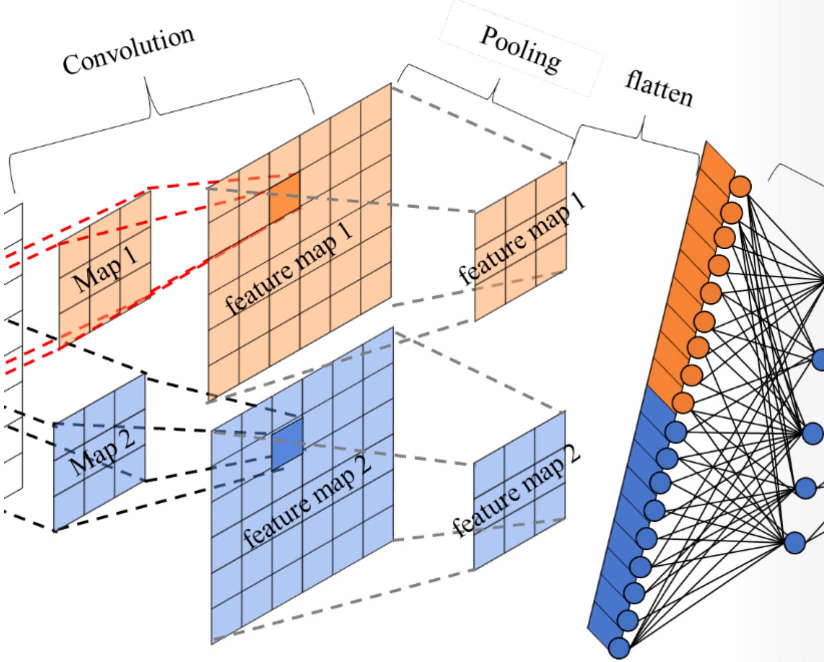
CONVOLUTIONAL NEURAL NETWORKS

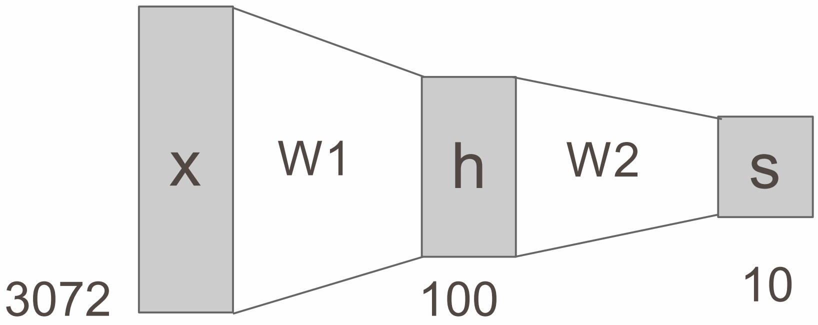
Chih-Chung Hsu (許志仲)
Institute of Data Science
National Cheng Kung University
https://cchsu.info

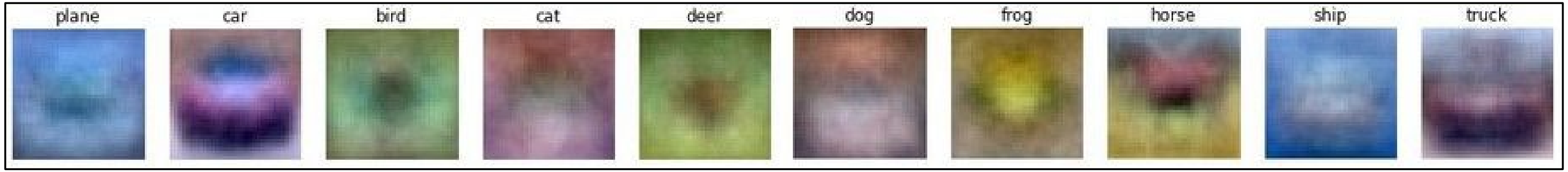


Last time: Neural Networks
Linear score function: 2-layer Neural Network

$$f = Wx$$

$$f = W_2 \max(0, W_1 x)$$





Next: Convolutional Neural Networks

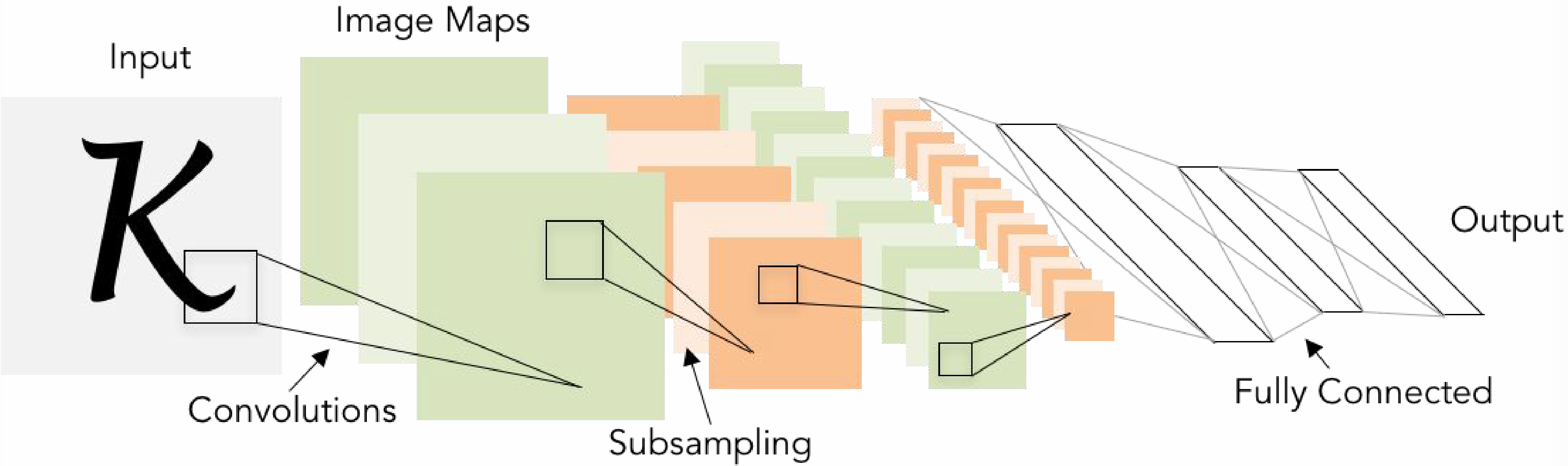


Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1

A bit of history...

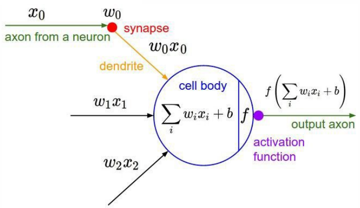
The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

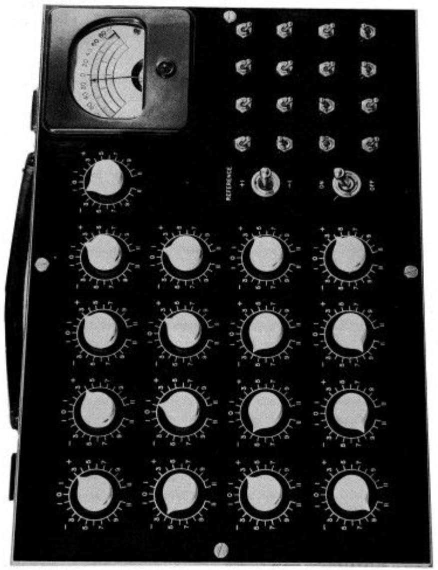
The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

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

recognized
letters of the alphabet

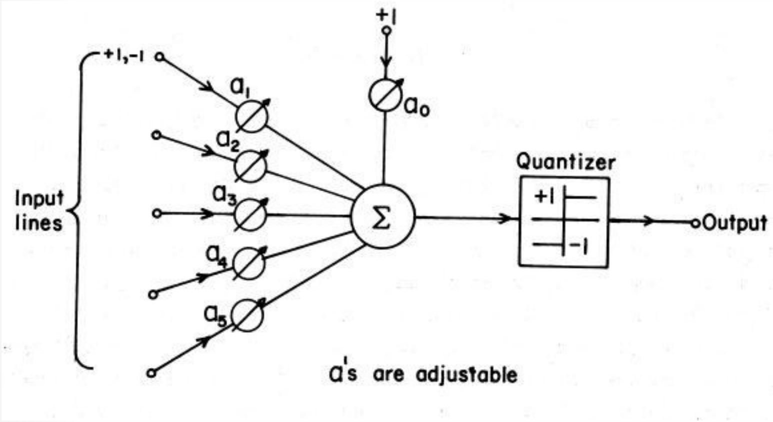
$$w_i(t+1) = w_i(t) + \alpha(d_j - y_j(t))x_{j,i},$$

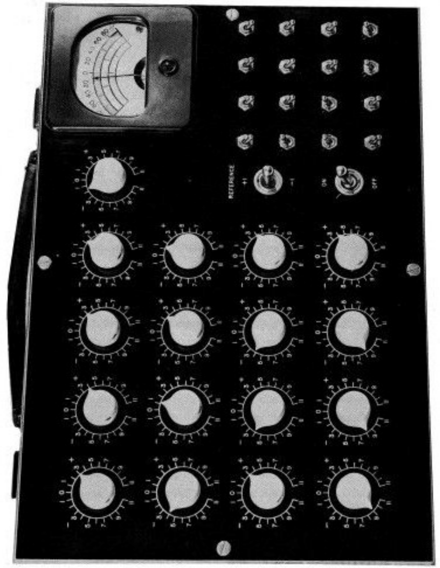


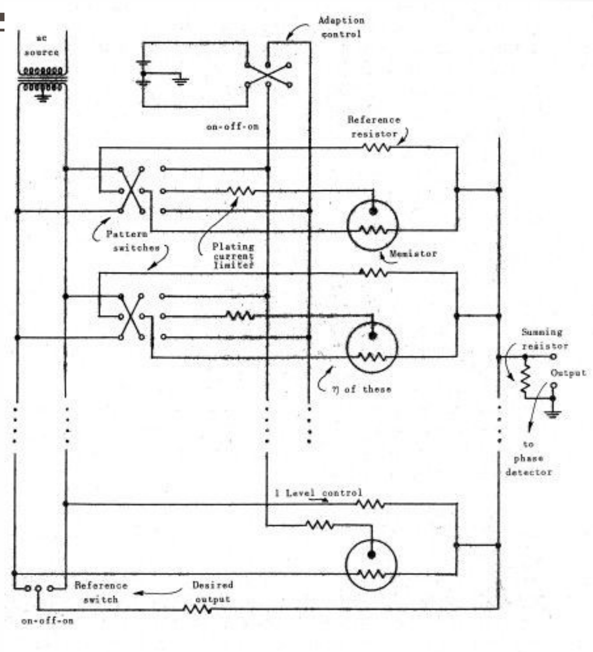


Frank Rosenblatt, ~1957: Perceptron

A bit of history...







Widrow and Hoff, ~1960: Adaline/Madaline

These figures are reproduced from Widrow 1960, Stanford Electronics Laboratories Technical Report with permission from Stanford University Special Collections.

A bit of history...

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

recognizable math

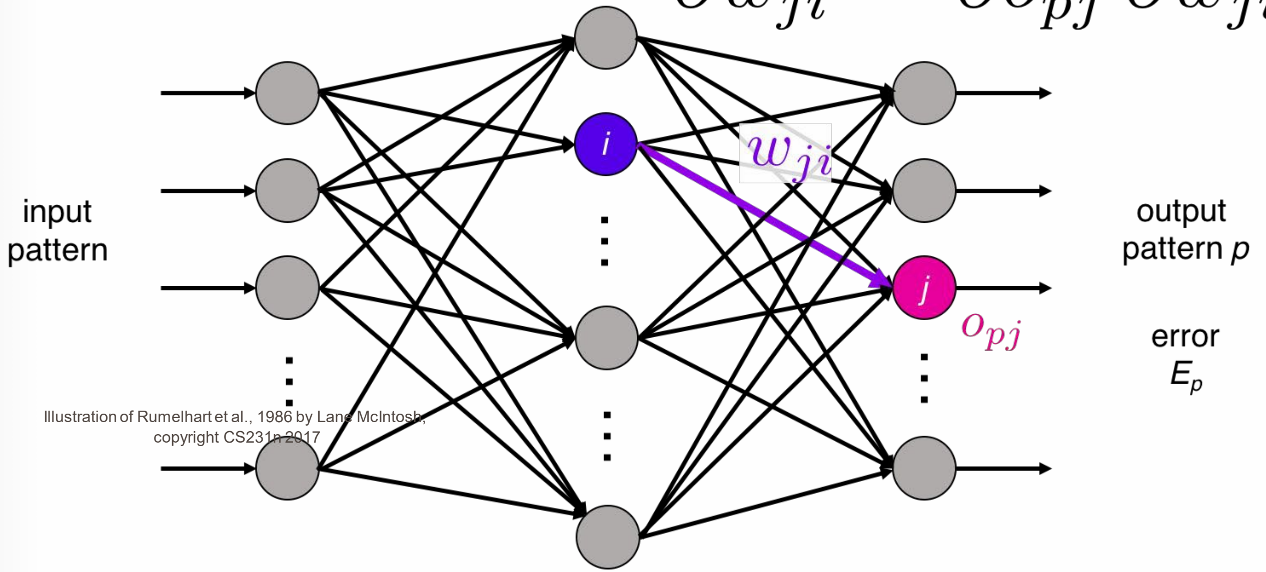


Illustration of Rumelhart et al., 1986 by Lane McIntosh, copyright CS231n 2017

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

A bit of history...

[Hinton and Salakhutdinov 2006]

Reinvigorated research in Deep Learning

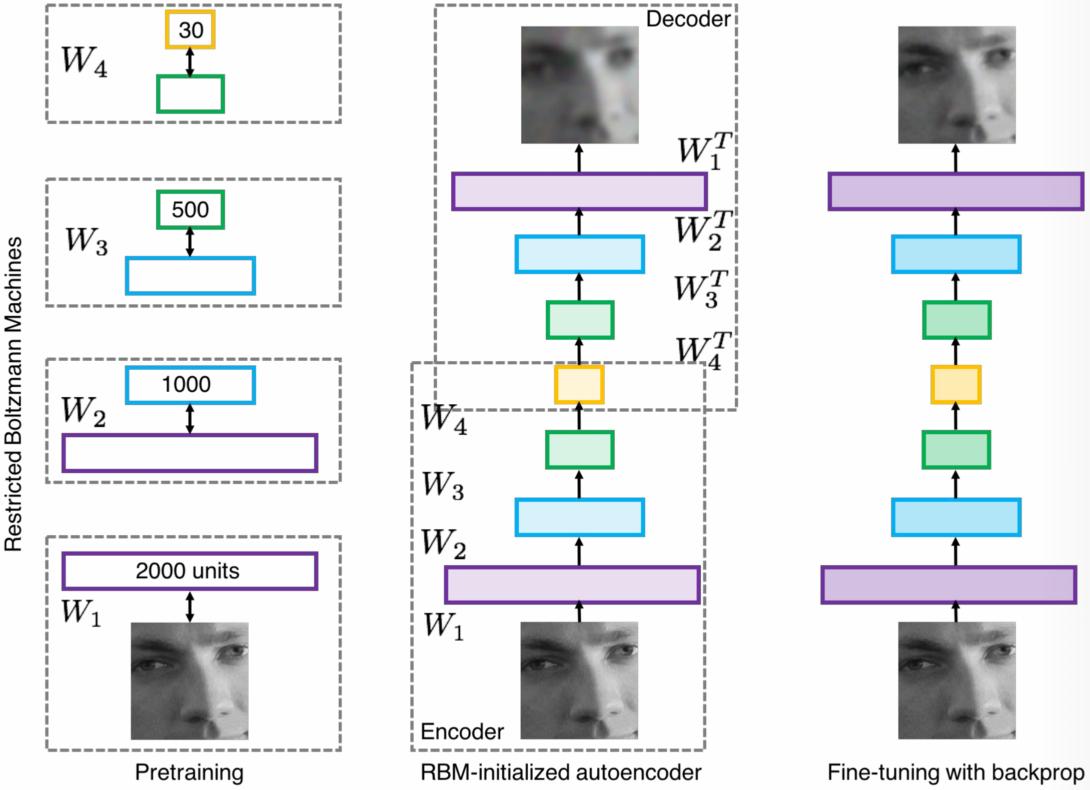


Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

First strong results

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

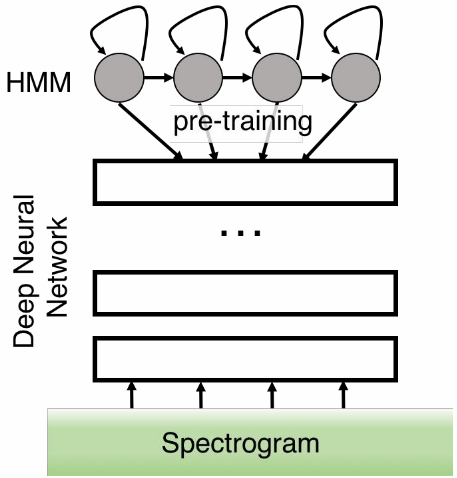
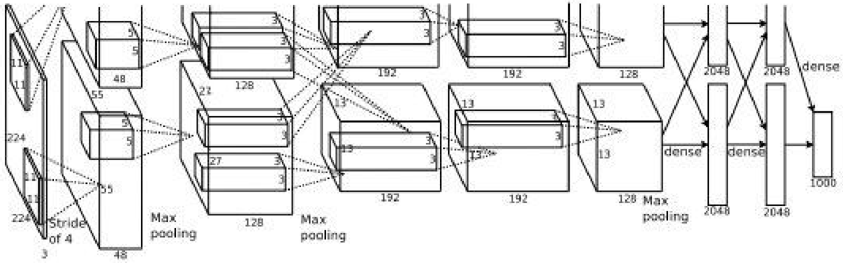
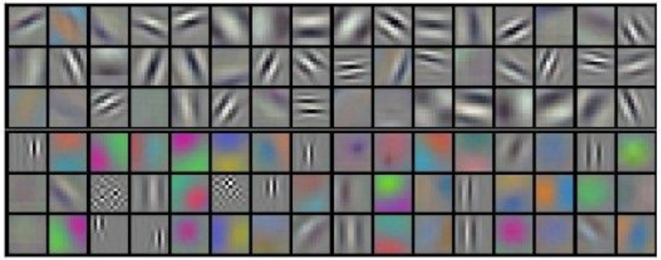


Illustration of Dahl et al. 2012 by Lane McIntosh, copyright CS231n 2017

Imagenet classification with deep convolutional neural networks
Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012





A bit of history:

Hubel & Wiesel,

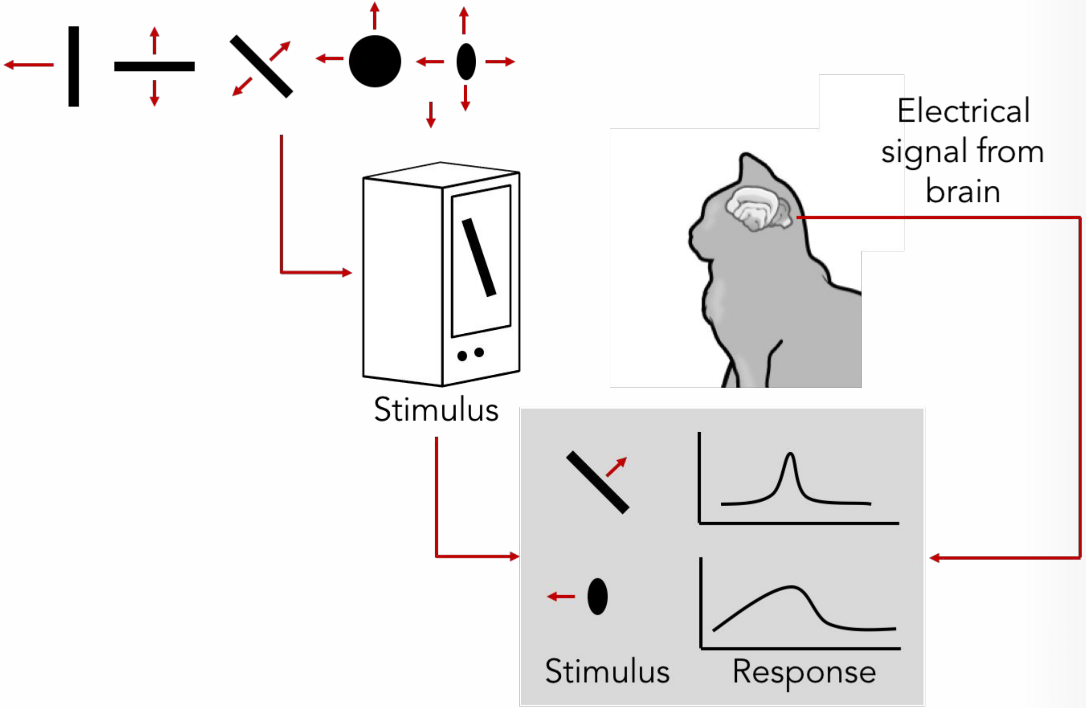
1959

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...

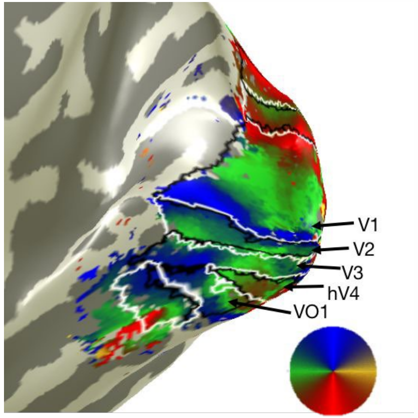


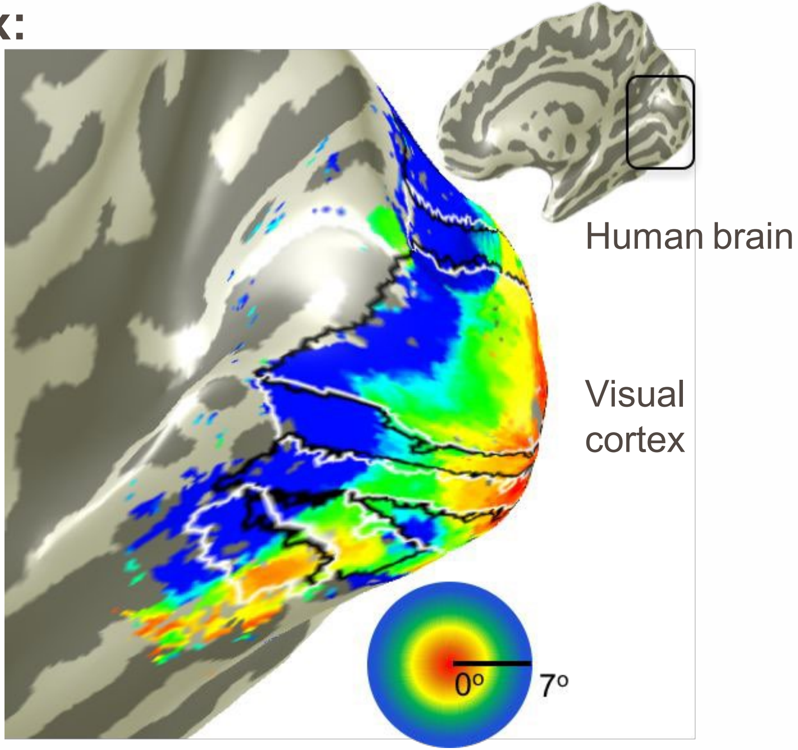
A bit of history

Topographical mapping in the cortex:

nearby cells in cortex represent

nearby regions in the visual field





Retinotopy images courtesy of Jesse Gomez in the Stanford Vision & Perception Neuroscience Lab.

Hierarchical organization

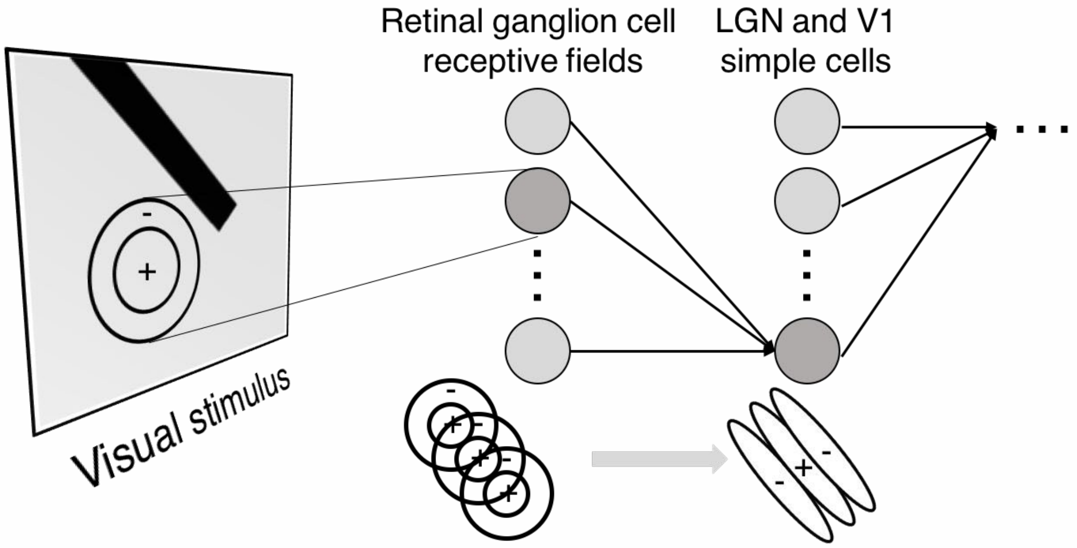
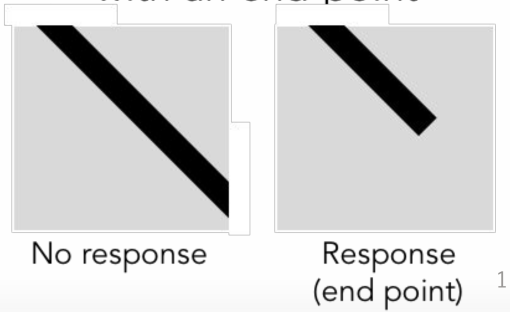


Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017

Simple cells:
Response to light orientation

Complex cells:
Response to light orientation and movement

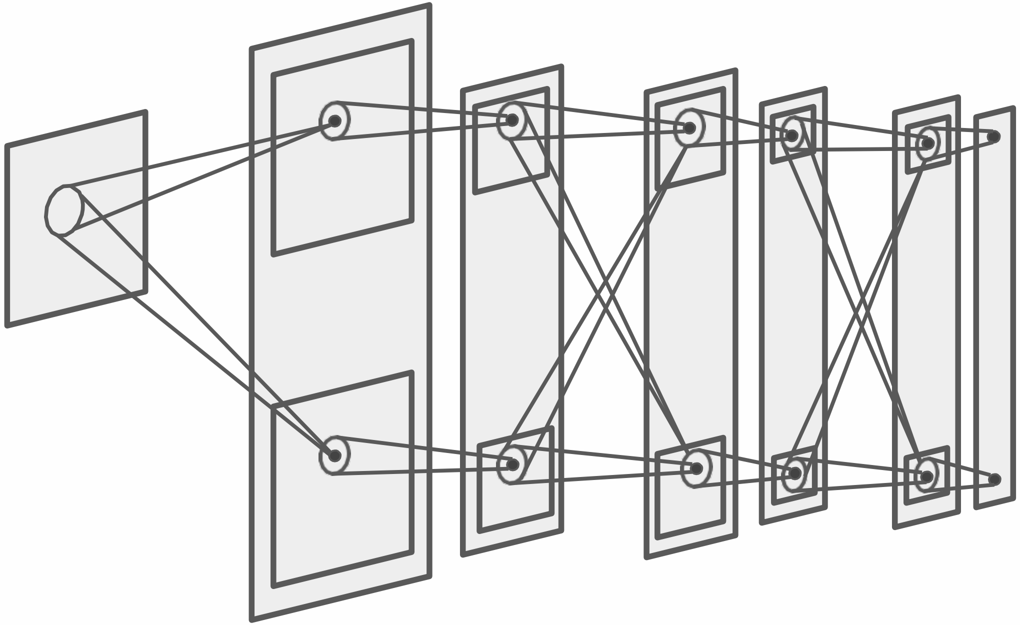
Hypercomplex cells:
response to movement with an end point



A bit of history:

Neocognitron

[Fukushima 1980]

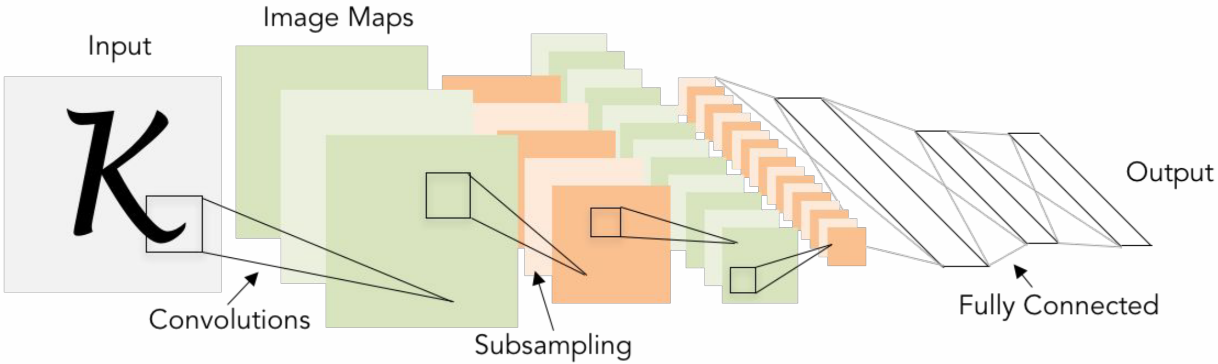


"sandwich" architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

A bit of history:

Gradient-based learning applied to document recognition

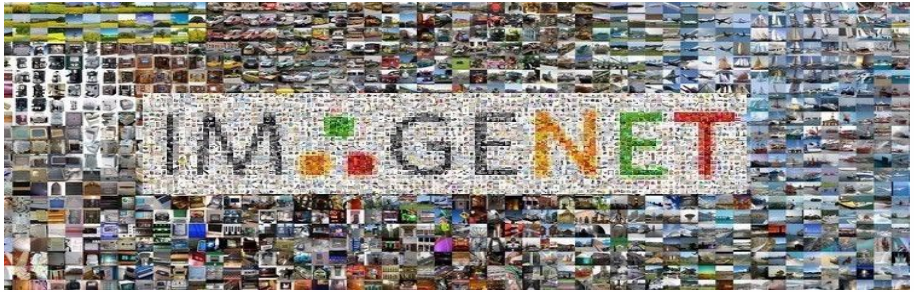
[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks [Krizhevsky, Sutskever, Hinton, 2012]



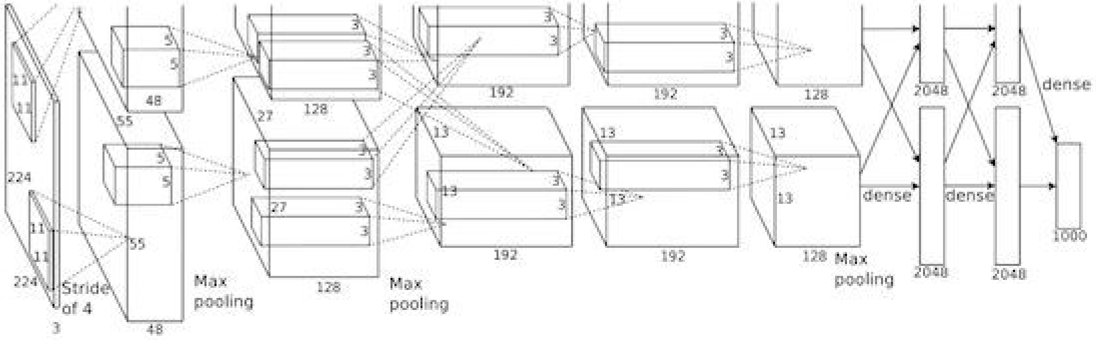


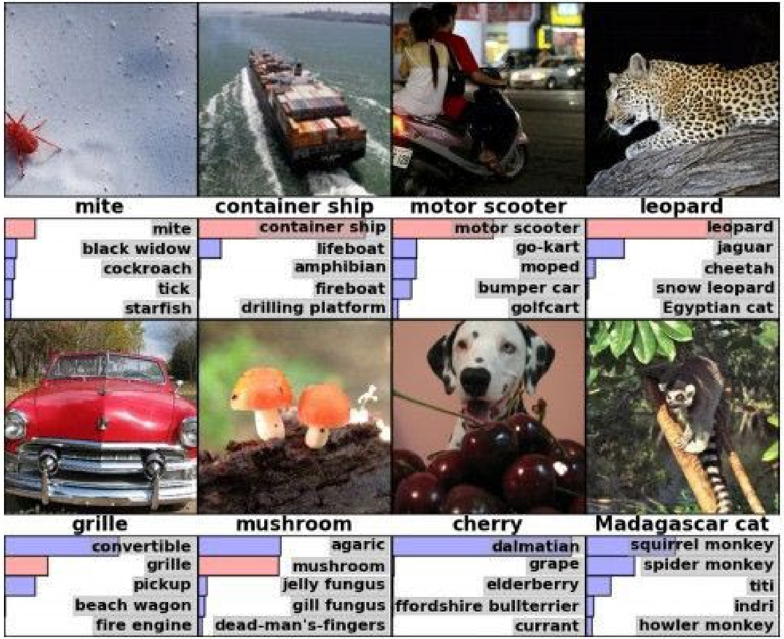
Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

"AlexNet"

17

Fast-forward to today: ConvNets are everywhere

Classification

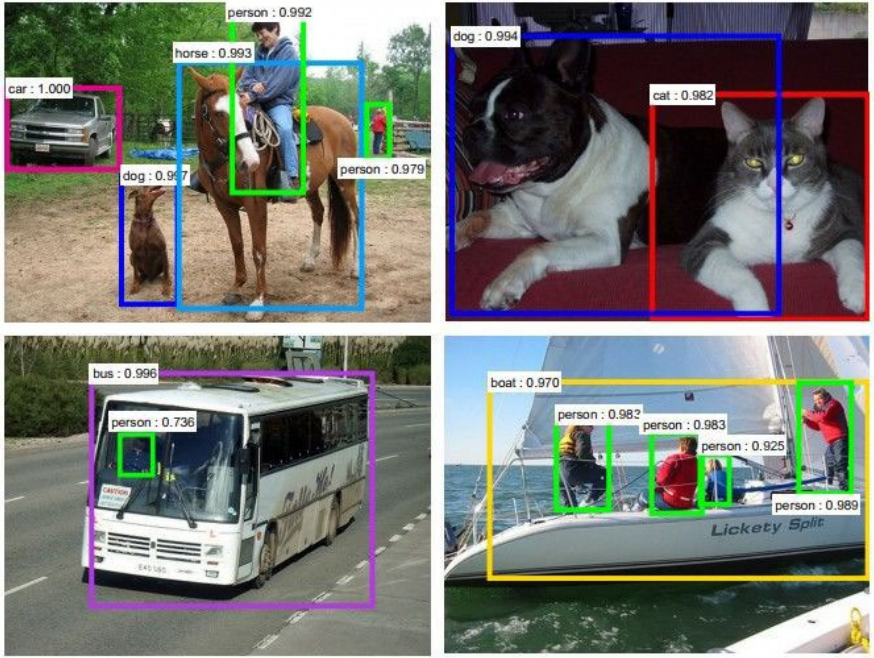


Retrieval

Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Fast-forward to today: ConvNets are everywhere

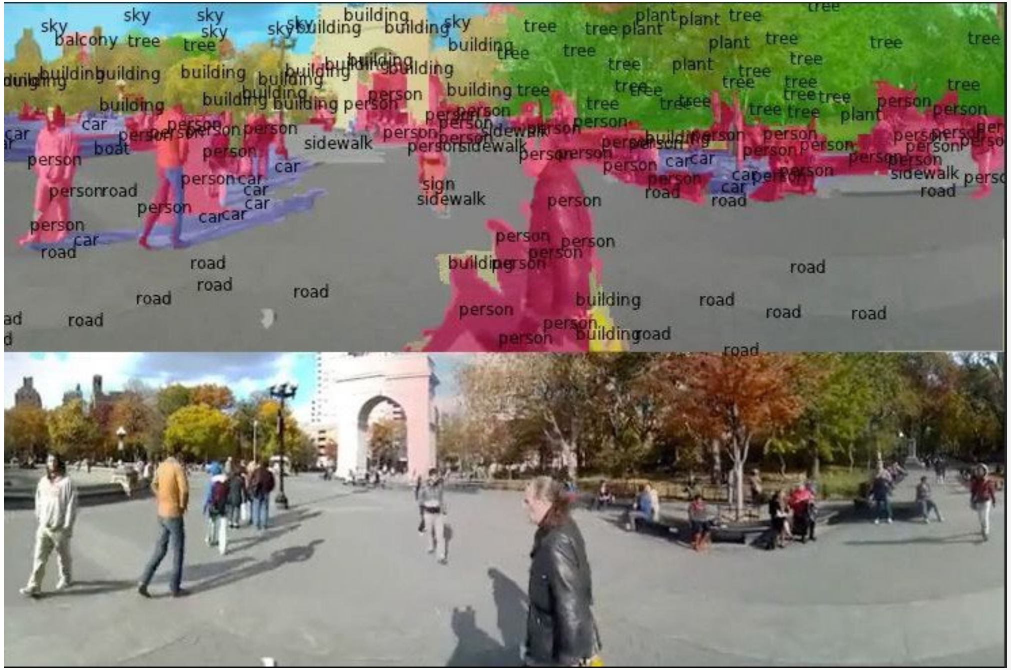
Detection



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

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

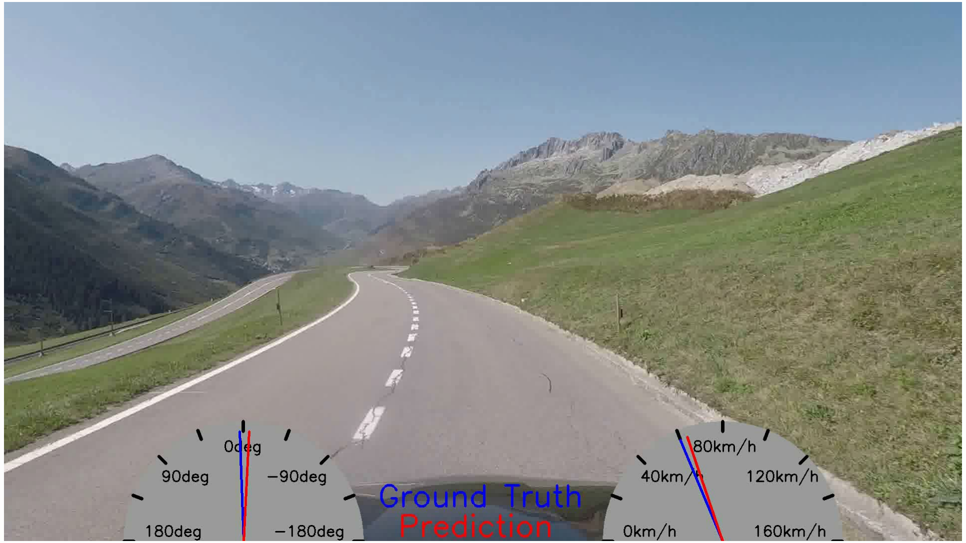
Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

Fast-forward to today: ConvNets are everywhere



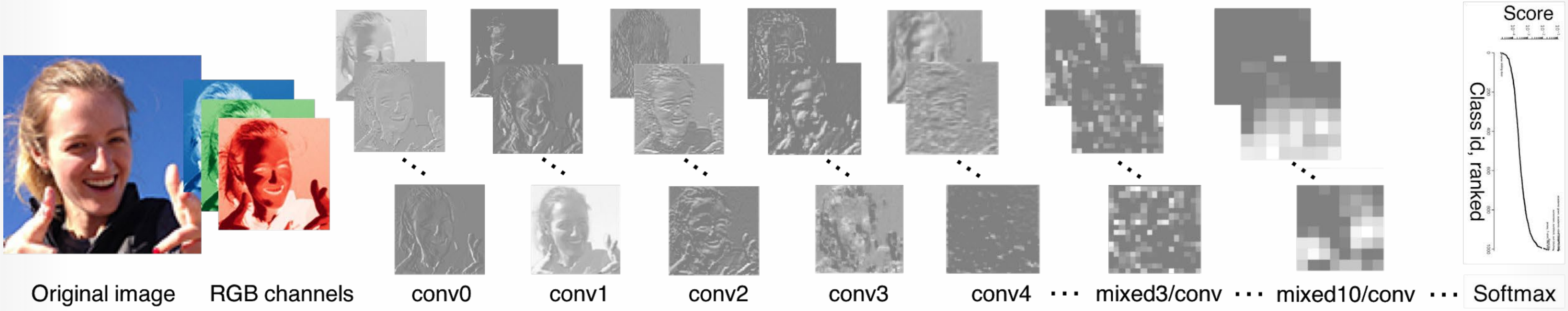
self-driving cars

NVIDIA Tesla line

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

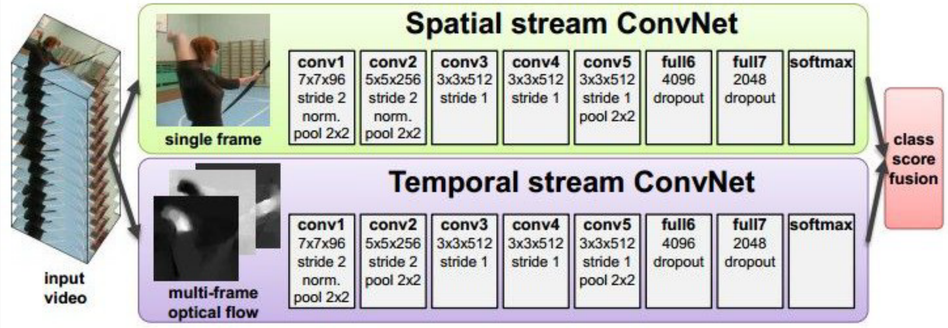
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

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.

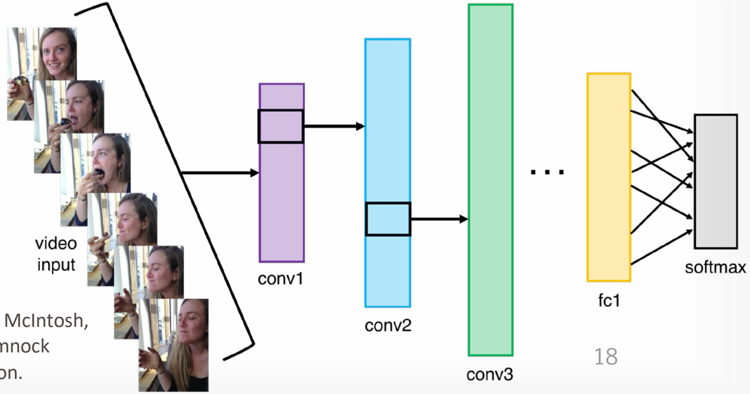
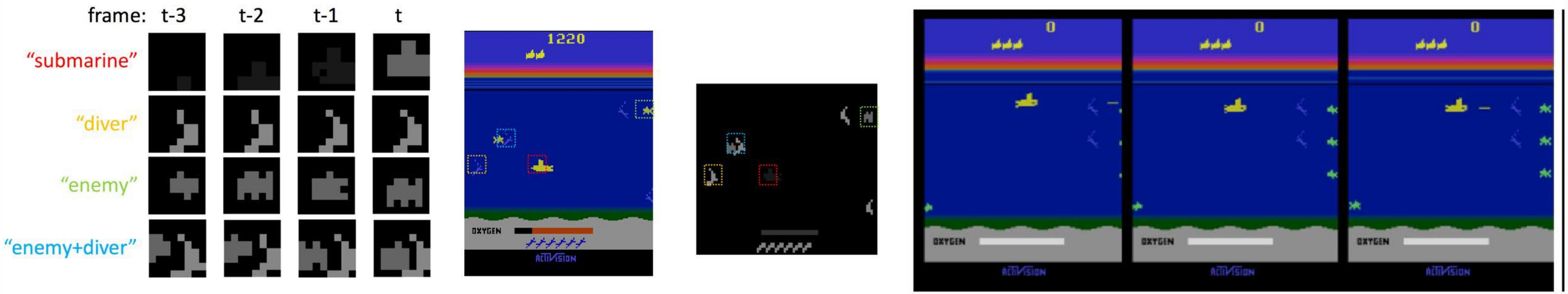


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

Fast-forward to today: ConvNets are everywhere

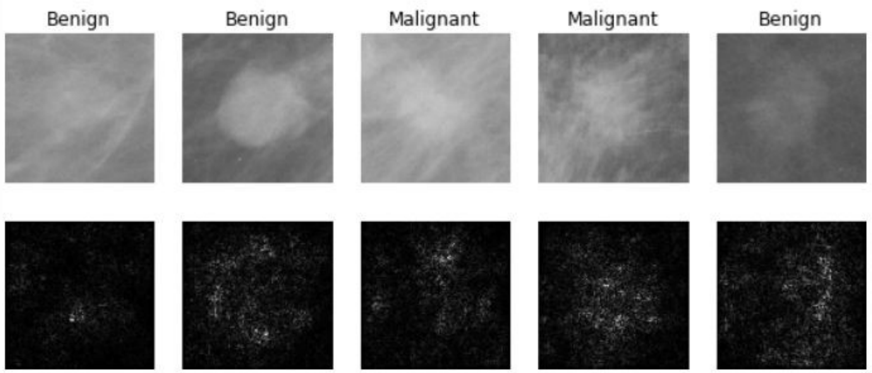
Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Toshev, Szegedy 2014]



[Guo et al. 2014]

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.]

Fast-forward to today: ConvNets are everywhere

[This image](#) by Christin Khan is in the public domain and originally came from the U.S. NOAA.

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

Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010

Correct | Minor errors | Somewhat related

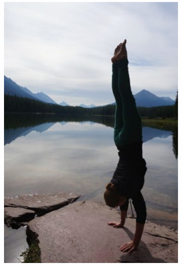
A white teddy bear sitting in the grass

A man in a baseball uniform throwing a ball

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

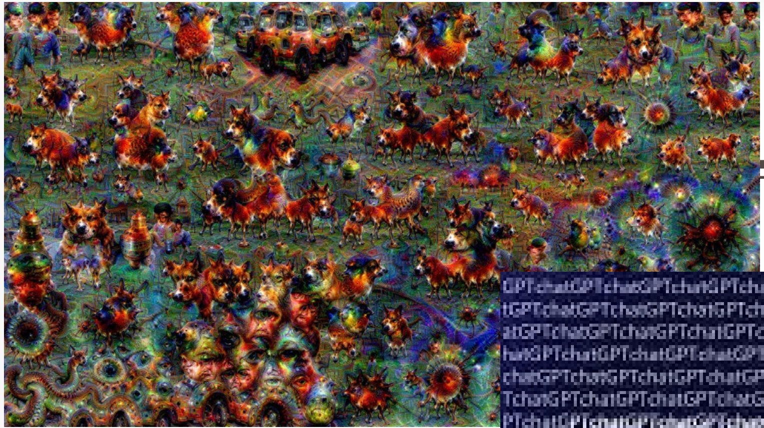


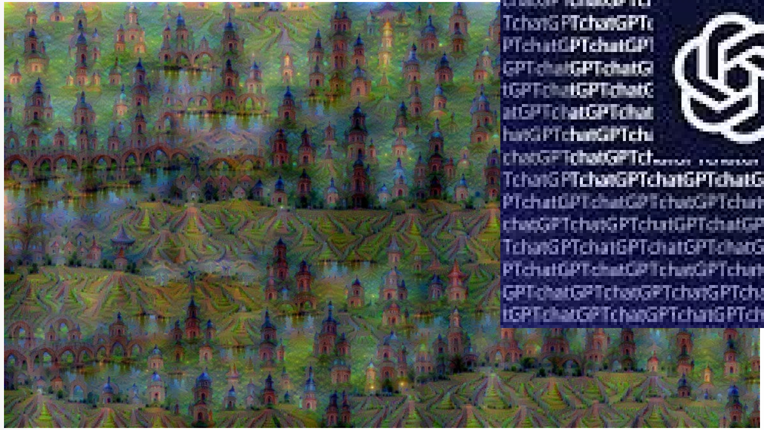
A woman standing on a beach holding a surfboard

Image Captioning

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

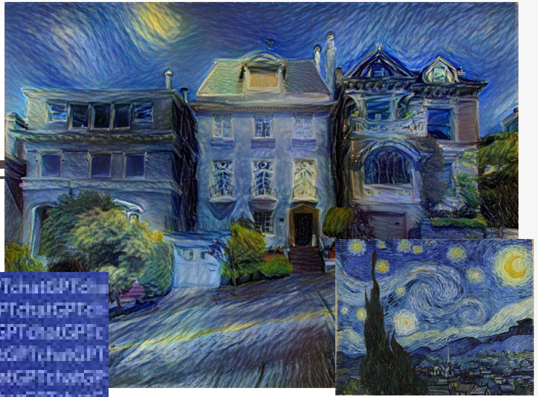
Captions generated by Justin Johnson using Neuraltalk2





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

CONVOLUTIONAL NEURAL NETWORKS

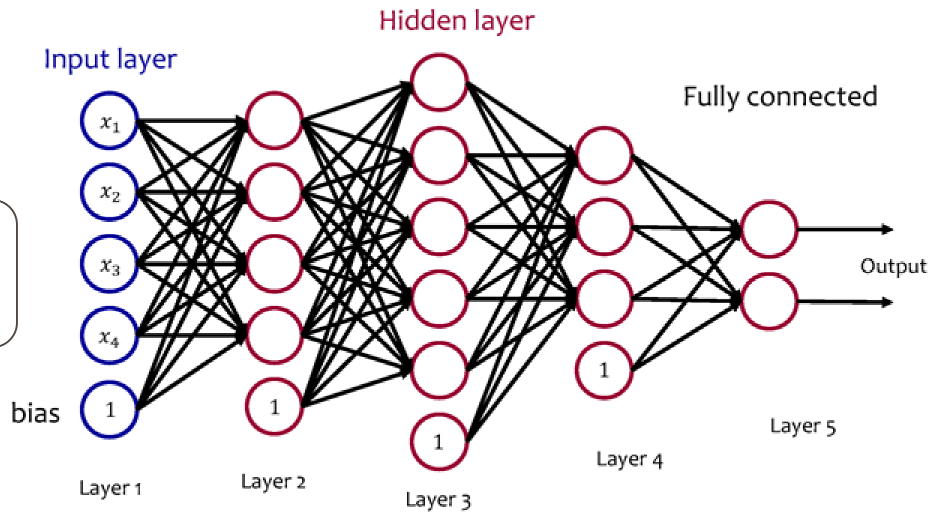
(First without the brain stuff)

Convolutional Neural Networks 捲積神經網路

- Also known as CNN, ConvNet, DCN
- CNN = a multi-layer neural network with
  - Local connectivity
  - Weight sharing
  - Convolution operation
- Parameters reduction
- 保留Spatial information
  - Pixel之間的關係

Definition of Terms in Deep Learning

餵資料
(Data)



吐答案
(Label)

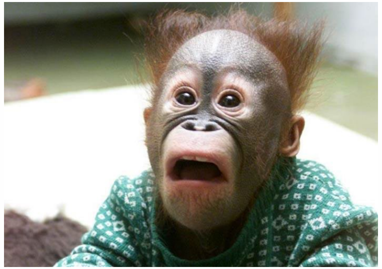
Neurons
Activation function
非線性函數

Connection
Weights (權重)

Parameters (參數)

Number of parameters in DNN

- A lot of number of weights need to be optimized
  - Data: image (100*100=10000), 7-layer NN
    - Assumed you have 60,000 training samples
    - Layer 1-6: 5000 neuros, last layer: 50 neuros
    - 10000*5000+7*(5000*5000)+5000*50 parameters …
    - #parameters = 227550000 for one sample
      - Total parameters to be learned from all samples
      - 227550000 * 60000
    - Weight precision: 32 bit (4 byte)

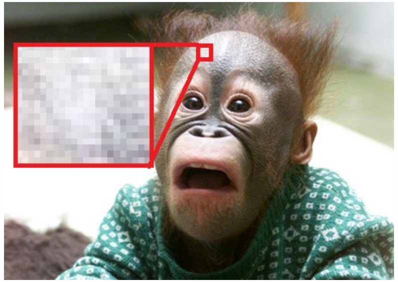


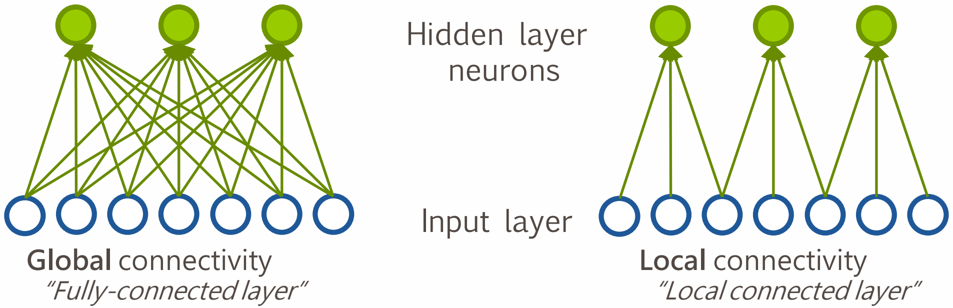
  - So you may aware that we have enough computational resource
    - As the case, we need 5XX GB memory per image!!
    - 5XX GB GPU memory, do you have one?

  - CNN today
    - Reduce the number of parameters as well as improve the performance on image type data

Step 1: Local Connectivity

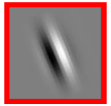
- Assumed that the input neurons: 7
- #nodes in 2nd layer: 3
- #Parameters
  - Global connectivity: 3 x 7 = 21
  - Local connectivity: 3 x 3 = 9



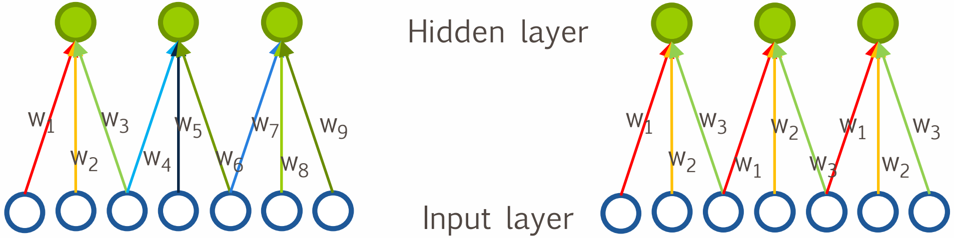


Step 2: Weight Sharing

- Assumed that the input neurons: 7
- #nodes in 2nd layer: 3
- #Parameters
  - Without weight sharing: 3 x 3 = 9
  - With weight sharing : 3 x 1 = 3



convolution kernel:
Use a small kernel to convolve whole image to extract their "local feature"

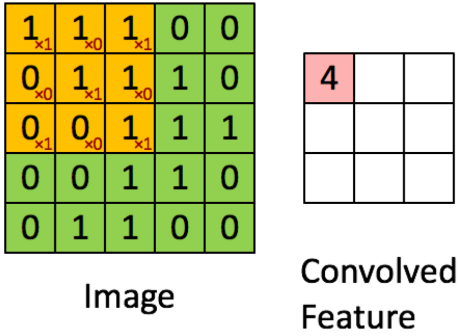


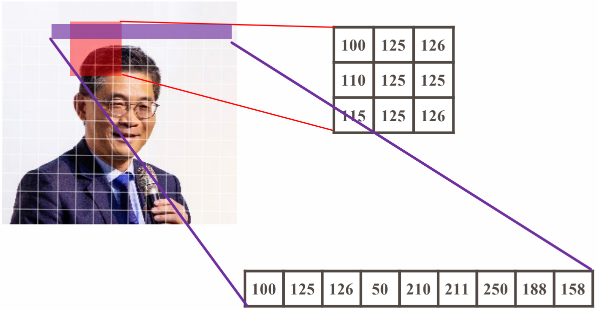
Without weight sharing
"Local connected layer"

With weight sharing
"Convolutional Layer"

Weight sharing?

- Just apply "convolution operation"
  - Improve the performance for images
  - Reduce #parameters





1. Conv: Keep the relation in 2D
2. Vectorization: keep horizontal relation only

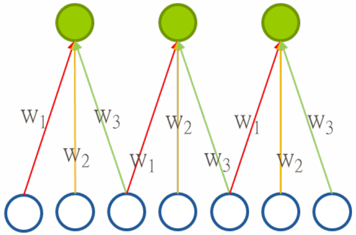
Convolution Operation

- Conv, is based on so-called Filter/Kernel to extract the local features from an image
  - Extract the local "variation"
    - Sample: get the line pattern from an image

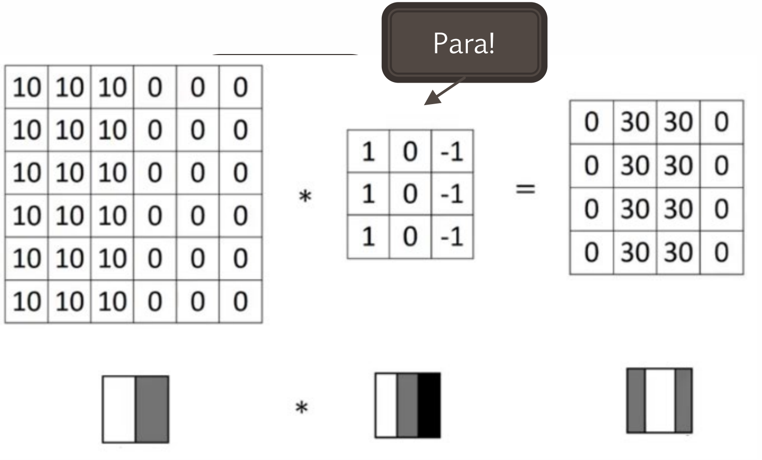
Reduce the #para

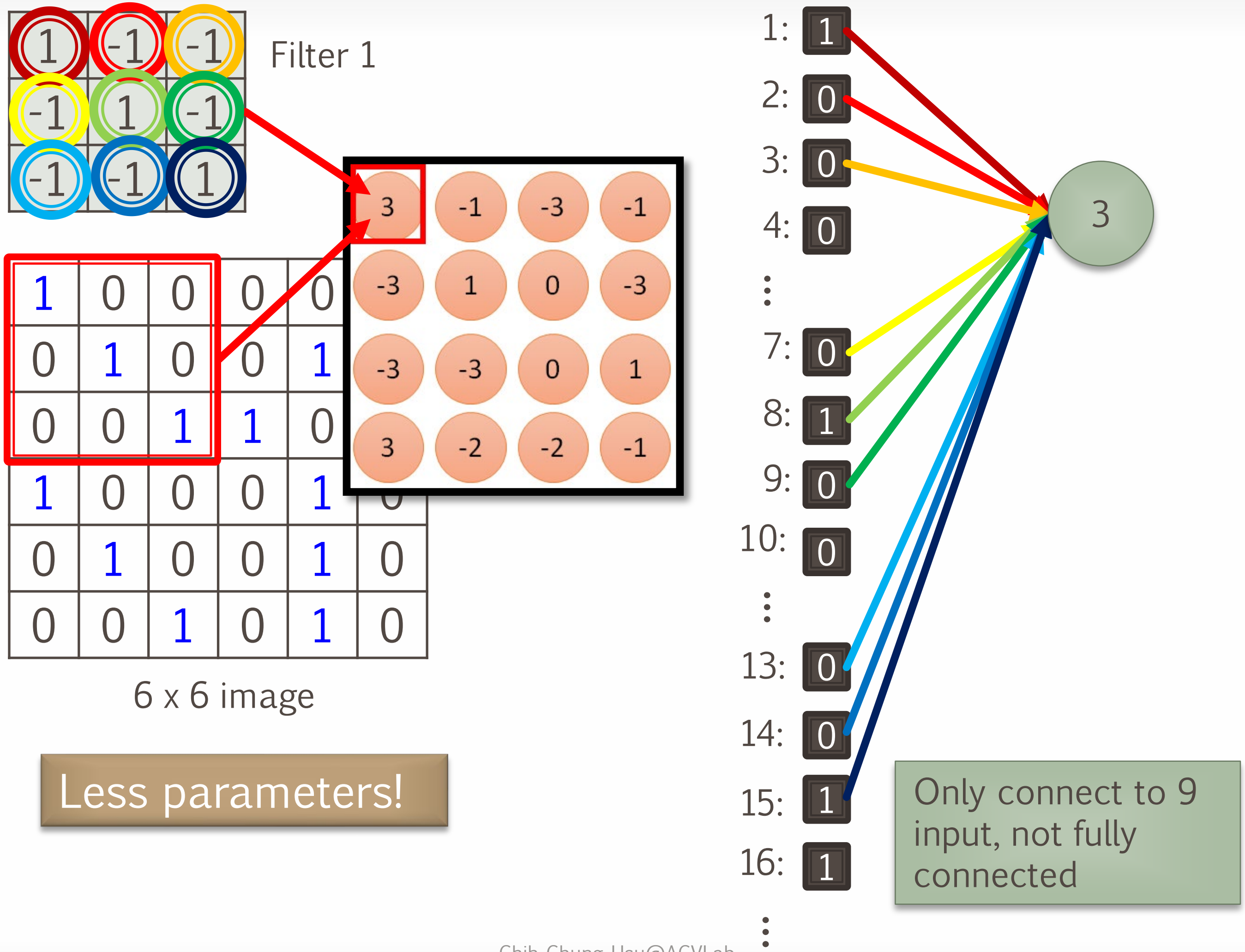
How many para do we need for an image?

ANS: 9
A kernel can scan whole image to extract their local features

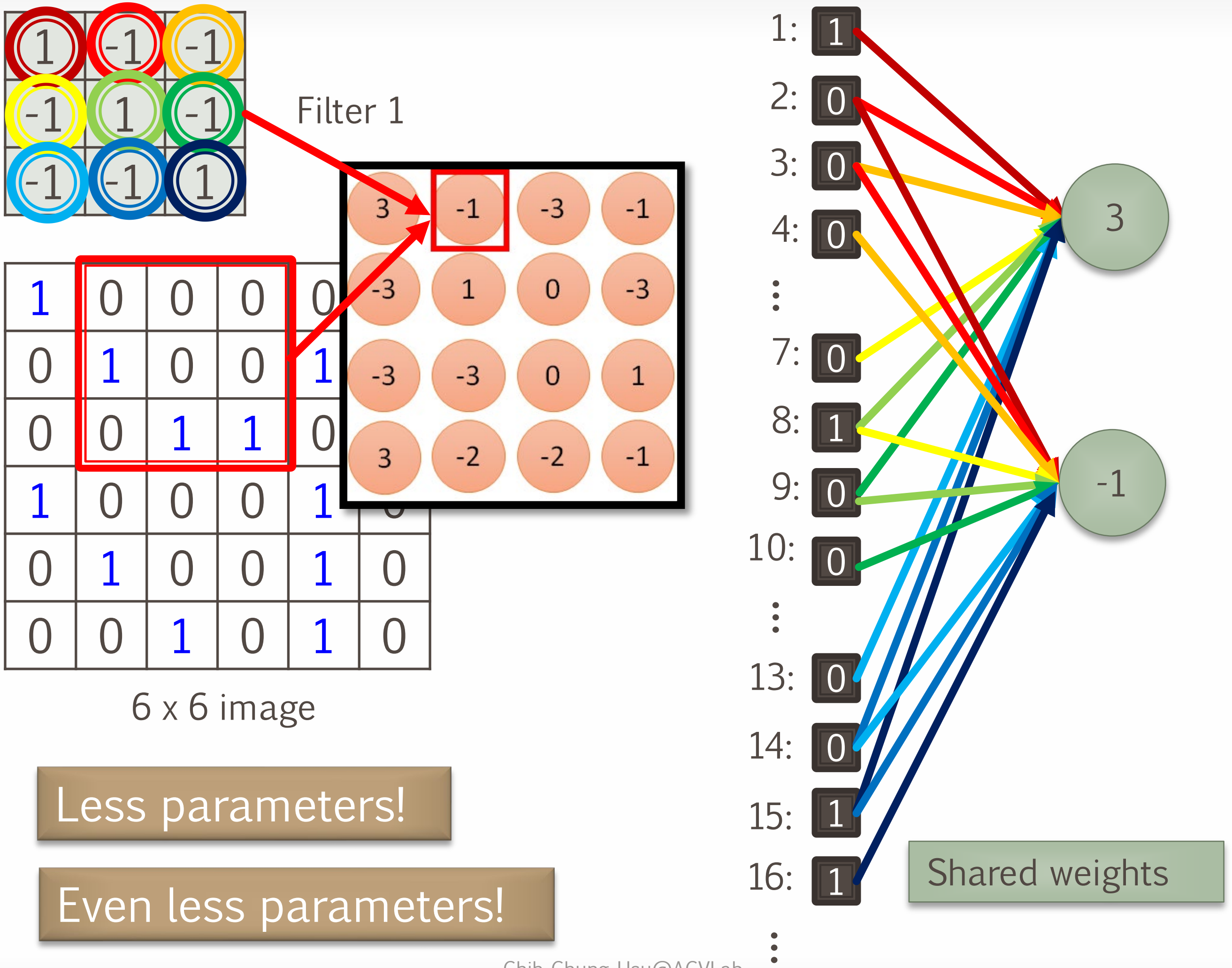


With weight sharing
"Convolutional Layer"





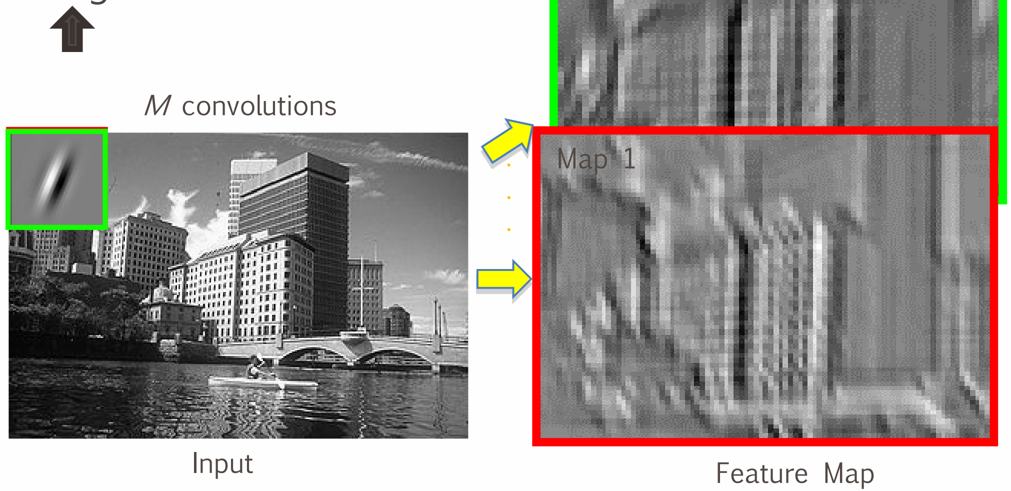
1 -1 -1
-1 1 -1
-1 -1 1
Filter 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Less parameters!
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
Only connect to 9 input, not fully connected



1 -1 -1
-1 1 -1
-1 -1 1
Filter 1
3 -1 -3 -1
-3 1 0 -3
-3 -3 0 1
3 -2 -2 -1
1 0 0 0 0
0 1 0 0 1
0 0 1 1 0
1 0 0 0 1
0 1 0 0 1 0
0 0 1 0 1 0
6 x 6 image
Less parameters!
Even less parameters!
1: 1
2: 0
3: 0
4: 0
7: 0
8: 1
9: 0
10: 0
13: 0
14: 0
15: 1
16: 1
3
-1
Shared weights

Visualization on convolution

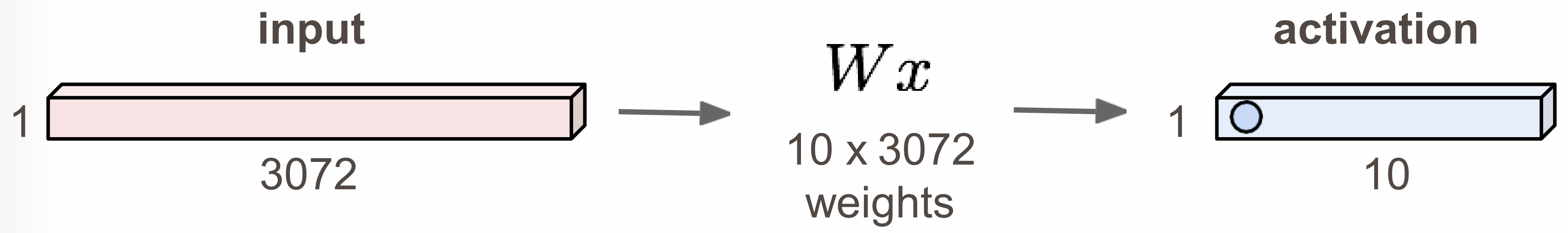
- 2D-Convolution
  - Filtering
  - Kernel
  - Weighted moving sum



CNN IN STRUCTURE VIEW

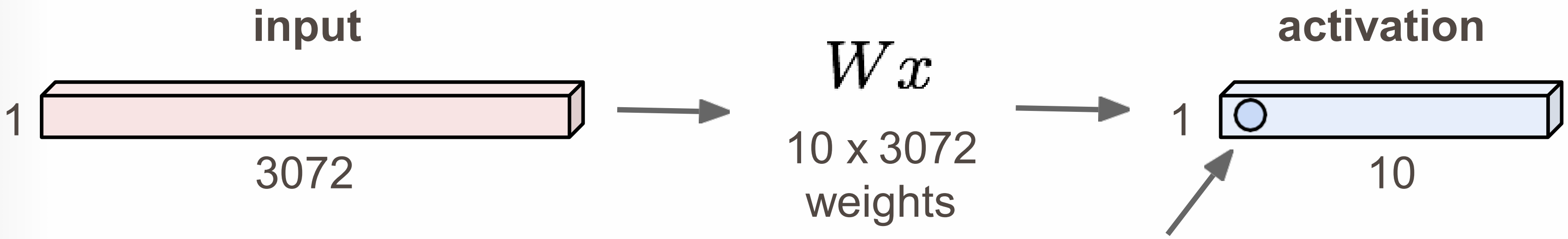
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer

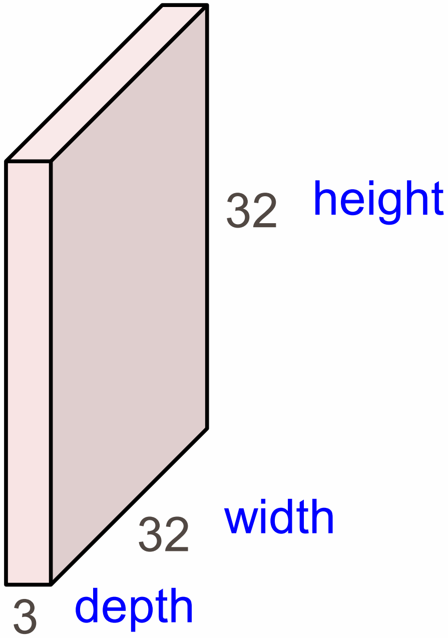
32x32x3 image -> stretch to 3072 x 1



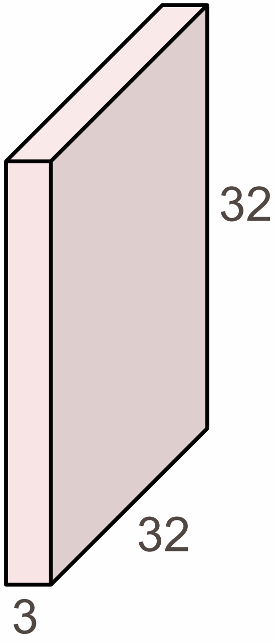
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

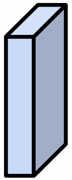
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer





32x32x3 image

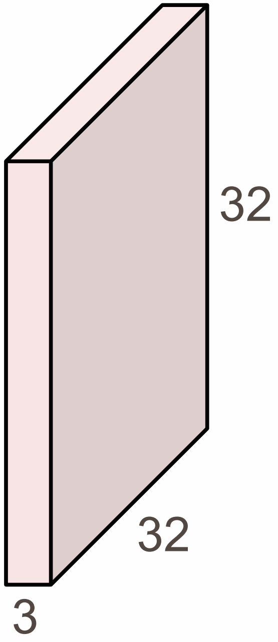
5x5x3 filter

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

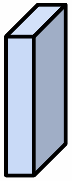
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image

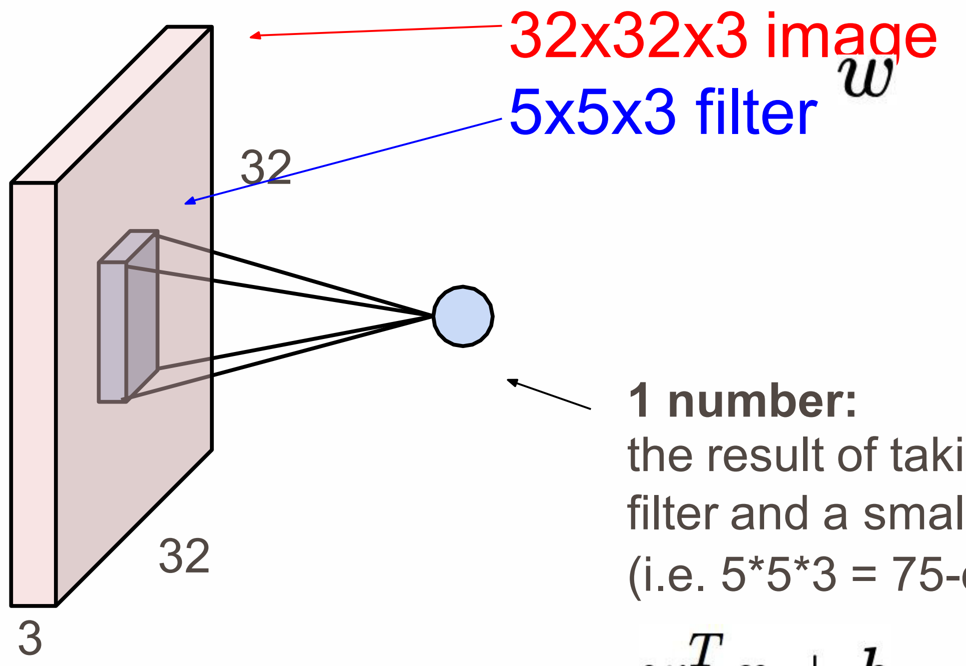


5x5x3 filter



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

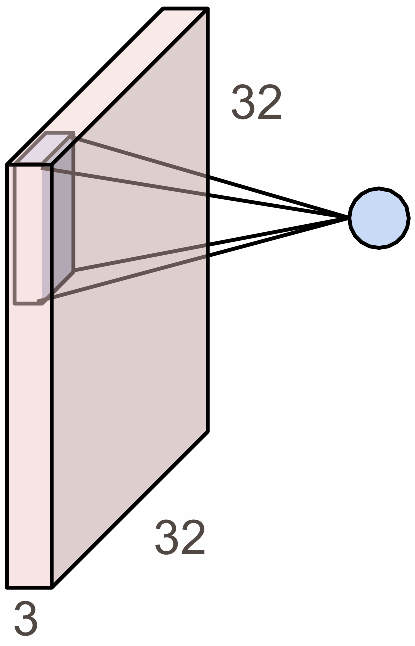
Convolution Layer



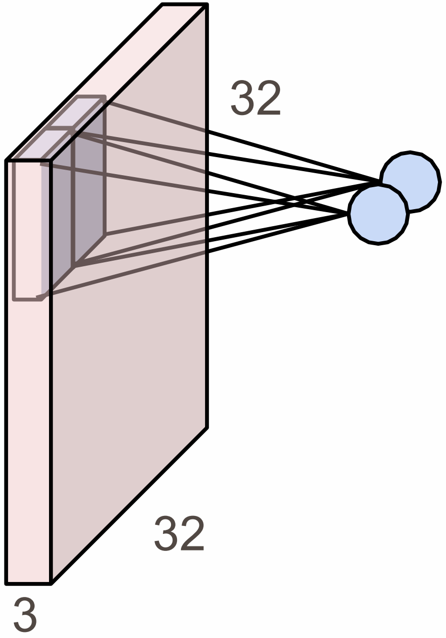
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

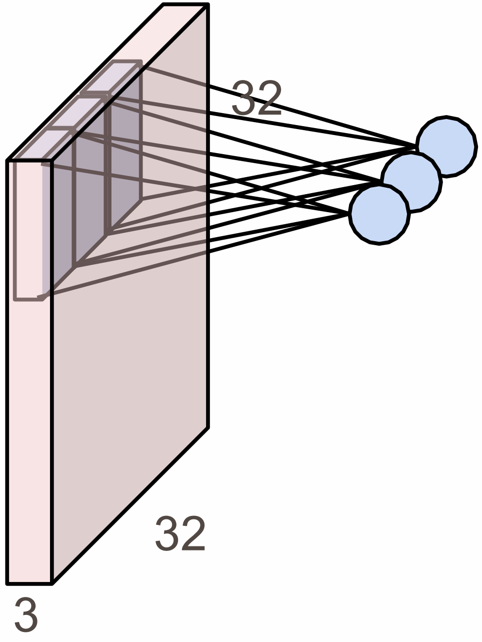
Convolution Layer



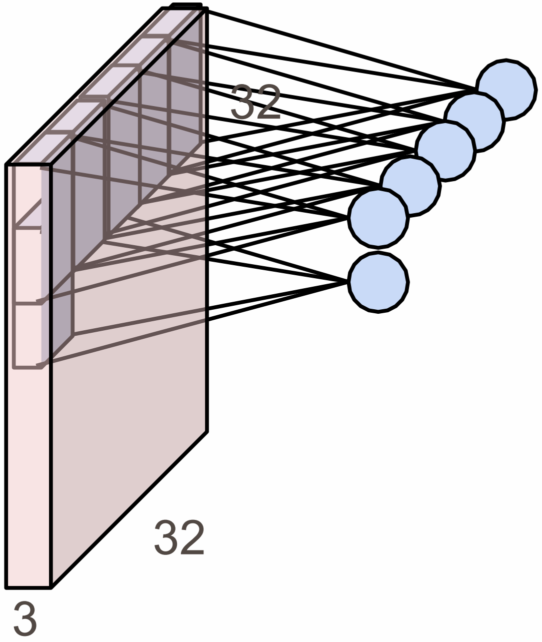
Convolution Layer



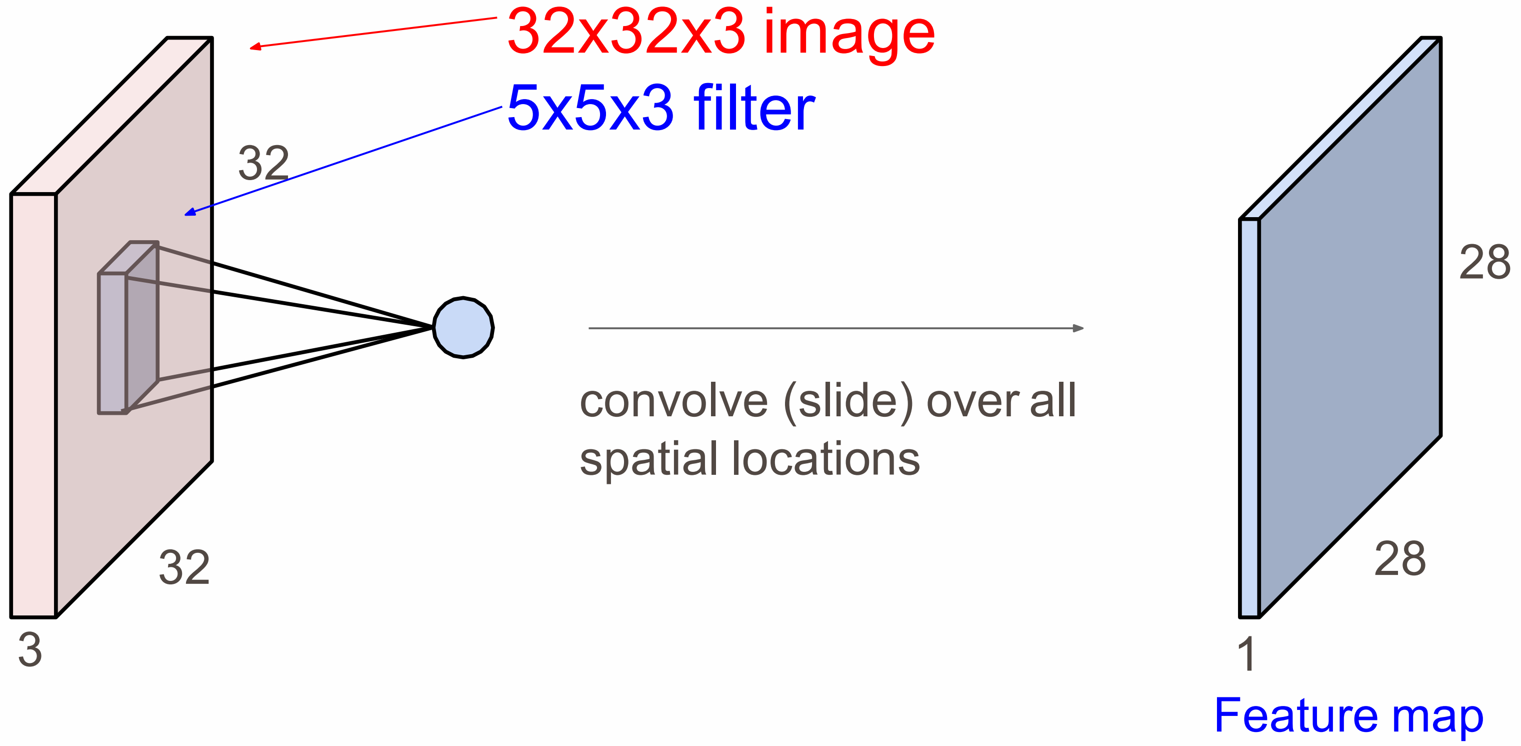
Convolution Layer



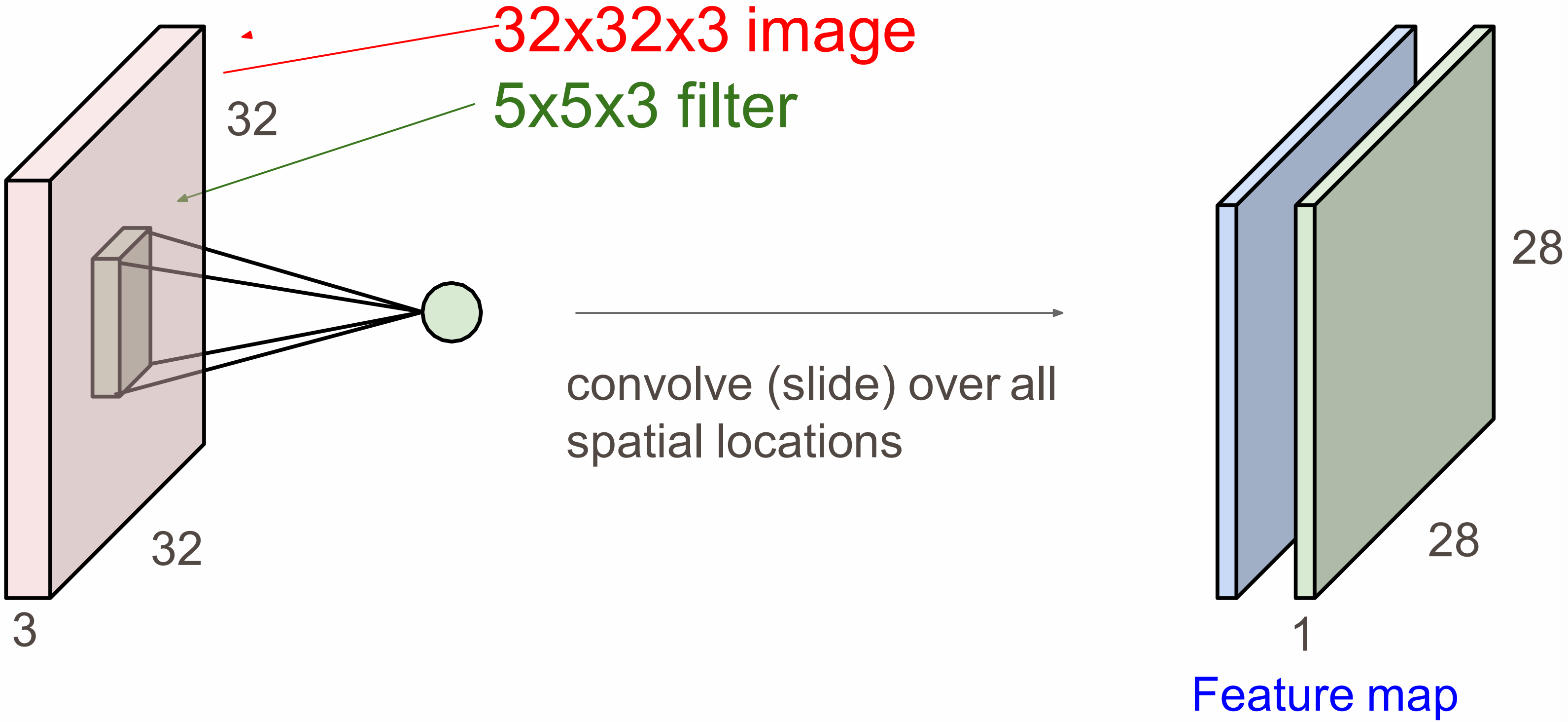
Convolution Layer



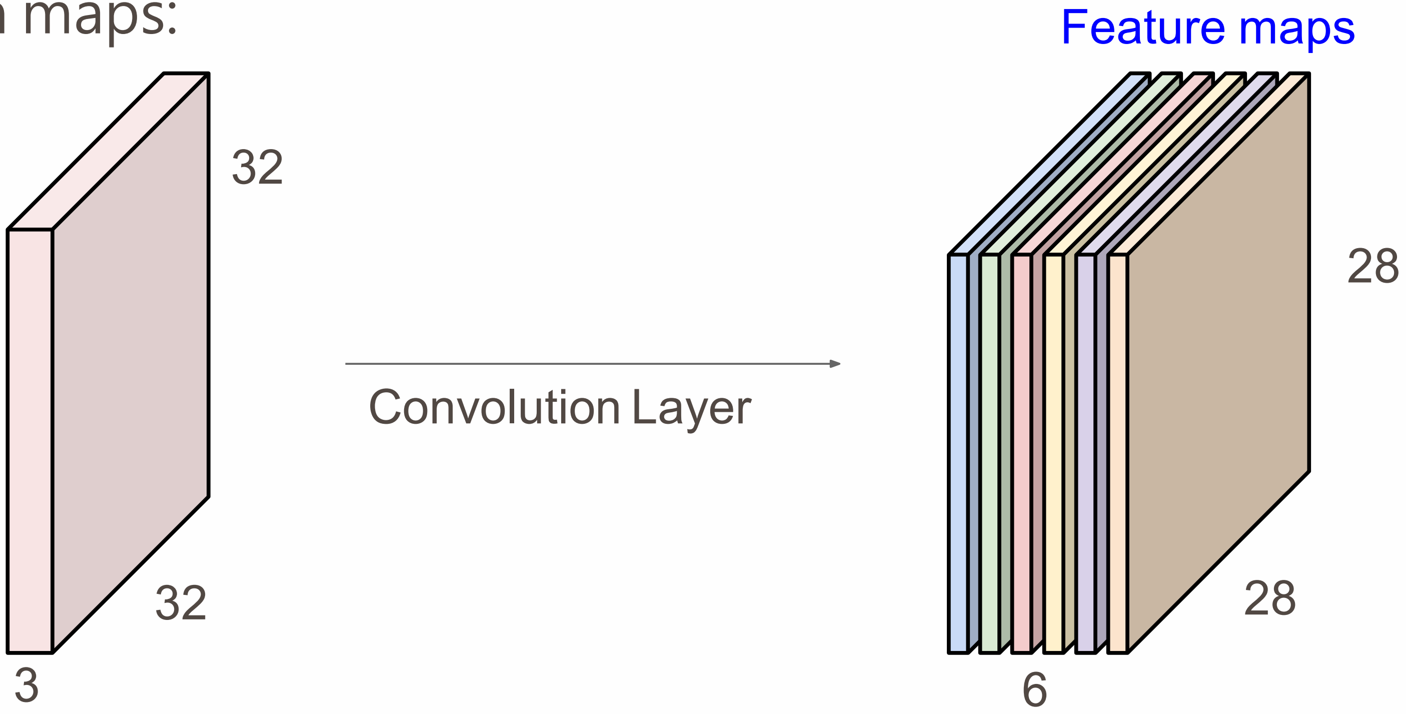
Convolution Layer



consider a second, green filter

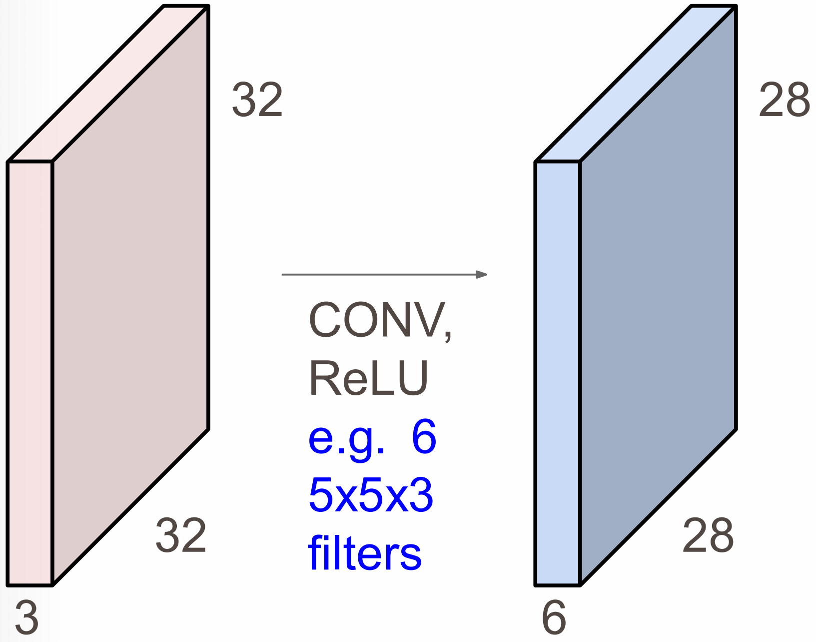


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

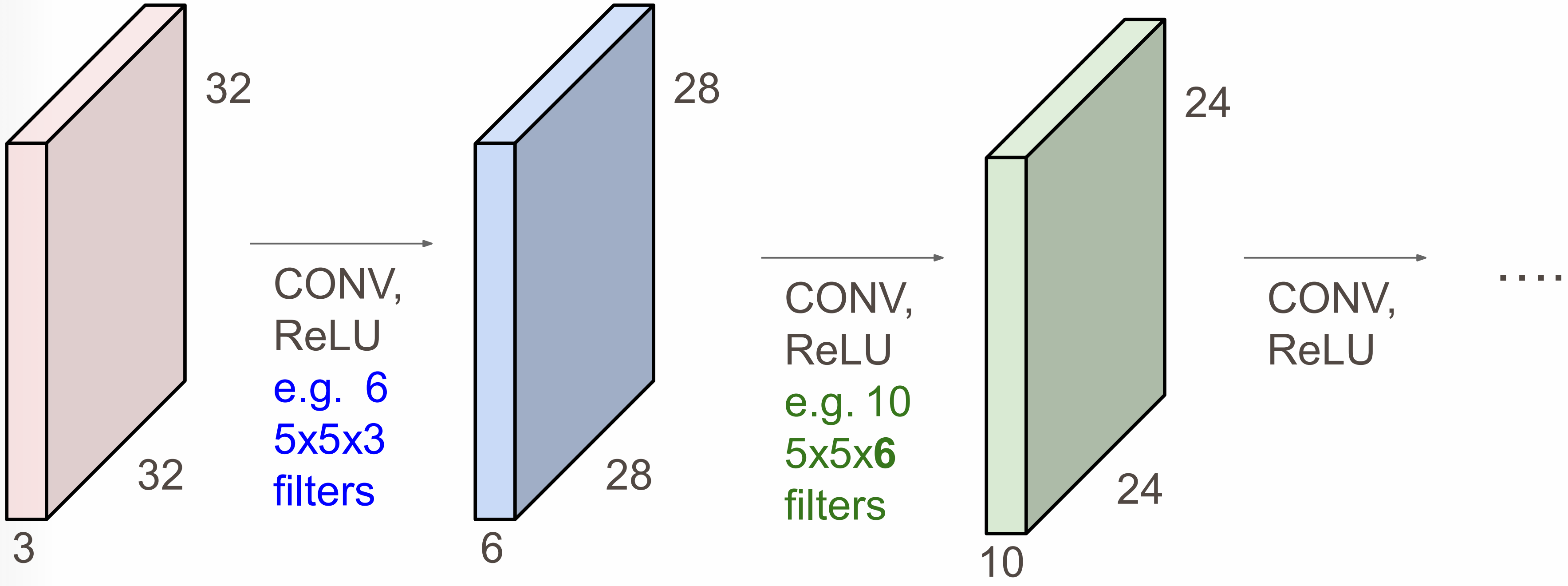


We stack these up to get a "new image" of size 28x28x6!

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



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



CNN – Convolution

Those are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Matrix

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

Matrix

⋮ ⋮

Property 1

Each filter detects a small pattern (3 x 3).

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3 -1

CNN – Convolution

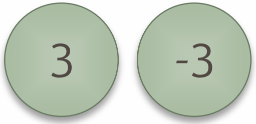
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



We set stride=1 below

CNN – Convolution

1	-1	-1
-1	1	-1
-1	-1	1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Property 2

CNN – Convolution

Filter 1

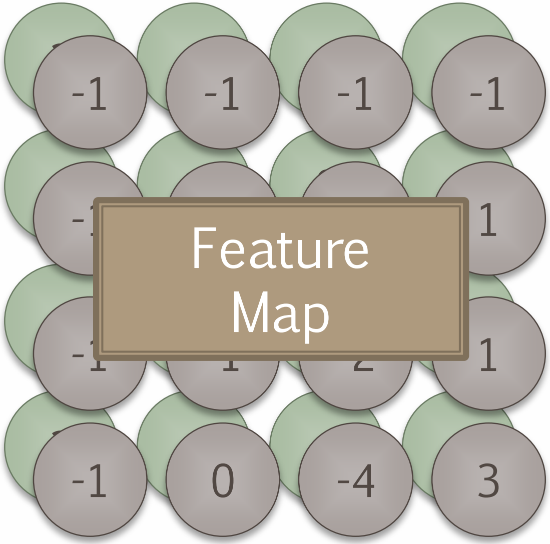
-1	1	-1
-1	1	-1
-1	1	-1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Do the same process for every filter



4 x 4 image

CNN – Colorful image

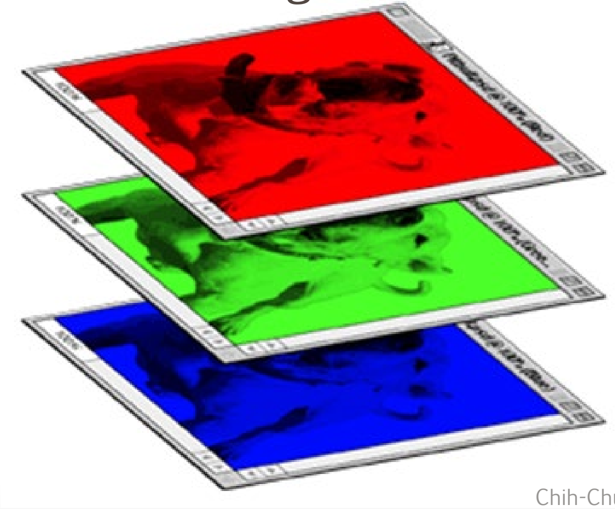
Filter 1

1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

-1	1	-1
-1	1	-1
-1	1	-1

Colorful image



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Convolution v.s. Fully Connected

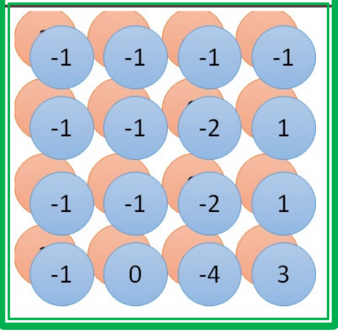
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

image

1	-1	-1
-1	1	-1
-1	-1	1

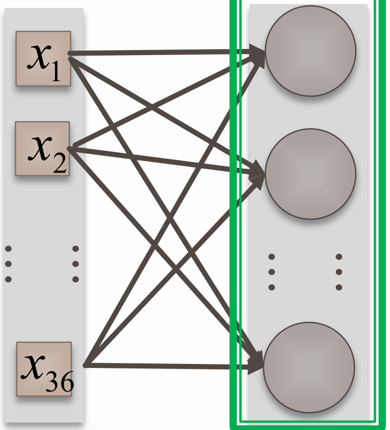
-1	1	-1
-1	1	-1
-1	1	-1

convolutio
n

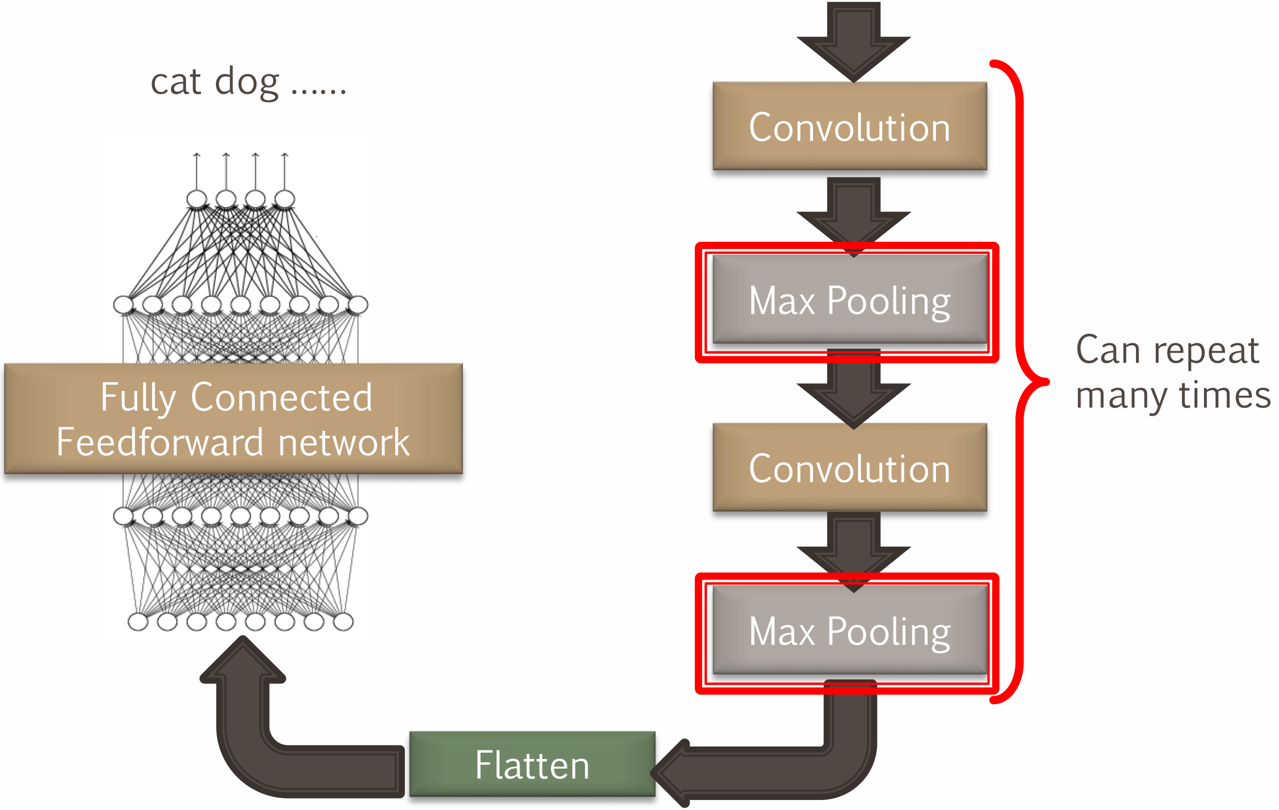


Fully-
connected

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0



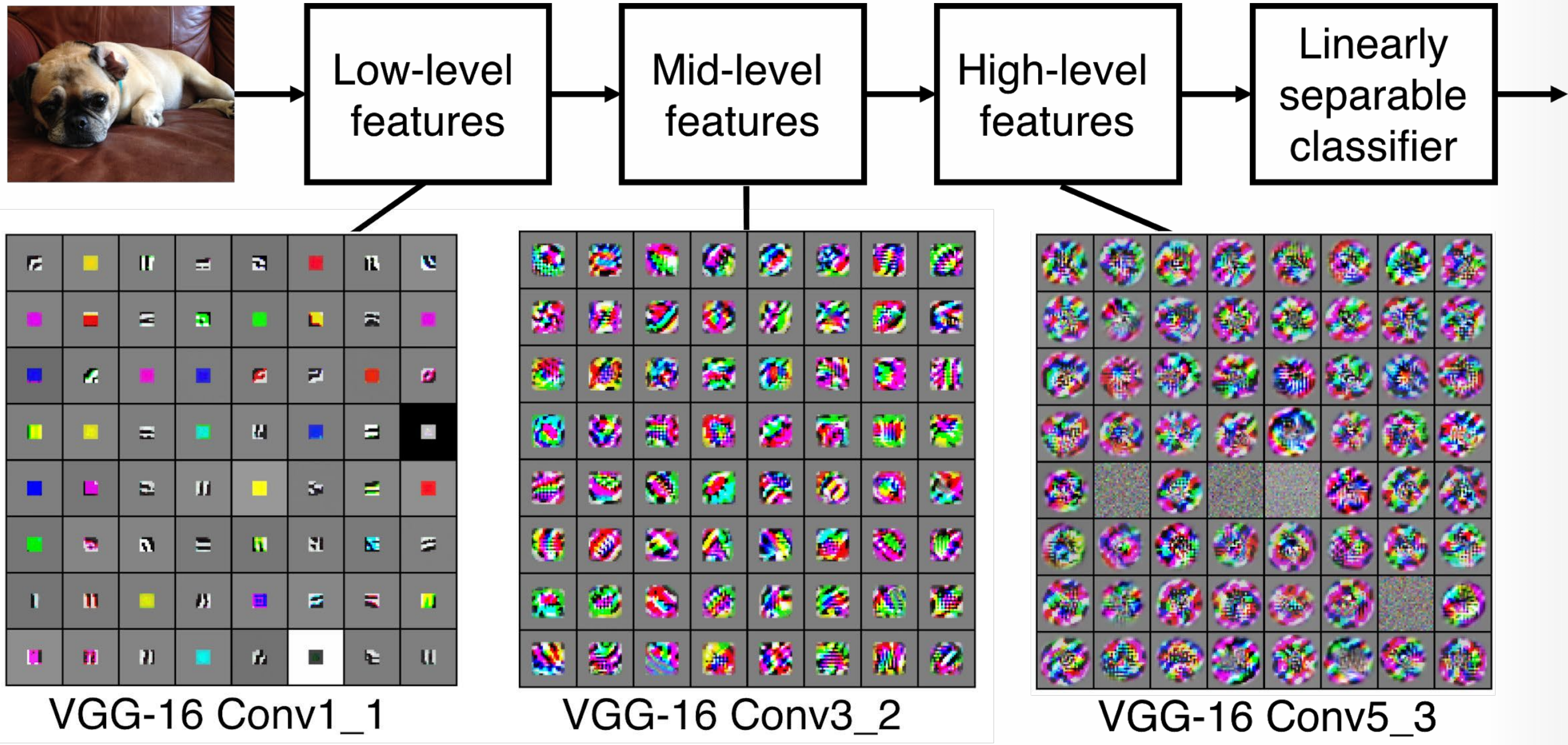
The whole CNN



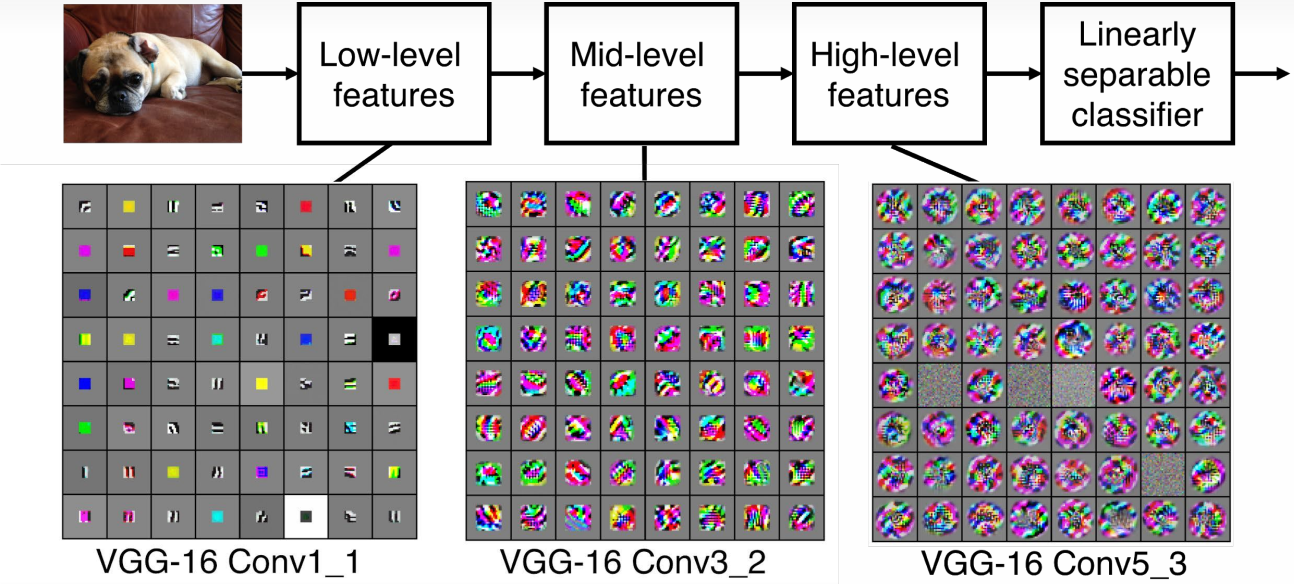
Preview

[Zeiler and Fergus 2013]

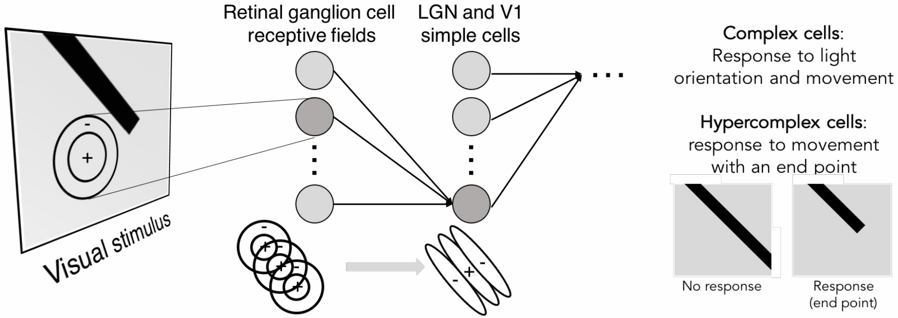
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].



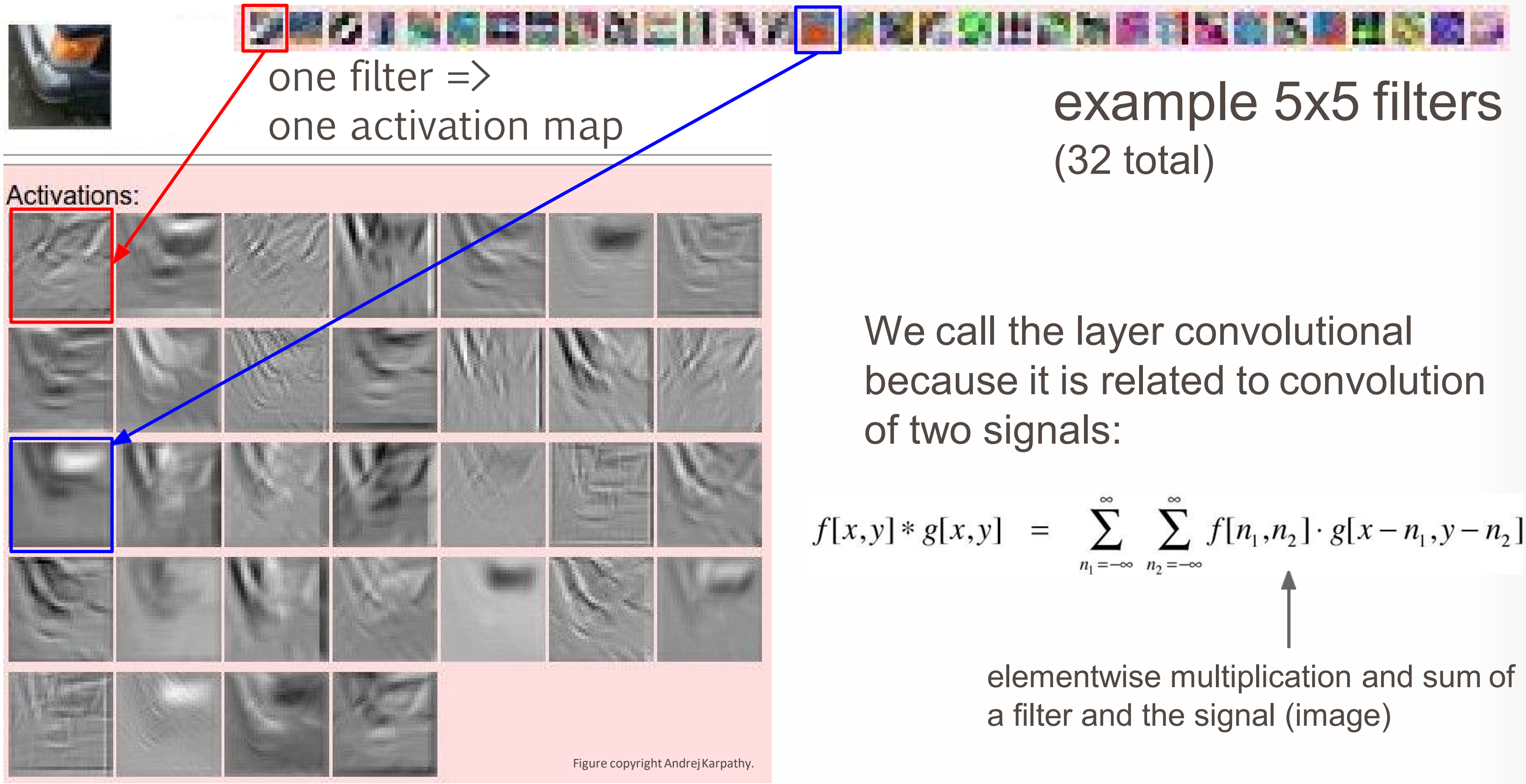
Preview



Low-level features
Mid-level features
High-level features
Linearly separable classifier
VGG-16 Conv1_1
VGG-16 Conv3_2
VGG-16 Conv5_3



Retinal ganglion cell receptive fields
LGN and V1 simple cells
Visual stimulus
Complex cells:
Response to light orientation and movement
Hypercomplex cells:
response to movement with an end point
No response
Response (end point)

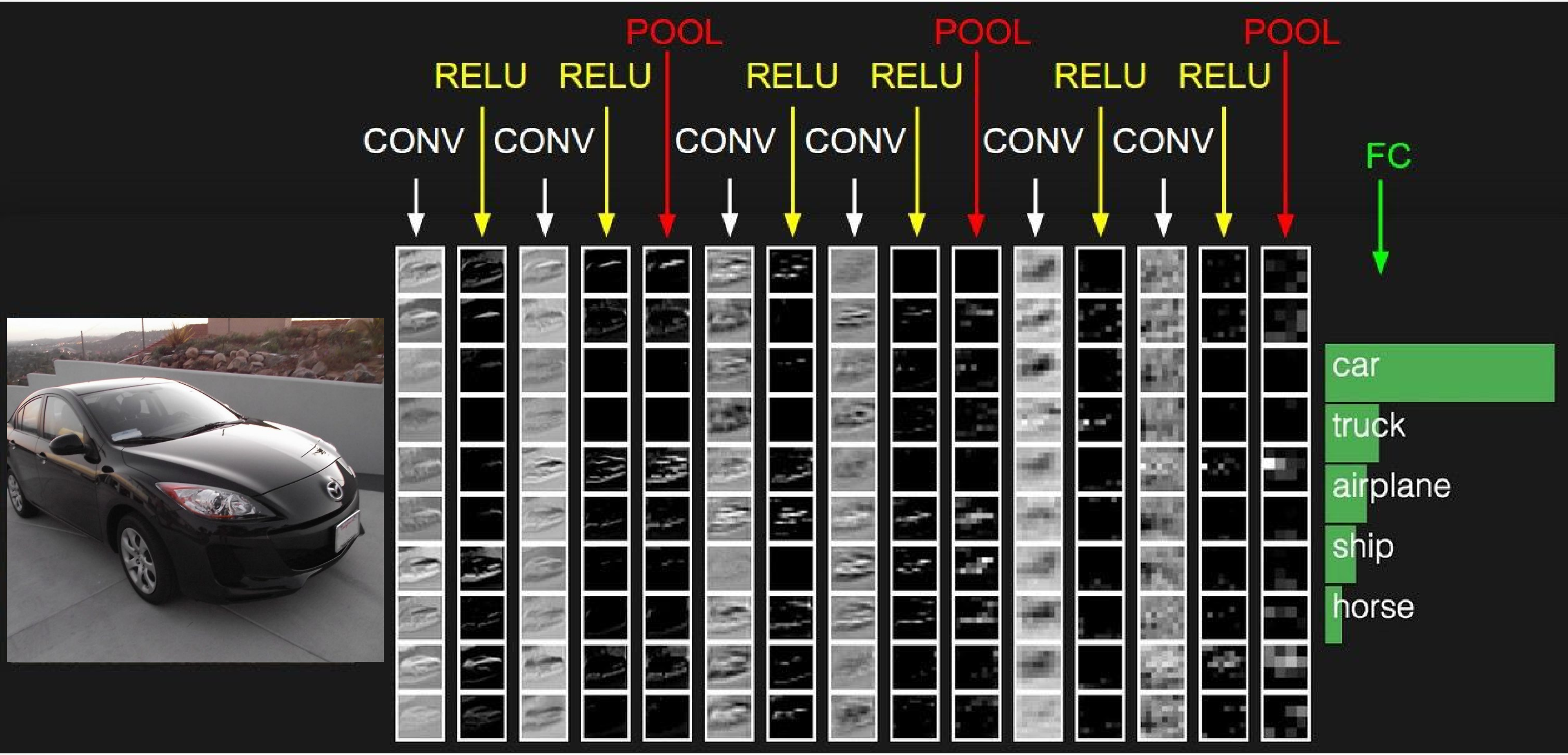


example 5x5 filters
(32 total)

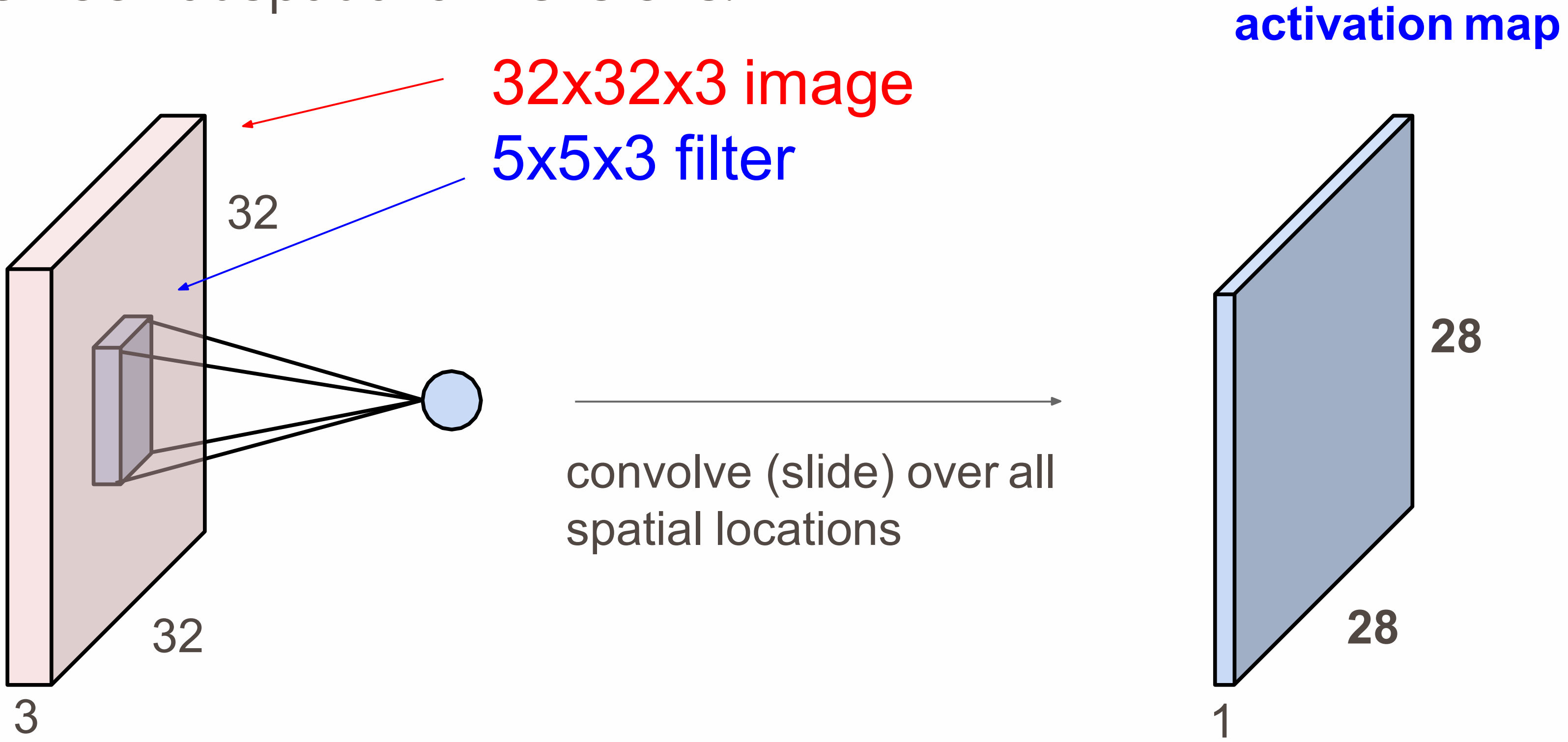
We call the layer convolutional because it is related to convolution of two signals:

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

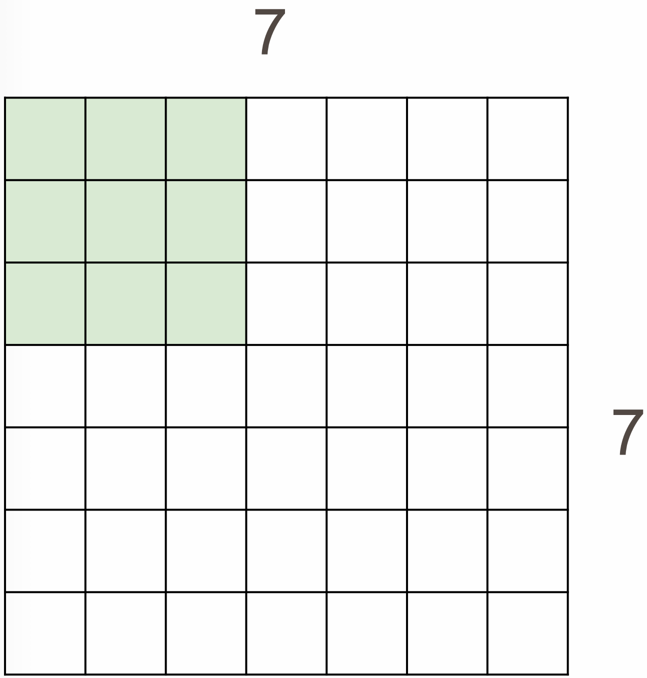
elementwise multiplication and sum of a filter and the signal (image) preview:


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:

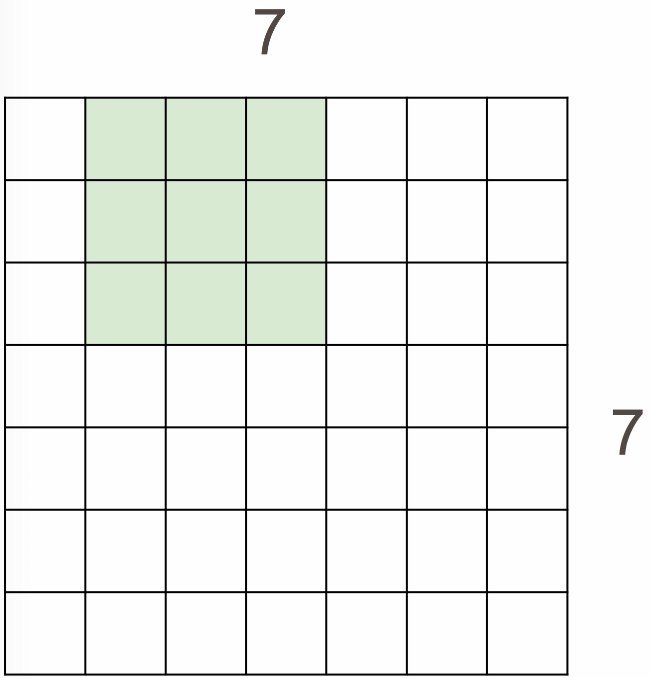


A closer look at spatial dimensions:



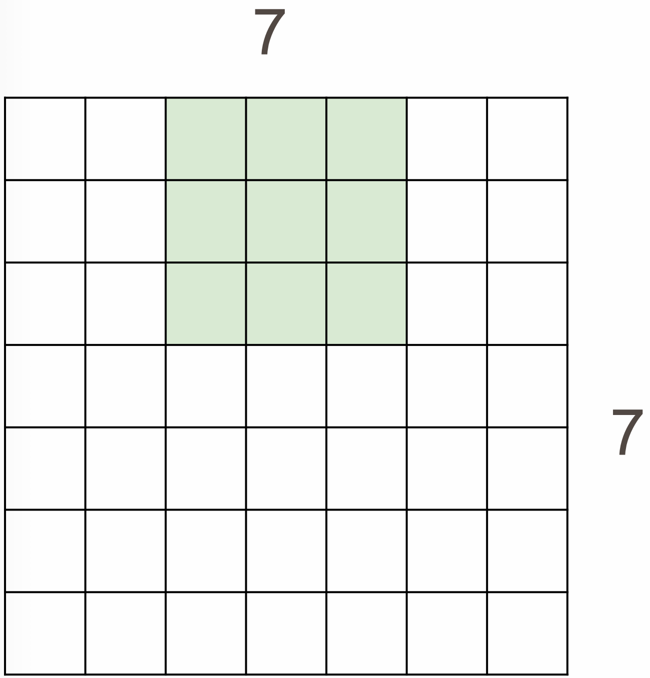
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



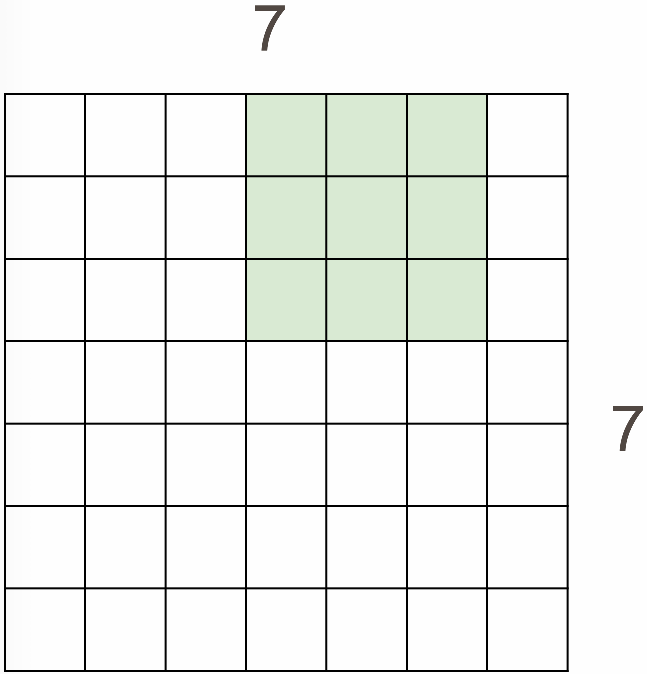
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



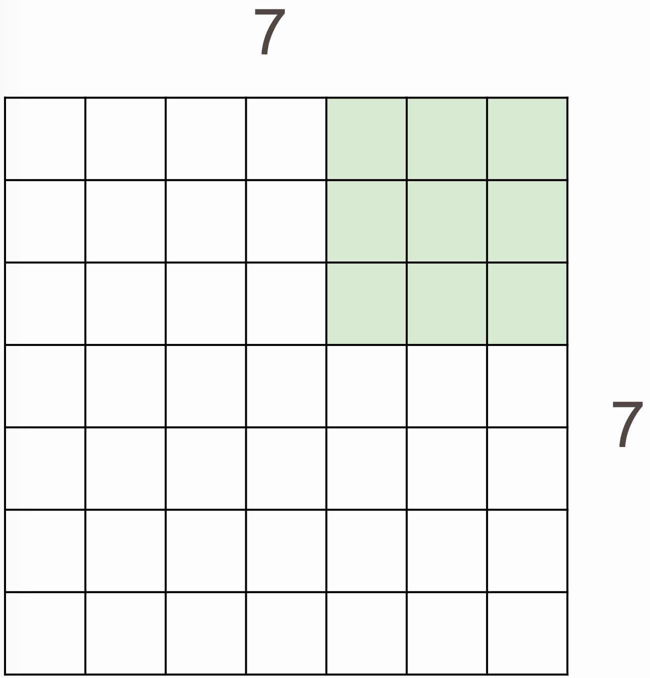
7x7 input (spatially)
assume 3x3 filter

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

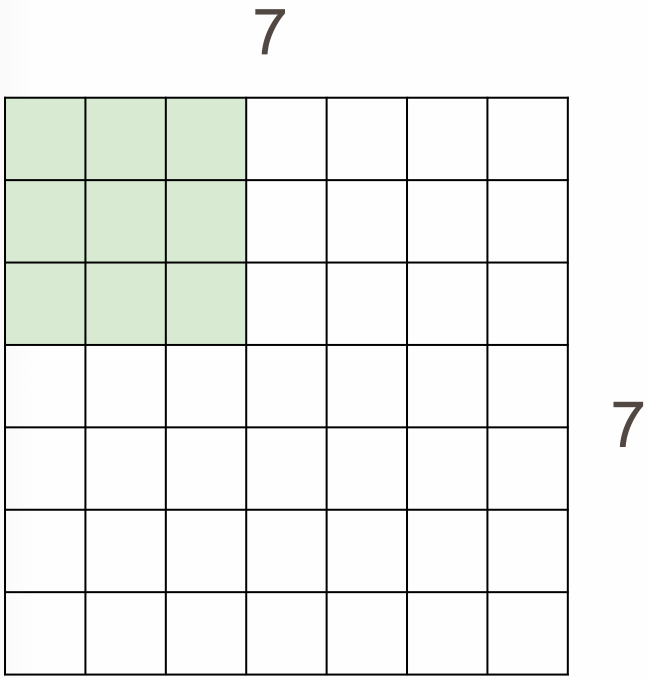
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

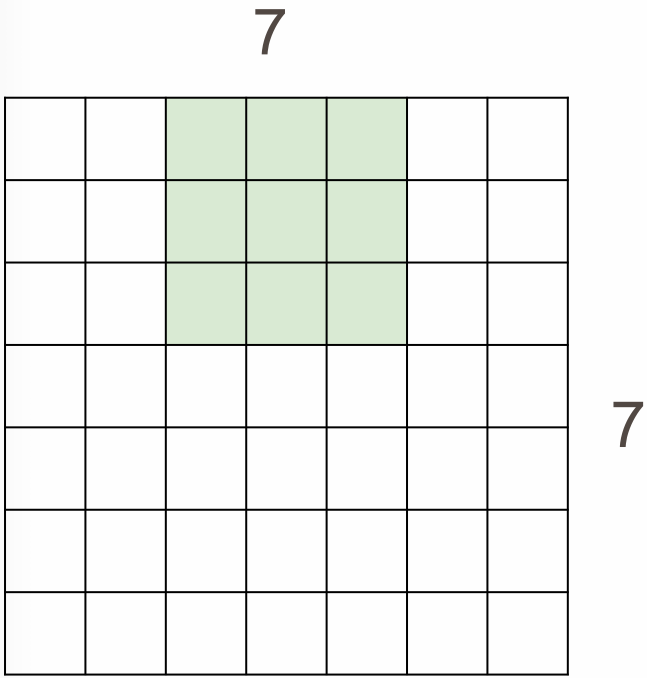
=> 5x5 output

A closer look at spatial dimensions:



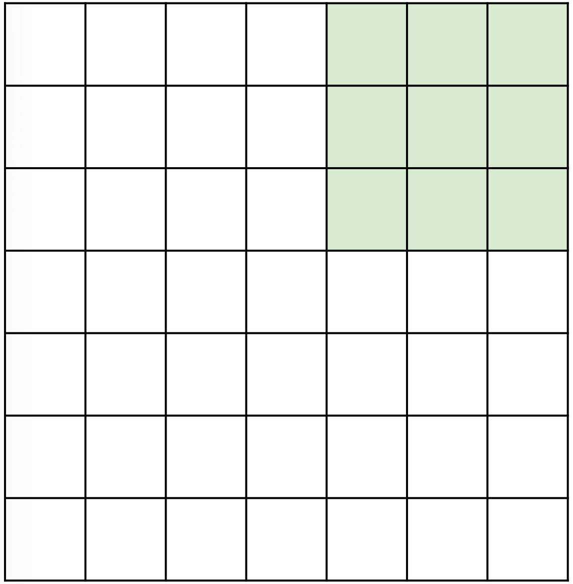
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



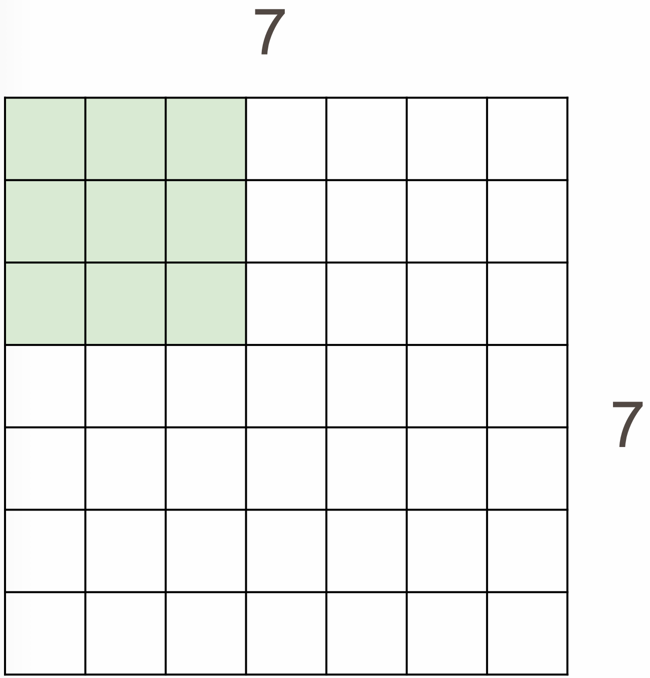
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



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

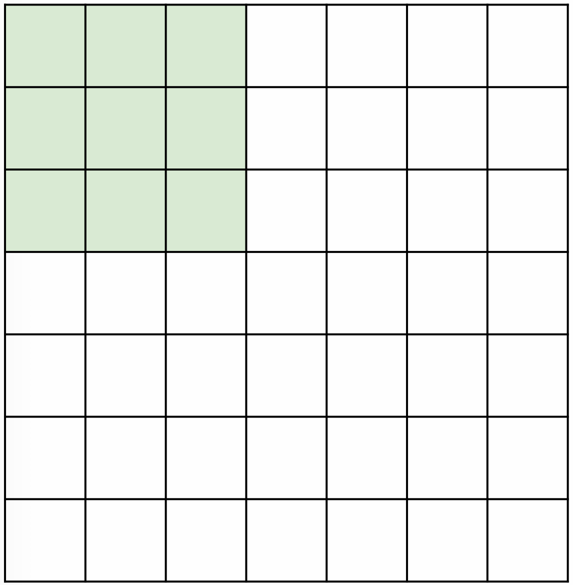
A closer look at spatial dimensions:



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

A closer look at spatial dimensions:

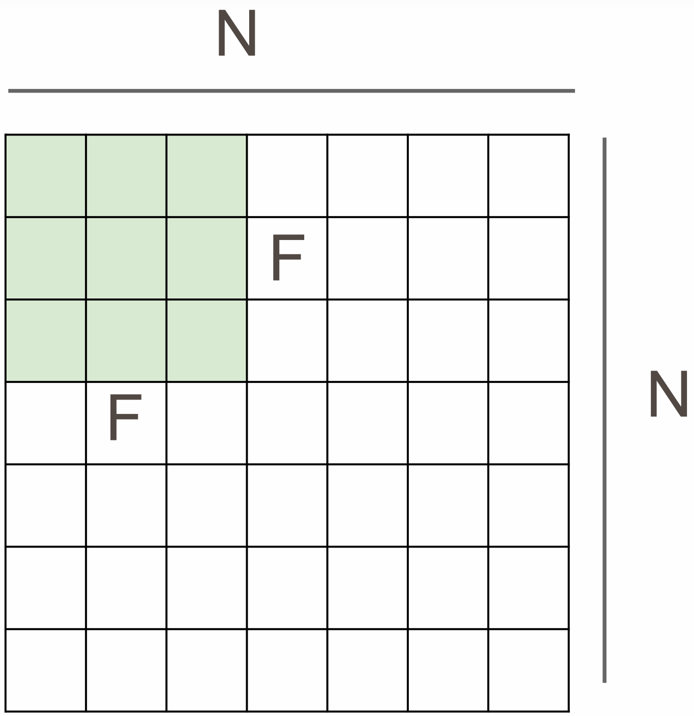
7



7

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

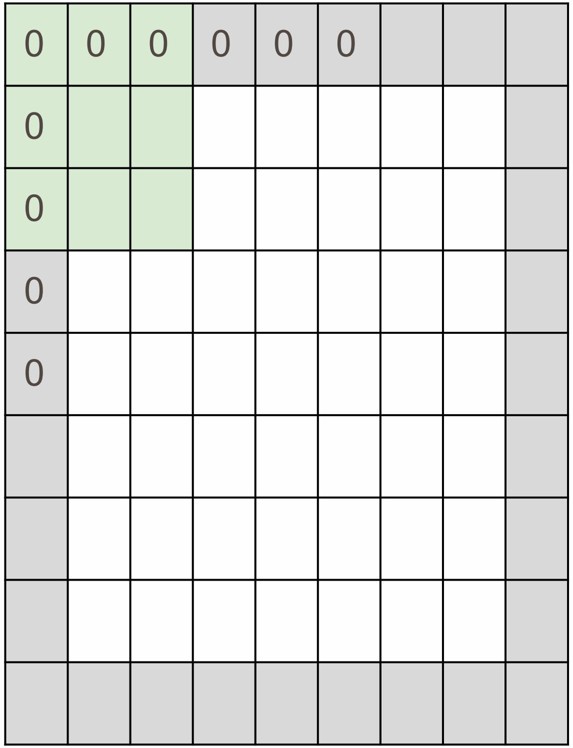
doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



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 :\

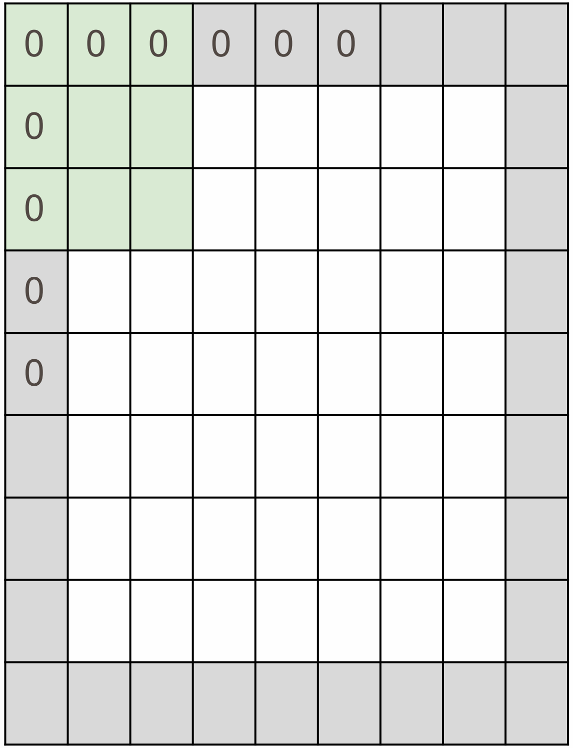
In practice: Common to zero pad the border



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

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

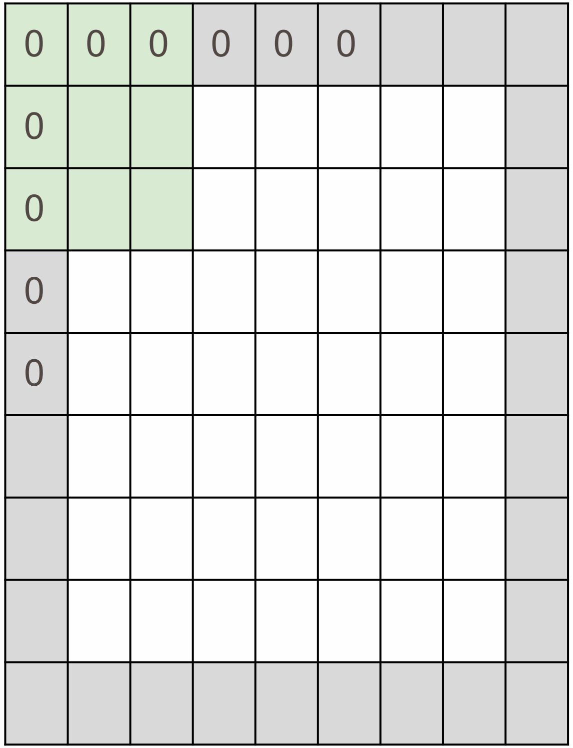
In practice: Common to zero pad the border



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

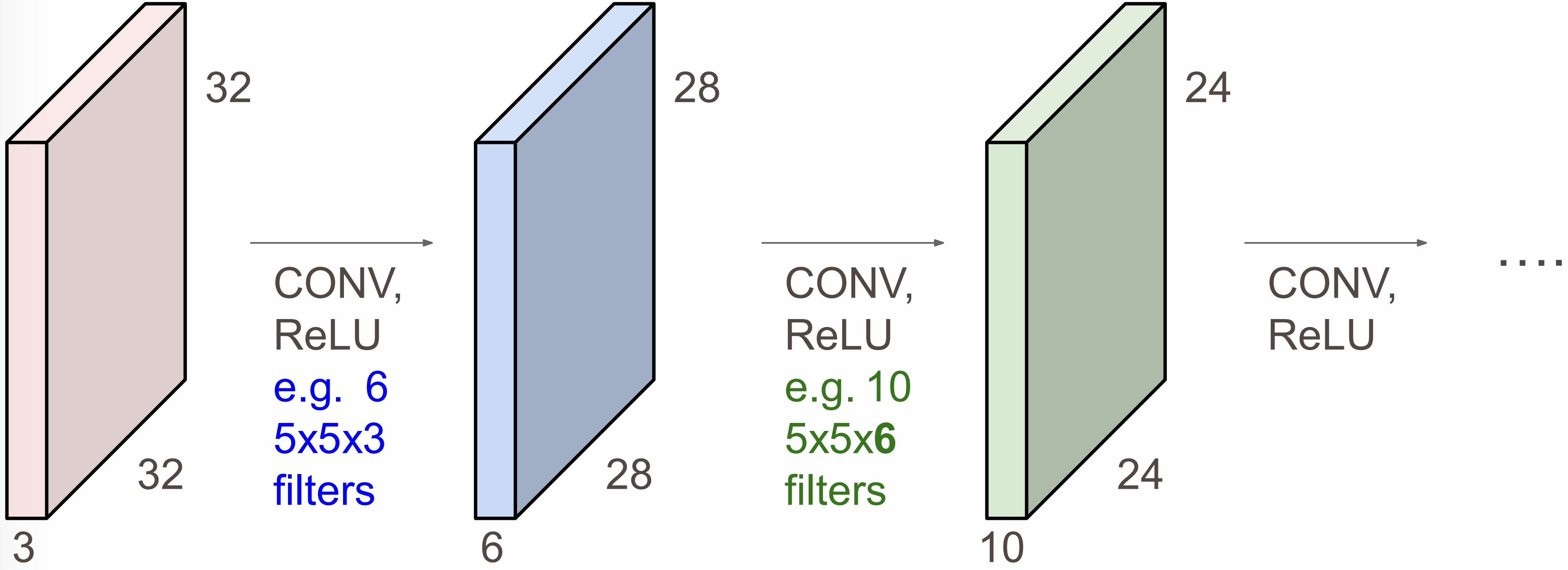


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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...

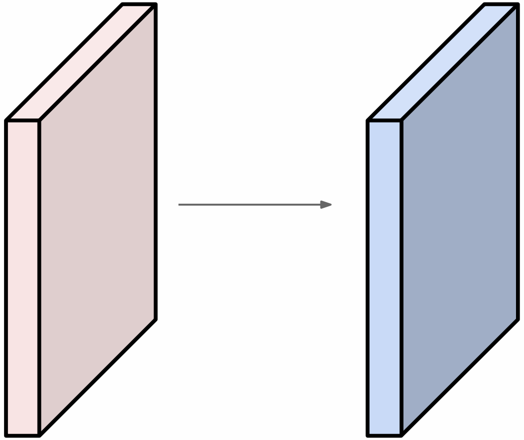
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Examples time

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

Output volume size: ?



Examples time

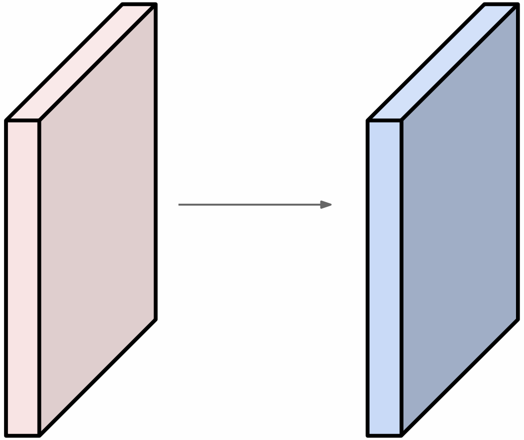
Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size:

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

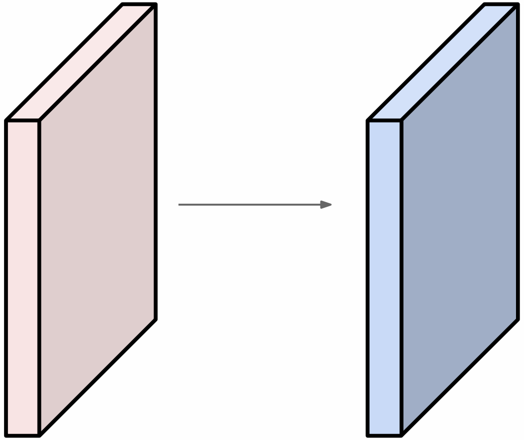
32x32x10



Examples time

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

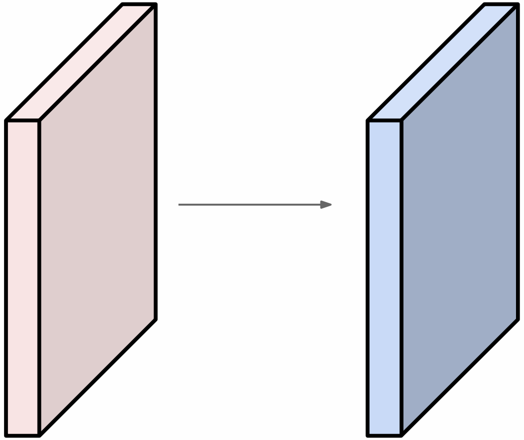
Number of parameters in this layer?



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
=> 76*10 = **760** (+1 for bias)

Convolution layer: summary

Let's assume input is W_1 x H_1 x C
Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of W_2 x H_2 x K
where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

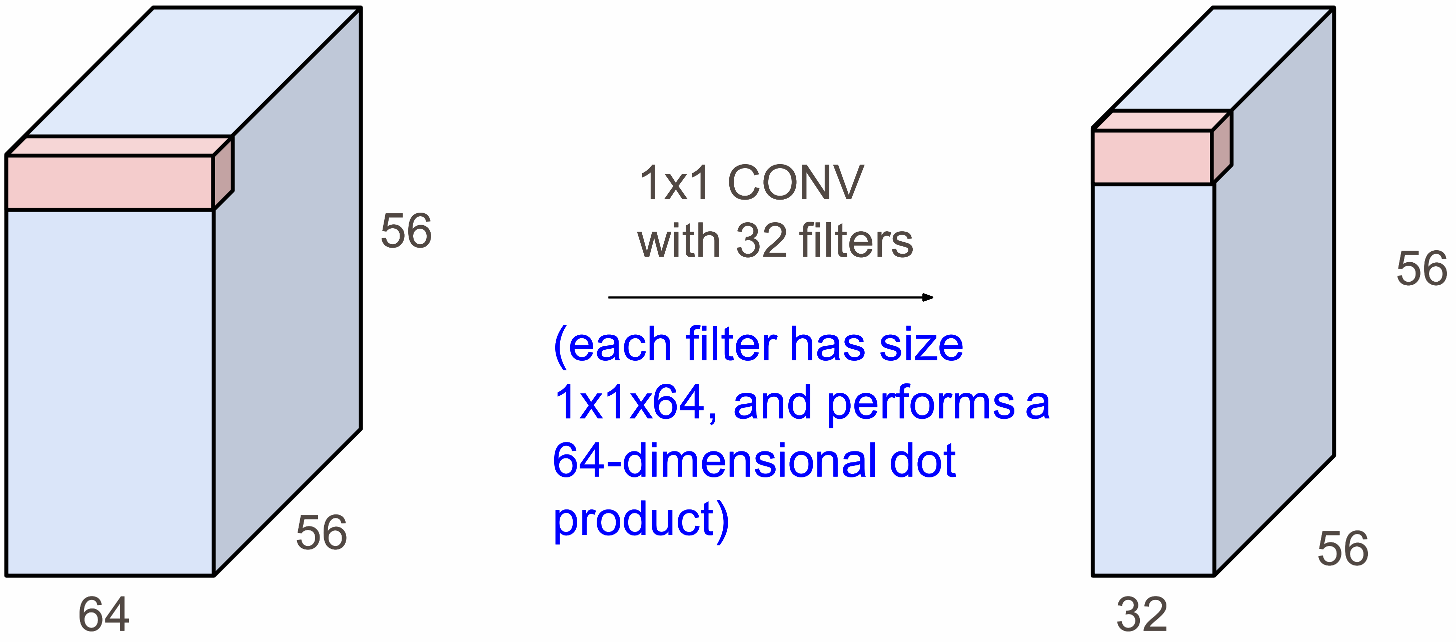
Number of parameters: F^2CK and K biases

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

(btw, 1x1 convolution layers make perfect sense)



Example: CONV layer in PyTorch

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

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.

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).

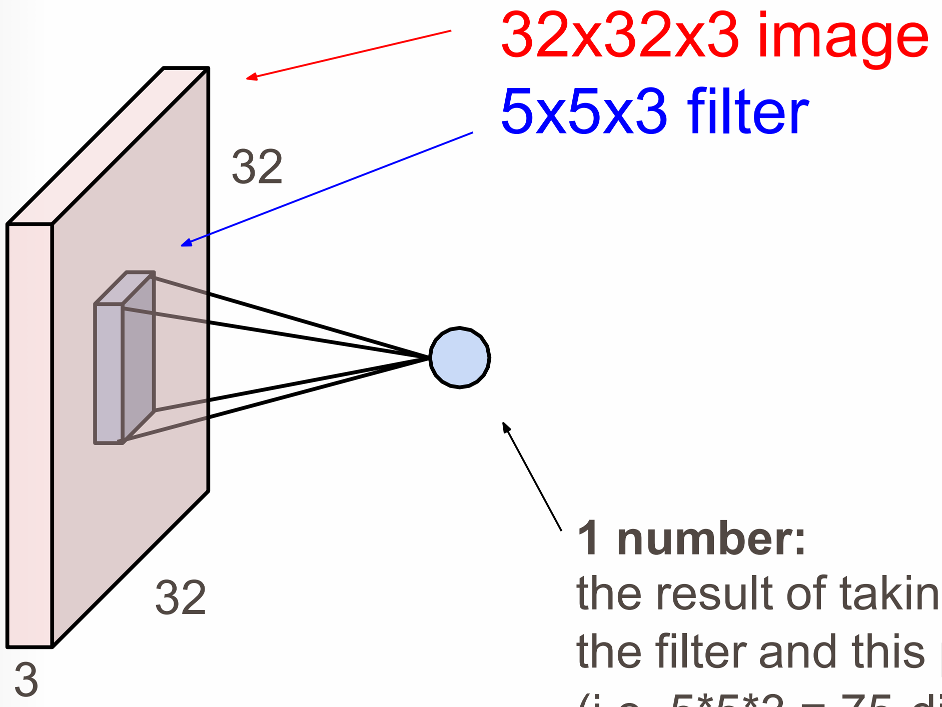
This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

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

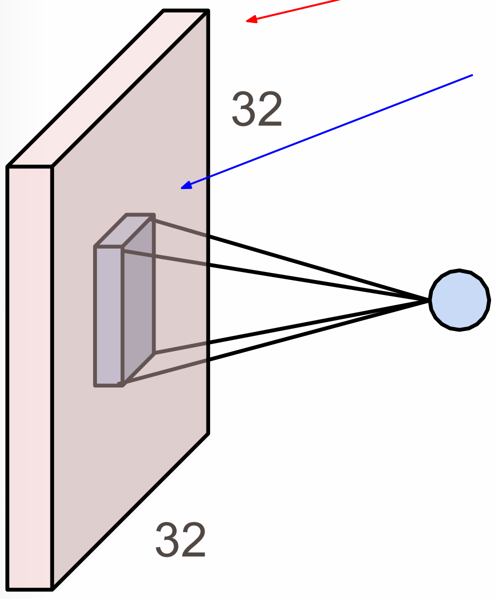


1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)

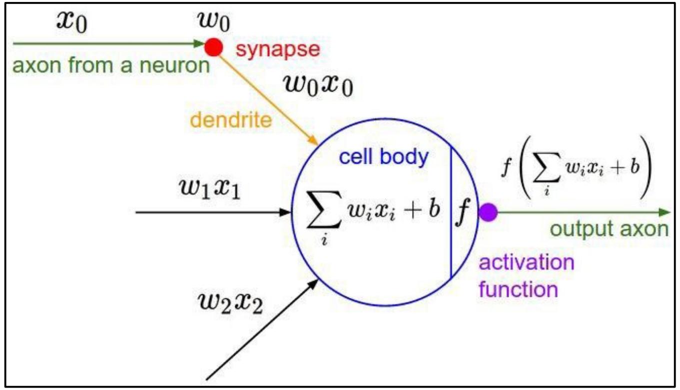
The brain/neuron view of CONV Layer

32x32x3 image

5x5x3 filter

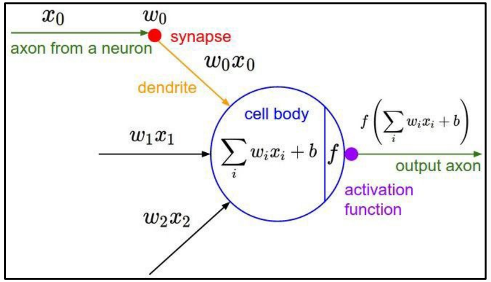


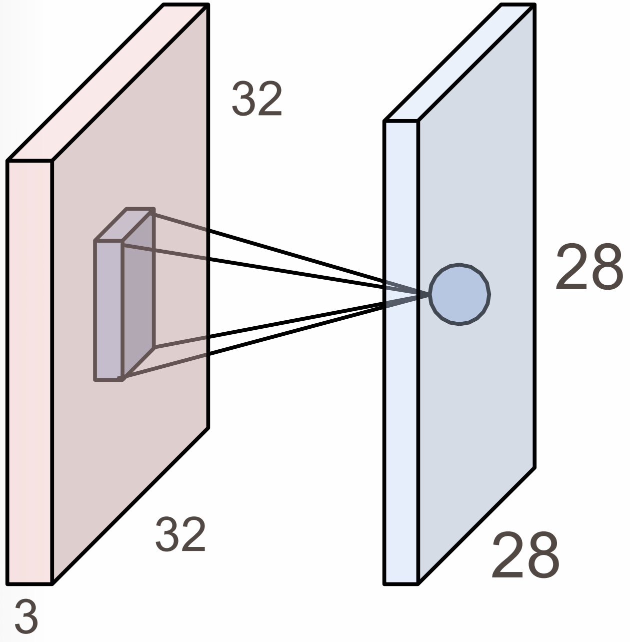
1 number:
the result of taking a dot product between the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)



It's just a neuron with local connectivity...

The brain/neuron view of CONV Layer



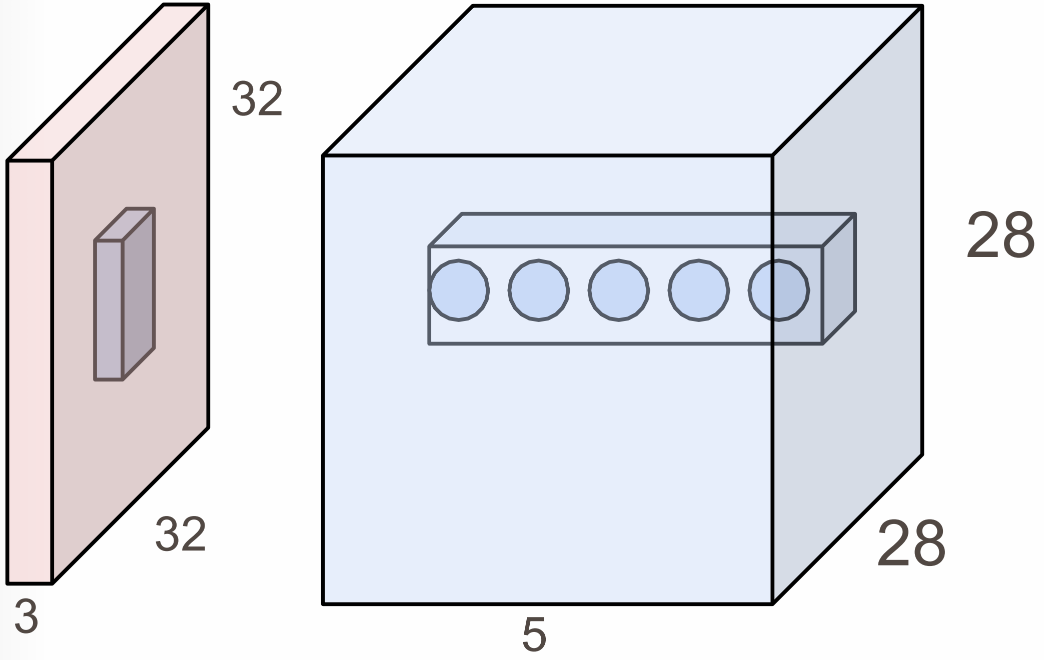


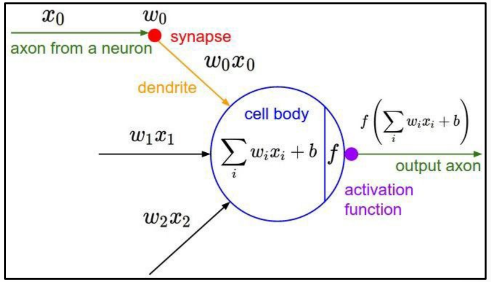
An activation map is a 28x28 sheet of neuron outputs:

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





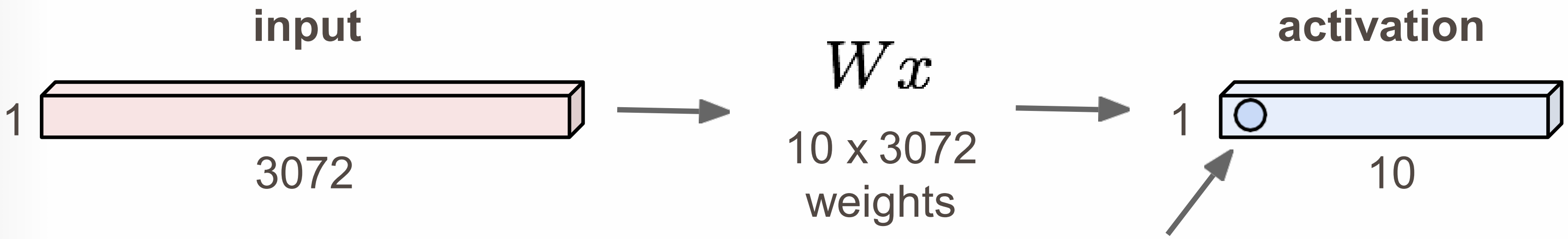
E.g. with 5 filters,
CONV layer consists of neurons arranged in a 3D grid (28x28x5)

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

Reminder: Fully Connected Layer

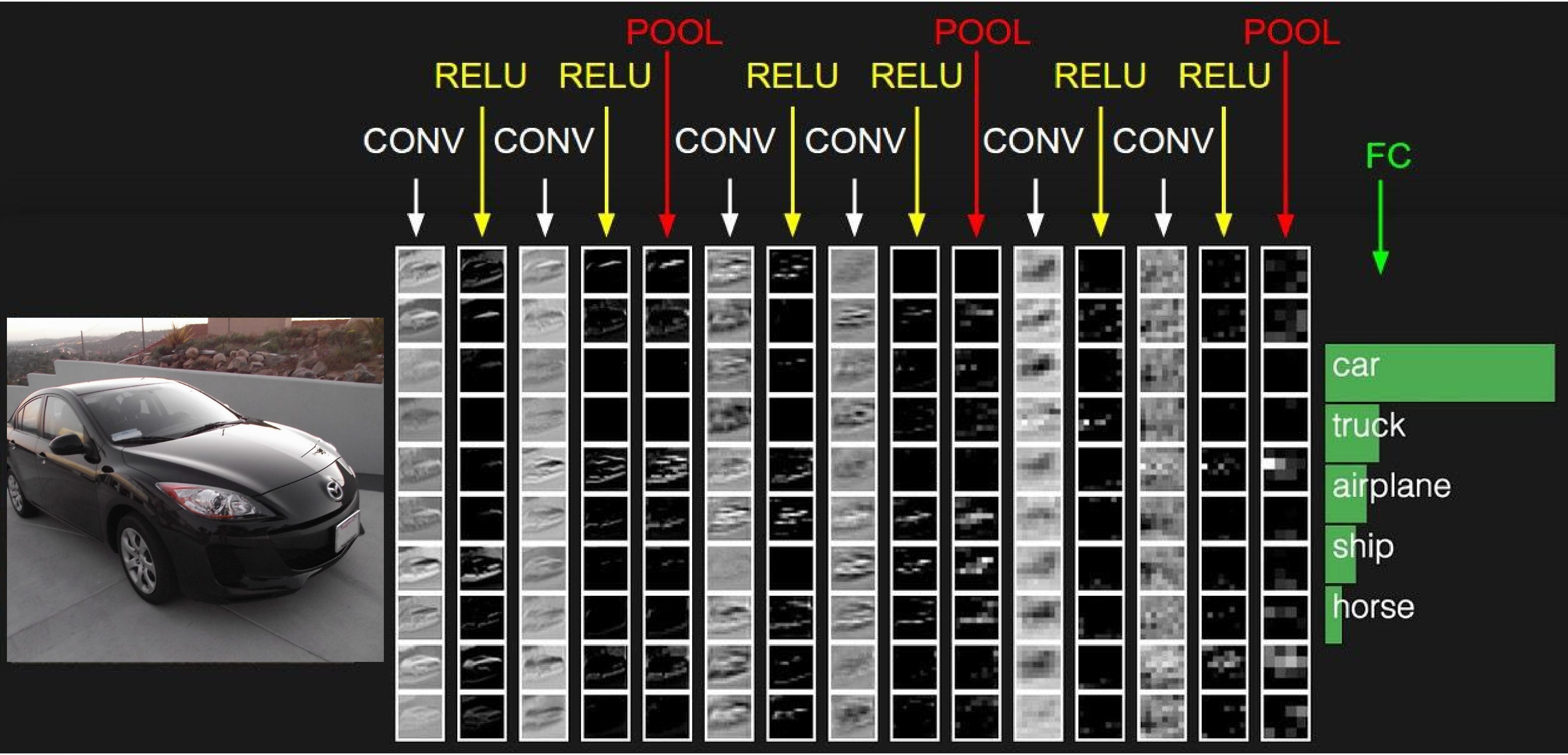
32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume



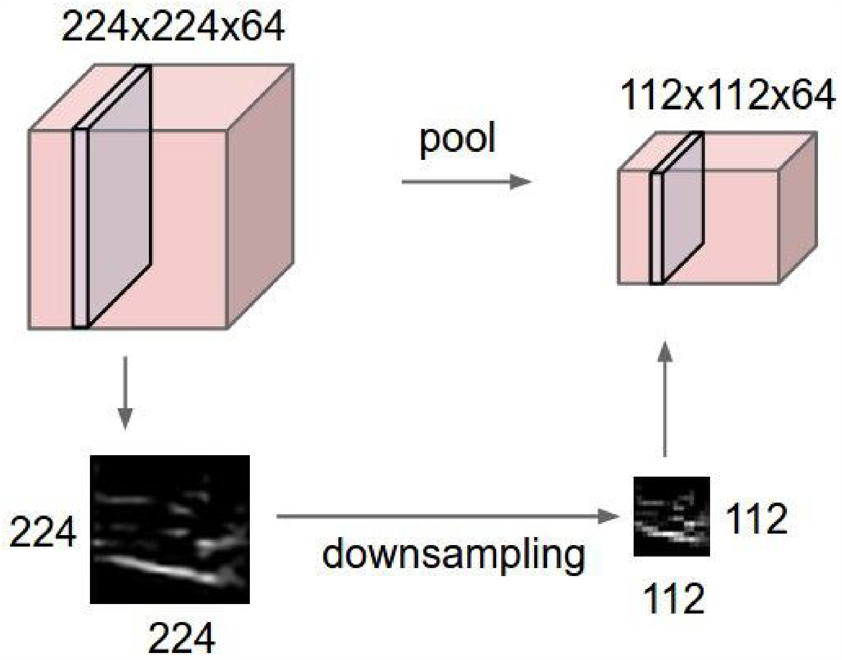
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

two more layers to go: POOL/FC

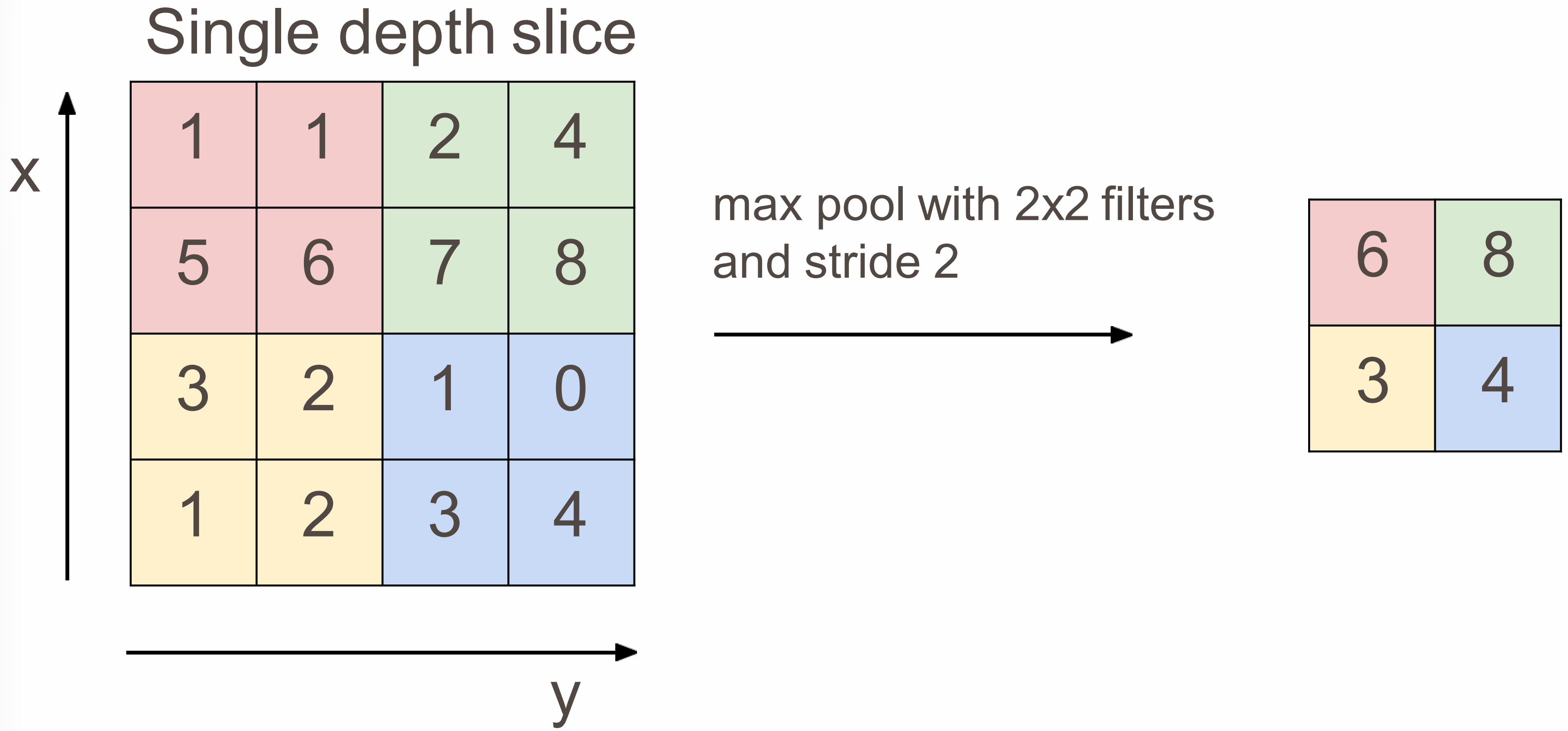


Pooling layer

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



MAX POOLING



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

Parameters and convolutions

Convolution layer: summary

Let's assume input is W_1 x H_1 x C
Conv layer needs 2 hyperparameters:

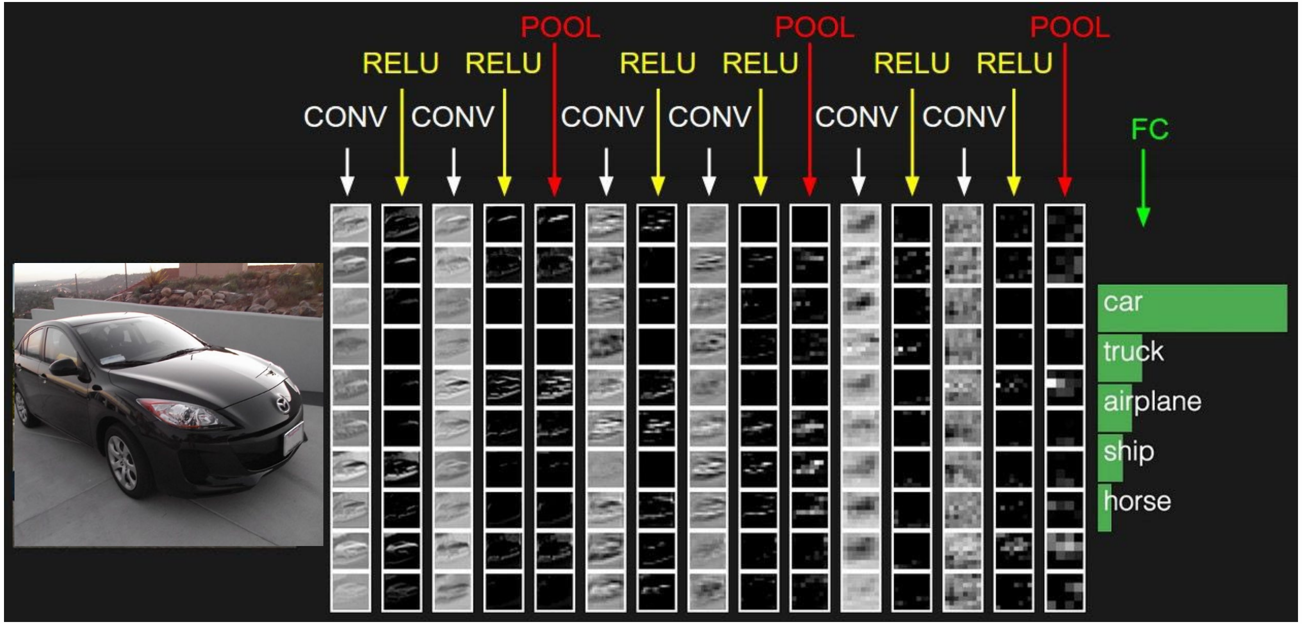
- The spatial extent **F**
- The stride **S**

This will produce an output of W_2 x H_2 x C where:

- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0

Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



[ConvNetJS demo: training on CIFAR-10]

ConvNetJS CIFAR-10 demo

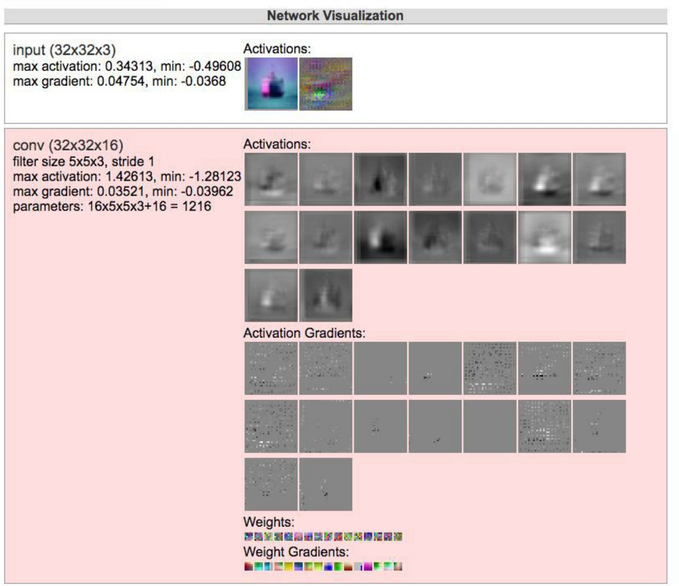
Description

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.

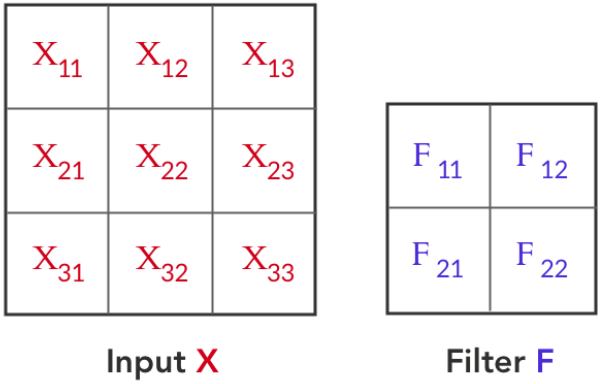
Report questions/bugs/suggestions to @karpathy.



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

Backpropagation

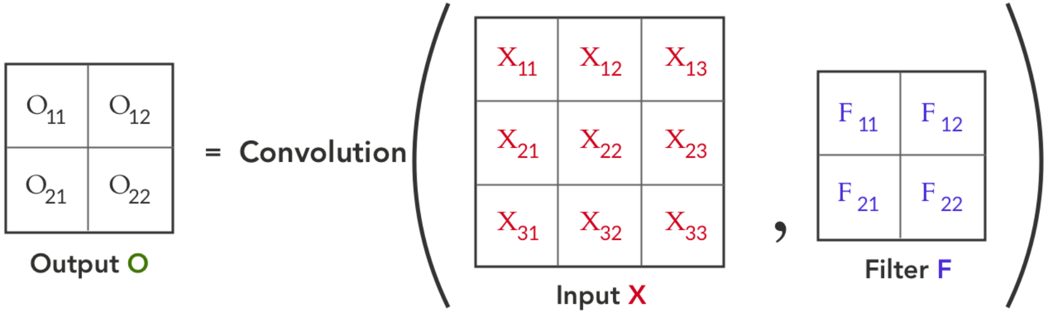
- Now, lets assume the function ***f is a convolution** between* **Input X and a Filter F.** Input X is a 3x3 matrix and Filter F is a 2x2 matrix, as shown below:



Ref: https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

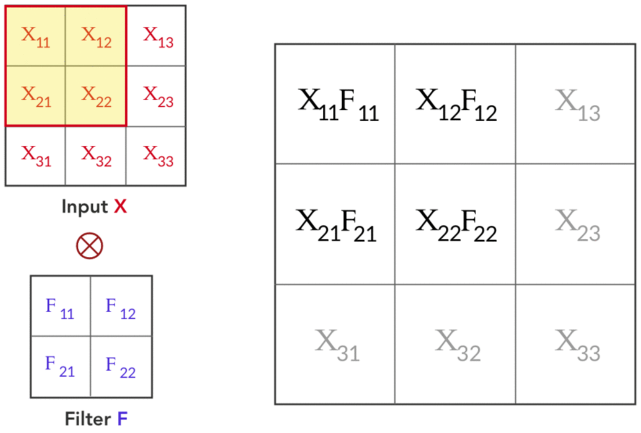
Backpropagation

- Convolution between Input X and Filter F, gives us an output O. This can be represented as



Ref: https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

Backpropagation (Forward part)



$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Ref: https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

Backpropagation (Forward part)

Local Gradients → A

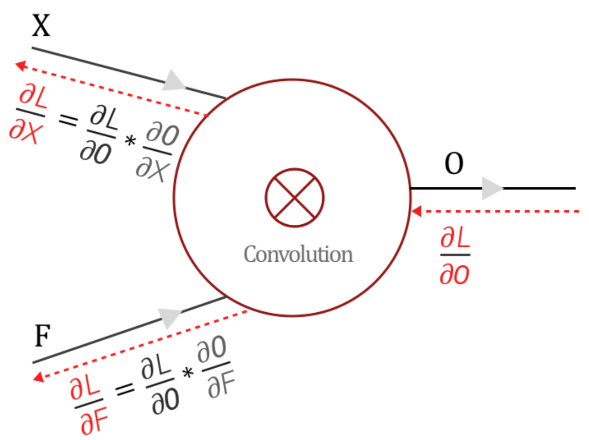
$$O_{11} = X_{11}F_{11} + X_{12}F_{12} + X_{21}F_{21} + X_{22}F_{22}$$

Finding derivatives with respect to F_{11} , F_{12} , F_{21} and F_{22}

$$\frac{\partial O_{11}}{\partial F_{11}} = X_{11} \quad \frac{\partial O_{11}}{\partial F_{12}} = X_{12} \quad \frac{\partial O_{11}}{\partial F_{21}} = X_{21} \quad \frac{\partial O_{11}}{\partial F_{22}} = X_{22}$$

Similarly, we can find the local gradients for O_{12} , O_{21} and O_{22}

- So similar processes can be applied in anywhere



$\frac{\partial O}{\partial X}$ & $\frac{\partial O}{\partial F}$ *are local gradients*

$\frac{\partial L}{\partial z}$ *is the loss from the previous layer which has to be backpropagated to other layers*

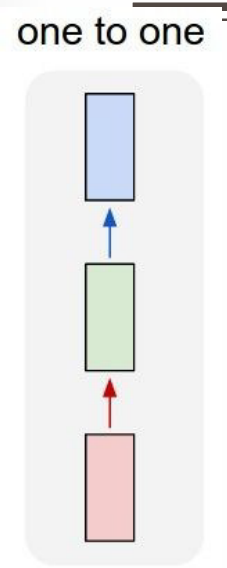
Ref: https://medium.com/@pavisj/convolutions-and-backpropagations-46026a8f5d2c

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like
  - [(CONV-RELU)*N-POOL]*M-(FC-RELU)*K,SOFTMAX
  - where N is usually up to ~5, M is large, 0 <= K <= 2.
  - - but recent advances such as ResNet/GoogLeNet have challenged this paradigm

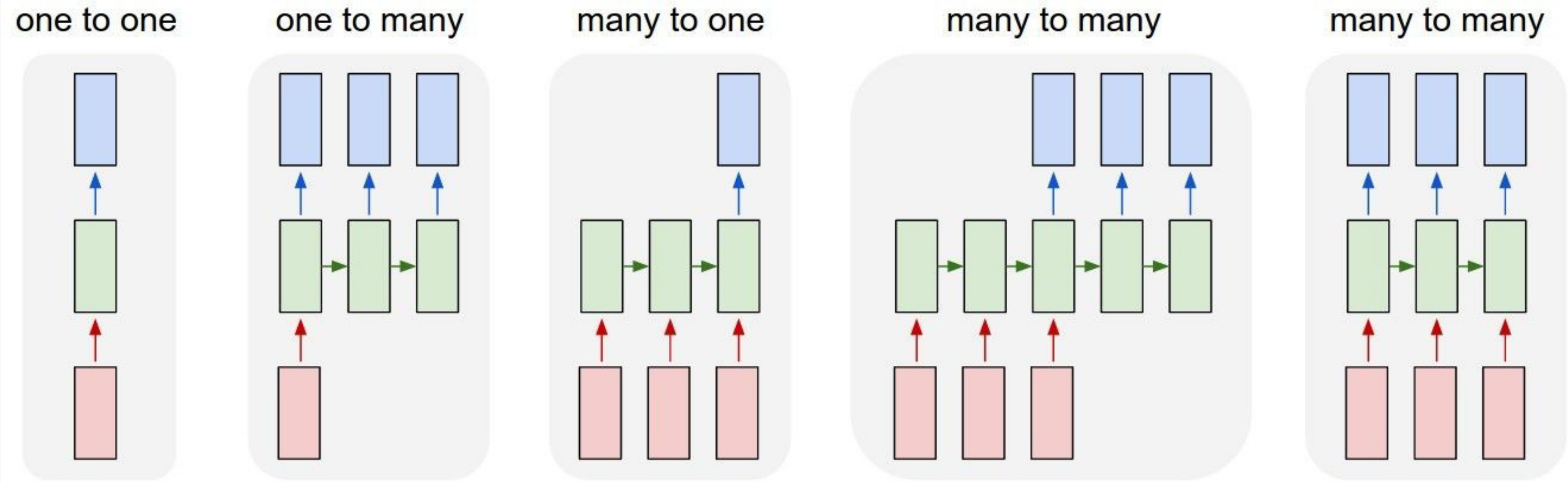
RECURRENT NETWORKS

"Vanilla" Neural Network



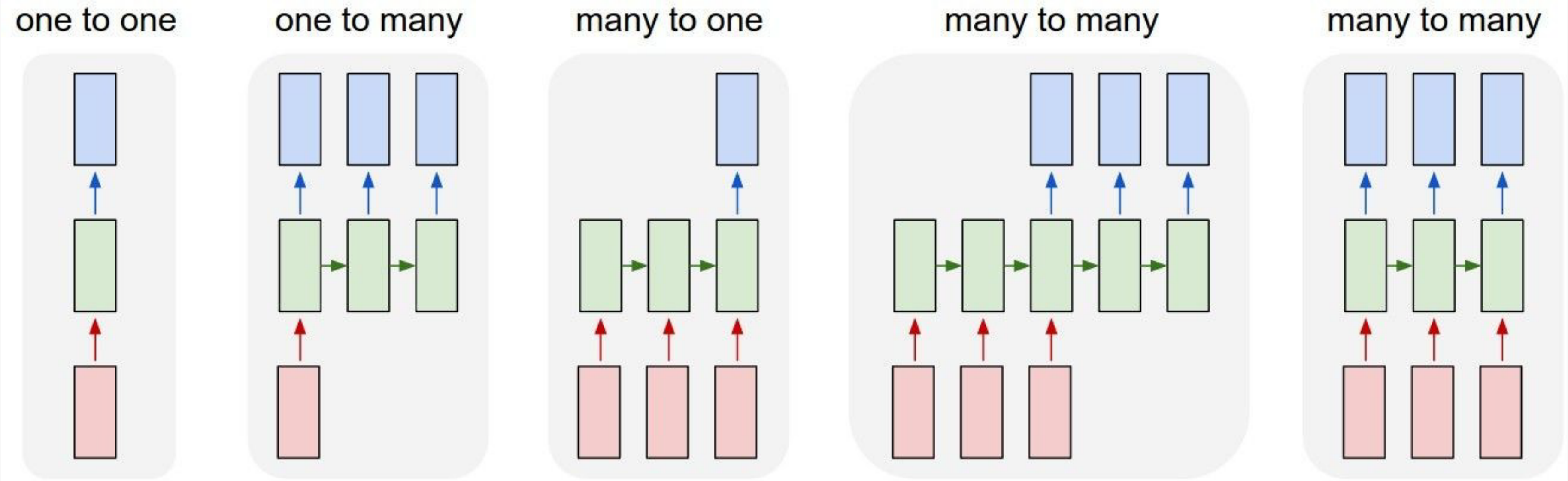
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences



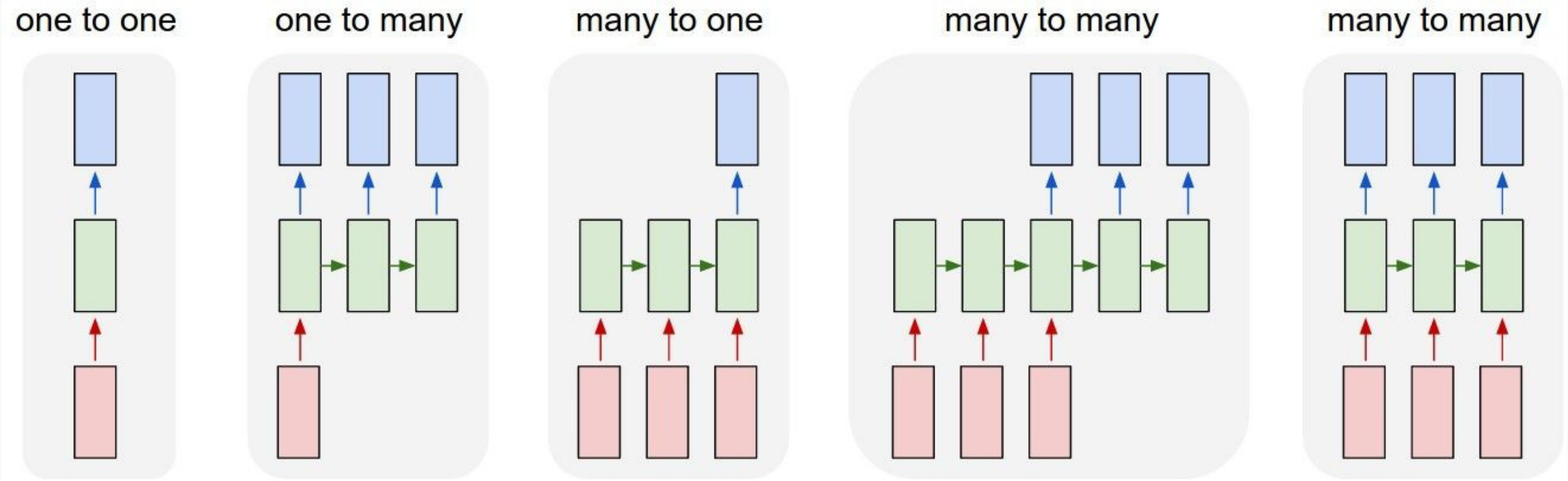
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences



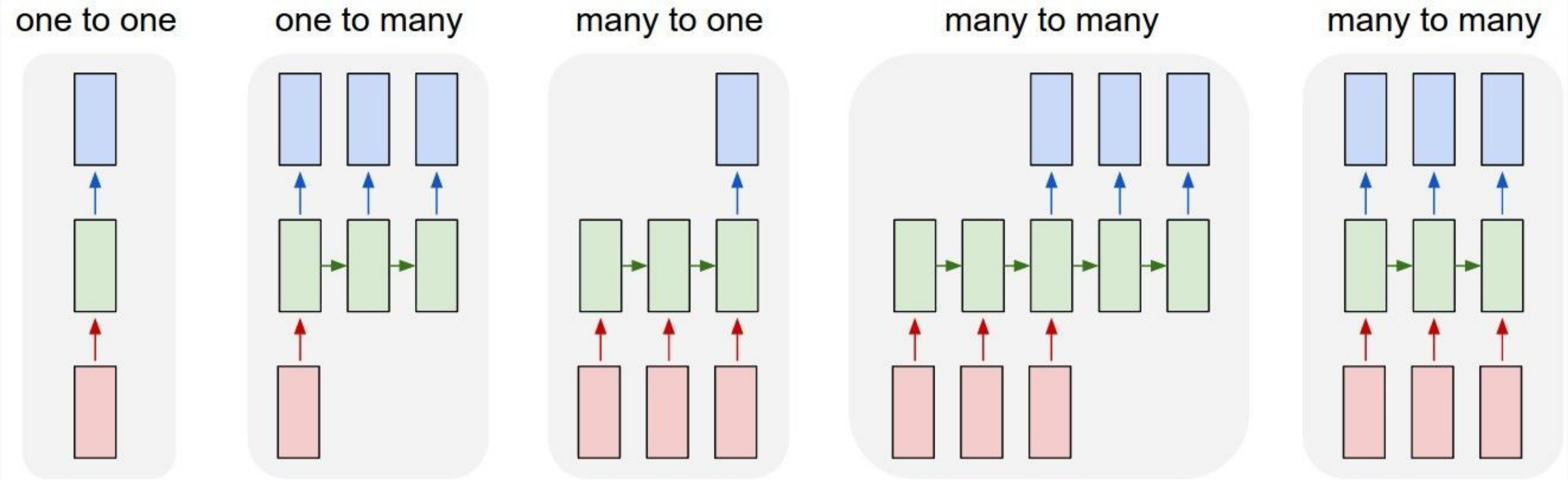
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences



E.g. **Video Captioning**
Sequence of video frames -> caption

Recurrent Neural Networks: Process Sequences



e.g. **Video classification on frame level**

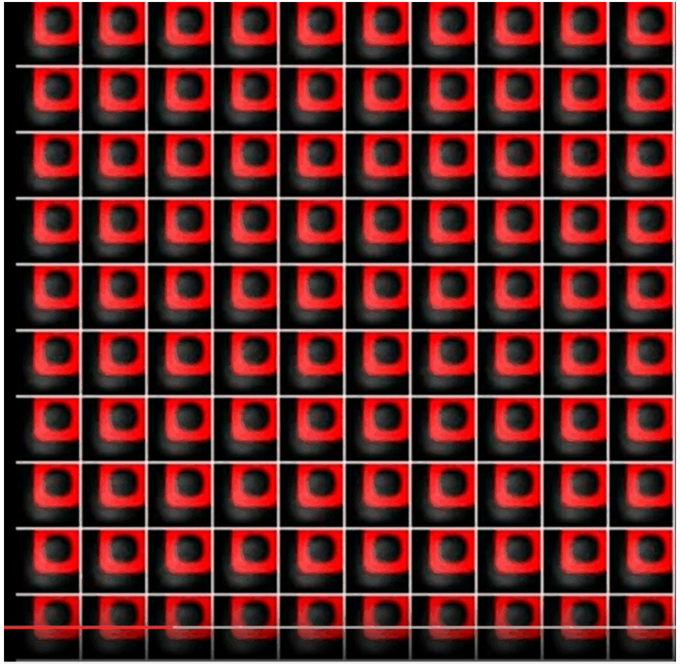
Sequential Processing of Non-Sequence Data

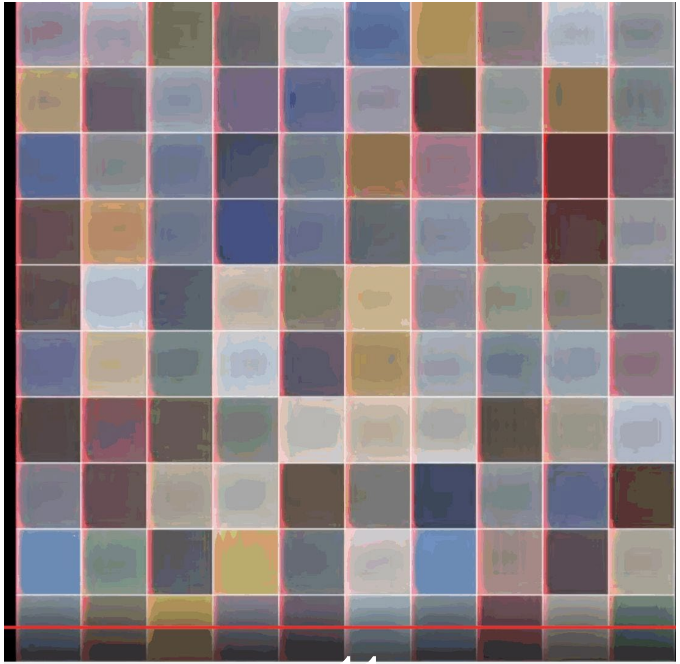
Classify images by taking a series of “glimpses”

Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

Sequential Processing of Non-Sequence Data

Generate images one piece at a time!



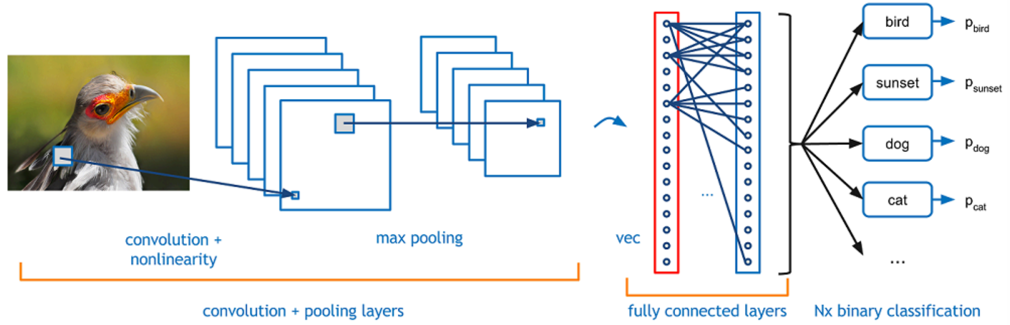


Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015

LENET

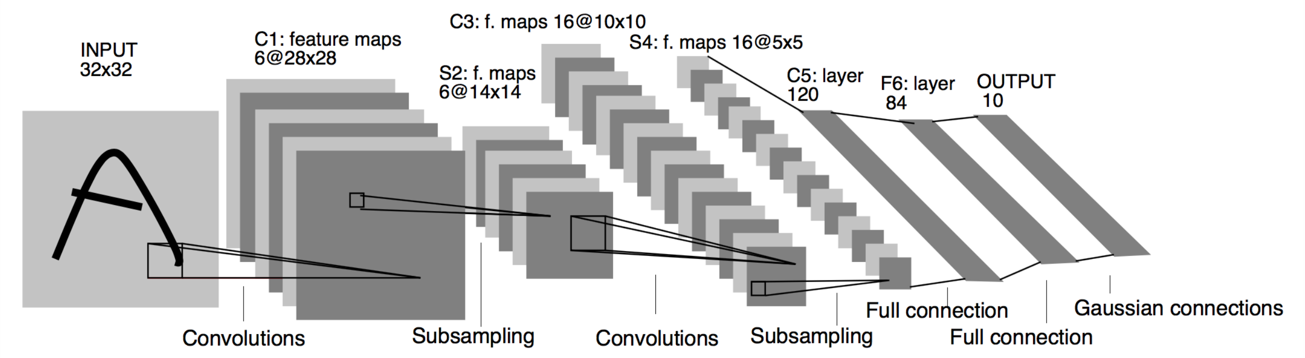
Build a Deep Convolution Neural Network

- It's a multilayer CNN:
  - 每一層 (Layer) 代表 *N* 個不同的 Kernel 計算出來的Feature maps，因此每一層會有 *N* Channels
  - 層與層的連結：上一層的 Feature maps，經過 *K* 個不同 Kernel 計算得出下一層的 *K* 個 Feature maps (*K* channels)
- 最終，由於CNN每個 Layer 是一個4-D結構
  - [Batch size × Width × Height × #Channels]
  - 將每個 Channel 都拉成 Vector，在把所有的 Vector Aggregate 起來



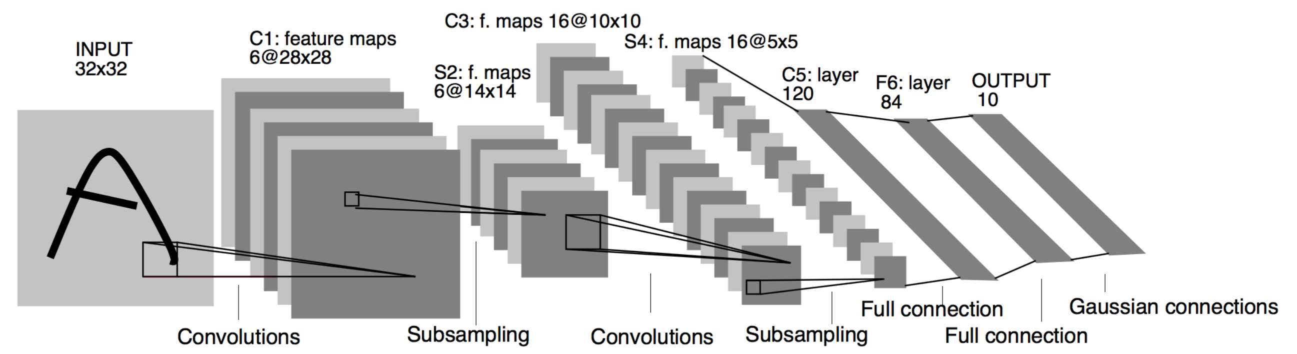
LeNet5

- Introduced by Yann LeCun.
- Raw image of 32 × 32 pixels as input
- The size of kernels：5*5



LeNet5

- C1, C3, C5 : Convolutional layer.
- 5 × 5 Convolution matrix.
- S2 , S4 : Subsampling layer.
- Subsampling by factor 2.
- F6 : Fully connected layer.

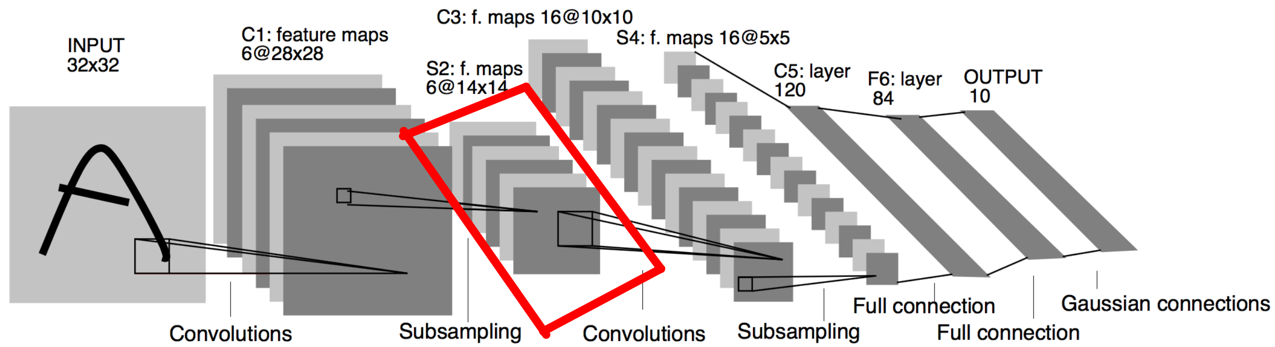


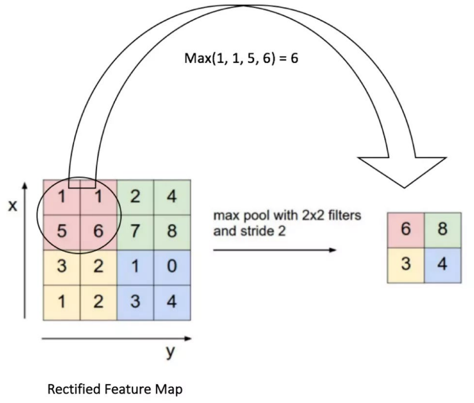
LeNet5

- 輸入圖，先用6個 Kernels 對圖做 Convolution，獲得6張特徵圖
- 每個 Convolution layer 都是拿來學 “特徵”
  - C1, C3, and C5 layers
- 回想捲積 Conv
  - 不同的 Kernel (Filter)，會得到不同的影像結果: 特徵
- 所有的 C 、FC，都會經過一次的 Activation function
  - Sigmoid by default

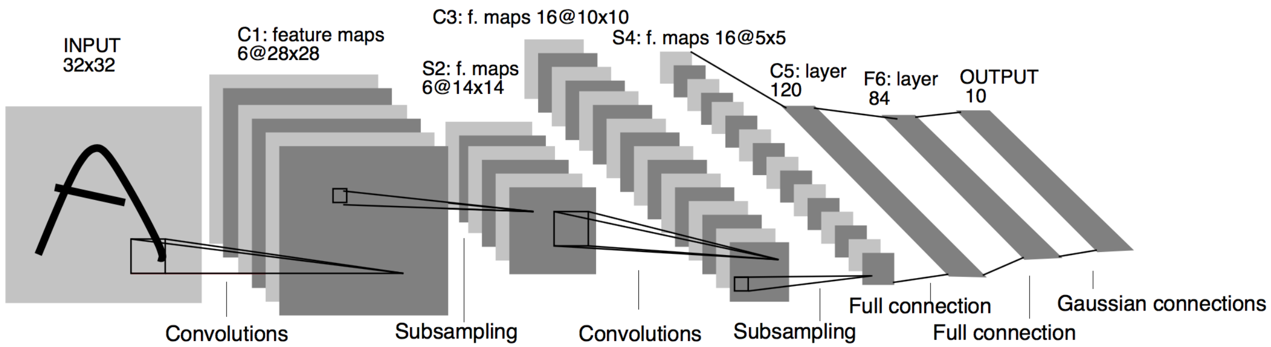
Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

LeNet5: Subsampling (Pooling)

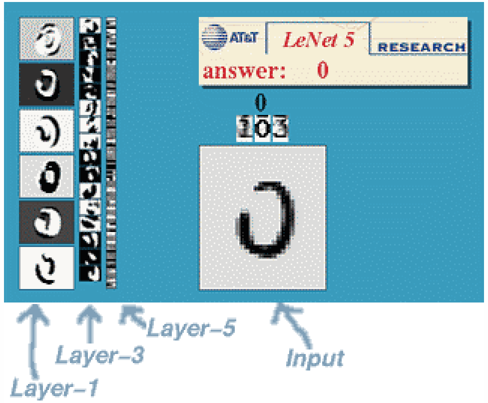




LeNet5



- C5 是怎麼回事?
  - 注意，S4已經是 5*5大小
  - 5*5 影像經過 5*5 的 Kernel
    - 1*1的結果，就是一個點

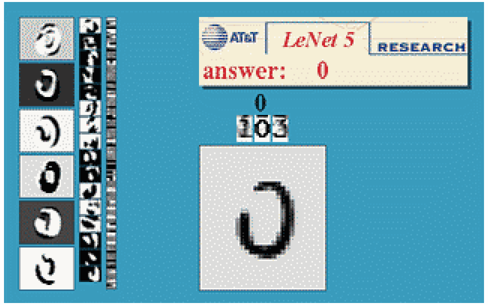


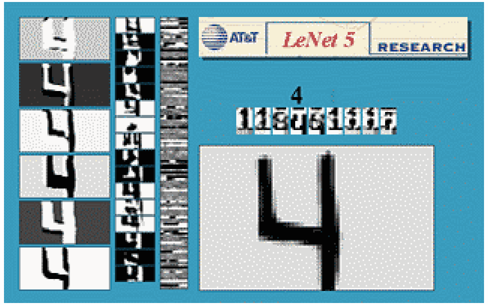
LeNet5參數量計算

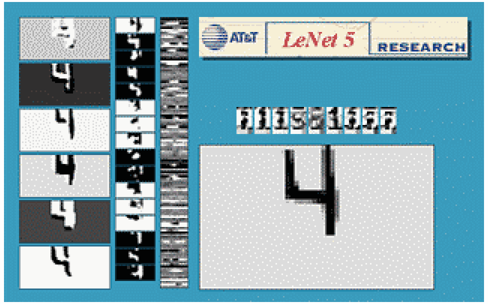
- C1
- Input size：32*32
- Kernel size：5*5
- #Kernels ：6
- Output size：28*28*6
- #Trainable variables： 5*5+1)*6=156
- #Connections ：(5*5+1)*6*(32-2-2)*(32-2-2)=122304

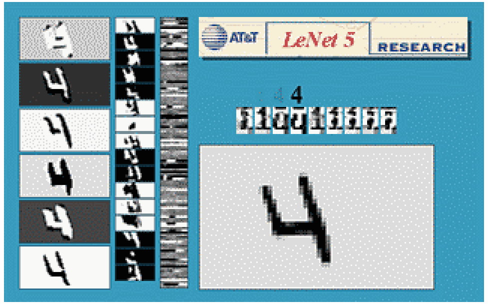
- S2
- Input size ：28*28*6
- Kernel size ：2*2
- #Kernels：1
- Output size ：14*14*6
- #Trainable variables ：2*6=12，2=(w,b)
- #Connections：(2*2+1)*1*14*14*6 = 5880

LeNet5

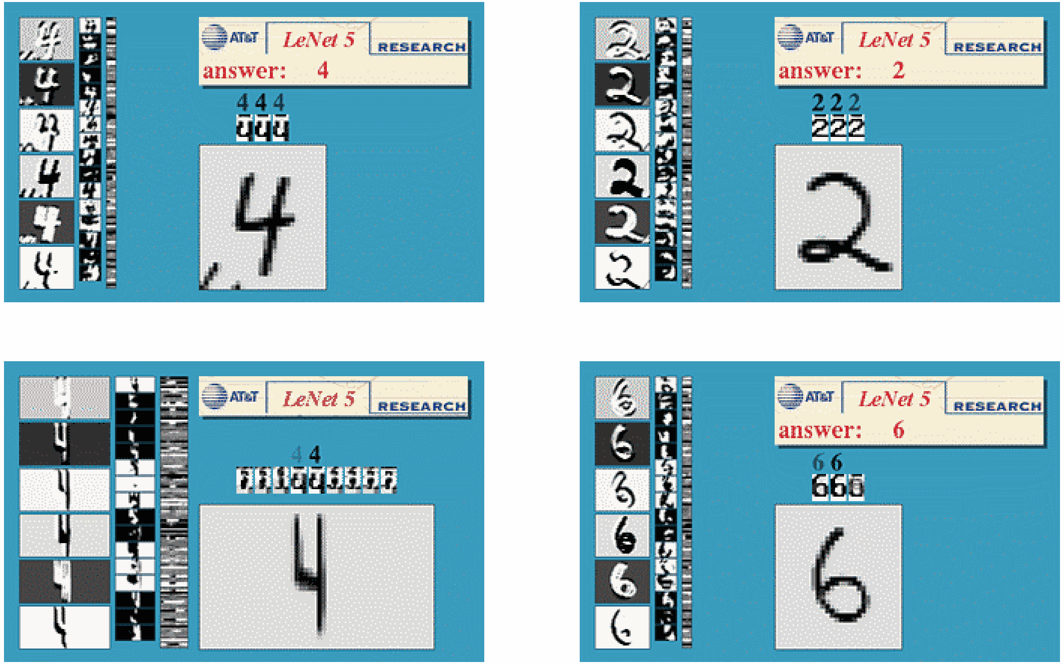

AT&T
LeNet 5
RESEARCH
answer: 0
0
103


AT&T
LeNet 5
RESEARCH
4


AT&T
LeNet 5
RESEARCH

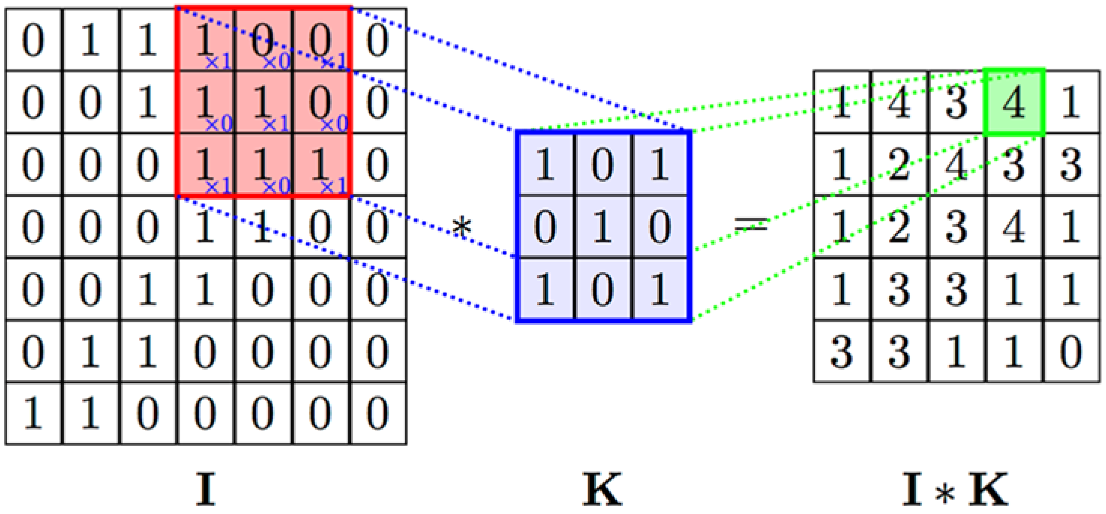

AT&T
LeNet 5
RESEARCH
4

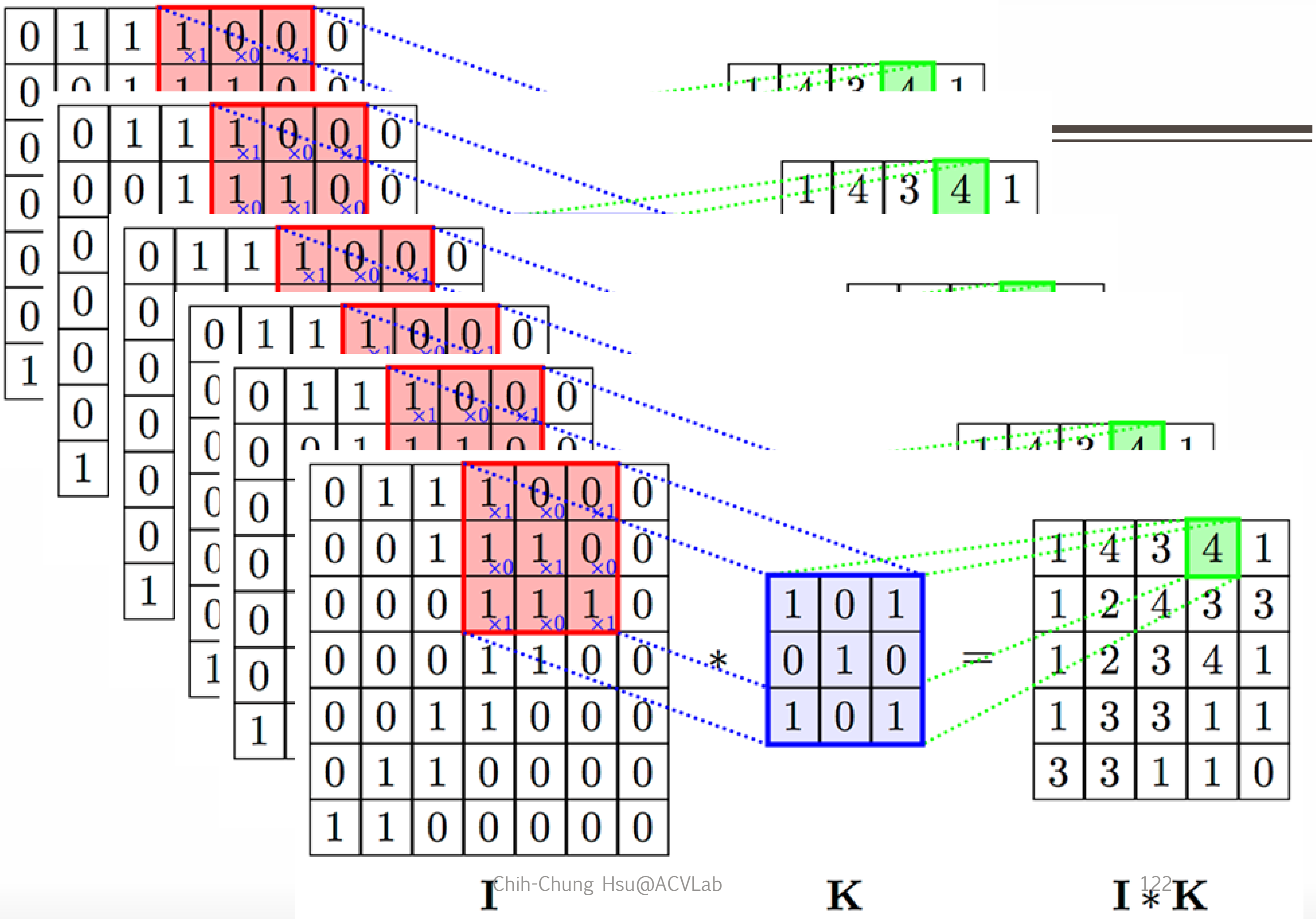
LeNet5


AT&T
LeNet 5
RESEARCH
answer: 4
AT&T
LeNet 5
RESEARCH
answer: 2
AT&T
LeNet 5
RESEARCH
AT&T
LeNet 5
RESEARCH
answer: 6

LeNet 5

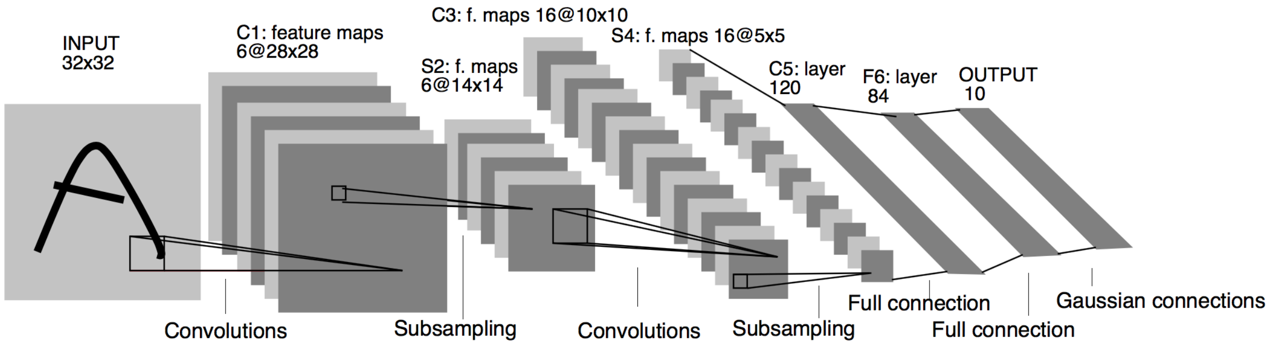
- 如何從1張圖變成6張特徵圖?
  - 1 kernel (K) 產生一張特徵圖 (I*K)




I
K
I * K

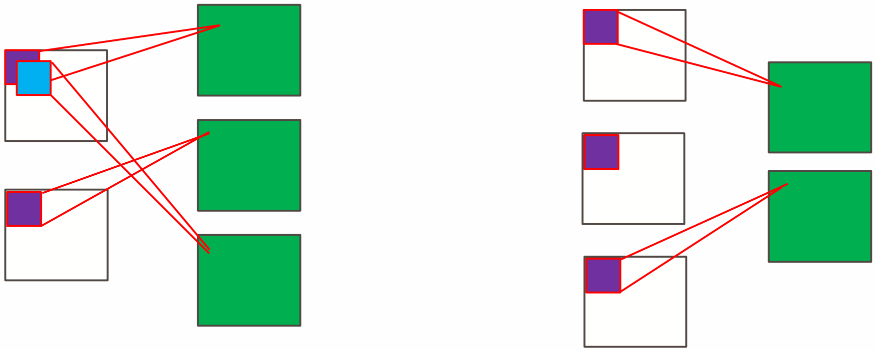
LeNet 5

- 6個Feature maps 轉到 16個 Feature maps?



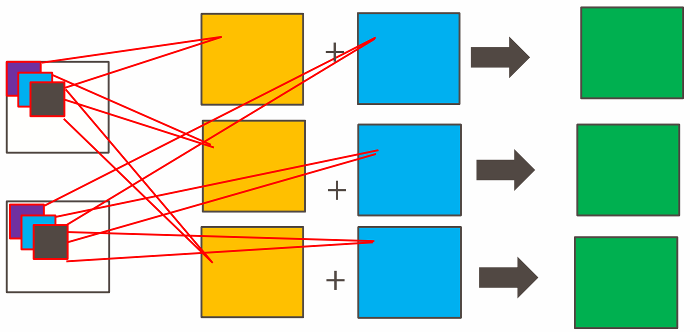
LeNet 5

- 6個Feature maps 轉到 16個 Feature maps?
  - 觀念：
    - 一個feature map 當成是輸入，那麼對接的有16個kernels，會產生16個下一層feature maps
      - 那不就會有16*6個feature map?
    - 為了保持一致性，Conv. Layers有幾種策略
    - 1. 直接對應：少變多，複製，多變少，間隔取樣



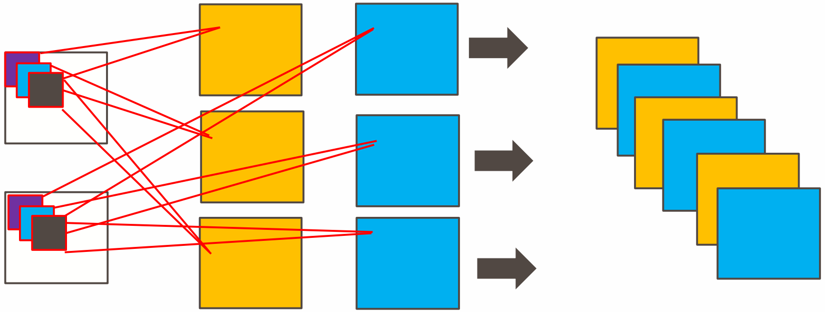
LeNet 5

- 6個Feature maps 轉到 16個 Feature maps?
  - 觀念：
    - 一個feature map 當成是輸入，那麼對接的有16個kernels，會產生16個下一層feature maps
      - 那不就會有16*6個feature map?
    - 為了保持一致性，Conv. Layers有幾種策略
    - 2. Aggregated：一律產生對接個kernels的特徵圖，再疊加
      - i.e., W*H*C ➔ 3*3*2 to 3*3*3, 27-dimensional dot product



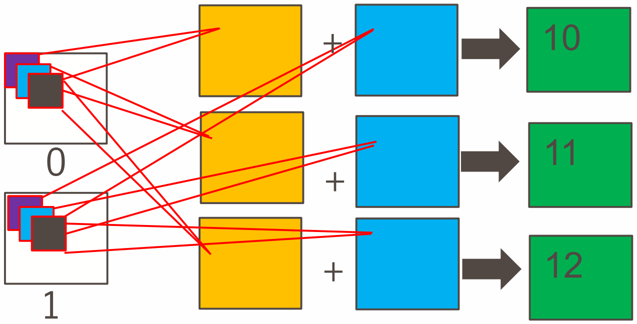
LeNet 5

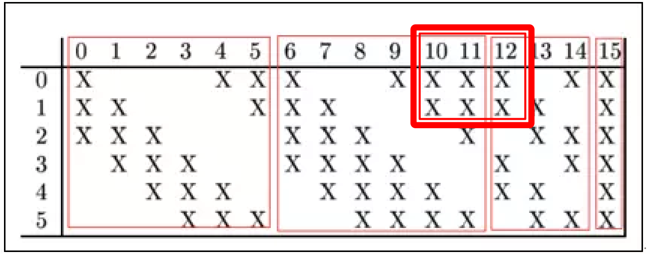
- 6個Feature maps 轉到 16個 Feature maps?
  - 觀念：
    - 一個feature map 當成是輸入，那麼對接的有16個kernels，會產生16個下一層feature maps
      - 那不就會有16*6個feature map?
    - 為了保持一致性，Conv. Layers有幾種策略
    - 3. Full conv: 產生對接個kernels的特徵圖*輸入數量



LeNet 5

- 6個Feature maps 轉到 16個 Feature maps?
  - 觀念：
    - 一個feature map 當成是輸入，那麼對接的有16個kernels，會產生16個下一層feature maps
      - 那不就會有16*6個feature map?
    - 為了保持一致性，Conv. Layers有幾種策略
    - 4. Manually determine: 自由設計





	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

LeNet 5

- Feature map 對應策略
  - 直接對應
    - 太過簡化，無法有效用到所有的特徵
  - Aggregated
    - 產生多個feature maps再疊加，有較多的特徵，且保留對接的層數
  - Full conv
    - 最多特徵，然而參數過多無法控制
- 目前大宗的都是Aggregated策略
  - i.e., shown in previous slides