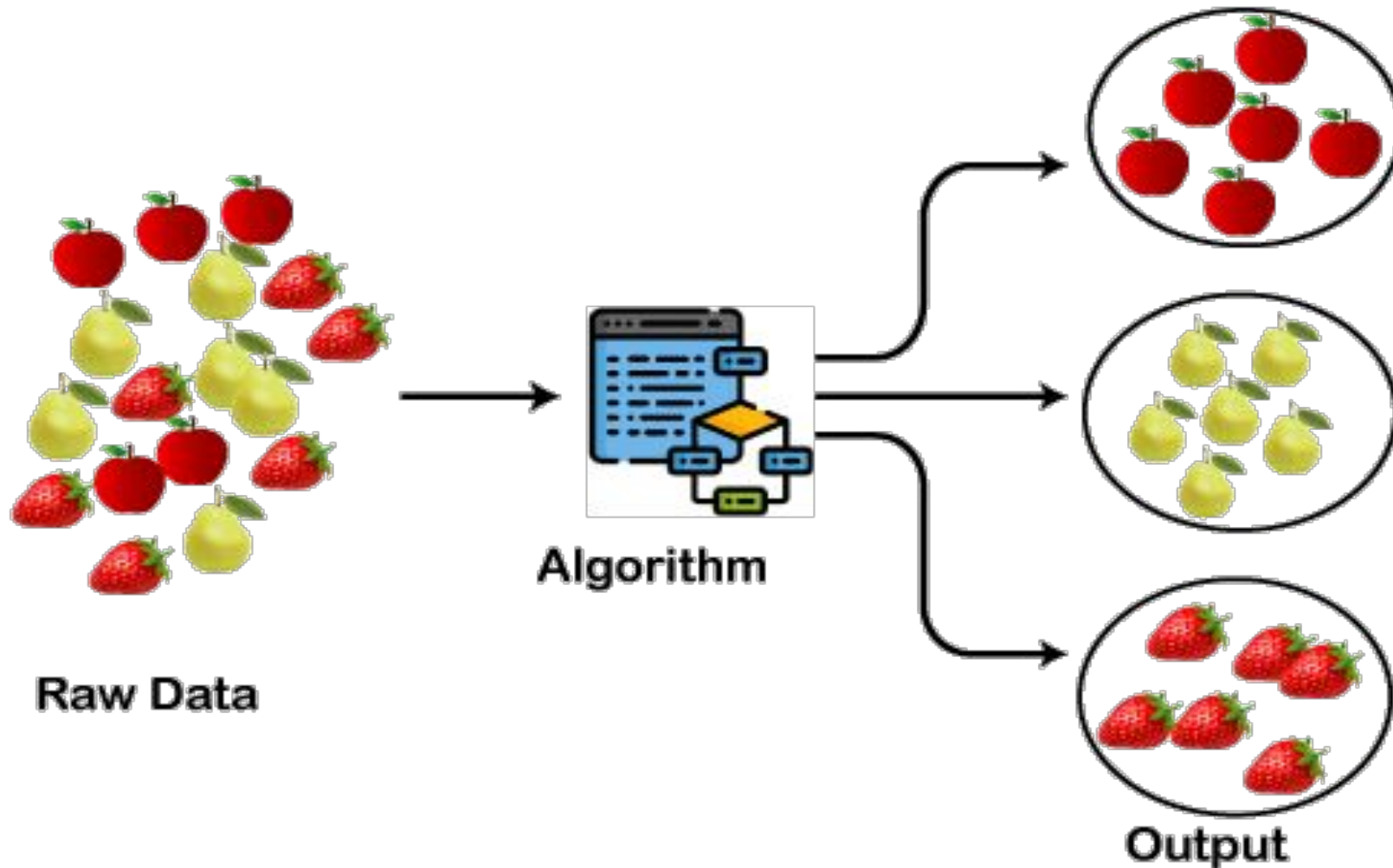


An abstract graphic on a dark blue background featuring a complex network of white lines and dots. The dots, representing nodes, are of varying sizes and are connected by thin white lines, creating a web-like structure that spans the width of the image. The overall aesthetic is technological and digital.

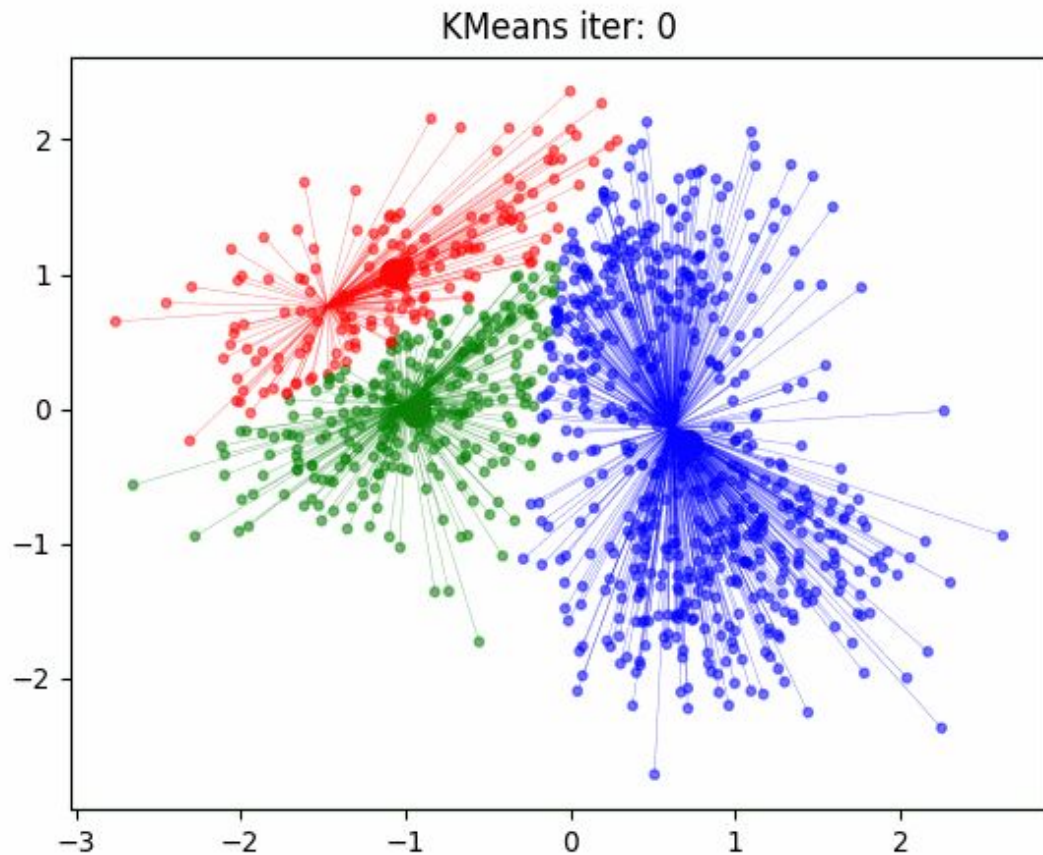
# Day 6 GNITS AI Workshop

# What is Clustering ?

Finding groups of objects such that objects in a group will be similar to one another and different from objects in other groups

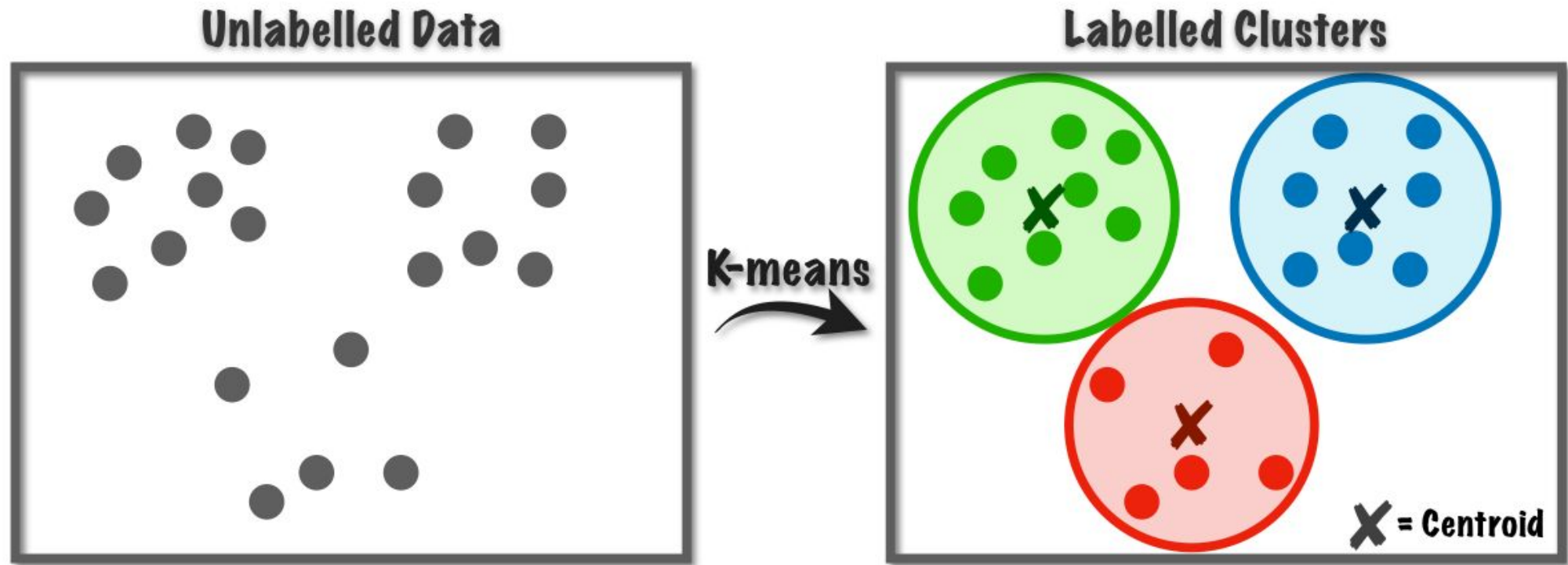


# K Means Clustering ?



- K-Means clustering is an unsupervised iterative clustering technique.
- It partitions the given data set into k predefined distinct clusters.
- Unsupervised machine learning is a type of algorithm that learns patterns from untagged data.

# K Means Clustering

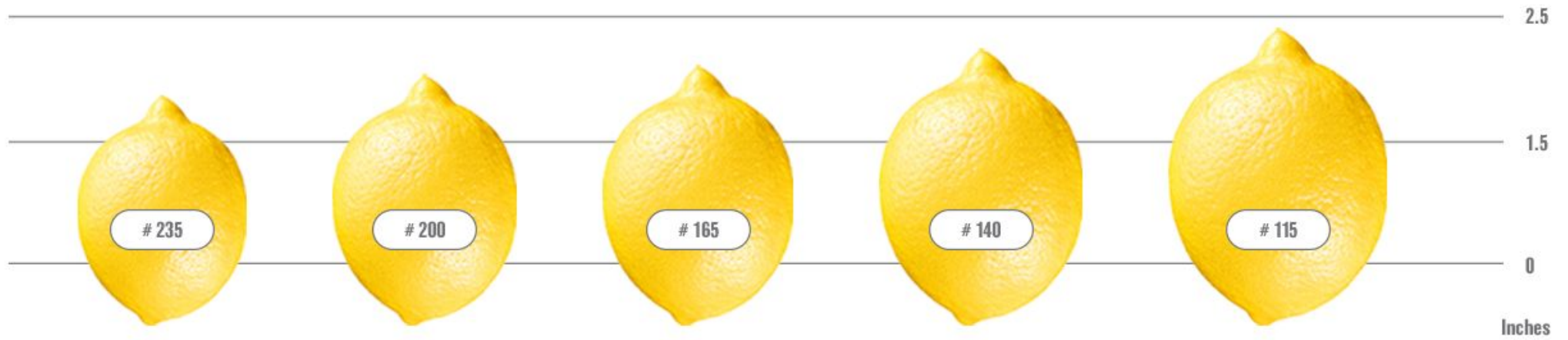


It partitions the data set such that

- Each data point belongs to a cluster with the nearest mean.
- Data points belonging to one cluster have high degree of similarity.
- Data points belonging to different clusters have high degree of dissimilarity.

# K Means Clustering

Numbers refer to sizing as noted above





# K Means Clustering

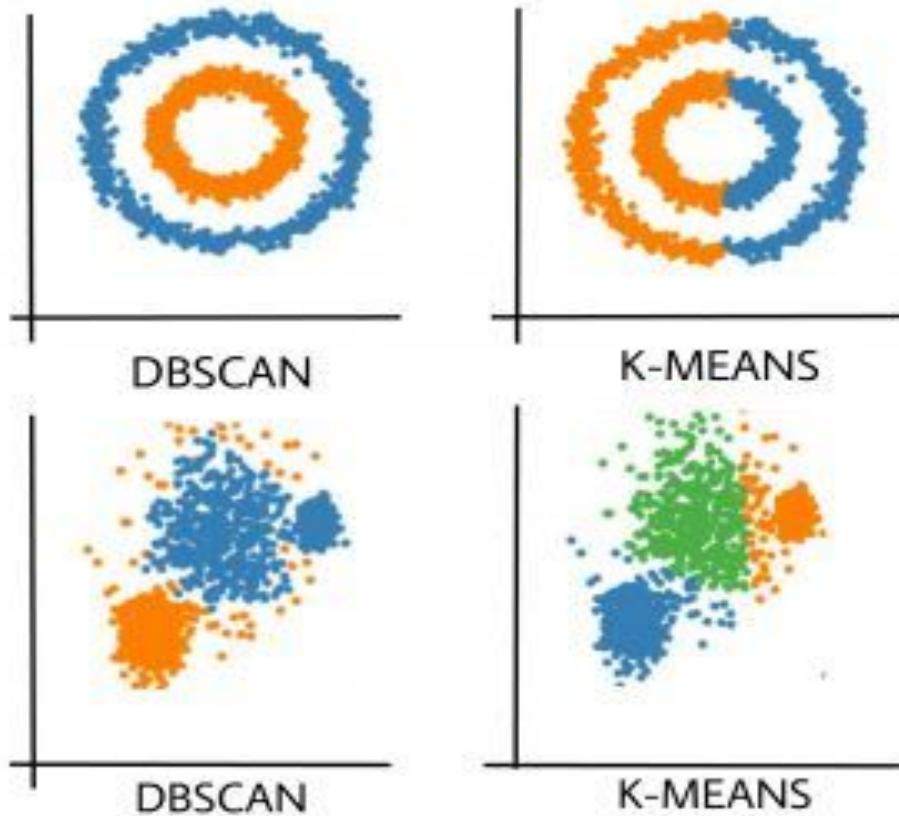
- Choose the number of clusters K.
- Randomly select any K data points as cluster centers.
- Select cluster centers in such a way that they are as farther as possible from each other.
- Calculate the distance between each data point and each cluster center.
- The distance may be calculated either by using given distance function or by using euclidean distance formula.

# K Means Clustering

- Assign each data point to some cluster.
- A data point is assigned to that cluster whose center is nearest to that data point.
- Re-compute the center of newly formed clusters.
- The center of a cluster is computed by taking mean of all the data points contained in that cluster
- Keep repeating the procedure from Step-03 to Step-05 until any of the following stopping criteria is met.
- Center of newly formed clusters do not change.

# Density based Spatial Clustering ( DBSCAN )

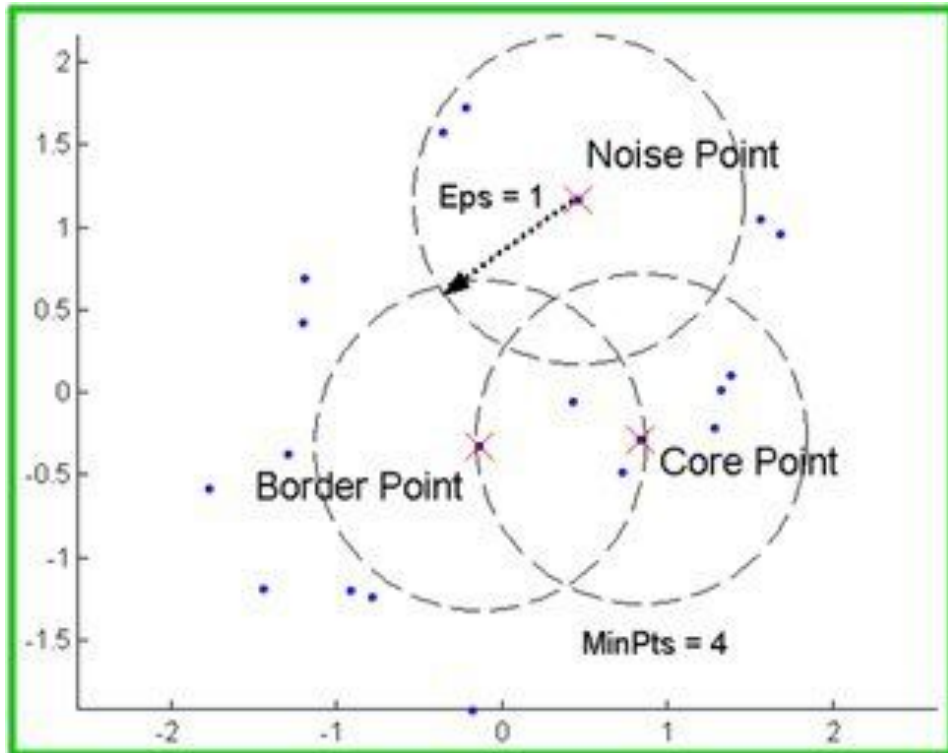
**Dense regions belong to clusters and sparser region belong to noise**



- Follow <https://www.youtube.com/watch?v=RDZUdRSDOok>



# Density based Spatial Clustering ( DBSCAN )



How to measure density?

**Density at point  $p$ :** number of points within a circle of radius  $Eps$ .

**Dense Region:** A circle of radius  $Eps$  that contains at least  $MinPts$  points

**Characterization of points :**

A point is a **core point** if it has more than a specified number of points ( $MinPts$ ) within  $Eps$ . These points belong in a dense region and are at the interior of a cluster

A **border point** has fewer than  $MinPts$  within  $Eps$  but is in the neighborhood of a core point.

A **noise point** is any point that is not a core point or a border point.

- Follow <https://www.youtube.com/watch?v=RDZUdRSDOok>

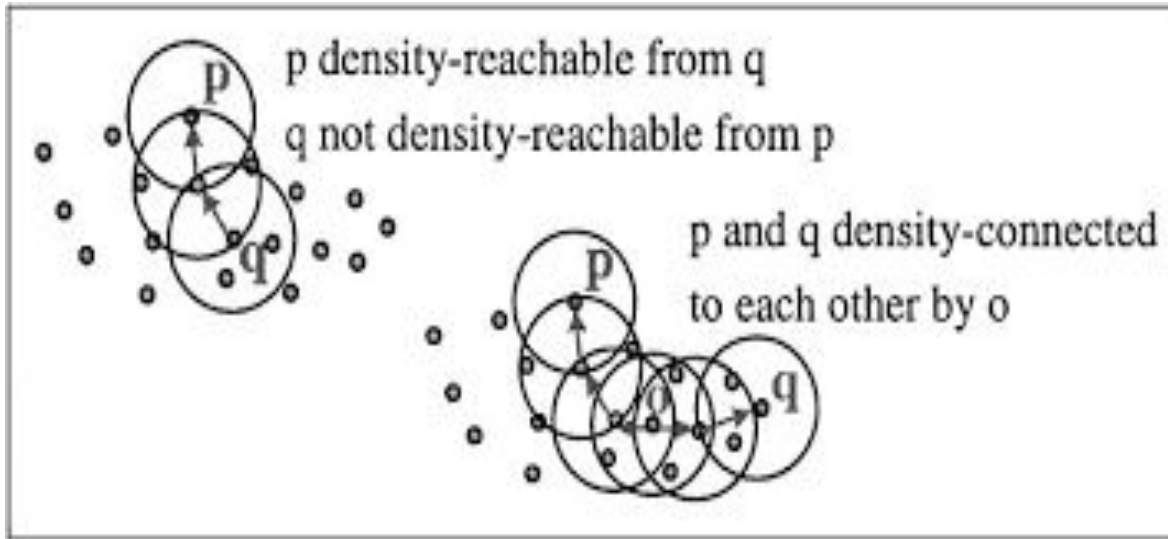
# Density based Spatial Clustering ( DBSCAN )

## Density edge :

We place an edge between two core points  $q$  and  $p$  if they are within distance  $Eps$ .

## Density-connected points:

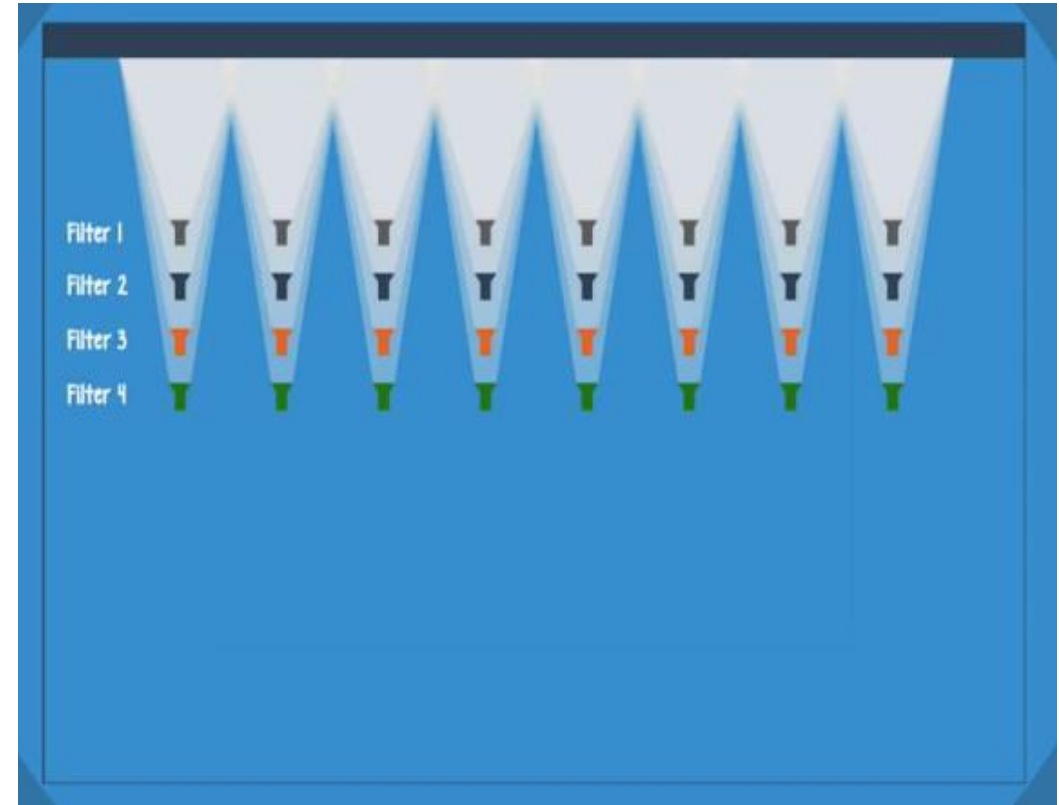
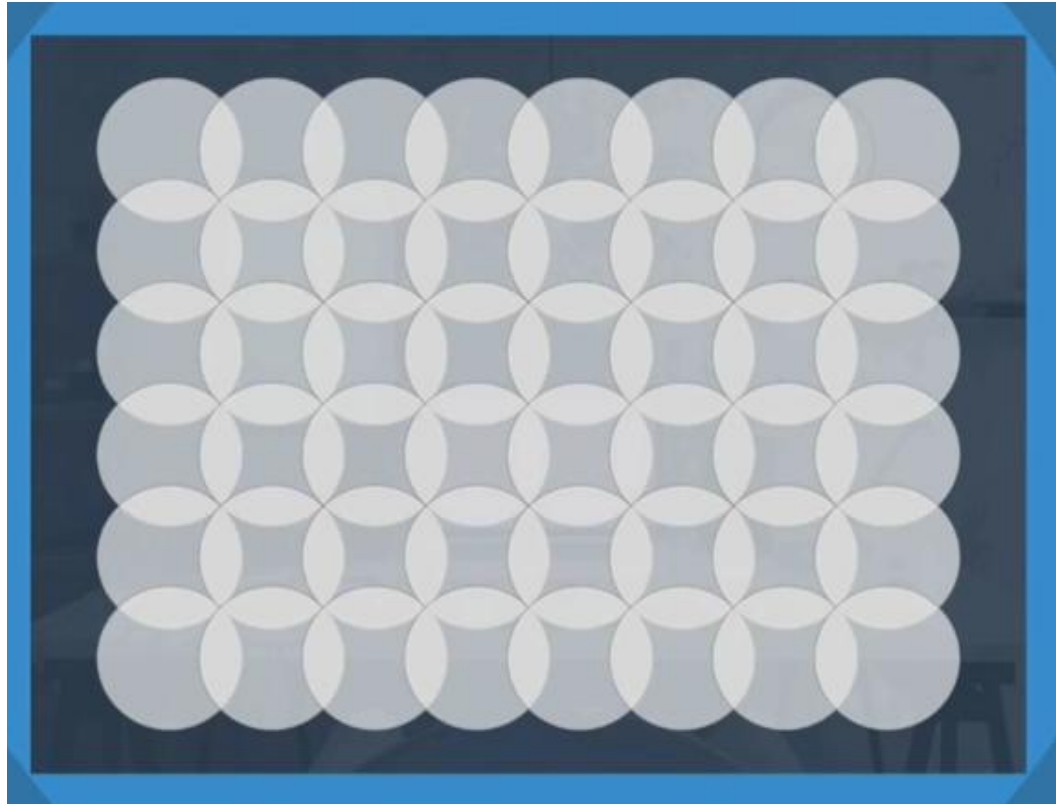
A point  $p$  is density-connected to a point  $q$  if there is a path of edges from  $p$  to  $q$



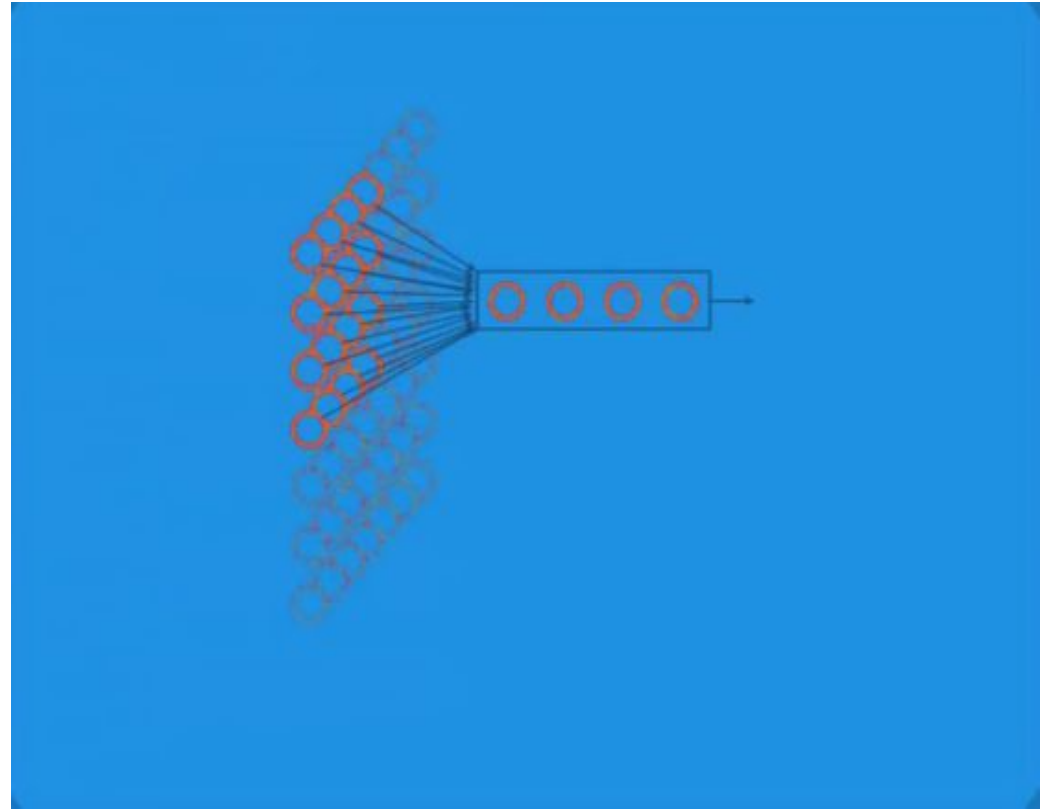
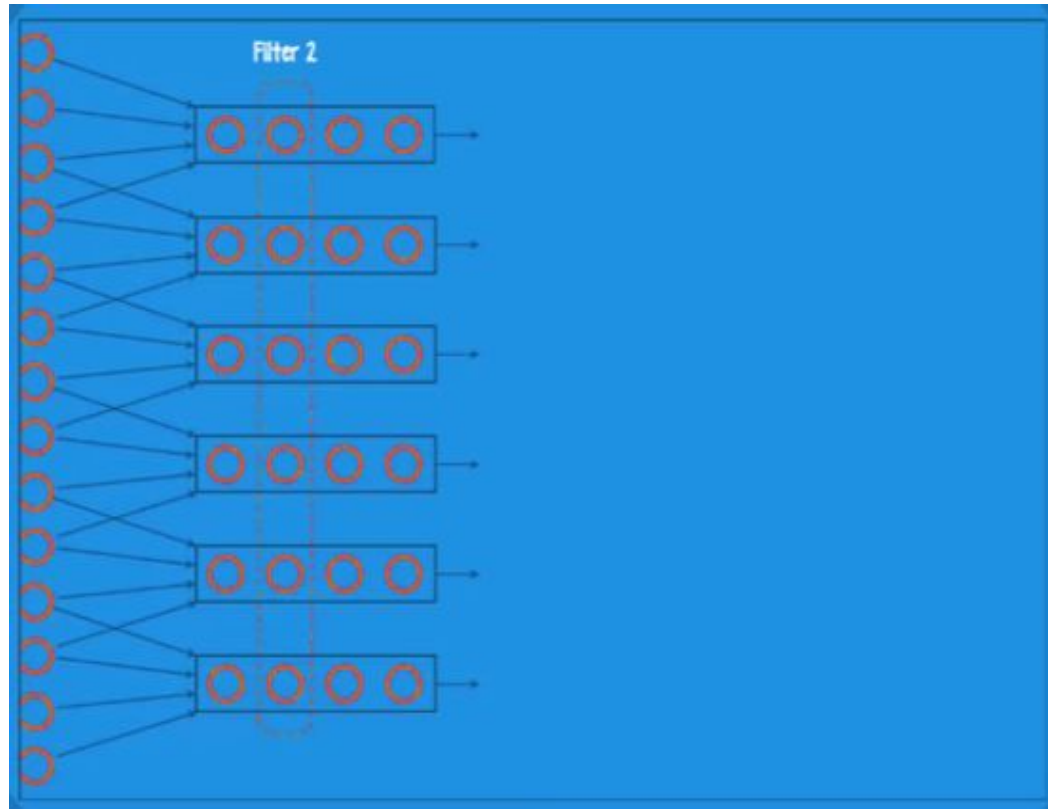
- Follow <https://www.youtube.com/watch?v=RDZUdRSDOok>

Questions ?

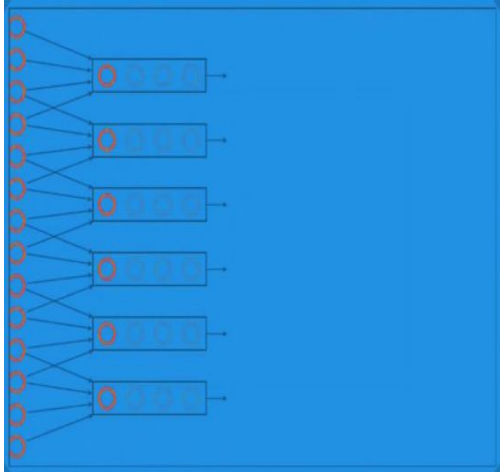
# CONVOLUTION NEURAL NETWORKS



# CONVOLUTION NEURAL NETWORKS

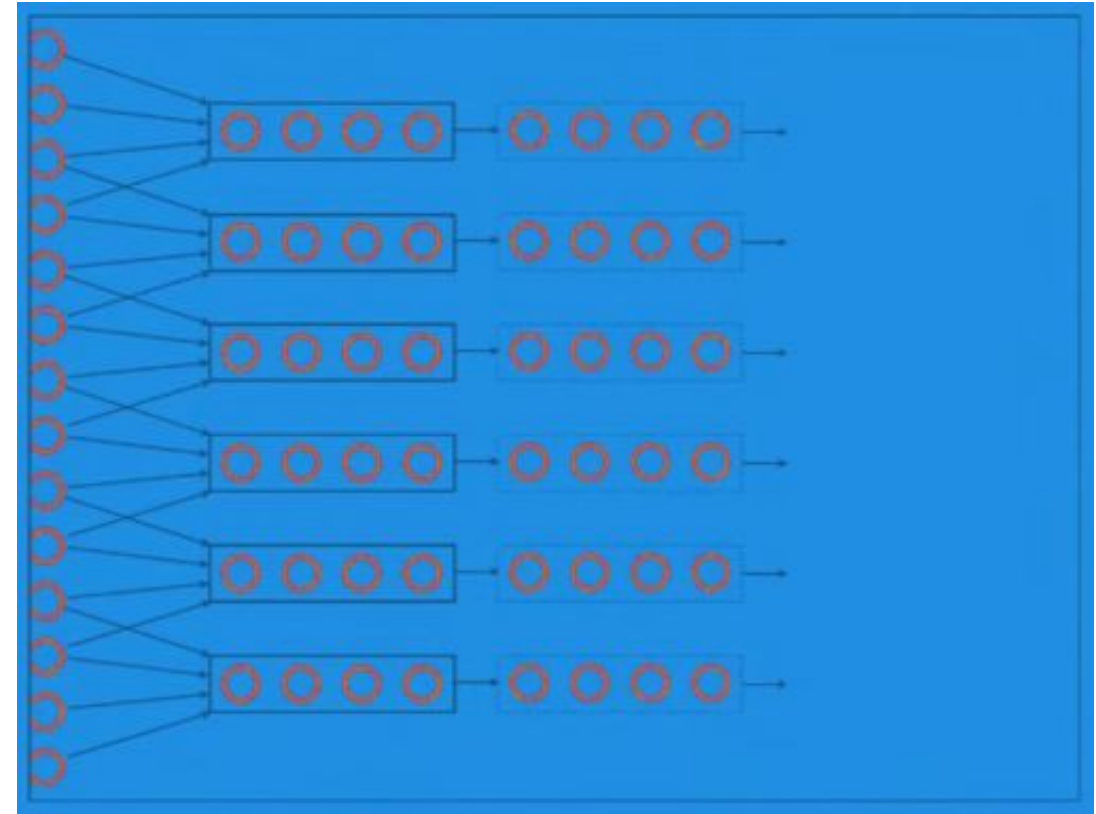
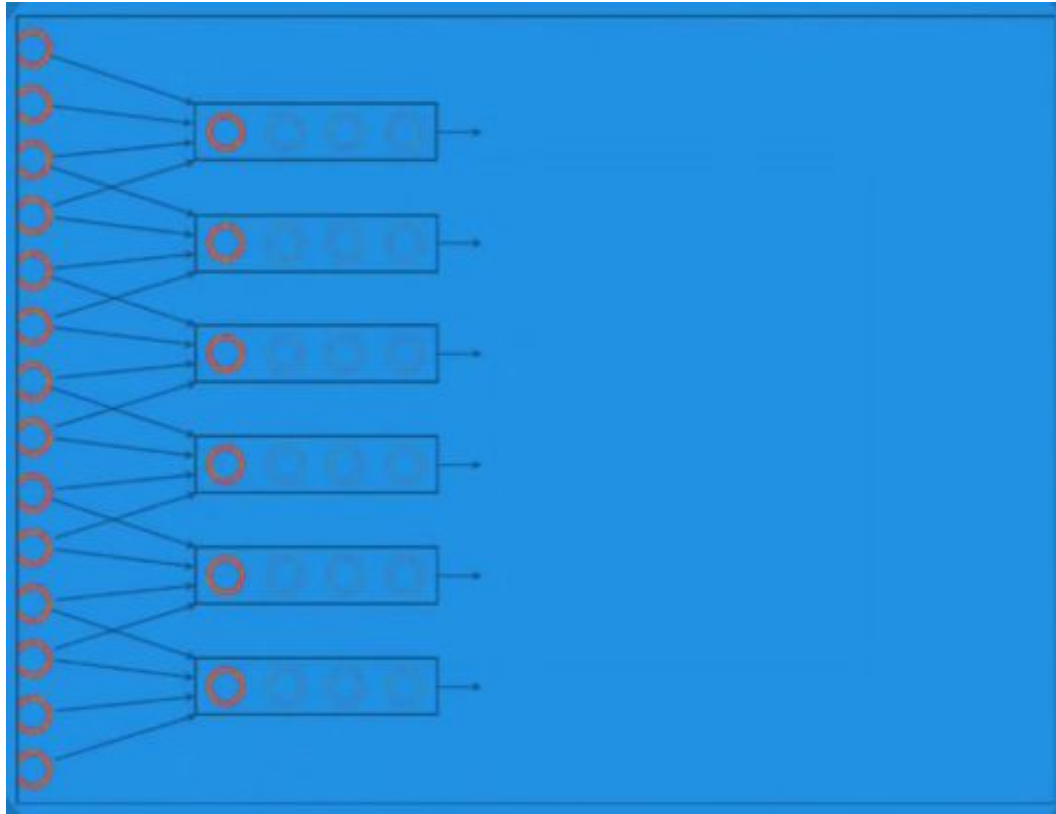


# CONVOLUTION NEURAL NETWORKS

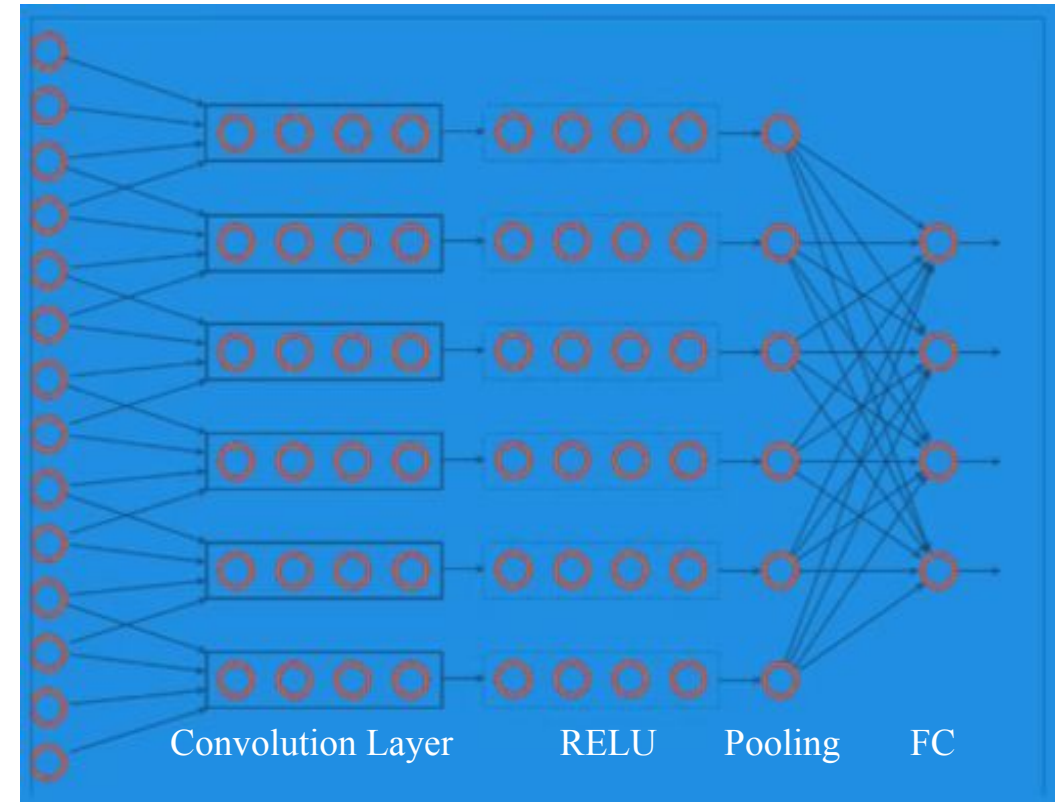
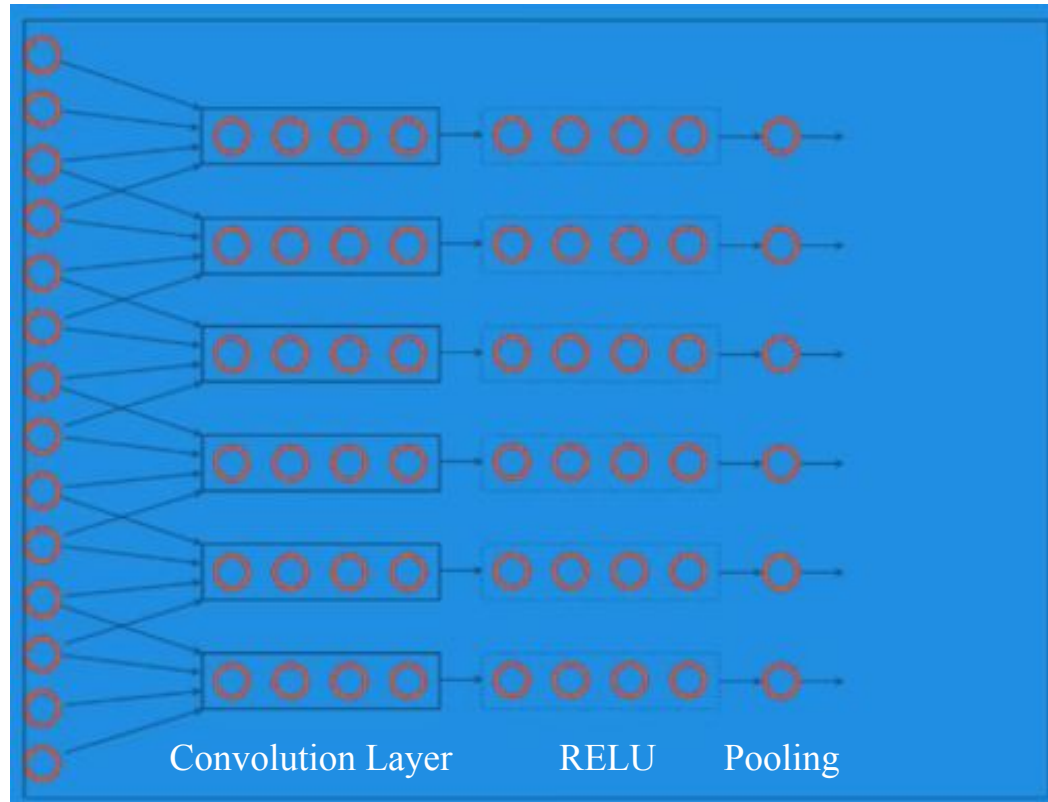




# CONVOLUTION NEURAL NETWORKS

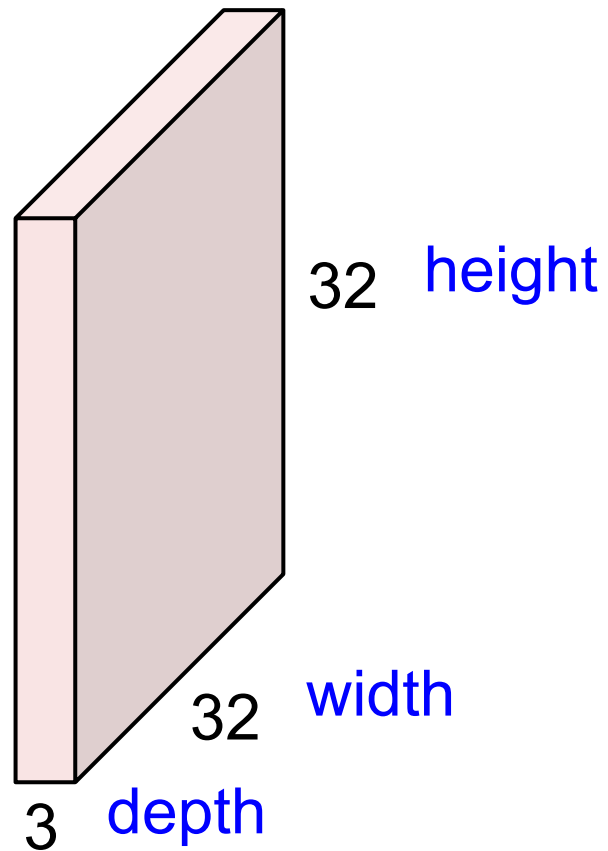


# CONVOLUTION NEURAL NETWORKS



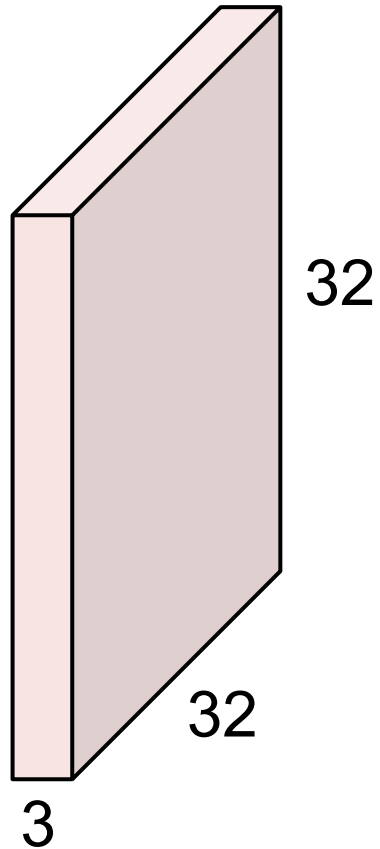
# Convolution Layer

32x32x3 image



# Convolution Layer

32x32x3 image



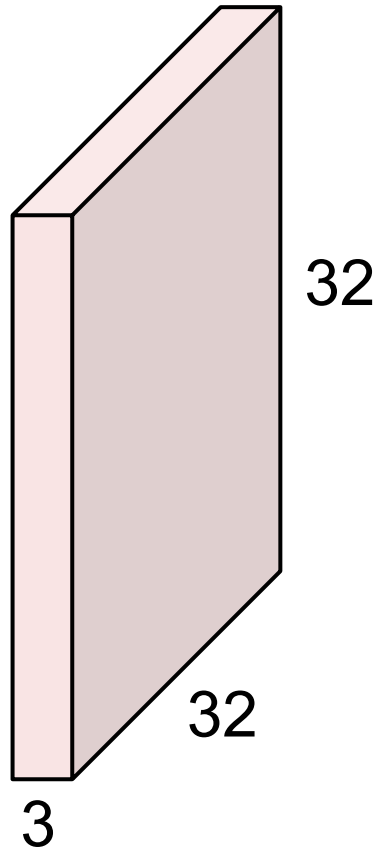
5x5x3 filter



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

# Convolution Layer

32x32x3 image



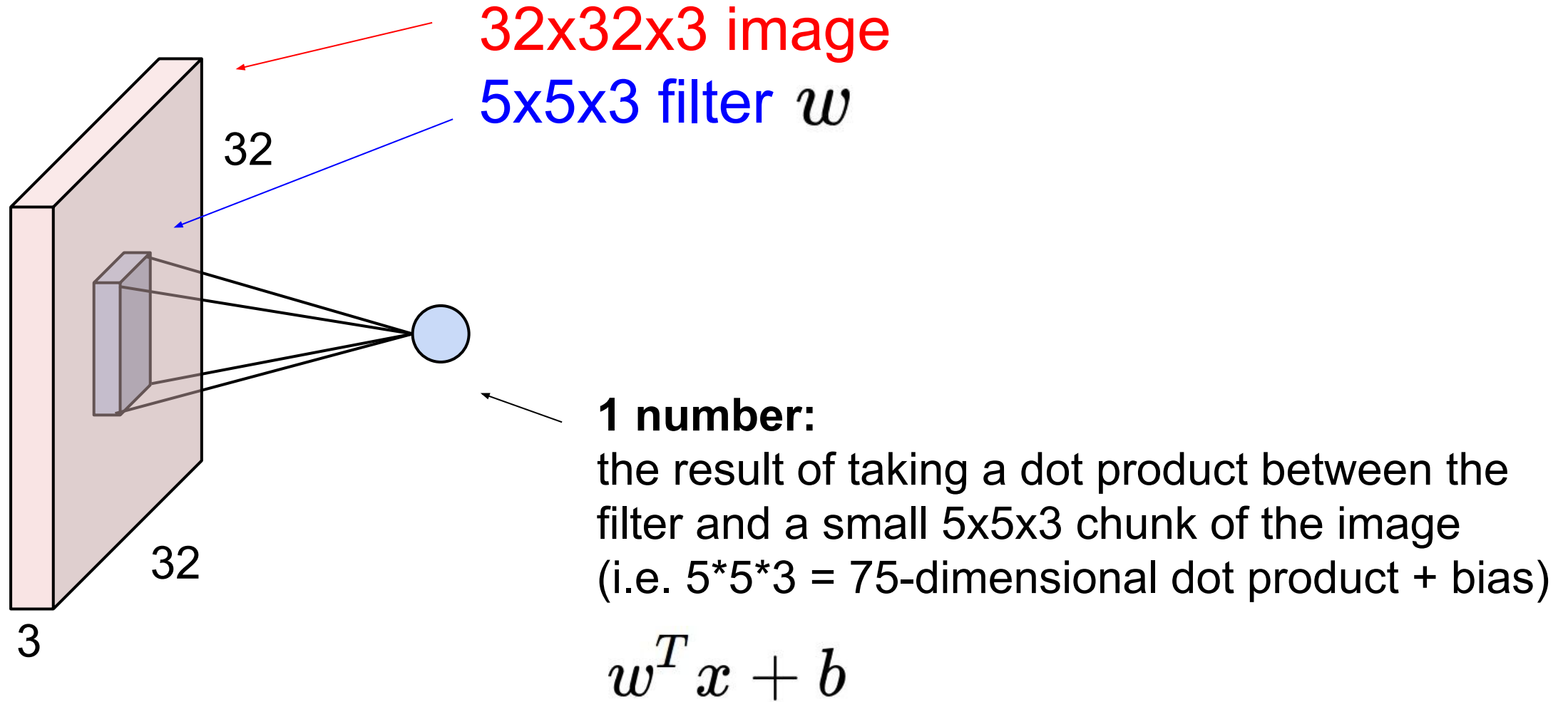
Filters always extend the full depth of the input volume

5x5x3 filter



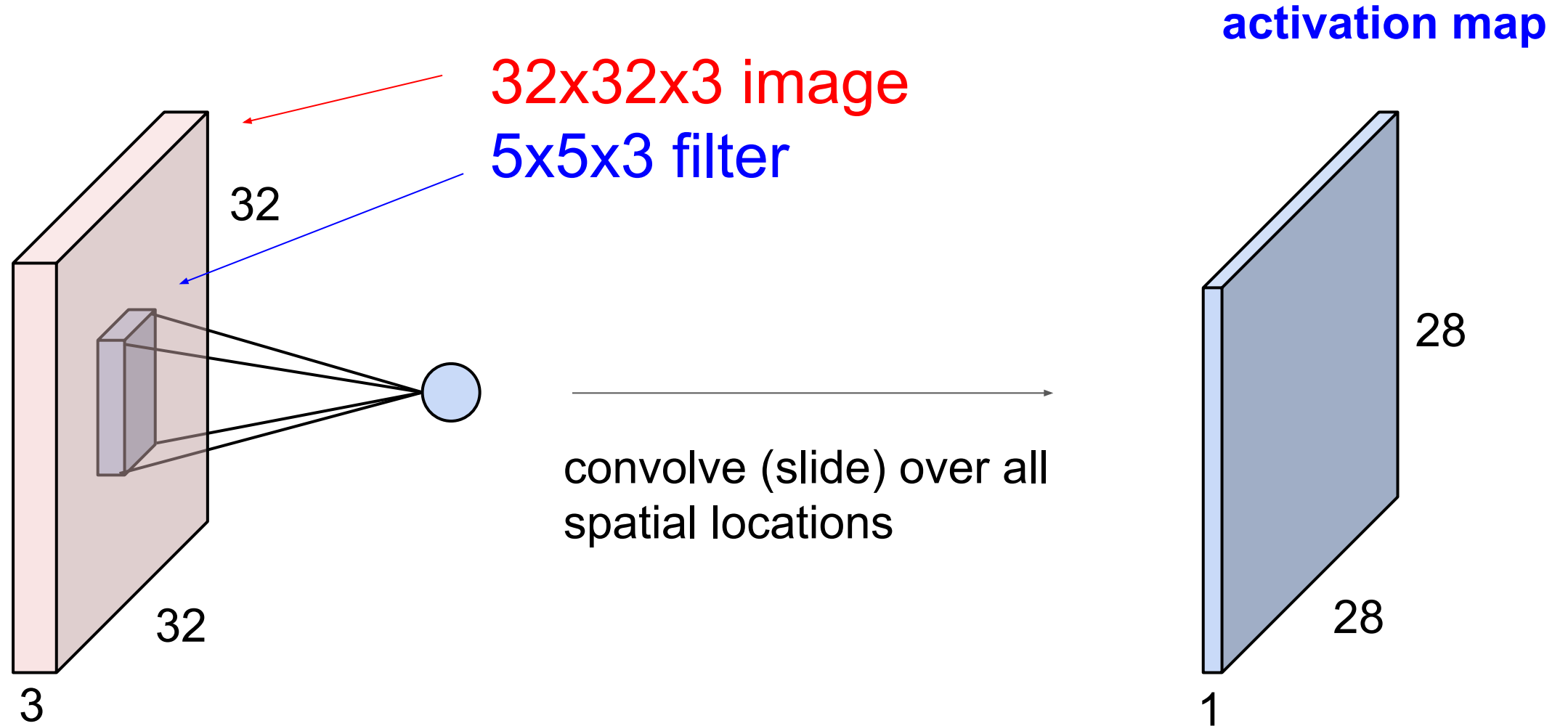
**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

# Convolution Layer





# Convolution Layer

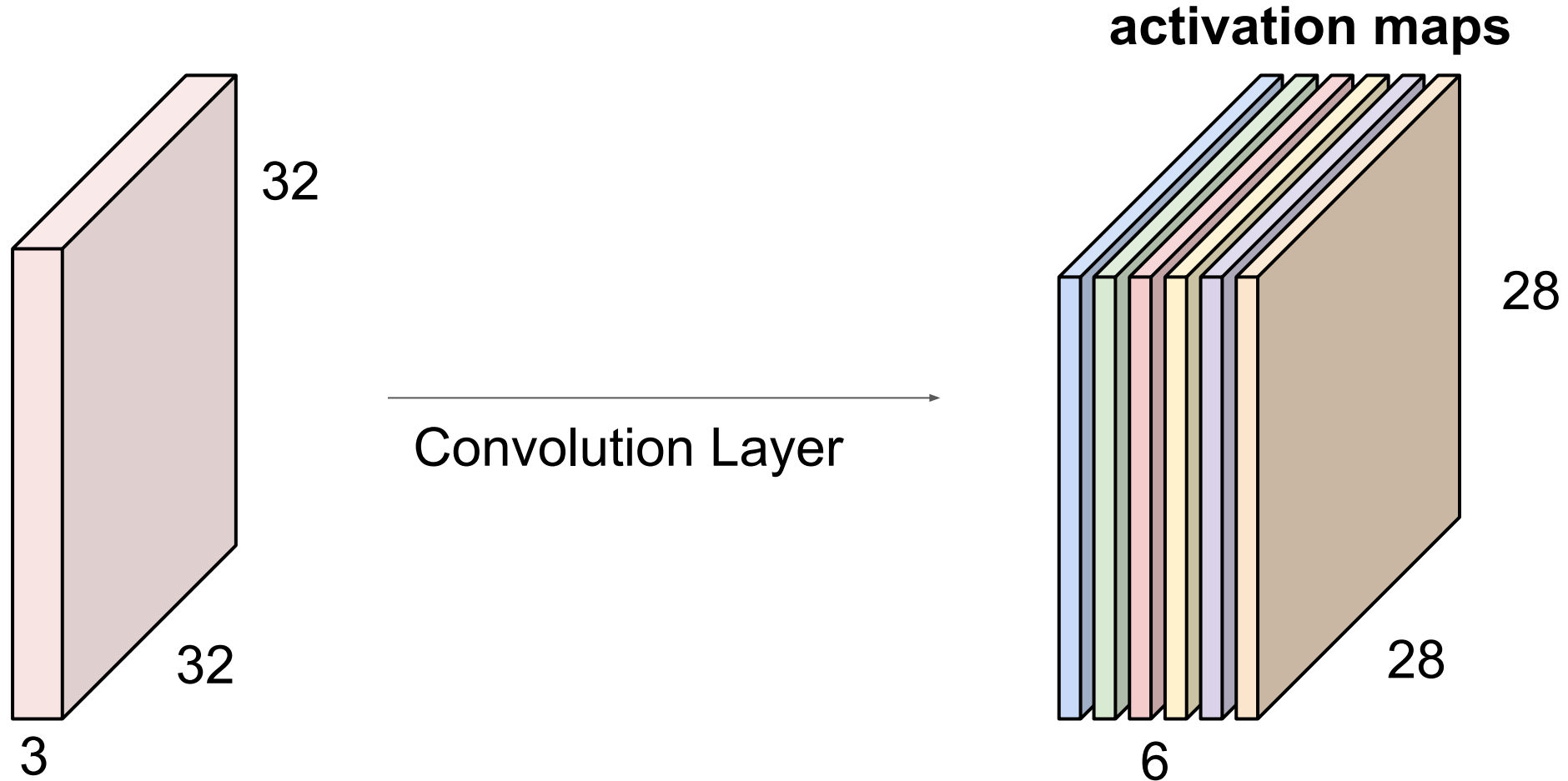


# Convolution Layer

consider a second, **green** filter

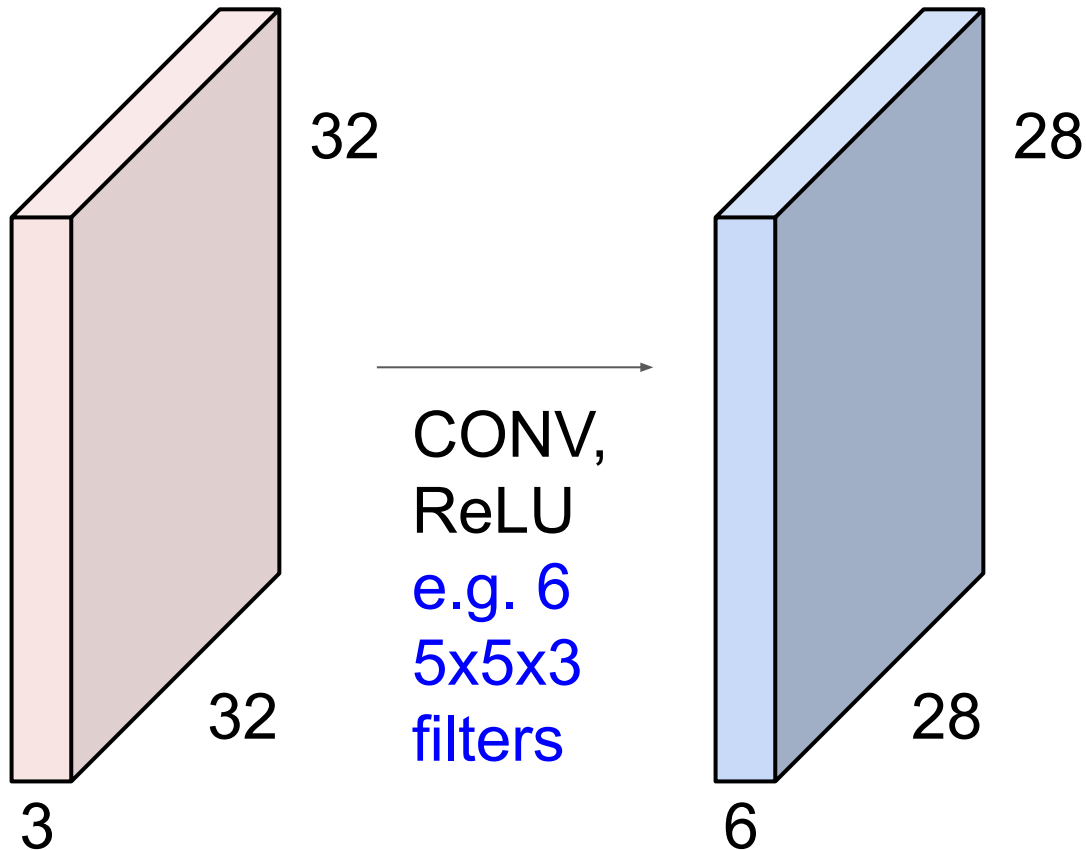


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

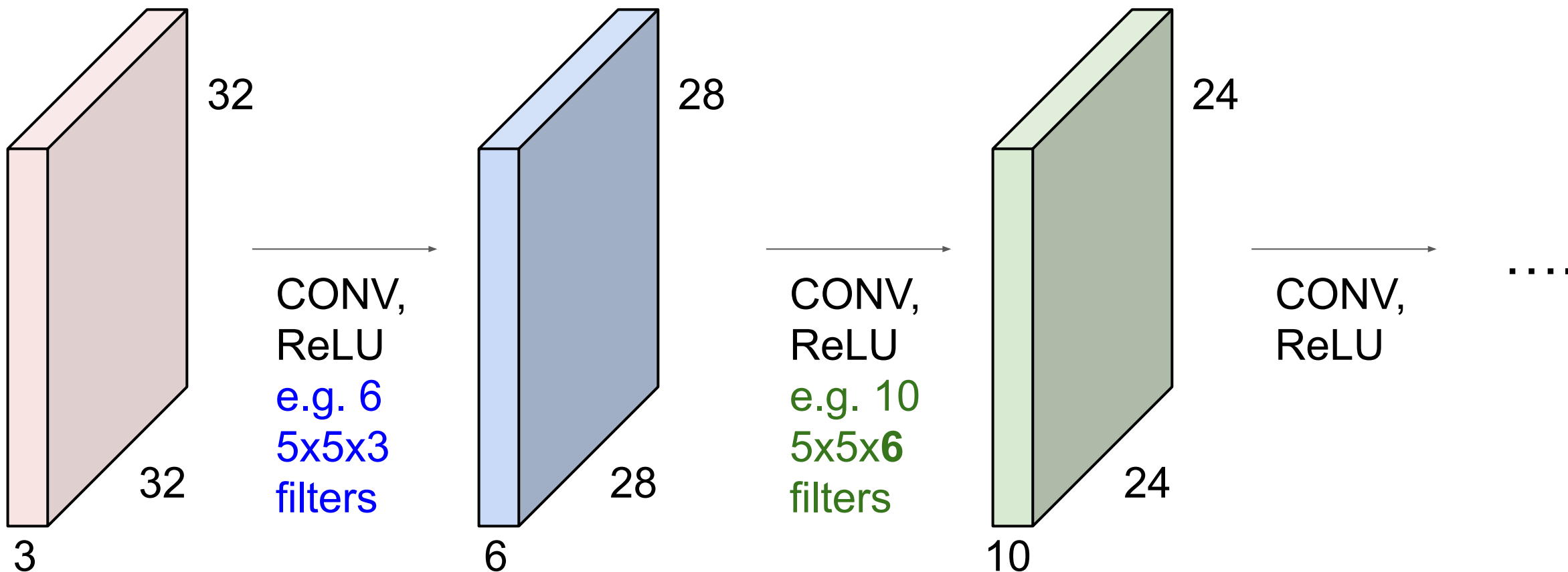


We stack these up to get a “new image” of size 28x28x6!

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

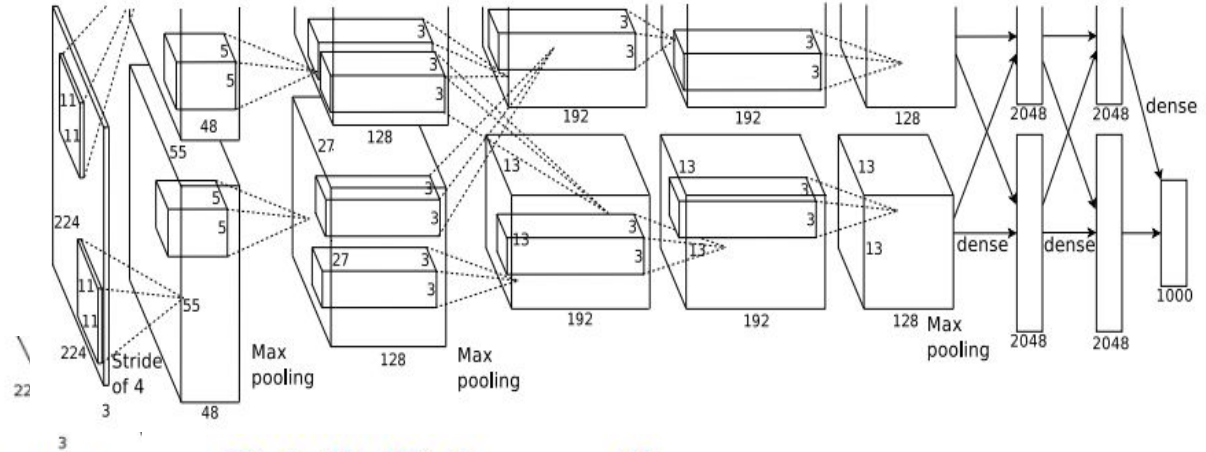


**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions



# CONVOLUTION NEURAL NETWORKS

# AlexNet



### Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

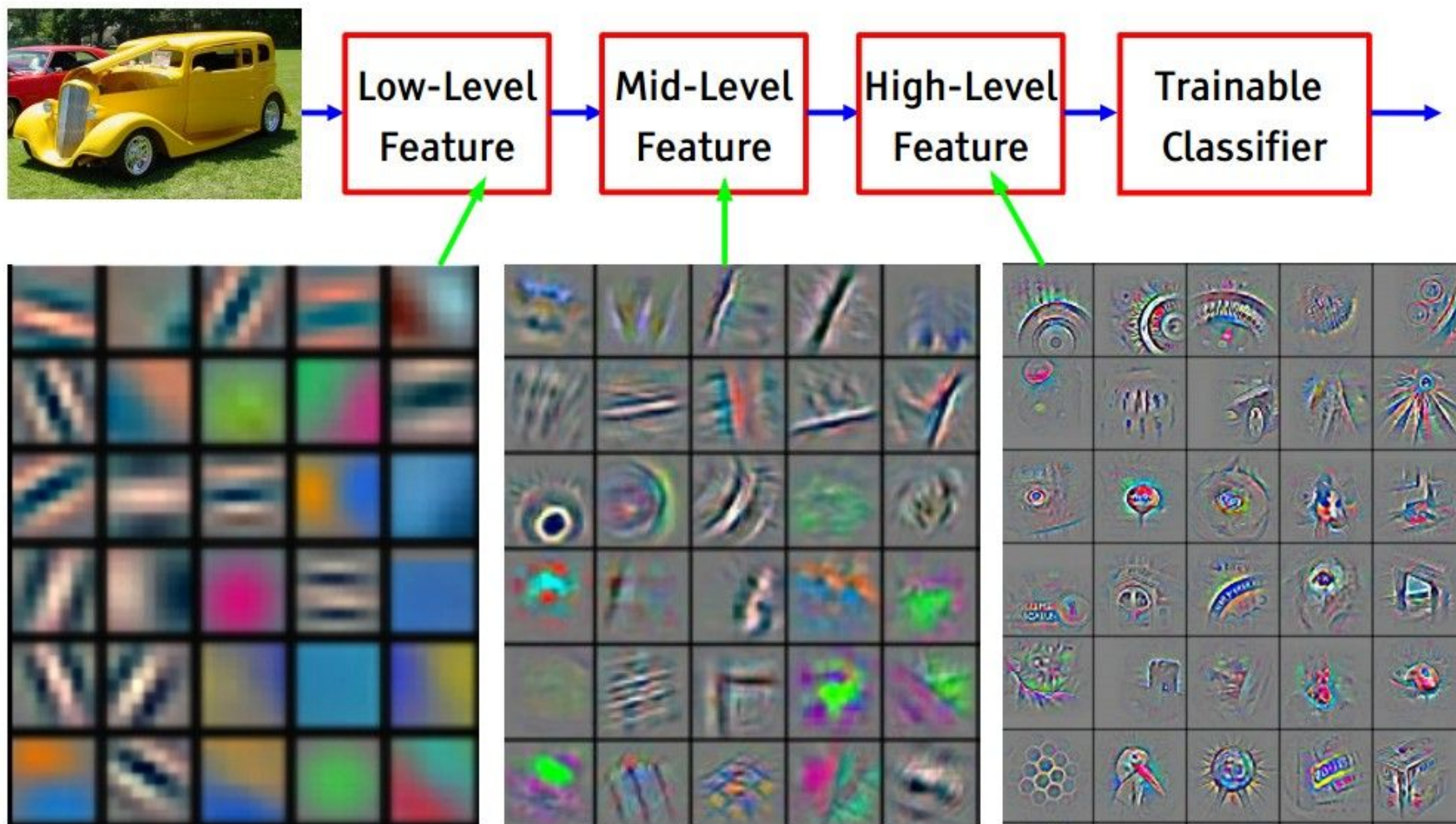
### Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate  $1e-2$ , reduced by 10 manually when val accuracy plateaus
- L2 weight decay  $5e-4$
- 7 CNN ensemble: 18.2%  $\rightarrow$  15.4%



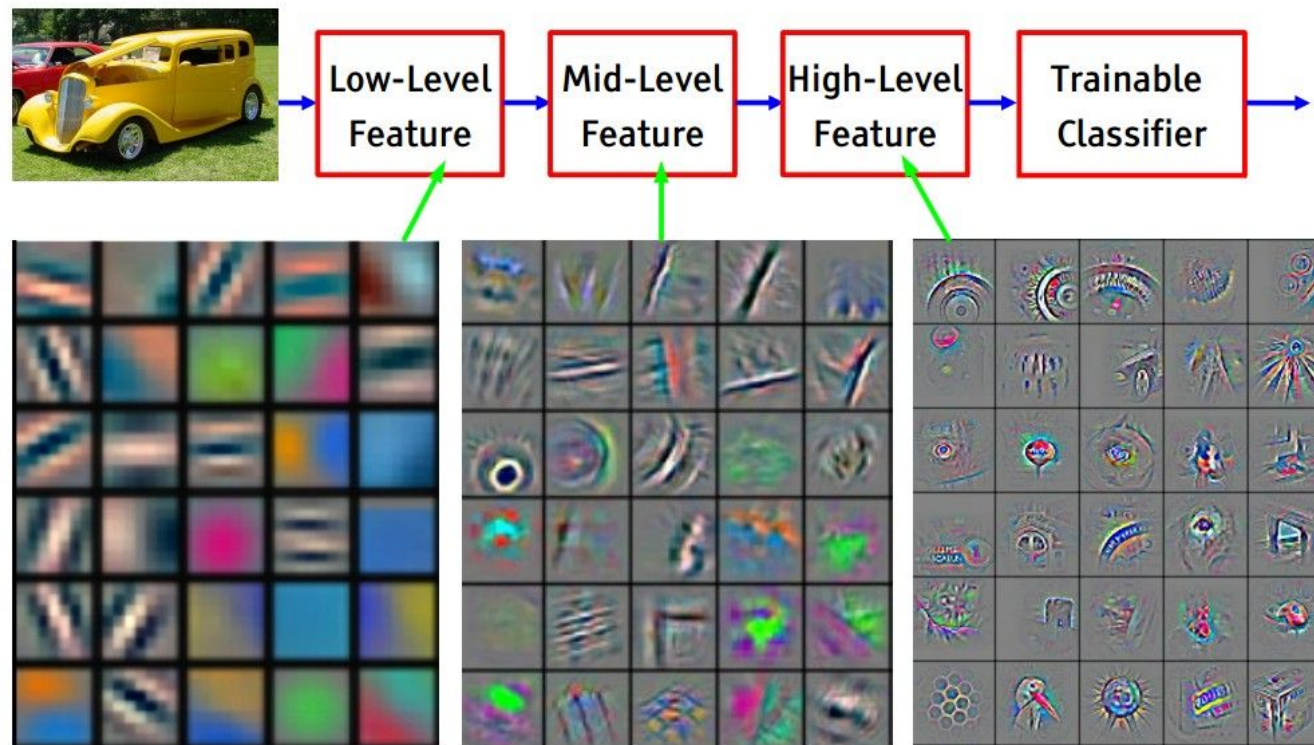
# Preview

[From recent Yann LeCun slides]



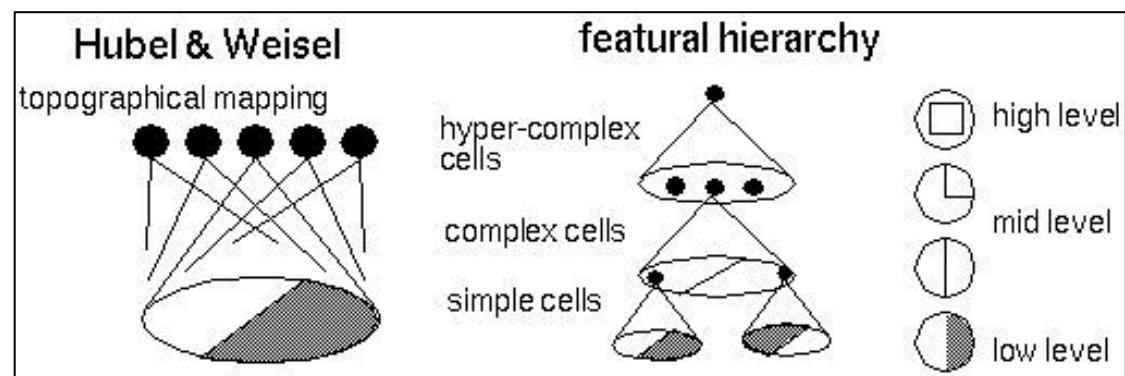
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

# Preview



*[From recent Yann LeCun slides]*

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]



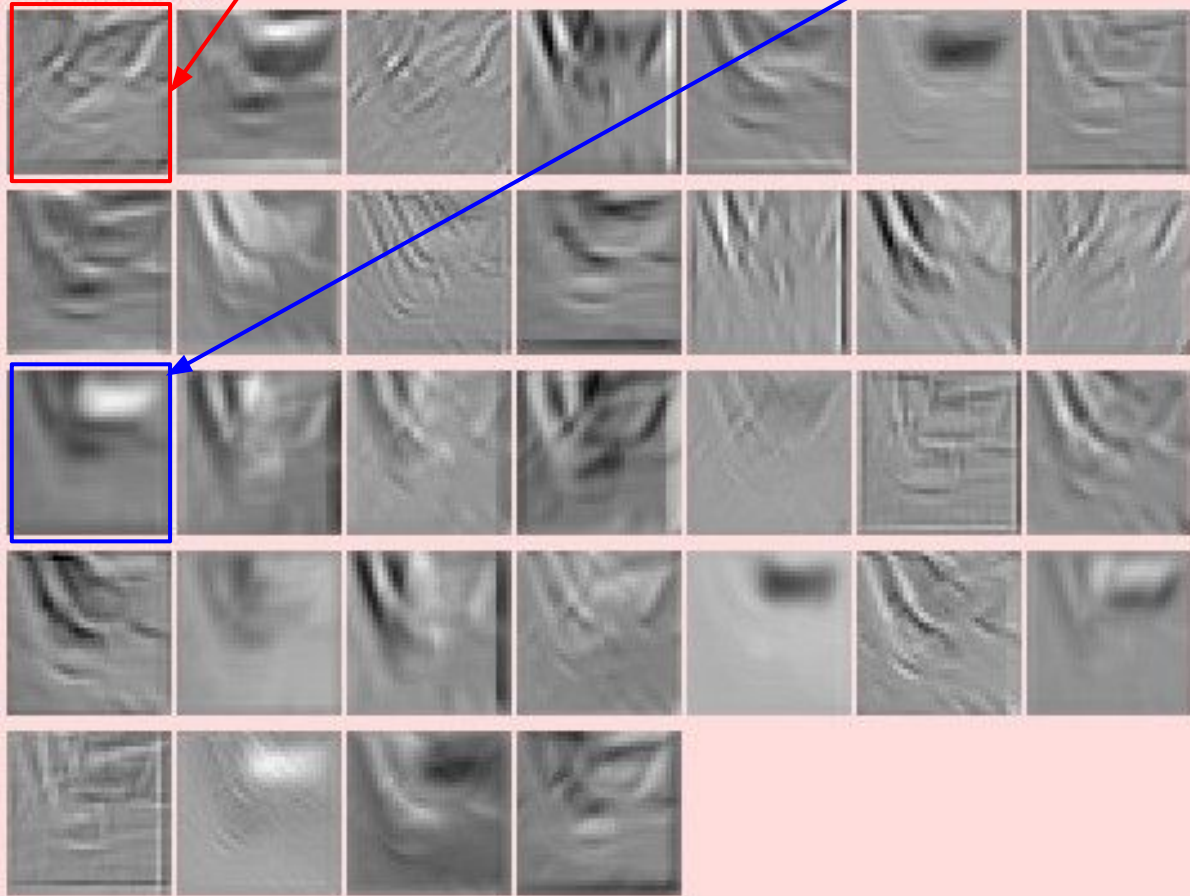




one filter =>  
one activation map

example 5x5 filters  
(32 total)

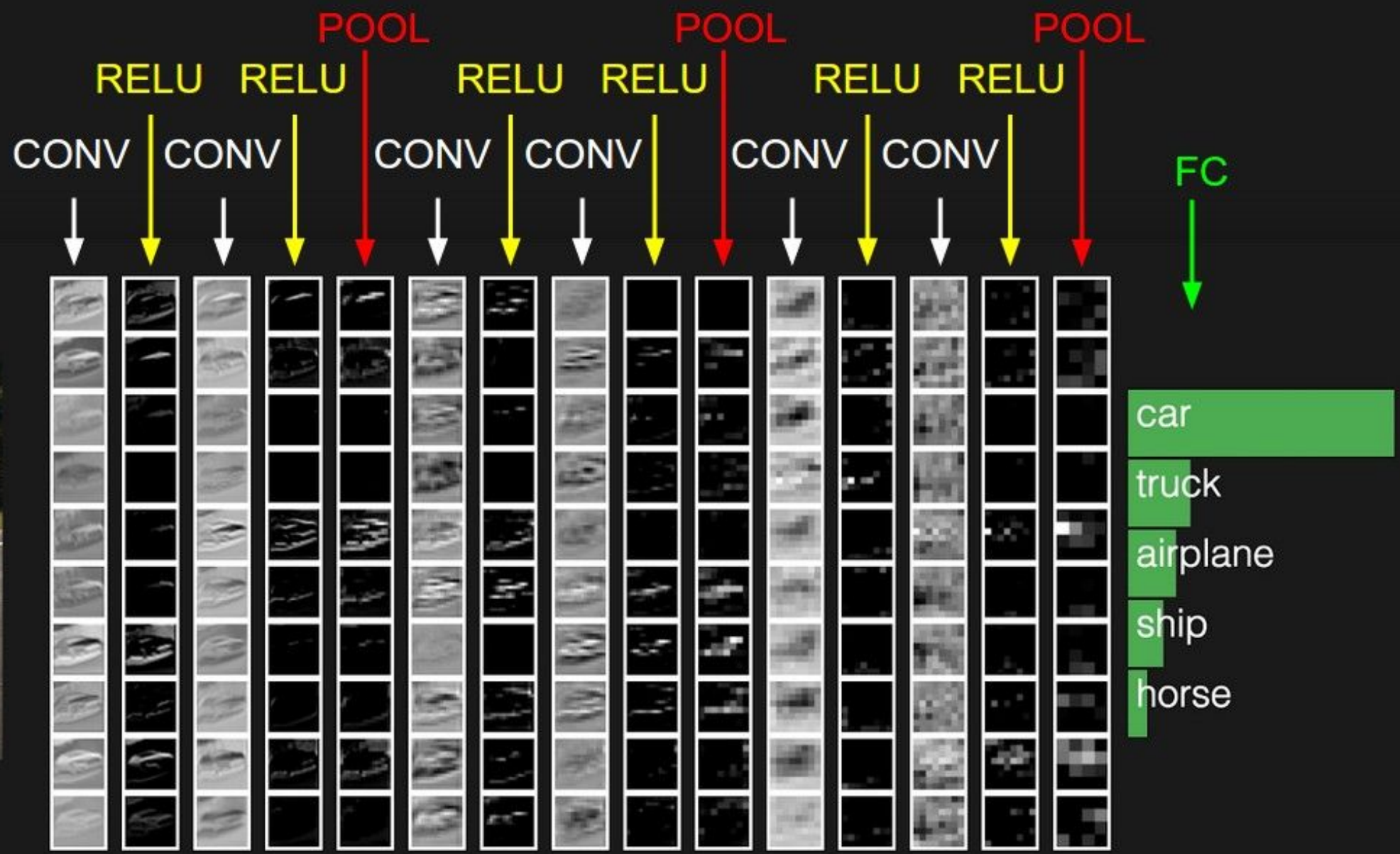
Activations:



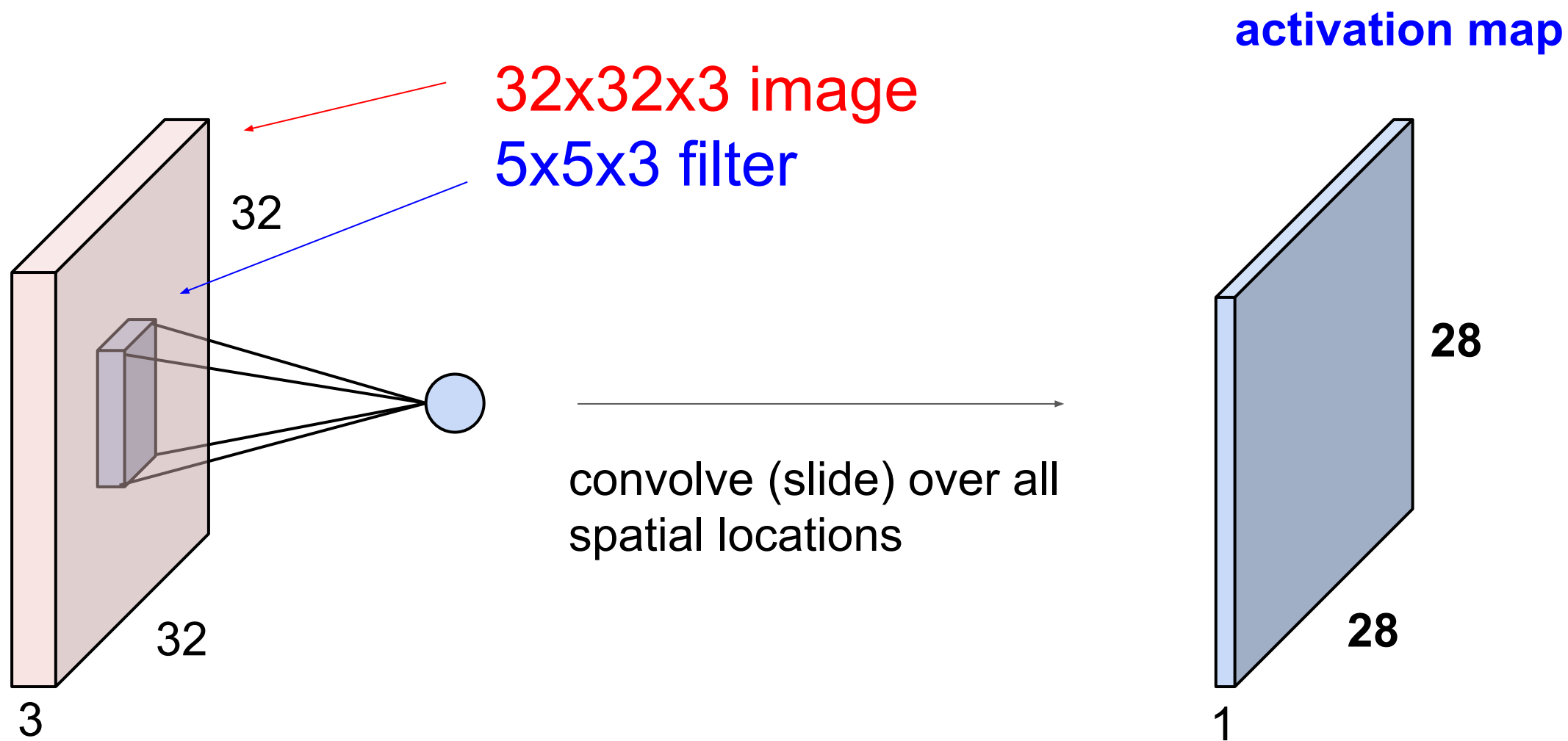
We call the layer convolutional  
because it is related to convolution  
of two signals:

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

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



## A closer look at spatial dimensions:



# A closer look at spatial dimensions:

7

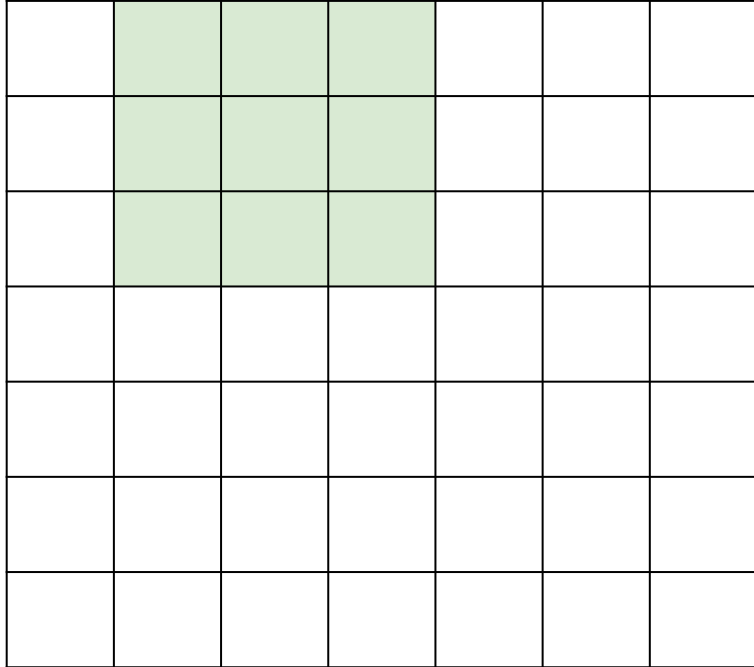

7

7x7 input (spatially)  
assume 3x3 filter



## A closer look at spatial dimensions:

7

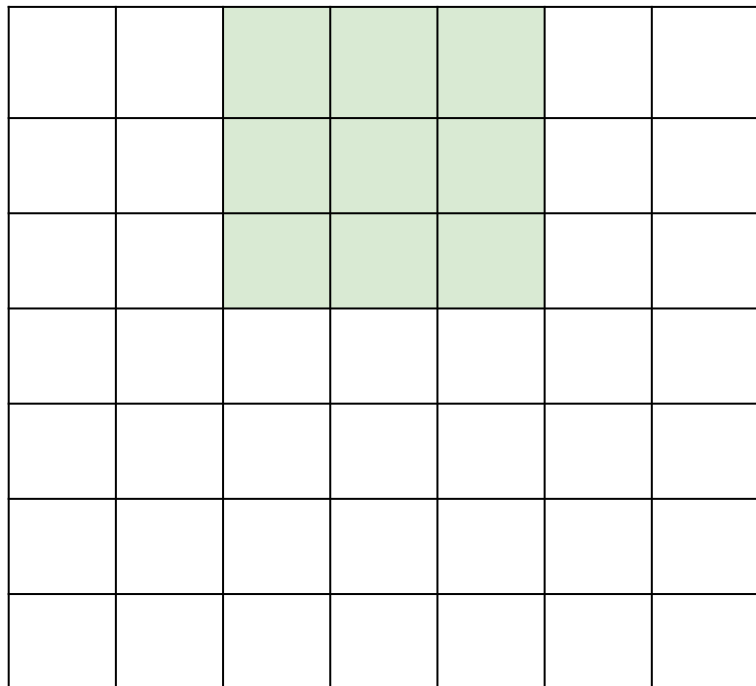


7x7 input (spatially)  
assume 3x3 filter

7

# A closer look at spatial dimensions:

7



7

7x7 input (spatially)  
assume 3x3 filter

# A closer look at spatial dimensions:

7


7

7x7 input (spatially)  
assume 3x3 filter

## A closer look at spatial dimensions:

7

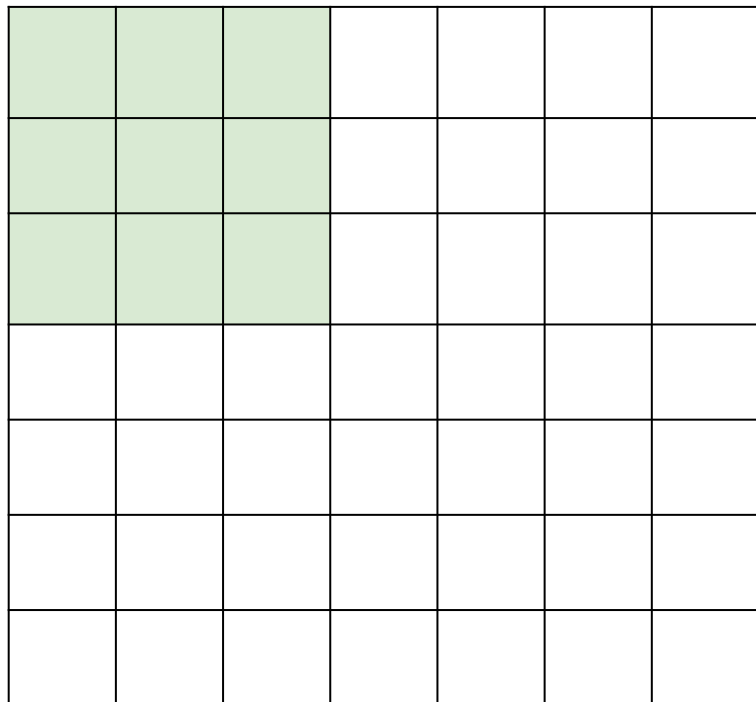

7

7x7 input (spatially)  
assume 3x3 filter

**=> 5x5 output**

A closer look at spatial dimensions:

7

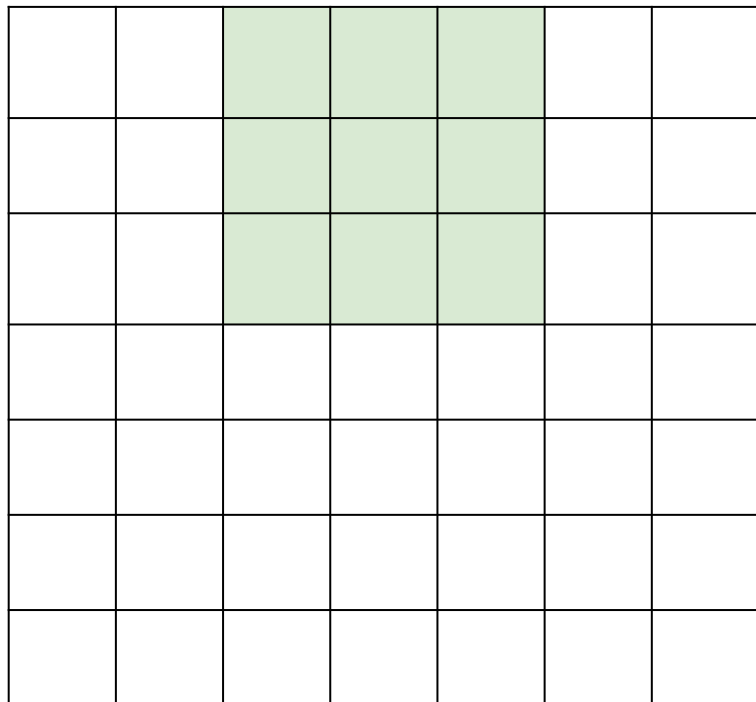


7

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

## A closer look at spatial dimensions:

7



7

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

A closer look at spatial dimensions:

7

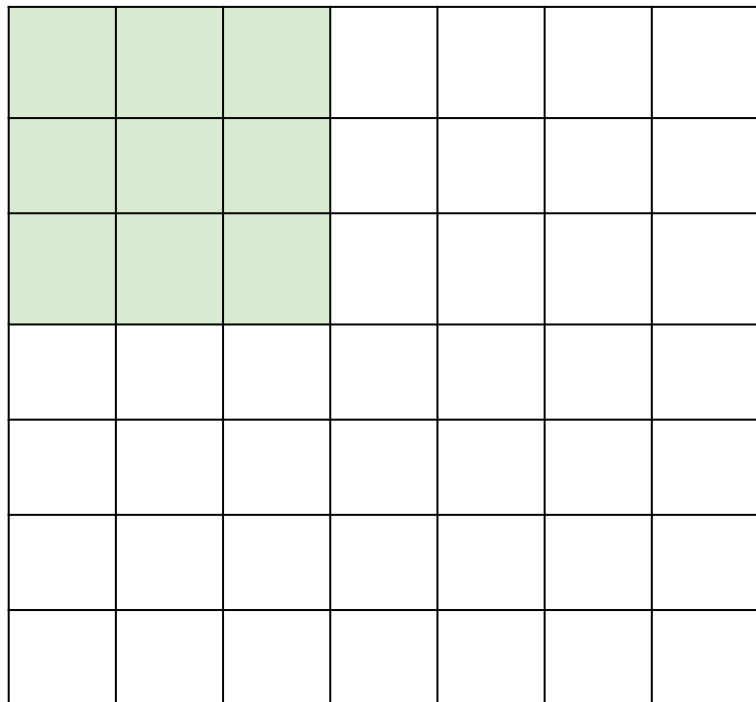

7

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



A closer look at spatial dimensions:

7



7

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.

N

			F			
	F					

N

Output size:  
 $(N - F) / \text{stride} + 1$

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

stride 1  $\Rightarrow (7 - 3) / 1 + 1 = 5$

stride 2  $\Rightarrow (7 - 3) / 2 + 1 = 3$

stride 3  $\Rightarrow (7 - 3) / 3 + 1 = 2.33 : \backslash$

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7

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

**pad with 1 pixel** border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

# In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

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

0	0	0	0	0	0			
0								
0								
0								
0								

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  $F \times F$ , and zero-padding with  $(F-1)/2$ . (will preserve size spatially)

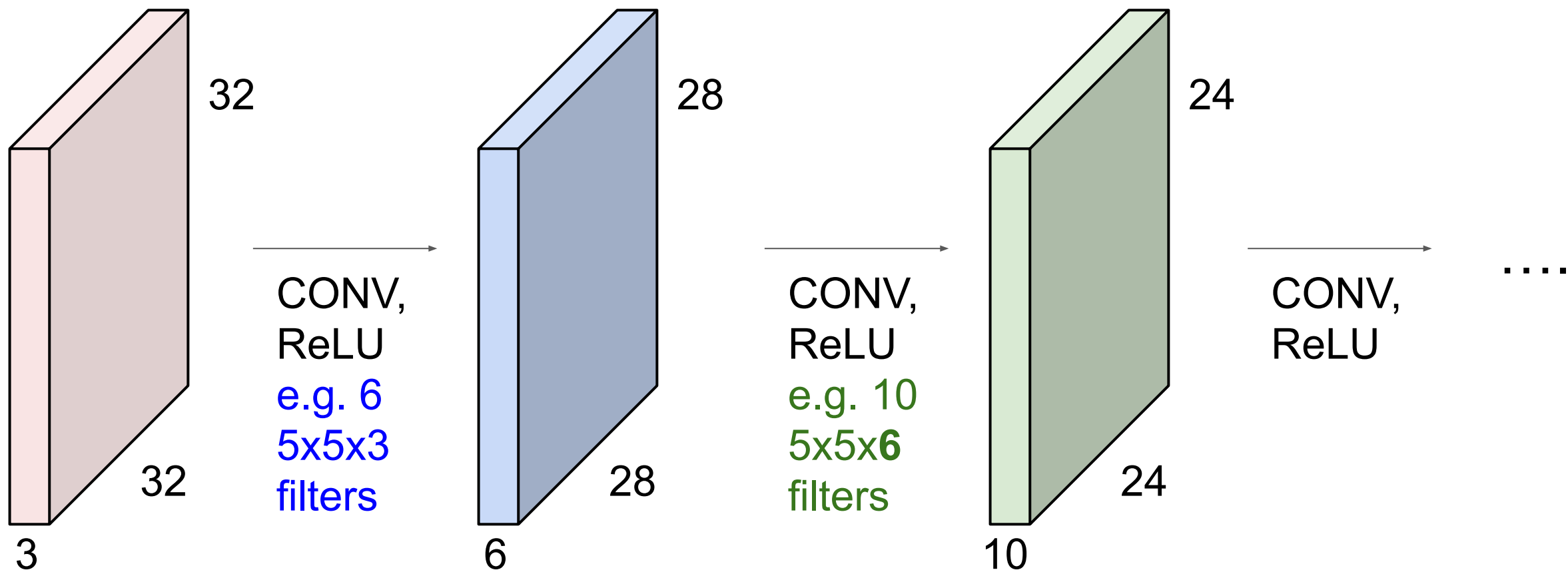
e.g.  $F = 3 \Rightarrow$  zero pad with 1

$F = 5 \Rightarrow$  zero pad with 2

$F = 7 \Rightarrow$  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.

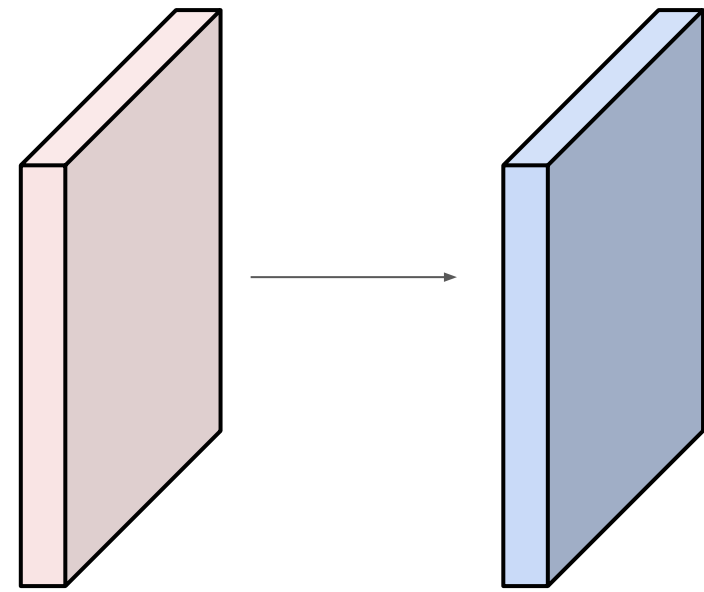


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Output volume size: ?





Examples time:

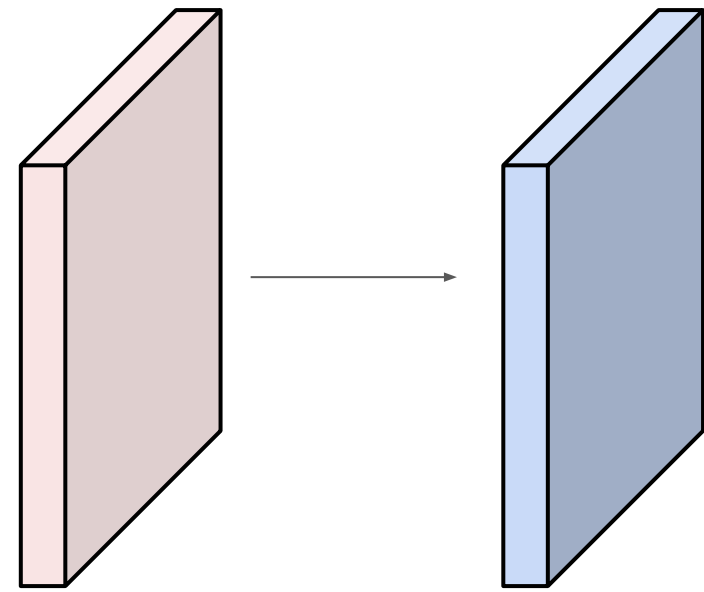
Input volume: **32x32x3**

**10** **5x5** filters with stride **1**, pad **2**

Output volume size:

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

**32x32x10**

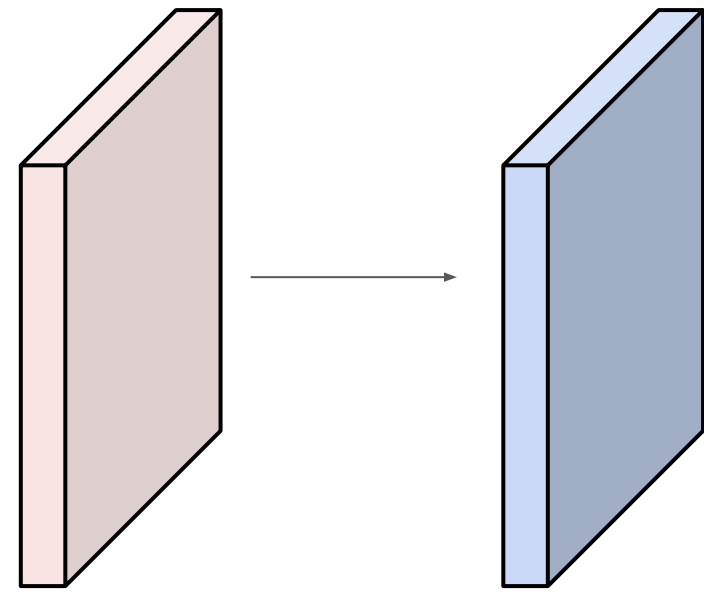


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

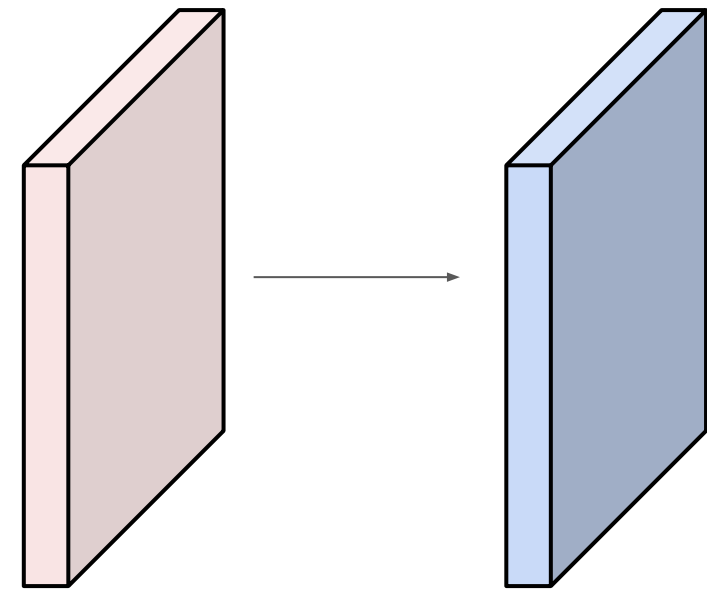
Number of parameters in this layer?



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.** 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:

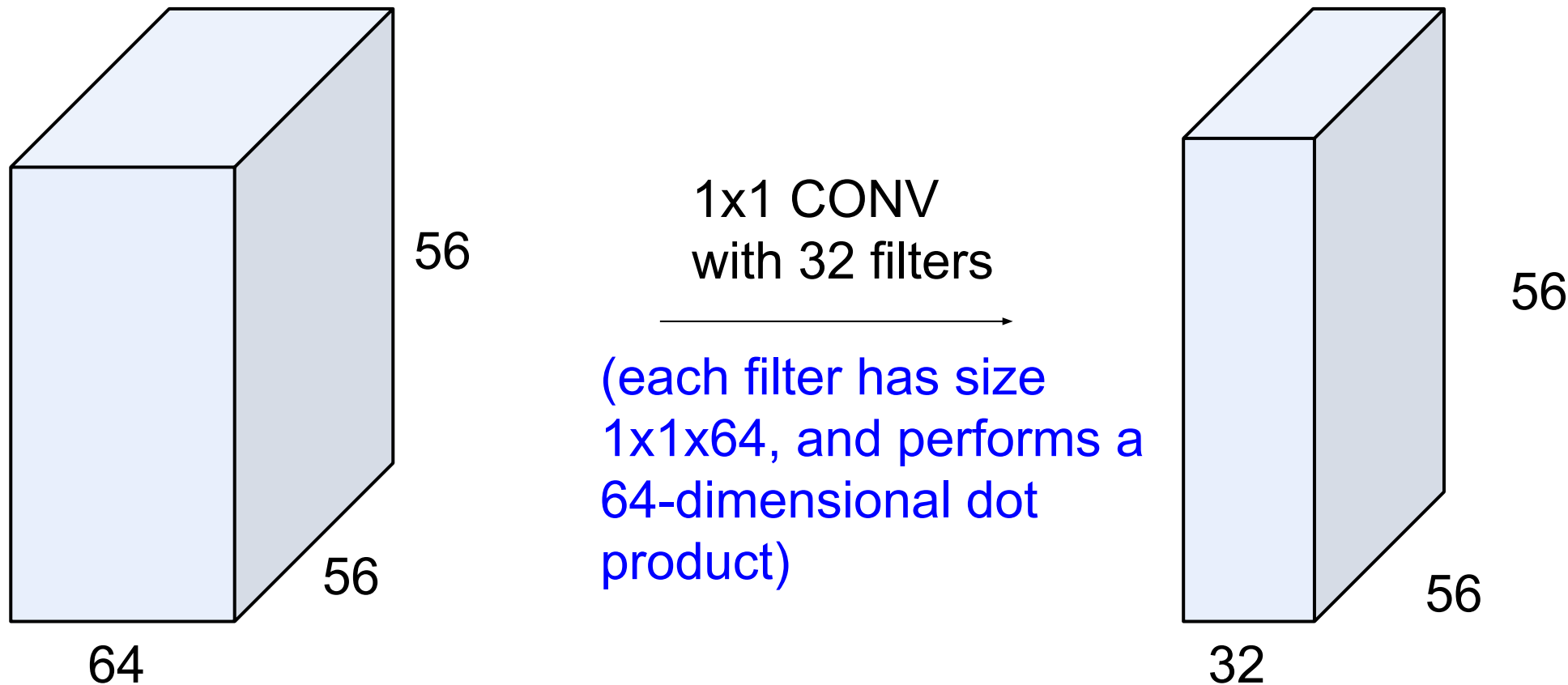
**Summary.** To summarize, the Conv Layer:

- 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.

$K = (\text{powers of 2, e.g. 32, 64, 128, 512})$

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

(btw, 1x1 convolution layers make perfect sense)





# Example: CONV layer in Torch

**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  $P$ .

## SpatialConvolution

```
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The `input` tensor in `forward(input)` is expected to be a 3D tensor (`nInputPlane` x `height` x `width`).

The parameters are the following:

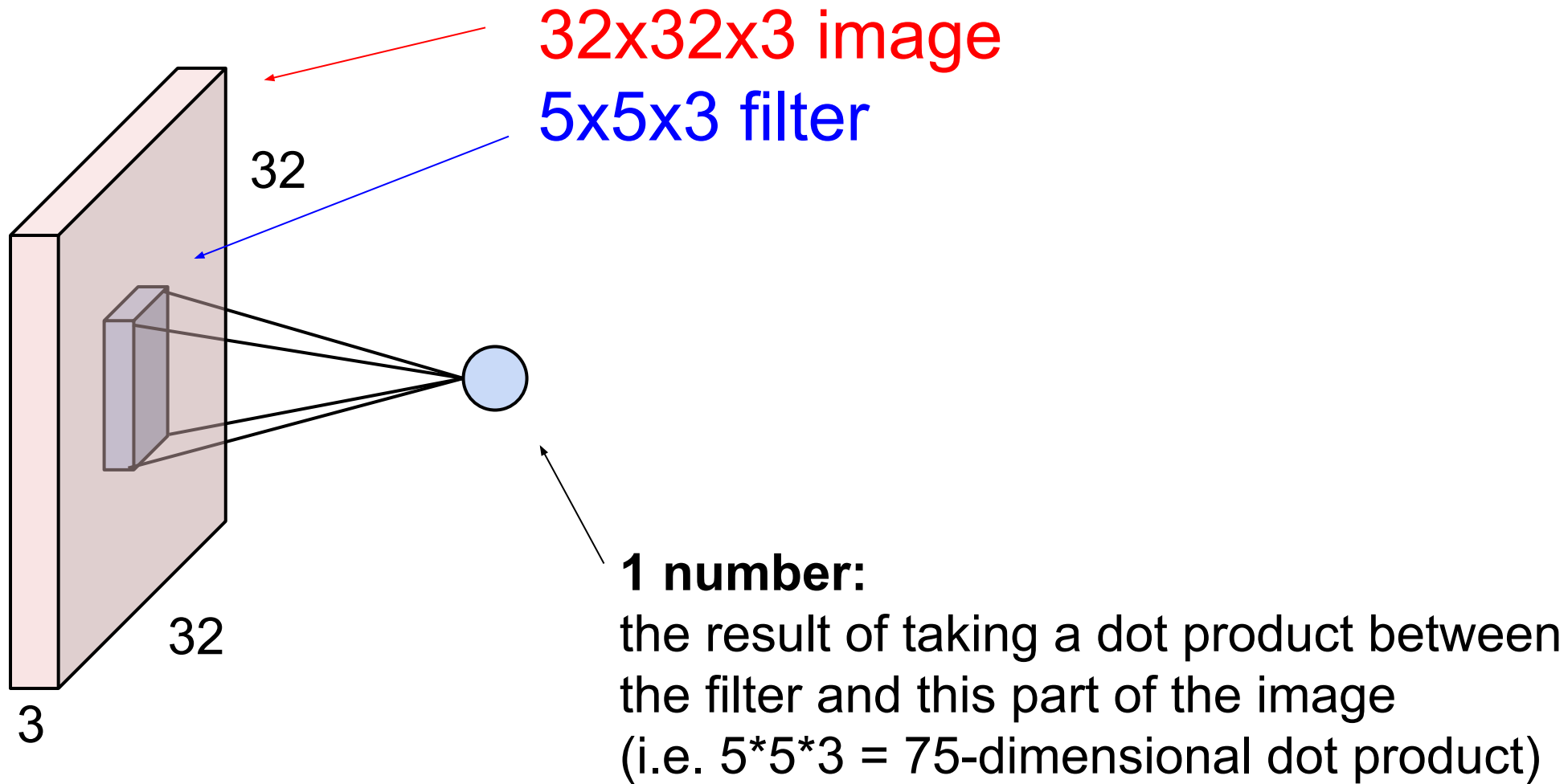
- `nInputPlane` : The number of expected input planes in the image given into `forward()`.
- `nOutputPlane` : The number of output planes the convolution layer will produce.
- `kW` : The kernel width of the convolution
- `kH` : The kernel height of the convolution
- `dW` : The step of the convolution in the width dimension. Default is `1`.
- `dH` : The step of the convolution in the height dimension. Default is `1`.
- `padW` : The additional zeros added per width to the input planes. Default is `0`, a good number is  $(kW-1)/2$ .
- `padH` : The additional zeros added per height to the input planes. Default is `padW`, a good number is  $(kH-1)/2$ .

Note that depending of the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane` x `height` x `width`, the output image size will be `nOutputPlane` x `oheight` x `owidth` where

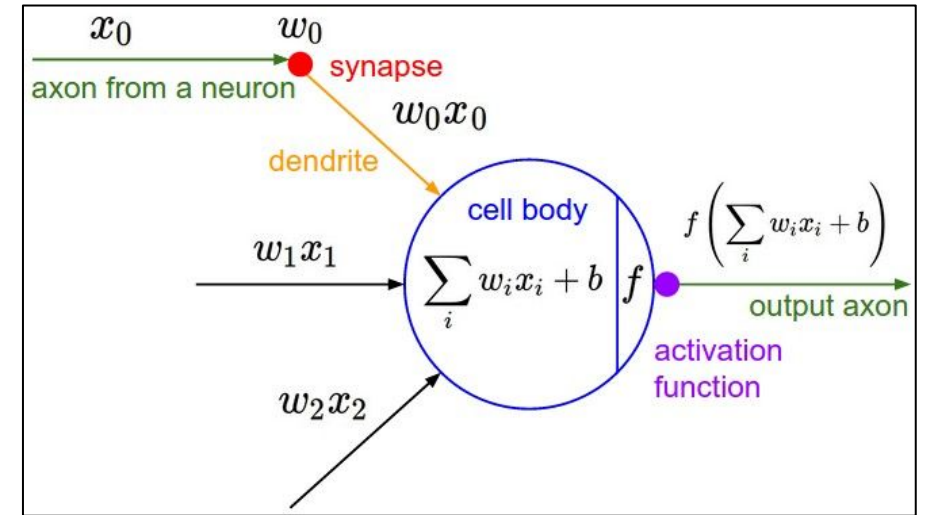
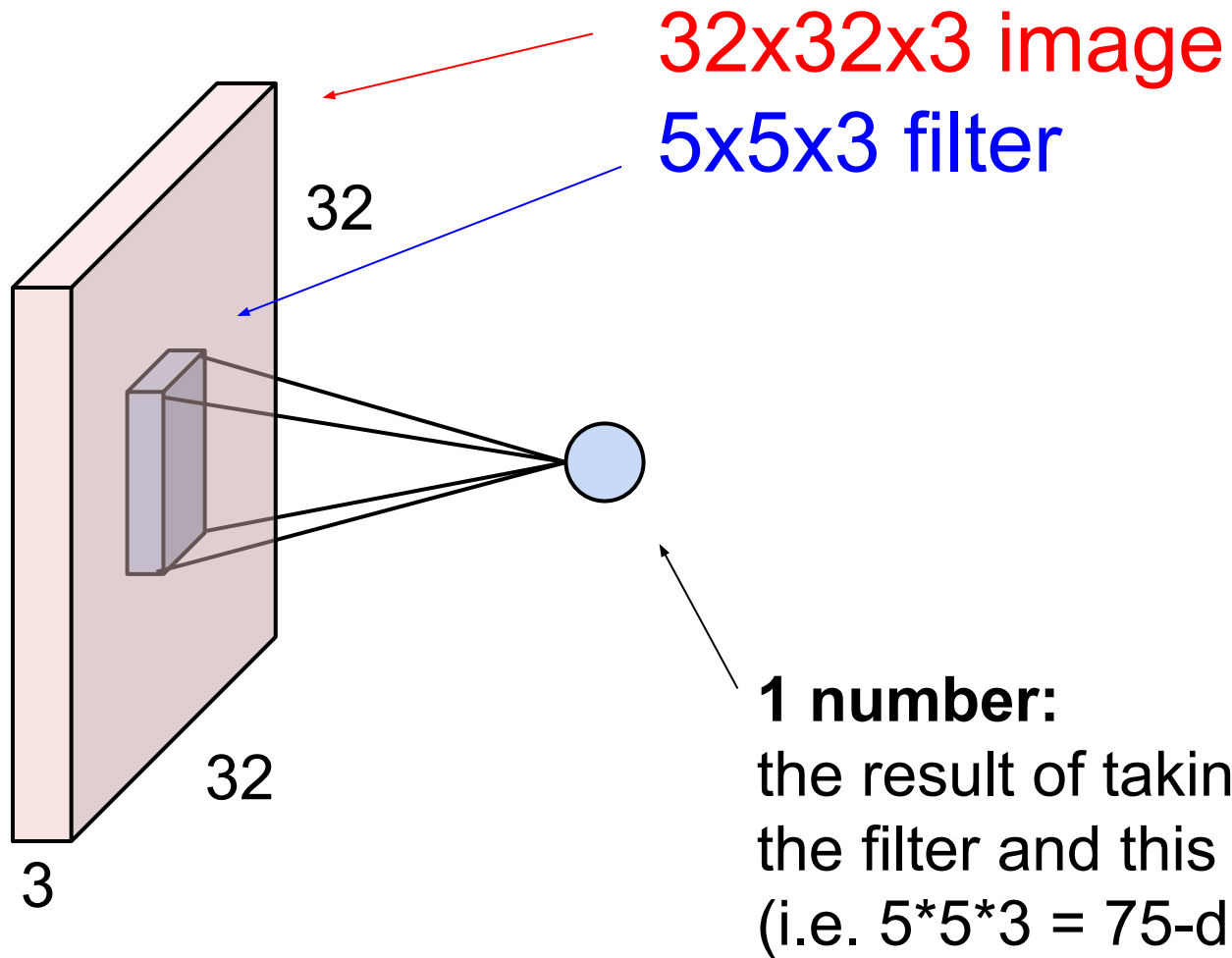
```
owidth = floor((width + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

# The brain/neuron view of CONV Layer



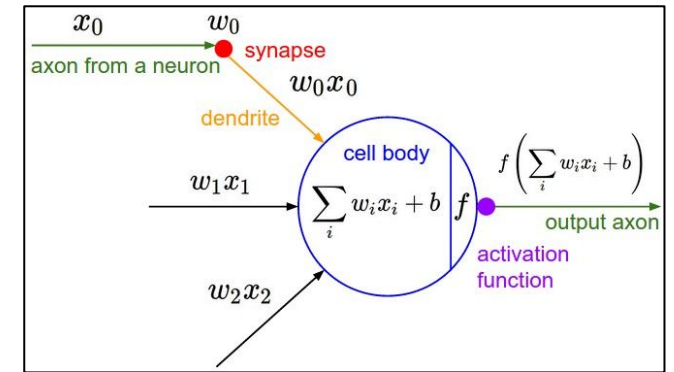
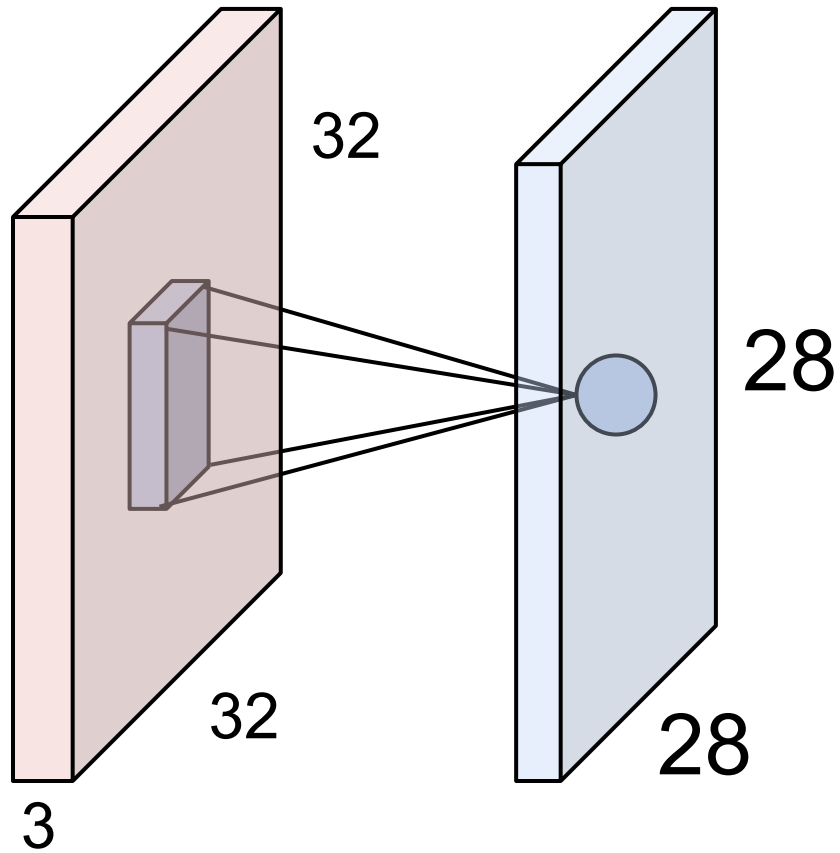


# The brain/neuron view of CONV Layer



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

# The brain/neuron view of CONV Layer

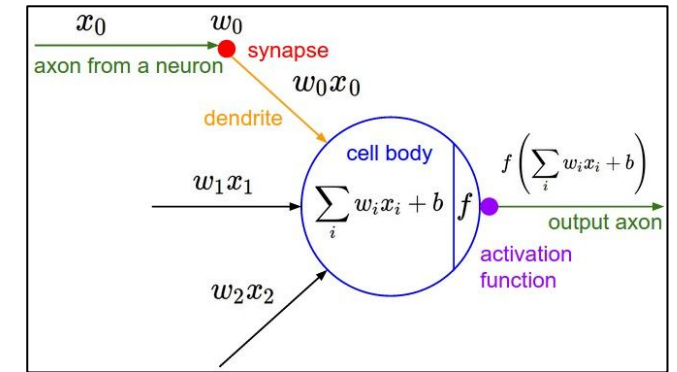
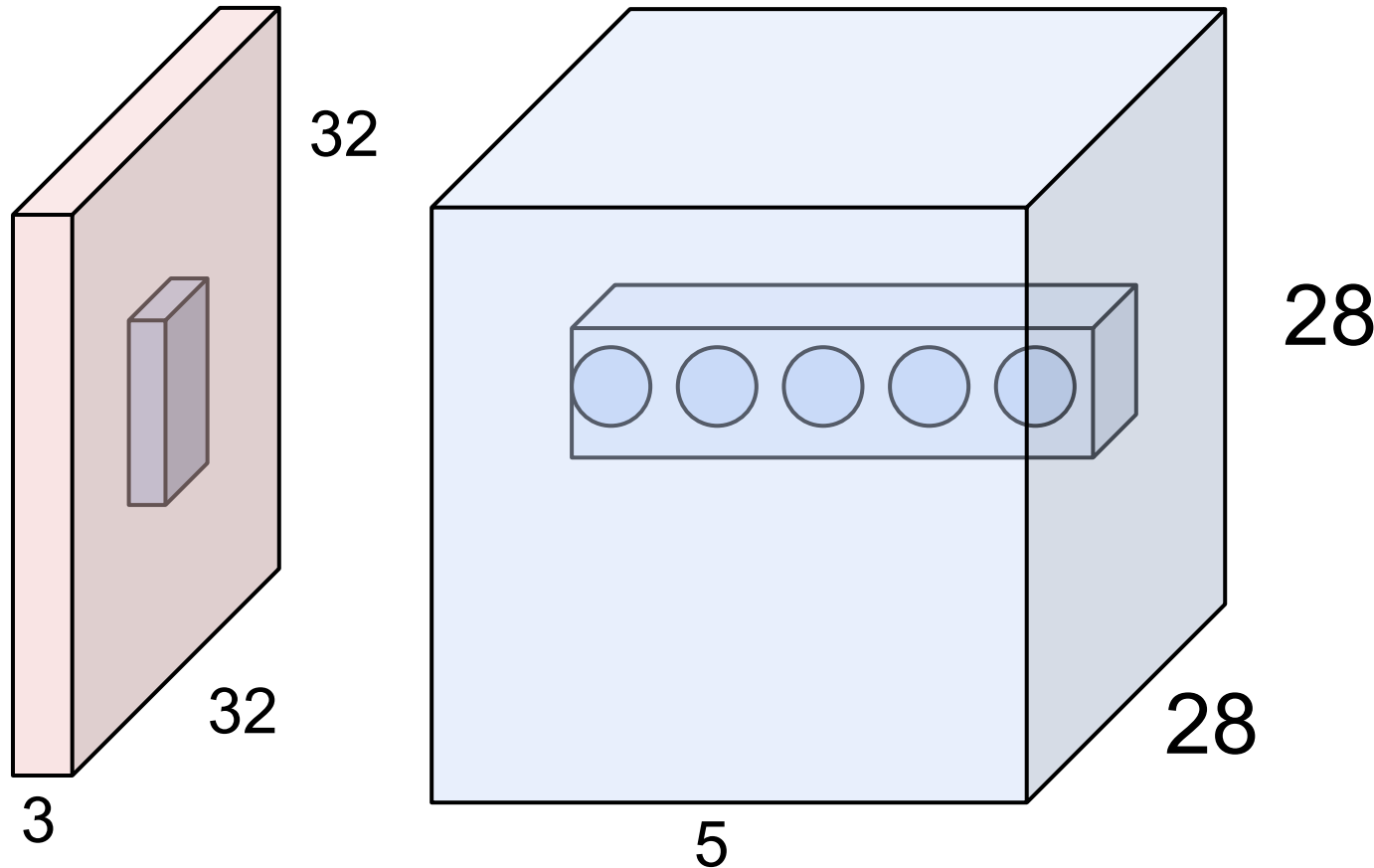


An activation map is a 28x28 sheet of neuron outputs:

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

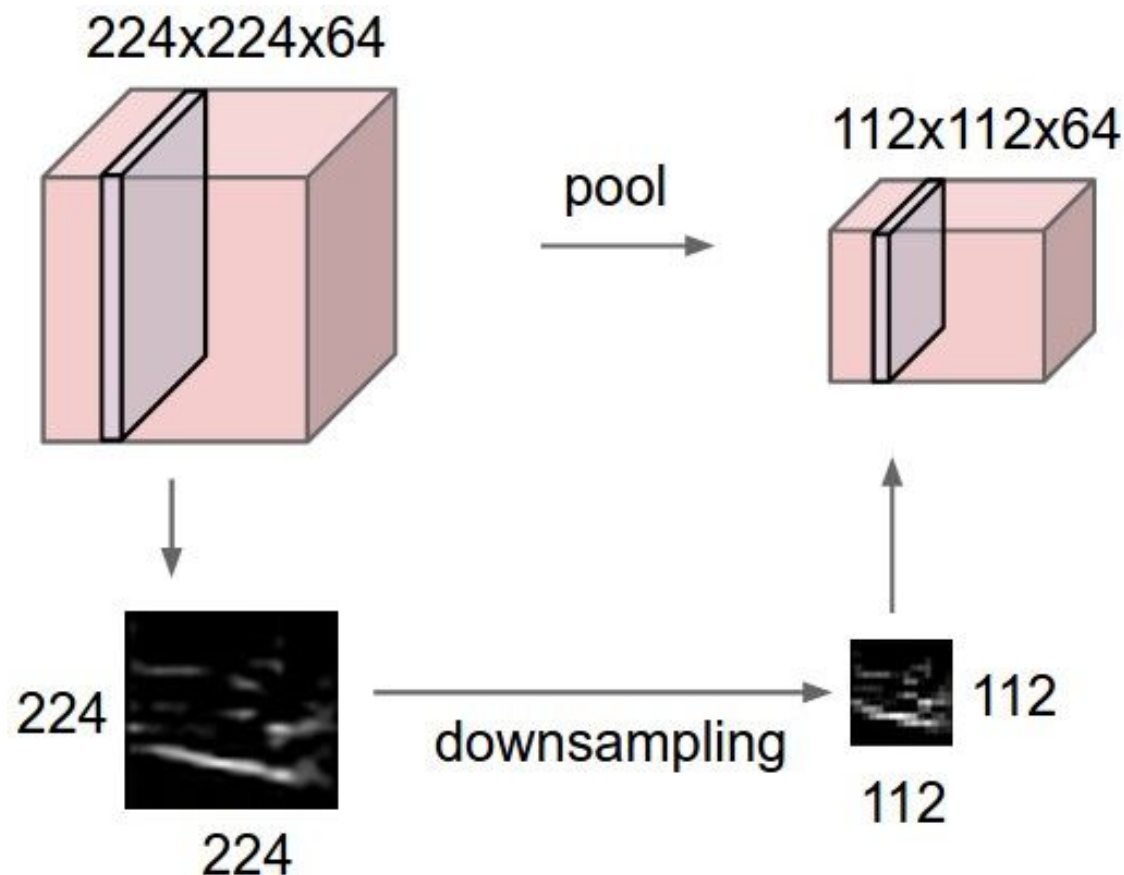


E.g. with 5 filters,  
CONV layer consists of  
neurons arranged in a 3D grid  
(28x28x5)

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

# 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

- 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:

$$F = 2, S = 2$$

$$F = 3, S = 2$$

- 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



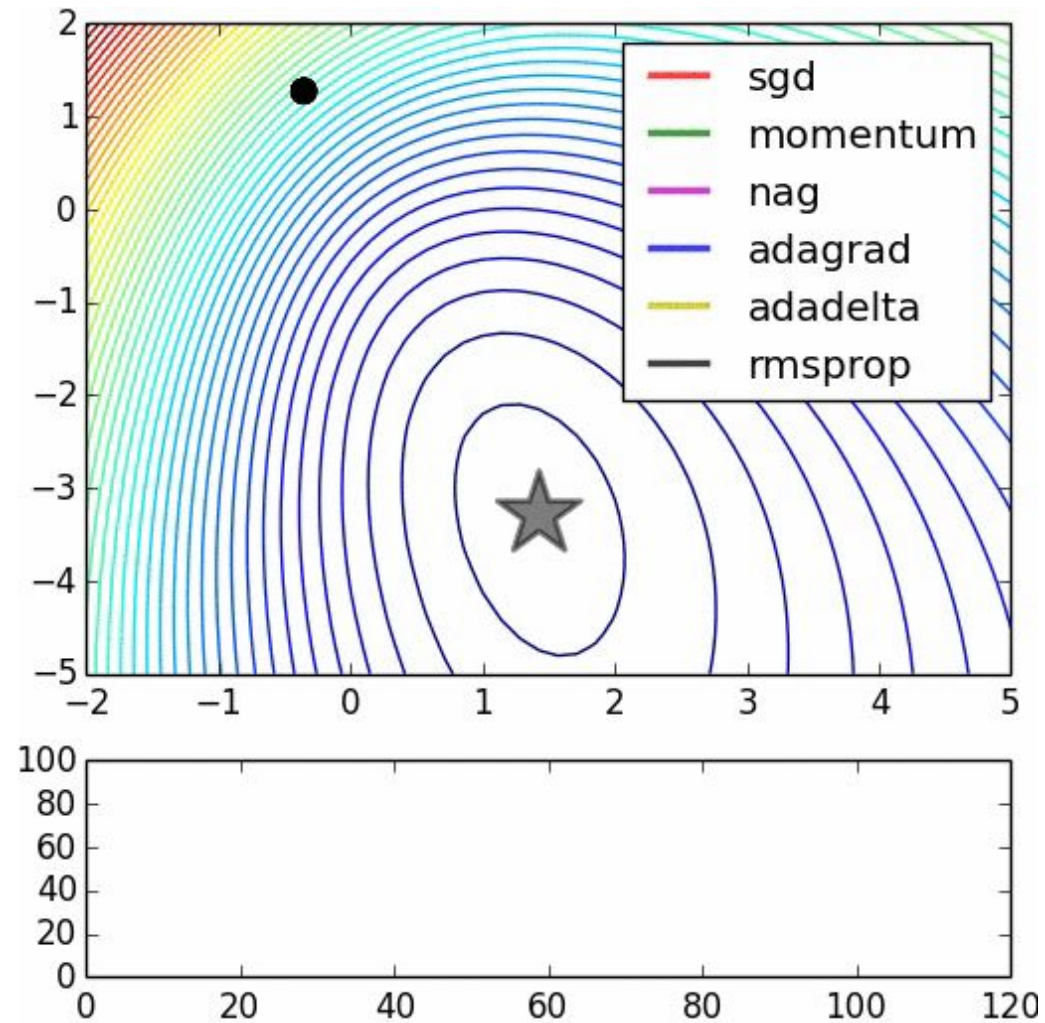


Image credits: Alec Radford

# Adam update

[Kingma and Ba, 2014]

```
# Adam
m,v = #... initialize caches to zeros
for t in xrange(1, big_number):
    dx = # ... evaluate gradient
    m = beta1*m + (1-beta1)*dx # update first moment
    v = beta2*v + (1-beta2)*(dx**2) # update second moment
    mb = m/(1-beta1**t) # correct bias
    vb = v/(1-beta2**t) # correct bias
    x += - learning_rate * mb / (np.sqrt(vb) + 1e-7)
```

momentum

bias correction  
(only relevant in first few  
iterations when t is small)

RMSProp-like

The bias correction compensates for the fact that  $m, v$  are initialized at zero and need some time to “warm up”.

# L-BFGS

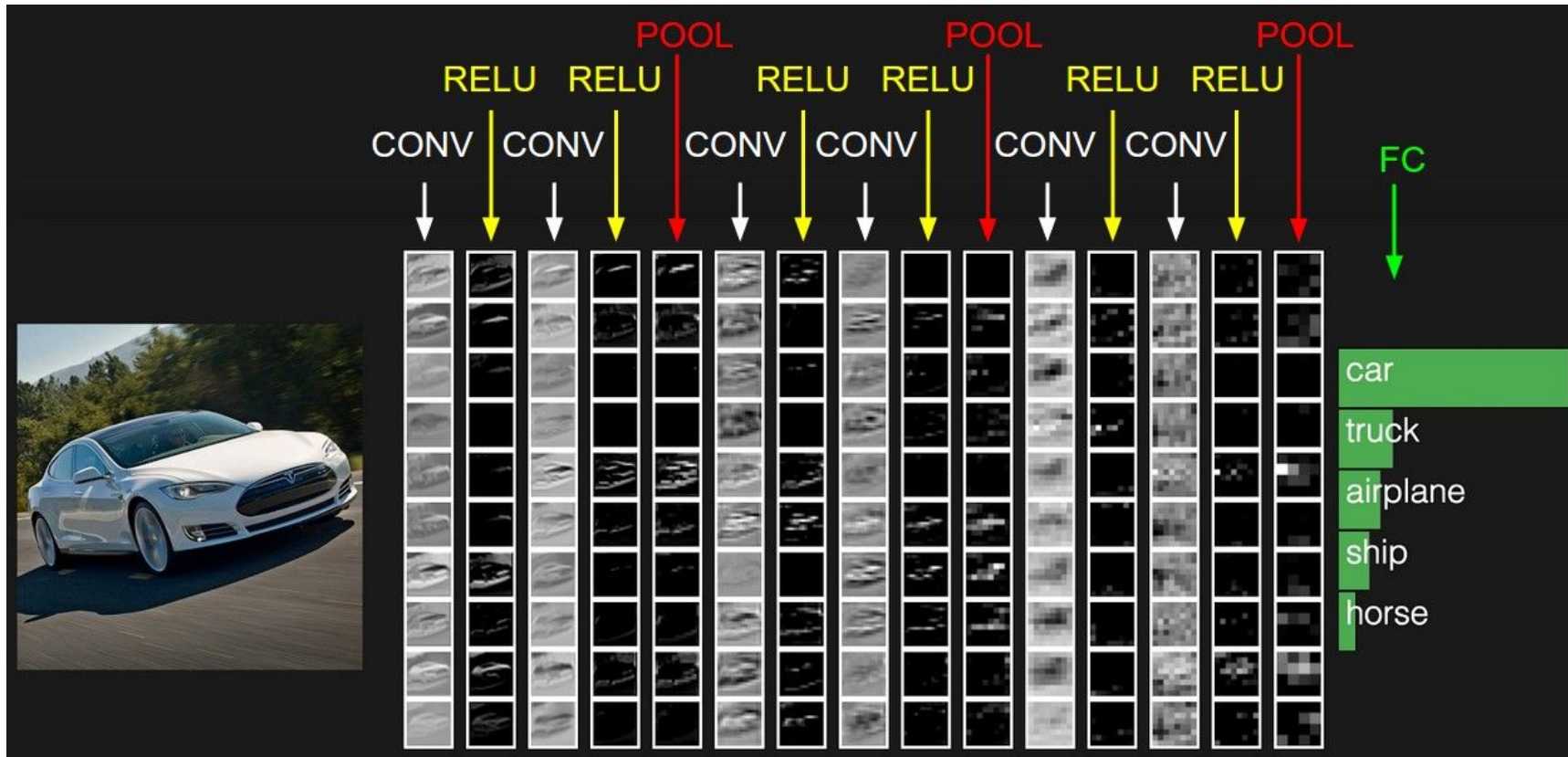
- **Usually works very well in full batch, deterministic mode** i.e. if you have a single, deterministic  $f(x)$  then L-BFGS will probably work very nicely
- **Does not transfer very well to mini-batch setting.** Gives bad results. Adapting L-BFGS to large-scale, stochastic setting is an active area of research.

# In practice:

- **Adam** is a good default choice in most cases
- If you can afford to do full batch updates then try out **L-BFGS** (and don't forget to disable all sources of noise)

# Fully Connected Layer (FC layer)

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

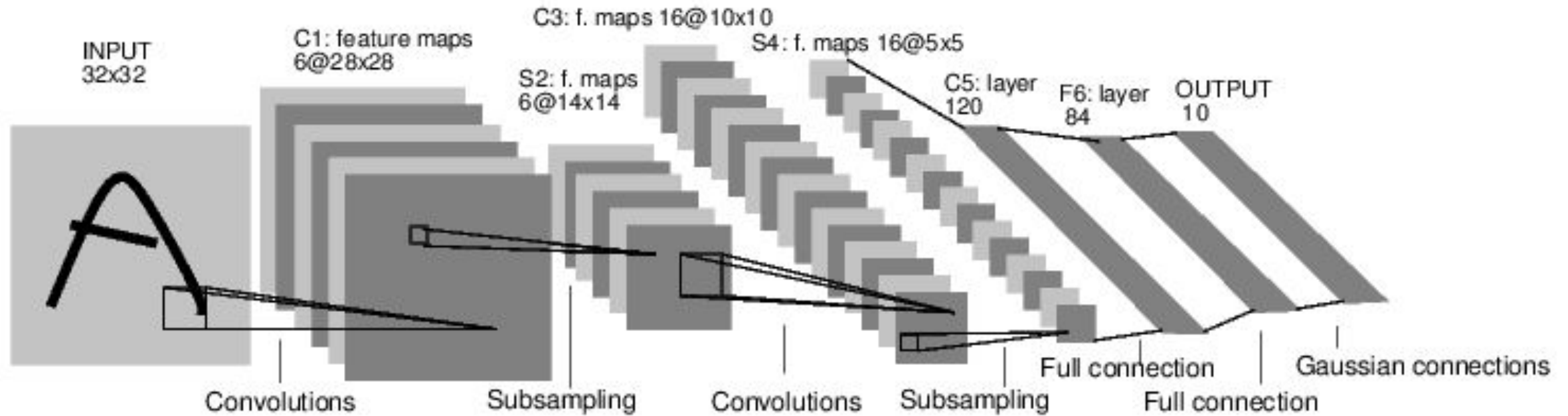


## [ConvNetJS demo: training on CIFAR-10]

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

# Case Study: LeNet-5

[LeCun et al., 1998]

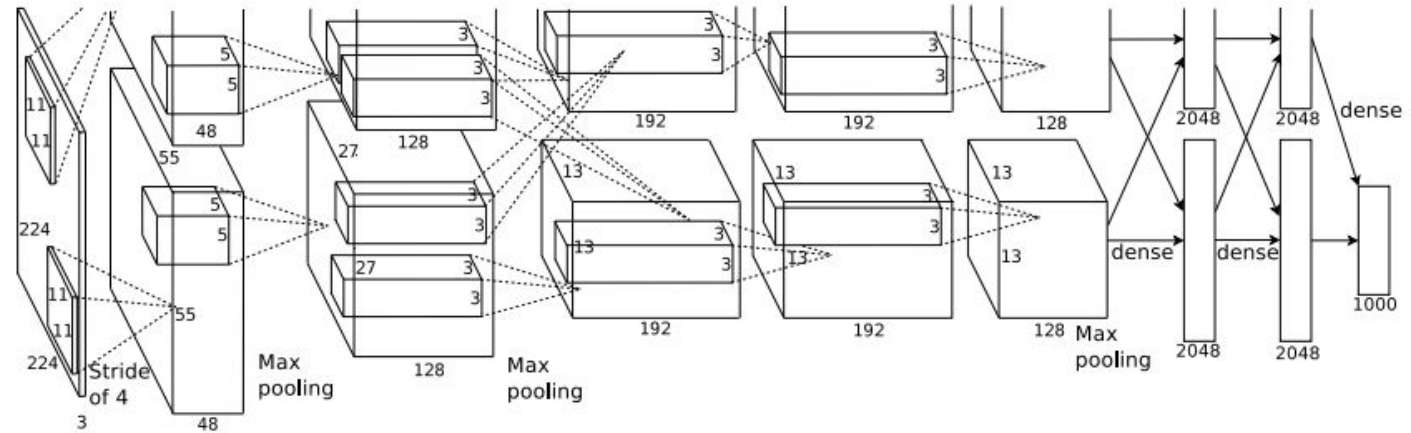


Conv filters were 5x5, applied at stride 1  
Subsampling (Pooling) layers were 2x2 applied at stride 2  
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]



# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

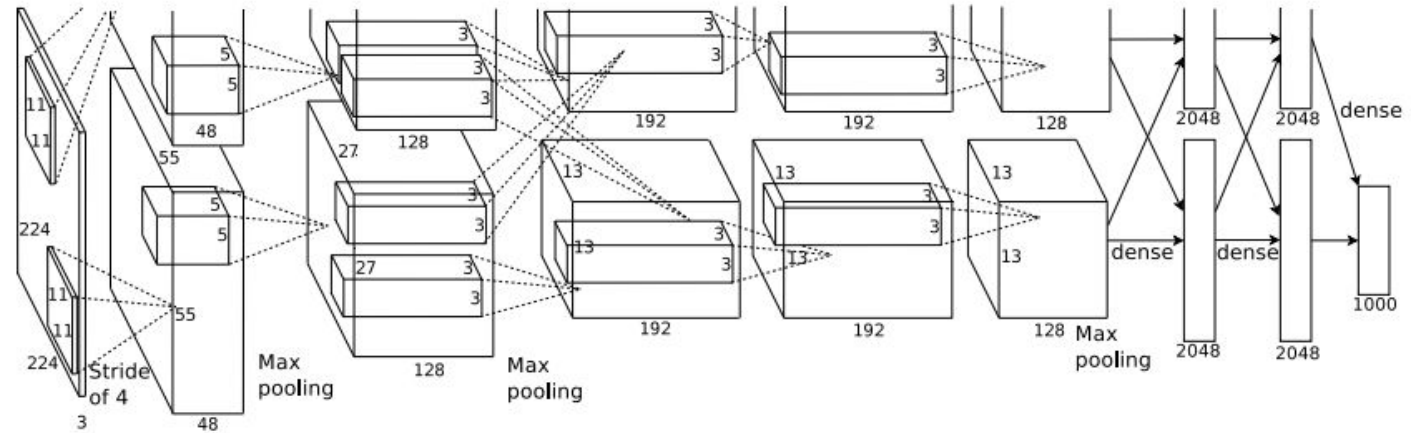
**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

Q: what is the output volume size? Hint:  $(227-11)/4+1 = 55$

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

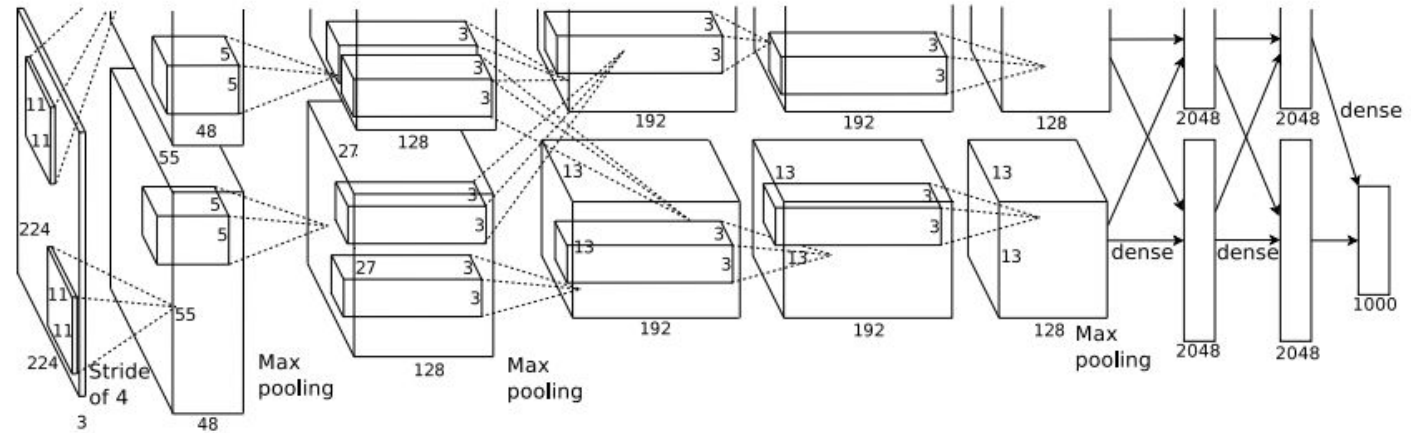
=>

Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

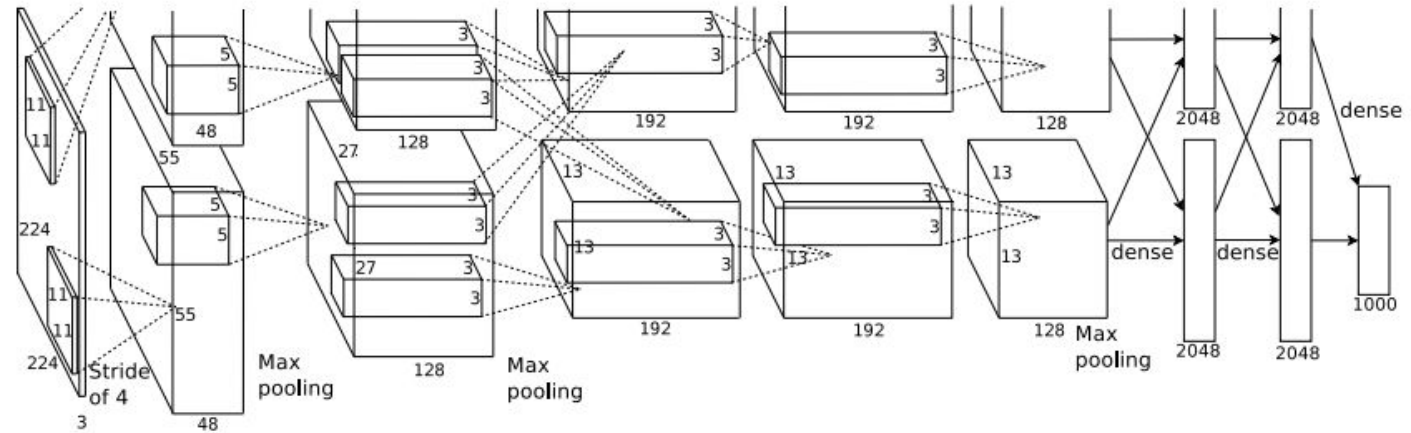
=>

Output volume **[55x55x96]**

Parameters:  $(11*11*3)*96 = 35K$

# Case Study: AlexNet

[Krizhevsky et al. 2012]



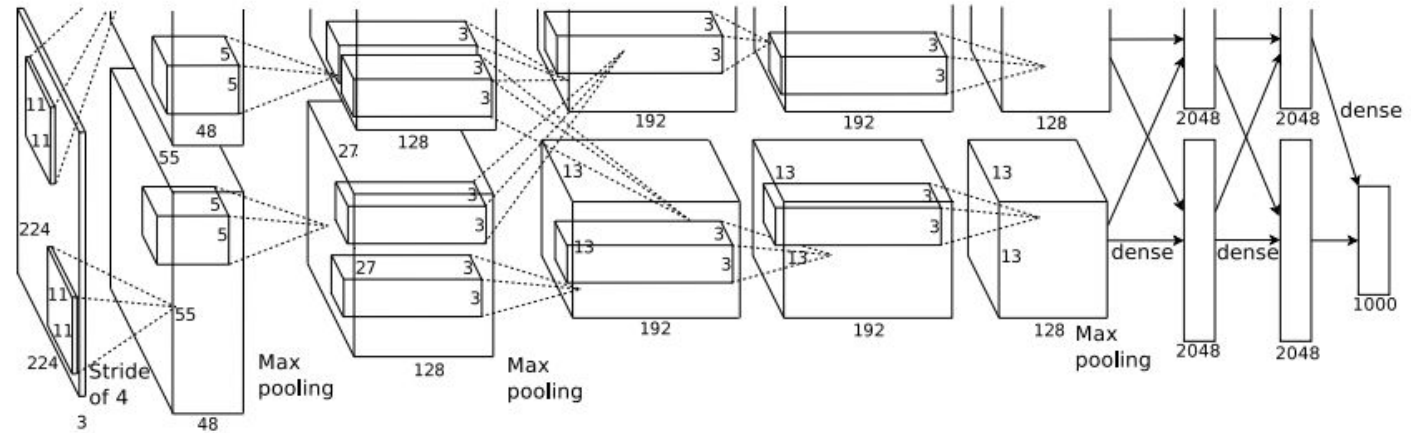
Input: 227x227x3 images  
After CONV1: 55x55x96

**Second layer (POOL1):** 3x3 filters applied at stride 2

Q: what is the output volume size? Hint:  $(55-3)/2+1 = 27$

# Case Study: AlexNet

[Krizhevsky et al. 2012]



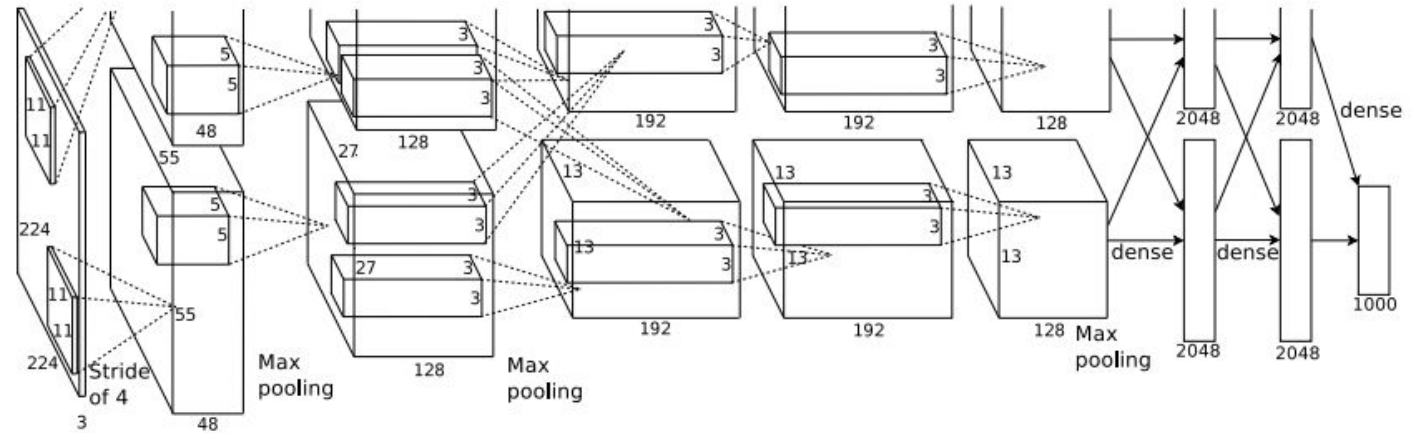
Input: 227x227x3 images  
After CONV1: 55x55x96

**Second layer (POOL1):** 3x3 filters applied at stride 2  
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

# Case Study: AlexNet

[Krizhevsky et al. 2012]

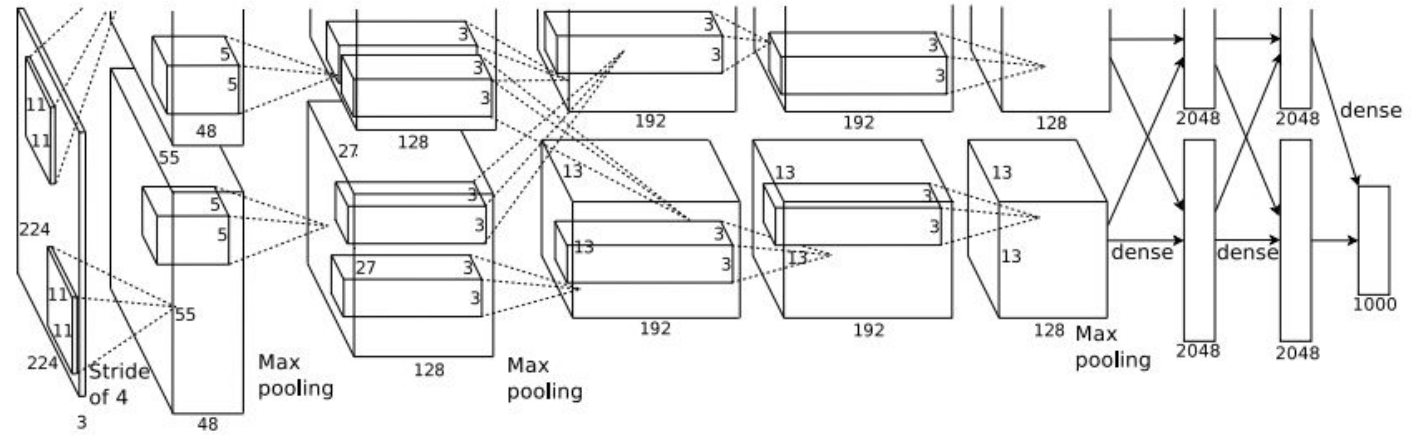


Input: 227x227x3 images  
After CONV1: 55x55x96

**Second layer (POOL1):** 3x3 filters applied at stride 2  
Output volume: 27x27x96  
Parameters: 0!

# Case Study: AlexNet

[Krizhevsky et al. 2012]



Input: 227x227x3 images

After CONV1: 55x55x96

After POOL1: 27x27x96

...



# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

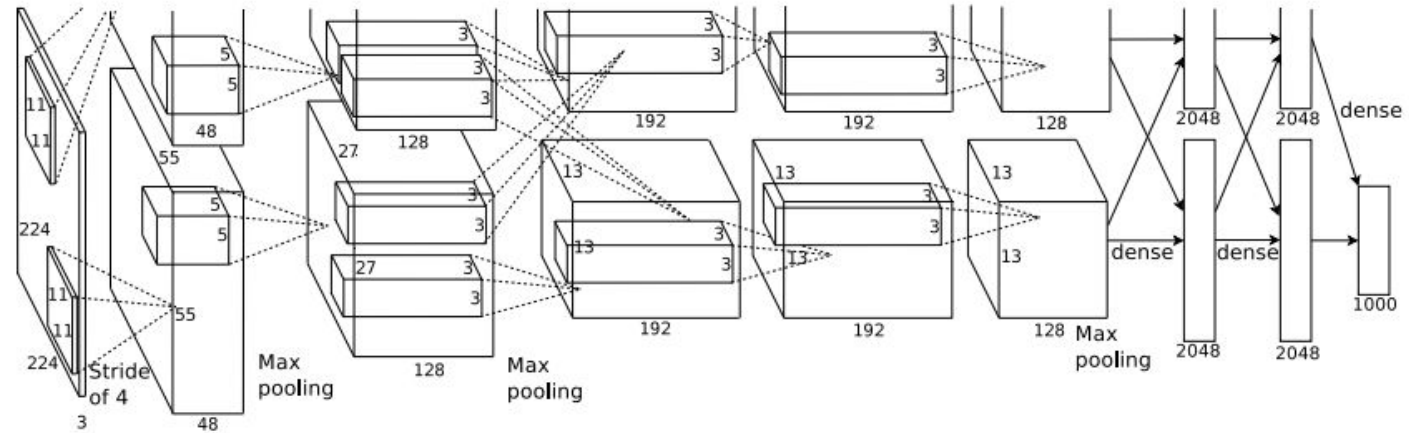
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)





# Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] **CONV1**: 96 11x11 filters at stride 4, pad 0

[27x27x96] **MAX POOL1**: 3x3 filters at stride 2

[27x27x96] **NORM1**: Normalization layer

[27x27x256] **CONV2**: 256 5x5 filters at stride 1, pad 2

[13x13x256] **MAX POOL2**: 3x3 filters at stride 2

[13x13x256] **NORM2**: Normalization layer

[13x13x384] **CONV3**: 384 3x3 filters at stride 1, pad 1

[13x13x384] **CONV4**: 384 3x3 filters at stride 1, pad 1

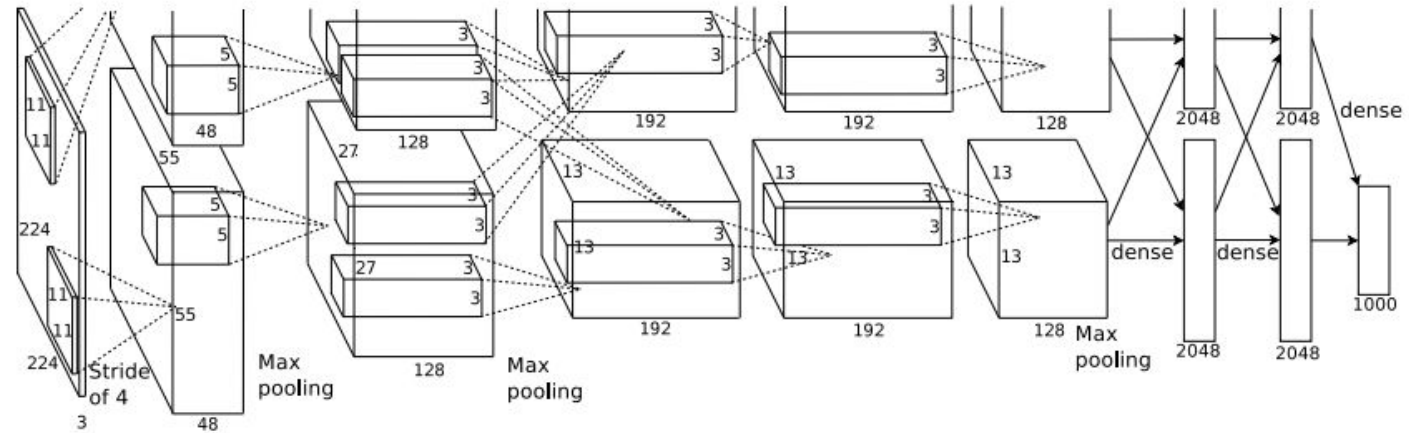
[13x13x256] **CONV5**: 256 3x3 filters at stride 1, pad 1

[6x6x256] **MAX POOL3**: 3x3 filters at stride 2

[4096] **FC6**: 4096 neurons

[4096] **FC7**: 4096 neurons

[1000] **FC8**: 1000 neurons (class scores)

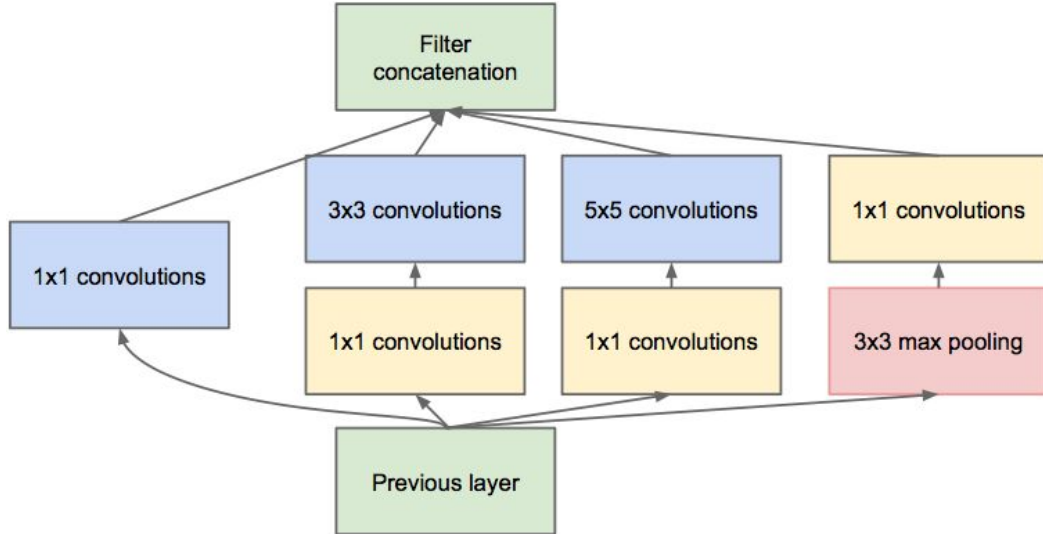
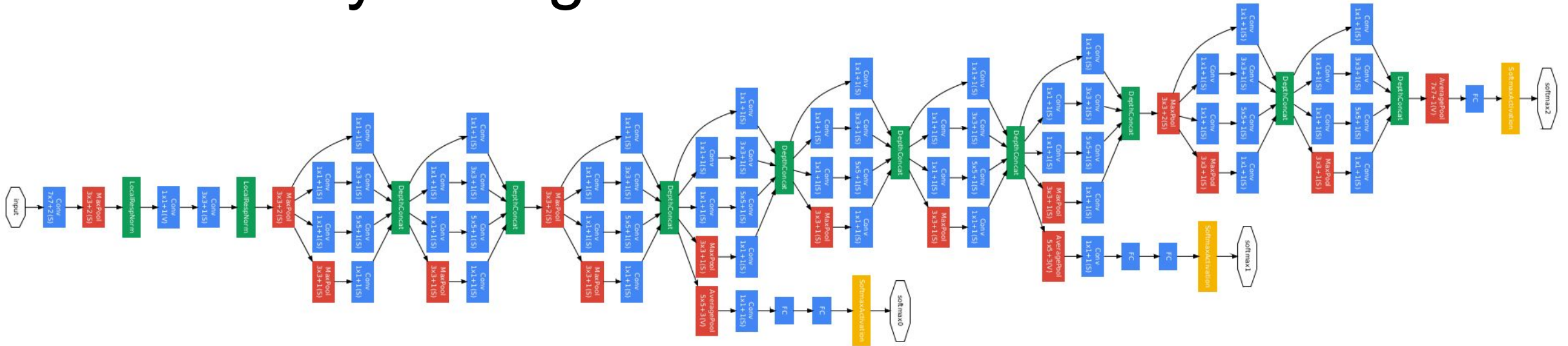


## Details/Retrospectives:

- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

# Case Study: GoogLeNet

[Szegedy et al., 2014]



## Inception module

ILSVRC 2014 winner (6.7% top 5 error)

# Case Study: GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Fun features:


- Only 5 million params! (Removes FC layers completely)

**Compared to AlexNet:**

- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

# Case Study: ResNet [He et al., 2015]


ILSVRC 2015 winner (3.6% top 5 error)



## MSRA @ ILSVRC & COCO 2015 Competitions

- **1st places** in all five main tracks
  - ImageNet Classification: “Ultra-deep” (quote Yann) **152-layer** nets
  - ImageNet Detection: **16%** better than 2nd
  - ImageNet Localization: **27%** better than 2nd
  - COCO Detection: **11%** better than 2nd
  - COCO Segmentation: **12%** better than 2nd

\*improvements are relative numbers

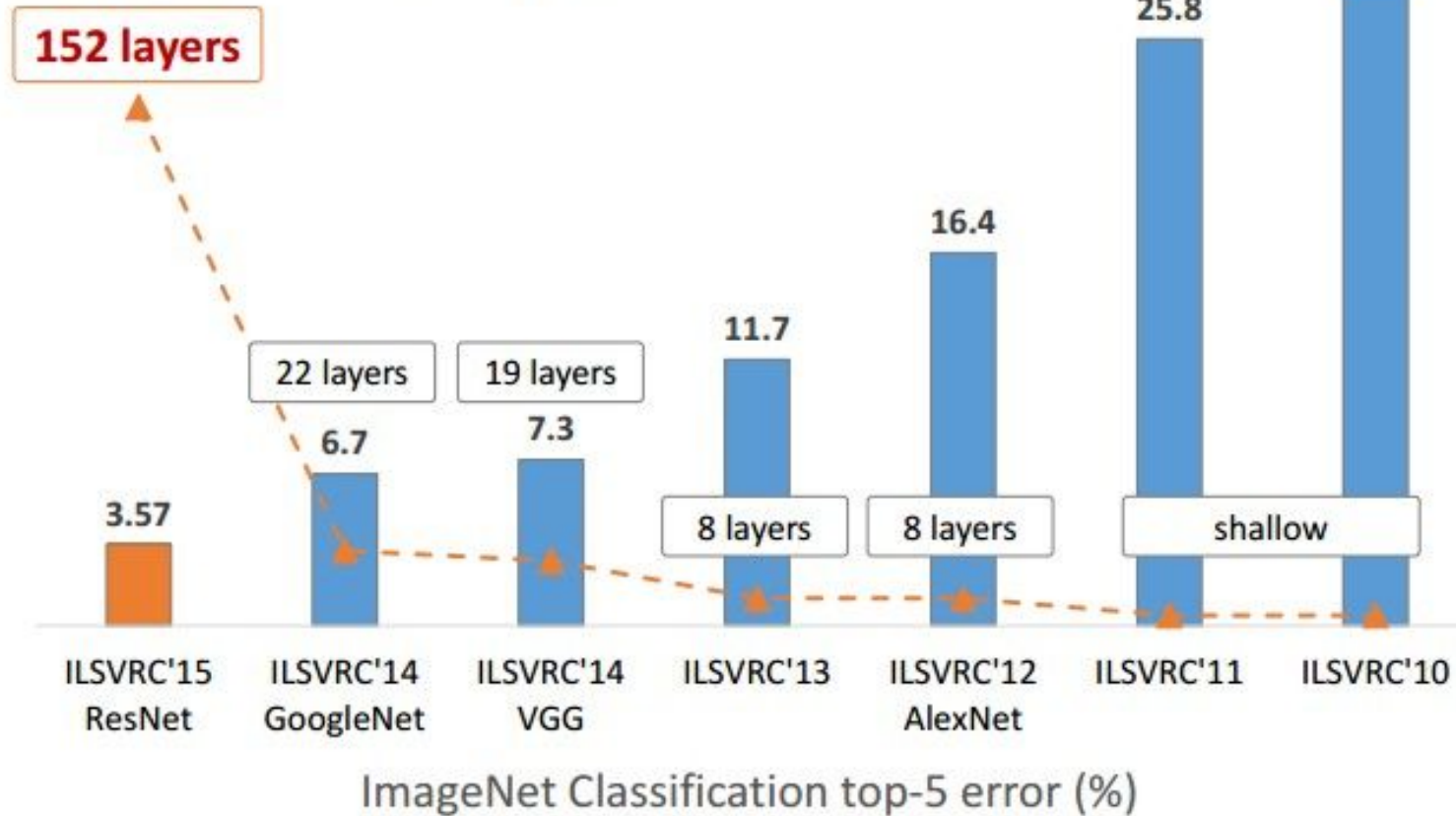


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. “Deep Residual Learning for Image Recognition”. arXiv 2015.

Slide from Kaiming He’s recent presentation <https://www.youtube.com/watch?v=1PGLj-uKT1w>



# Revolution of Depth

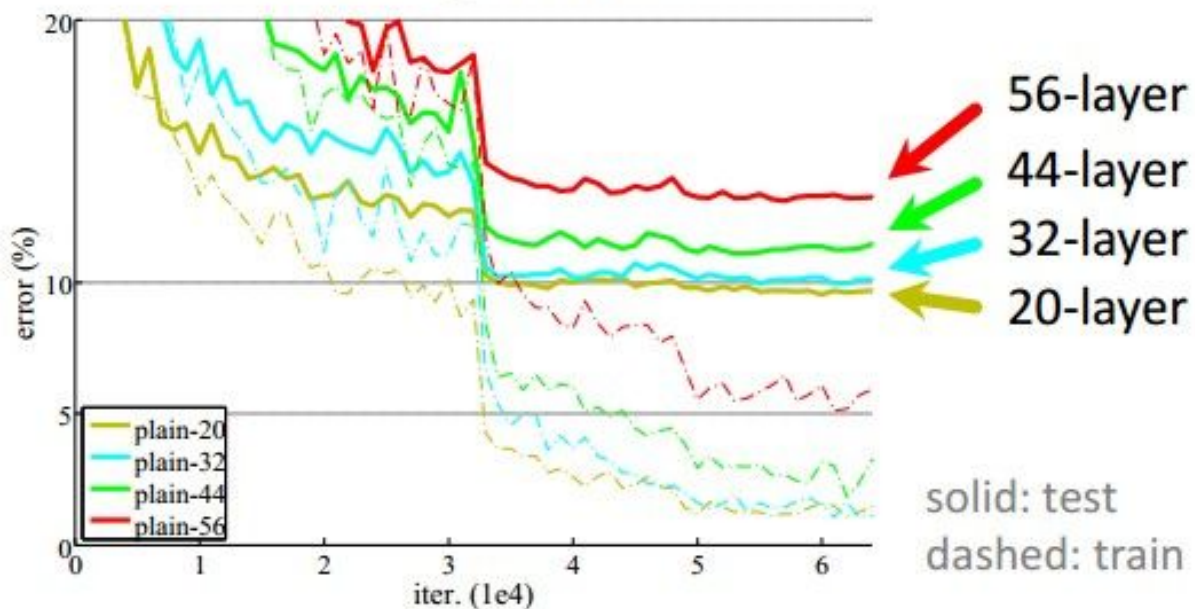


Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

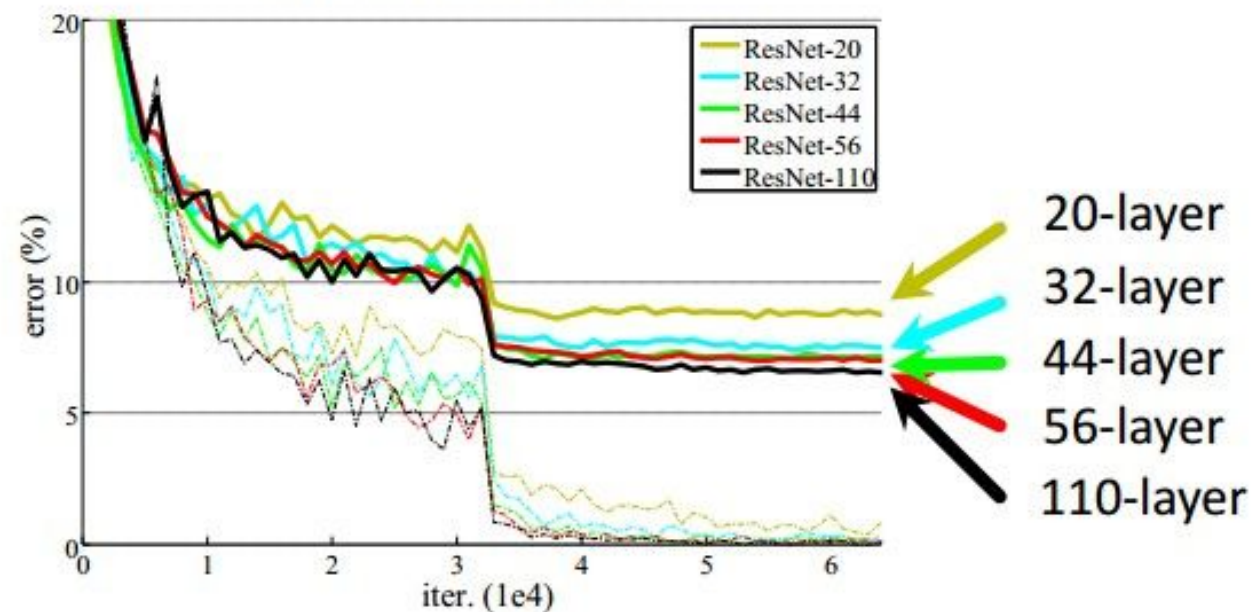
(slide from Kaiming He's recent presentation)

# CIFAR-10 experiments

CIFAR-10 plain nets

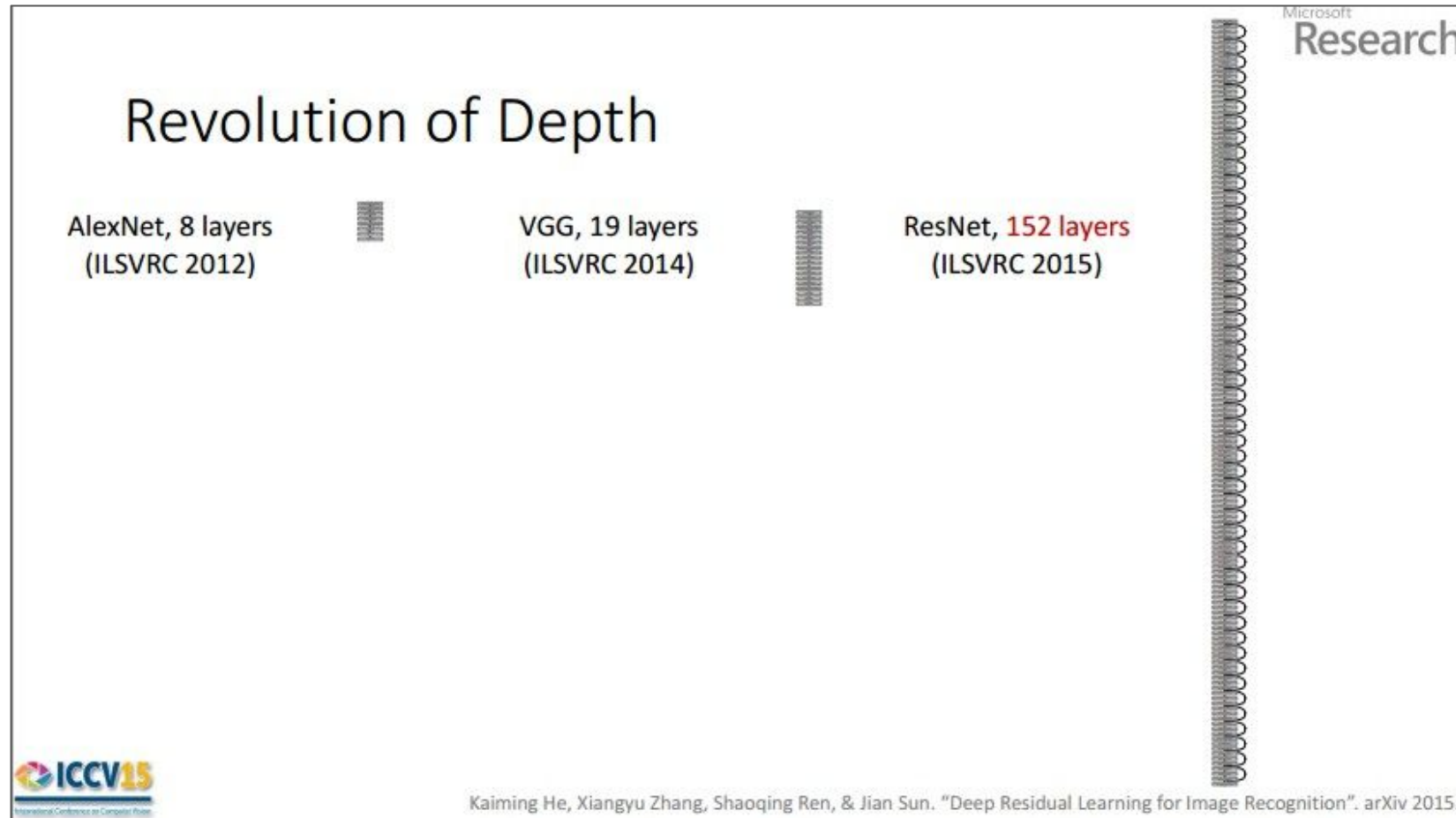


CIFAR-10 ResNets



# Case Study: ResNet [He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



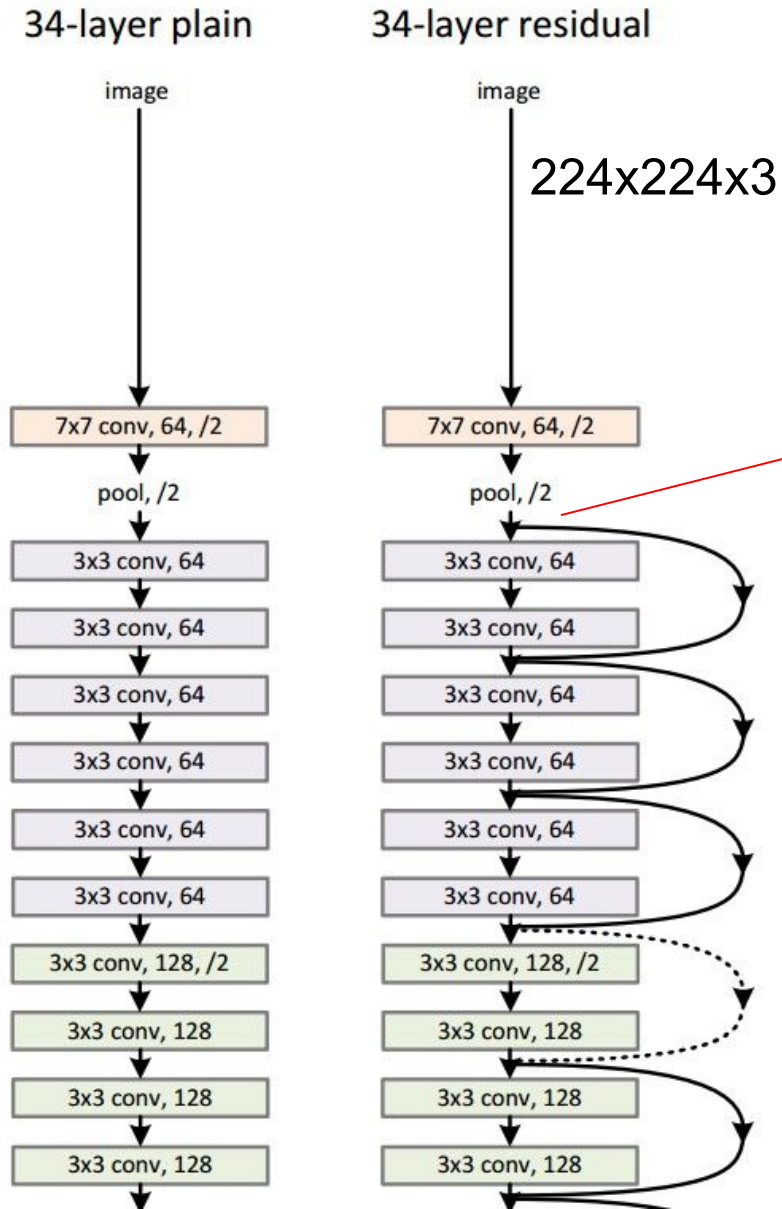
2-3 weeks of training  
on 8 GPU machine

at runtime: faster  
than a VGGNet!  
(even though it has  
8x more layers)

(slide from Kaiming He's recent presentation)

# Case Study: ResNet

[He et al., 2015]

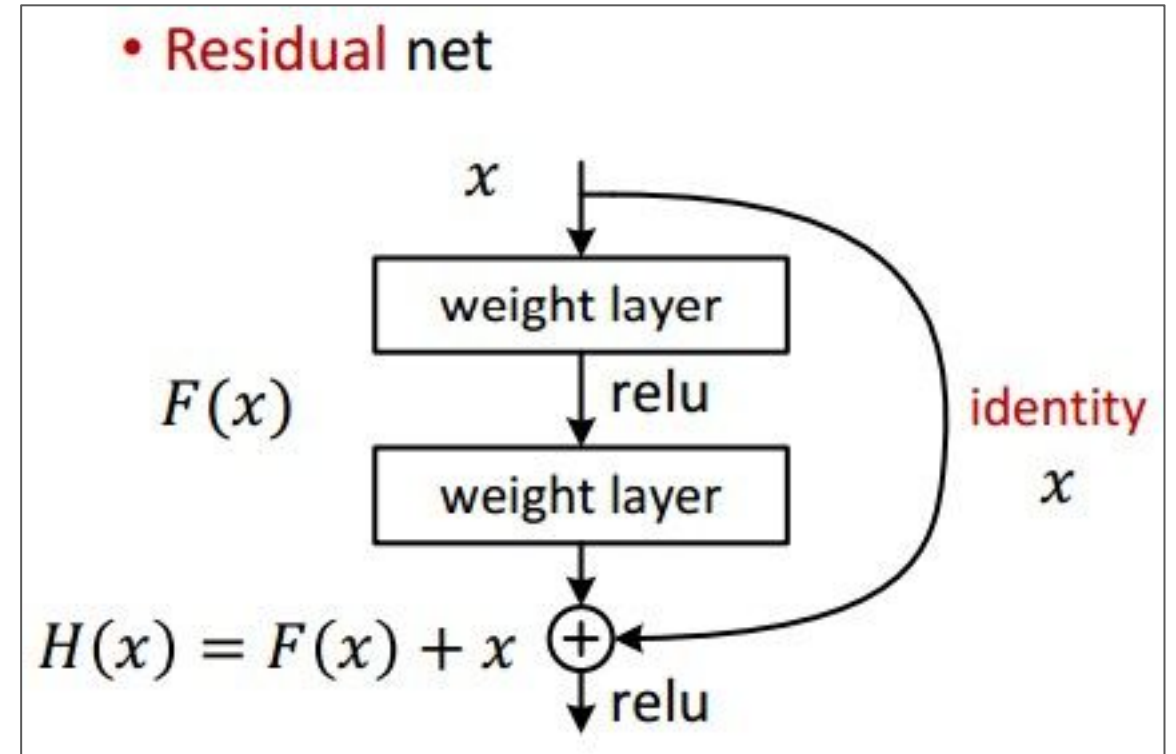
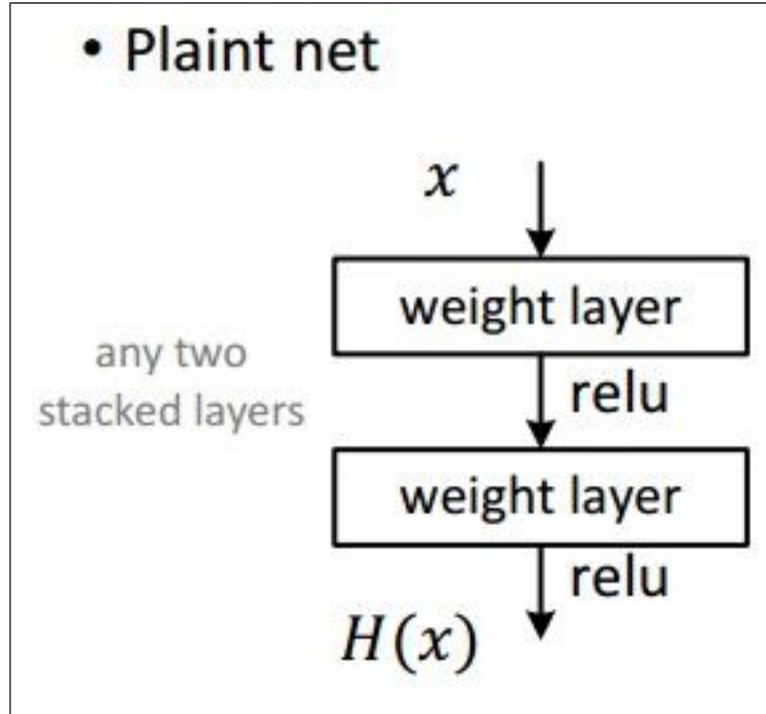


spatial dimension  
only 56x56!



# Case Study: ResNet

[He et al., 2015]

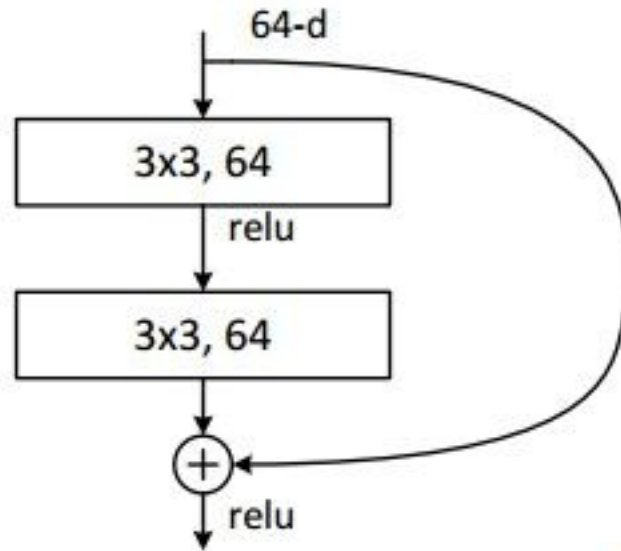


# Case Study: ResNet *[He et al., 2015]*

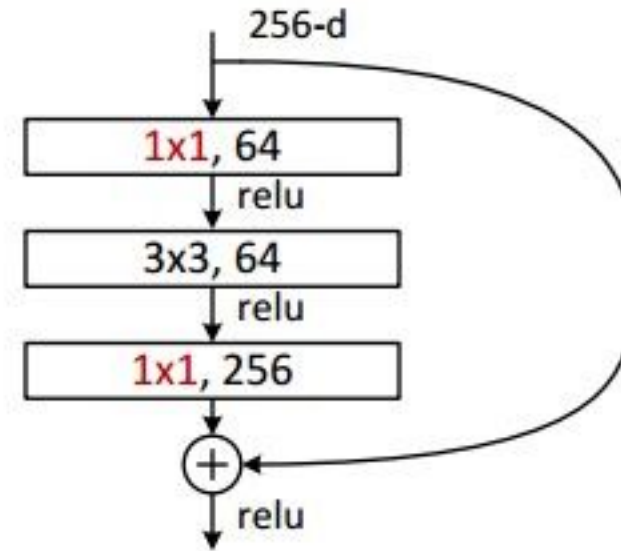
- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of  $1e-5$
- No dropout used

# Case Study: ResNet

[He et al., 2015]



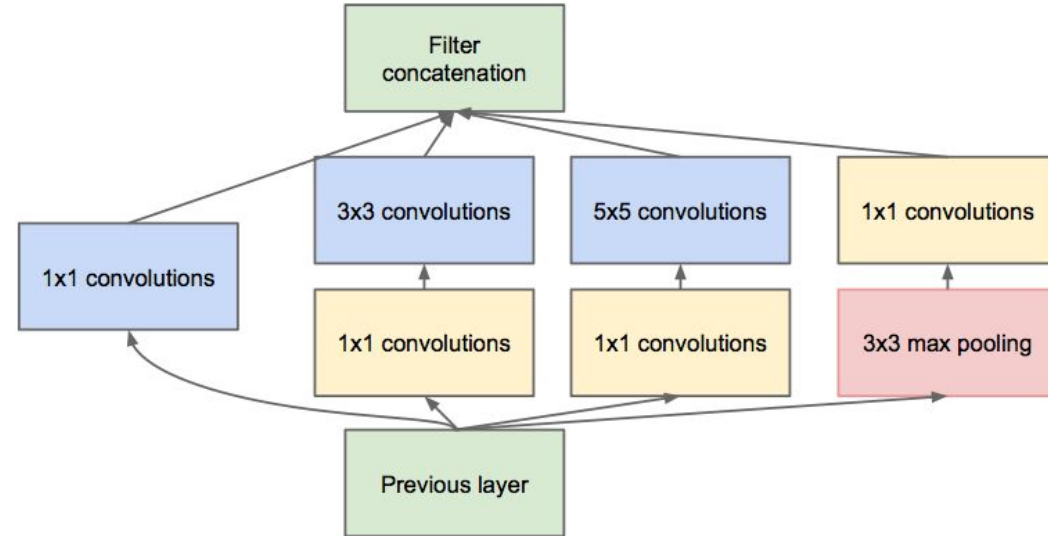
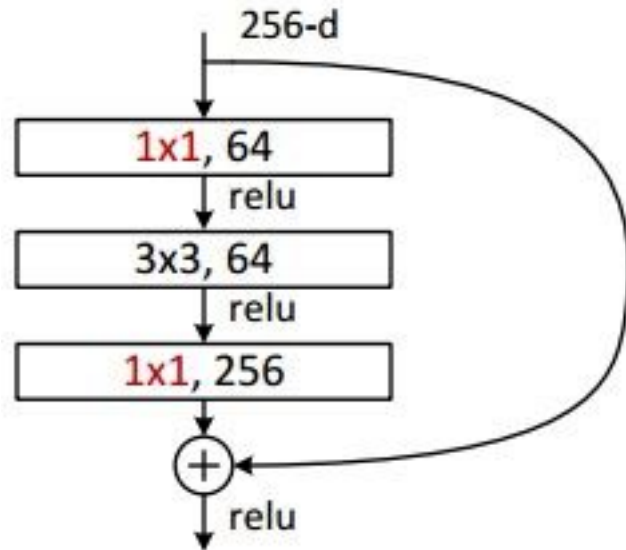
all-3x3



**bottleneck**  
(for ResNet-50/101/152)

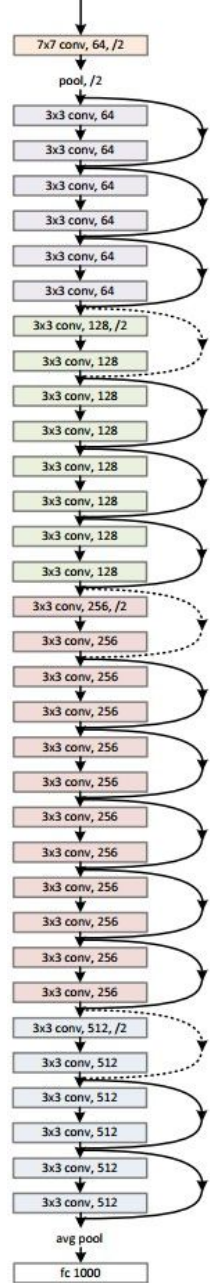
# Case Study: ResNet

[He et al., 2015]



(this trick is also used in GoogLeNet)

# Case Study: ResNet [He et al., 2015]



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	$112 \times 112$	$7 \times 7$ , 64, stride 2				
conv2_x	$56 \times 56$	$3 \times 3$ max pool, stride 2				
		$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$28 \times 28$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 8$
conv4_x	$14 \times 14$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 36$
conv5_x	$7 \times 7$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
	$1 \times 1$	average pool, 1000-d fc, softmax				
FLOPs		$1.8 \times 10^9$	$3.6 \times 10^9$	$3.8 \times 10^9$	$7.6 \times 10^9$	$11.3 \times 10^9$

# Datasets

1. CIFAR-10/CIFAR-100: These are datasets of small images, each containing one of 10 or 100 object classes, respectively. CIFAR-10 contains 60,000 32x32 color images in 10 classes, with 6,000 images per class, while CIFAR-100 contains 60,000 32x32 color images in 100 classes, with 600 images per class.
2. ImageNet: This is a large-scale image dataset with more than 14 million images in over 20,000 categories. It is commonly used for training deep CNNs.
3. COCO: This is a large-scale object detection, segmentation, and captioning dataset with more than 330,000 images.
4. Pascal VOC: This is a dataset for object detection, segmentation, and classification tasks, containing 20 object categories.
5. Stanford Dogs/Cats: These are datasets of images of dogs and cats, each containing 12,500 images.
6. Medical Image Datasets: There are several datasets available for medical image analysis, such as the Medical Segmentation Decathlon, BraTS, and ISIC Skin Lesion Analysis.

Questions ?