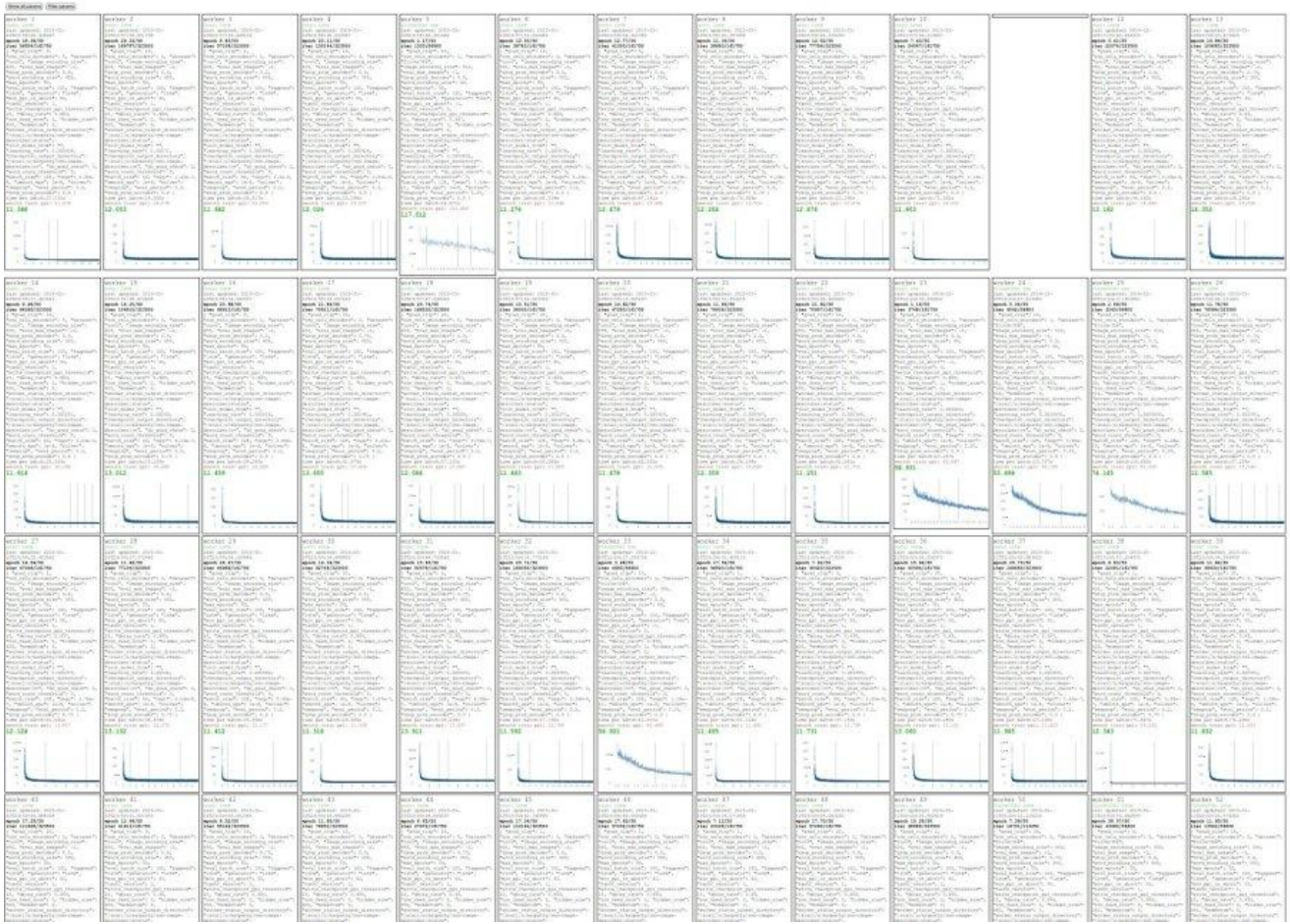
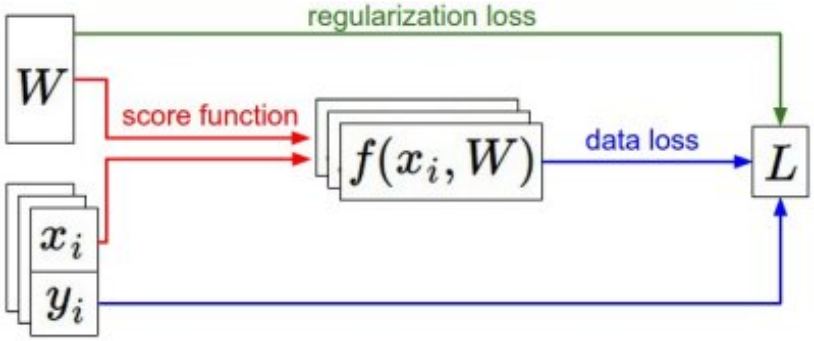
Administrative

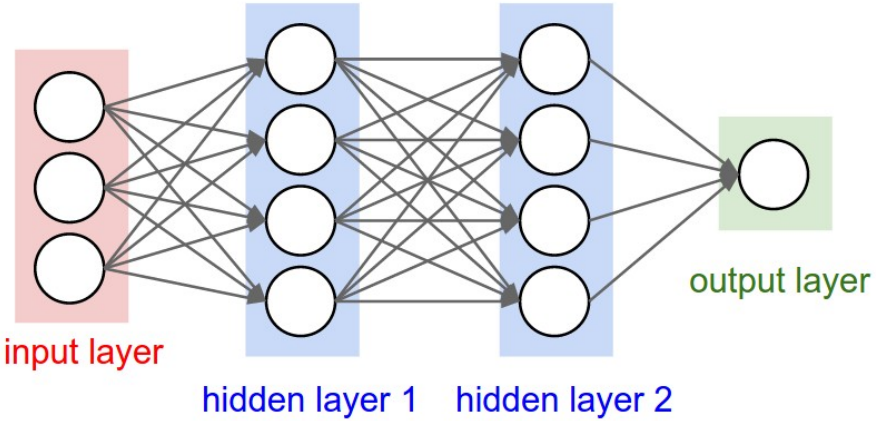
- I will hold office hours instead of Fei-Fei today, 4pm in Gates 300
- Project proposal due this Friday!
- Good advice: Start on A2 early. It can run long.

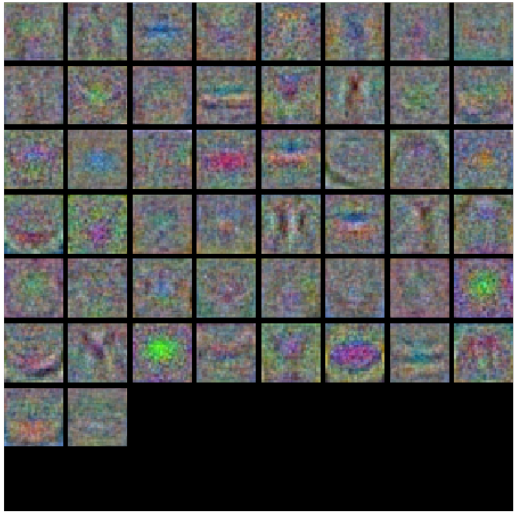
My cross-validation
“command center”



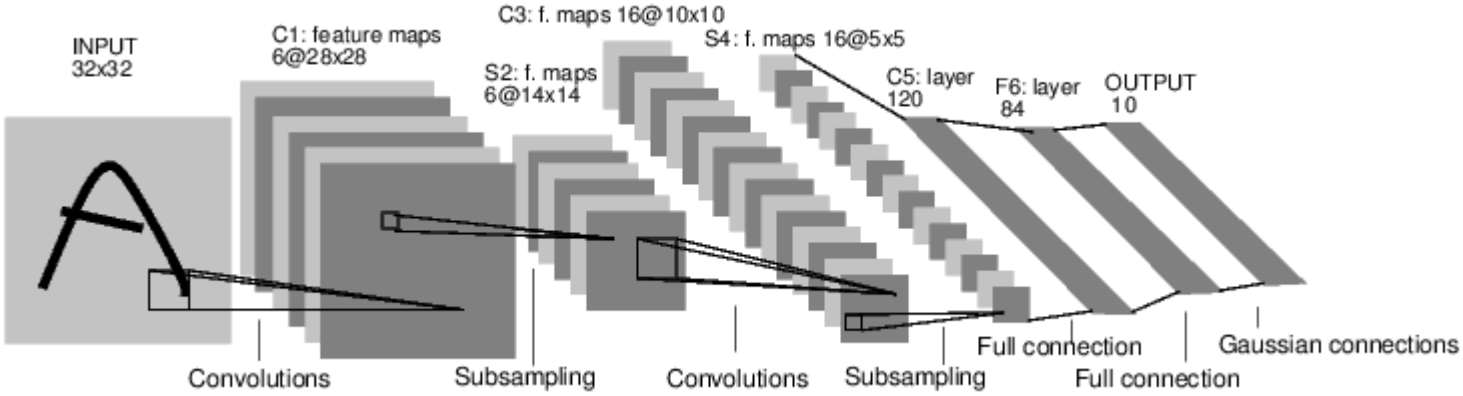
Where we are right now...







Lecture 7: Convolutional Neural Networks



[LeNet-5, LeCun 1980]

A bit of history:

Hubel & Wiesel,

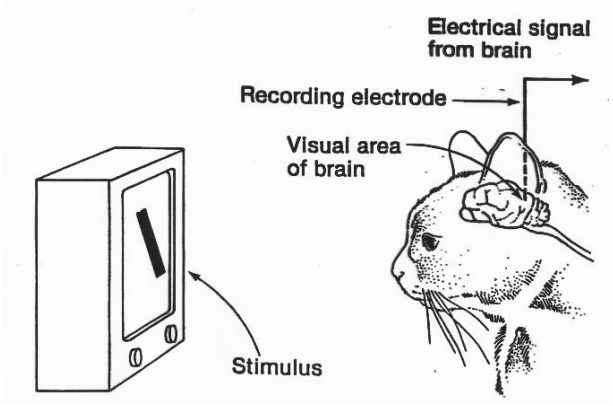
1959

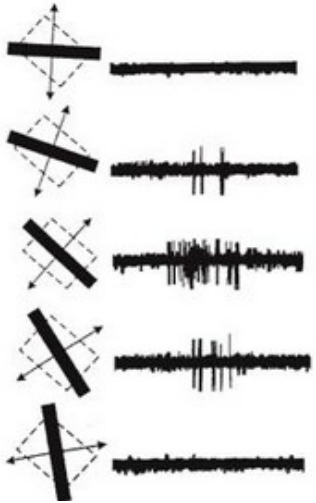
RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

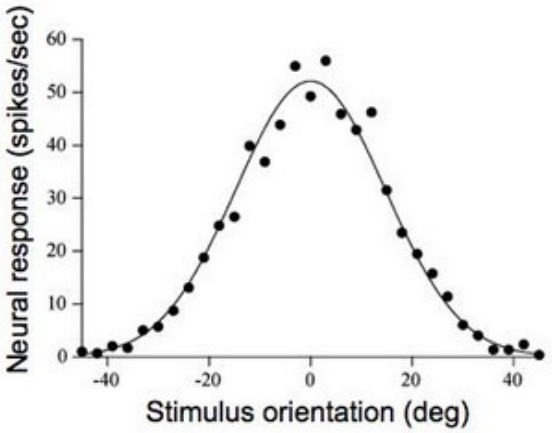
1962

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

1968...

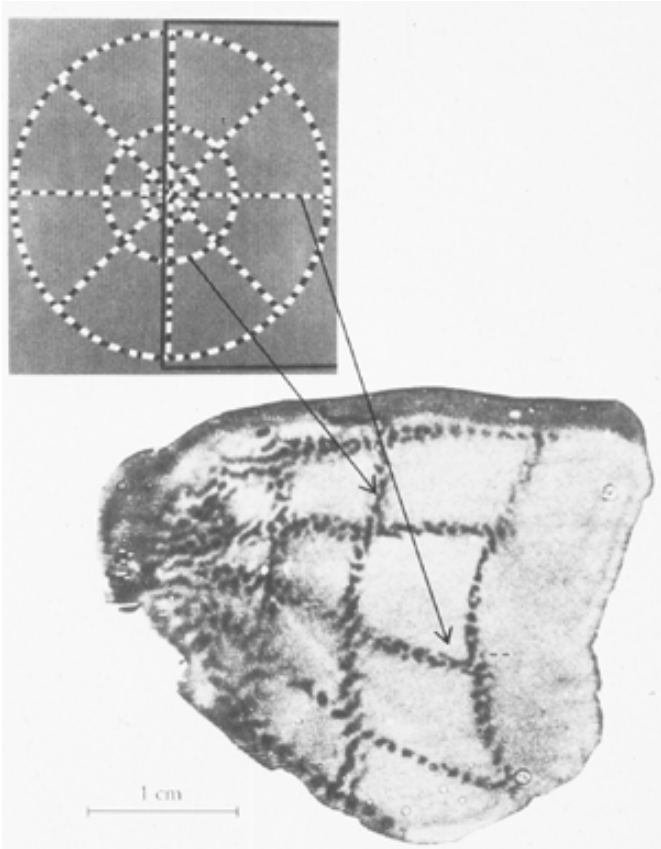






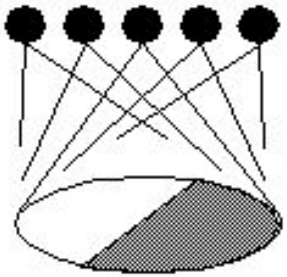
A bit of history

Topographical mapping in the cortex:
nearby cells in cortex represented
nearby regions in the visual field

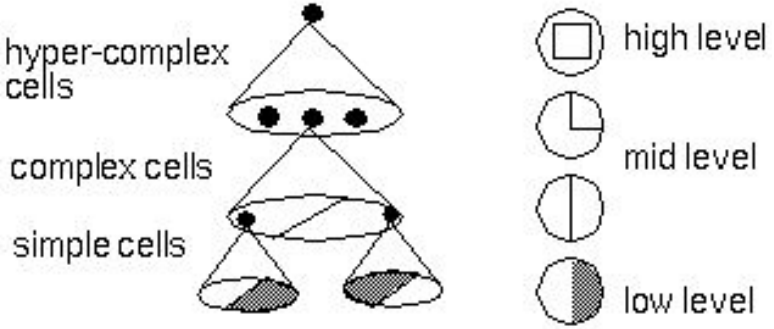


Hubel & Weisel

topographical mapping

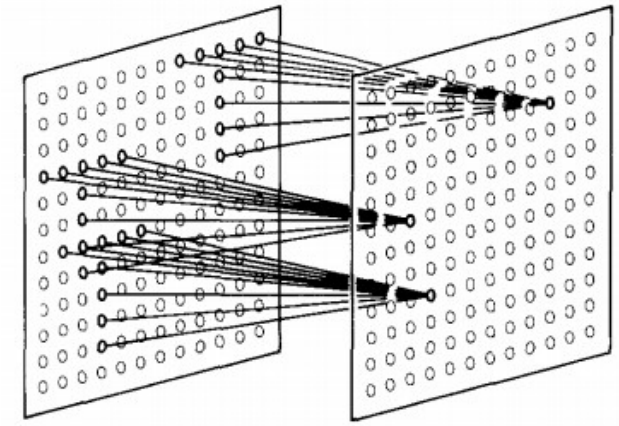


featural hierarchy

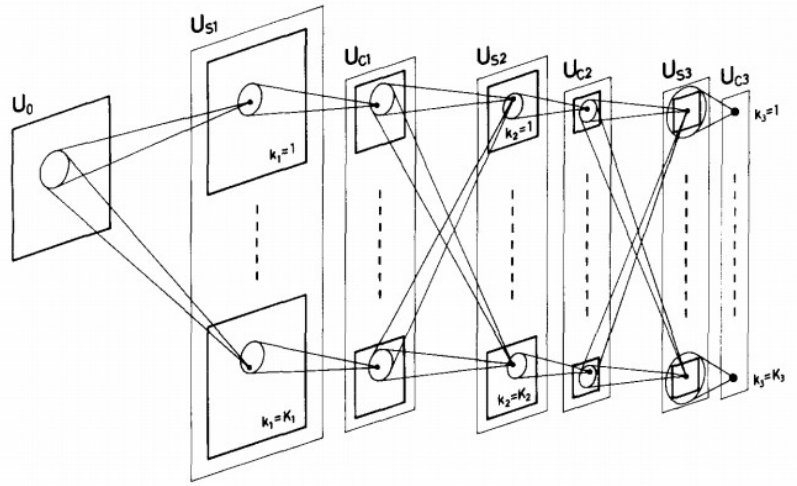


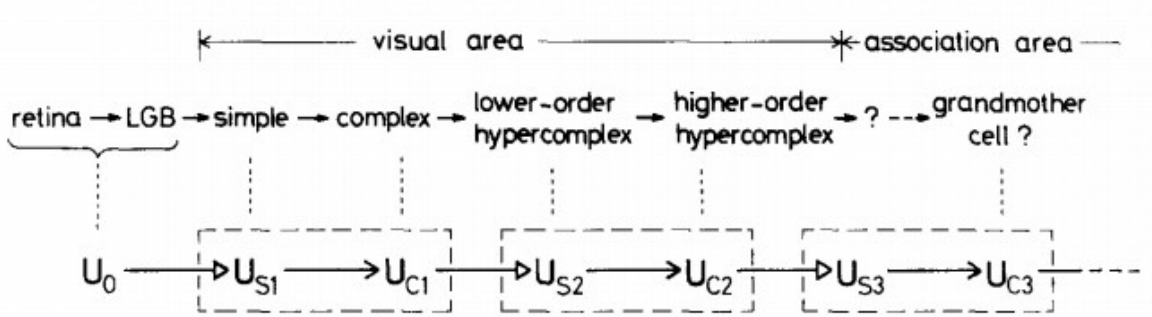
A bit of history:

Neurocognitron
[Fukushima 1980]



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

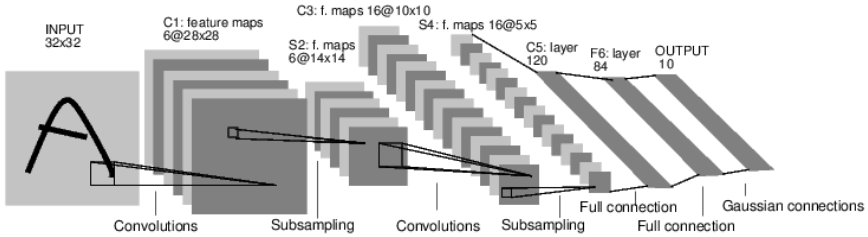




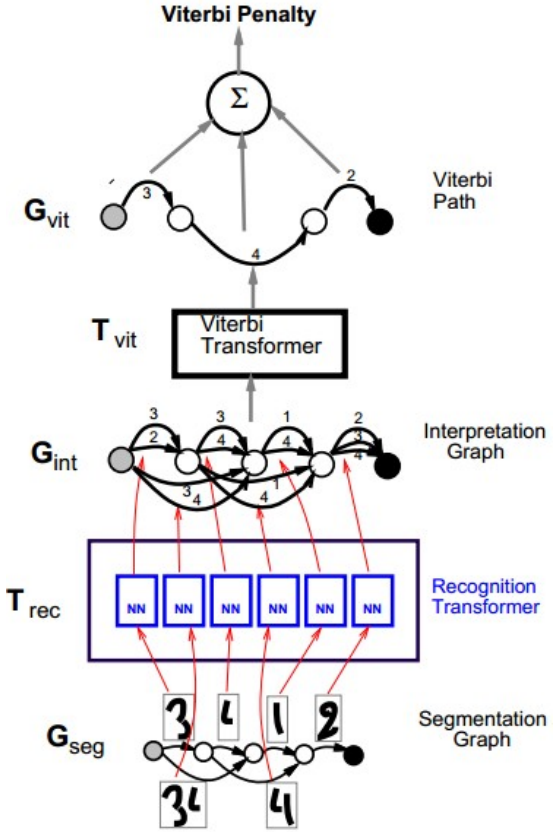
A bit of history:

Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]



LeNet-5

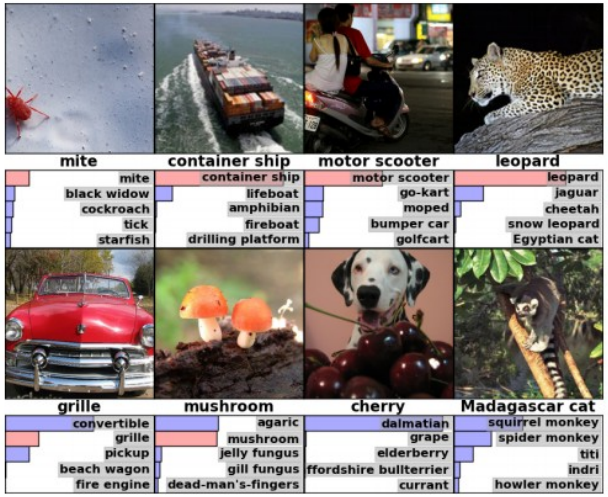


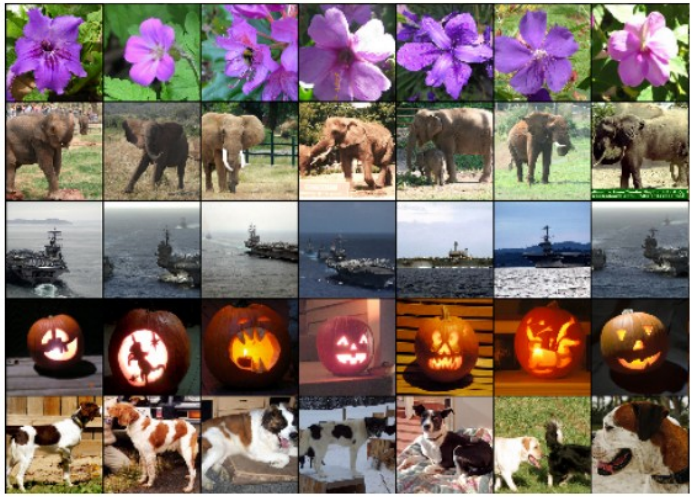
Fast-forward to today: ConvNets are everywhere

[Goodfellow 2014]

Classification

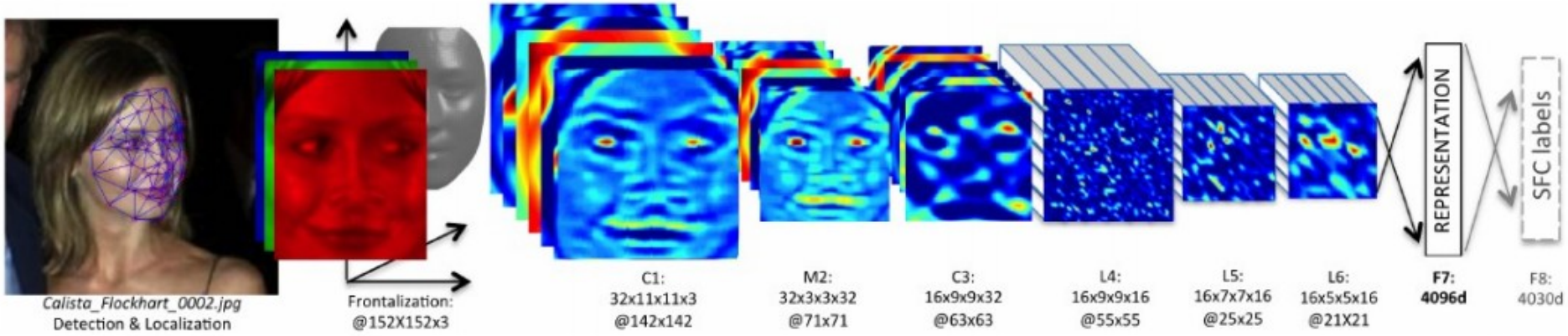
Retrieval



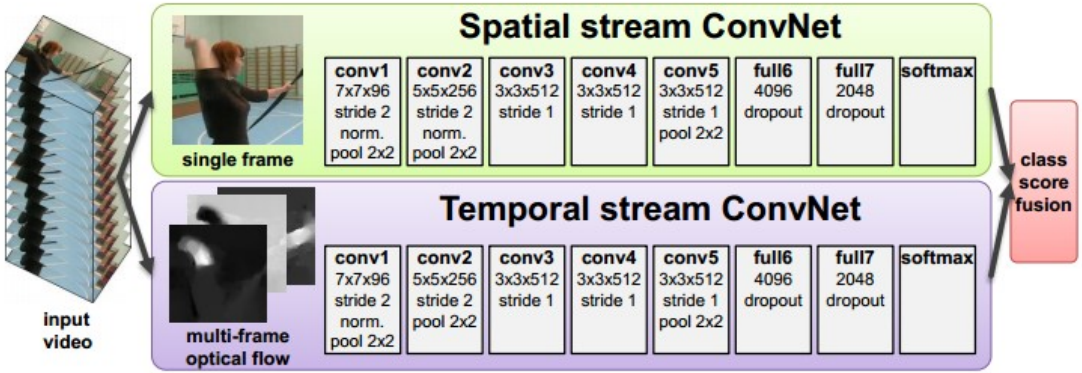


[Krizhevsky 2012]

Fast-forward to today: ConvNets are everywhere



[Taigman et al. 2014]



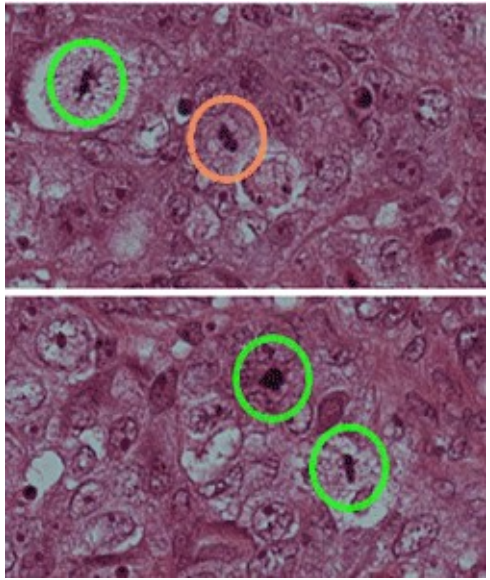
[Simonyan et al. 2014]

Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]

[Mnih 2013]

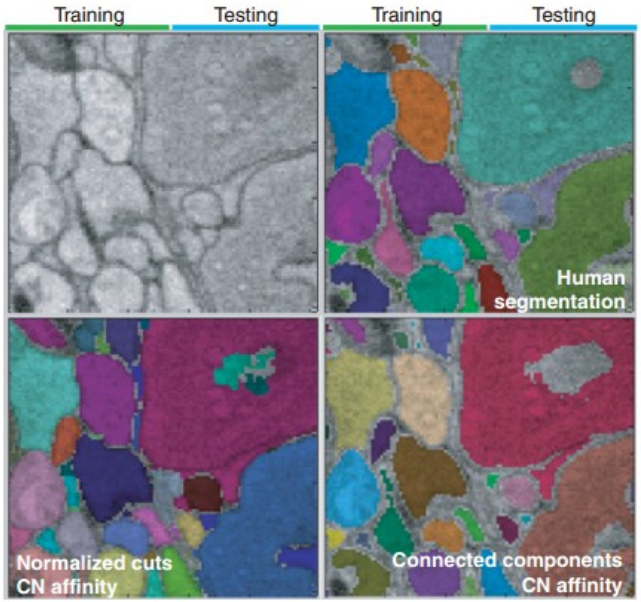
Fast-forward to today: ConvNets are everywhere



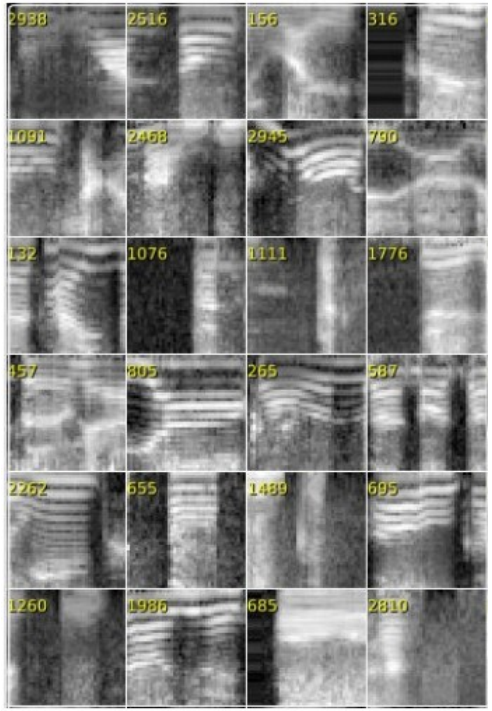
[Ciresan et al. 2013]

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

Fast-forward to today: ConvNets are everywhere



[Turaga et al., 2010]



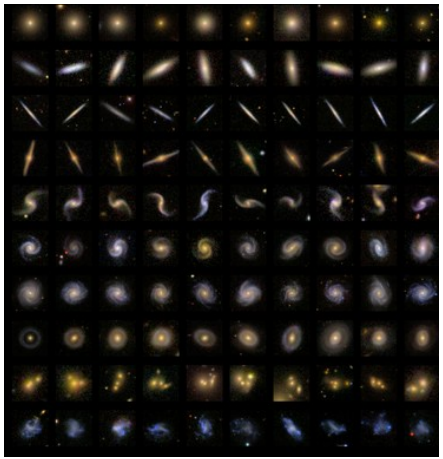
I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. **Two guys (one naive and one loud mouthed a **) take a road trip to stop a wedding but have the worst possible luck when a maniac in a freaky, make-shift tank/truck hybrid decides to play cat-and-mouse with them.** Things are further complicated when they pick up a ridiculously whorish hitchhiker. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chase/suspense scenes. Nice pacing and comic timing make this movie more than passable for the horror/slasher buff. **Definitely worth checking out.**

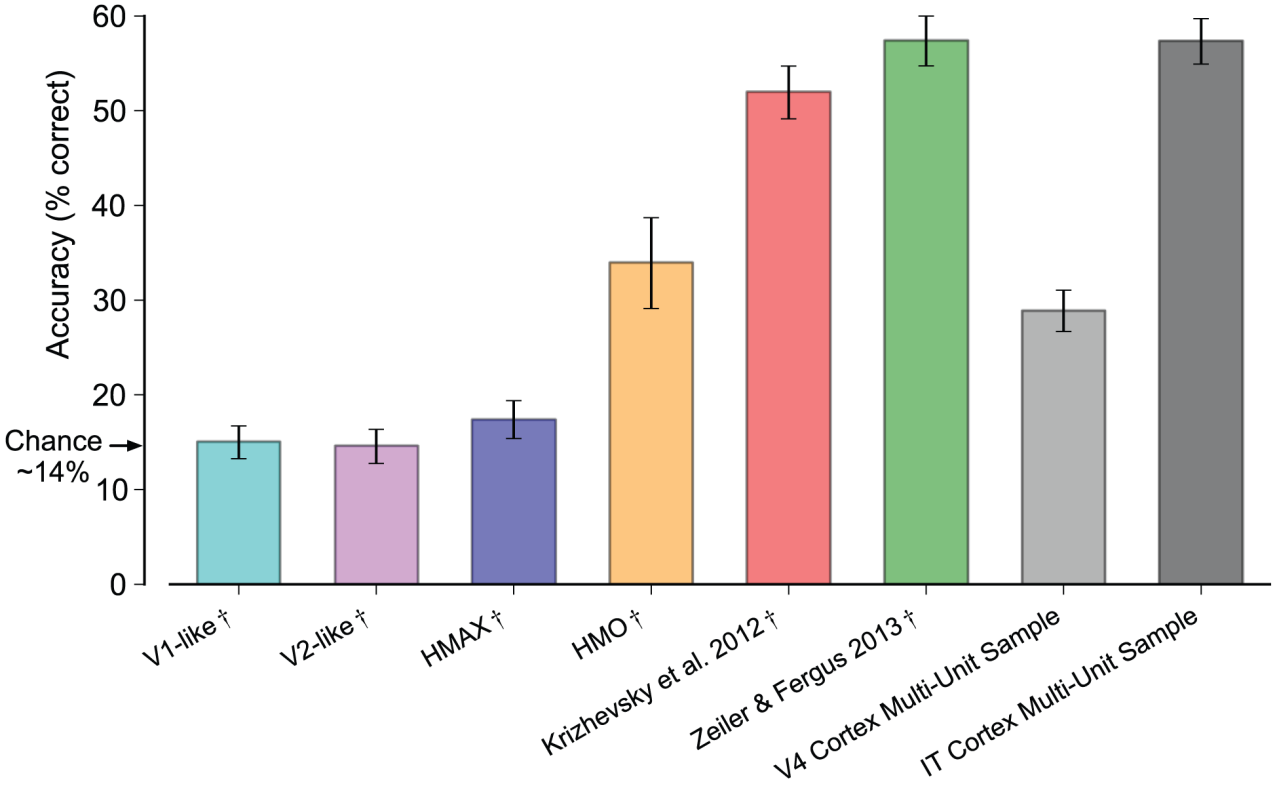
I just saw this on a local independent station in the New York City area. **The cast showed promise but when I saw the director, George Cosmotos, I became suspicious. And sure enough, it was every bit as bad, every bit as pointless and stupid as every George Cosmotos movie I ever saw.** He's like a stupid man's Michael Bey – with all the awfulness that accolade promises. There's no point to the conspiracy, no burning issues that urge the conspirators on. We are left to ourselves to connect the dots from one bit of graffiti on various walls in the film to the next. Thus, the current budget crisis, the war in Iraq, Islamic extremism, the fate of social security, 47 million Americans without health care, stagnating wages, and the death of the middle class are all subsumed by the sheer terror of graffiti. A truly, stunningly idiotic film.

Graphics is far from the best part of the game. **This is the number one best TH game in the series.** Next to Underground. **It deserves strong love. It is an insane game.** There are massive levels, massive unlockable characters... it's just a massive game. **Waste your money on this game. This is the kind of money that is wasted properly.** And even though graphics suck, thats doesn't make a game good. Actually, the graphics were good at the time. Today the graphics are crap. WHO CARES? As they say in Canada, This is the fun game, aye. (You get to go to Canada in THPS3) Well, I don't know if they say that, but they might. who knows. Well, Canadian people do. Wait a minute, I'm getting off topic. This game rocks. Buy it, play it, enjoy it, love it. It's PURE BRILLIANCE.

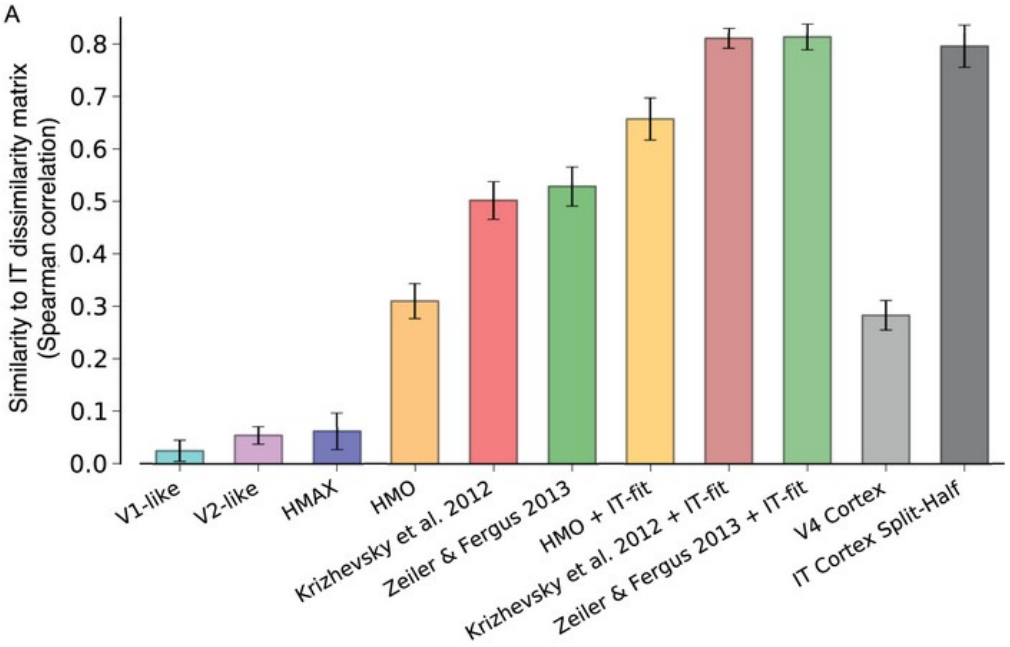
The first was good and original. I was a not bad horror/comedy movie. So I heard a second one was made and I had to watch it . What really makes this movie work is Judd Nelson's character and the sometimes clever script. **A pretty good script for a person who wrote the Final Destination films and the direction was okay.** Sometimes there's scenes where it looks like it was filmed using a home video camera with a grainy - look. Great made - for - TV movie. **It was worth the rental and probably worth buying just to get that nice eerie feeling and watch Judd Nelson's Stanley doing what he does best.** I suggest newcomers to watch the first one before watching the sequel, just so you'll have an idea what Stanley is like and get a little history background.

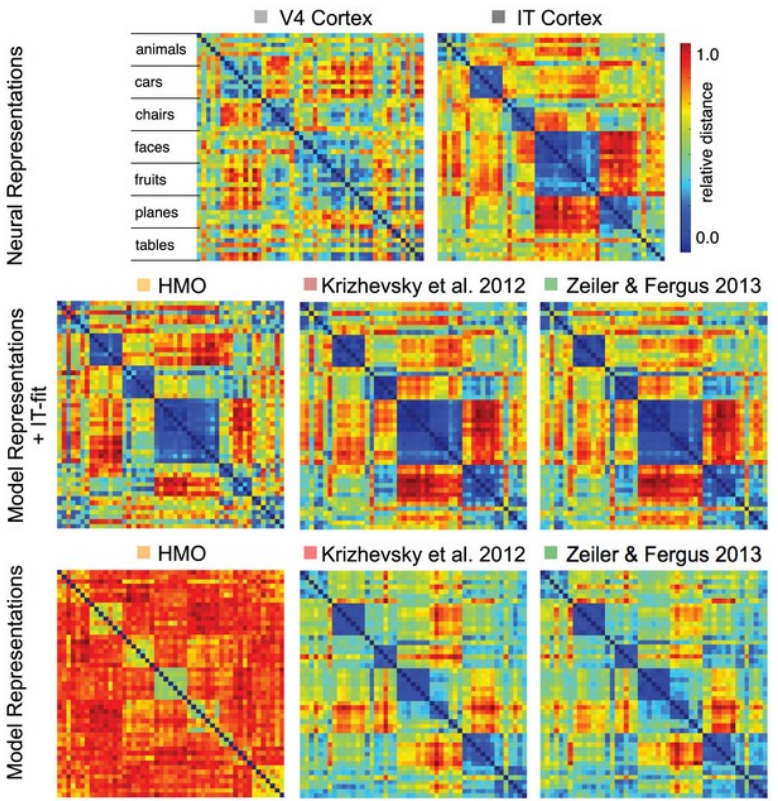
[Denil et al. 2014]





Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition [Cadieu et al., 2014]

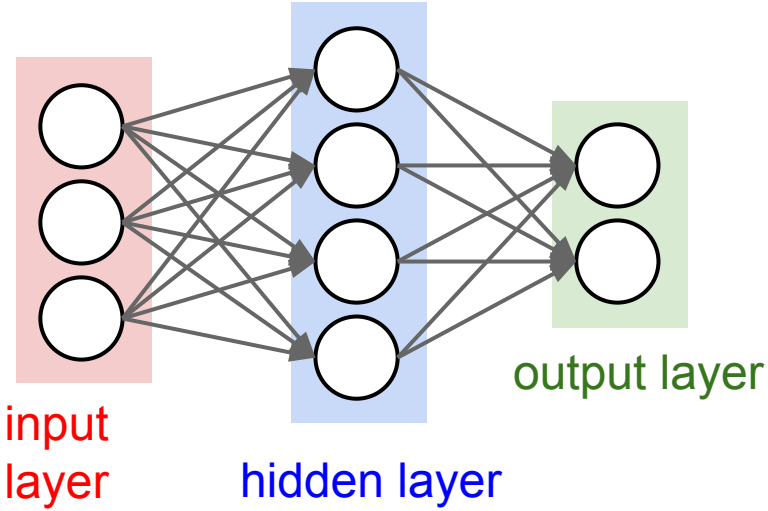




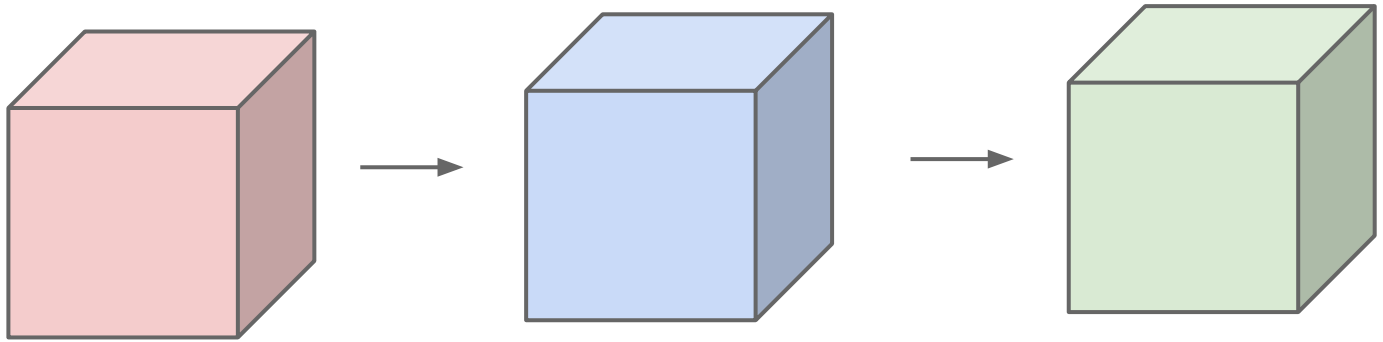
Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition [Cadieu et al., 2014]


CONVNETS
HOW DO THEY WORK

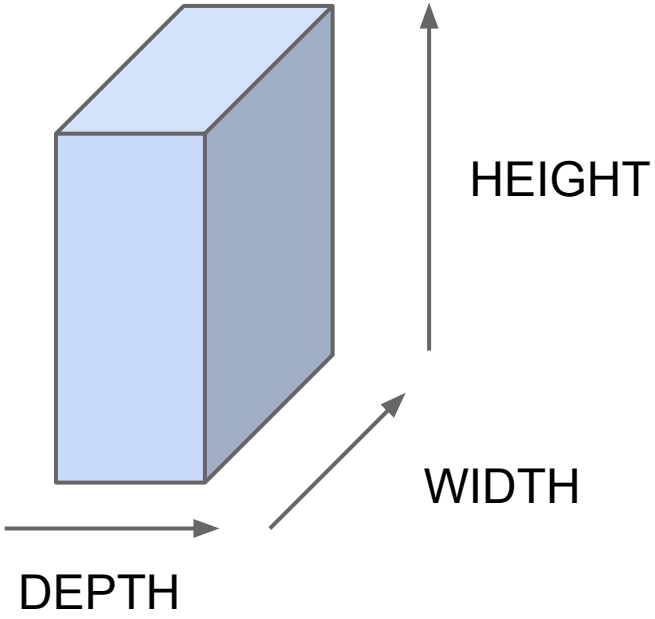
before:



now:



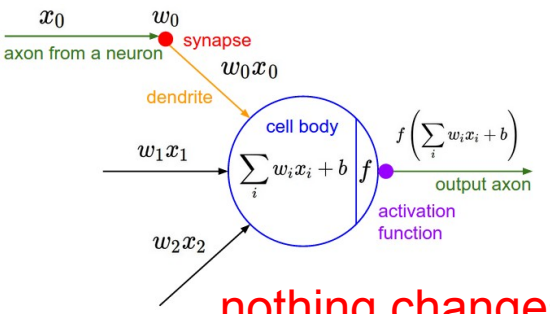
All Neural Net activations arranged in **3 dimensions:**



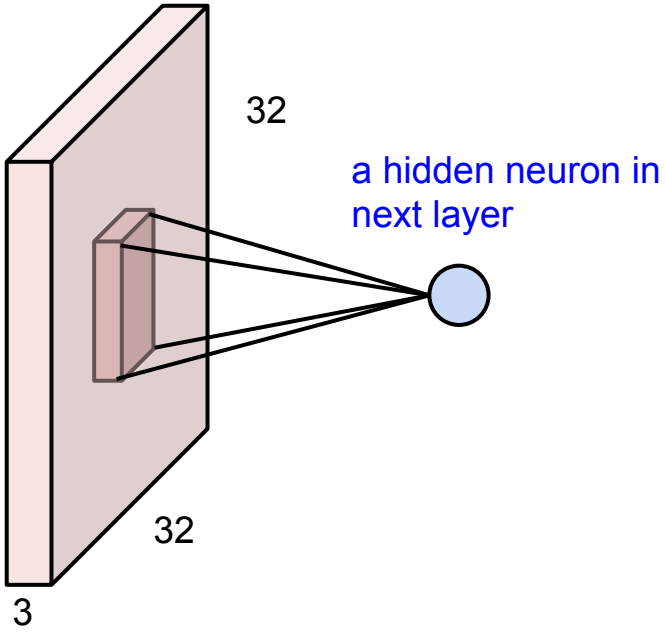
For example, a CIFAR-10 image is a 32x32x3 volume
32 width, 32 height, 3 depth (RGB channels)

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**

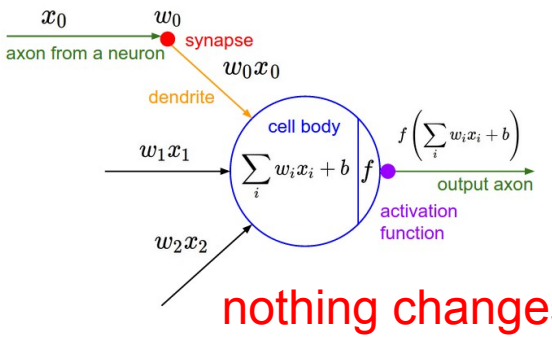


nothing changes



Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity



nothing changes

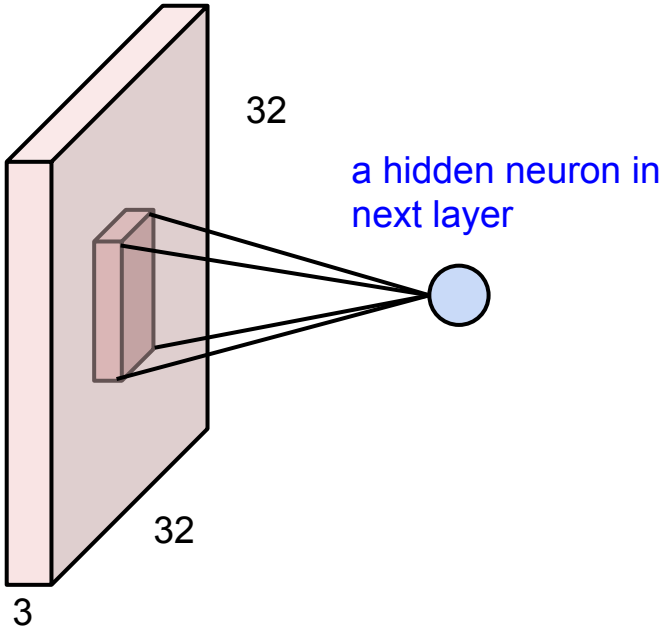


image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

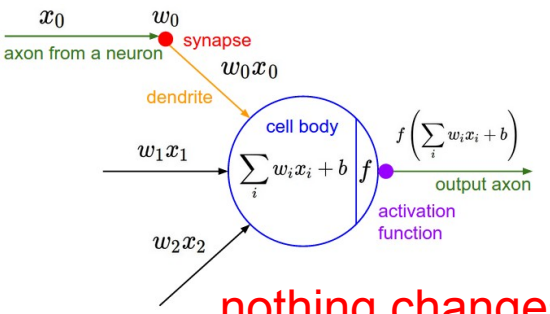
now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

note that connectivity is:

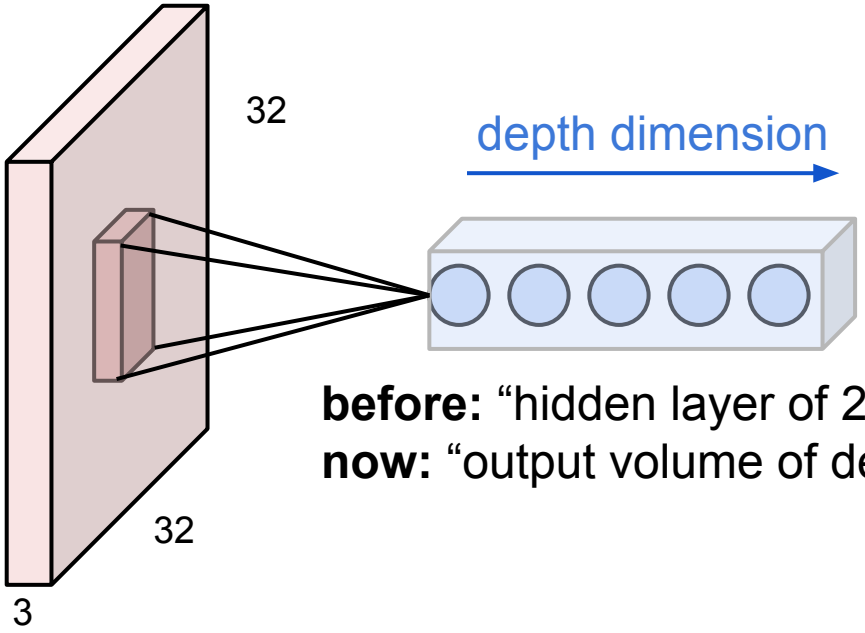
- <u>local in space</u> (5x5 inside 32x32)
- <u>but full in depth</u> (all 3 depth channels)

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**



nothing changes

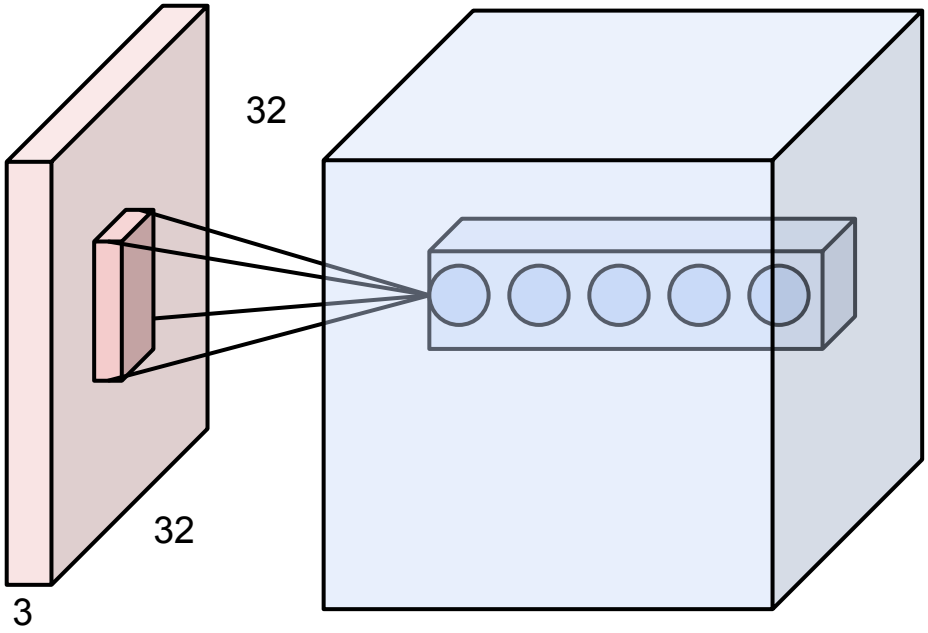


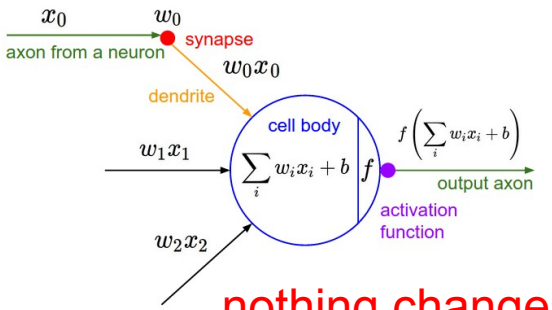
before: “hidden layer of 200 neurons”
now: “output volume of depth 200”

Multiple neurons all looking at the same region of the input volume, stacked along depth.

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**

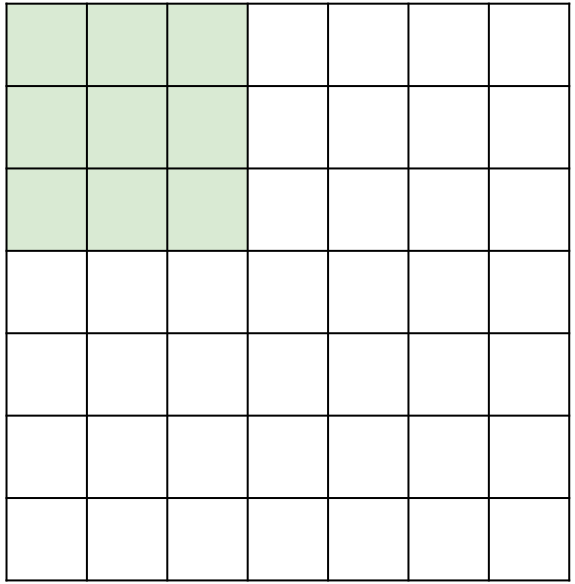




nothing changes

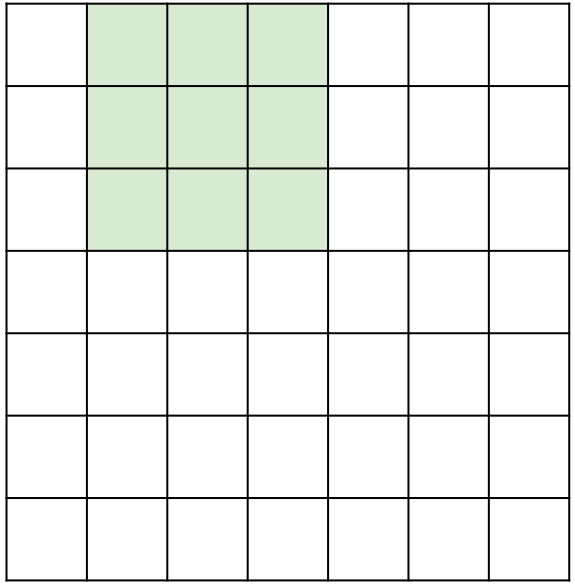
These form a single [1 x 1 x depth] "depth column" in the output volume

Replicate this column of hidden neurons across space, with some **stride.**



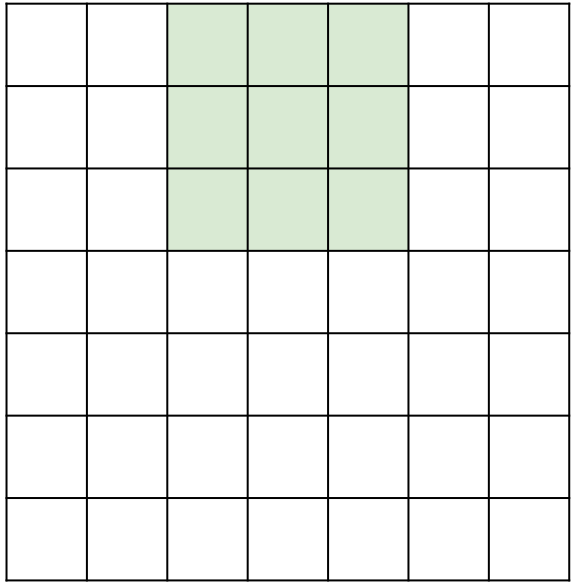
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



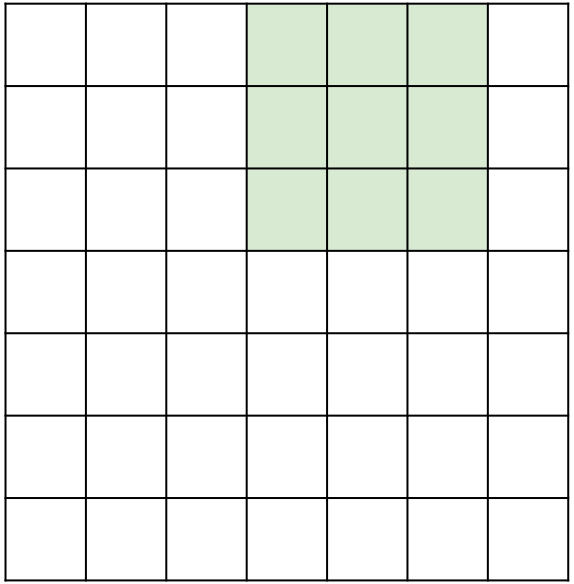
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



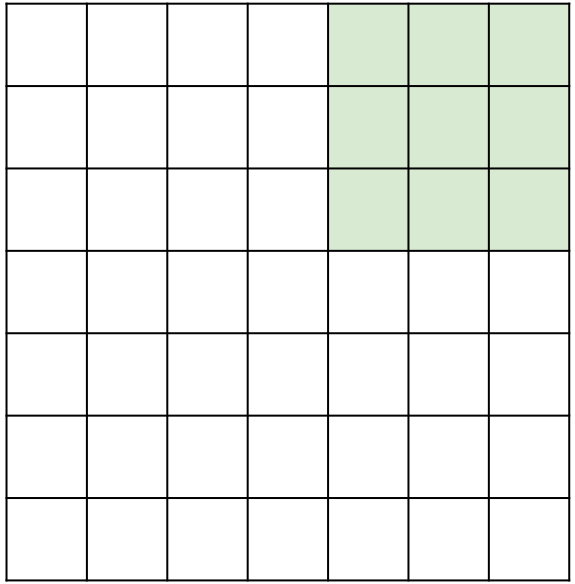
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



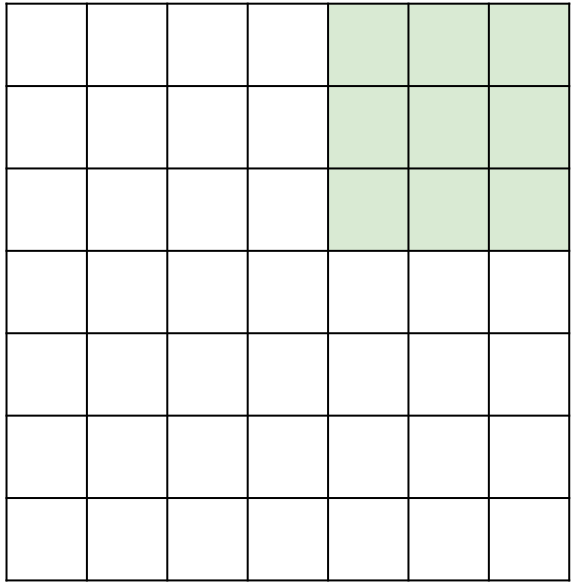
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



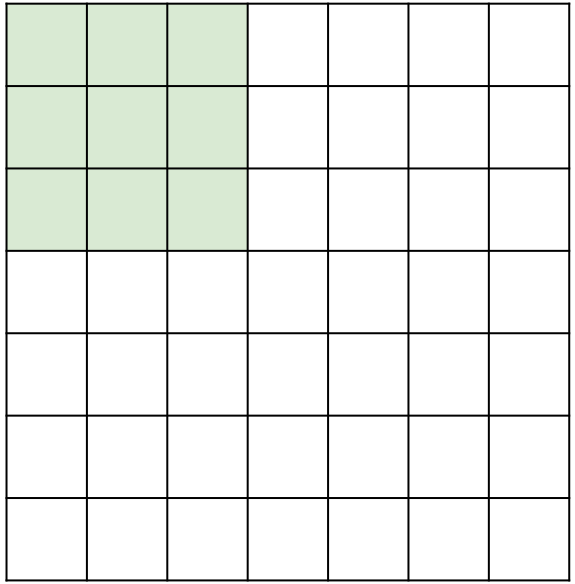
7x7 input
assume 3x3 connectivity, stride 1

Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

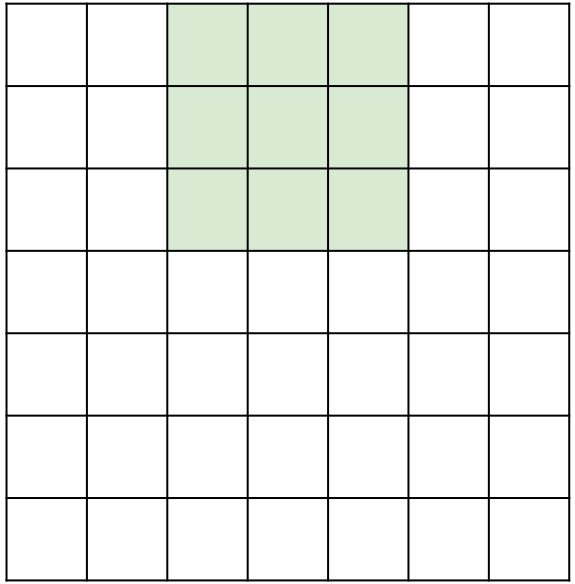
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

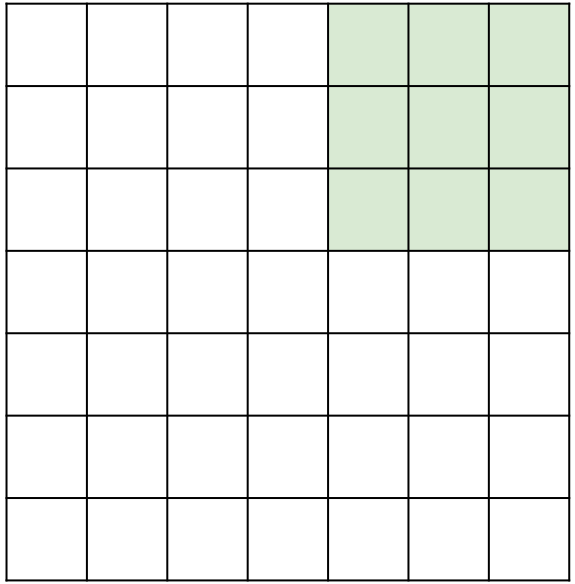
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

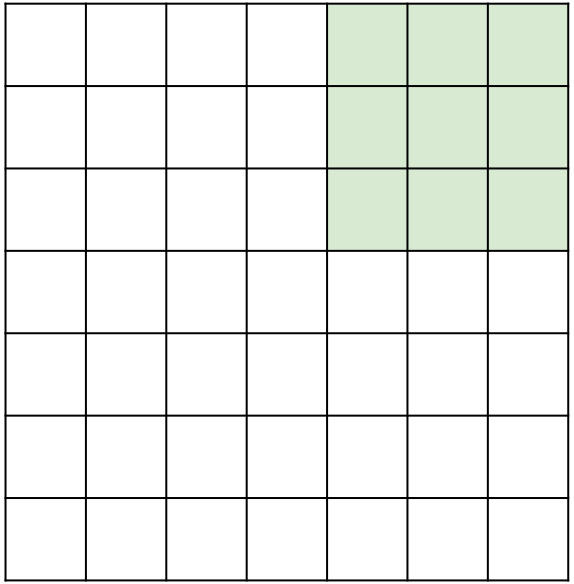
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

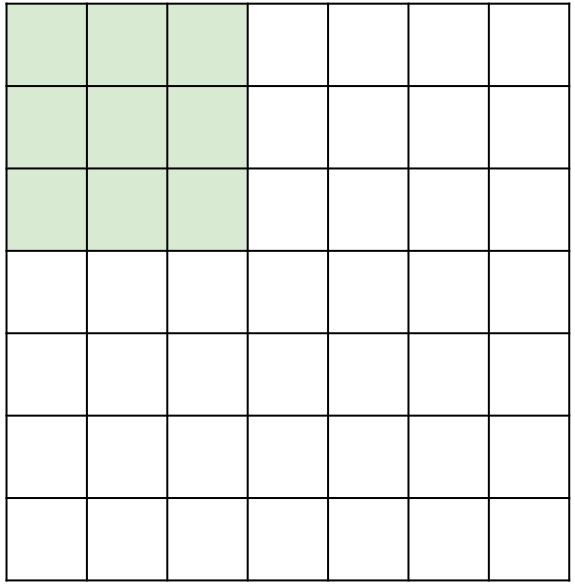
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

Replicate this column of hidden neurons across space, with some **stride.**

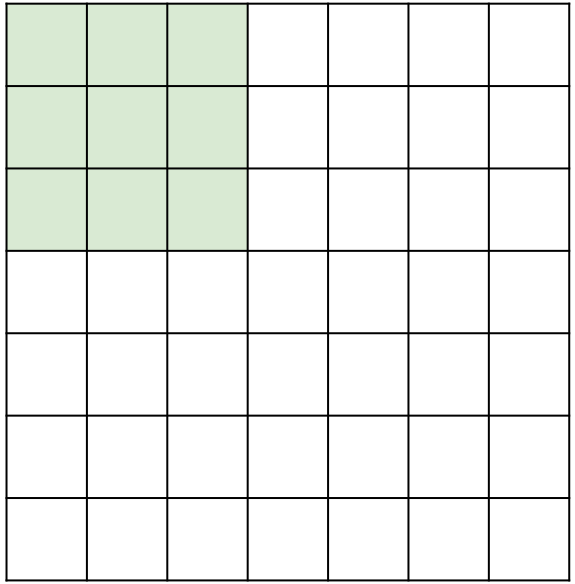


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

what about stride 3?

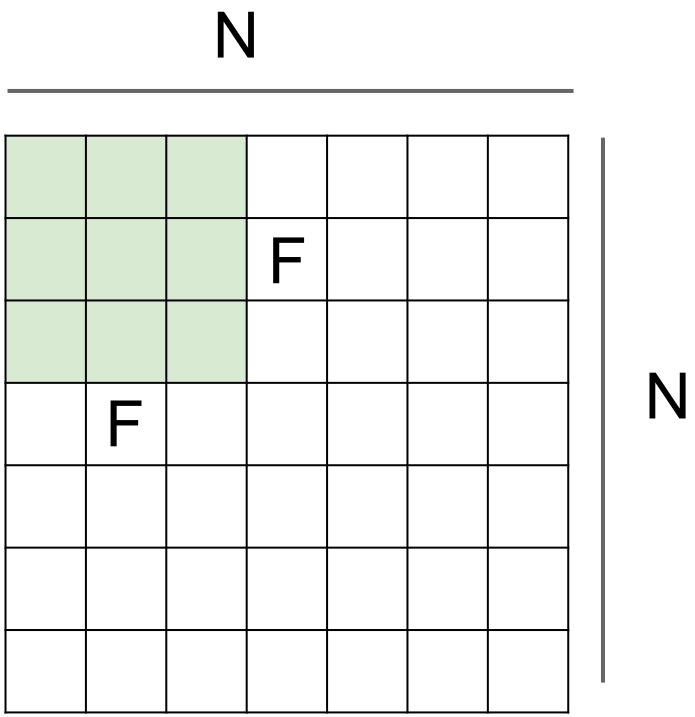
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?
=> **3x3 output**

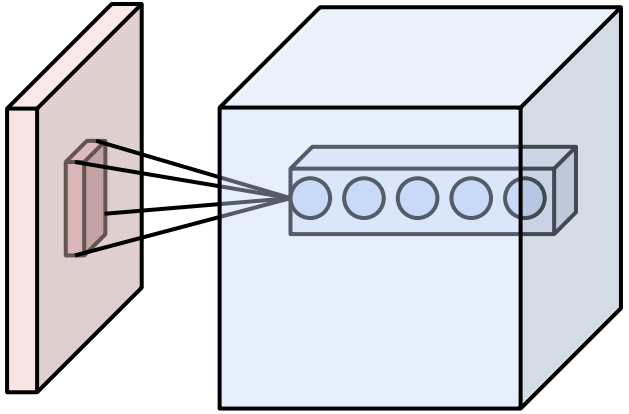
what about stride 3? Cannot.



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 = … :\

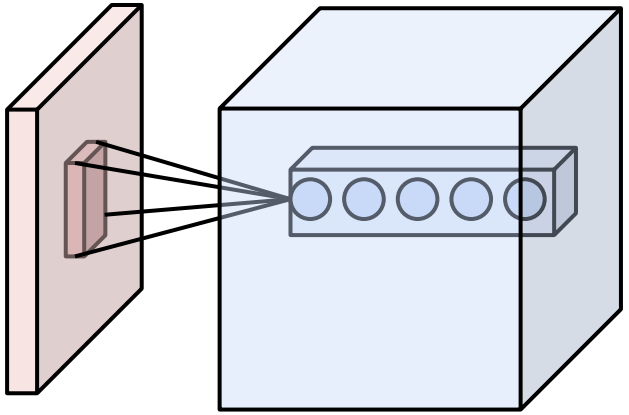
Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

Output volume: ?

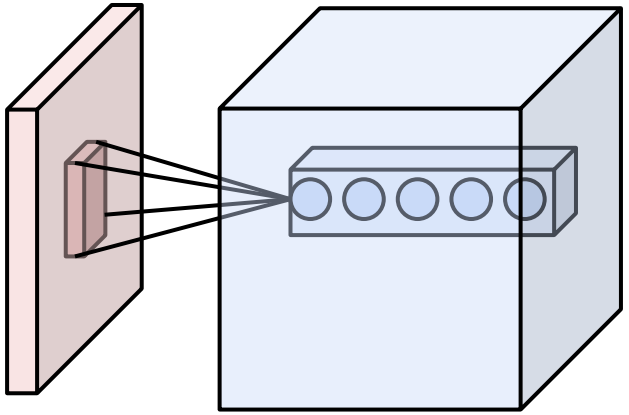
Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

Output volume: (32 - 5) / 1 + 1 = 28, so: **28x28x5**
How many weights for each of the 28x28x5 neurons?

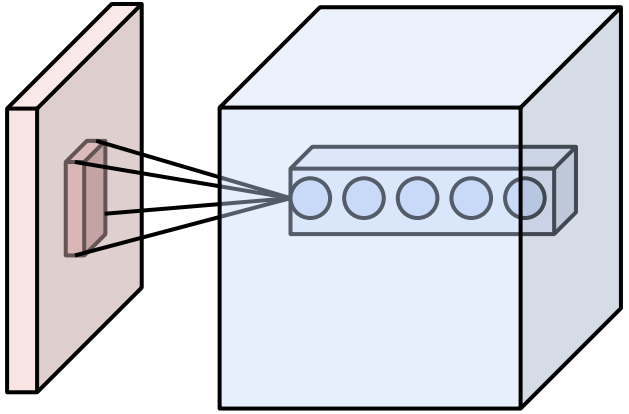
Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

Output volume: (32 - 5) / 1 + 1 = 28, so: **28x28x5**
How many weights for each of the 28x28x5 neurons? **5x5x3 = 75**

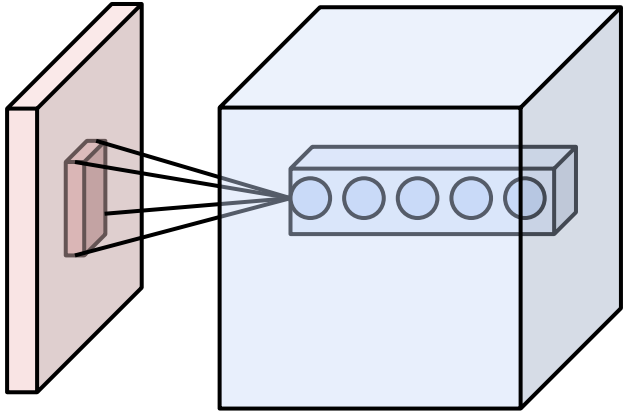
Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 2**
Number of neurons: **5**

Output volume: ?

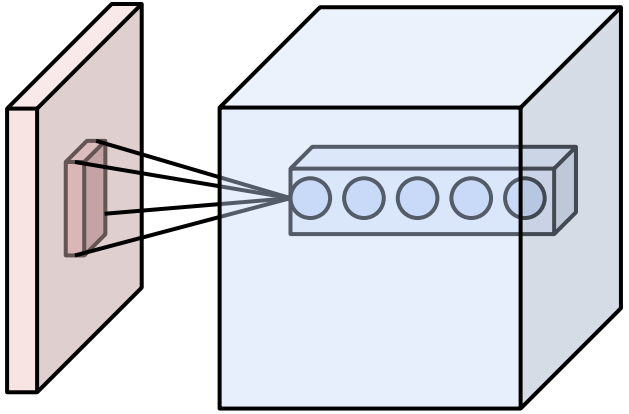
Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 2**
Number of neurons: **5**

Output volume: ? Cannot: (32-5)/2 + 1 = 14.5 :\

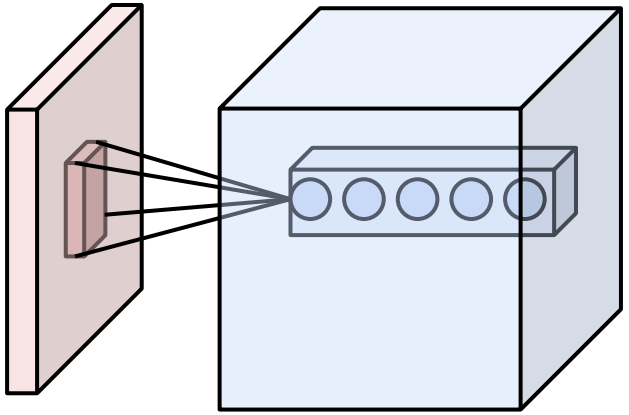
Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 3**
Number of neurons: **5**

Output volume: ?

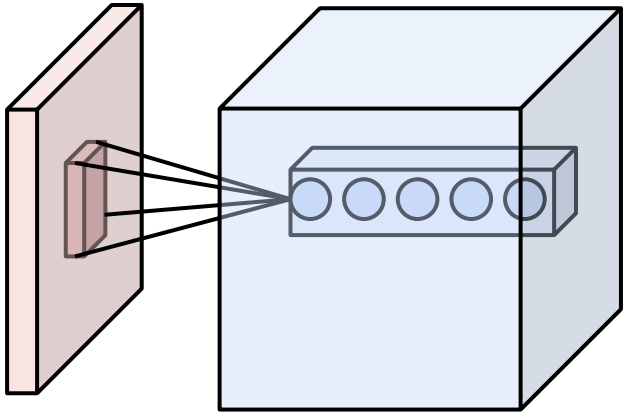
Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 3**
Number of neurons: **5**

Output volume: (32 - 5) / 3 + 1 = 10, so: **10x10x5**
How many weights for each of the 10x10x5 neurons?

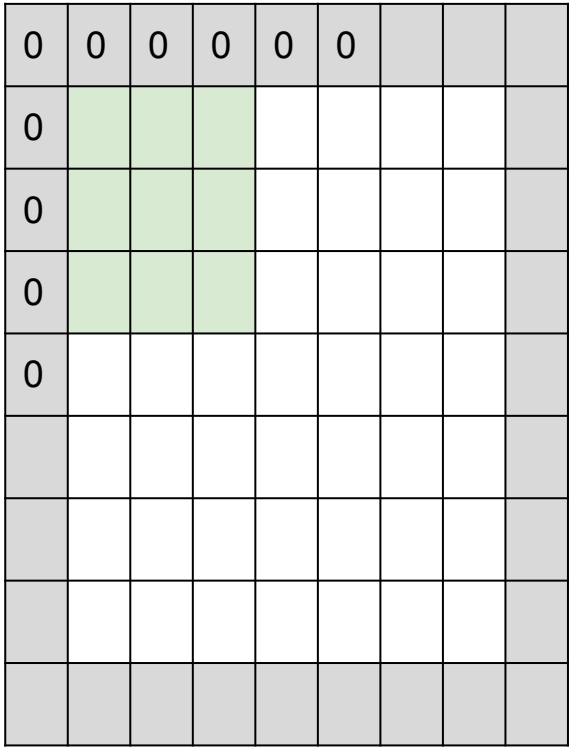
Examples time:



Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 3**
Number of neurons: **5**

Output volume: (32 - 5) / 3 + 1 = 10, so: **10x10x5**
How many weights for each of the 10x10x5 neurons? **5x5x3 = 75** (unchanged)

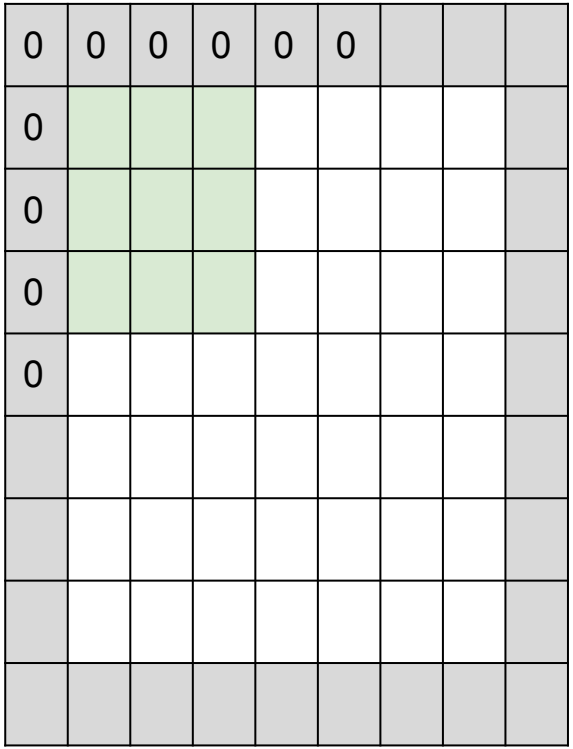
In practice: Common to zero pad the border



(in each channel)

e.g. input 7x7
neuron with receptive field 3x3, stride 1
pad with 1 pixel border => what is the output?

In practice: Common to zero pad the border



(in each channel)

e.g. input 7x7
neuron with receptive field 3x3, stride 1
pad with 1 pixel border => what is the output?

7x7 => preserved size!

in general, common to see stride 1, size F, and zero-padding with (F-1)/2.
(Will preserve input size spatially)

“Same convolution” (preserves size)

Input [9x9]

3x3 neurons, stride 1, pad **1** => [9x9]
3x3 neurons, stride 1, pad **1** => [9x9]

- No headaches when sizing architectures
- Works well

“Valid convolution” (shrinks size)

Input [9x9]

3x3 neurons, stride 1, pad **0** => [7x7]
3x3 neurons, stride 1, pad **0** => [5x5]

- **Headaches** with sizing the full architecture
- **Works Worse!** Border information will “wash away”, since those values are only used once in the forward function

Summary:

Input volume of size [W1 x H1 x D1]

using K neurons with receptive fields F x F and applying them at strides of S gives

Output volume: [W2, H2, D2]

W2 = (W1-F)/S+1	H2 = (H1-F)/S+1	D2 = K

There's one more problem…

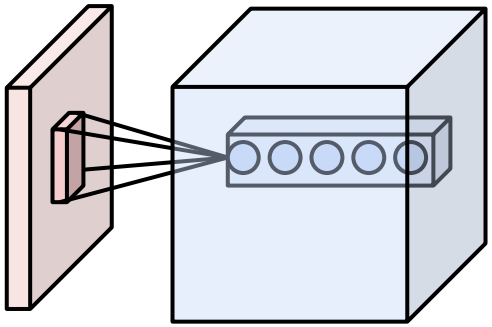
Assume input **[32 x 32 x3]**

30 neurons with receptive fields **5x5**, applied at **stride 1/pad1:**

=> Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)

Each neuron has 5*5*3 (=75) weights

=> Number of weights in such layer: 39720 * 75 ~= **3 million :**



There's one more problem…

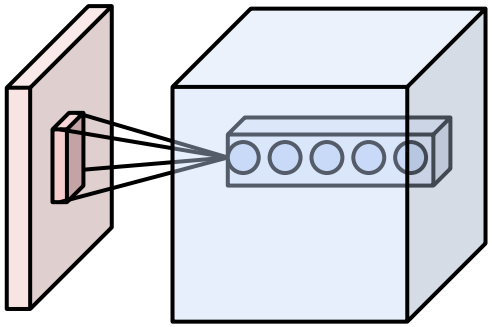
Assume input **[32 x 32 x3]**

30 neurons with receptive fields **5x5**, applied at **stride 1/pad 2:**

=> Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)

Each neuron has 5*5*3 (=75) weights

=> Number of weights in such layer: 39720 * 75 ~= **3 million :**



Example trained filters

There's one more problem…

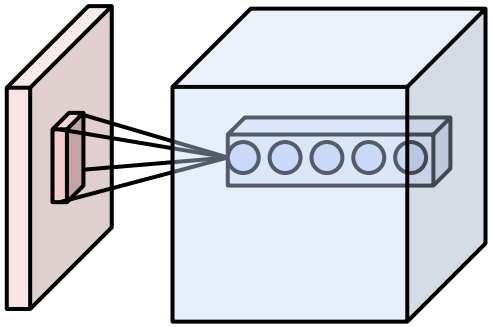
Assume input **[32 x 32 x3]**

30 neurons with receptive fields **5x5**, applied at **stride 1/pad1:**

=> Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)

Each neuron has 5*5*3 (=75) weights

=> Number of weights in such layer: 30720 * 75 ~= **3 million :**



Example trained weights

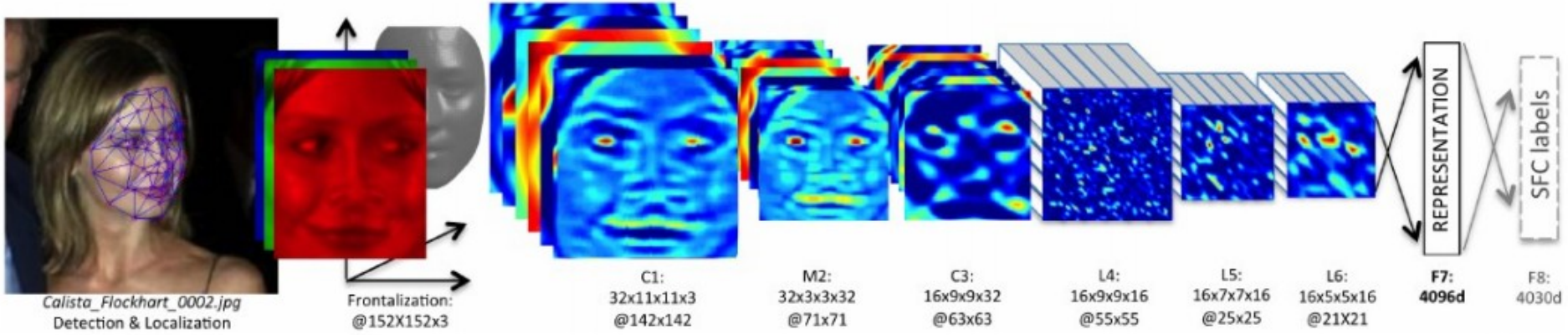
IDEA: lets not learn the same thing across all spatial locations

Our first ConvNet layer had size **[32 x 32 x3]**
If we had **30** neurons with receptive fields **5x5**, **stride 1, pad 1**
Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)
Each neuron has 5*5*3 (=75) weights

Before:
#weights in such layer: (32*32*30) * 75 = **3 million :**

Now: (paramater sharing)
#weights in the layer: 30 * 75 = **2250**.

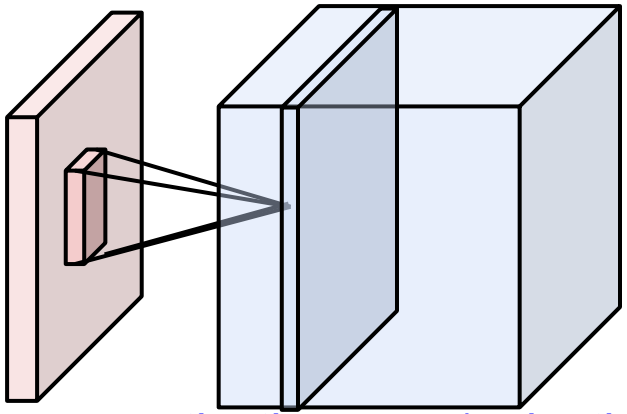
Note: sometimes it's not a good idea to share the parameters



We'd like to be able to learn different things at different spatial positions

These layers are called **Convolutional Layers**

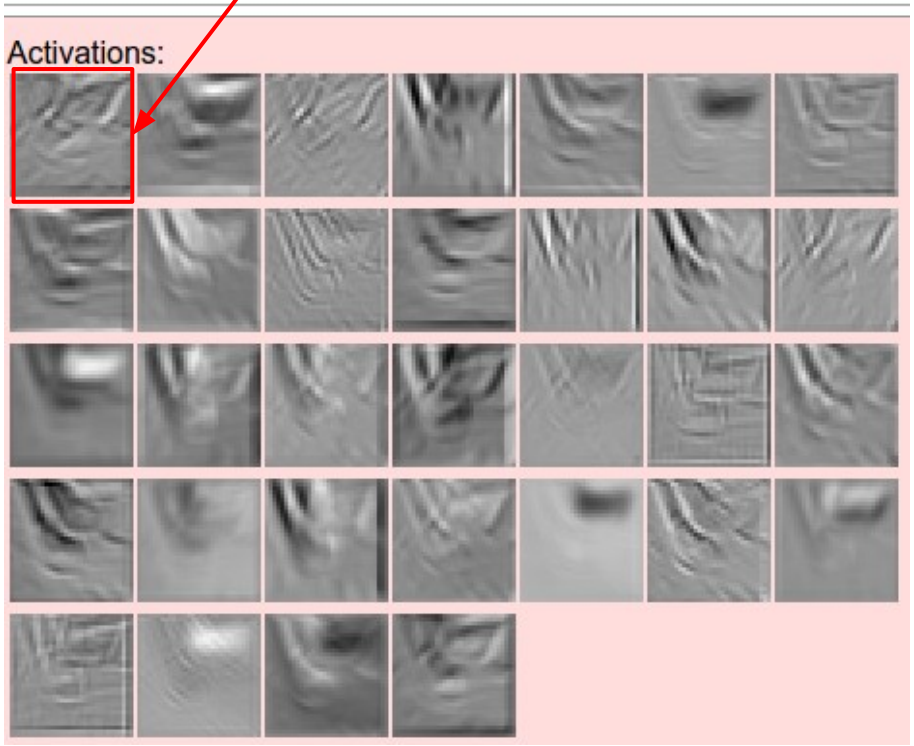
1. Connect neurons only to local receptive fields
2. Use the same neuron weight parameters for neurons in each “depth slice” (i.e. across spatial positions)



one activation map (a depth slice), computed with one set of weights

one filter = one depth slice (or activation map)

5x5 filters



Can call the neurons “filters”

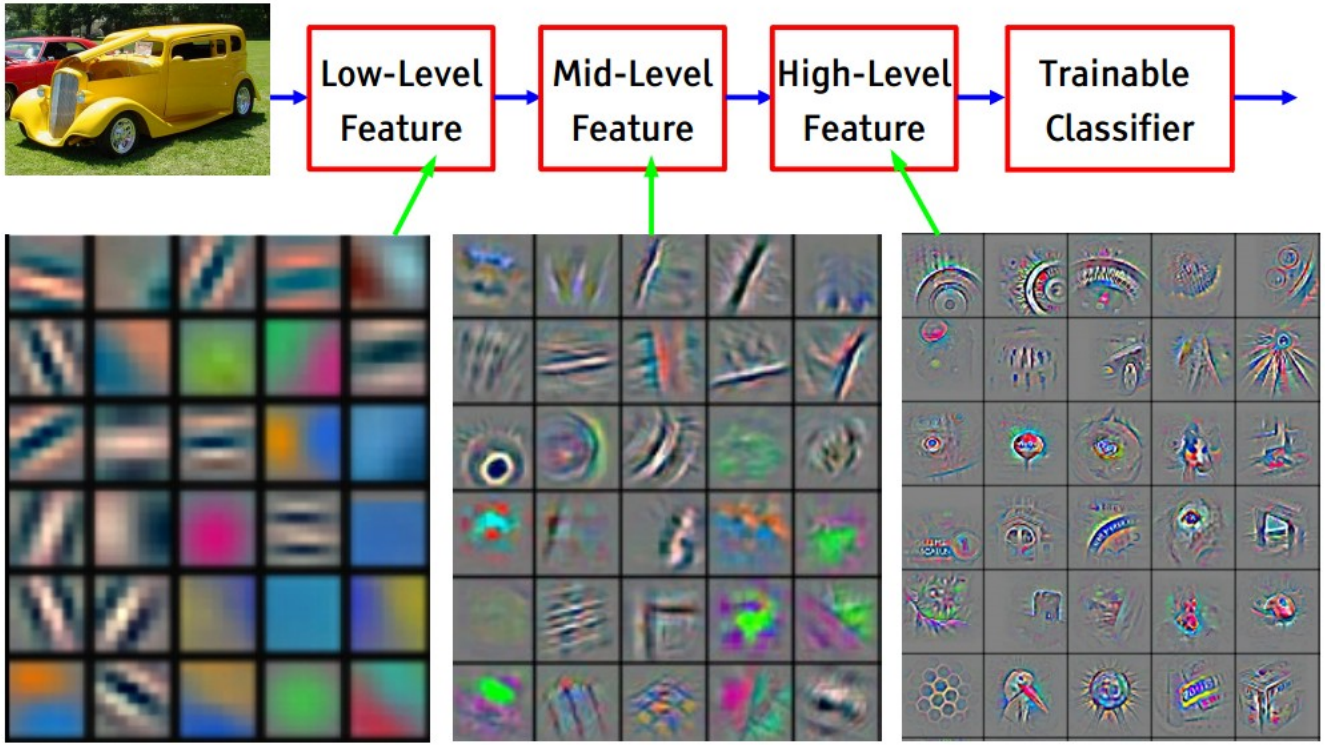
We call the layer convolutional because it is related to convolution of two signals (kind of):

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

elementwise multiplication and sum o
a filter and the signal (image)
= np.dot(w,x) + b

Fast-forward to today

[From recent Yann LeCun slides]

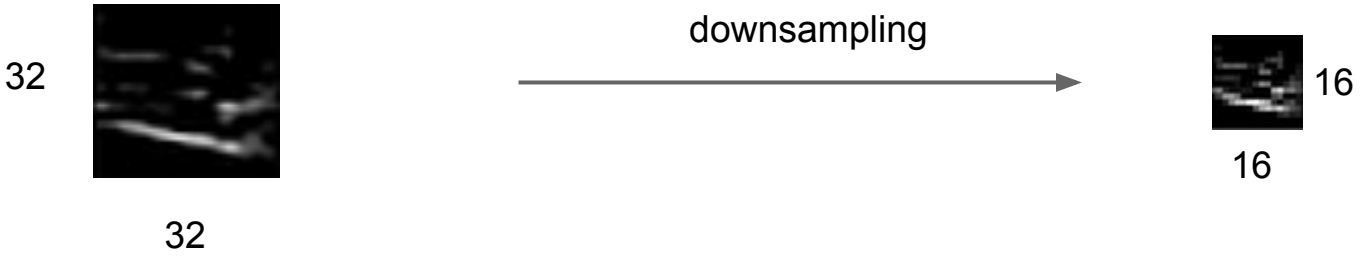


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

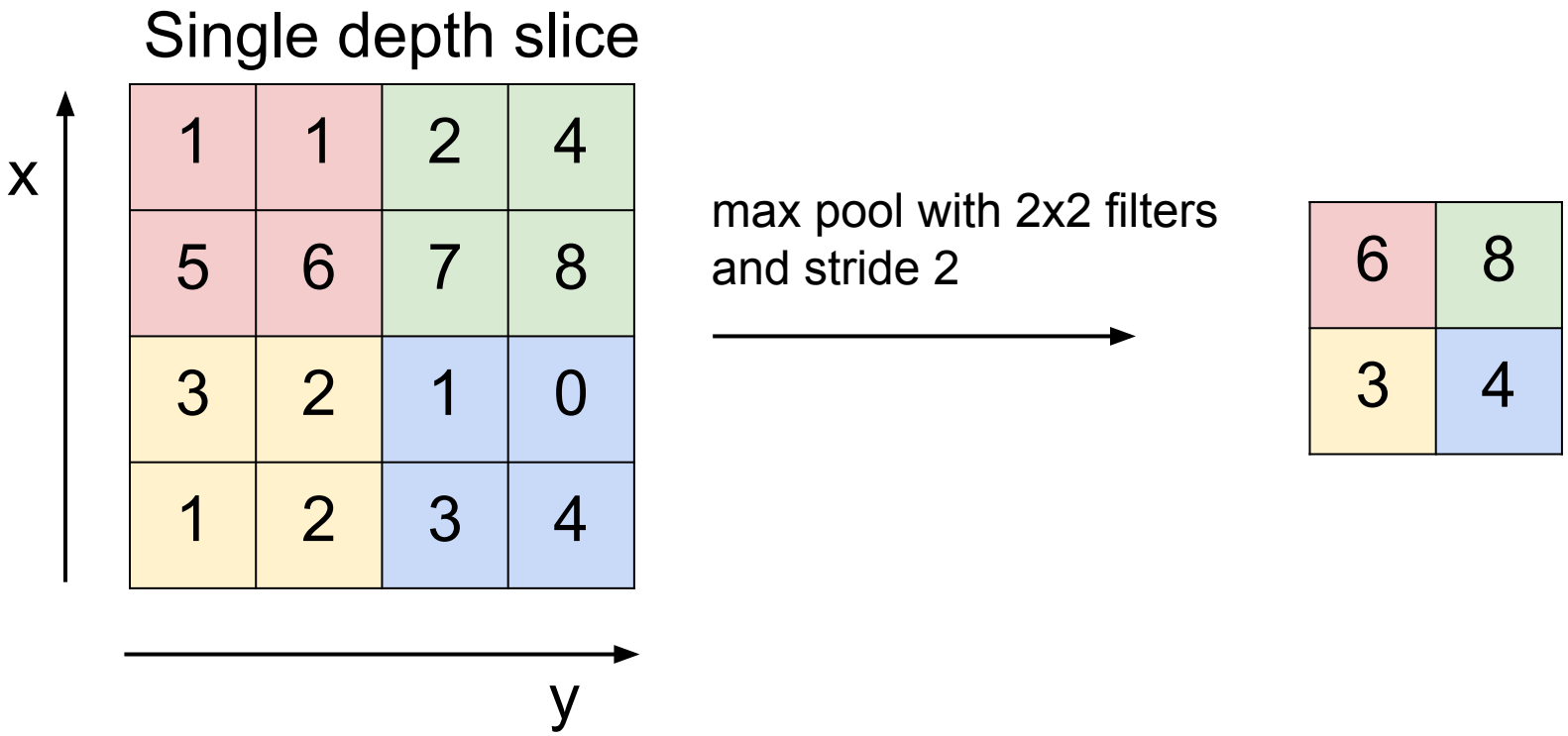

CONV
ReLU
CONV
ReLU
truck
car
airplane
ship
horse

In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

- convenience layer: makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)



MAX POOLING


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

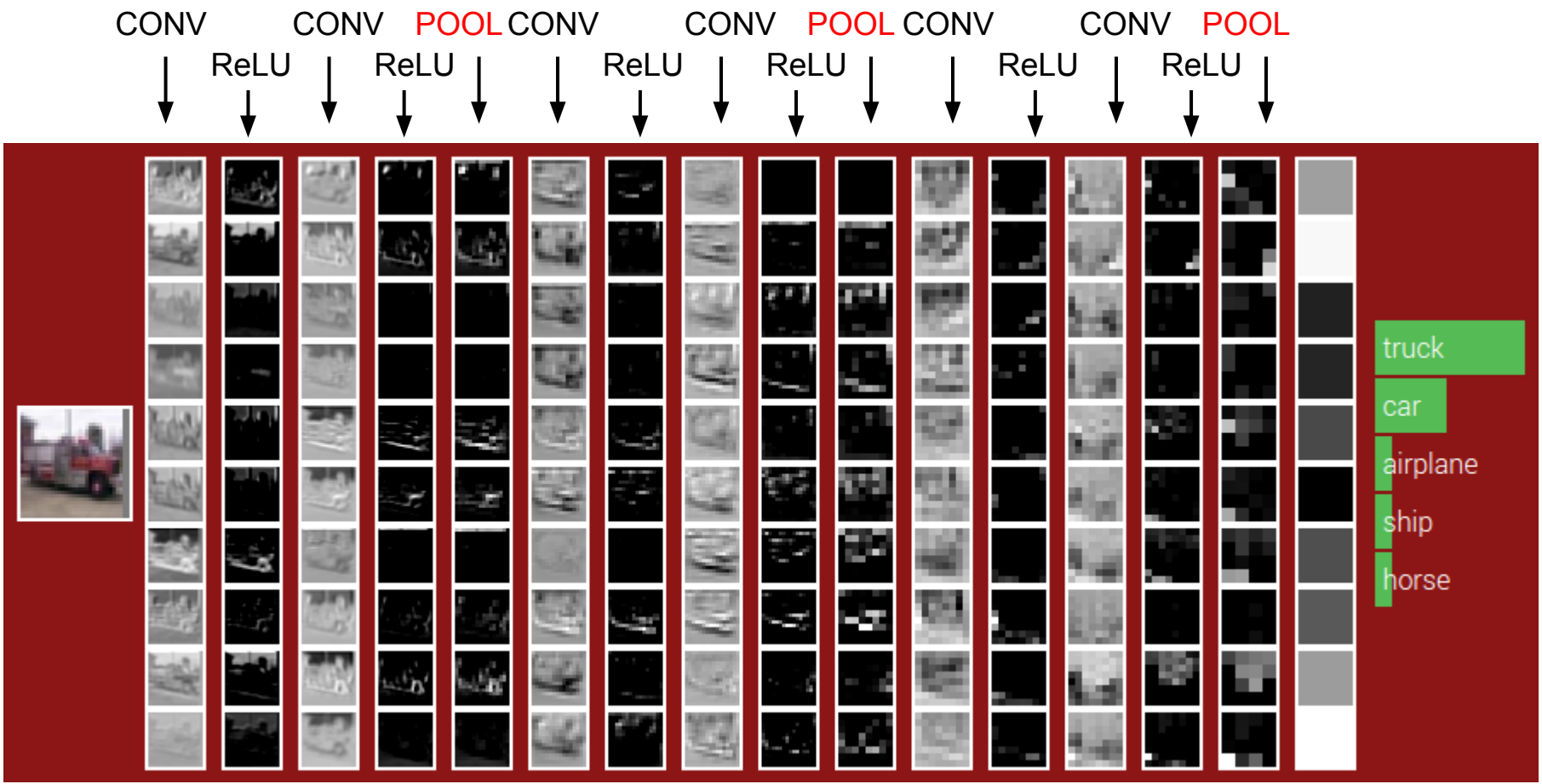
In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

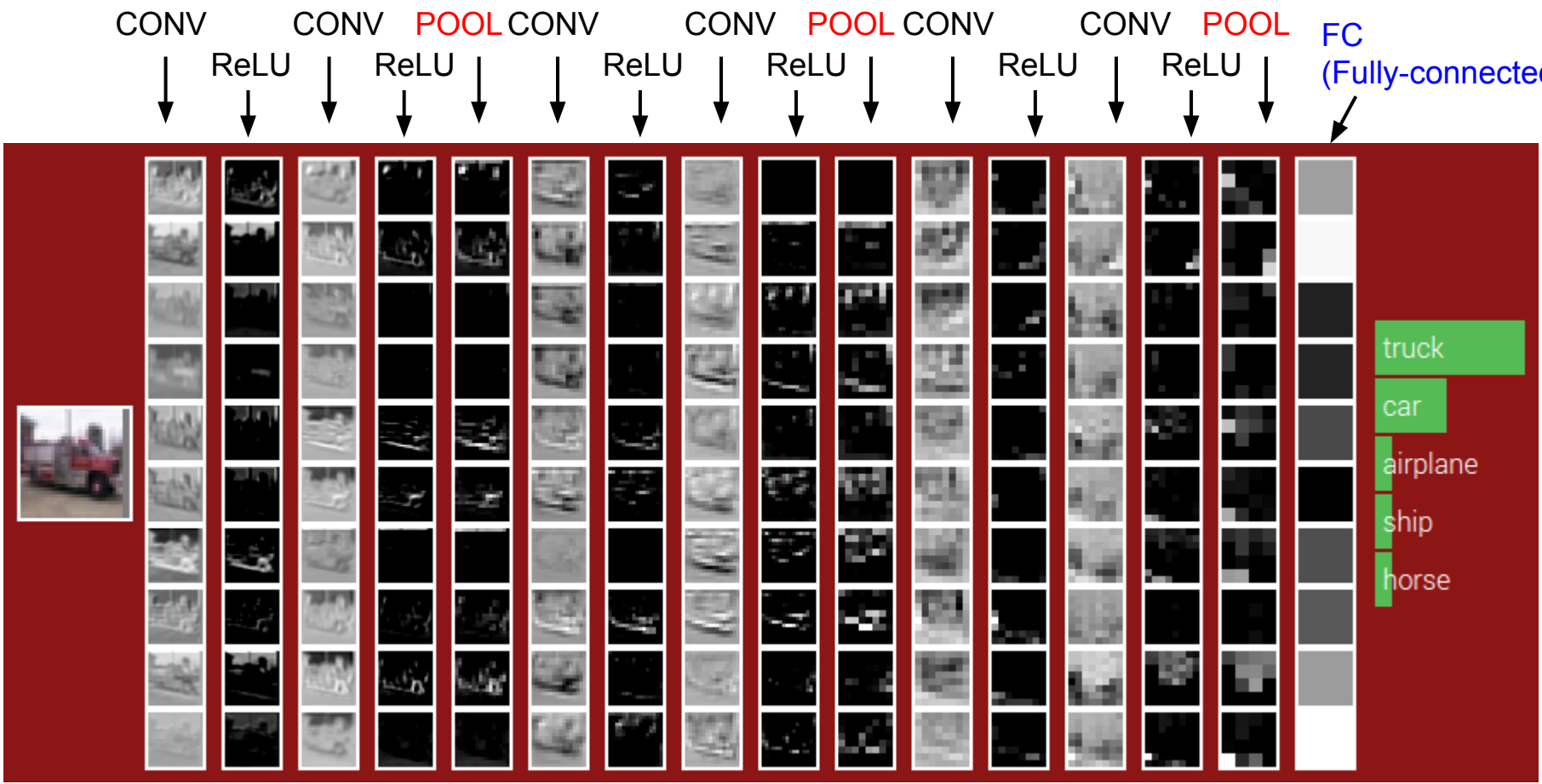
- convenience layer: makes the representations smaller and more manageable without losing too much information. Computes MAX operation (most common)

Input volume of size [W1 x H1 x D1]
Pooling unit receptive fields F x F and applying them at strides of S gives
Output volume: [W2, H2, D1]
W2 = (W1-F)/S+1, H2 = (H1-F)/S+1

Note: pooling happens independently across each slice, preserving number of slices
E.g. a pooling “neuron” of size 2x2 will perform MAX operation over 4 numbers.


CONV
ReLU
CONV
ReLU
POOL
truck
car
airplane
ship
horse


CONV
ReLU
CONV
ReLU
POOL
CONV
ReLU
CONV
ReLU
POOL
CONV
ReLU
CONV
ReLU
POOL
truck
car
airplane
ship
horse


CONV
ReLU
CONV
ReLU
POOL
CONV
ReLU
CONV
ReLU
POOL
CONV
ReLU
CONV
ReLU
POOL
FC
(Fully-connected)
truck
car
airplane
ship
horse

Modern CNNs:

- use **filter sizes of 3x3** (maybe even 2x2 or 1x1!)
- use **pooling sizes of 2x2** (maybe even less - e.g. fractional pooling!)
- **stride 1**
- **very deep**

(if too expensive for time/space, might have to downsample more, or make bigger strides, etc)

Eliminate sizing headaches TIPS/TRICKS

- start with image that has power-of-2 size
- for **conv layers**, use stride 1 filter size 3x3 pad input with a border of zeros (1 spatially)

This makes it so that: [W1,H1,**D1**] -> [W1,H1,**D2**] (i.e. spatial size exactly preserved)

- for **pool layers**, use pool size 2x2 (more = worse)

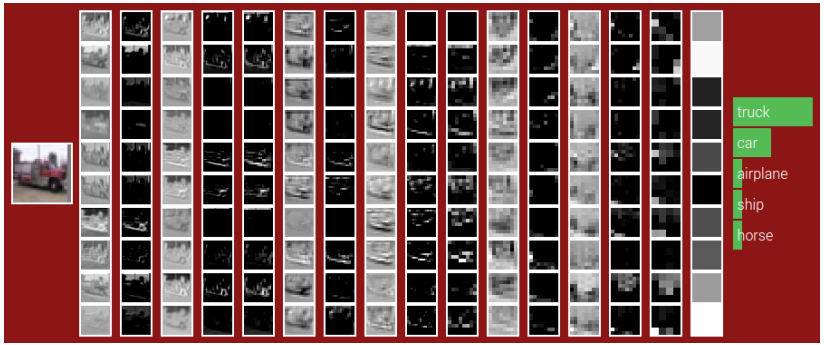
For big images, good heuristic is to violate the previous rules but only for the first layer. E.g. “AlexNet”:

input 227x227x3
first CONV layer: 96 11x11 filters, stride 4
=> (227 - 11)/4 + 1 = 55
=> output volume 55 x 55 x 96

...and from then stride 1 “same“ convolutions (with padding).

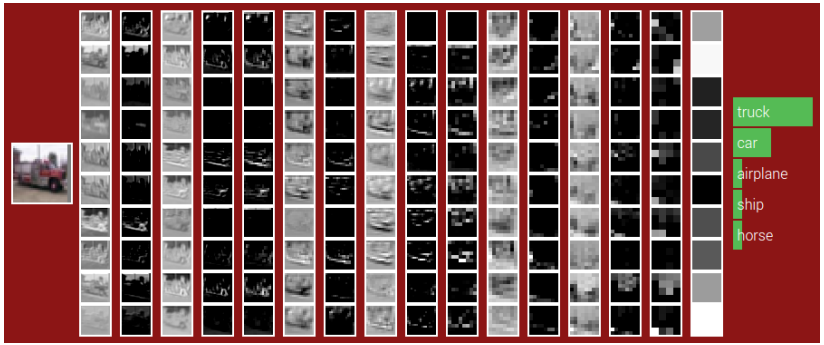
Example:

input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives:



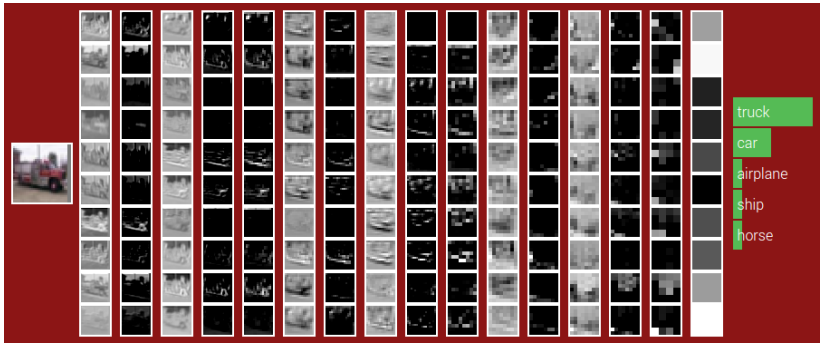
Example:

input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters:



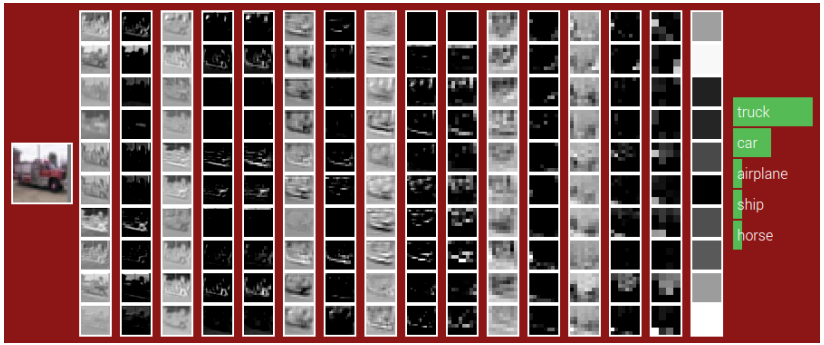
Example:

input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*3)*10 + 10 = 280
RELU



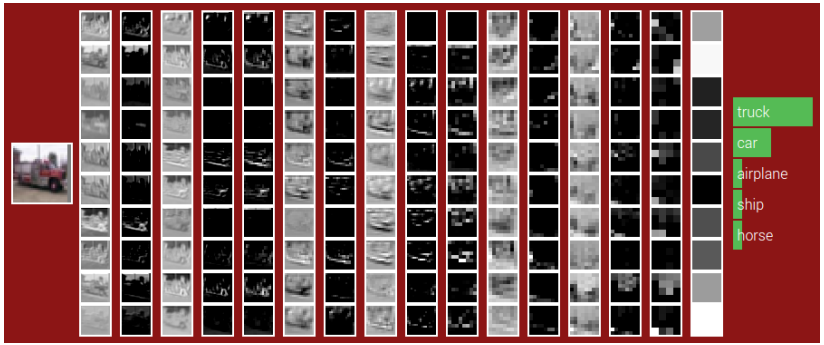
Example:

input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*3)*10 + 10 = 280
RELU
CONV with 10 3x3 filters, stride 1, pad 1:
gives:



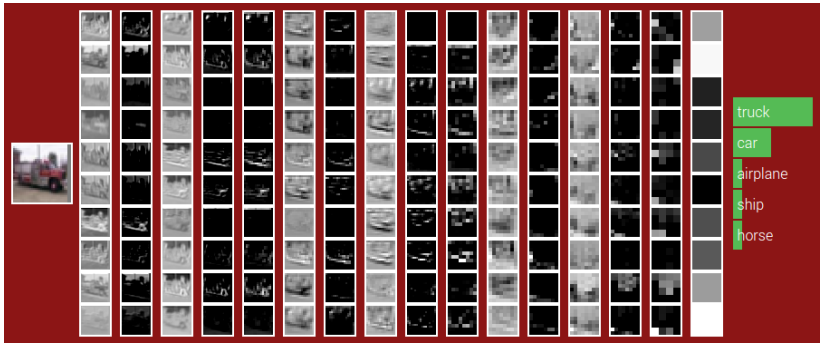
Example:

input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*3)*10 + 10 = 280
RELU
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters:



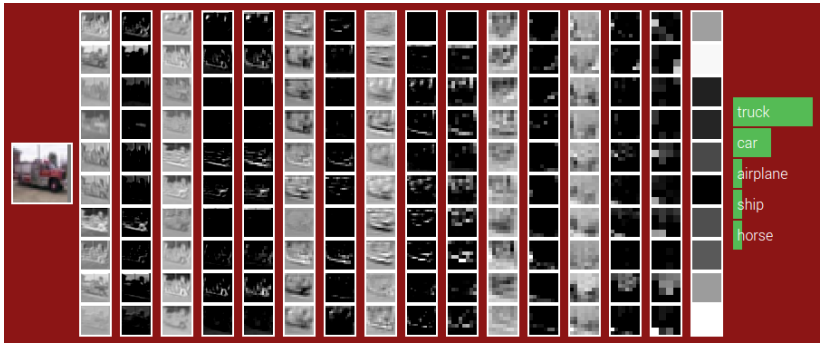
Example:

input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*3)*10 + 10 = 280
RELU
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*10)*10 + 10 = 910
RELU

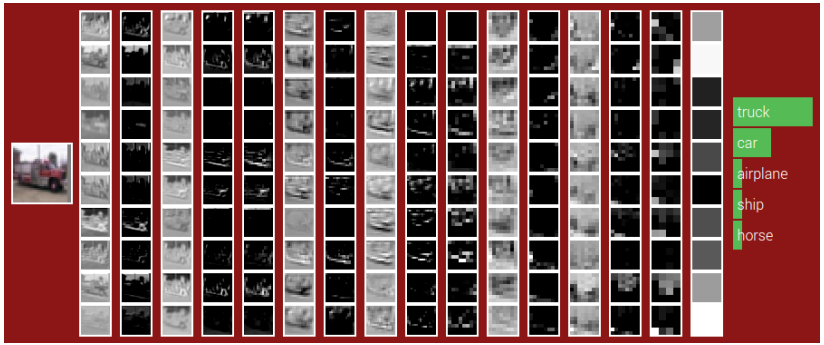


Example:

input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*3)*10 + 10 = 280
RELU
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*10)*10 + 10 = 910
RELU
POOL with 2x2 filters, stride 2:
gives:



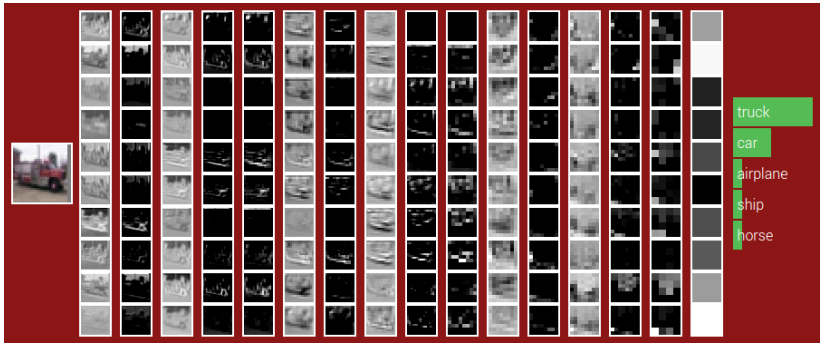
Example:



input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*3)*10 + 10 = 280
RELU
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*10)*10 + 10 = 910
RELU
POOL with 2x2 filters, stride 2:
gives: [16x16x10]
parameters:

Example:

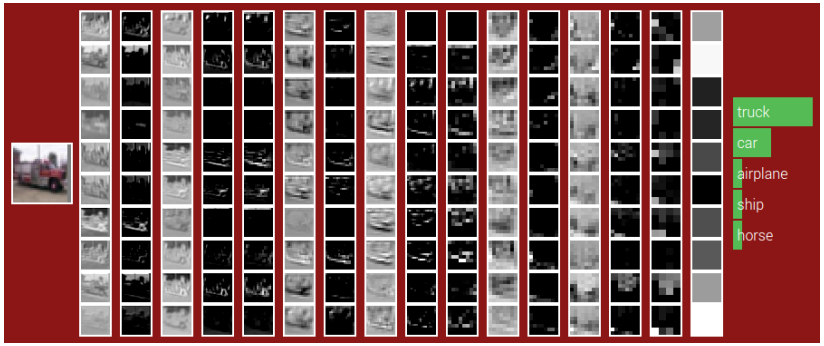
input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*3)*10 + 10 = 280
RELU
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*10)*10 + 10 = 910
RELU
POOL with 2x2 filters, stride 2:
gives: [16x16x10]
parameters: 0



Example:

input: [32x32x3]
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*3)*10 + 10 = 280
RELU
CONV with 10 3x3 filters, stride 1, pad 1:
gives: [32x32x10]
new parameters: (3*3*10)*10 + 10 = 910
RELU
POOL with 2x2 filters, stride 2:
gives: [16x16x10]
parameters: 0

(but in practice likely want to increase depth in larger layers)



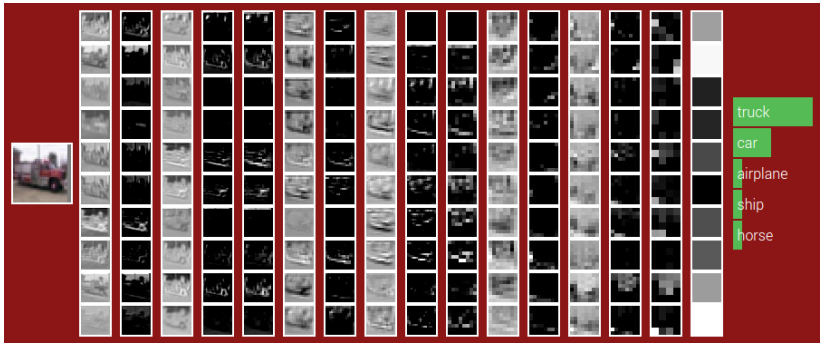
CONV with 10 3x3 filters, stride 1:
gives: [16x16x10]
new parameters: (3*3*10)*10 + 10 = 910
RELU
CONV with 10 3x3 filters, stride 1:
gives: [16x16x10]
new parameters: (3*3*10)*10 + 10 = 910
RELU
POOL with 2x2 filters, stride 2:
gives: [8x8x10]
parameters: 0

Example:

input: [32x32x3]
->

eventually we have after the last POOL:
[4x4x10]

Fully-Connected FC layer to 10 neurons
(which are our class scores)
Number of parameters:



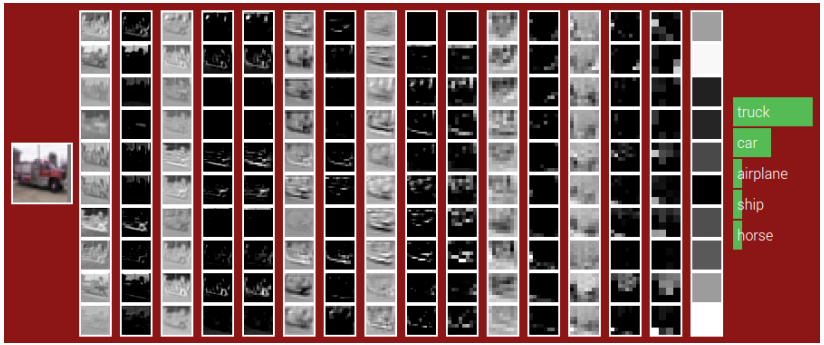
Example:

input: [32x32x3]
->

eventually we have after the last POOL:
[4x4x10]

Fully-Connected FC layer to 10 neurons
(which are our class scores)
Number of parameters:
10 * 4 * 4 * 10 + 10 = 1600

done!

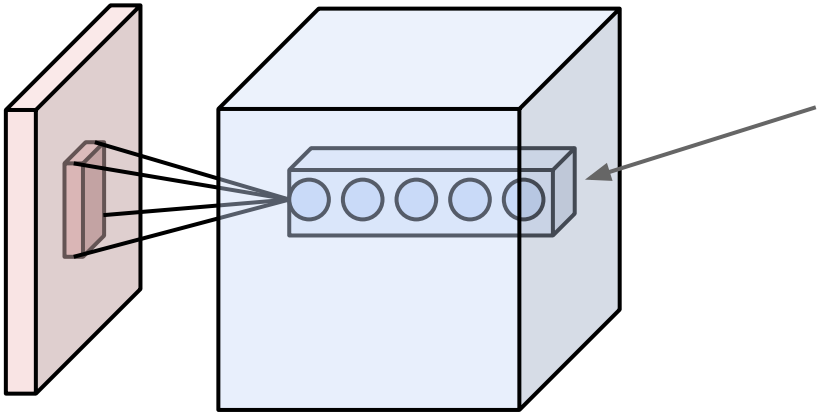


(show convnetjs example of training of CIFAR-10)

demo

Side note:

you may hear/read about **Normalization layers**
these have been deprecated; Unnecessary.



Case study: VGGNet / OxfordNet

(runner-up winner of ILSVRC 2014)

[Simonyan and Zisserman]

best model

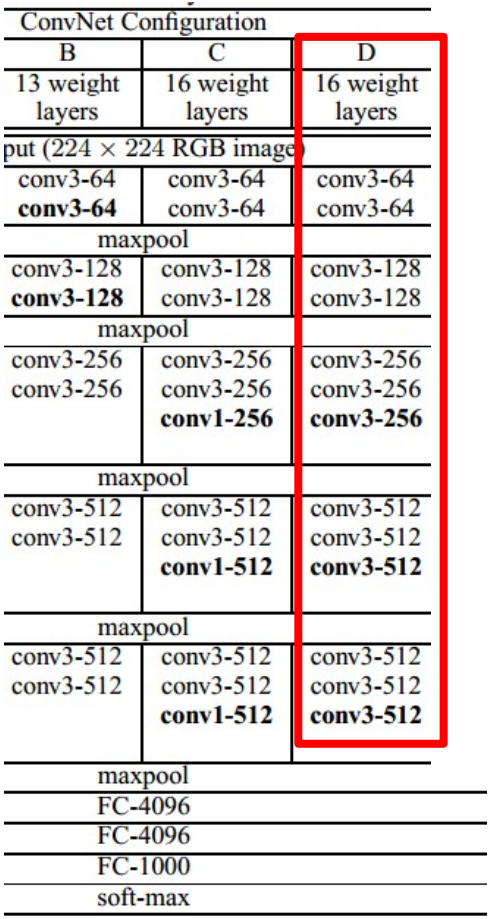
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-25 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

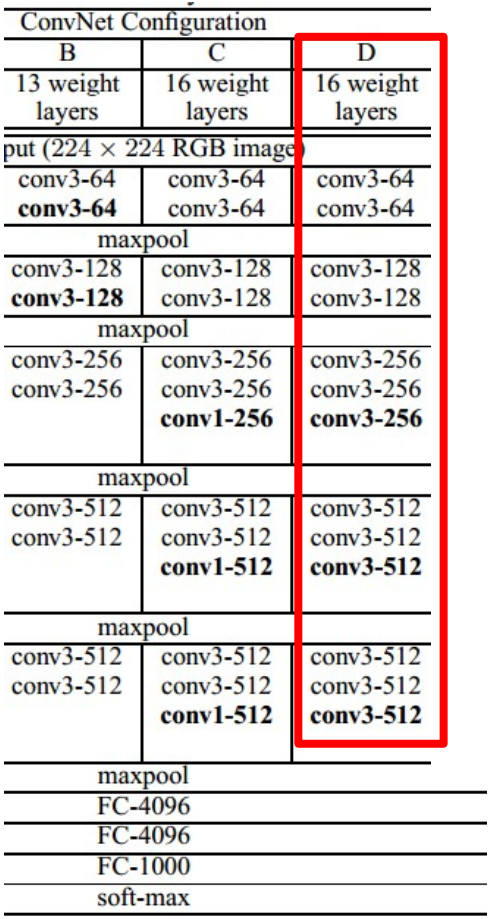


ConvNet Configuration		
B	C	D
13 weight layers	16 weight layers	16 weight layers
put (224 × 224 RGB image)		
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64
maxpool		
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128
maxpool		
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
FC-4096		
FC-4096		
FC-1000		
soft-max		

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters



ConvNet Configuration		
B	C	D
13 weight layers	16 weight layers	16 weight layers
put (224 × 224 RGB image)		
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64
maxpool		
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128
maxpool		
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**
maxpool		
FC-4096		
FC-4096		
FC-1000		
soft-max		

(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

Note:

Most memory is in early CONV

Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Summary:

- ConvNets are biologically-inspired architectures made up of Neural Net stuff
- Two key differences to Vanilla Neural Nets: neurons arranged in **3D volumes** have **local connectivity**, **share parameters**.
- Typical ConvNets look like: [CONV-RELU-POOL]xN,[FC-RELU]xM, SOFTMAX or [CONV-RELU-CONV-RELU-POOL]xN,[FC-RELU]xM,SOFTMAX (last FC layer should not have RELU - these are the class scores)
- Use **small filter sizes / pooling sizes** (but might be expensive). Otherwise: use smaller input, or higher filter sizes, strides
- Most memory/compute is usually in early Conv layers. Most parameters are usually in late, FC layers

Next Lecture:

Understanding / Visualizing ConvNets