DATA ANALYTICS USING DEEP LEARNING

GT 8803 // FALL 2019 // JOY ARULRAJ

LECTURE #10:CONVOLUTIONAL NEURAL NETWORKS

CREATING THE NEXT®

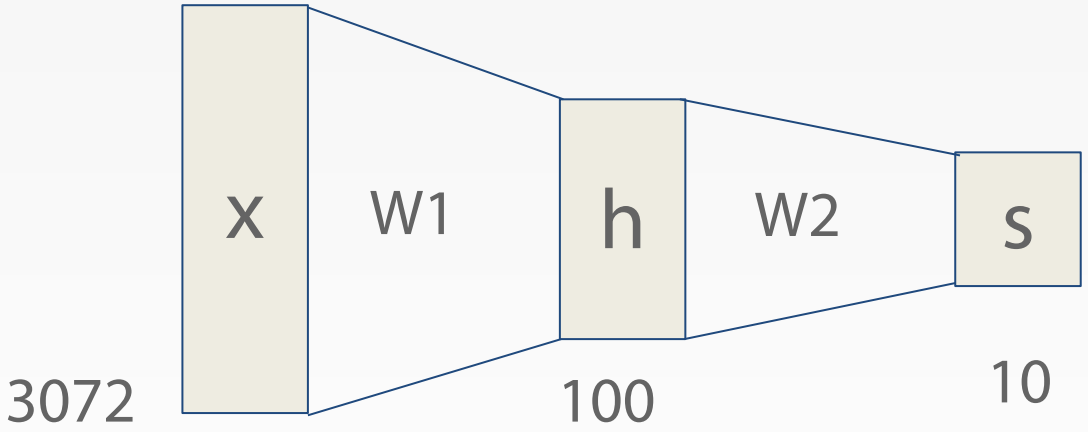
ADMINISTRIVIA

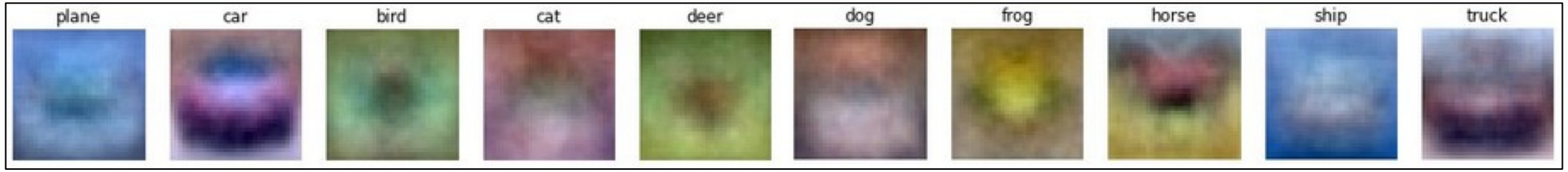
- Reminders
  - Assignment 1 due today
  - Assignment 2 released today
  - Proposal presentations postponed to Monday

LAST CLASS

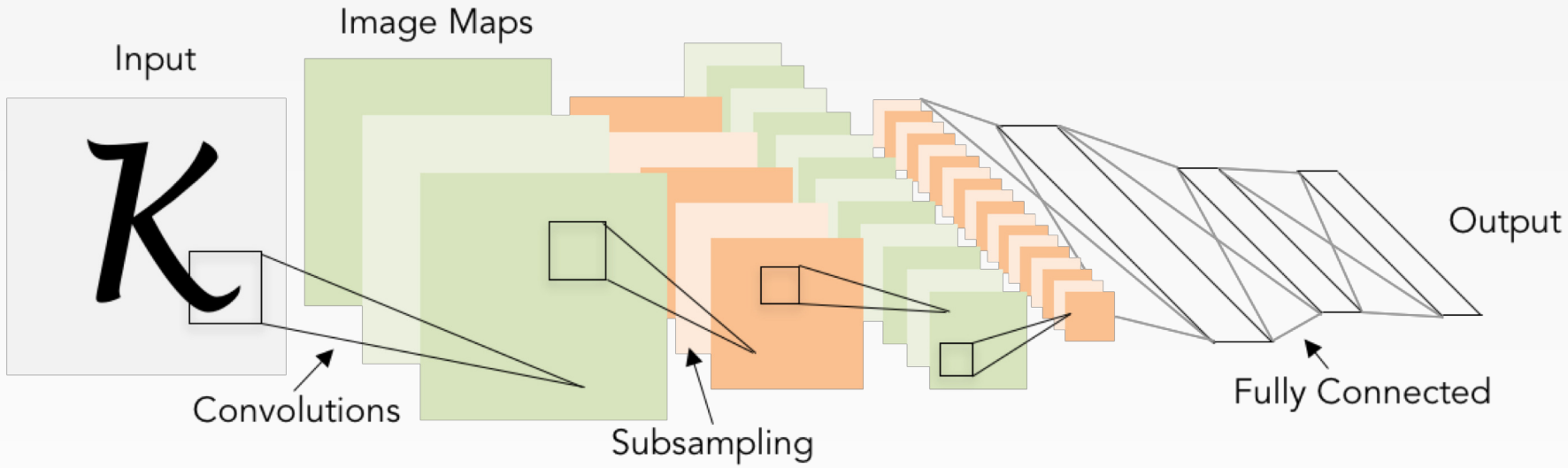
Linear score function: $f = Wx$

2-layer Neural Network $f = W_2 \max(0, W_1 x)$





TODAY'S CLASS



TODAY'S AGENDA

- History of Neural Networks
- Convolutional Layer
- Convolutional Neural Networks

HISTORY OF NEURAL NETWORKS

PERCEPTRON (1957)

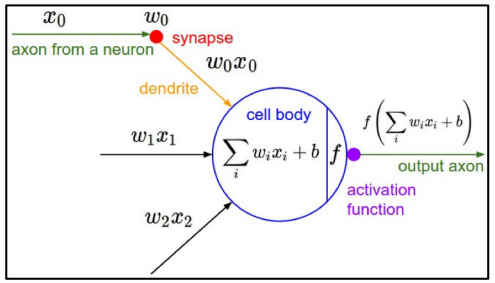
The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

- The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.
- recognized letters of the alphabet

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

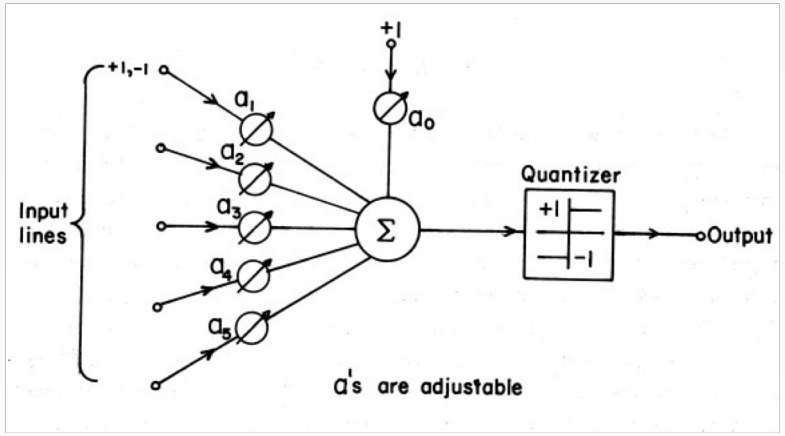
update rule:

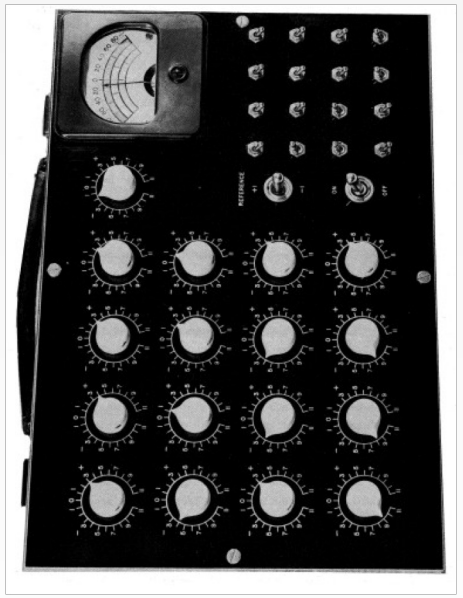
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i}$$

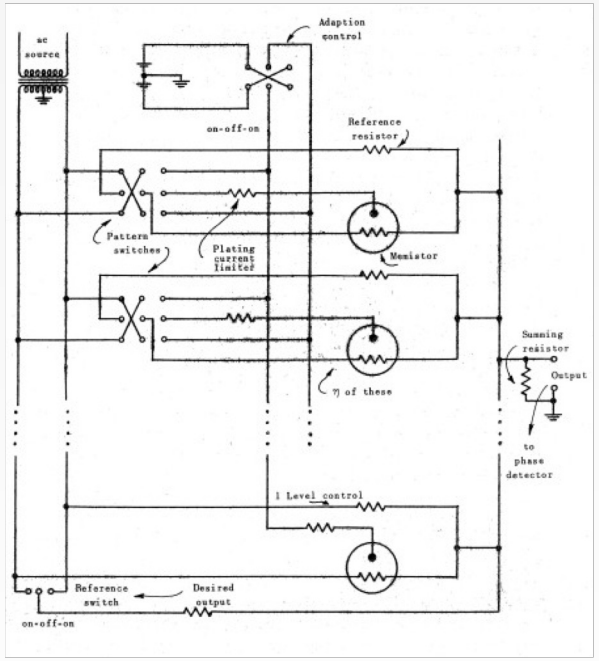


Frank Rosenblatt, ~1957: Perceptron

MADALINE (1960)



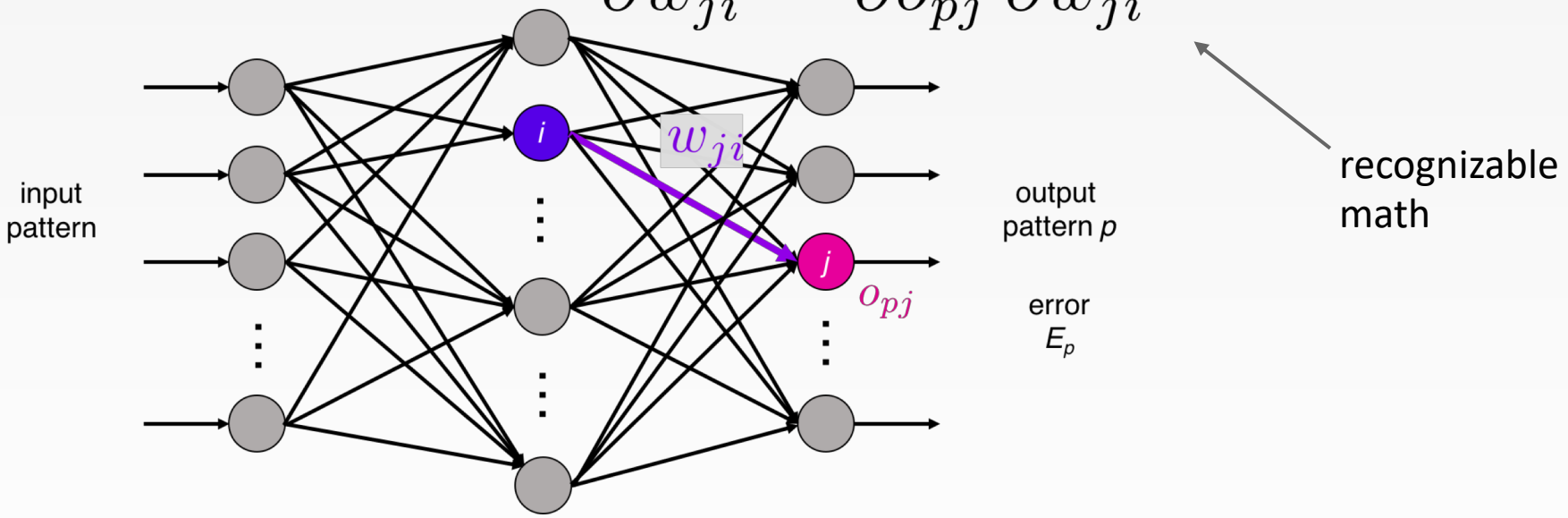




Widrow and Hoff, ~1960: Adaline/Madaline

BACK-PROPAGATION (1986)

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}$$

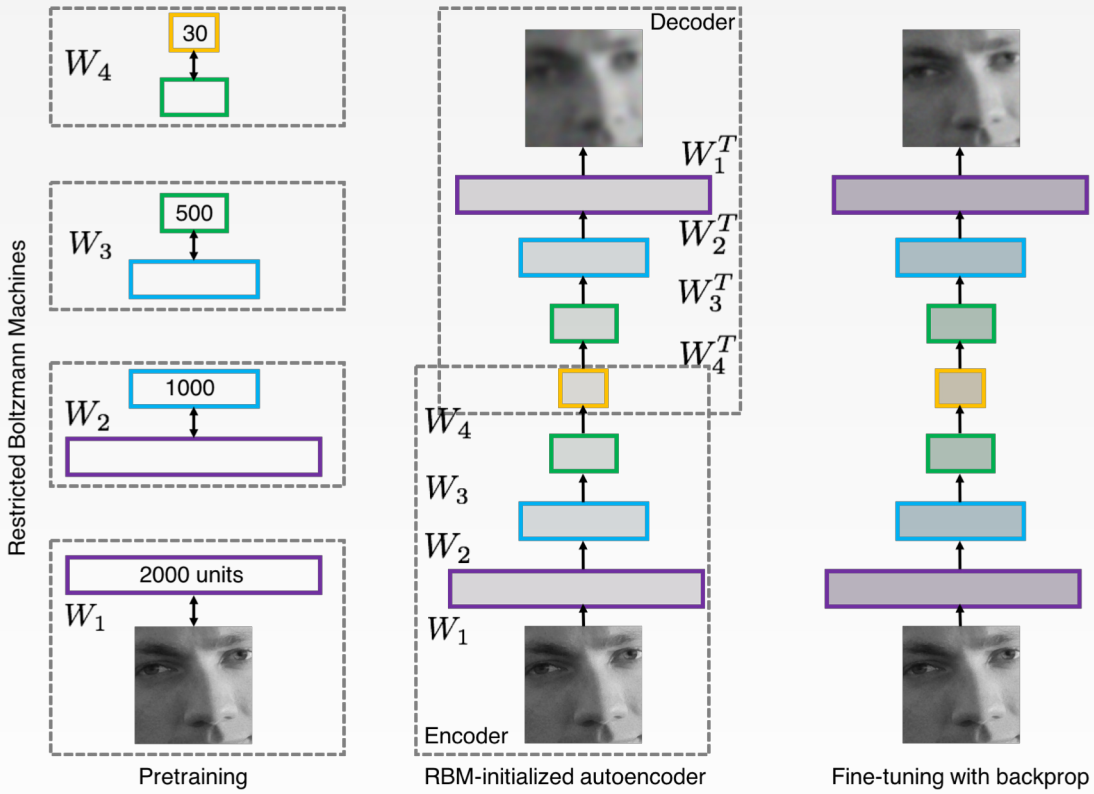


Rumelhart et al., 1986: First time back-propagation became popular

DEEP NEURAL NETWORK (2006)

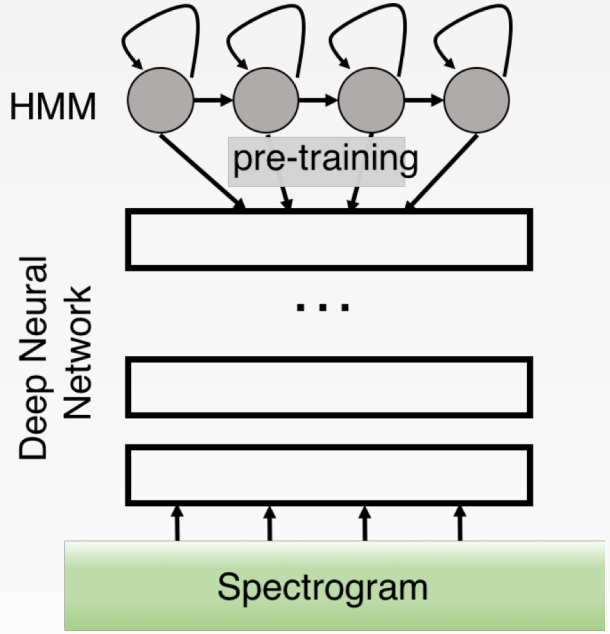
- Reinvigorated research in Deep Learning

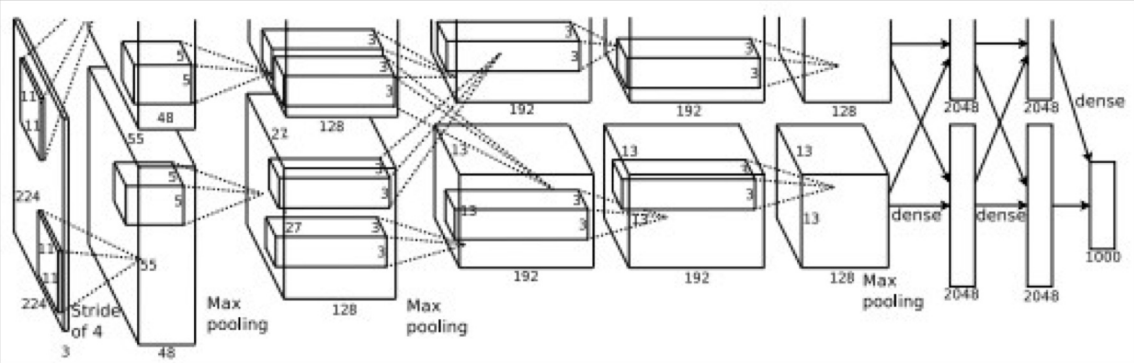
Hinton and Salakhutdinov, 2006

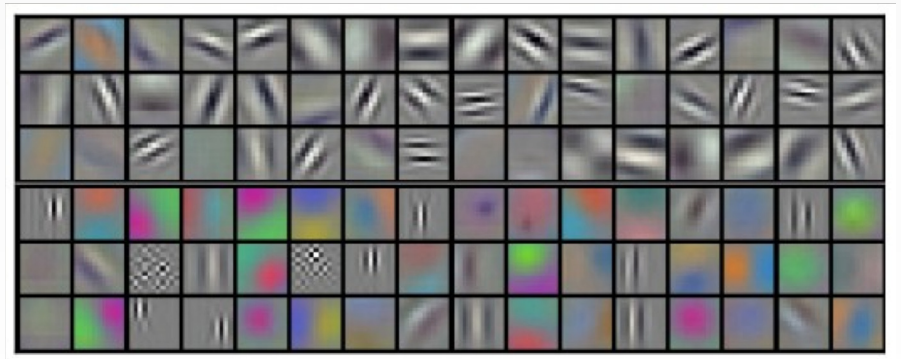


FIRST STRONG RESULTS (2012)

- ***Acoustic Modeling using Deep Belief Networks***
  - *Abdel-Rahman Mohamed, George Dahl, Geoffrey Hinton, 2010*
- ***Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition***
  - George Dahl, Dong Yu, Li Deng, Alex Acero, 2012
- ***Imagenet classification with deep convolutional neural networks***
  - Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012







WHAT GAVE RISE TO CNNs?

Hubel & Wiesel,

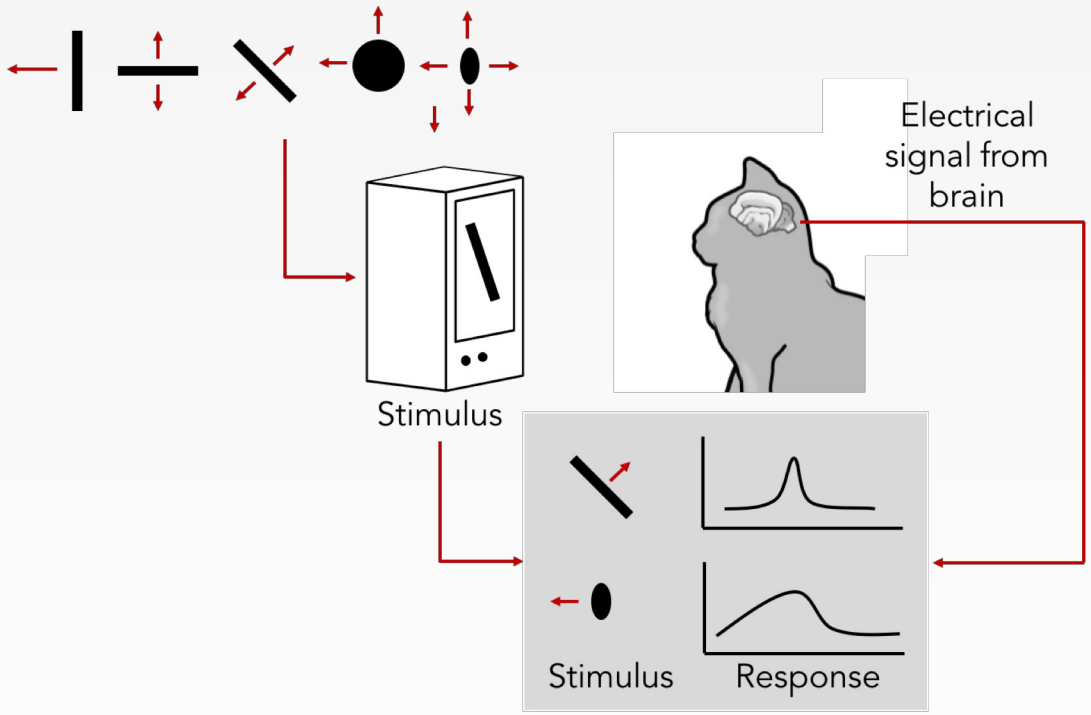
1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX

1962

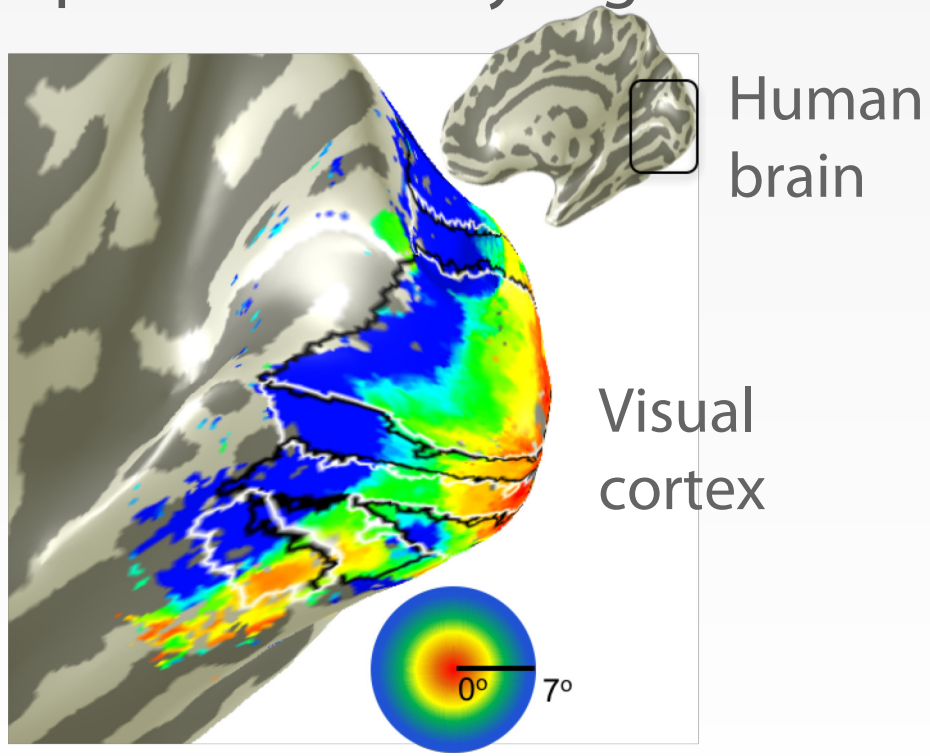
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...

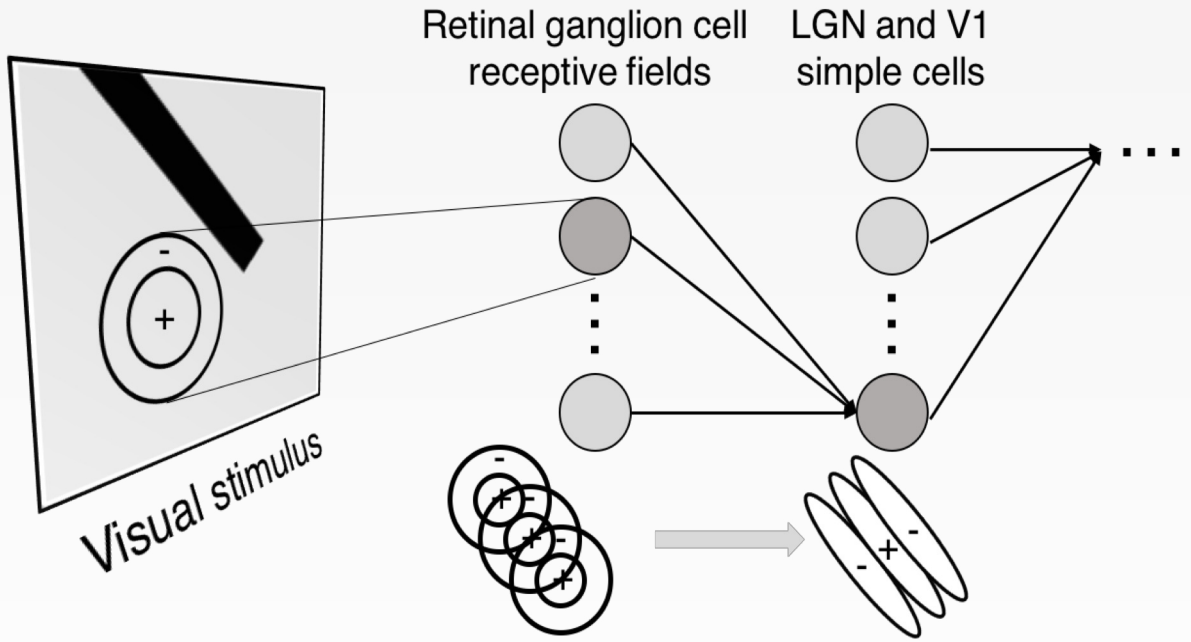


#1: SPATIAL ORGANIZATION

- **Topographical mapping in the cortex:**
  - nearby cells in cortex represent nearby regions in the visual field



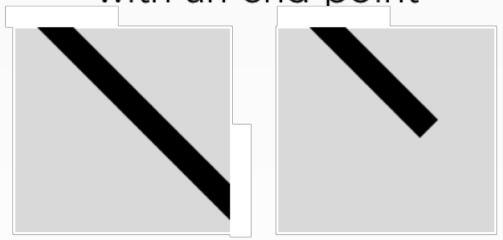
#2: HIERARCHICAL ORGANIZATION



Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

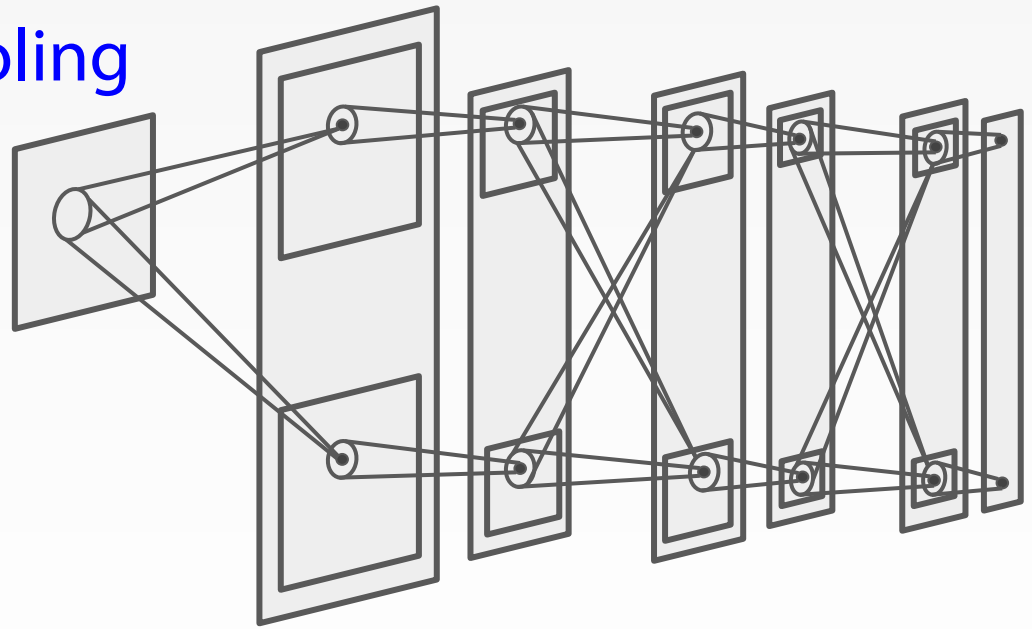
Hypercomplex cells:
response to movement with an end point



NEOCOGNITRON (1980)

- Sandwich architecture
  - (SCSCSC…)
  - simple cells: modifiable parameters
  - complex cells: perform pooling

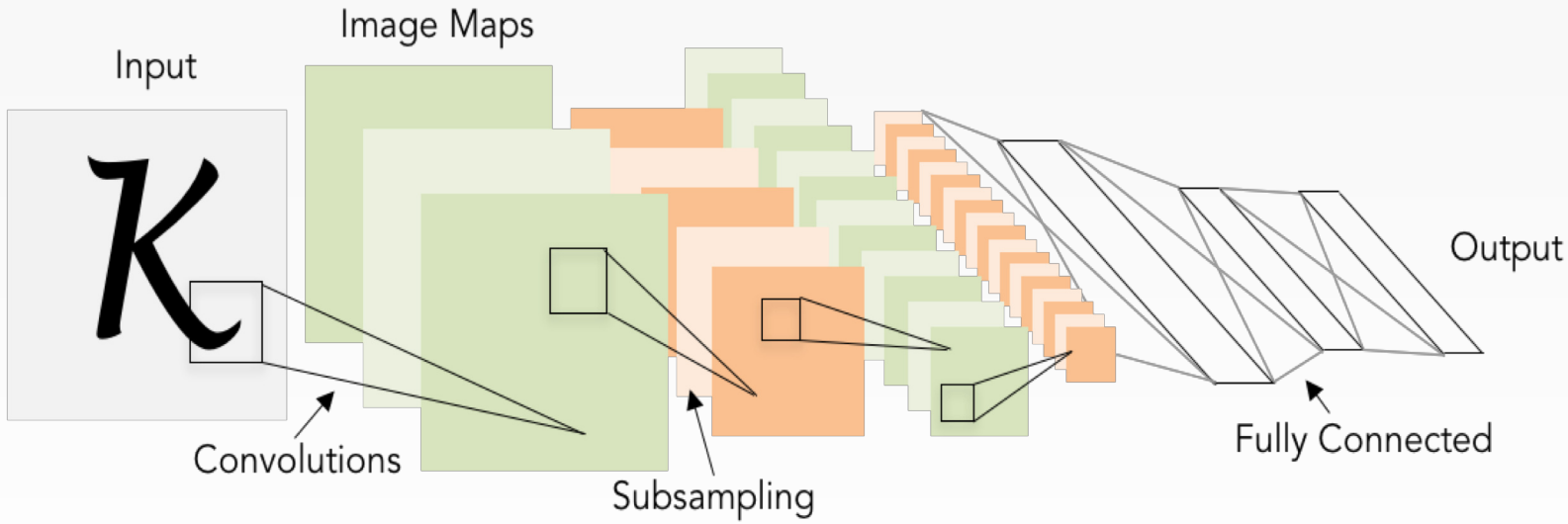
[Fukushima, 1980]



ZIPCODE RECOGNITION (1998)

Gradient-based learning applied to document recognition

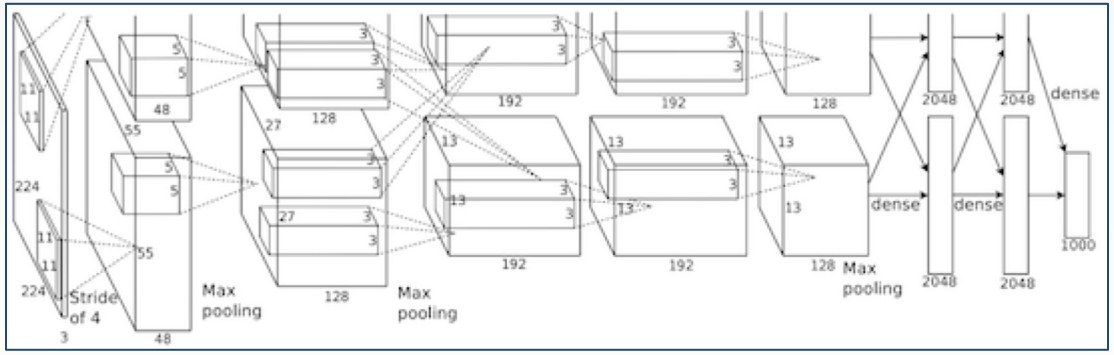
[*LeCun, Bottou, Bengio, Haffner 1998*]



ALEXNET (2012)

ImageNet Classification with Deep Convolutional Neural Networks

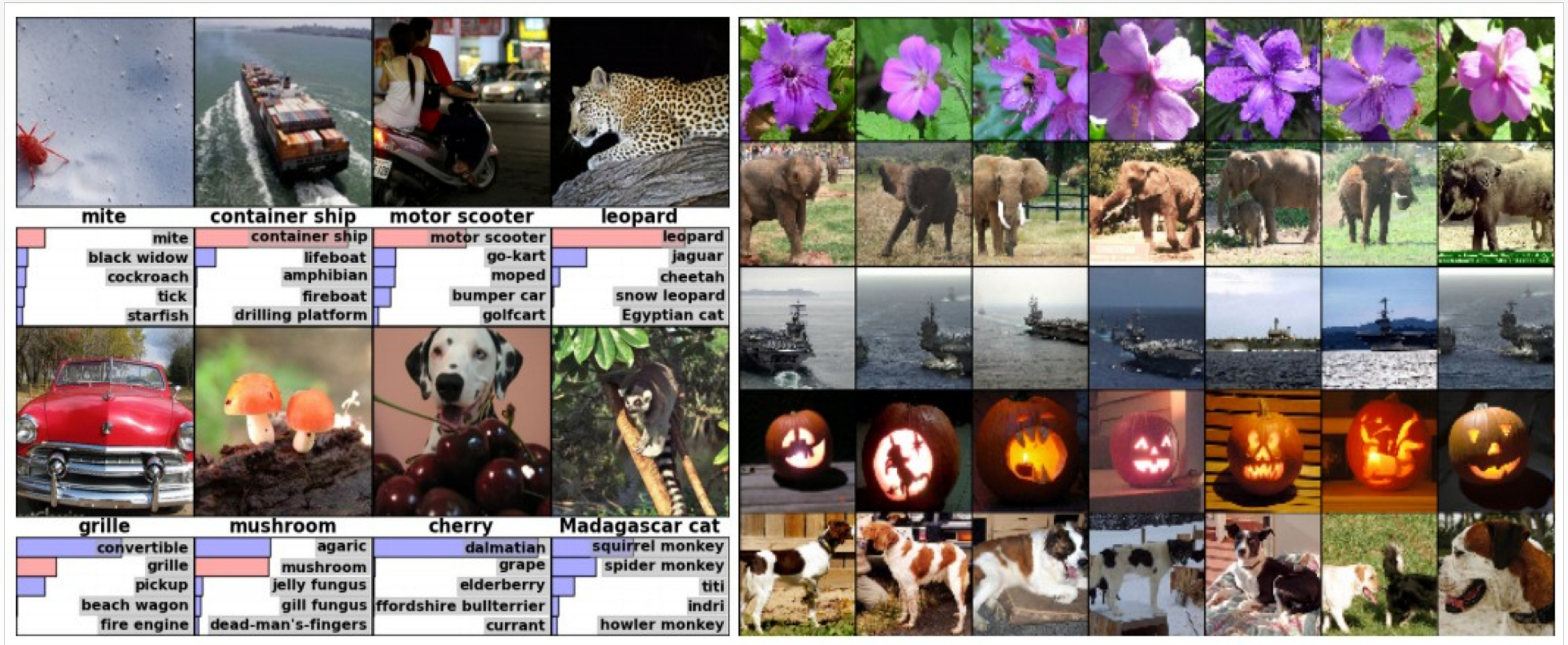
[Krizhevsky, Sutskever, Hinton, 2012]



CONVNETS ARE EVERYWHERE

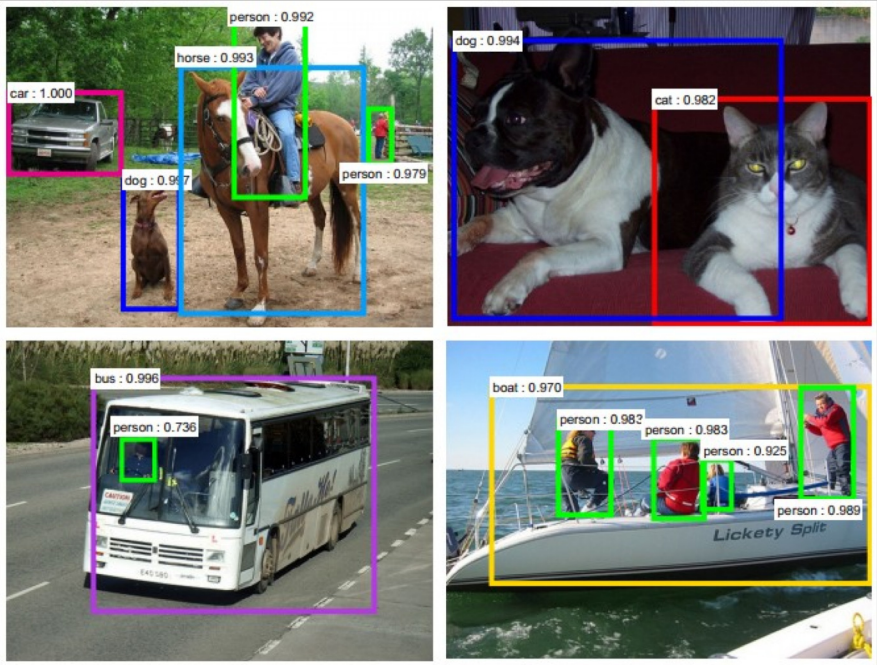
Classification

Retrieval

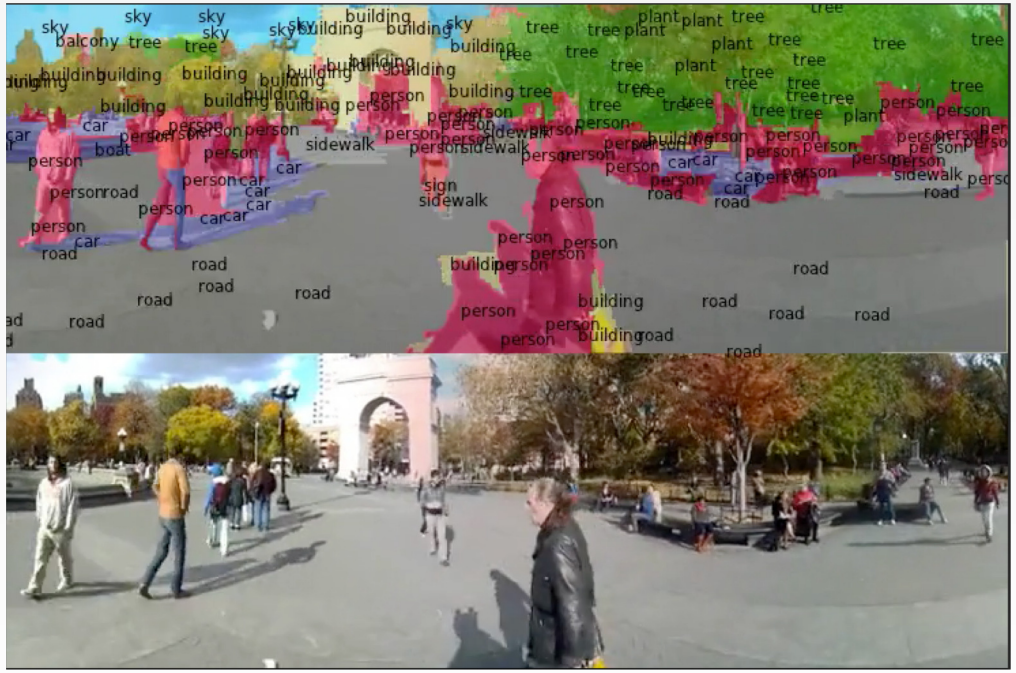


CONVNETS ARE EVERYWHERE

Detection



Segmentation



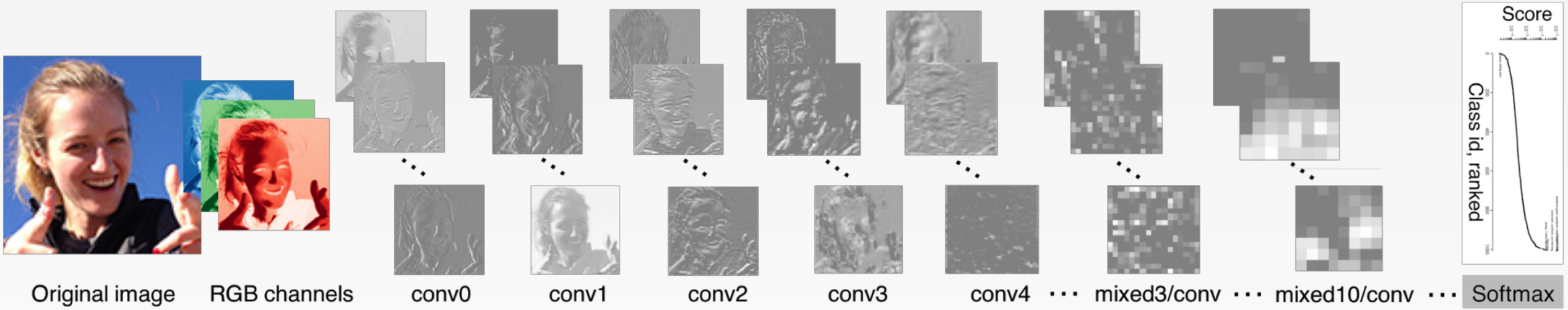
CONVNETS ARE EVERYWHERE

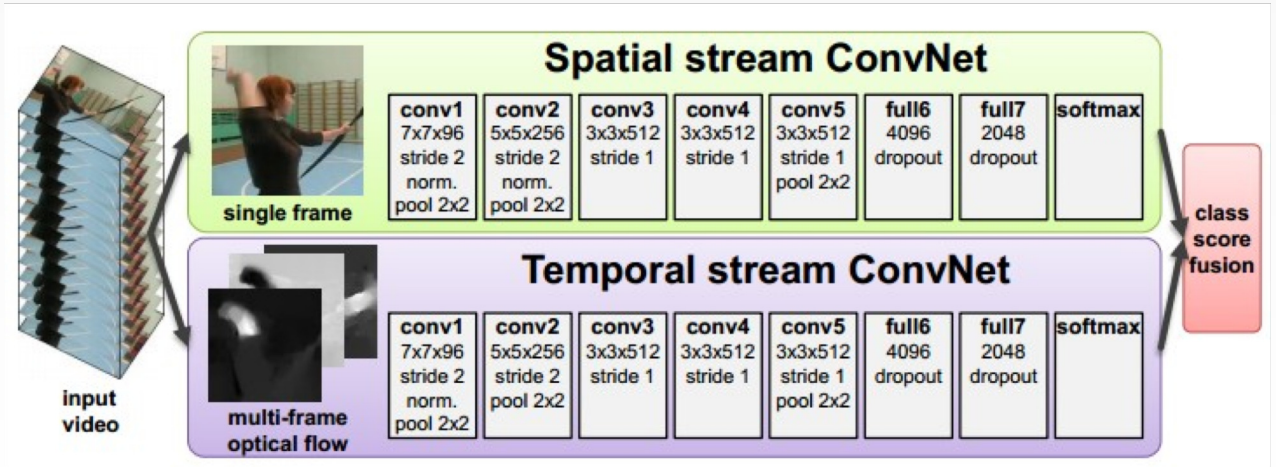
Self-driving cars

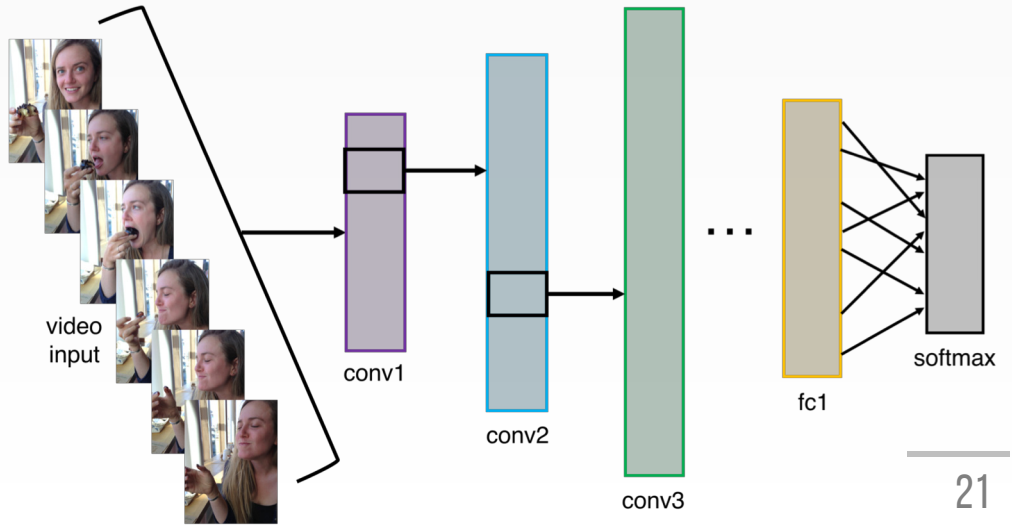
NVIDIA Tesla line

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

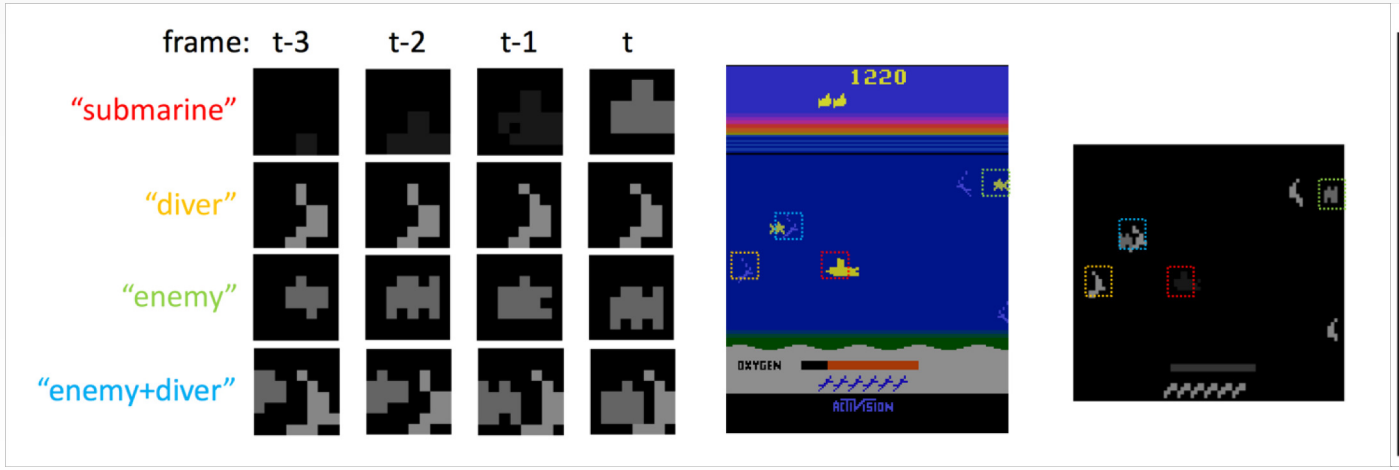
CONVNETS ARE EVERYWHERE

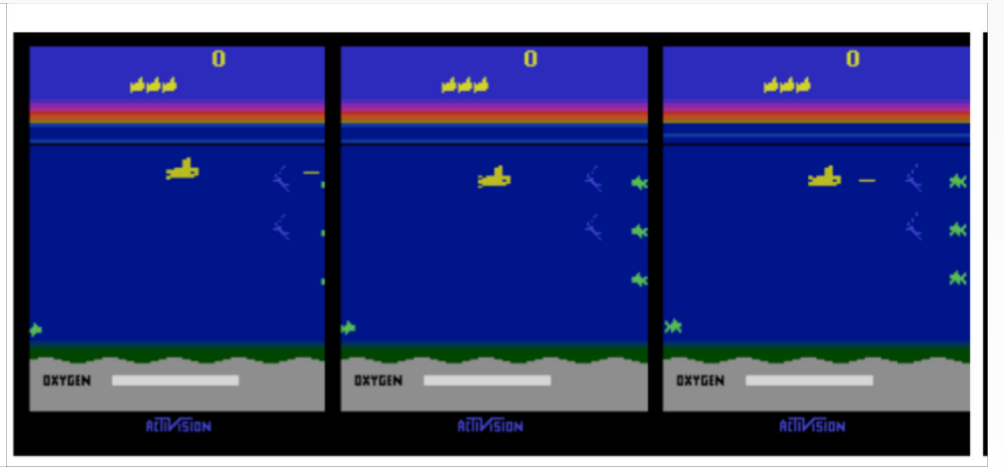




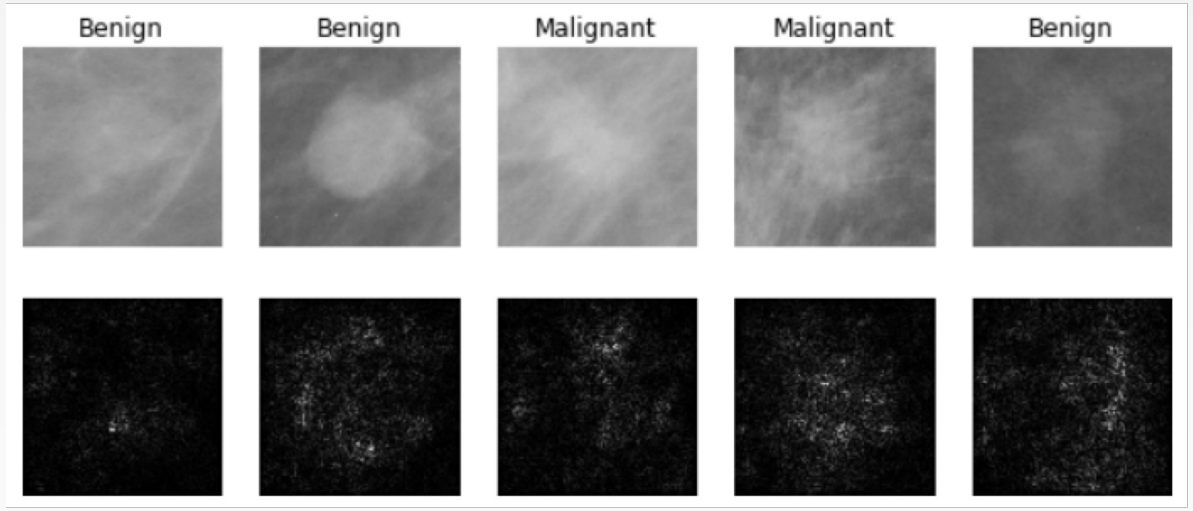


CONVNETS ARE EVERYWHERE





CONVNETS ARE EVERYWHERE



CONVNETS ARE EVERYWHERE

CONVNETS ARE EVERYWHERE

No errors

A white teddy bear sitting in the grass

Minor errors

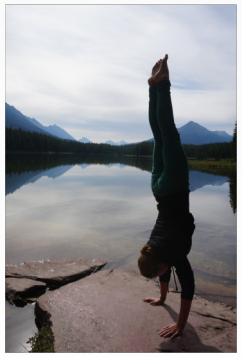
A man in a baseball uniform throwing a ball

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



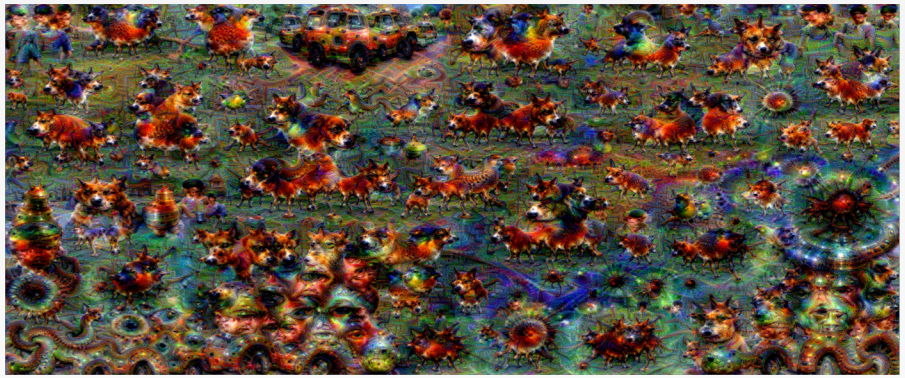
A woman standing on a beach holding a surfboard

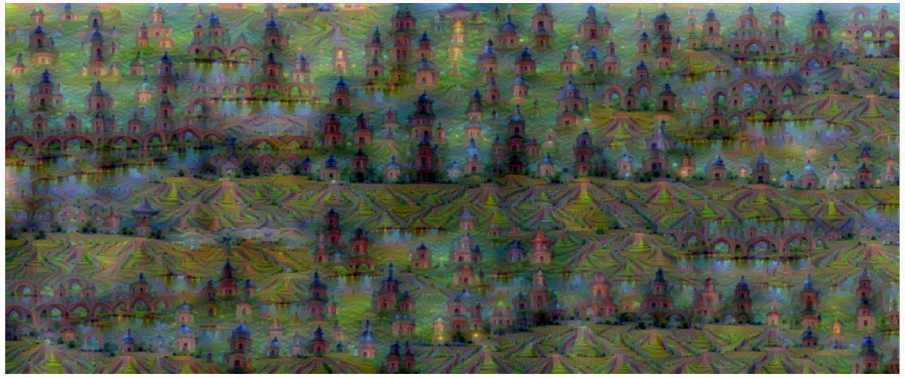
Image Captioning

[Vinyals et al., 2015]

[Karpathy and Fei-Fei, 2015]

CONVNETS ARE EVERYWHERE

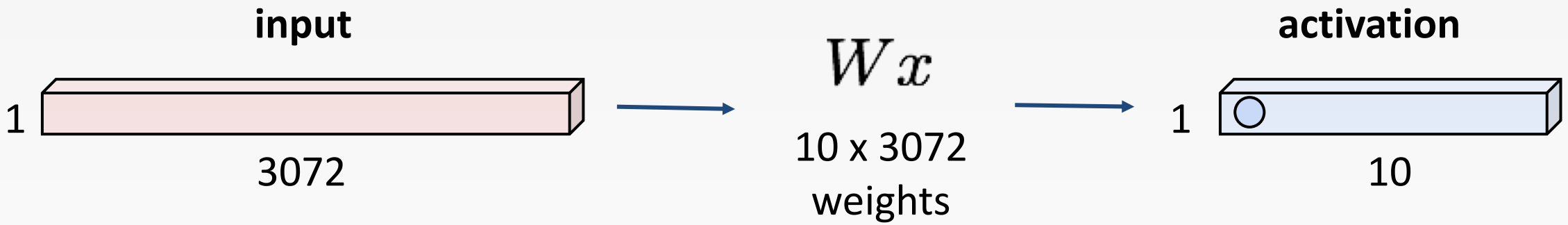


CONVOLUTIONAL LAYER

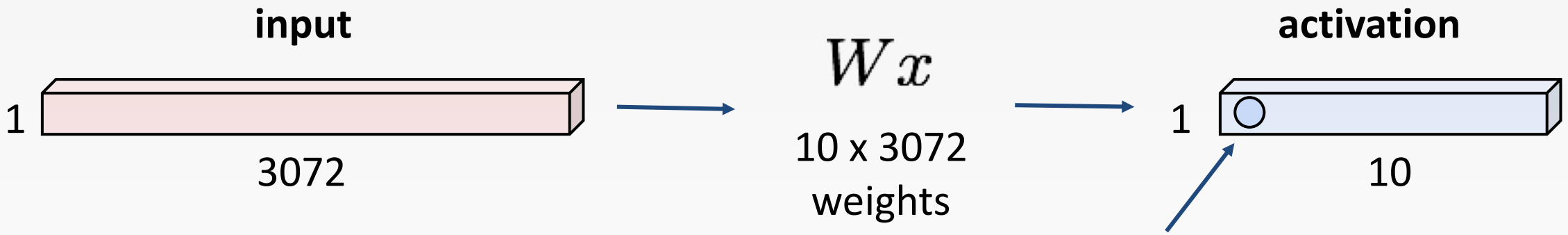
FULLY CONNECTED LAYER

32x32x3 image -> stretch to 3072 x 1



FULLY CONNECTED LAYER

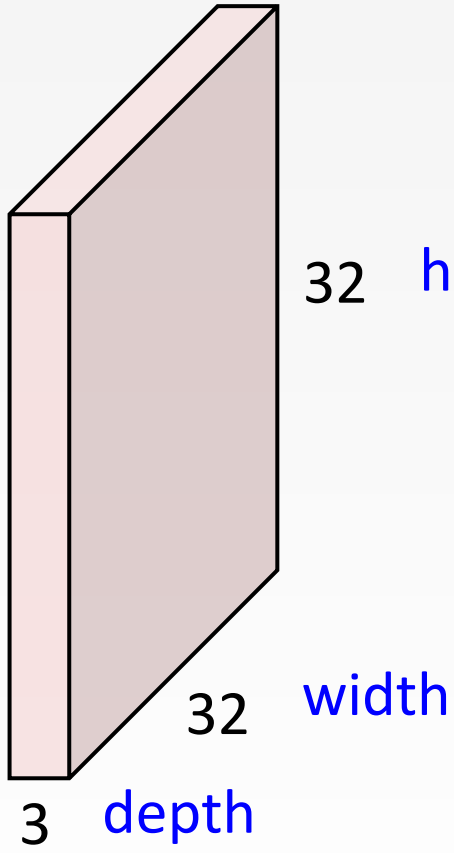
32x32x3 image -> stretch to 3072 x 1



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

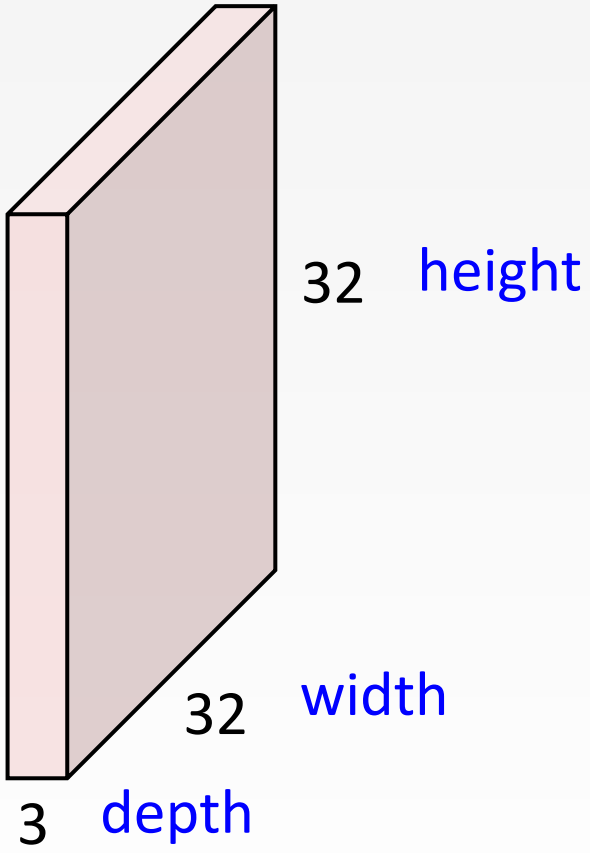
CONVOLUTIONAL LAYER

32x32x3 image -> preserve spatial structure

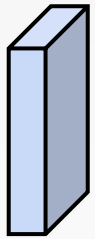


CONVOLUTIONAL LAYER

32x32x3 image -> preserve spatial structure



5x5x3 filter

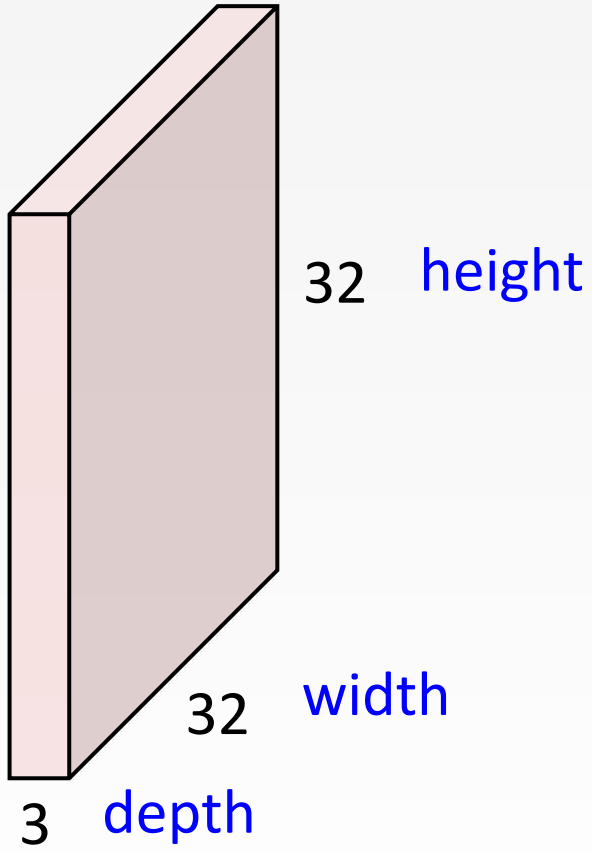


Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

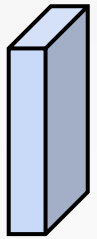
CONVOLUTIONAL LAYER

Filters always extend the full depth of the input volume

32x32x3 image -> preserve spatial structure

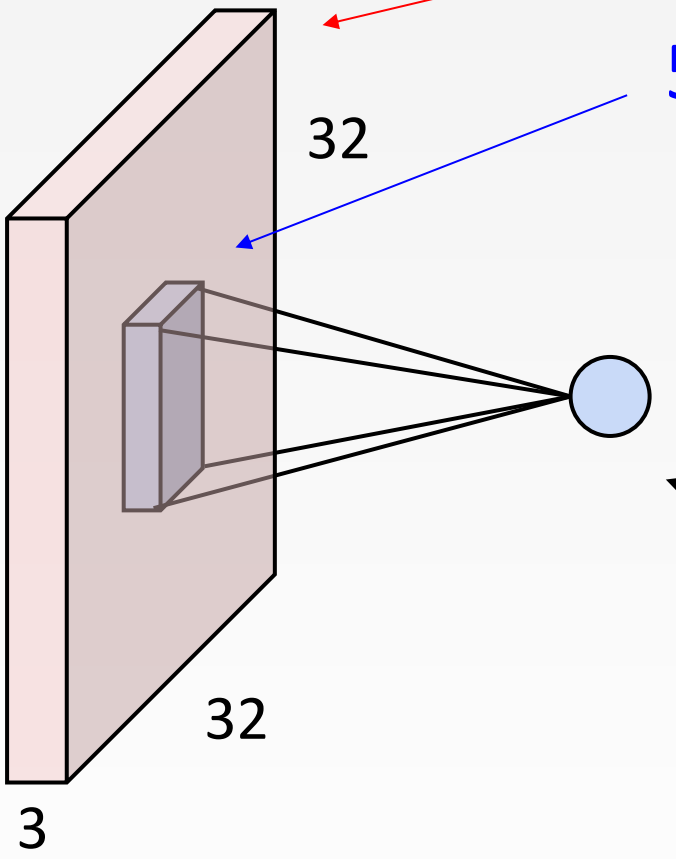


5x5x3 filter



Convolve the filter with the image
i.e. “slide over the image spatially, computing dot products”

CONVOLUTIONAL LAYER



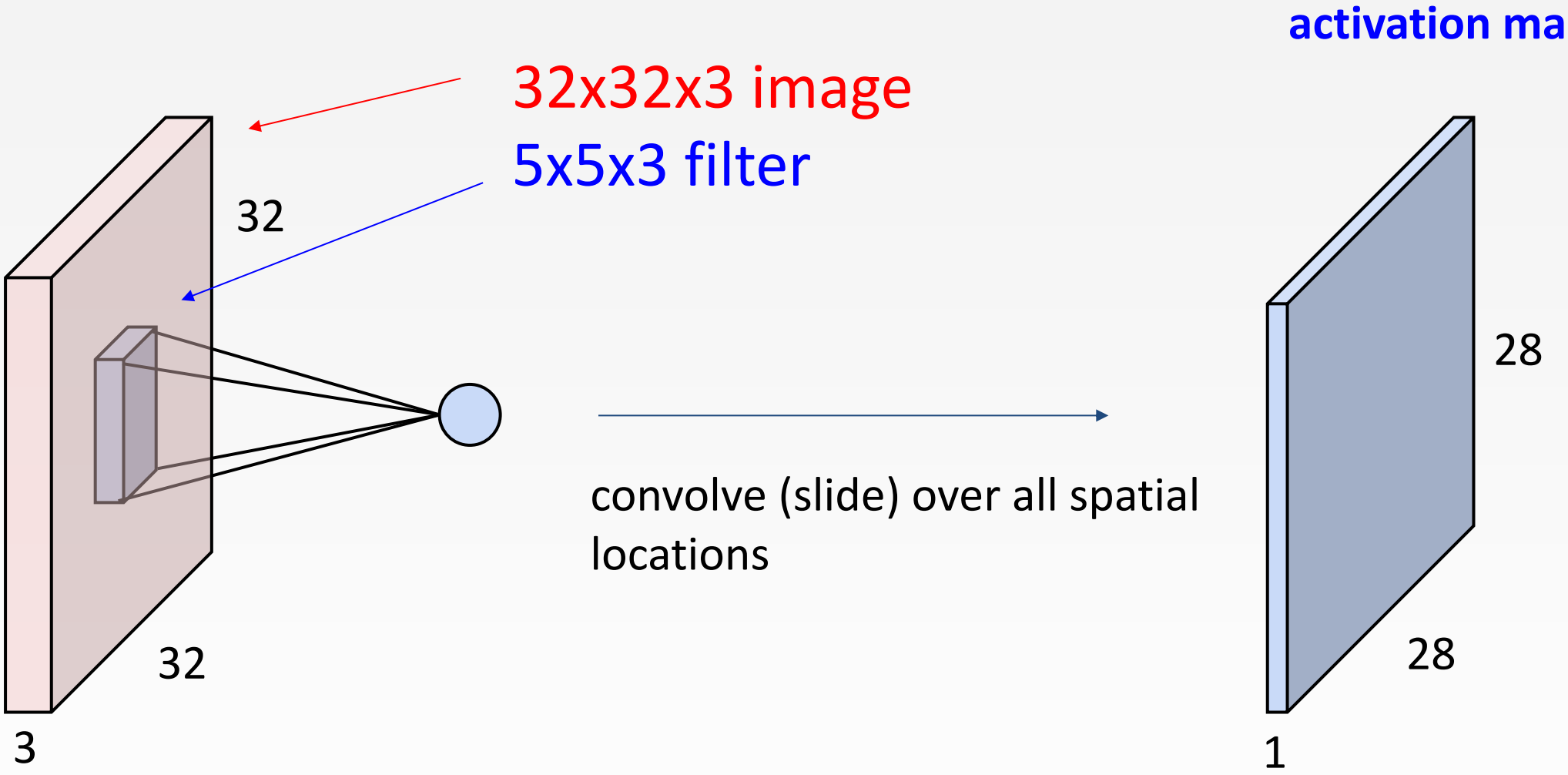
32x32x3 image

5x5x3 filter w

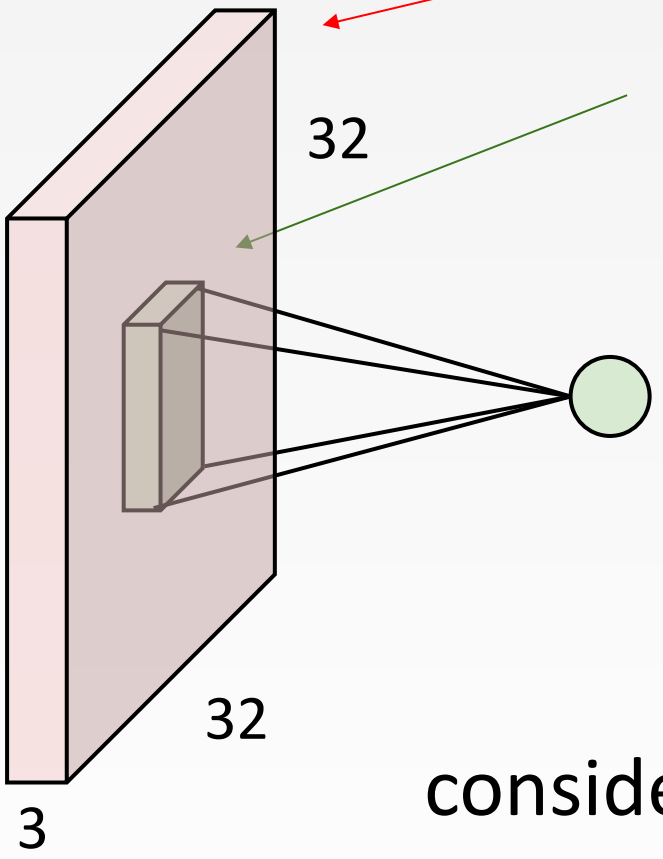
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

CONVOLUTIONAL LAYER



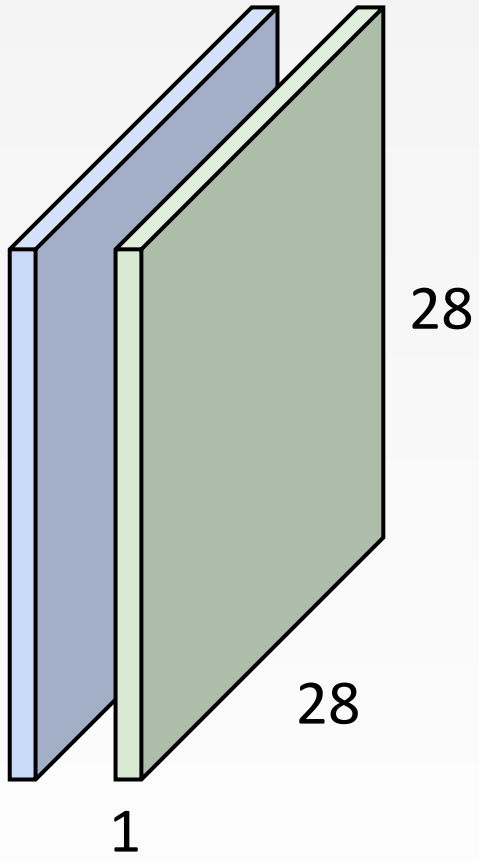
CONVOLUTIONAL LAYER



32x32x3 image

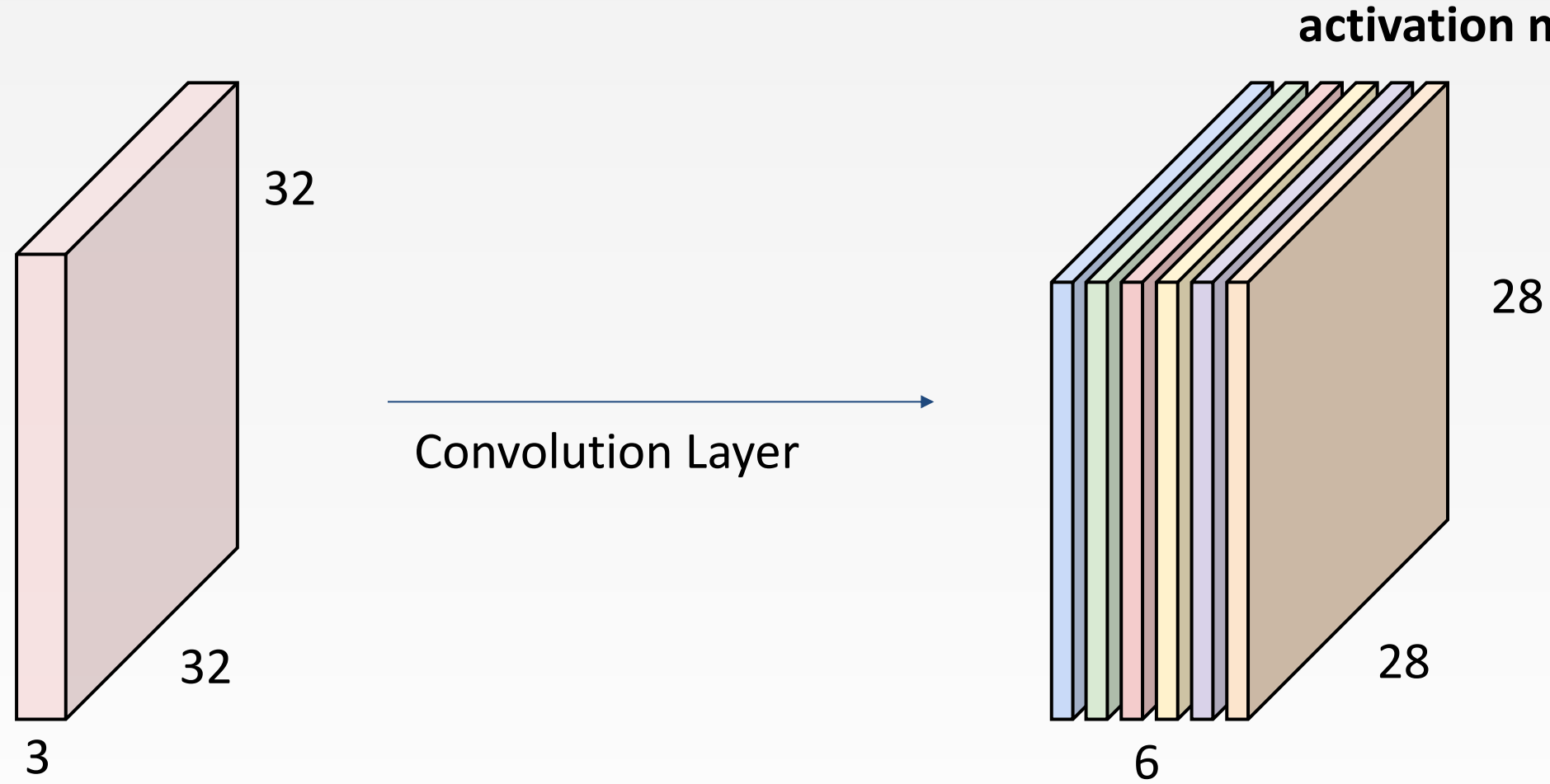
5x5x3 filter

convolve (slide) over all spatial locations



consider a second, green filter

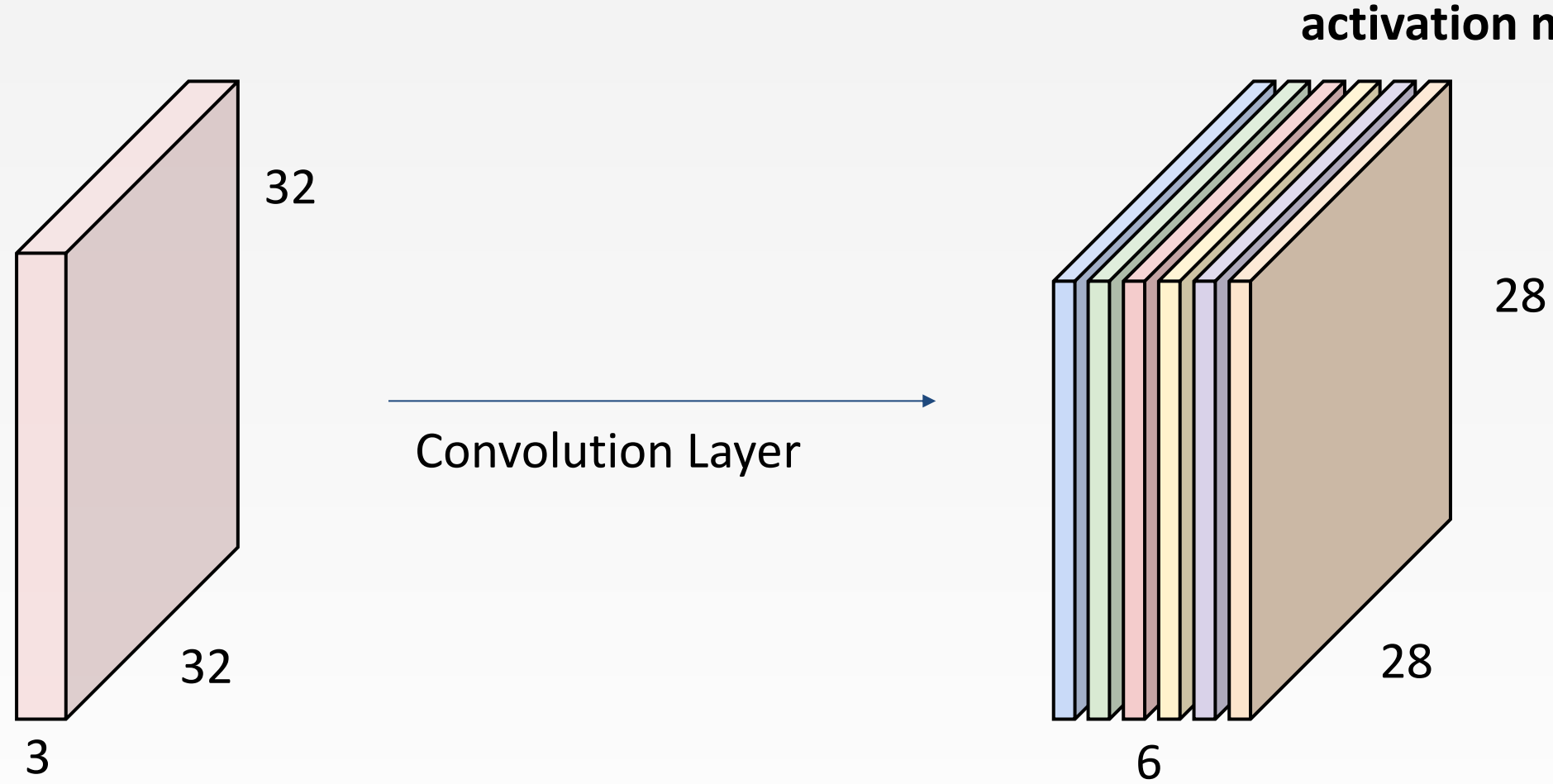
CONVOLUTIONAL LAYER



For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

We stack these up to get a "new image" of size 28x28x6!

CONVOLUTIONAL LAYER

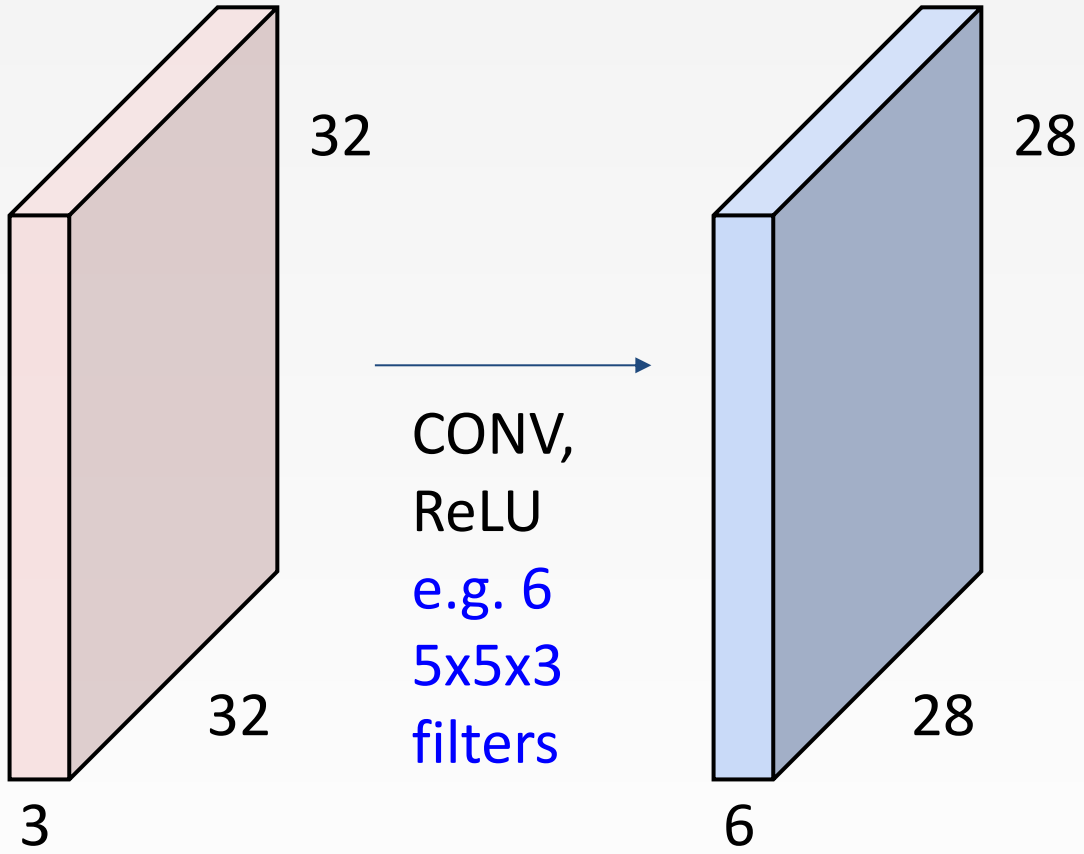


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

We stack these up to get a "new image" of size 28x28x6!

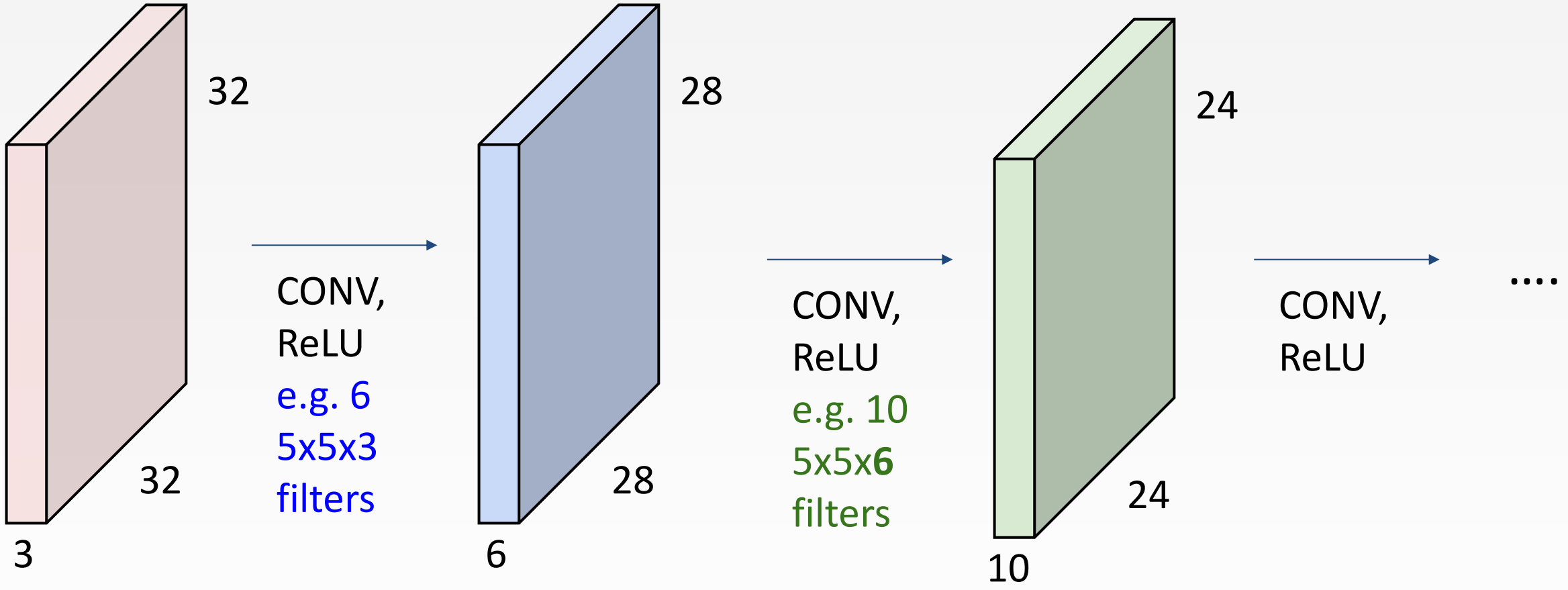
PREVIEW:

ConvNet is a sequence of Convolution Layers, interspersed with activation functions



PREVIEW:

ConvNet is a sequence of Convolution Layers, interspersed with activation functions

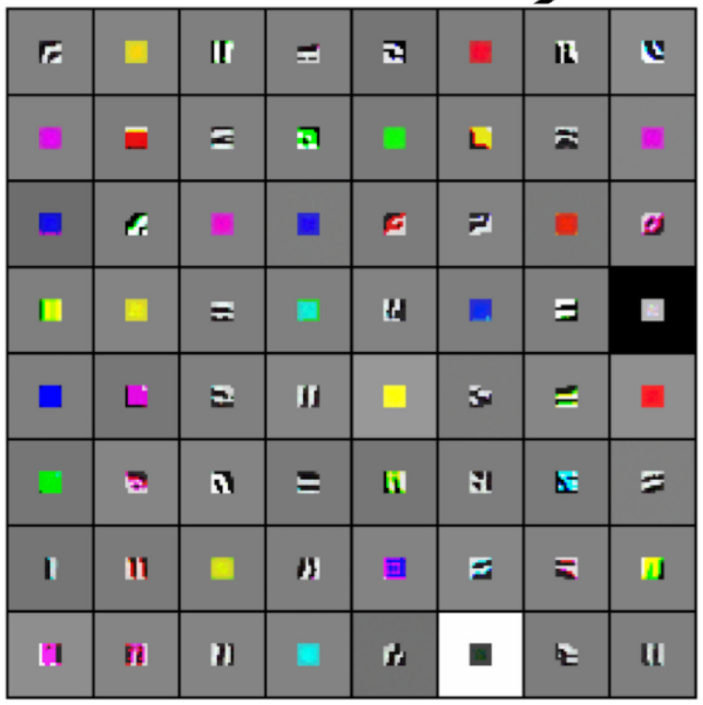


PREVIEW:

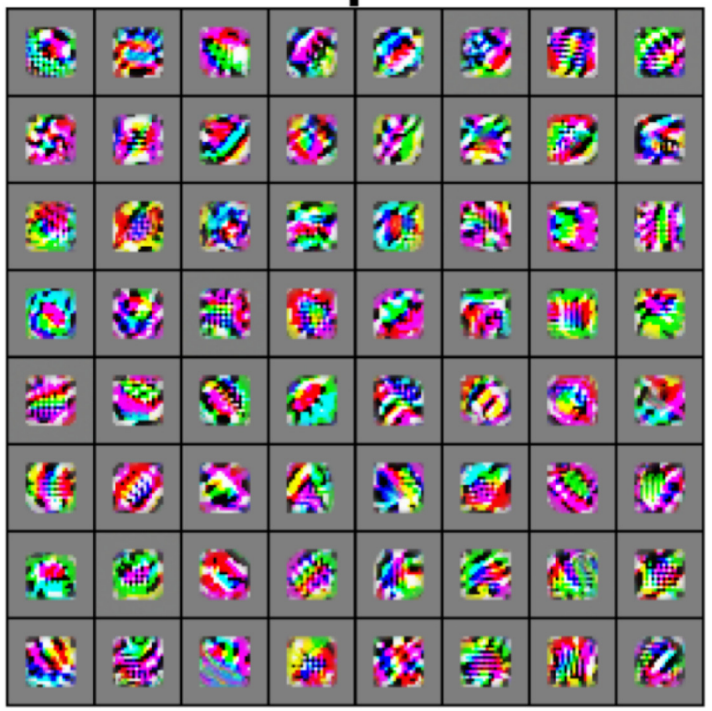
[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].

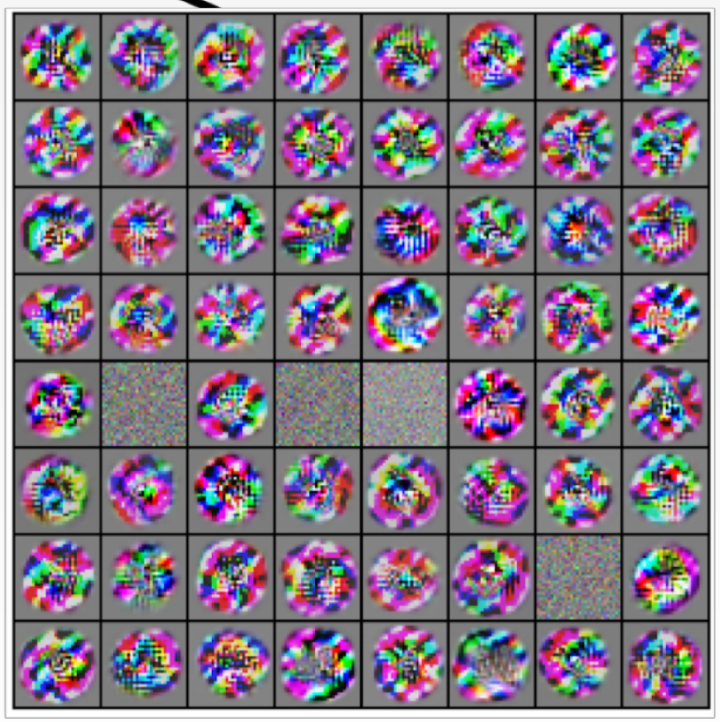
Low-level features → Mid-level features → High-level features → Linearly separable classifier



VGG-16 Conv1_1

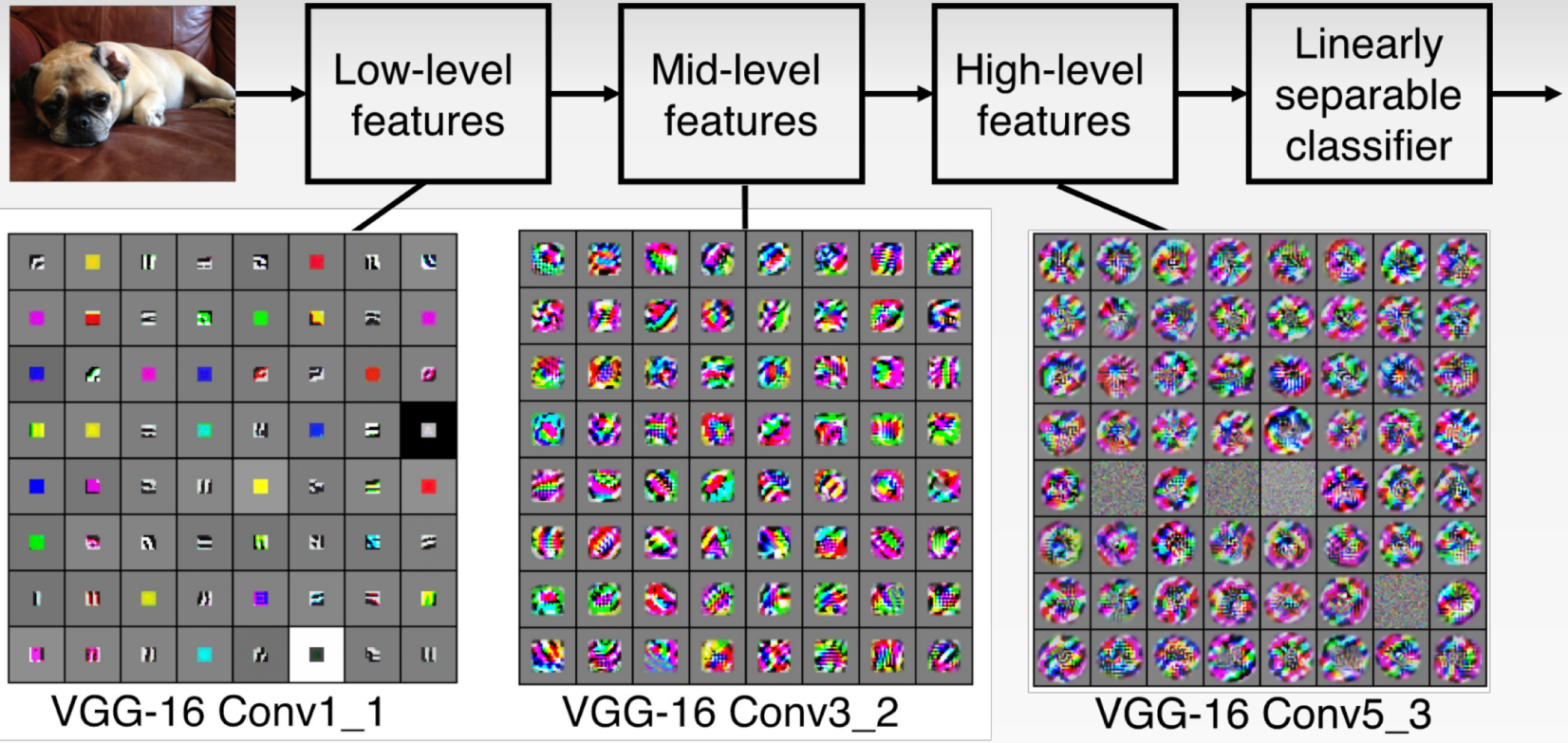


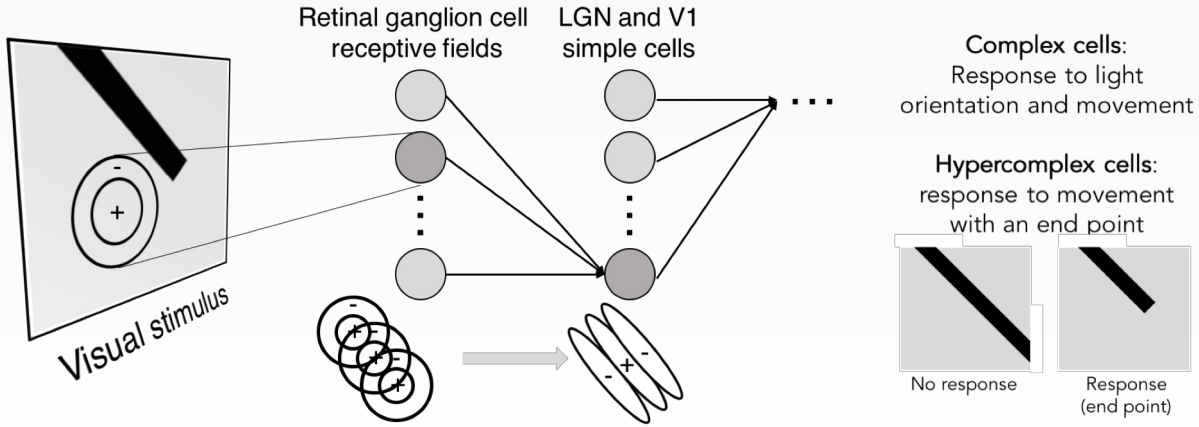
VGG-16 Conv3_2



VGG-16 Conv5_3

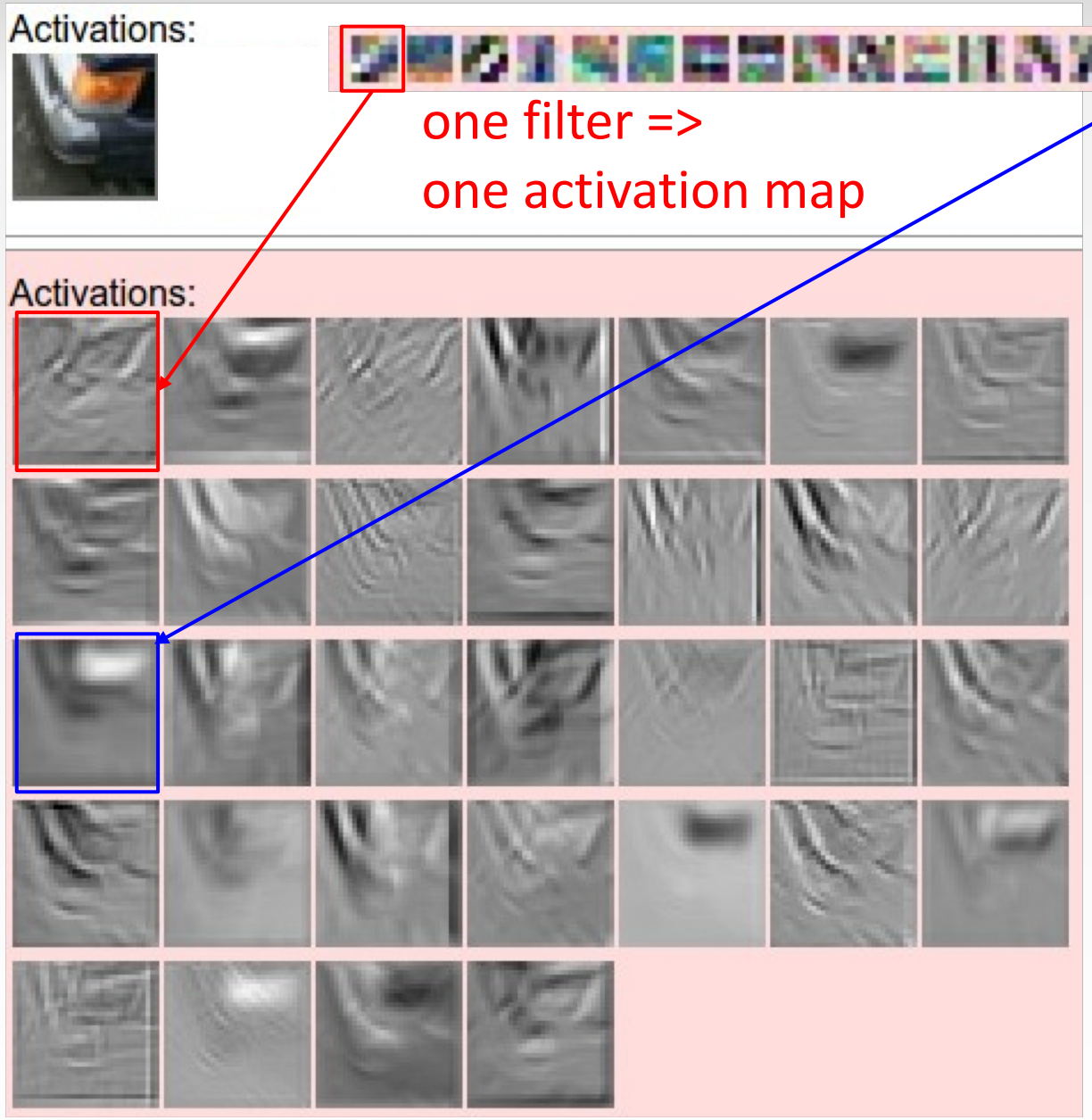
PREVIEW:





Complex cells:
Response to light orientation and movement

Hypercomplex cells:
response to movement with an end point

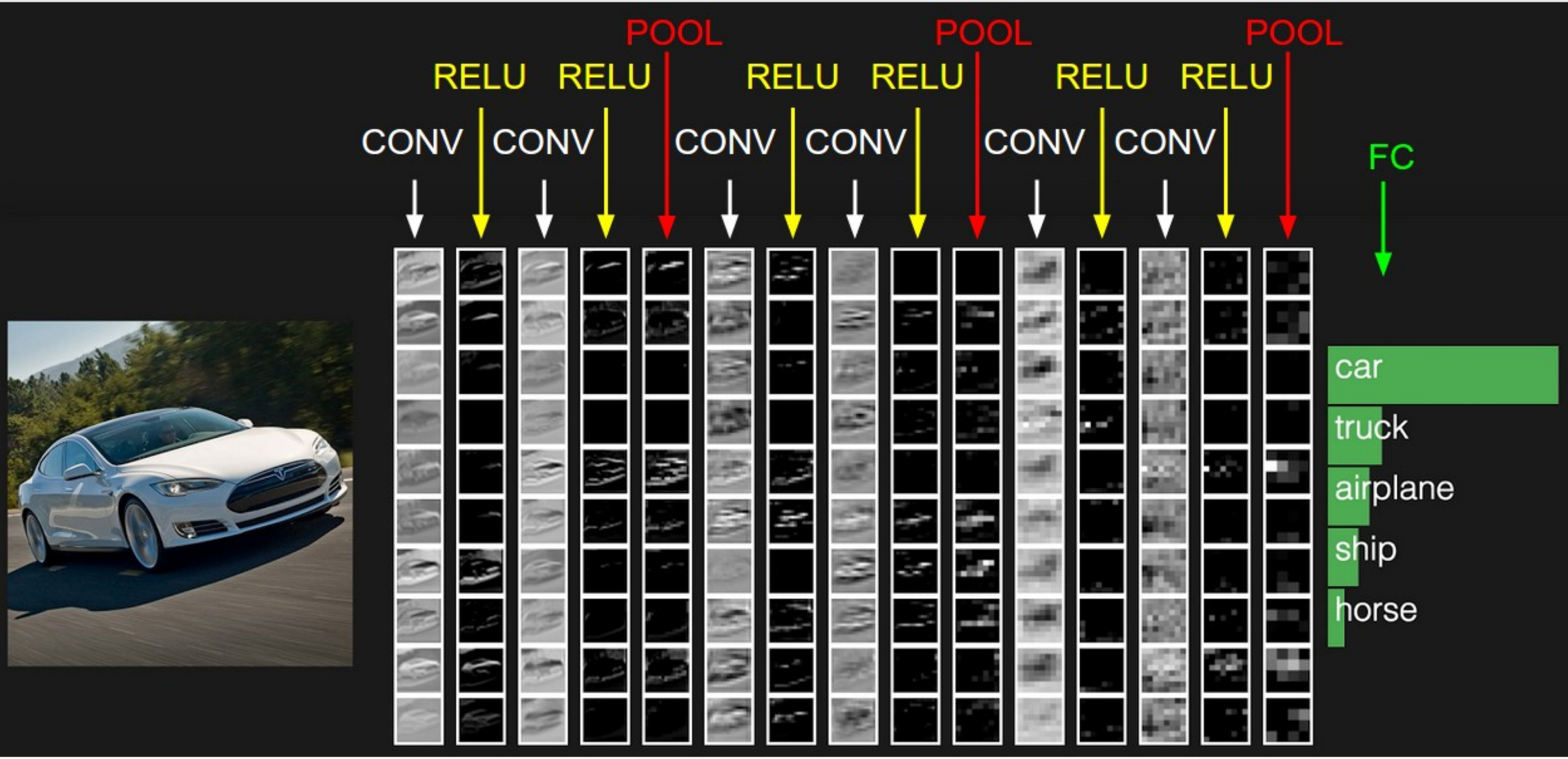


example 5x5 filters
(32 total)

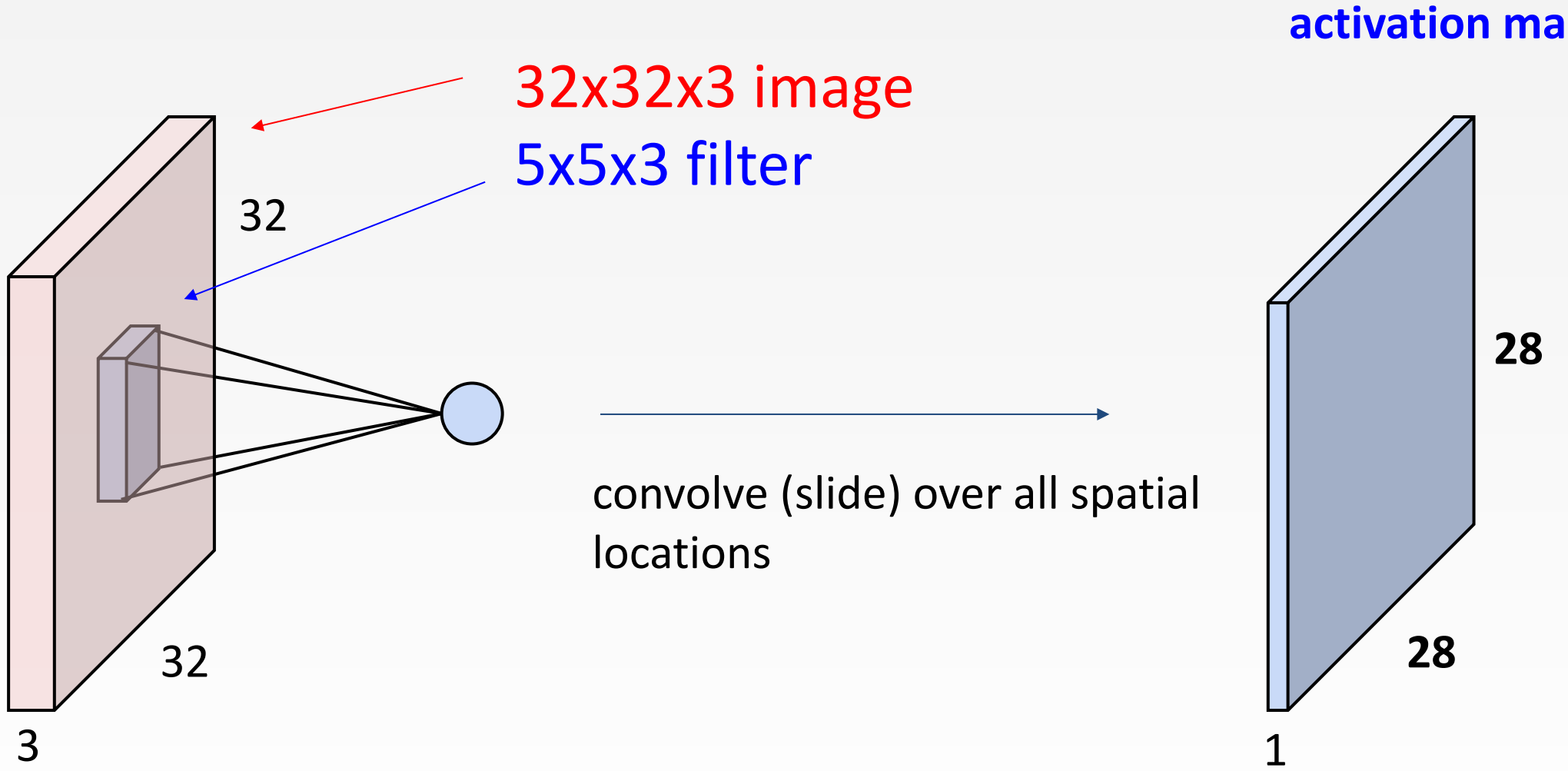
We call the layer convolutional because it is related to convolution of two signals:

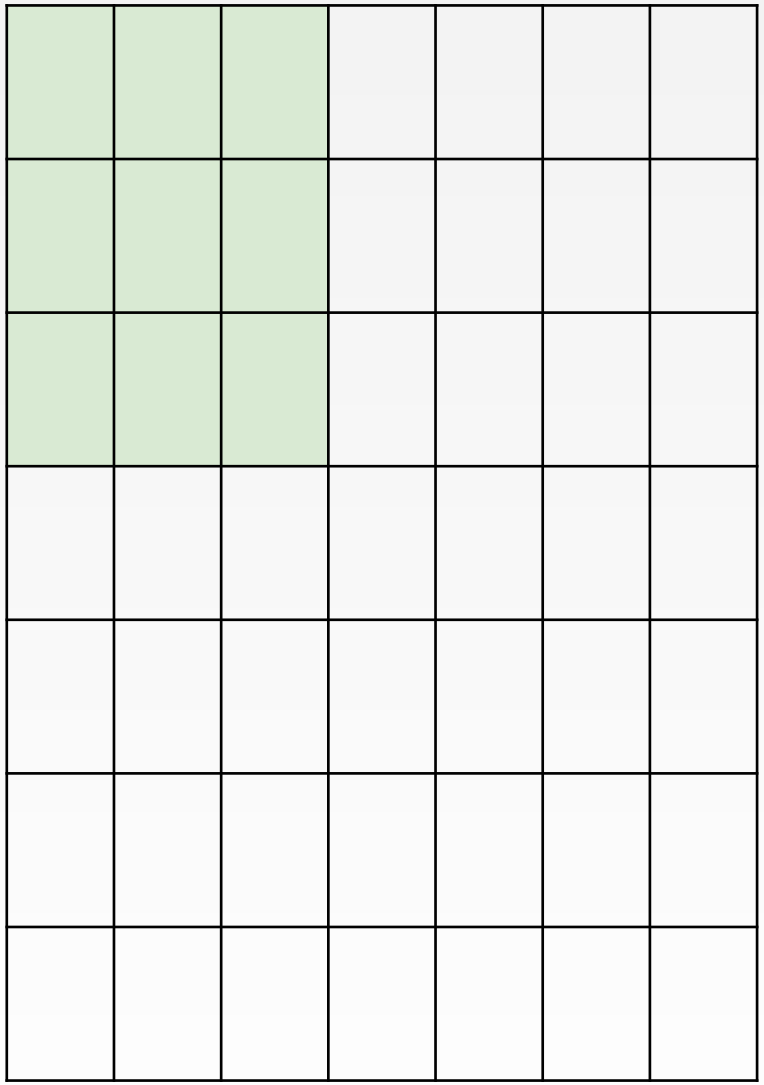
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

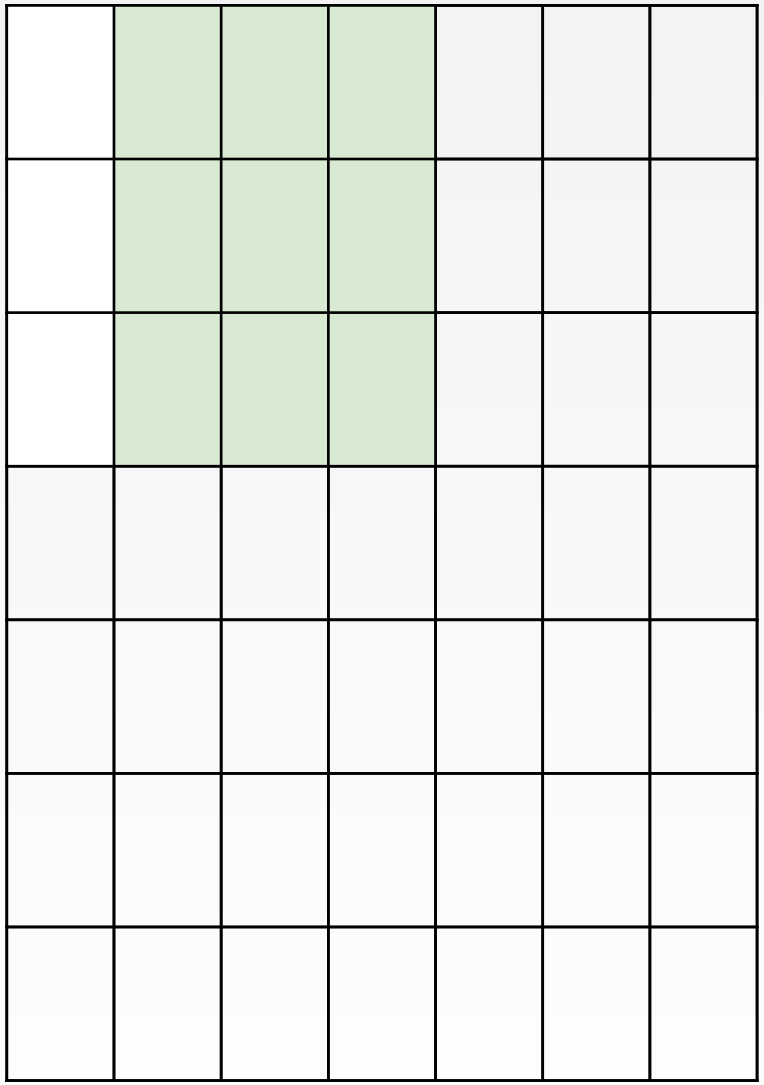

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

A CLOSER LOOK AT SPATIAL DIMENSIONS:

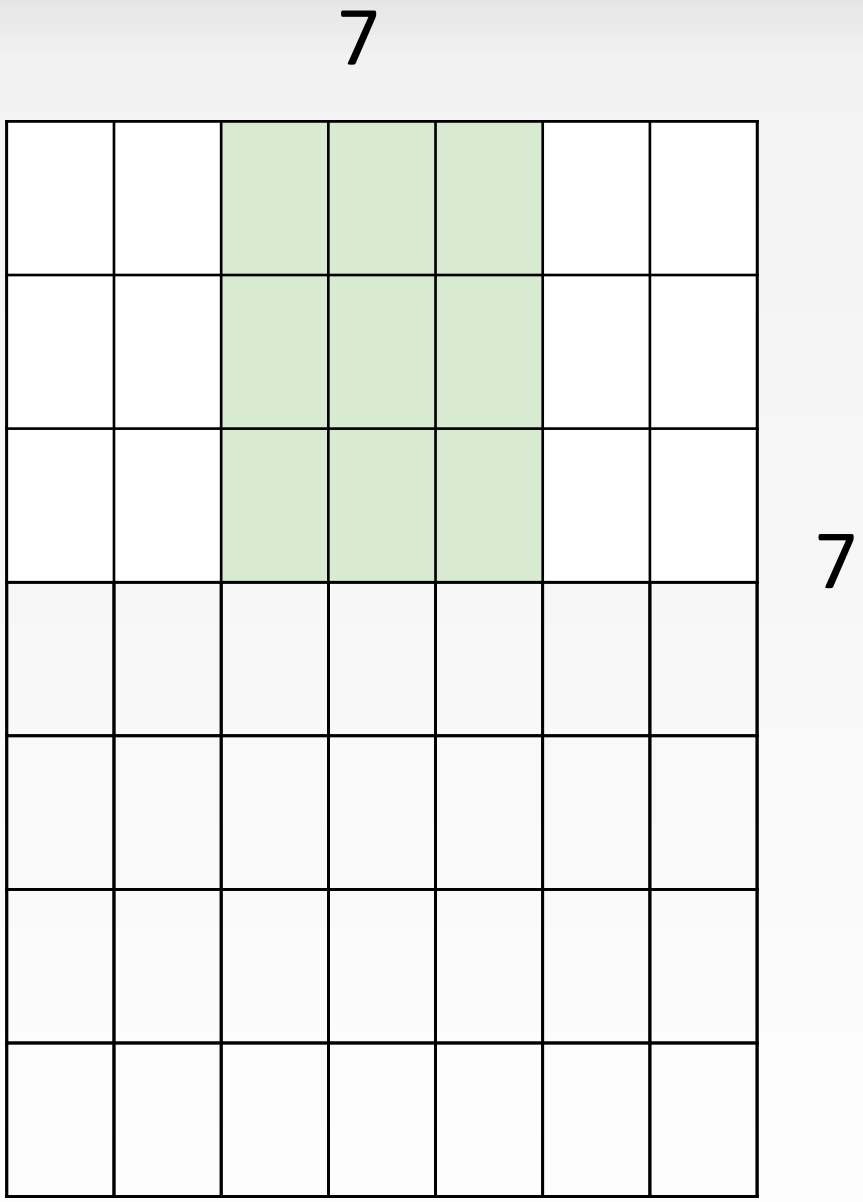




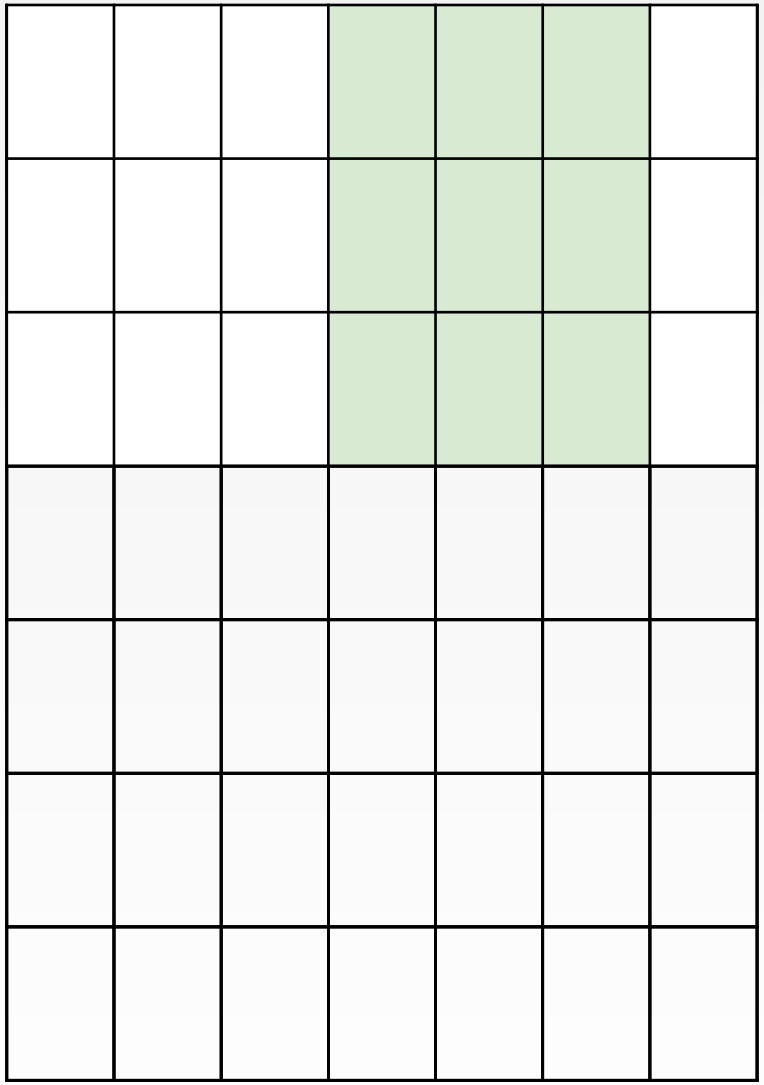
7x7 input (spatially)
assume 3x3 filter



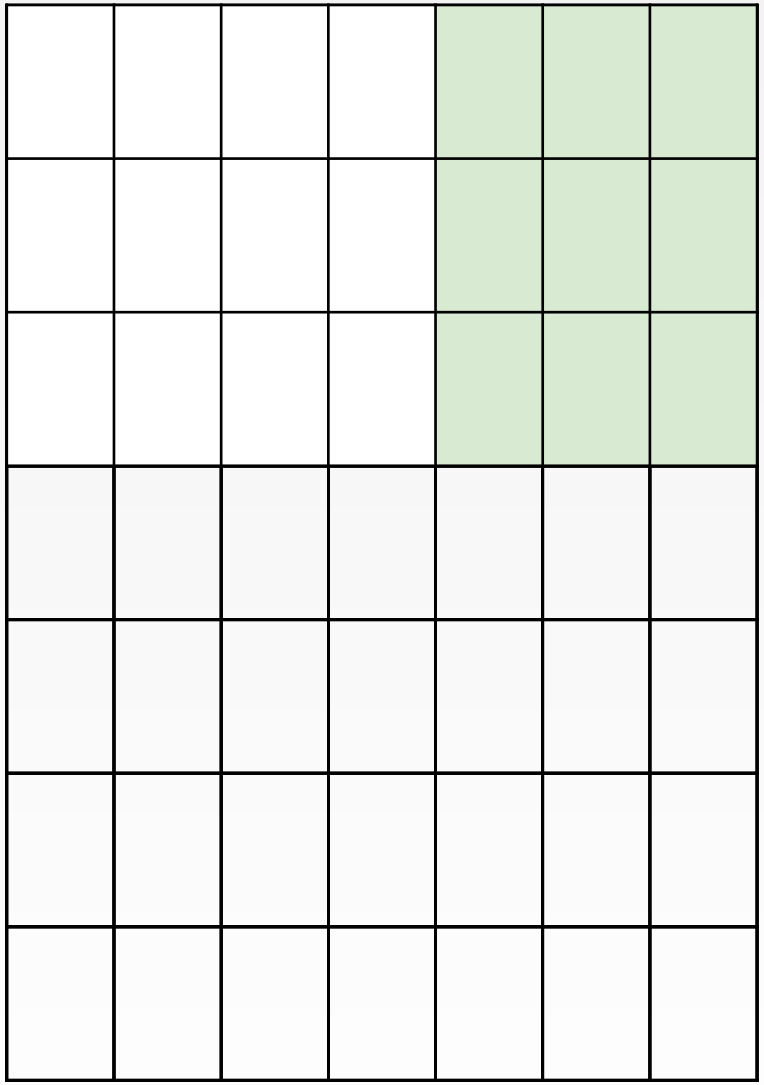
7x7 input (spatially)
assume 3x3 filter



7x7 input (spatially)
assume 3x3 filter



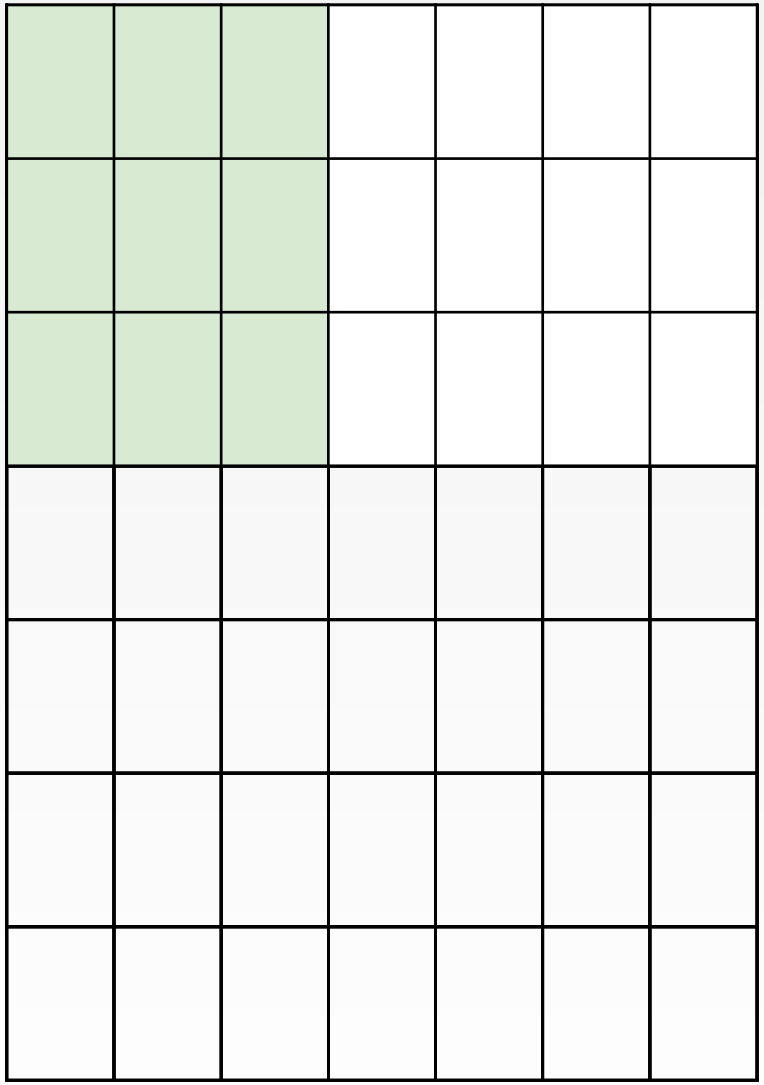
7x7 input (spatially)
assume 3x3 filter



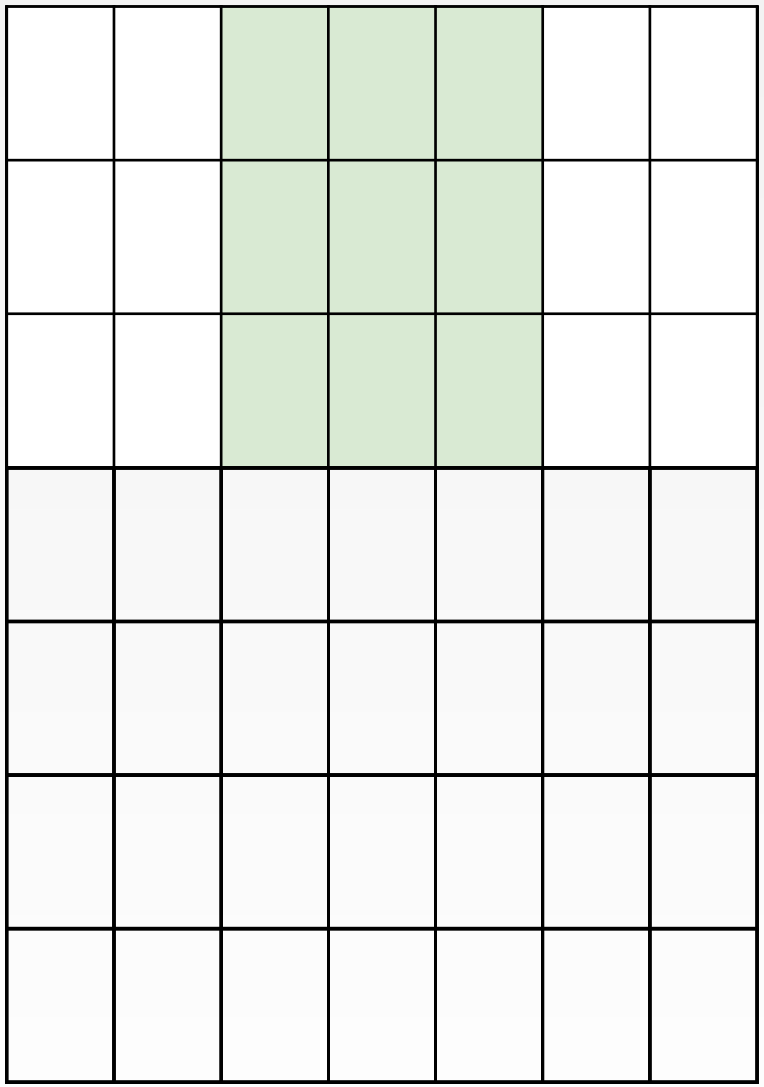
7x7 input (spatially)

assume 3x3 filter

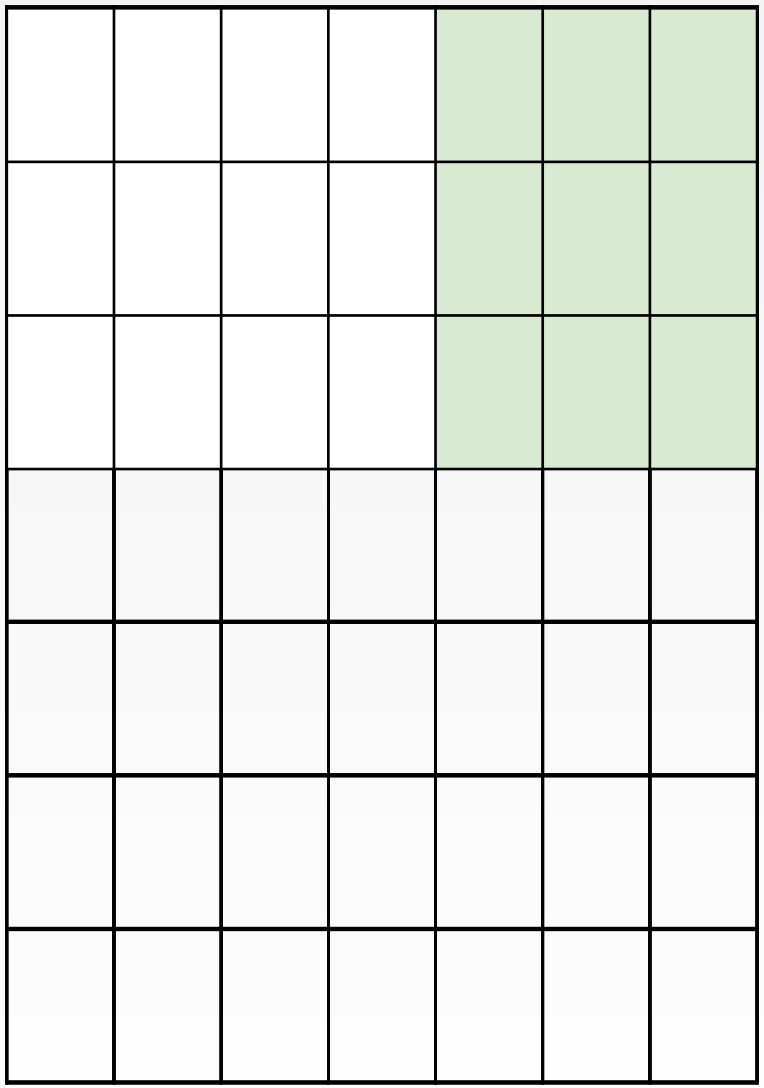
=> 5x5 output!



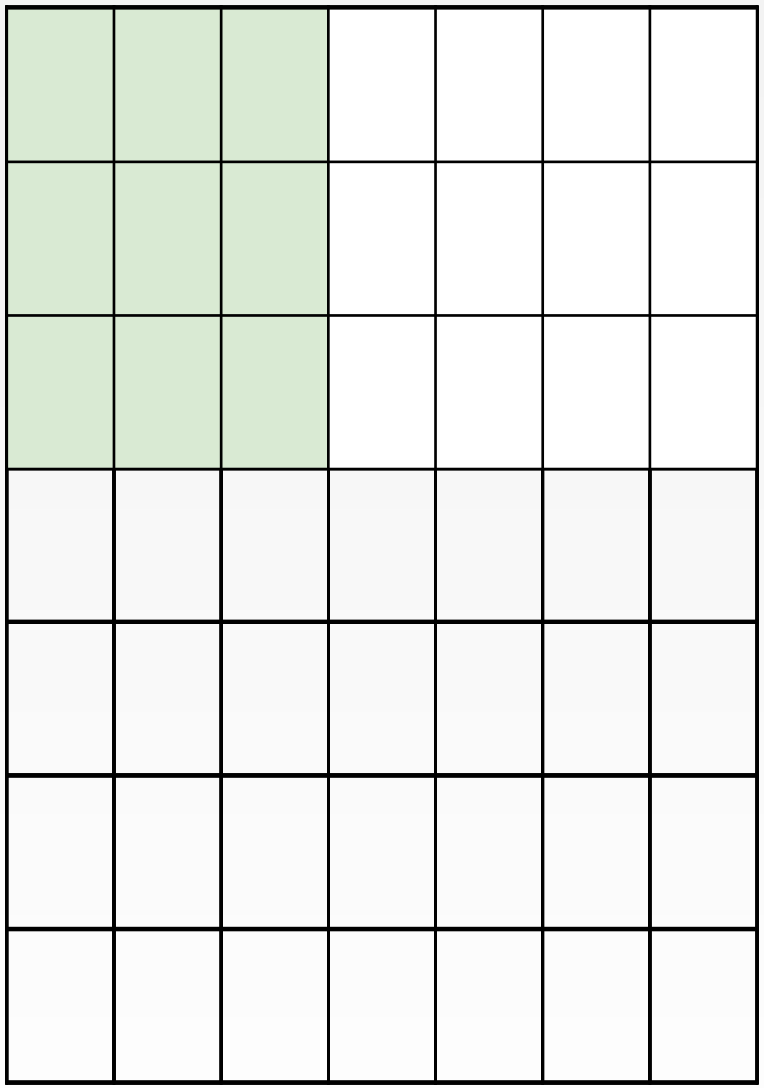
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



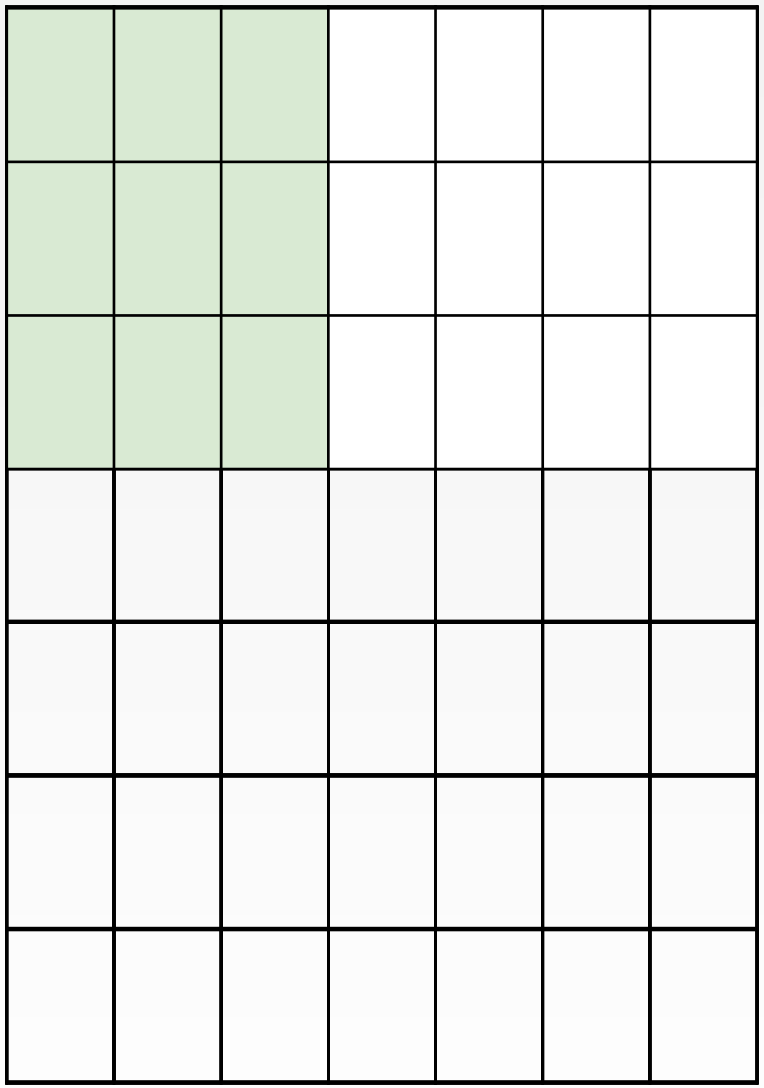
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

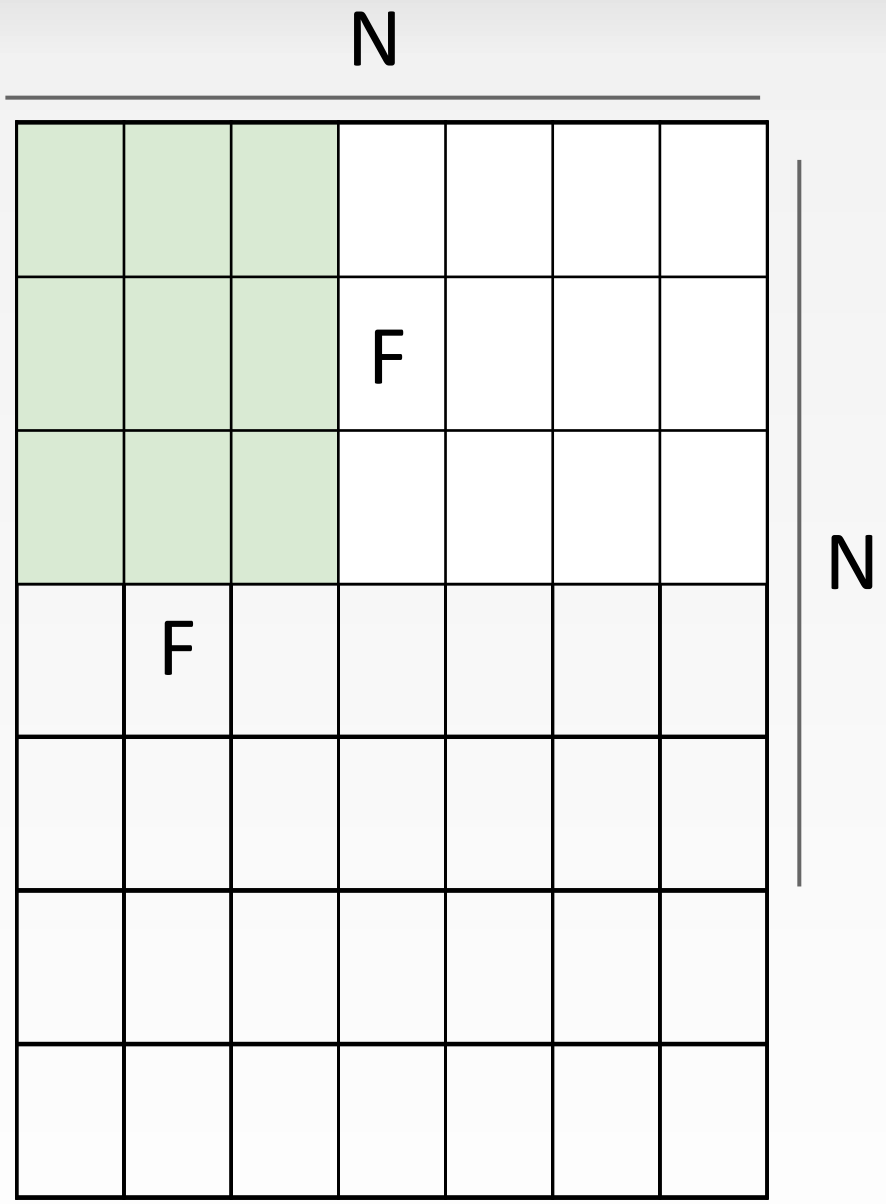


7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on 7x7 input with stride 3.



Output size:

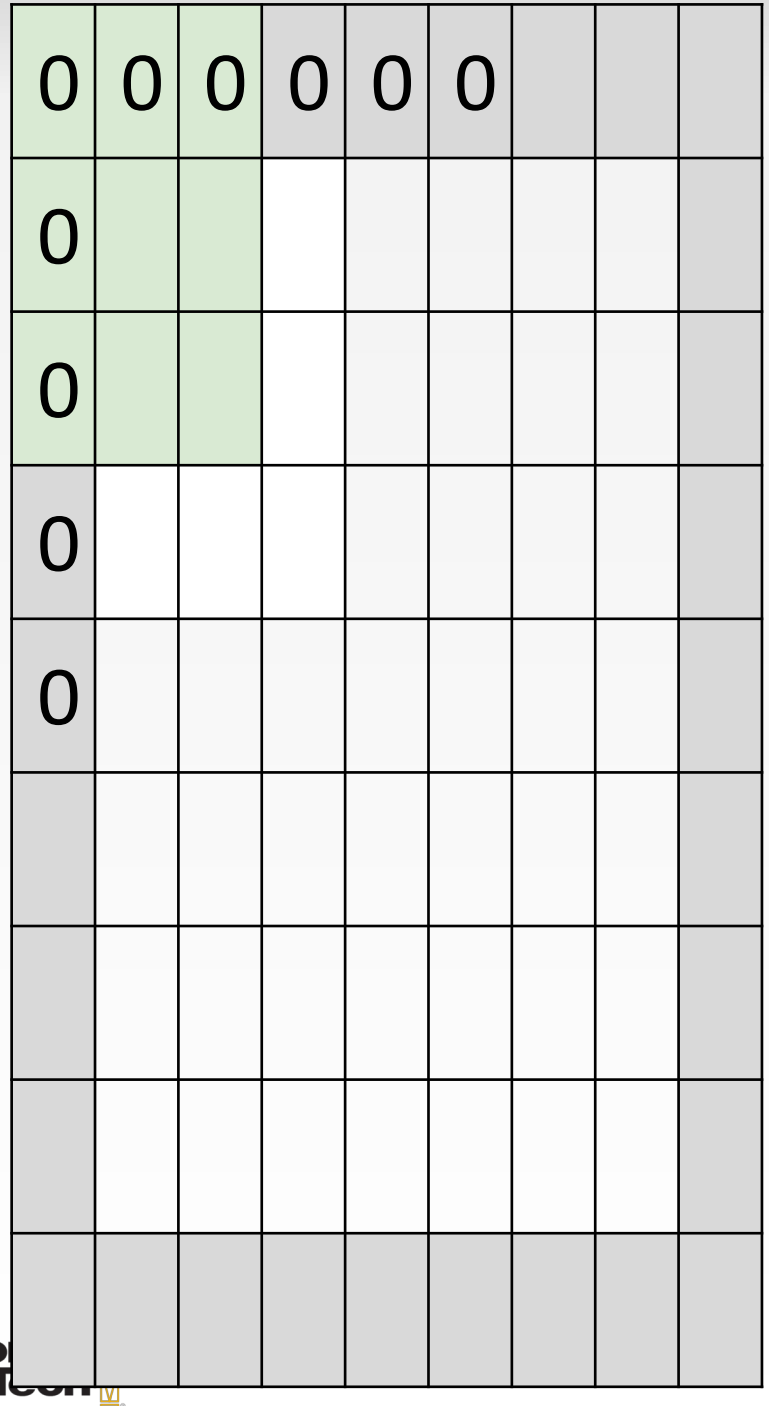
(N - F) / stride + 1

e.g. N = 7, F = 3:

stride 1 => (7 - 3)/1 + 1 = 5

stride 2 => (7 - 3)/2 + 1 = 3

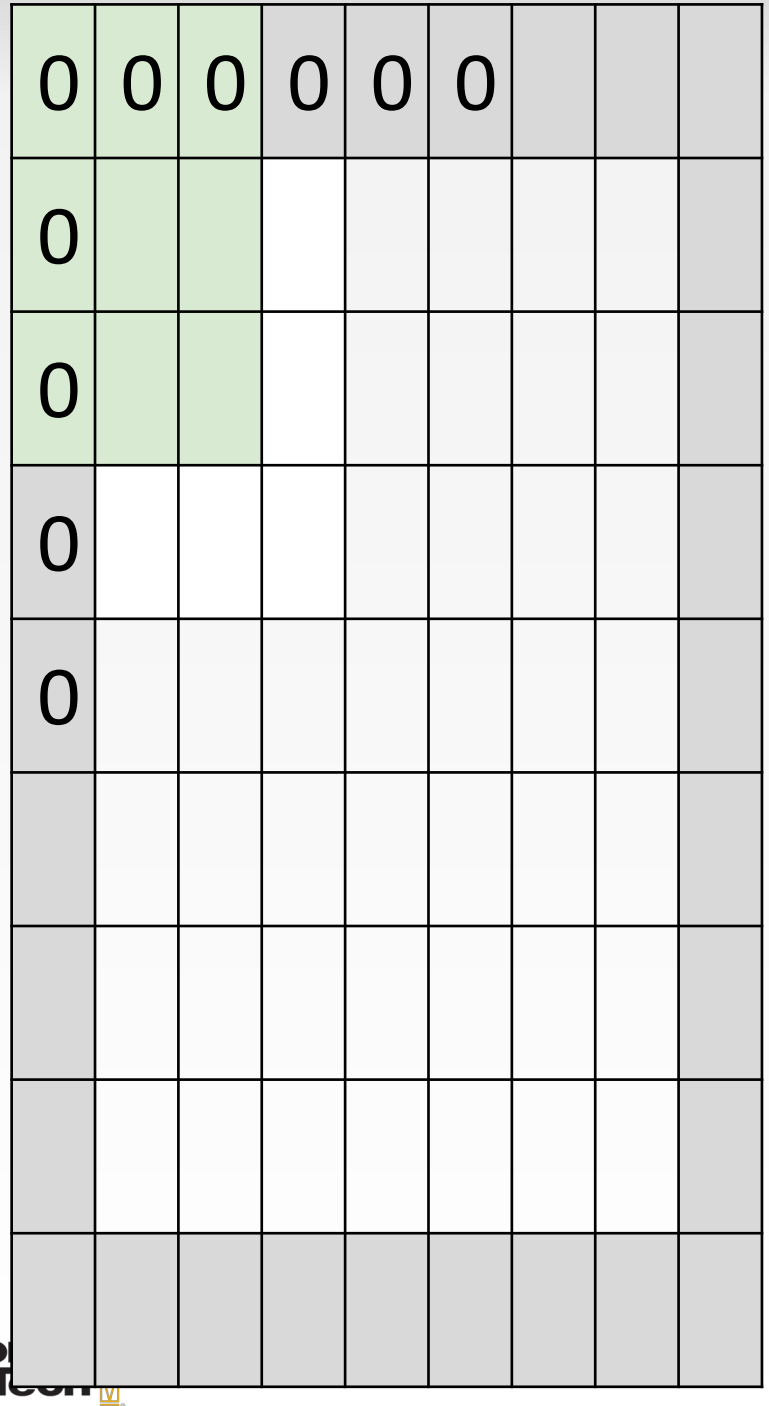
stride 3 => (7 - 3)/3 + 1 = 2.33 :\



In practice:
Common to zero pad the border

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border
=> what is the output?

(recall:)
(N - F) / stride + 1



In practice:
Common to zero pad the border

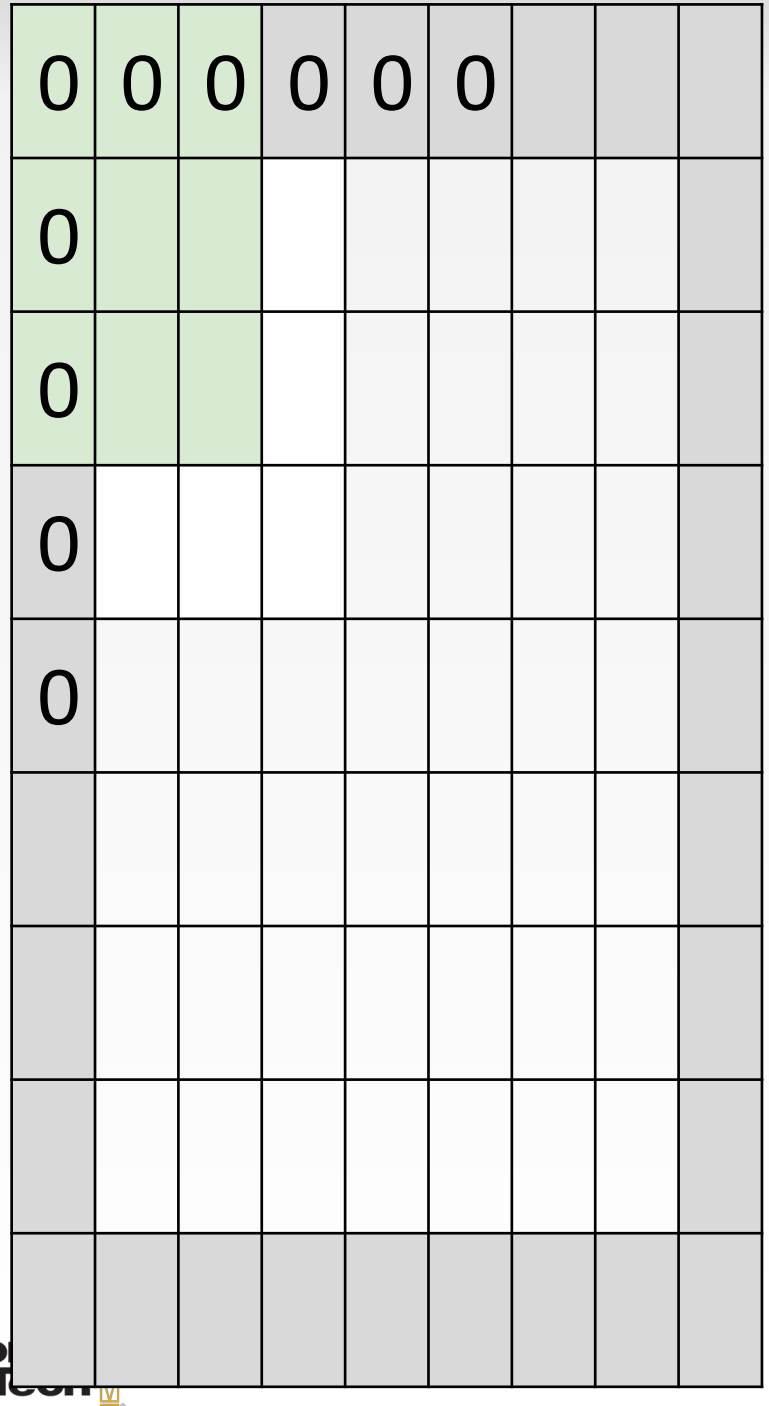
e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border

=> what is the output?

7x7 output!

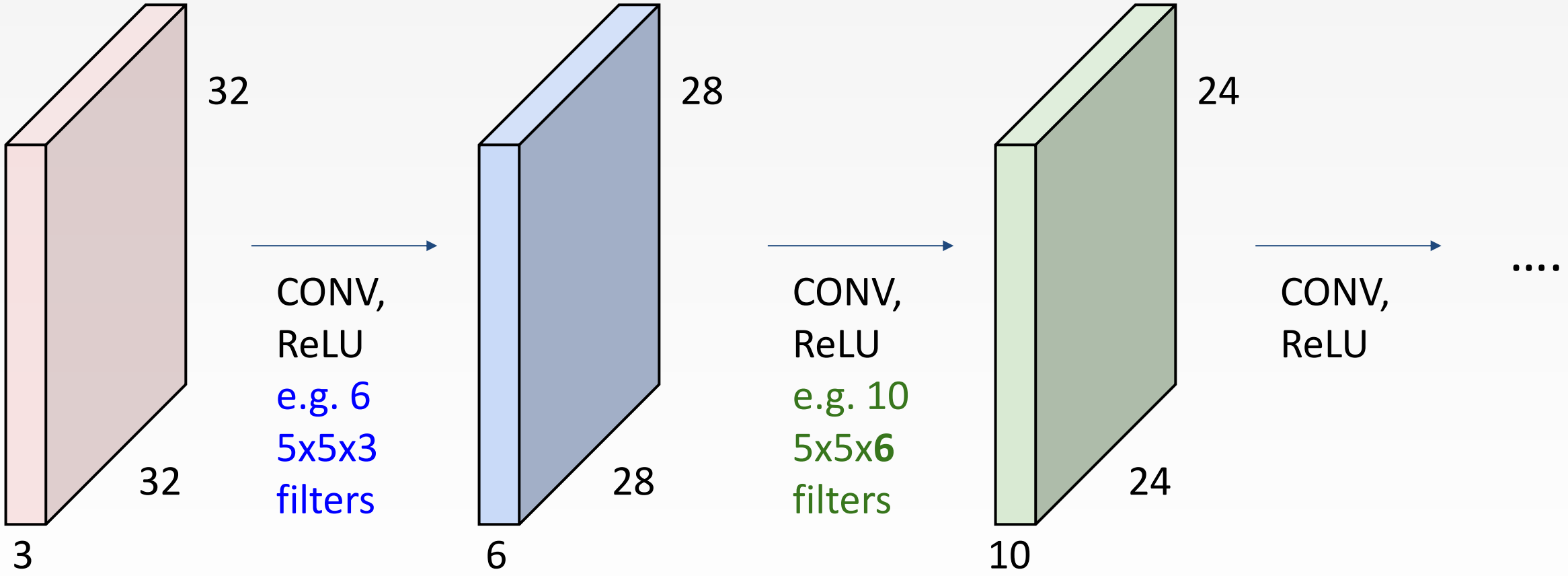


In practice:
Common to zero pad the border

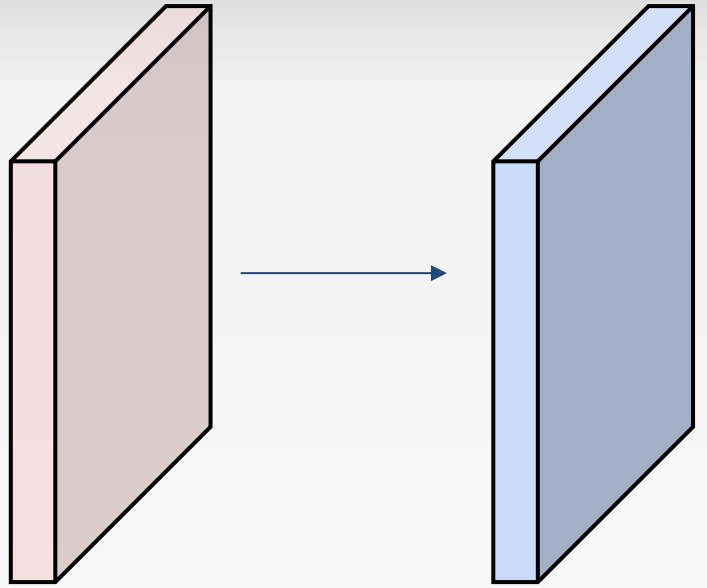
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border
=> what is the output?
7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

REMEMBER BACK TO…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



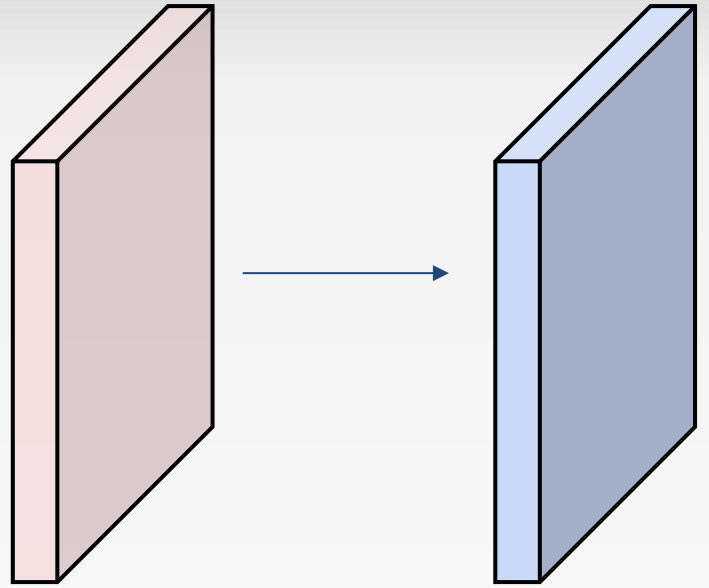
EXAMPLES TIME:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

EXAMPLES TIME:



Input volume: **32x32x3**

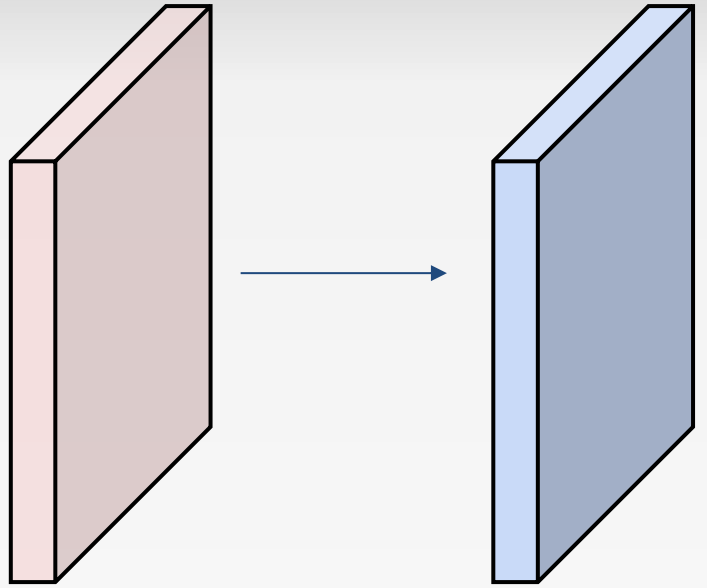
10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

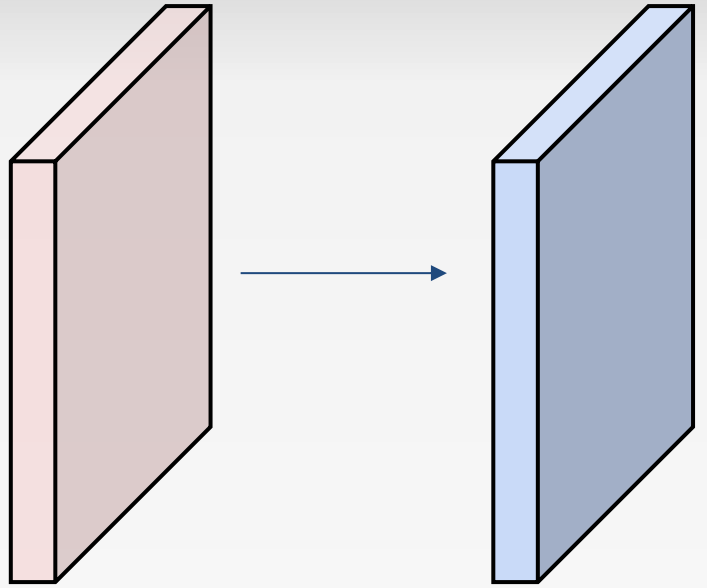
EXAMPLES TIME:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

EXAMPLES TIME:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

SUMMARY: CONV LAYER

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

SUMMARY: CONV LAYER

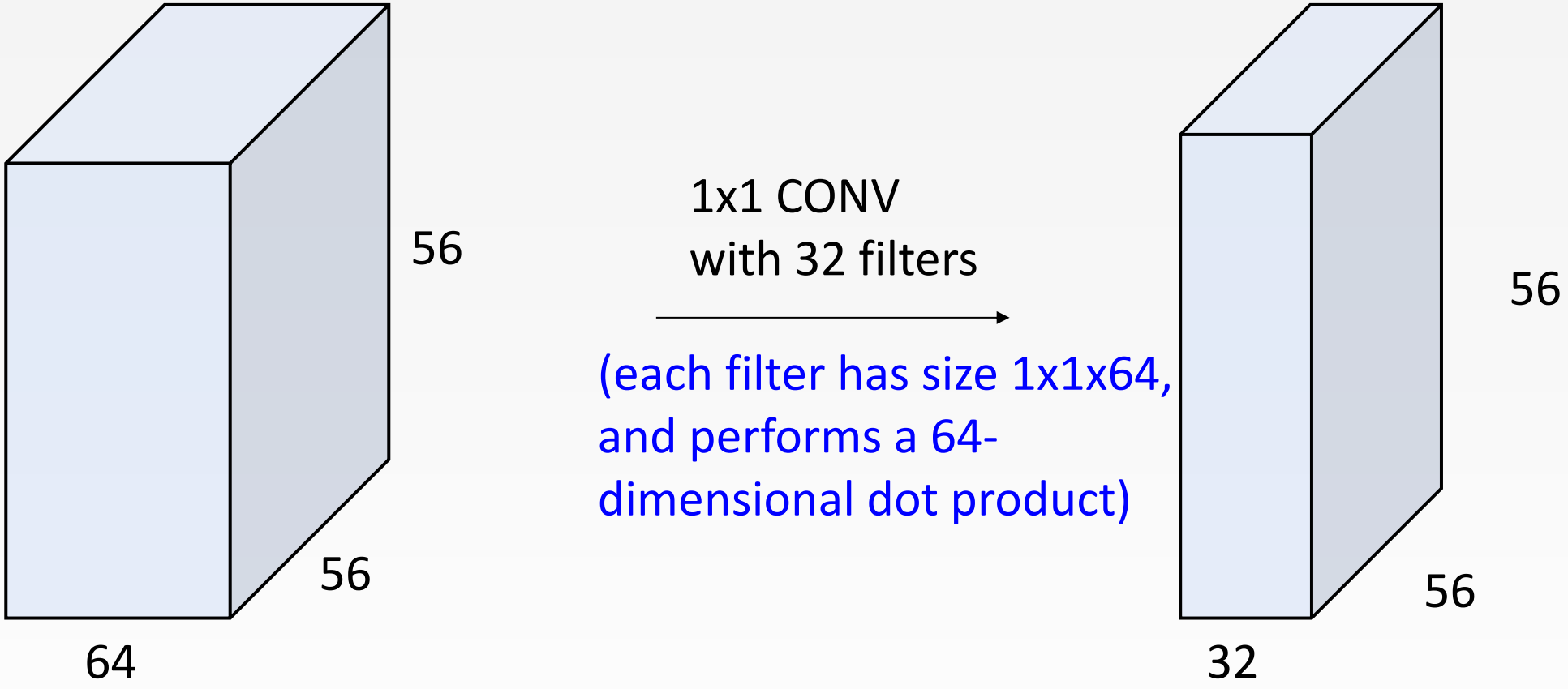
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0


56
56
64
1x1 CONV
with 32 filters
(each filter has size 1x1x64, and performs a 64-dimensional dot product)
56
56
32

Example: CONV layer in PyTorch

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

Conv2d

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit zero-paddings on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At groups=1, all inputs are convolved to all outputs.
  - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels, and producing half the output channels, and both subsequently concatenated.
  - At groups= `in_channels`, each input channel is convolved with its own set of filters, of size: $\left\lfloor \frac{C_{\text{out}}}{C_{\text{in}}} \right\rfloor$.

The parameters `kernel_size`, `stride`, `padding`, `dilation` can either be:

- a single `int` – in which case the same value is used for the height and width dimension
- a `tuple` of two ints – in which case, the first *int* is used for the height dimension, and the second *int* for the width dimension

Example: CONV layer in Keras

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.

Conv2D [source]

```
keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, d
```

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

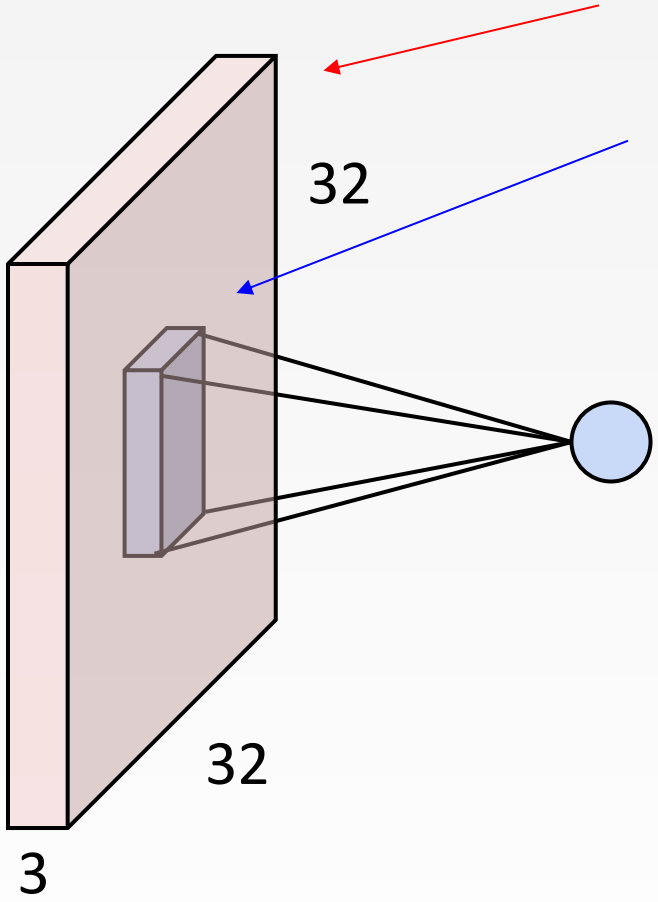
This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

When using this layer as the first layer in a model, provide the keyword argument `input_shape` (tuple of integers, does not include the batch axis), e.g. `input_shape=(128, 128, 3)` for 128x128 RGB pictures in `data_format="channels_last"`.

Arguments

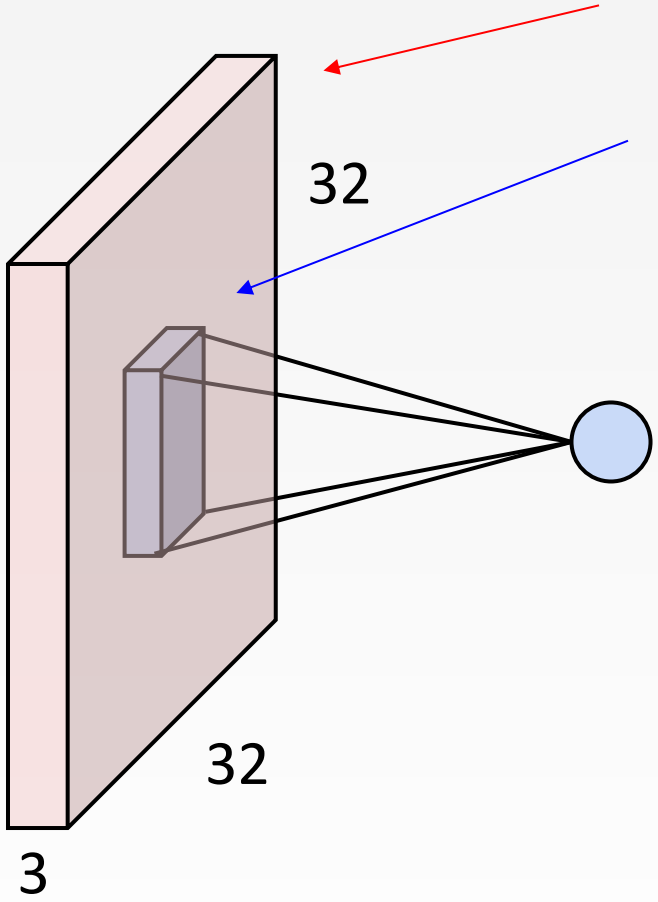
- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: one of `"valid"` or `"same"` (case-insensitive). Note that `"same"` is slightly inconsistent across backends with `strides` != 1, as described here
- **data_format**: A string, one of `"channels_last"` or `"channels_first"`. The ordering of the dimensions in the inputs. `"channels_last"` corresponds to inputs with shape `(batch, height, width, channels)` while `"channels_first"` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

THE BRAIN/NEURON VIEW OF CONV LAYER



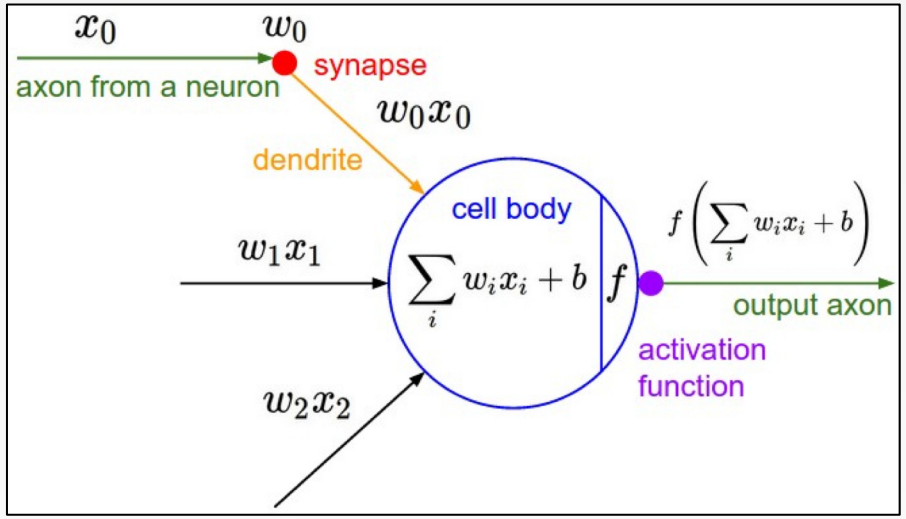
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

THE BRAIN/NEURON VIEW OF CONV LAYER



32x32x3 image

5x5x3 filter

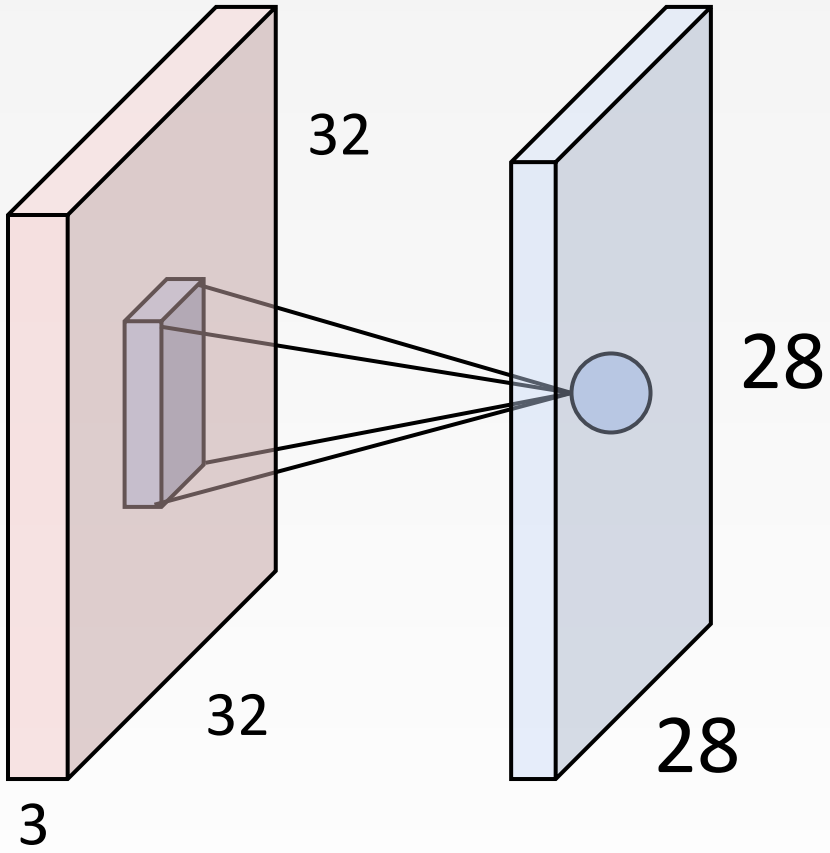


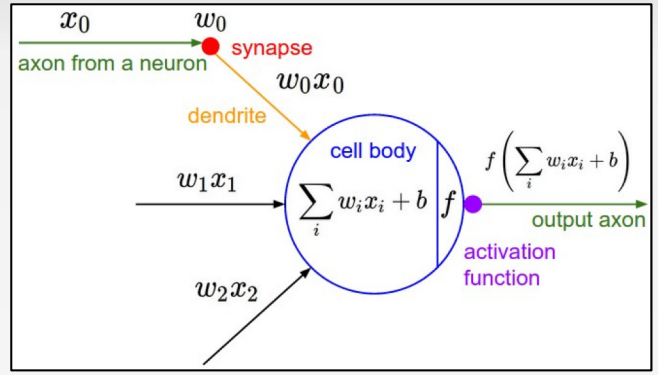
It's just a neuron with

local connectivity...

1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

THE BRAIN/NEURON VIEW OF CONV LAYER



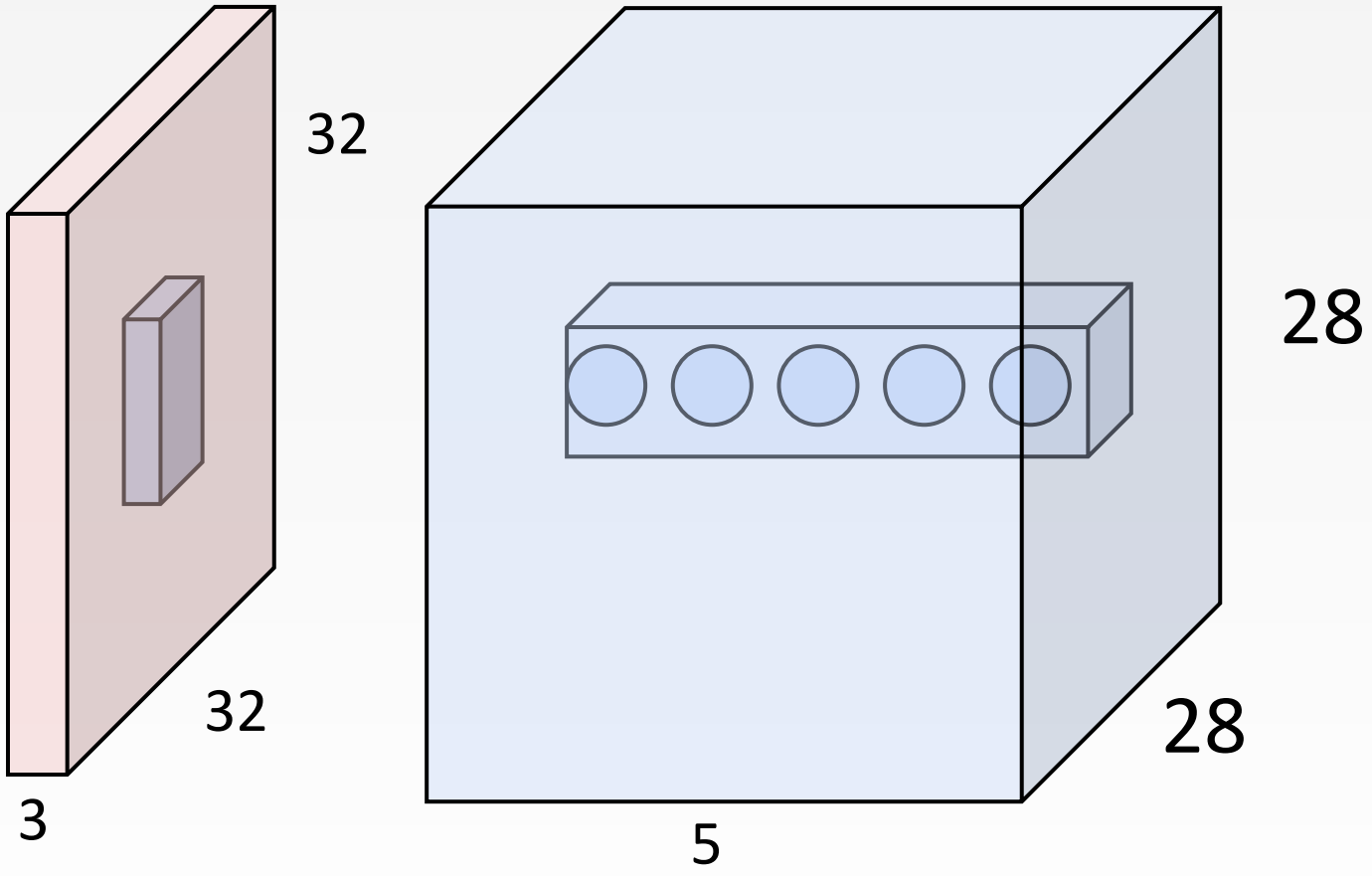


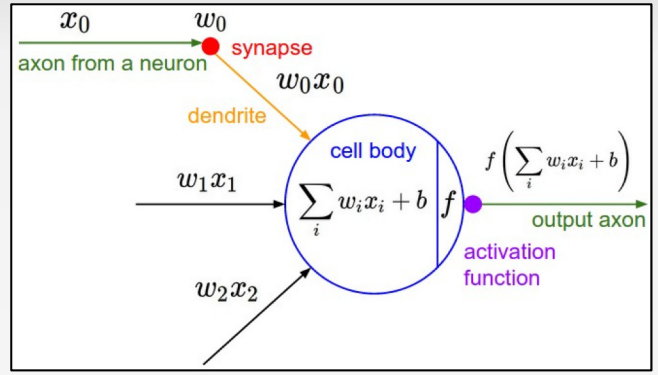
An activation map is a 28x28 sheet of neuron outputs:

1. Each is connected to a small region in the input
2. All of them share parameters

"5x5 filter" -> "5x5 **receptive field** for each neuron"

THE BRAIN/NEURON VIEW OF CONV LAYER





E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

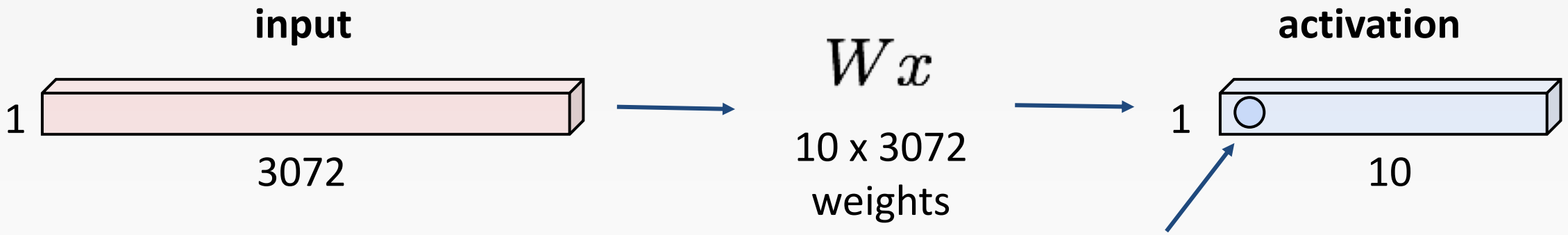
There will be 5 different neurons all looking at the same region in the input volume
but for different things

CONVOLUTIONAL NEURAL NETWORK

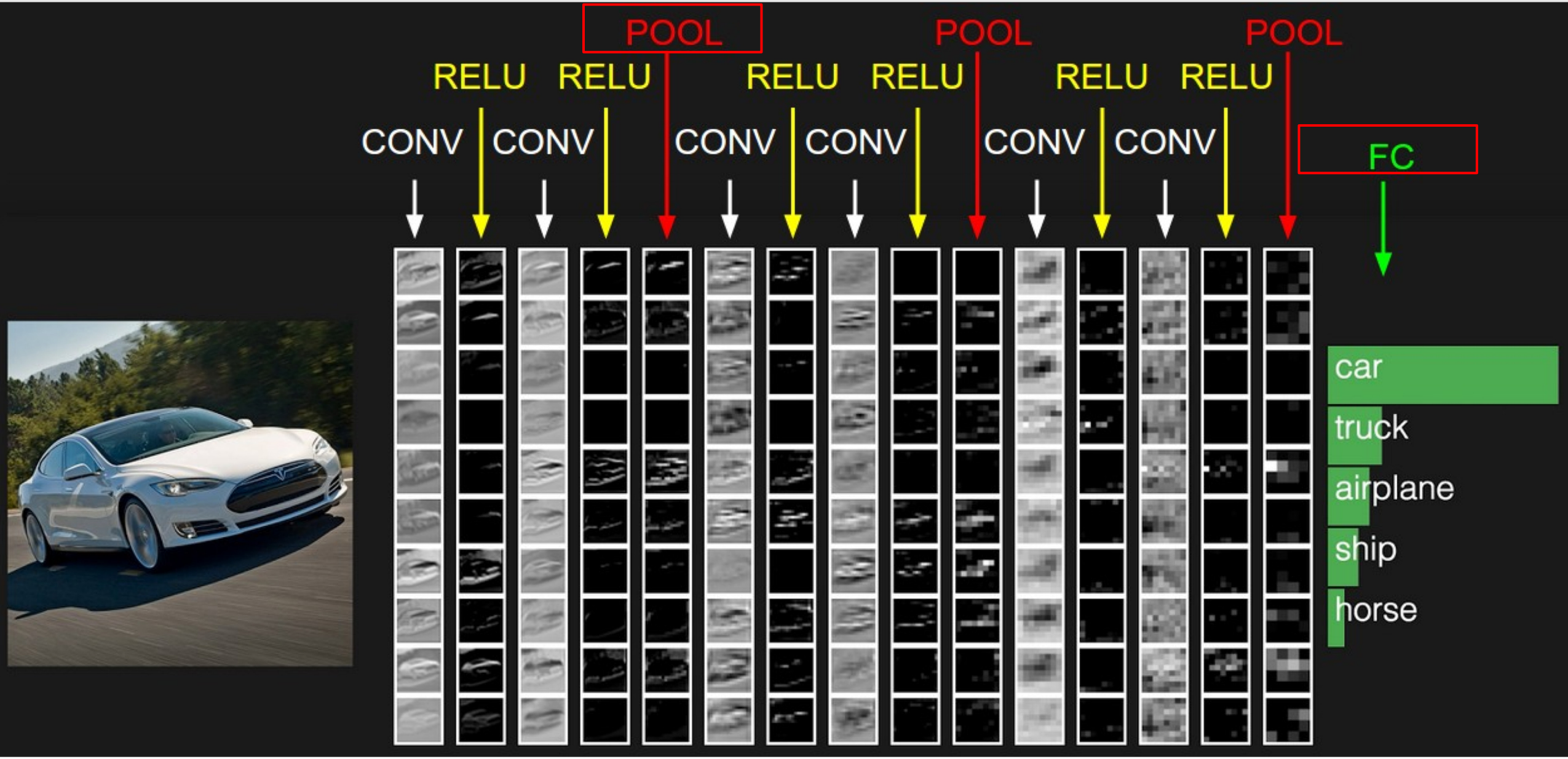
REMINDER: FULLY CONNECTED LAYER

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

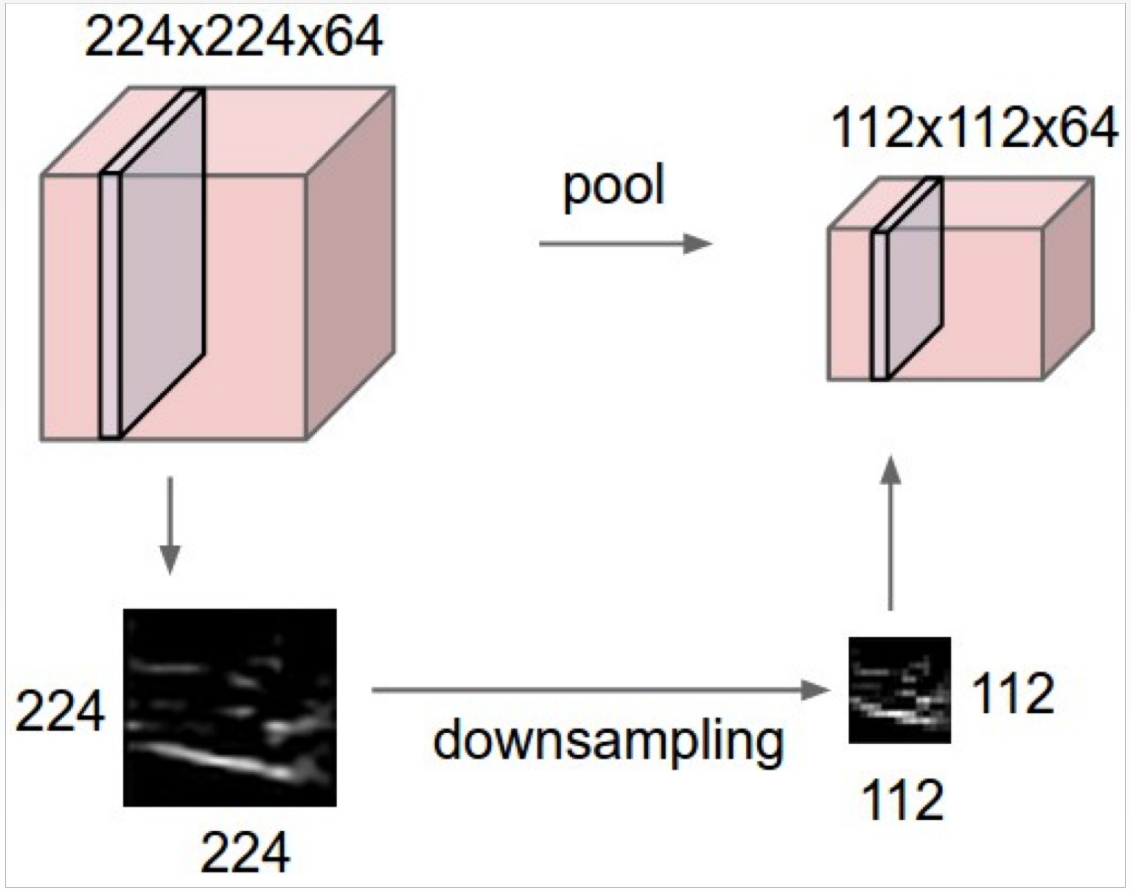


1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

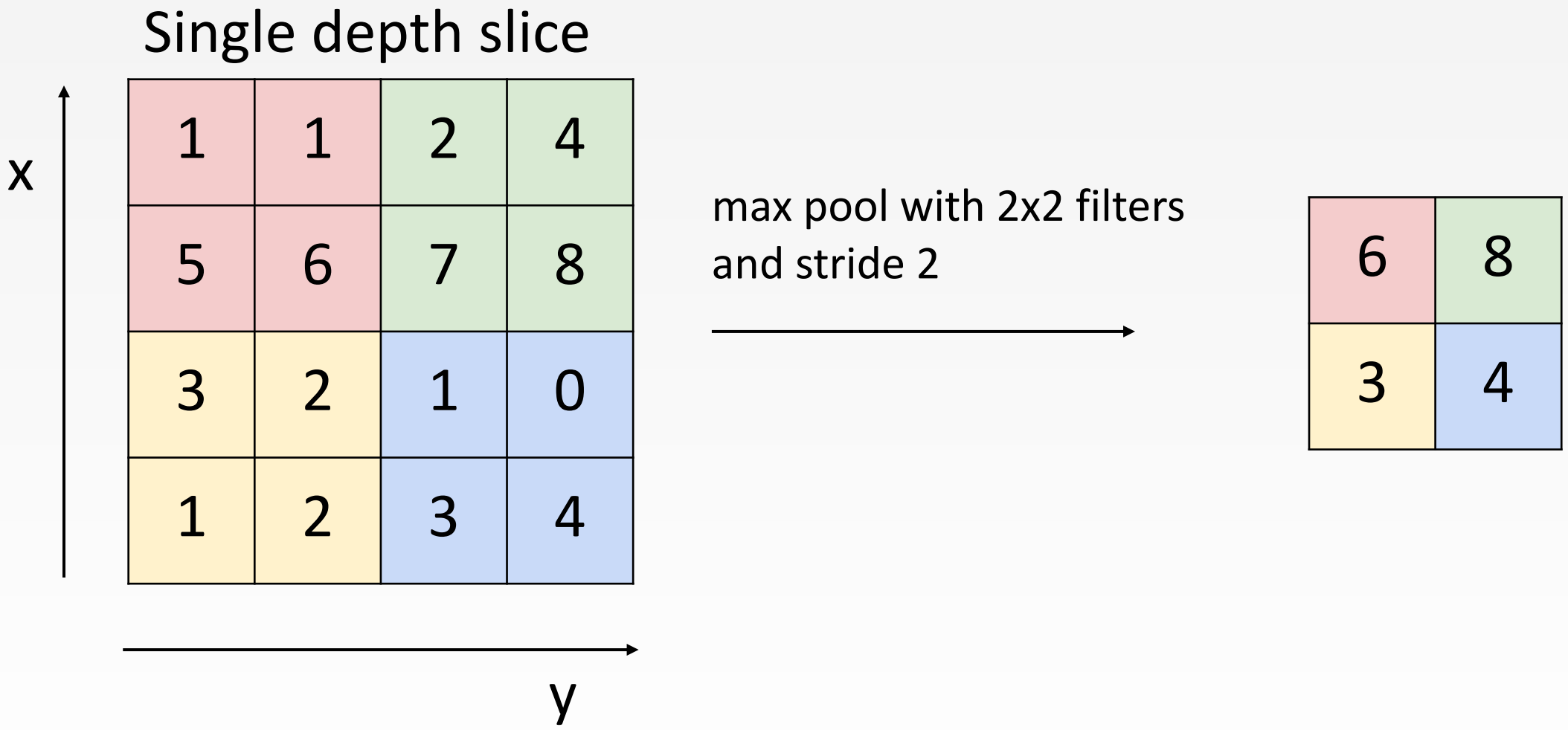

POOL
POOL
POOL
RELU
RELU
RELU
RELU
RELU
RELU
CONV
CONV
CONV
CONV
CONV
CONV
FC
car
truck
airplane
ship
horse

POOLING LAYER

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

POOLING LAYER

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

POOLING LAYER

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

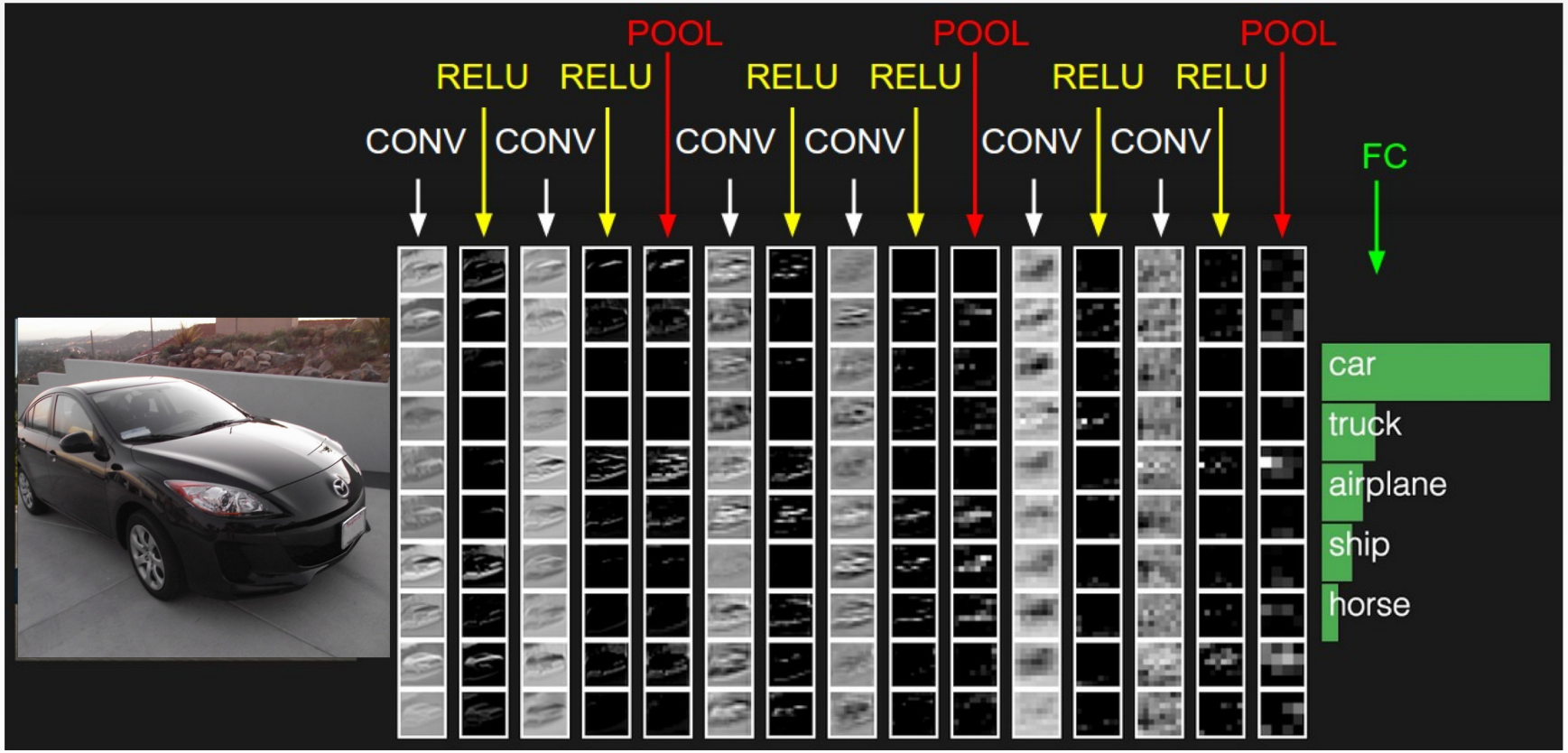
Common settings:

F = 2, S = 2

F = 3, S = 2

FULLY CONNECTED LAYER (FC LAYER)

- Contains neurons that connect to the entire input volume as in ordinary Neural Networks



CONVNET DEMO

ConvNetJS CIFAR-10 demo

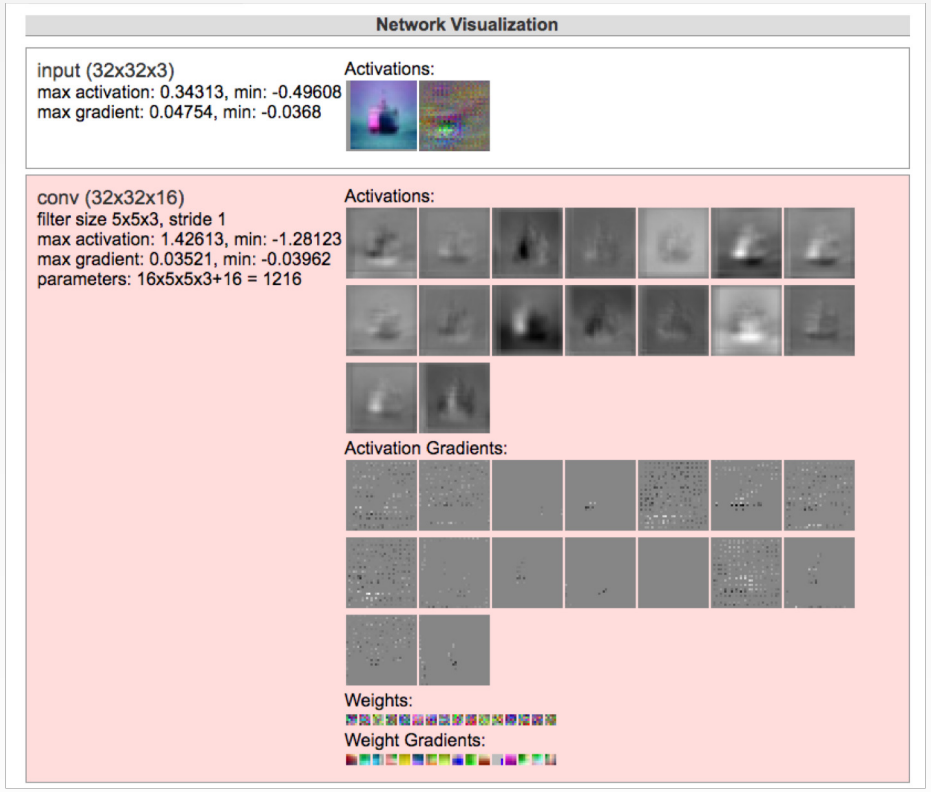
Description

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and verically.

By default, in this demo we're using Adadelta which is one of per-parameter adaptive step size methods, so we don't have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you'd like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.



http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

SUMMARY

- ConvNets stack CONV,POOL,FC layers
  - Trend towards smaller filters and deeper architectures
  - Trend towards getting rid of POOL/FC layers (just CONV)

SUMMARY

- Historically architectures looked like:

[(CONV-RELU)*N-POOL?]*M-(FC-RELU)*K,SOFTMAX

N is usually up to ~5, M is large, 0<=K<= 2.

- But recent advances such as ResNet / GoogLeNet have challenged this paradigm

NEXT LECTURE

- Deep learning hardware
  - CPU, GPU, TPU
- Deep learning software
  - PyTorch and TensorFlow
  - Static and Dynamic computation graphs