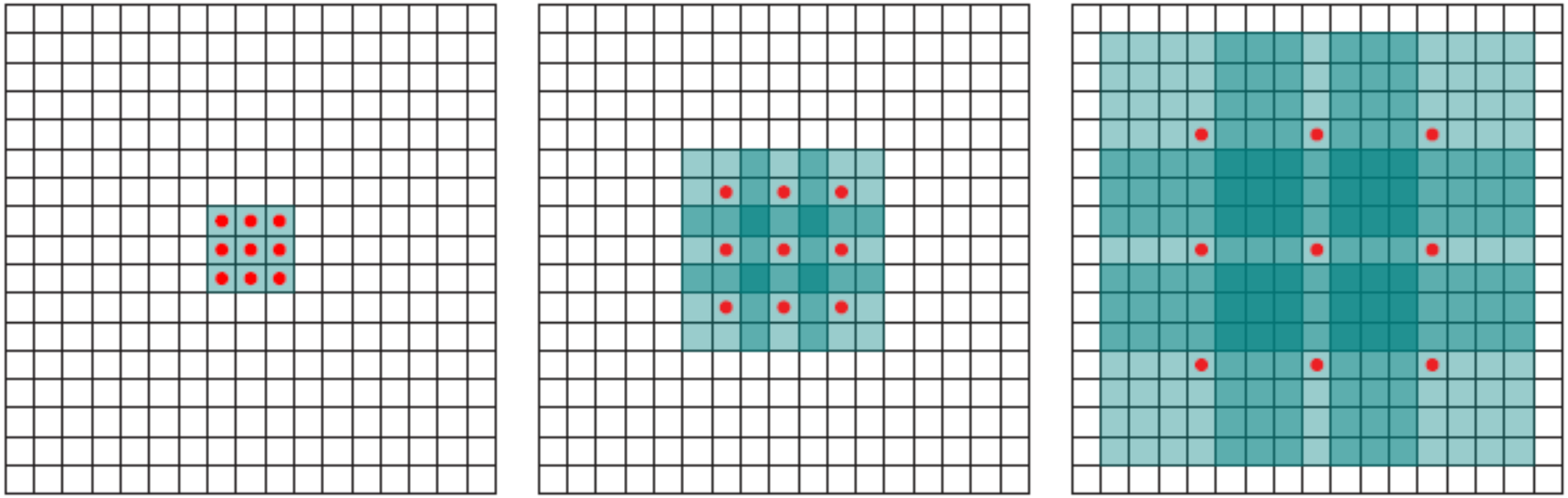
https://deepmind.com/blog/article/wavenet-generative-model-raw-audio

Dilated convolutions in 2D



https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5

Dilated convolutions increase receptive field sizes without reducing resolution



Convolution
3×3
Dilation factor 2

Convolution
3×3
Dilation factor 4

https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5

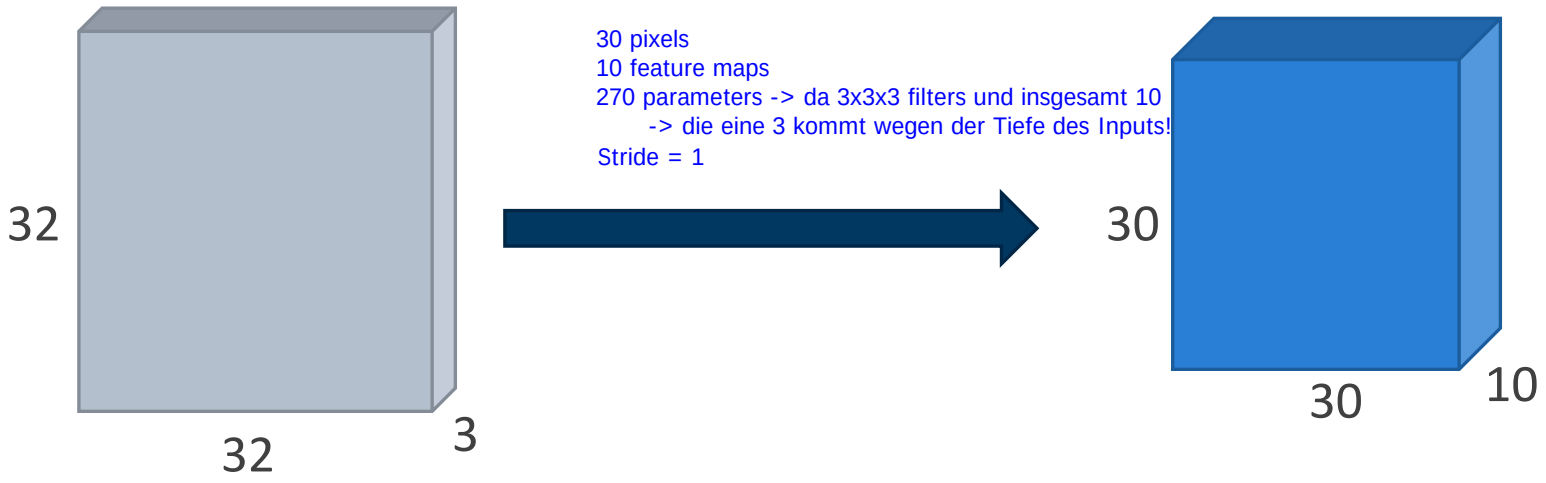
Let’s have a look at dimensions

Valide padding mean that we are not really/ actually padding

Input
$32 \times 32 \times 3$

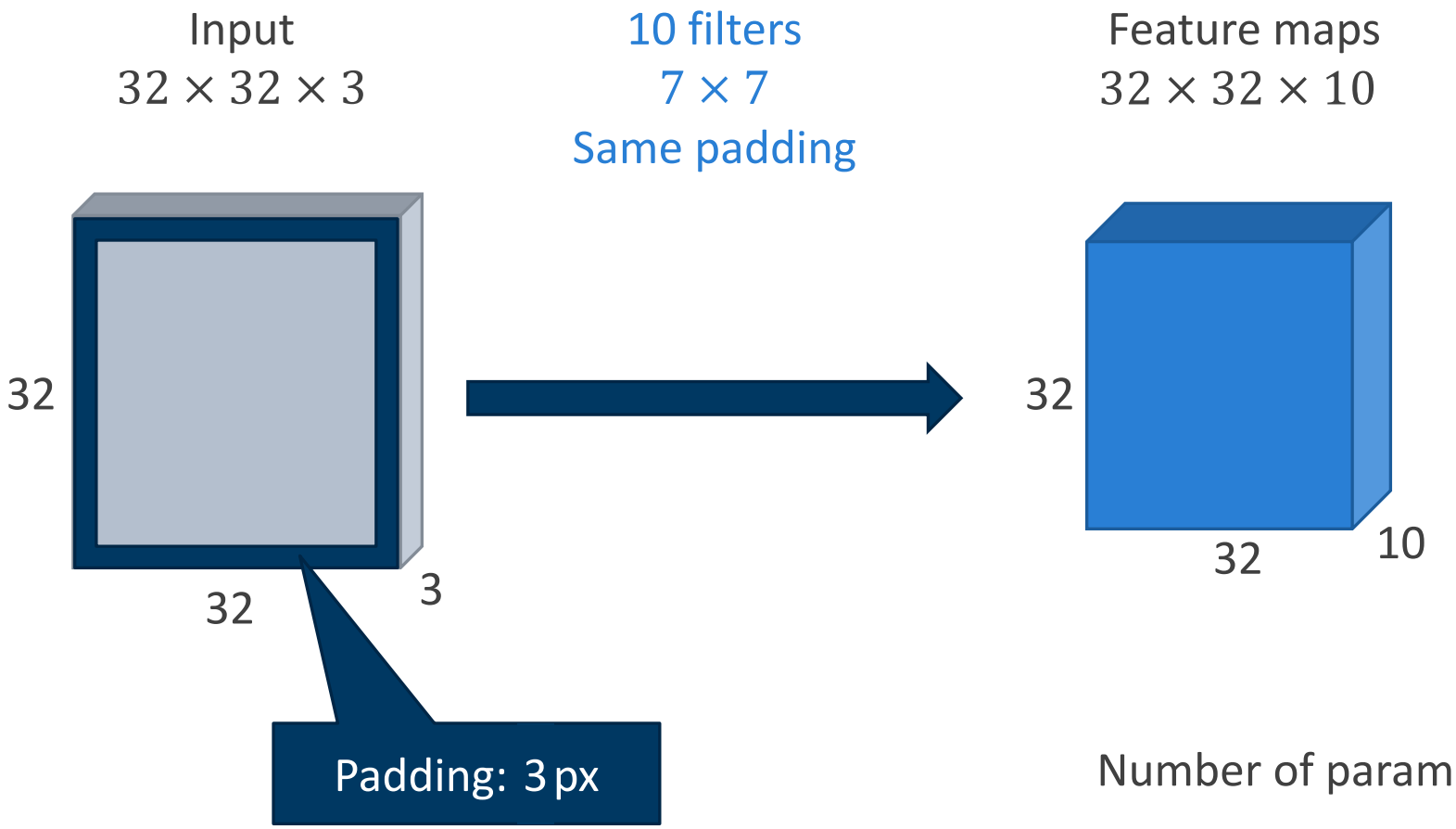
10 filters
3×3
Valid padding

Feature maps
$30 \times 30 \times 10$



Number of parameters: $10 \cdot 3 \cdot 3 \cdot 3 = 270$

Let’s have a look at dimensions



Number of parameters: $10 \cdot 7 \cdot 7 \cdot 3 = 1470$

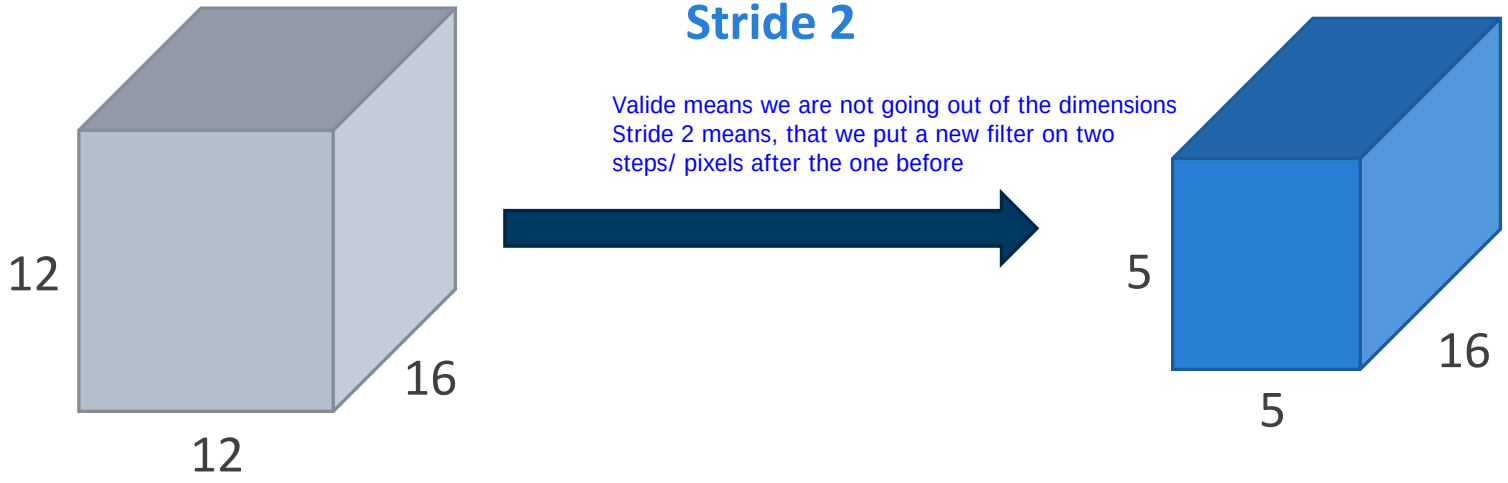
Let’s have a look at dimensions

Input
$12 \times 12 \times 16$

16 filters
3×3
Valid padding
Stride 2

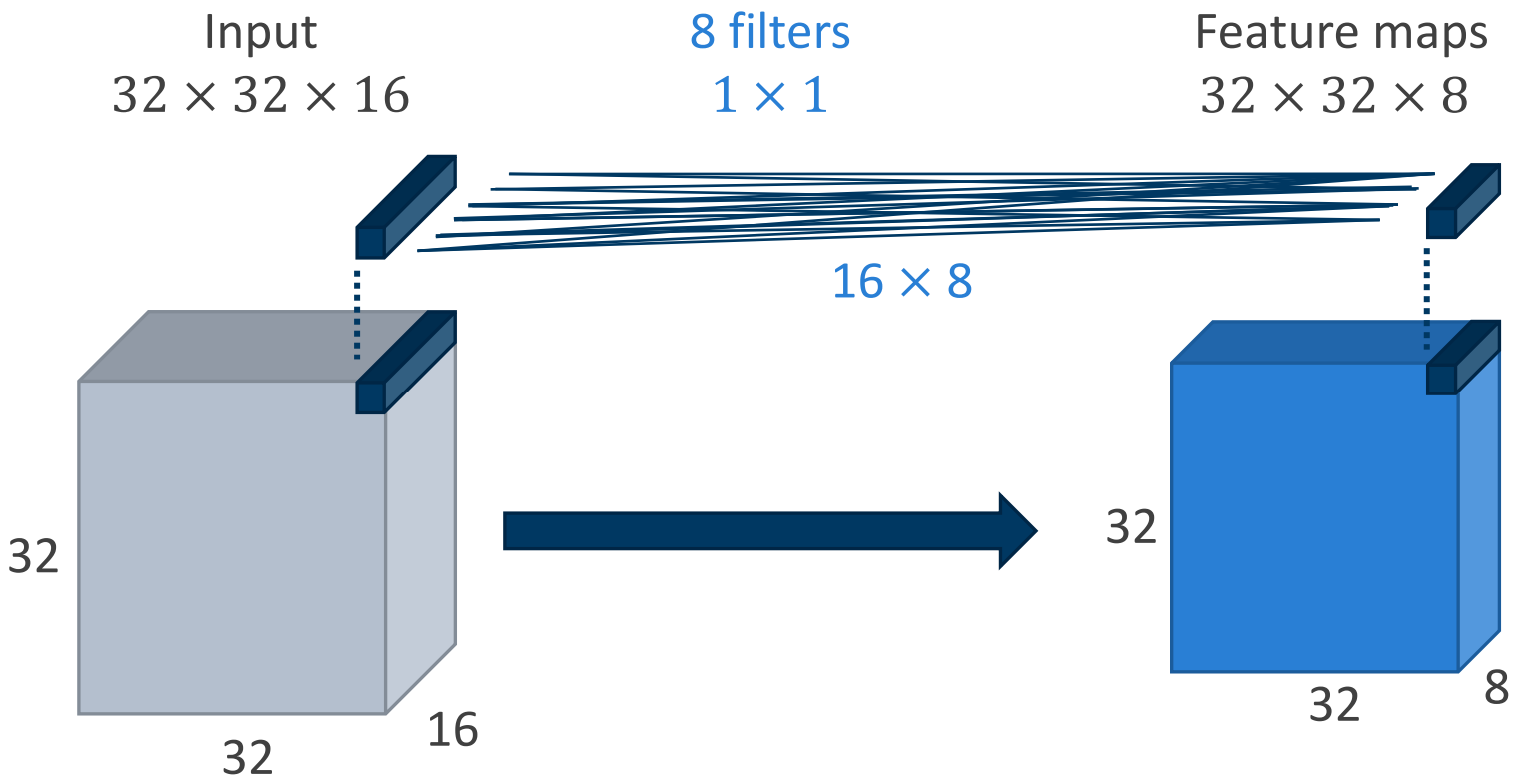
Valide means we are not going out of the dimensions
Stride 2 means, that we put a new filter on two steps/ pixels after the one before

Feature maps
$5 \times 5 \times 16$



Number of parameters: $16 \cdot 3 \cdot 3 \cdot 16 = 2304$

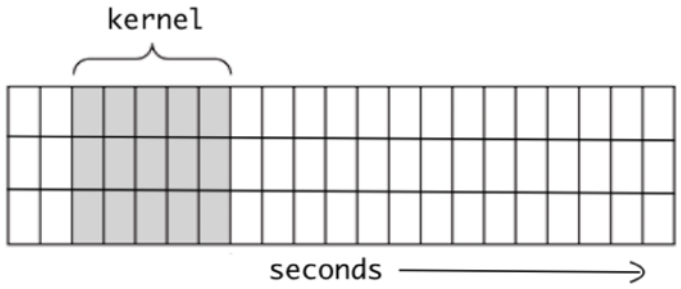
1 × 1 convolutions?



Number of parameters: $16 \cdot 1 \cdot 1 \cdot 8 = 128$

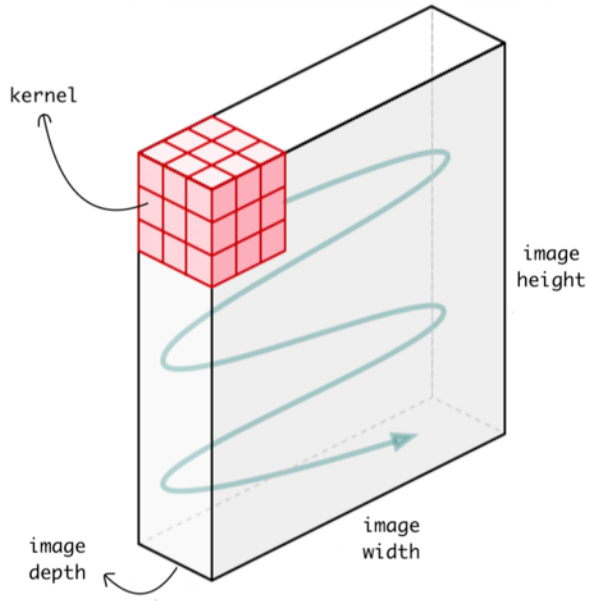
CNNs can be used for 1D, 2D as well as 3D data

1D



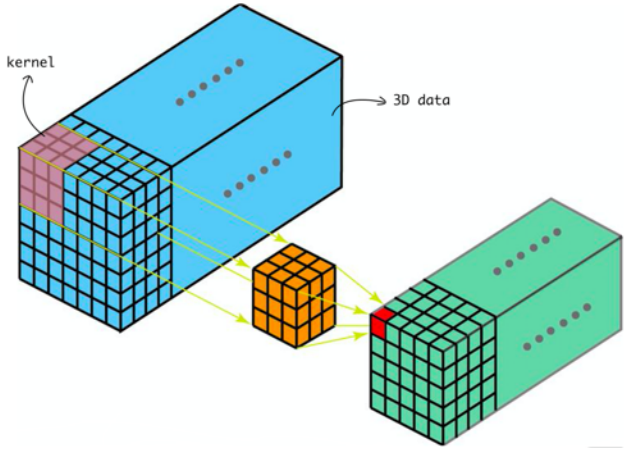
Timeseries

2D



Images

3D

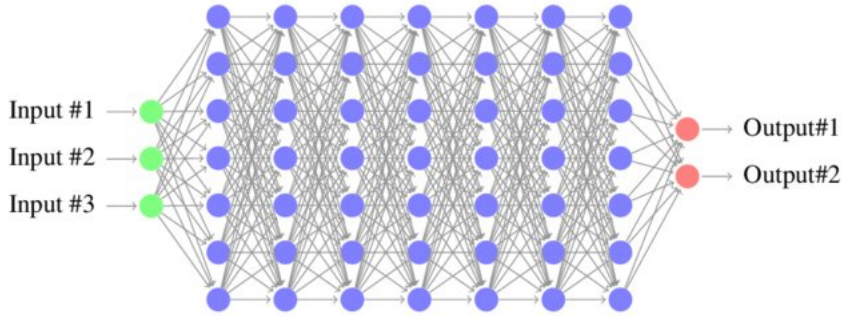


Volumetric images + movies

Source:https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610

Batch normalization

Weight initialization




```
W = 0.01 * np.random.randn(D_in, D_out)
```

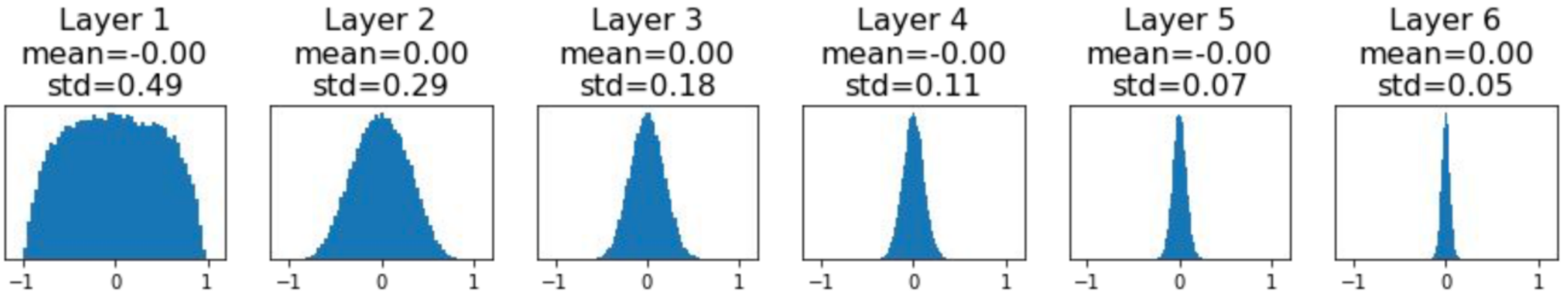
At the very beginning of training, all our gradients and activations are zero

Weight initialization: activation statistics

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for D_in, D_out in zip(dims[:-1], dims[1:]):
    W = 0.01 * np.random.randn(D_in, D_out)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

For deeper networks:
All activations tend to zero in deeper layers

→ No gradients
→ No learning



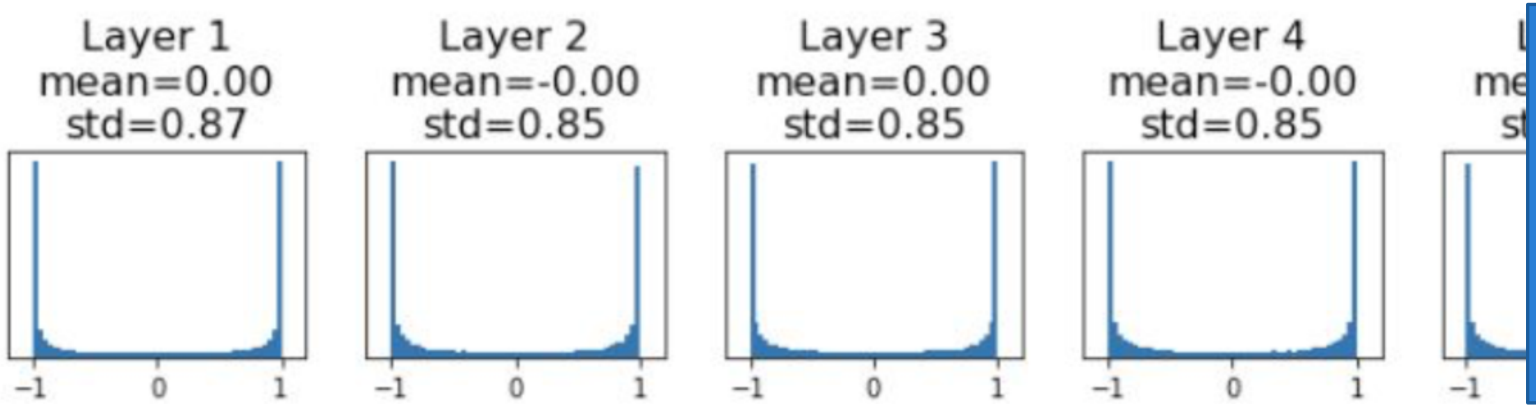
Slide credit: Fei Fei Li, Justin Johnson, Serena Yeung

Weight initialization: activation statistics

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for D_in, D_out in zip(dims[:-1], dims[1:]):
    W = 0.05 * np.random.randn(D_in, D_out)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

For deeper networks:
All activations saturate

→ No gradients
→ No learning



In case of ReLU
Activations/gradients explode

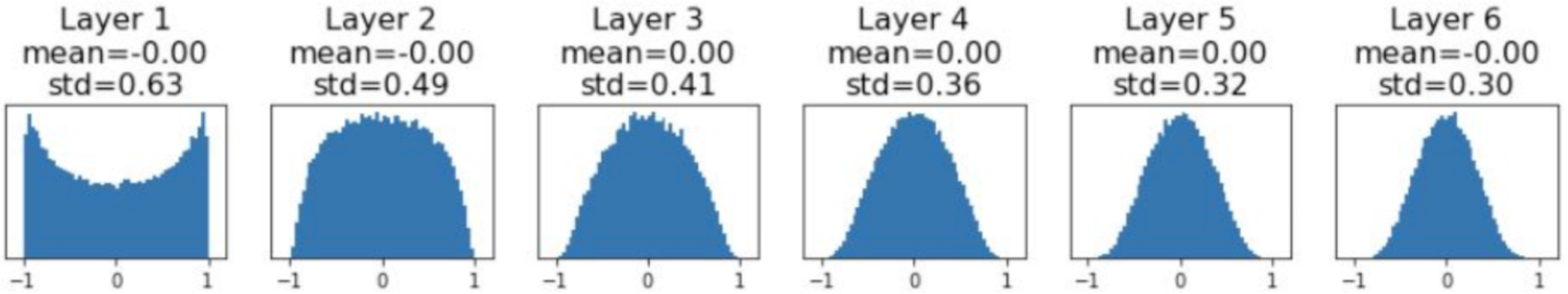
Slide credit: Fei Fei Li, Justin Johnson, Serena Yeung

we assume all the weights are independent

Weight initialization: Xavier initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for D_in, D_out in zip(dims[:-1], dims[1:]):
    W = np.random.randn(D_in, D_out) / np.sqrt(D_in)
    x = np.tanh(x.dot(W))
    hs.append(x)
```

Glorot & Bengio, AISTATS 2010

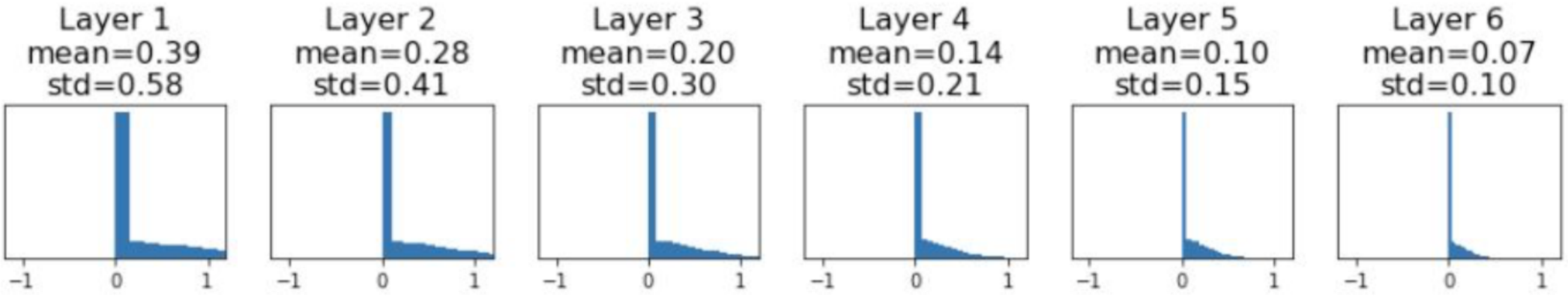


Slide credit: Fei Fei Li, Justin Johnson, Serena Yeung

Weight initialization: What about ReLU?

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for D_in, D_out in zip(dims[:-1], dims[1:]):
    W = np.random.randn(D_in, D_out) / np.sqrt(2 / D_in)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Glorot & Bengio, AISTATS 2010

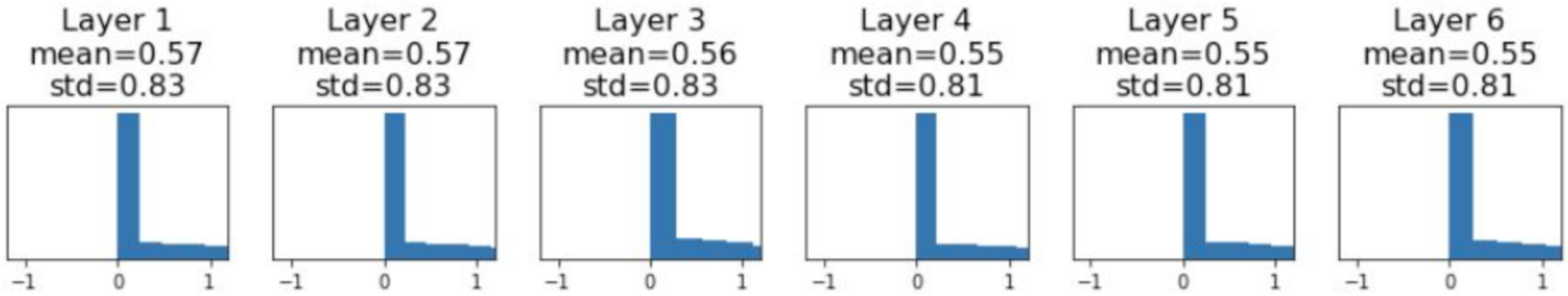


Slide credit: Fei Fei Li, Justin Johnson, Serena Yeung

Weight initialization: Kaiming / MSRA initialization

```
dims = [4096] * 7
hs = []
x = np.random.randn(16, dims[0])
for D_in, D_out in zip(dims[:-1], dims[1:]):
    W = np.random.randn(D_in, D_out) / np.sqrt(2 / D_in)
    x = np.maximum(0, x.dot(W))
    hs.append(x)
```

Glorot & Bengio, AISTATS 2010



Slide credit: Fei Fei Li, Justin Johnson, Serena Yeung

Proper initialization is an active area of research…

Understanding the difficulty of training deep feedforward neural networks
by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks
by Saxe et al, 2013

Random walk initialization for training very deep feedforward networks
by Sussillo and Abbott, 2014

Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification
by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks
by Krähenbühl et al., 2015

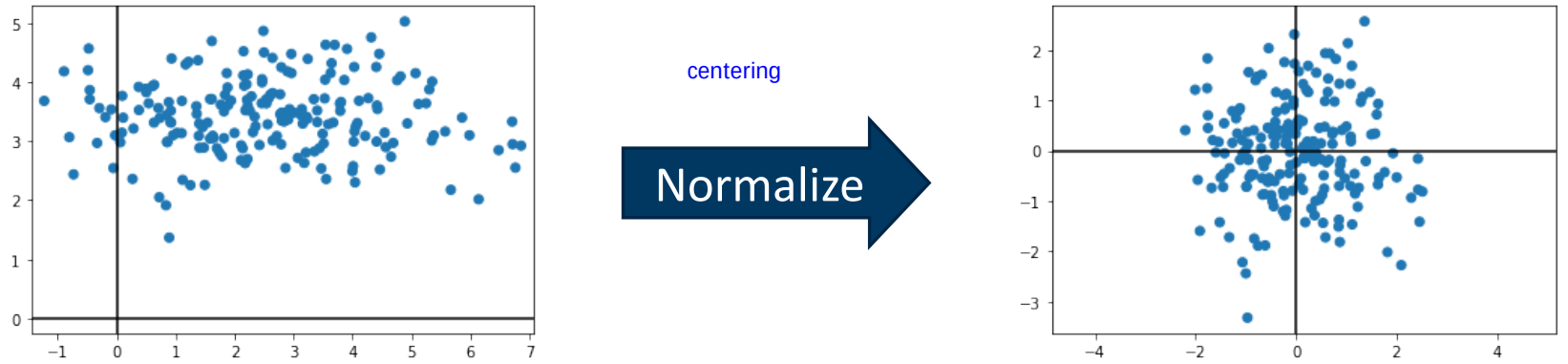
All you need is a good init,
by Mishkin and Matas, 2015

Fixup Initialization: Residual Learning Without Normalization
by Zhang et al, 2019

The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks
by Frankle and Carbin, 2019

Slide credit: Fei Fei Li, Justin Johnson, Serena Yeung

Batch normalization: motivation

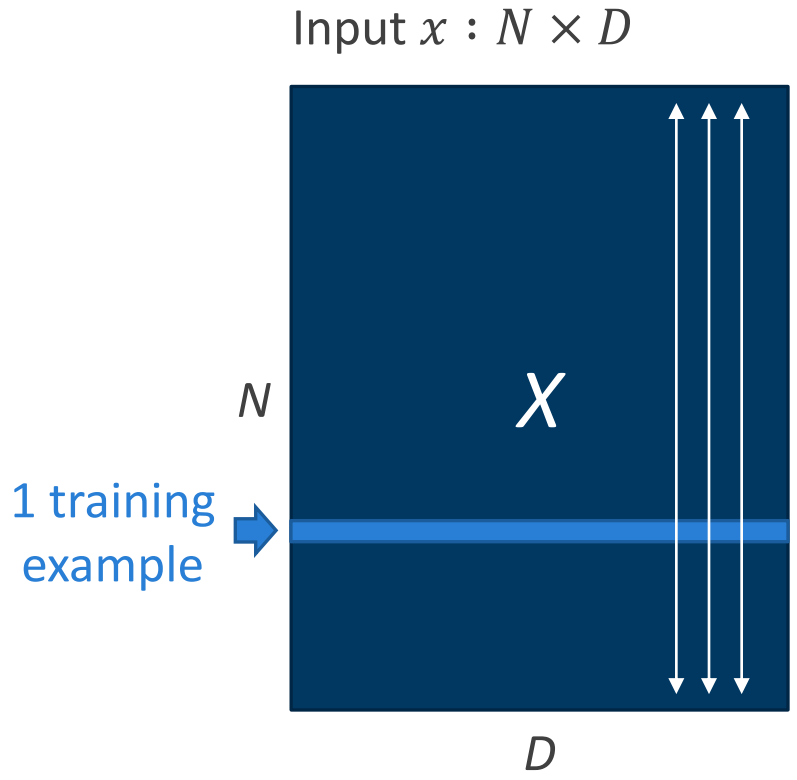


It is common practice to normalize the inputs to have zero mean, unit variance

→ Do the same for intermediate layers' activations

$$\text{For each layer: } \hat{x} = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]}}$$

Batch normalization



$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{i,j}$$

Per-channel mean (*D*-dim)

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}\left(x_{i,j} - \mu_j\right)^2$$

Per-channel variance (*D*-dim)

$$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\sigma_j^2 + \epsilon}}$$

Normalized *x* (shape: $N \times D$)

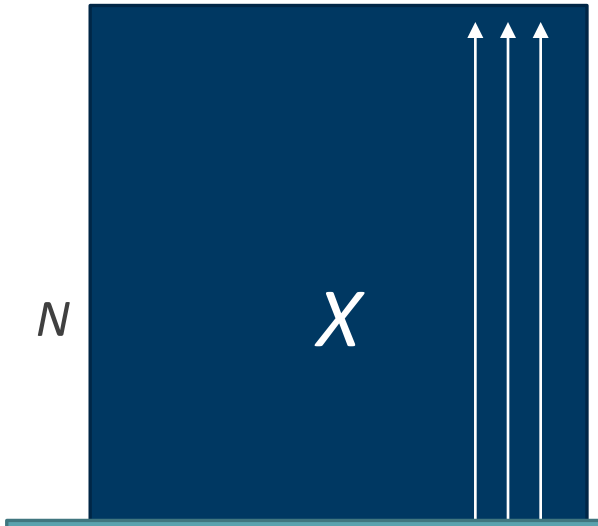
$$y_{i,j} = \gamma_j \hat{x}_{i,j} + \beta_j$$

β_j, γ_j learnable parameters

Ioffe & Szegedy 2015

Batch normalization at test time

Input x : $N \times D$



$\mu_j =$ Running average of values during training — Per-channel mean (*D*-dim)

$\sigma_j^2 =$ Running average of values during training — Per-channel variance (*D*-dim)

$\hat{x}_{i,j} = \frac{x_{i,j} - \mu_j}{\sqrt{\quad}}$ — Normalized *x* (shape: $N \times D$)

At test time batch norm is a linear transformation
→ Can be absorbed into weights of next layer
→ Batch norm does not change the representational capacity of a network

Ioffe & Szegedy 2015

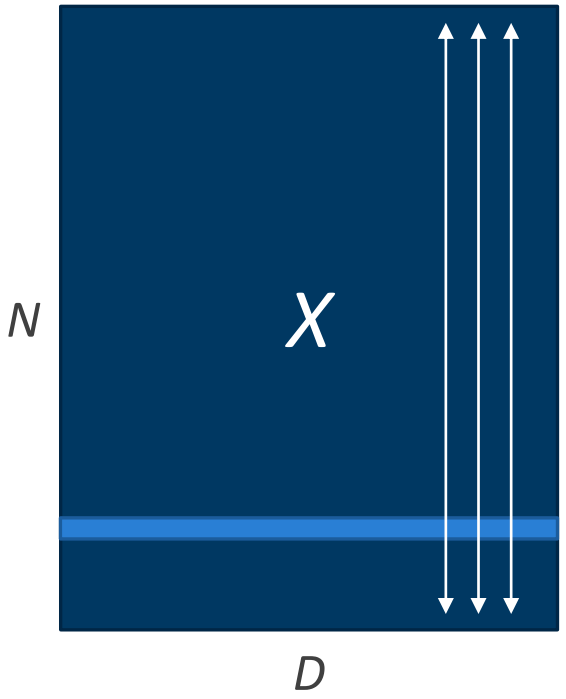
Batch norm: CNNs

Fully connected network

Input x $N \times D$

↓ Normalize

$\mu, \sigma, \beta, \gamma$ $1 \times D$



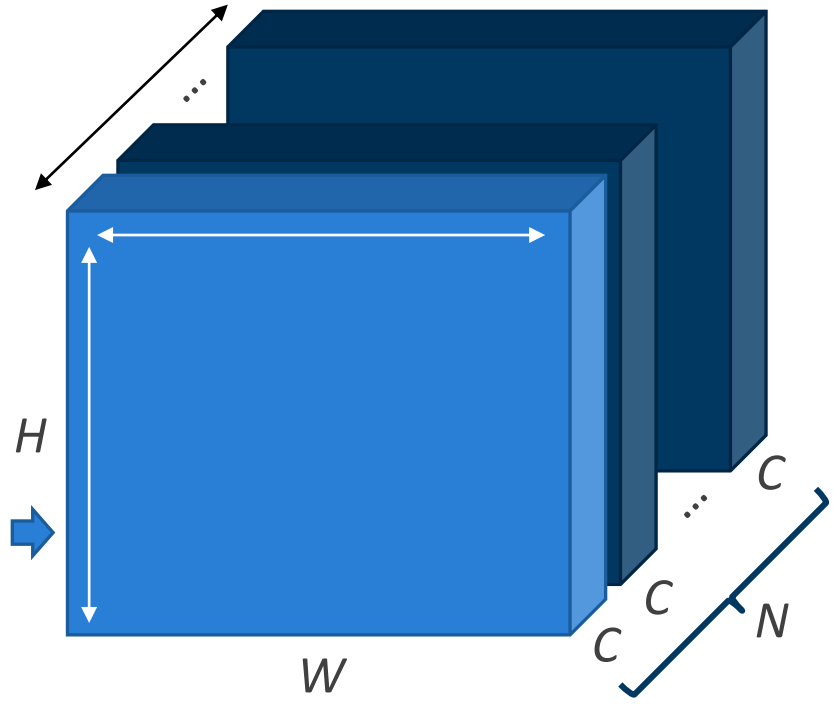
1 training example

CNN

$N \times H \times W \times C$

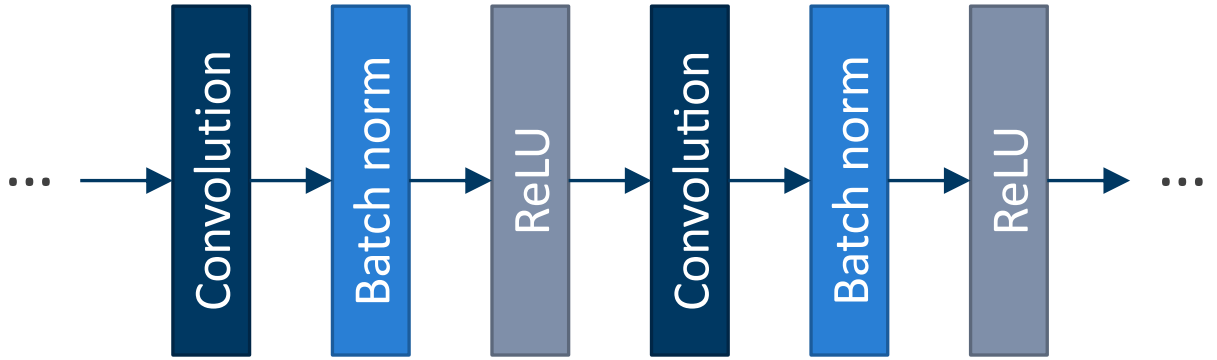
Normalize ↓ ↓ ↓

$1 \times 1 \times 1 \times C$

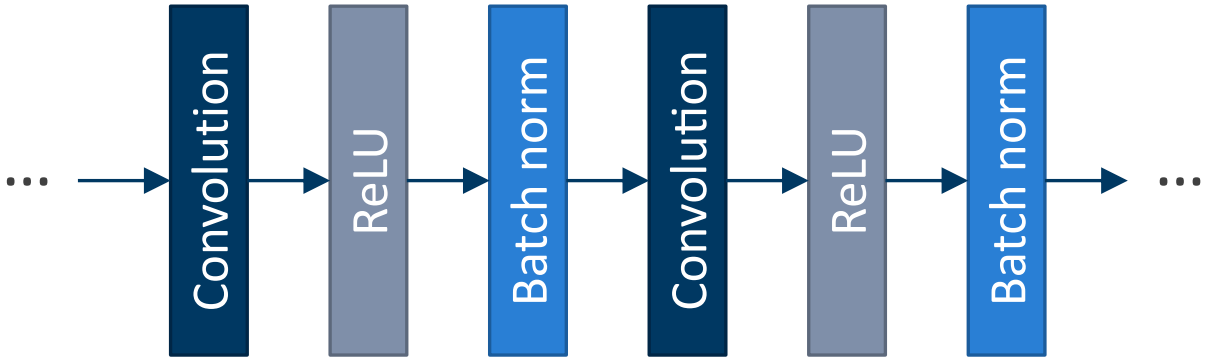


Where to place batch norm layer?

Original proposal
Ioffe & Szegedy 2015



Alternative placement



Layer normalization

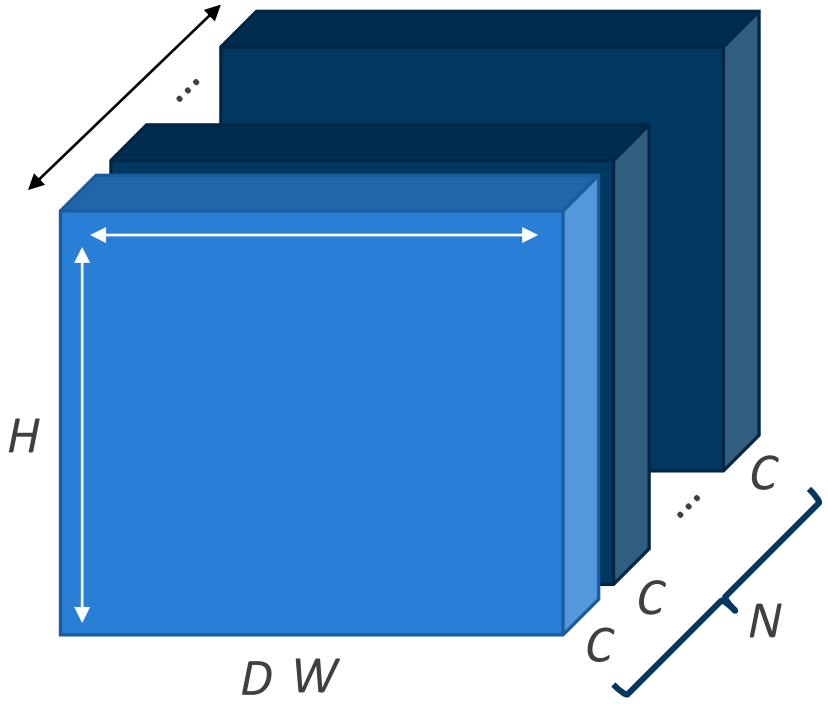
Same behavior at training and test time!

Batch norm

Input x $N \times H \times W \times C$

Normalize

$\mu, \sigma, \beta, \gamma$ $1 \times 1 \times 1 \times C$

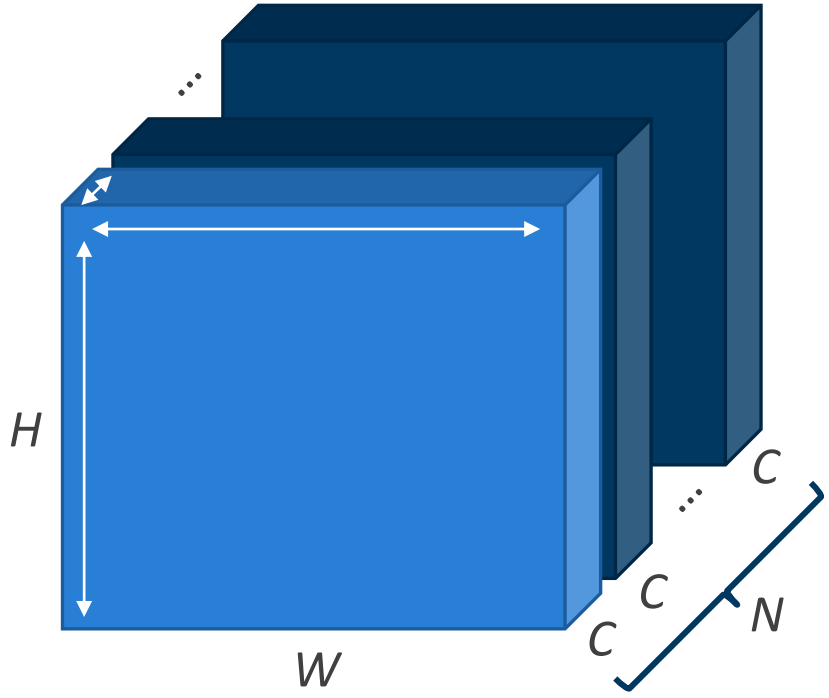


Layer norm

$N \times H \times W \times C$

Normalize

$N \times 1 \times 1 \times 1$



Instance normalization

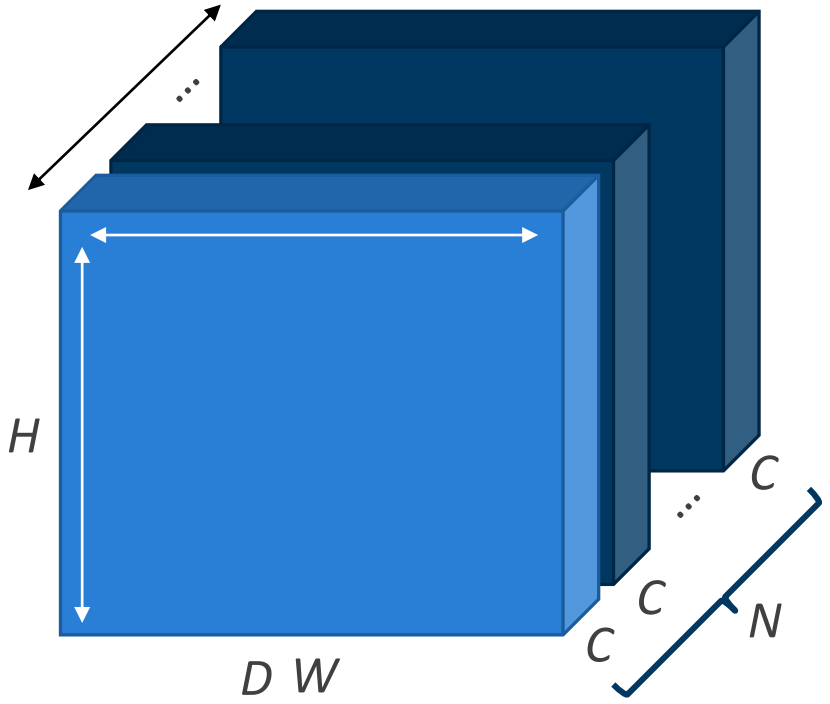
Same behavior at training and test time!

Batch norm

Input x $N \times H \times W \times C$

Normalize

$\mu, \sigma, \beta, \gamma$ $1 \times 1 \times 1 \times C$

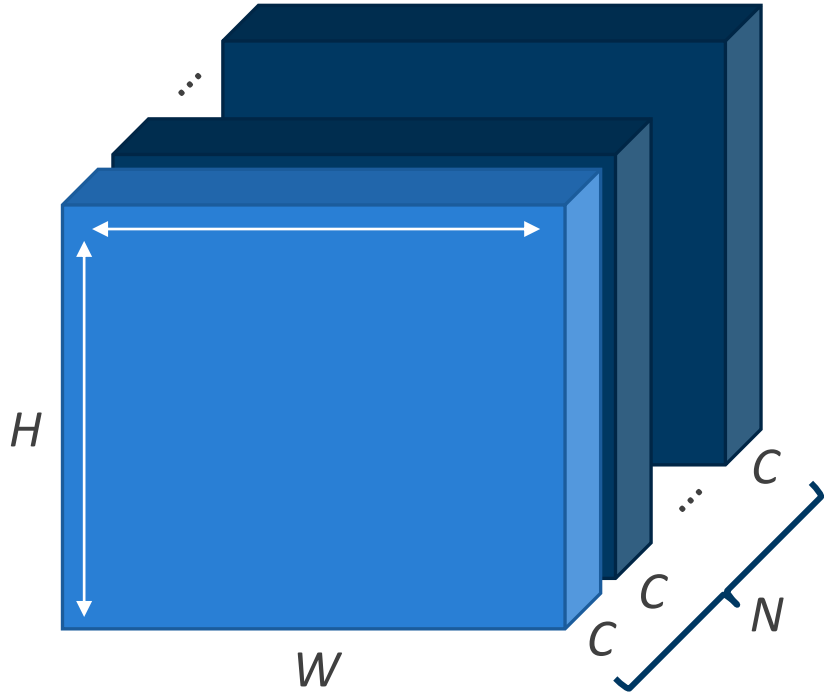


Instance norm

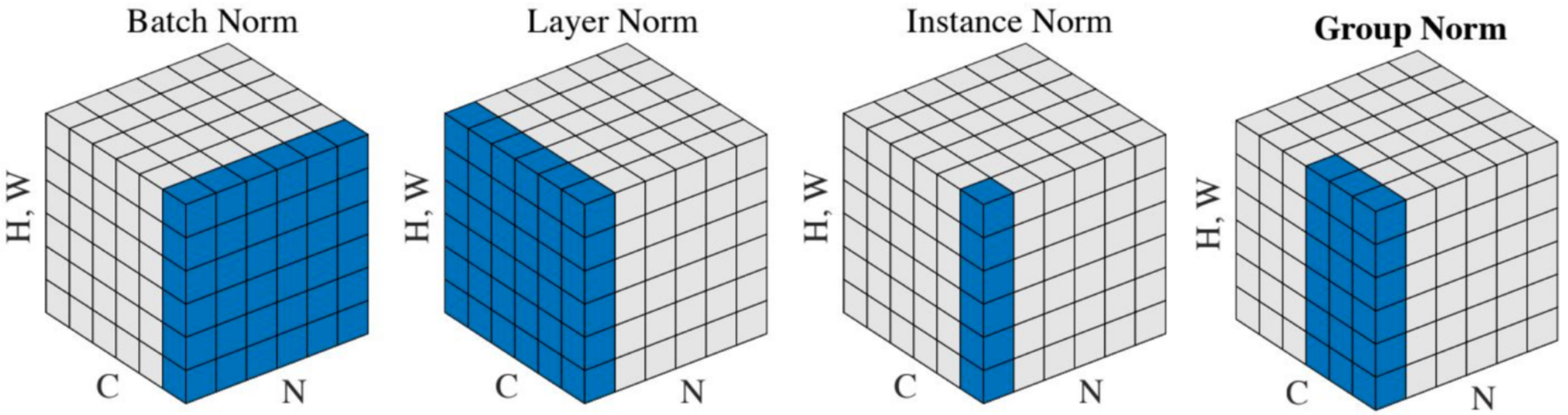
$N \times H \times W \times C$

Normalize

$N \times 1 \times 1 \times C$



Comparison of normalization layers



Wu and He, “Group Normalization”, ECCV 2018

Slide credit: Fei Fei Li, Justin Johnson, Serena Yeung

Questions!