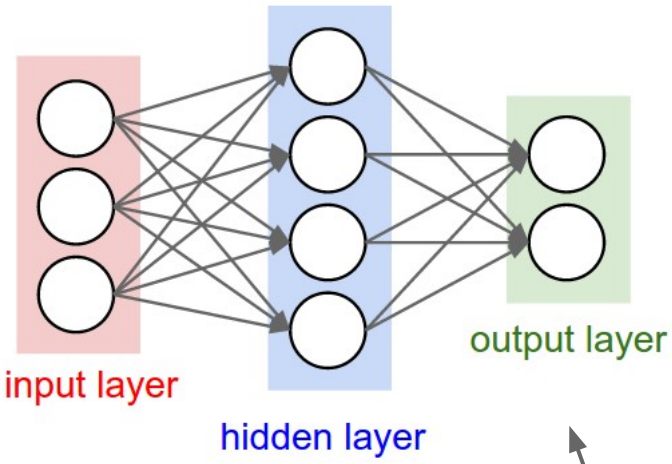
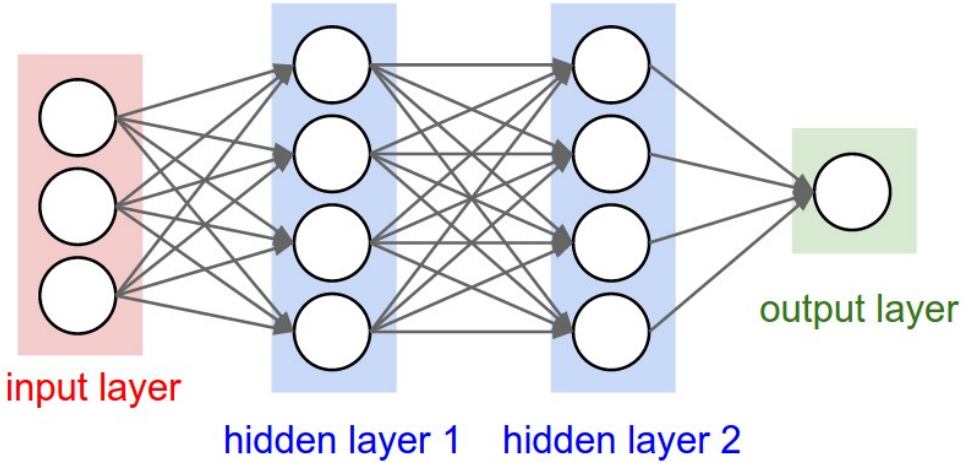
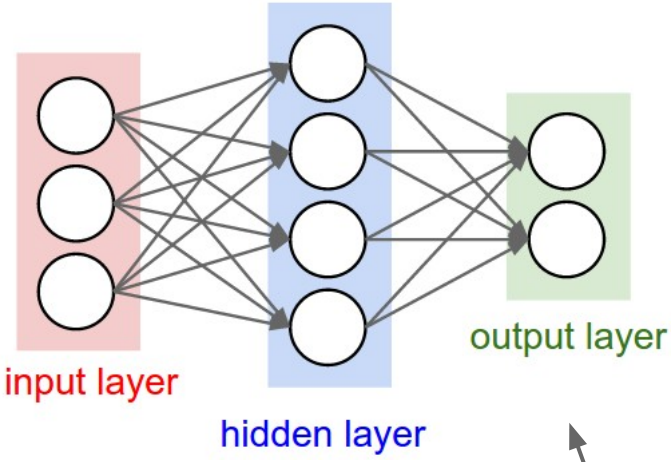
Neural Networks: Architectures

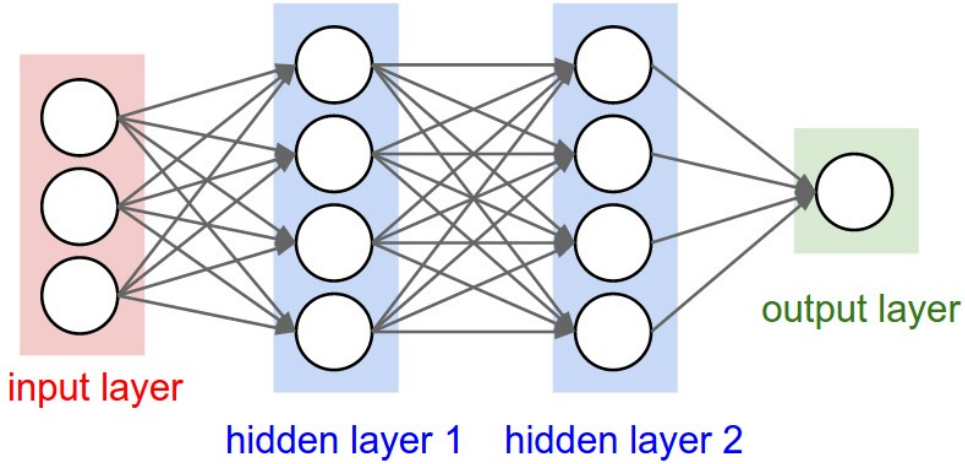




“Fully-connected” layers

Neural Networks: Architectures



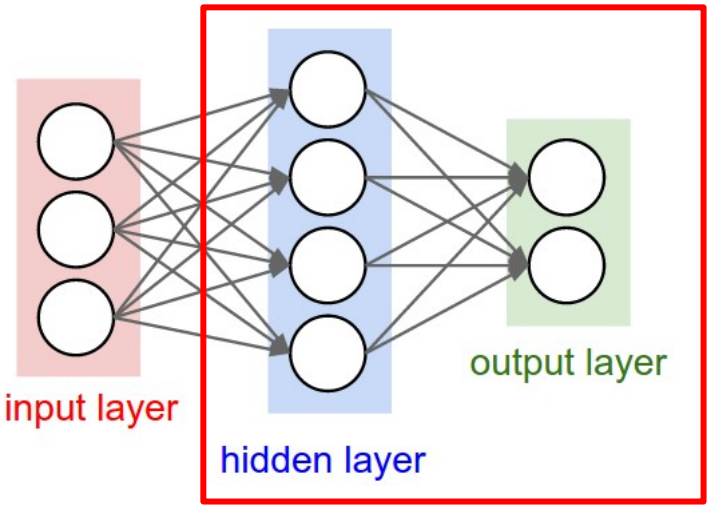


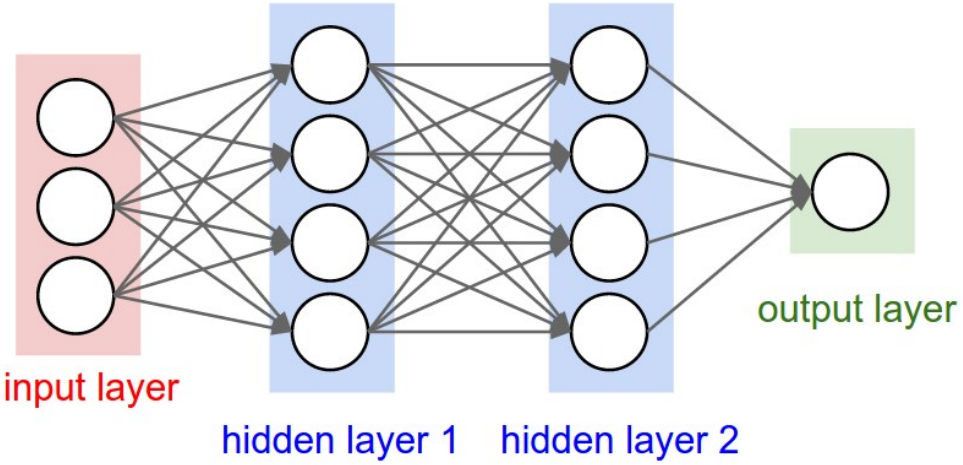
"2-layer Neural Net", or
"1-hidden-layer Neural Net"

"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

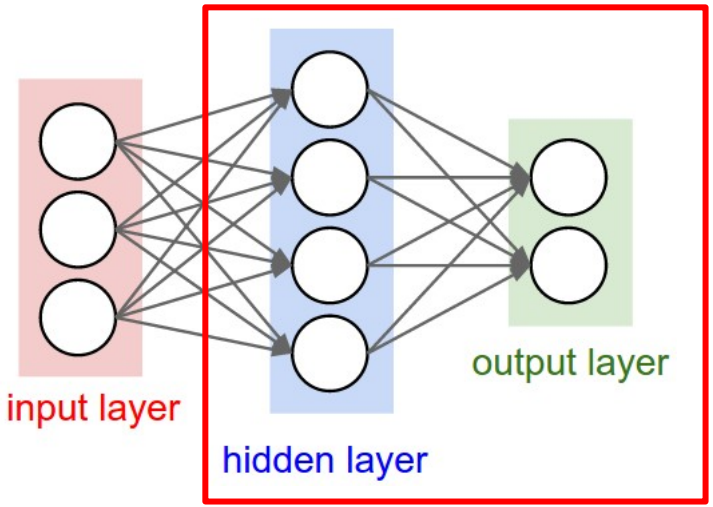
Neural Networks: Architectures

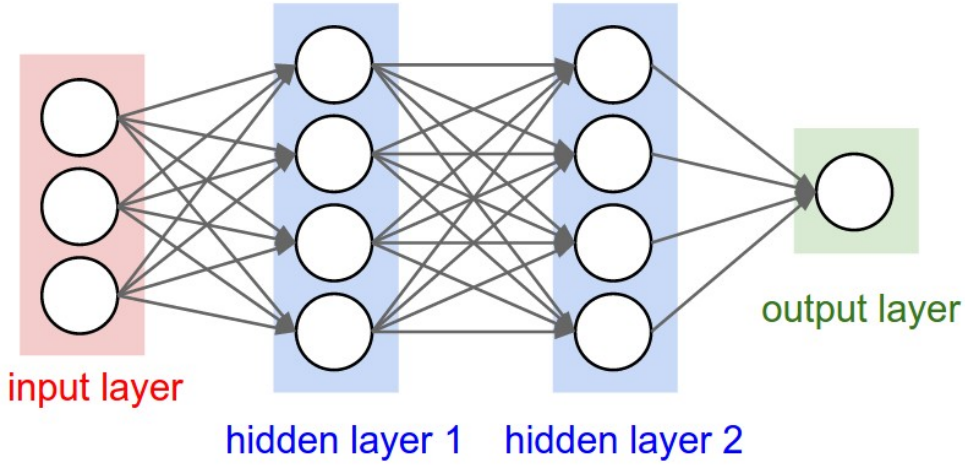




Number of Neurons: ?
Number of Weights: ?
Number of Parameters: ?

Neural Networks: Architectures

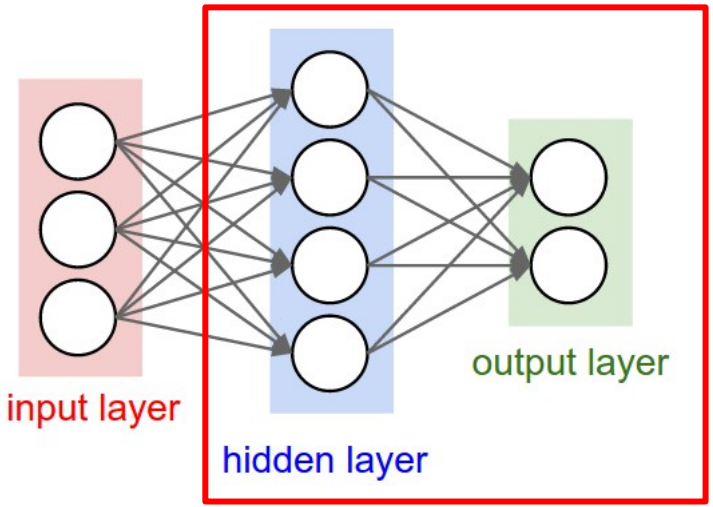


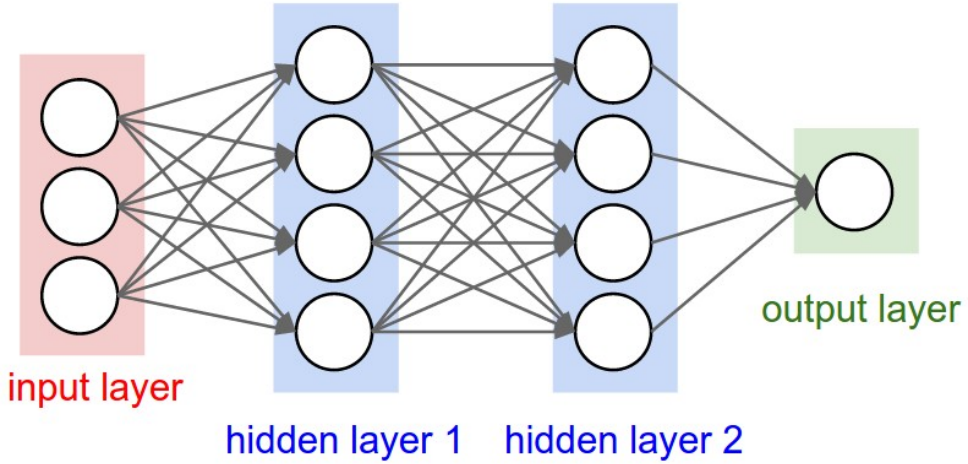


Number of Neurons: 4+2 = 6
Number of Weights: [4x3 + 2x4] = 20
Number of Parameters: 20 + 6 = 26 (biases!)

Number of Neurons: ?
Number of Weights: ?
Number of Parameters: ?

Neural Networks: Architectures

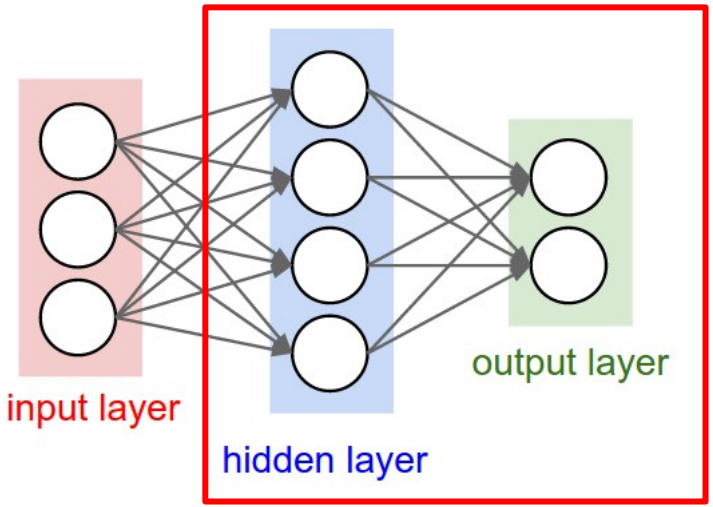


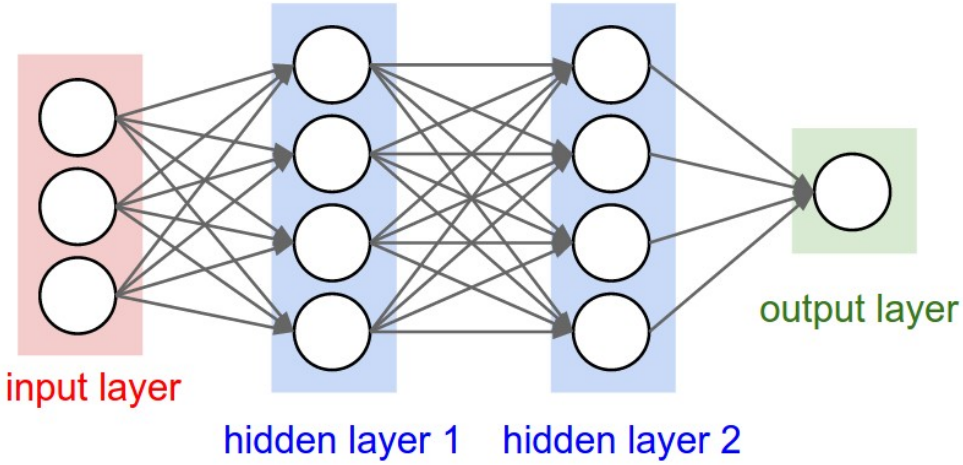


Number of Neurons: 4+2 = 6
Number of Weights: [4x3 + 2x4] = 20
Number of Parameters: 20 + 6 = 26 (biases!)

Number of Neurons: 4 + 4 + 1 = 9
Number of Weights: [4x3+4x4+1x4]=32
Number of Parameters: 32+9 = 41

Neural Networks: Architectures





Modern CNNs: ~10 million neurons
Human visual cortex: ~5 billion neurons

Image Classification: a core task in Computer Vision

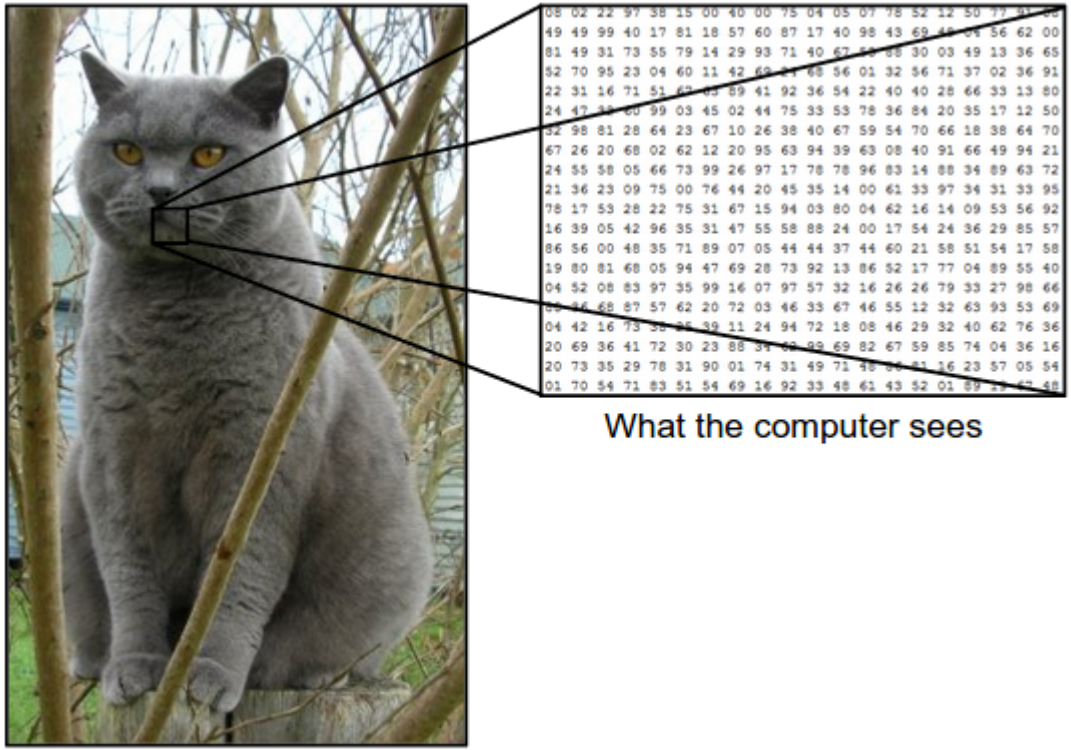
(assume given set of discrete labels)
{dog, cat, truck, plane, ...}

cat

The problem:
semantic gap

Images are represented as R^d arrays of numbers

- E.g. R^3 with integers between [0, 255], where d=3 represents 3 color channels (RGB)

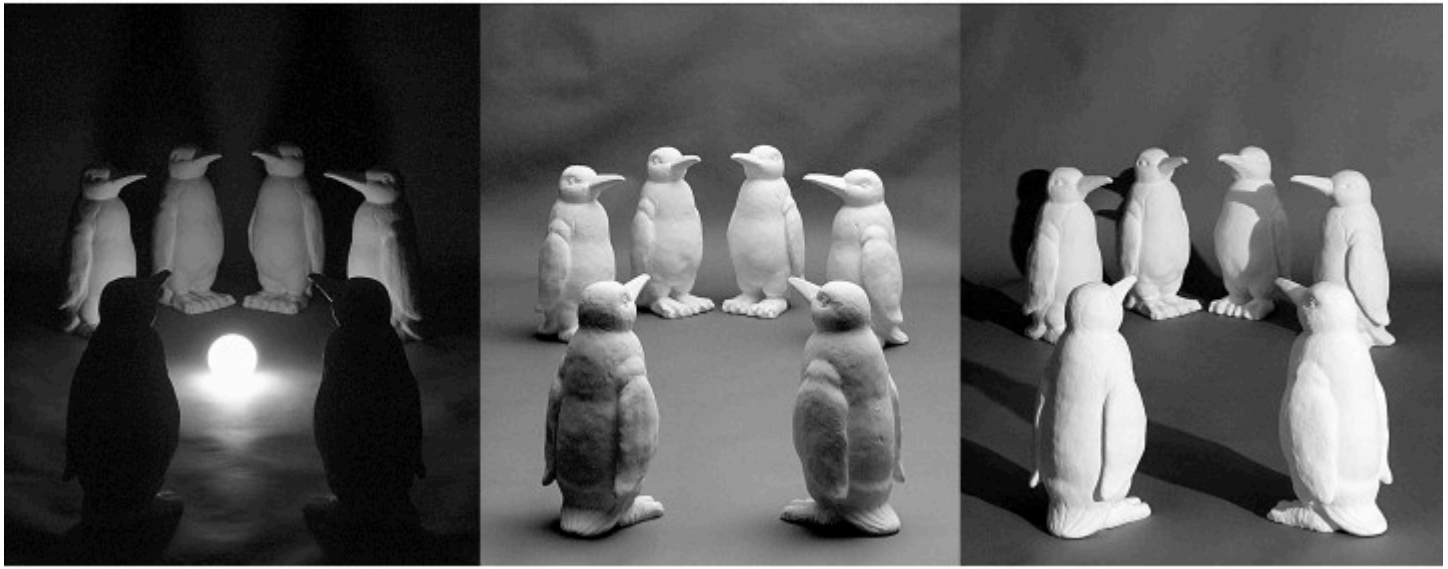


What the computer sees

Challenges: viewpoint variation

Michelangelo 1475-1564

Challenges: illumination



Challenges: scale

Challenges: deformation

Challenges: occlusion

Magritte, 1957

Challenges: background clutter

Kilmeny Niland. 1995

Challenges: intra-class variation






An image classifier

```
def predict(image):
    # ????
    return class_label
```

Unlike e.g. sorting a list of numbers,

no obvious way to hard-code the algorithm for recognizing a cat, or other classes.

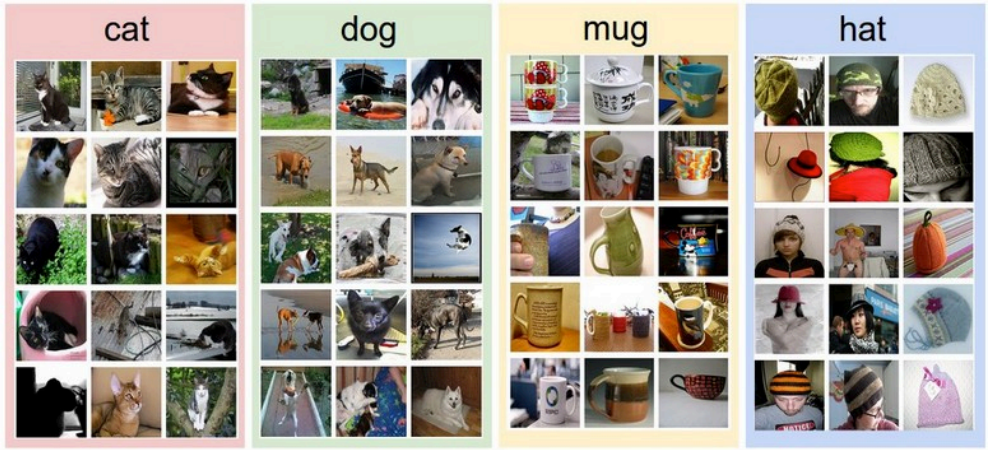
Data-driven approach:

1. Collect a dataset of images and label them
2. Use Machine Learning to train an image classifier
3. Evaluate the classifier on a withheld set of test images

```
def train(train_images, train_labels):
    # build a model of images -> labels

def predict(image):
    # evaluate the model on the image
    return class_label
```

Example training set

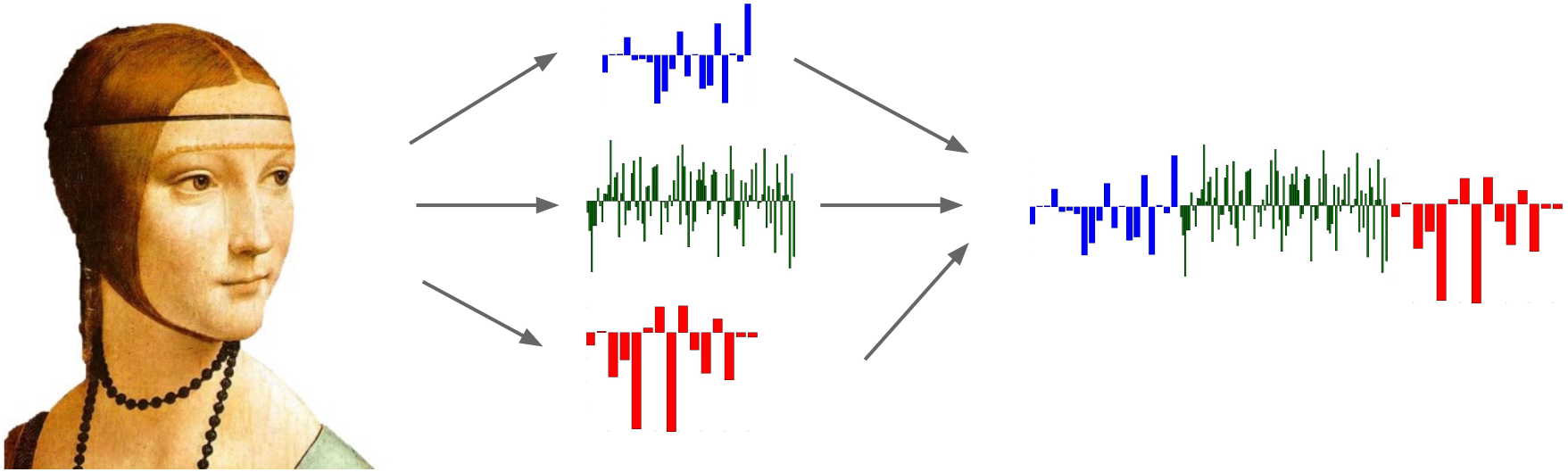


Brief aside: Image Features

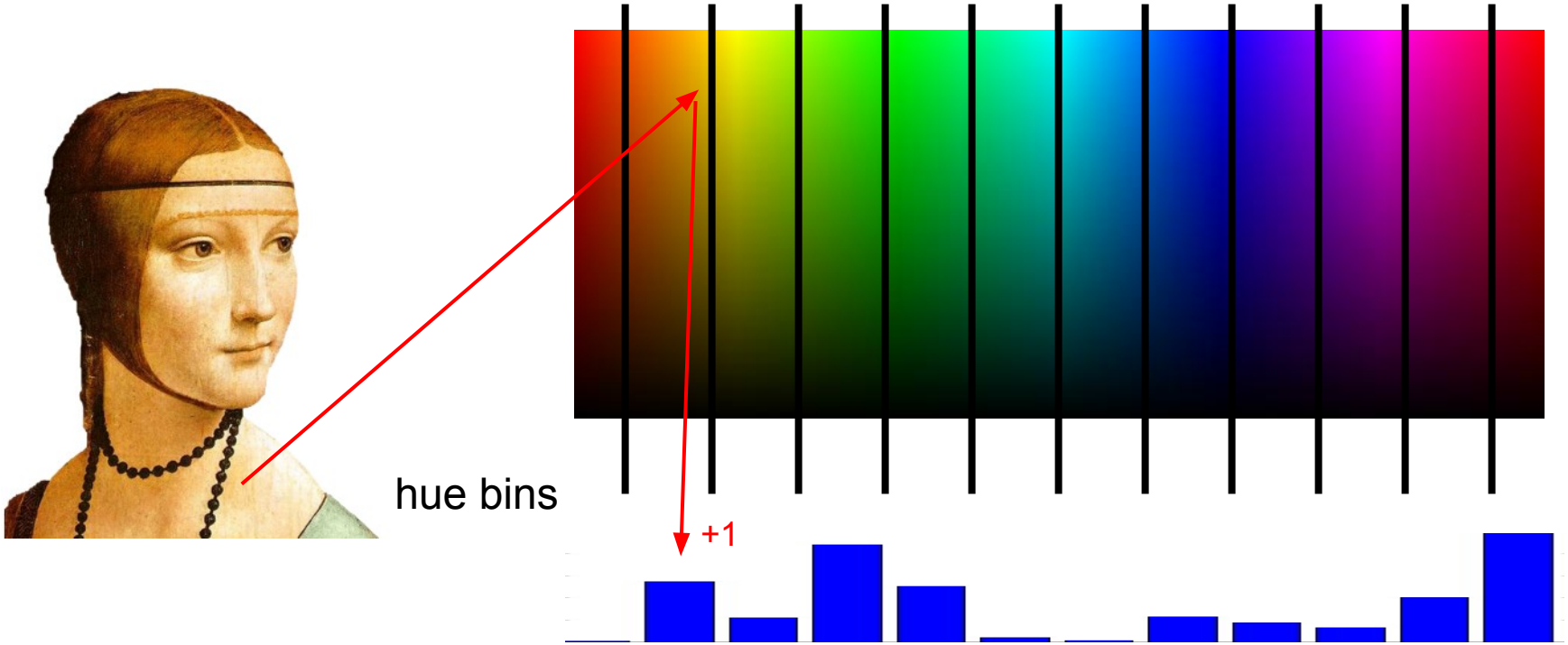
- In practice, very rare to see Computer Vision applications that train linear classifiers on pixel values

Brief aside: Image Features

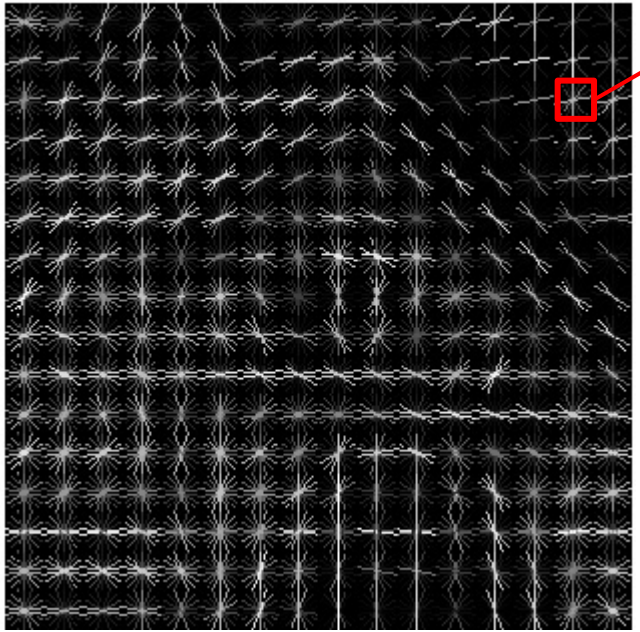
- In practice, very rare to see Computer Vision applications that train linear classifiers on pixel values



Example: Color (Hue) Histogram



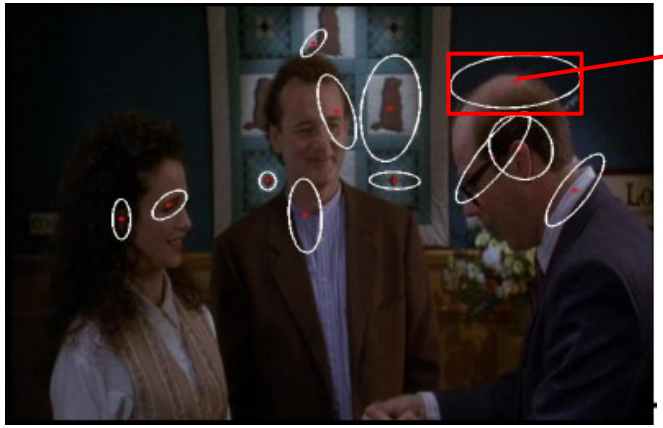
Example: HOG features



8x8 pixel region,
quantize the edge
orientation into 9 bins

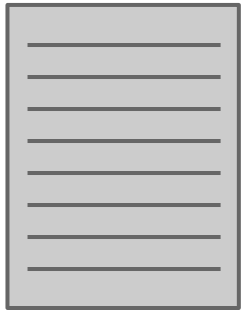
(images from vlfeat.org)

Example: Bag of Words



1. Resize patch to a fixed size (e.g. 32x32 pixels)
2. Extract HOG on the patch (get 144 numbers)

repeat for each detected feature



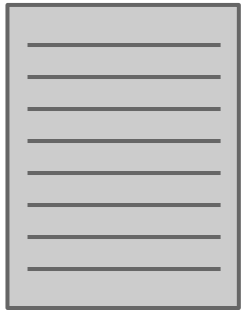
gives a matrix of size
[number_of_features x 144]

Problem: different images will have different numbers of features. Need fixed-sized vectors for linear classification

Example: Bag of Words

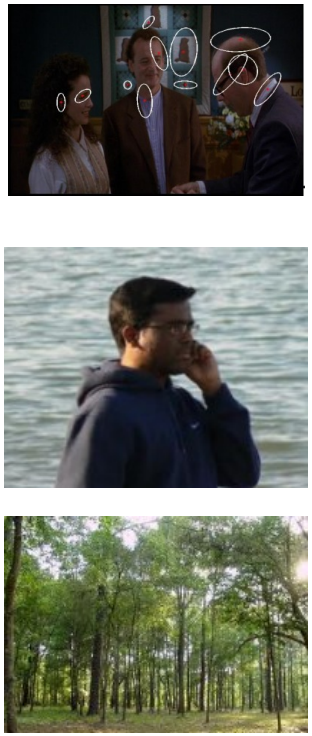
1. Resize patch to a fixed size (e.g. 32x32 pixels)
2. Extract HOG on the patch (get 144 numbers)

repeat for each detected feature

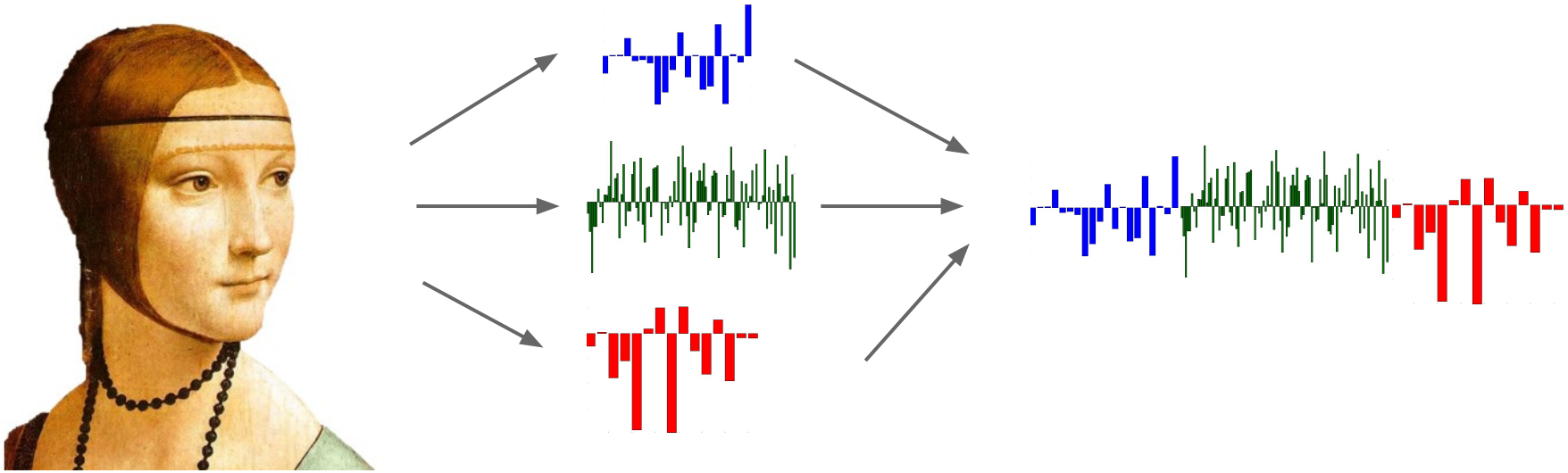


gives a matrix of size
[number_of_features x 144]

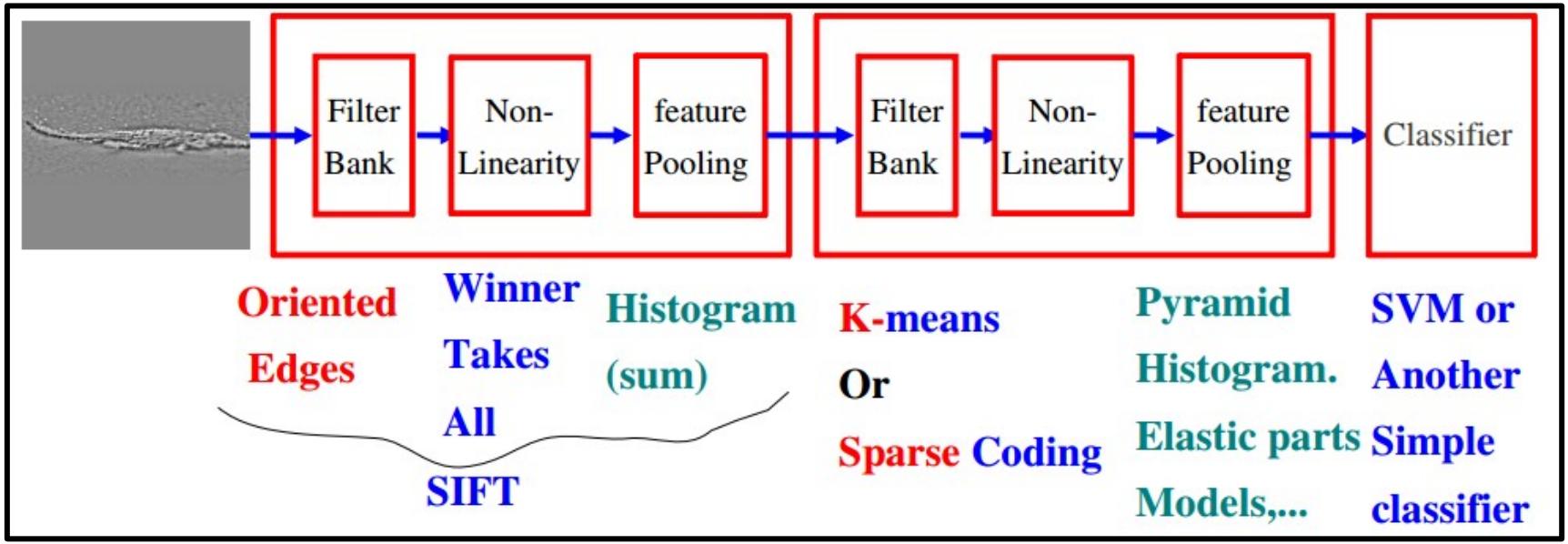
Example: Bag of Words



Brief aside: Image Features

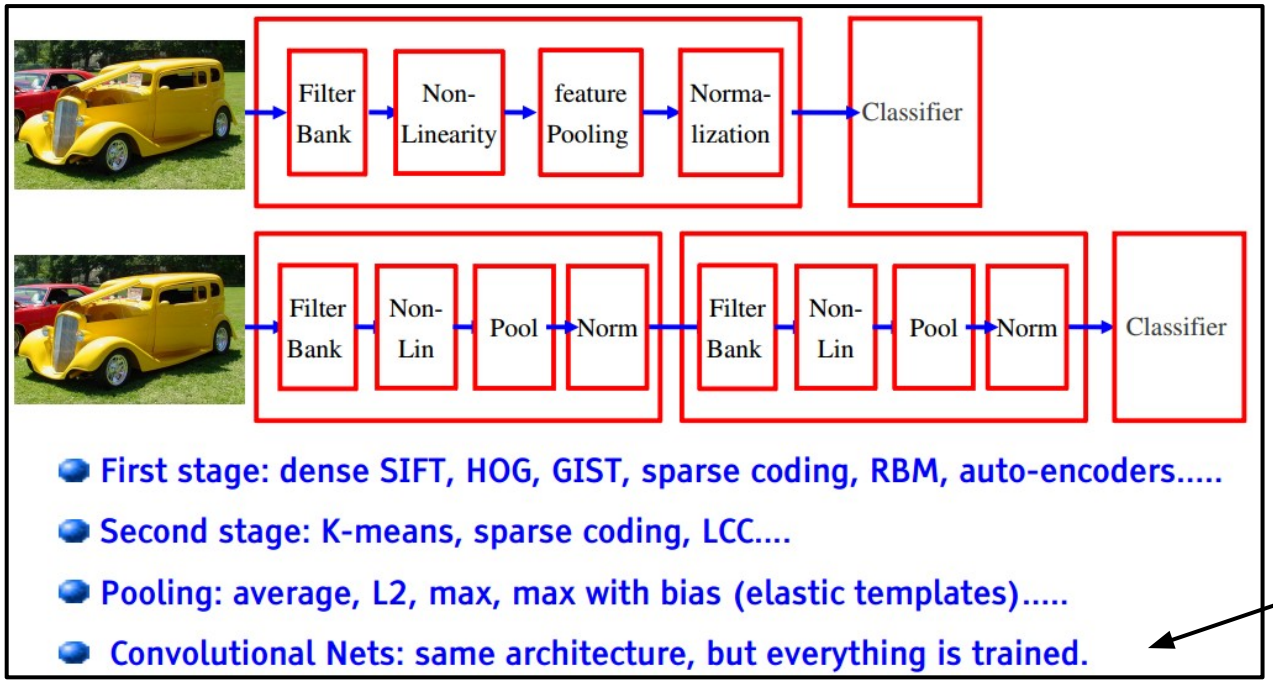


Most recognition systems are build on the same Architecture



(slide from Yann LeCun)

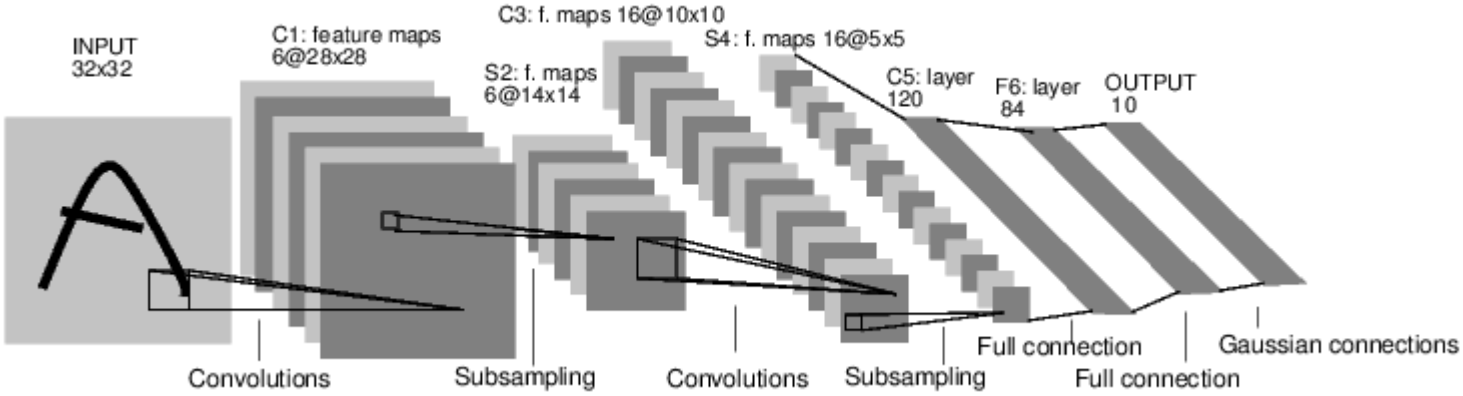
Most recognition systems are build on the same Architecture



CNNs: end-to-end models

(slide from Yann LeCun)

Lecture 7: Convolutional Neural Networks



[LeNet-5, LeCun 1980]

A bit of history:

Hubel & Wiesel,

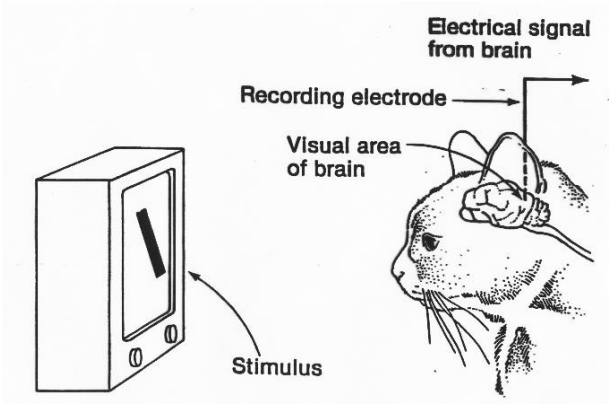
1959

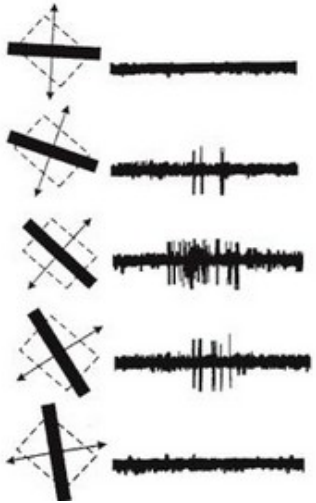
RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

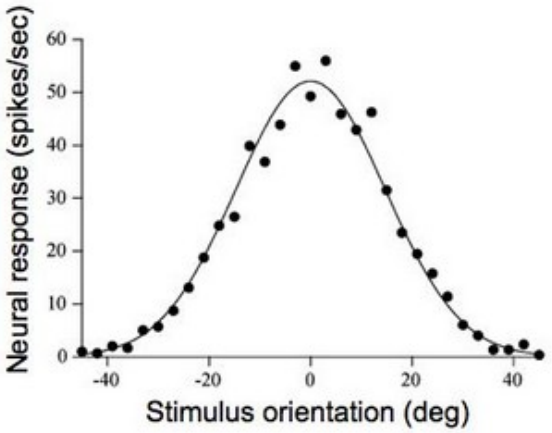
1962

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

1968...

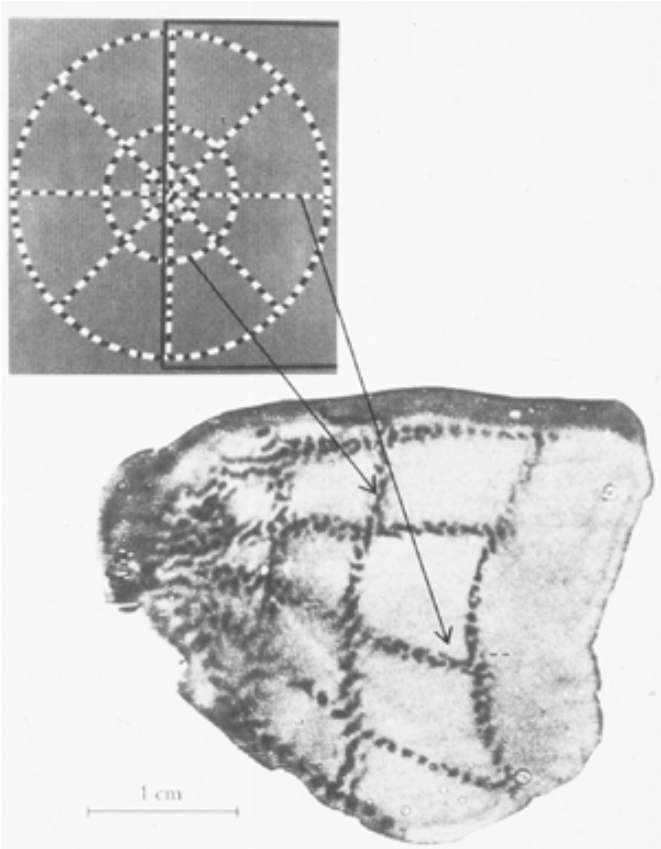






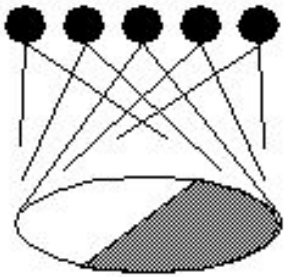
A bit of history

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

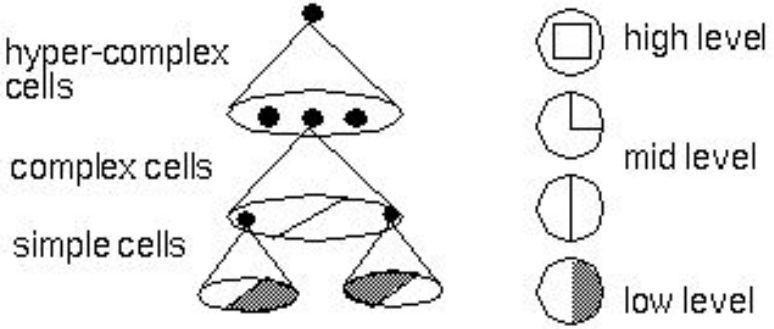


Hubel & Weisel

topographical mapping



featural hierarchy

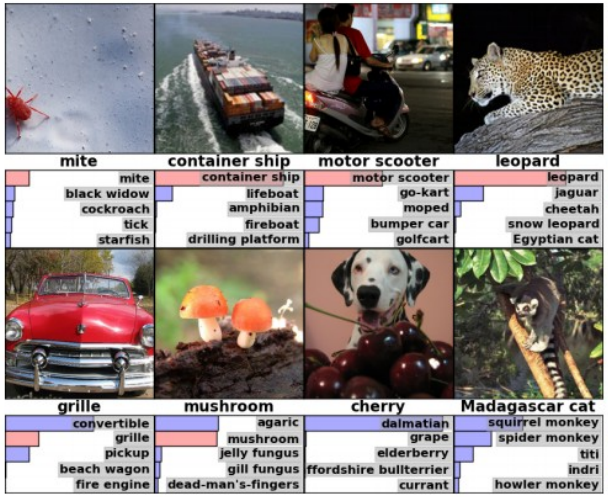


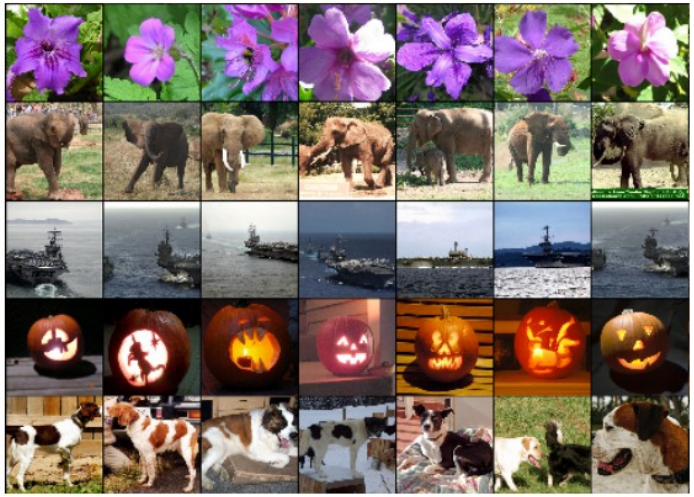
Fast-forward to today: ConvNets are everywhere

[Goodfellow 2014]

Classification

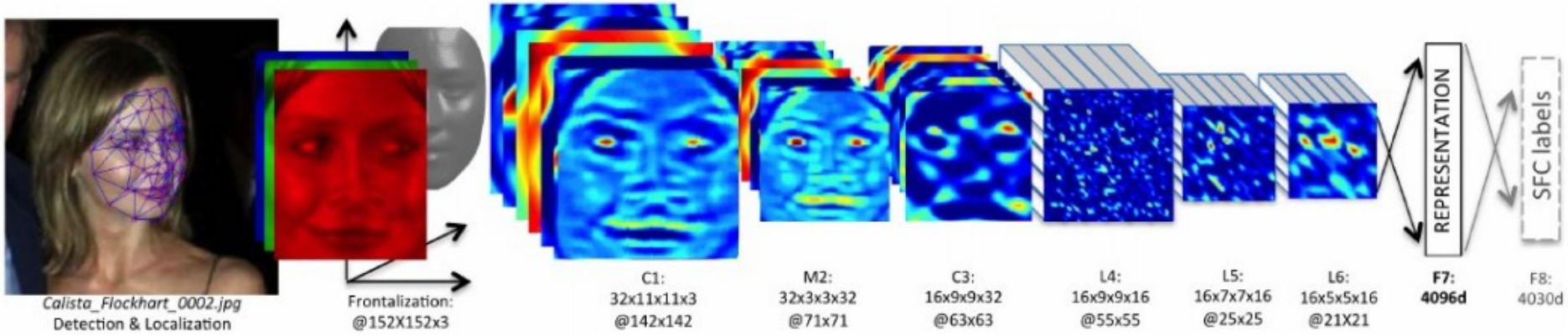
Retrieval



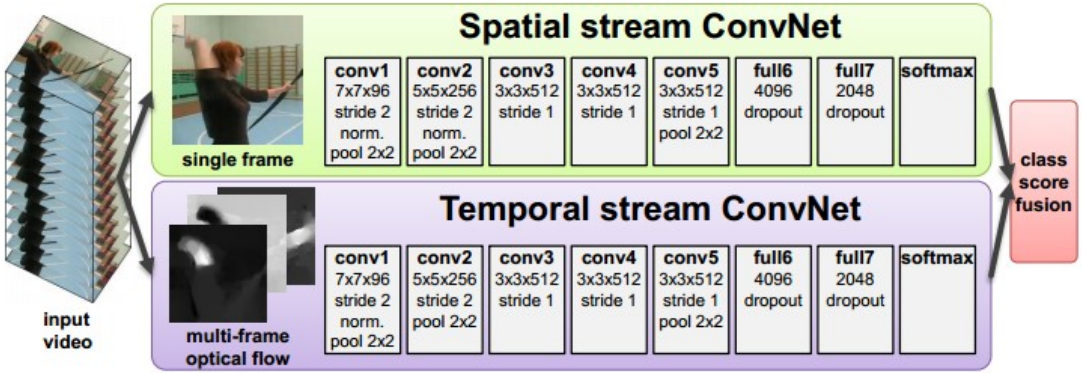


[Krizhevsky 2012]

Fast-forward to today: ConvNets are everywhere



[Taigman et al. 2014]



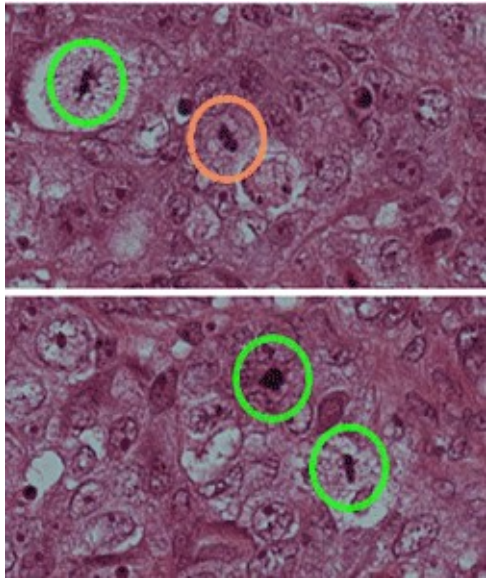
[Simonyan et al. 2014]

Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]

[Mnih 2013]

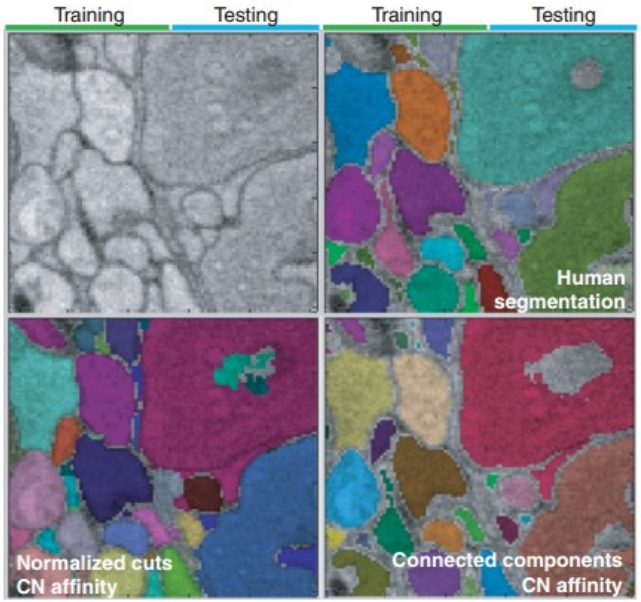
Fast-forward to today: ConvNets are everywhere



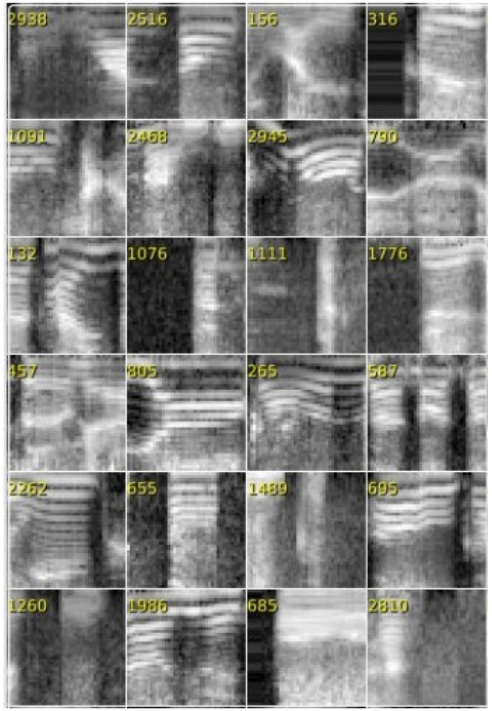
[Ciresan et al. 2013]

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

Fast-forward to today: ConvNets are everywhere



[Turaga et al., 2010]



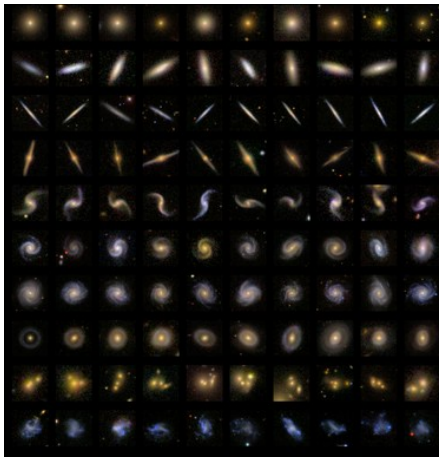
I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. **Two guys (one naive and one loud mouthed a **) take a road trip to stop a wedding but have the worst possible luck when a maniac in a freaky, make-shift tank/truck 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.

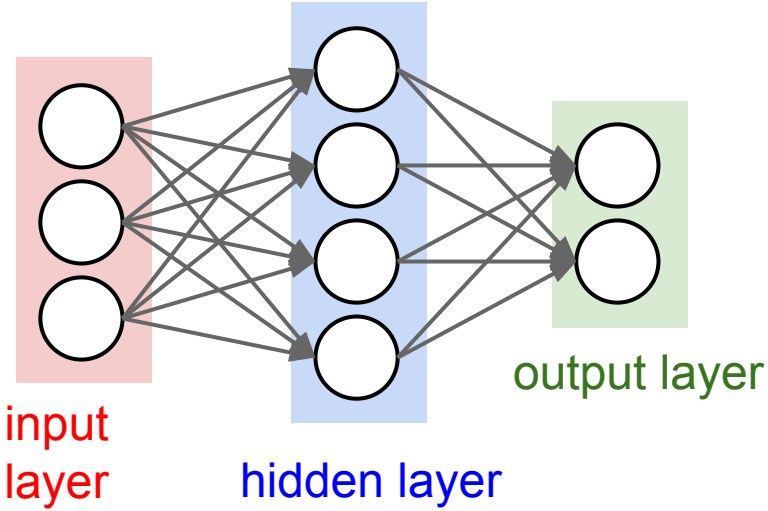
Graphics is far from the best part of the game. **This is the number one best TH game in the series.** Next to Underground. **It deserves strong love. It is an insane game.** There are massive levels, massive unlockable characters... it's just a massive game. **Waste your money on this game. This is 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.

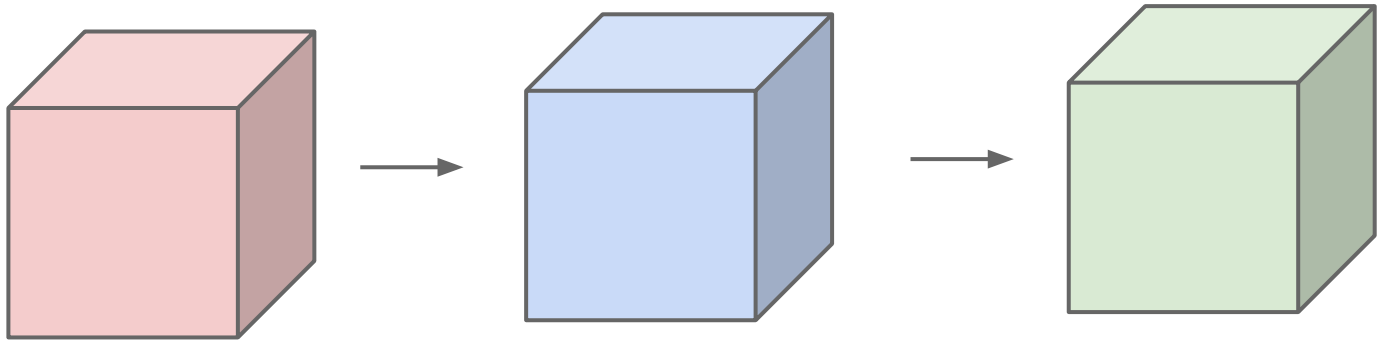
[Denil et al. 2014]



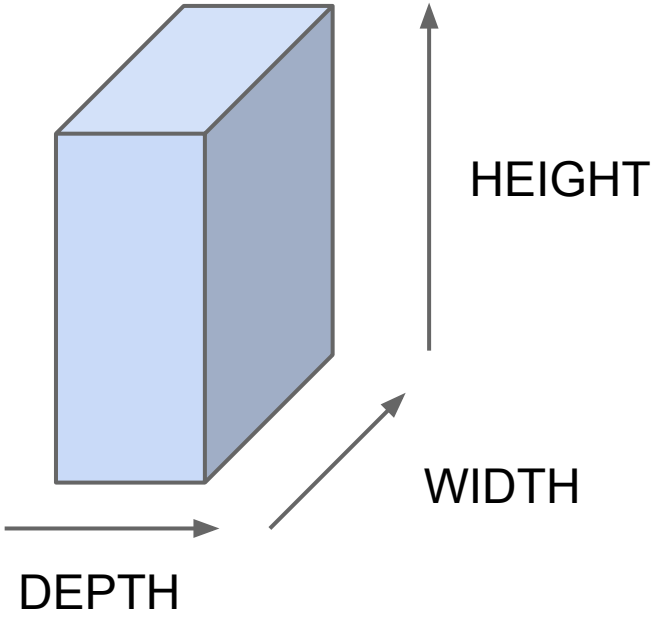
before:



now:



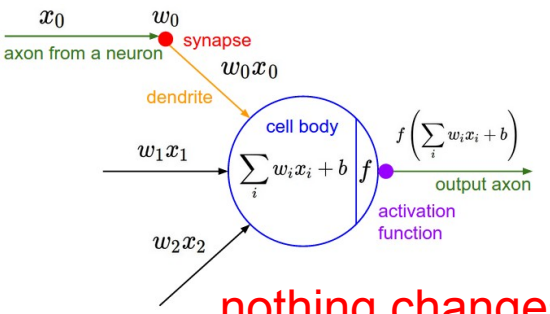
All Neural Net activations arranged in **3 dimensions:**



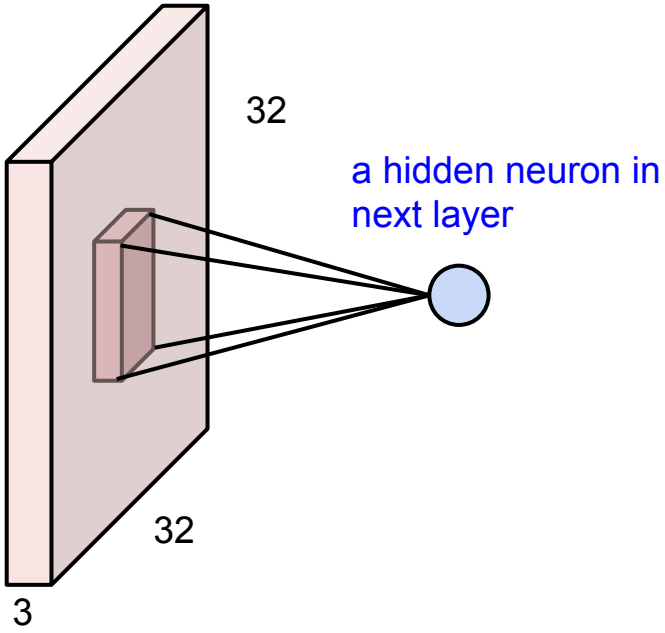
For example, a CIFAR-10 image is a 32x32x3 volume
32 width, 32 height, 3 depth (RGB channels)

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**

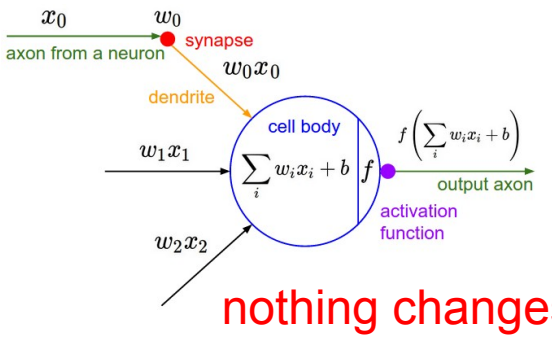


nothing changes



Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity



nothing changes

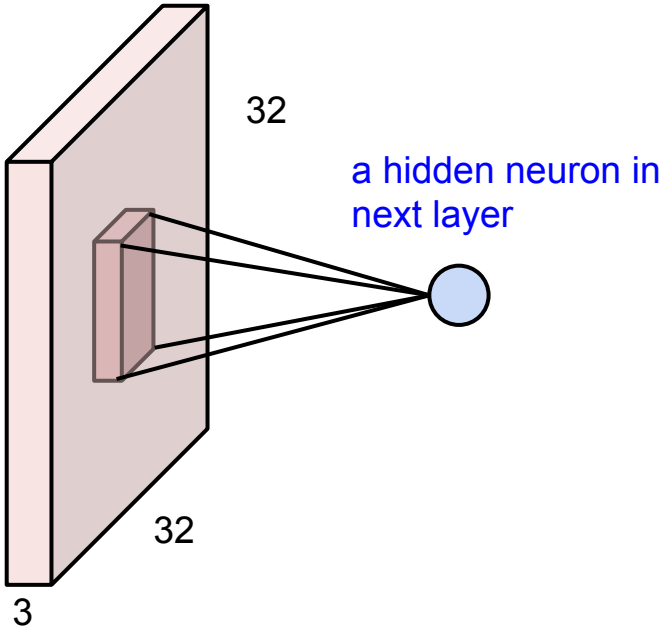


image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

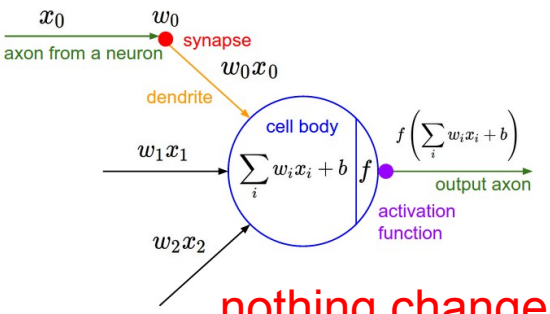
now: one neuron will connect to, e.g. 5x5x3 chunk and only have 5x5x3 weights.

note that connectivity is:

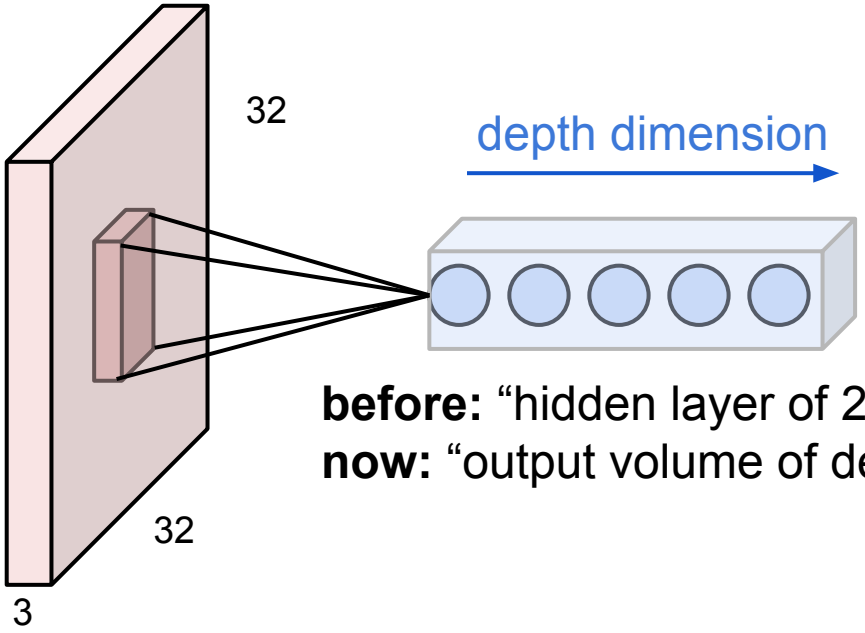
- <u>local in space</u> (5x5 inside 32x32)
- <u>but full in depth</u> (all 3 depth channels)

Convolutional Neural Networks are just Neural Networks BUT:

1. **Local connectivity**



nothing changes

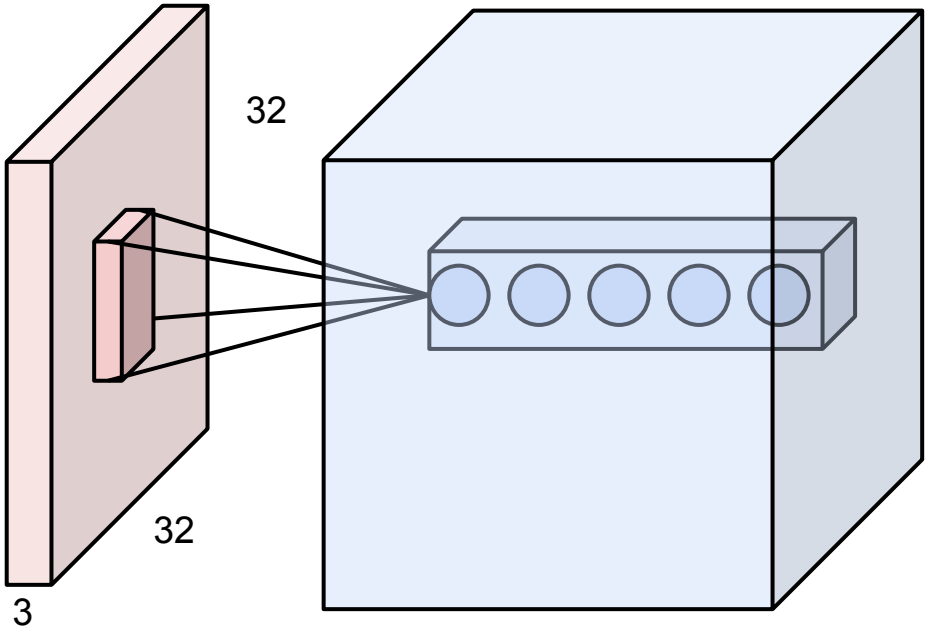


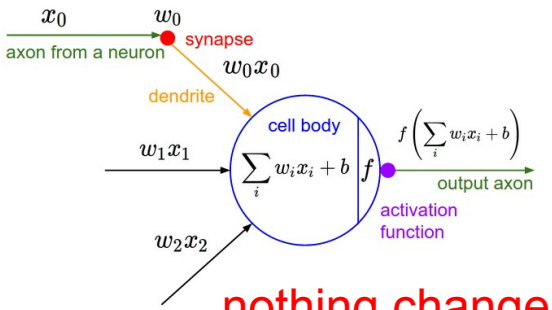
before: “hidden layer of 200 neurons”
now: “output volume of depth 200”

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**

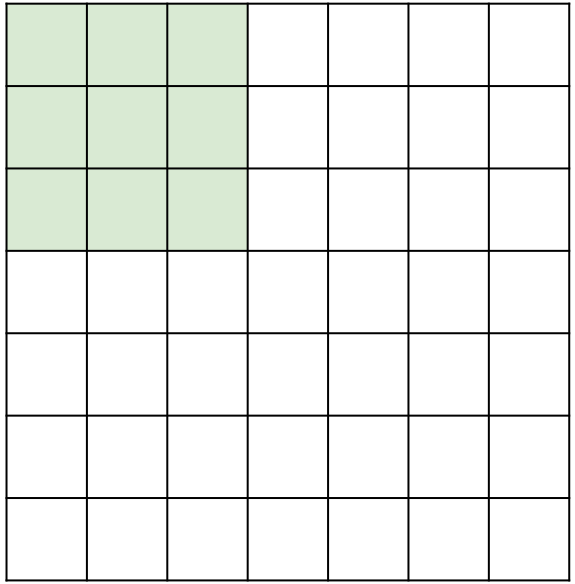




nothing changes

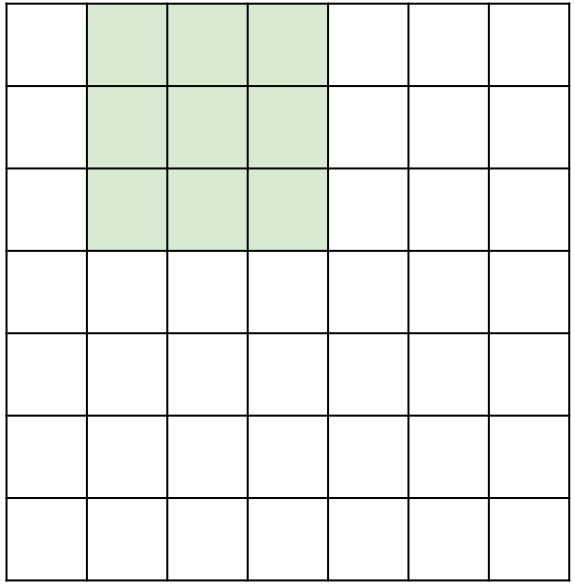
These form a single [1 x 1 x depth] "depth column" in the output volume

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



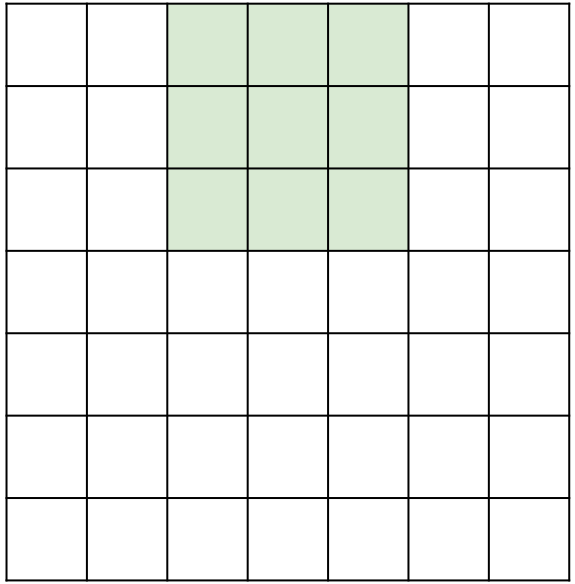
7x7 input
assume 3x3 connectivity, stride 1

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



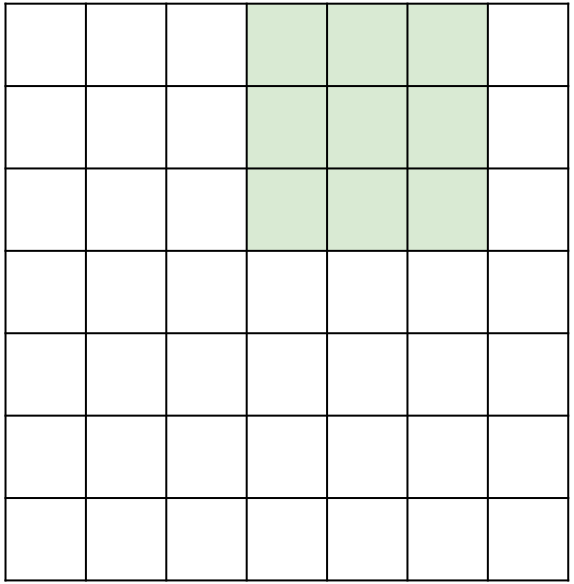
7x7 input
assume 3x3 connectivity, stride 1

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



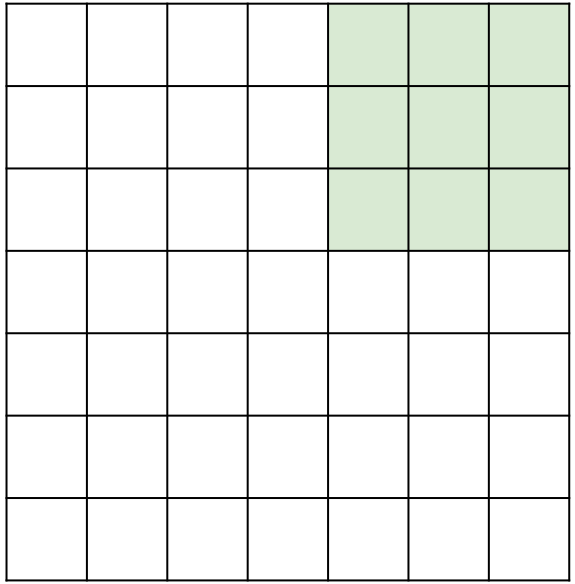
7x7 input
assume 3x3 connectivity, stride 1

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



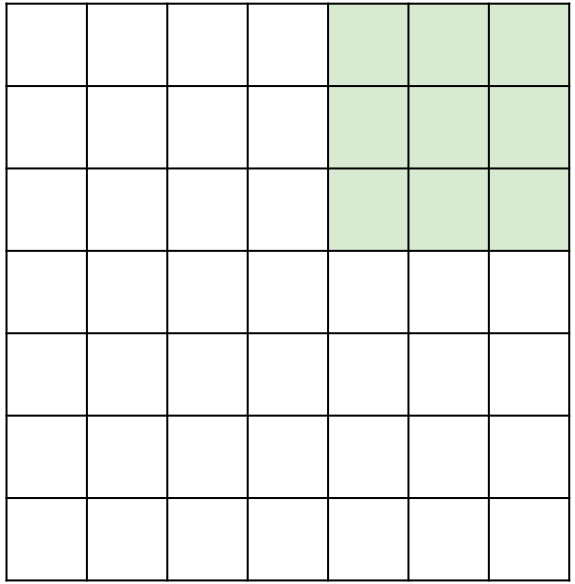
7x7 input
assume 3x3 connectivity, stride 1

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



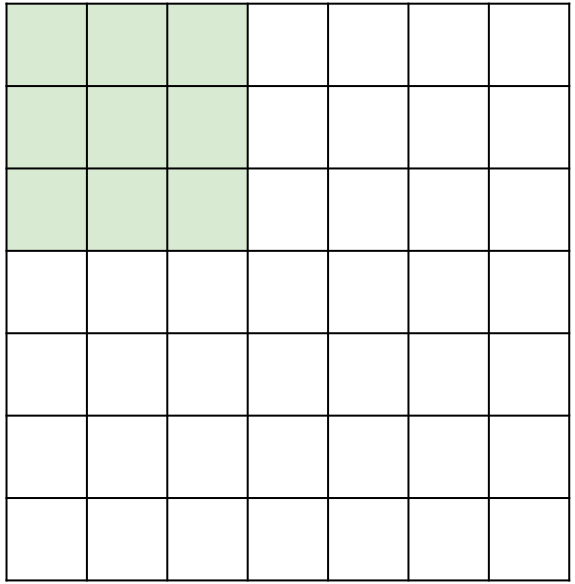
7x7 input
assume 3x3 connectivity, stride 1

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



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

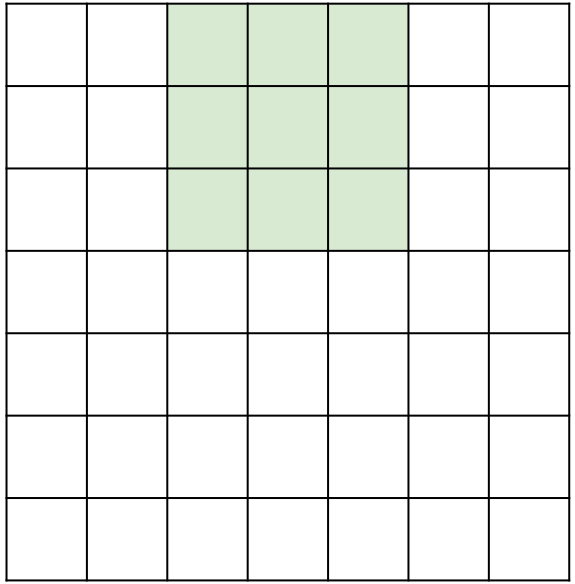
Replicate this column of hidden neurons across space, with some **stride.**



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

what about stride 2?

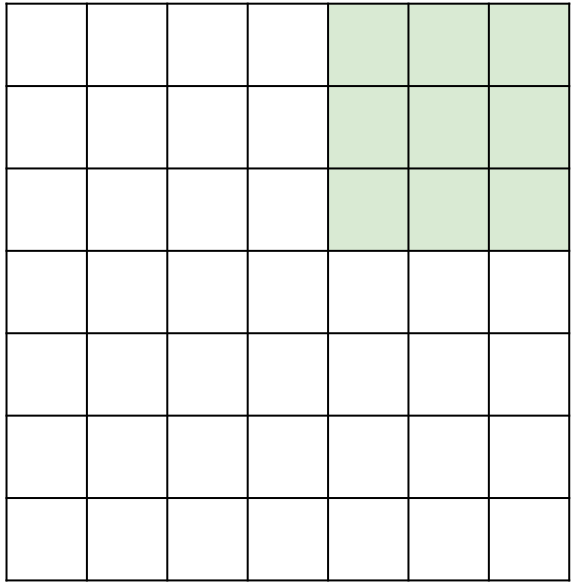
Replicate this column of hidden neurons across space, with some **stride.**



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

what about stride 2?

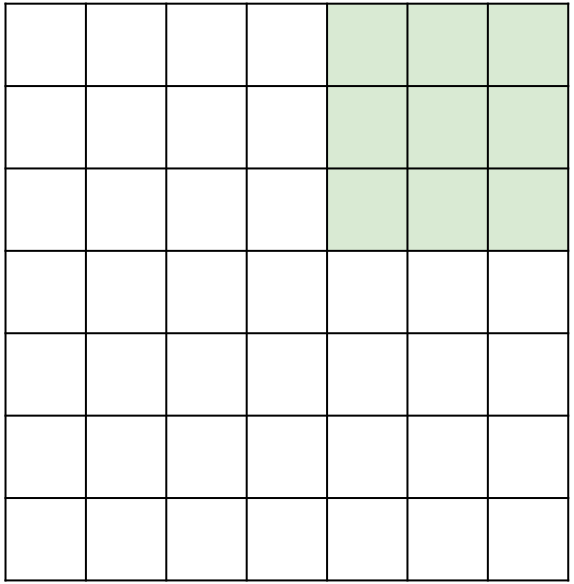
Replicate this column of hidden neurons across space, with some **stride.**



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

what about stride 2?

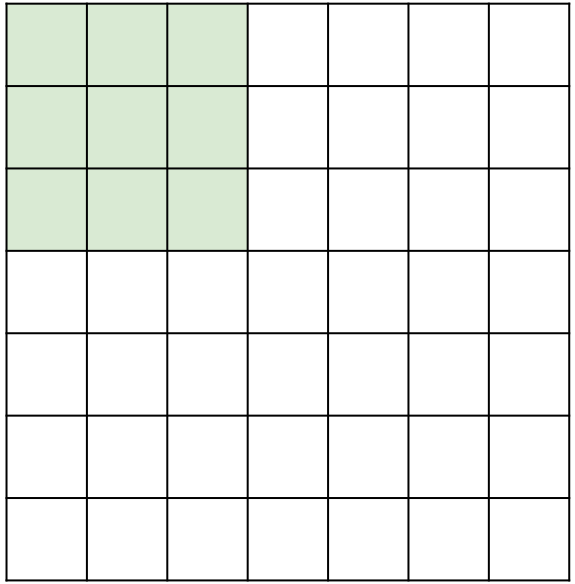
Replicate this column of hidden neurons across space, with some **stride.**



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.**

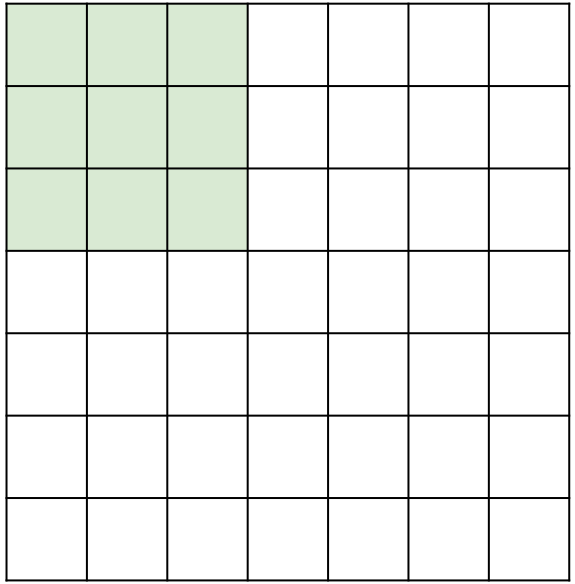


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

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

what about stride 3?

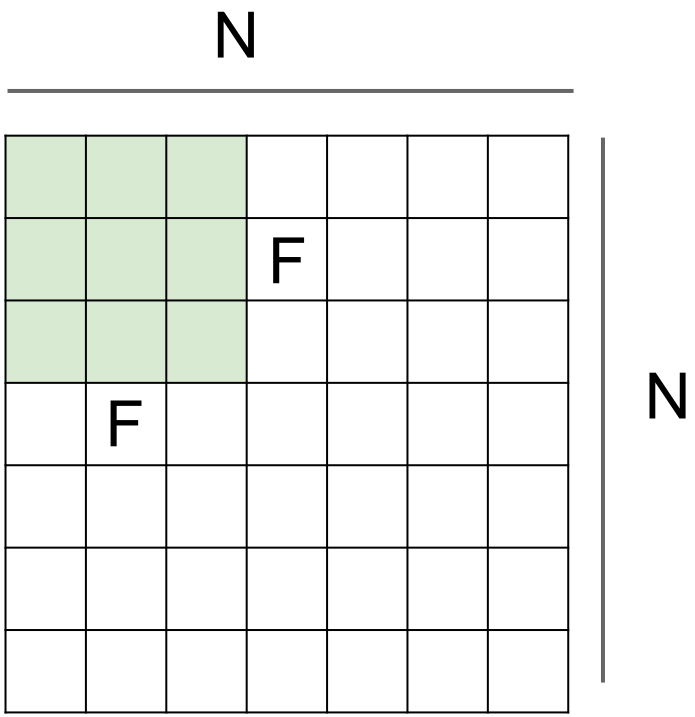
Replicate this column of hidden neurons across space, with some **stride.**



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

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

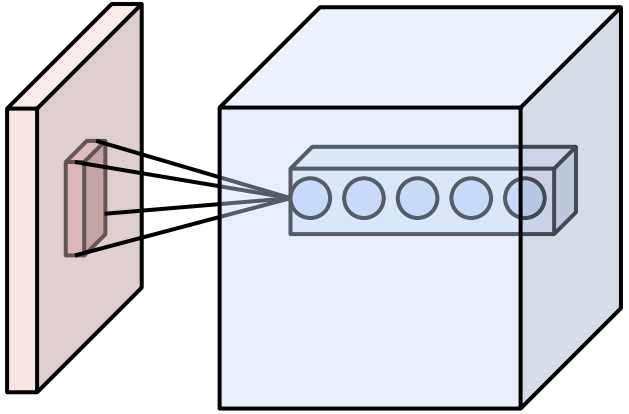
what about stride 3? Cannot.



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 = … :\

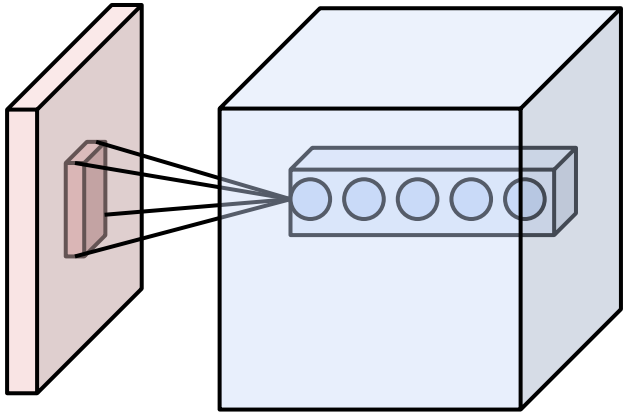
Examples time:



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

Output volume: ?

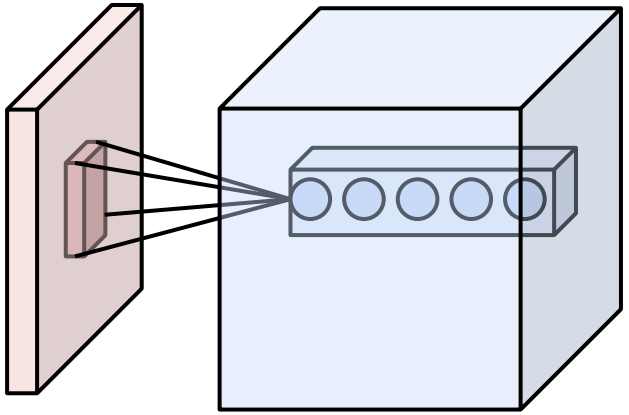
Examples time:



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?

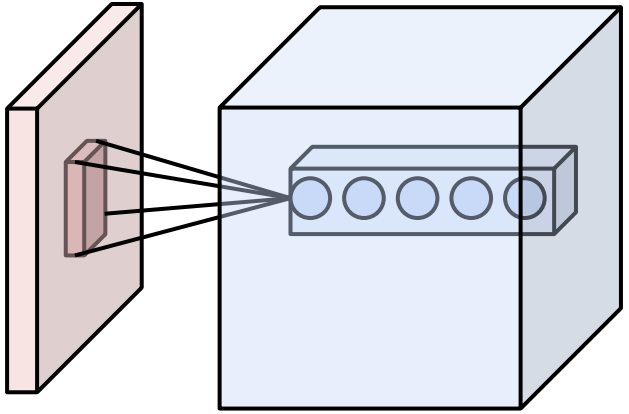
Examples time:



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**

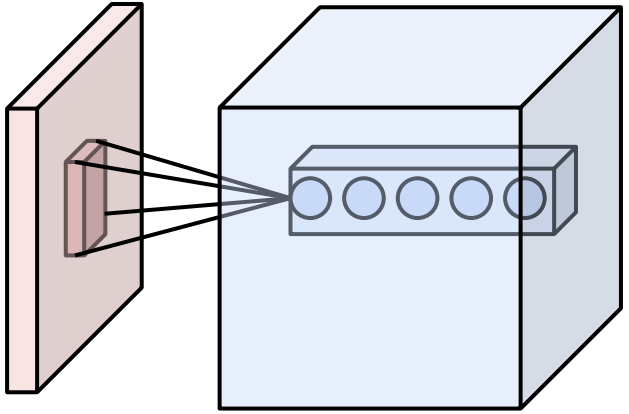
Examples time:



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

Output volume: ?

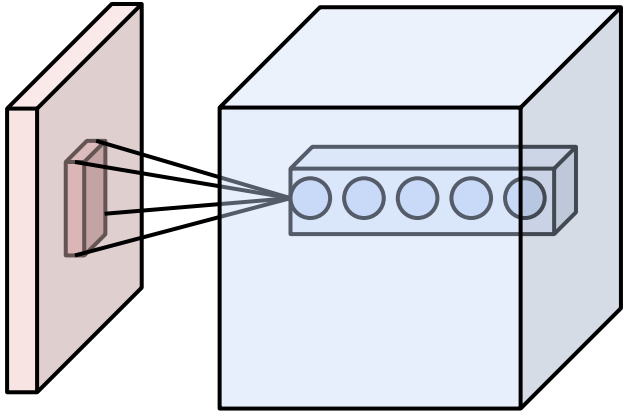
Examples time:



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

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

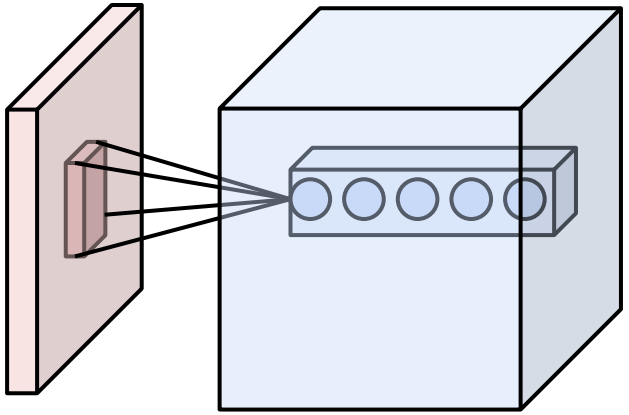
Examples time:



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

Output volume: ?

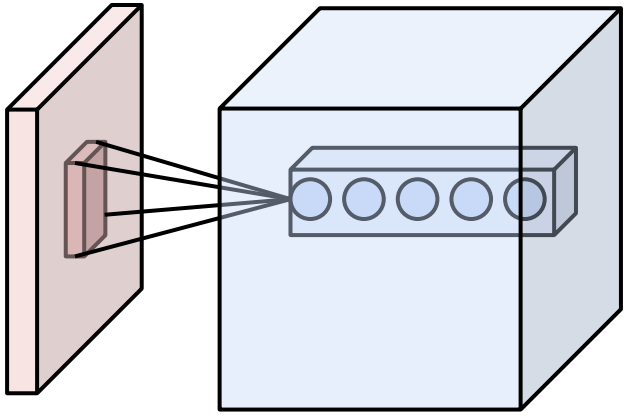
Examples time:



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)

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…

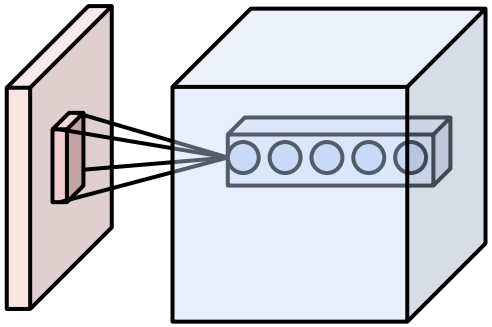
Assume input **[32 x 32 x3]**

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…

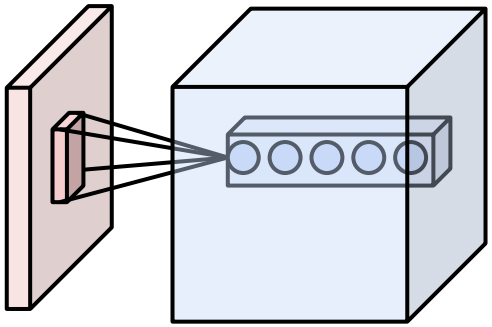
Assume input **[32 x 32 x3]**

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…

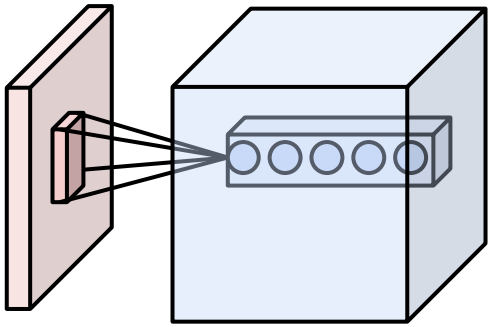
Assume input **[32 x 32 x3]**

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

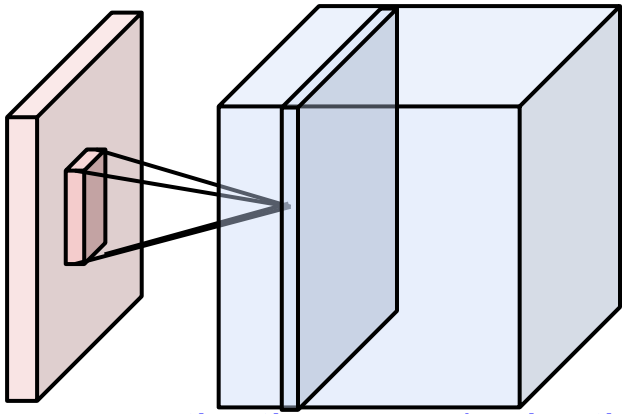
Our first ConvNet layer had size **[32 x 32 x3]**
If we had **30** neurons with receptive fields **5x5**, **stride 1, pad 1**
Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)
Each neuron has 5*5*3 (=75) weights

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

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

These layers are called **Convolutional Layers**

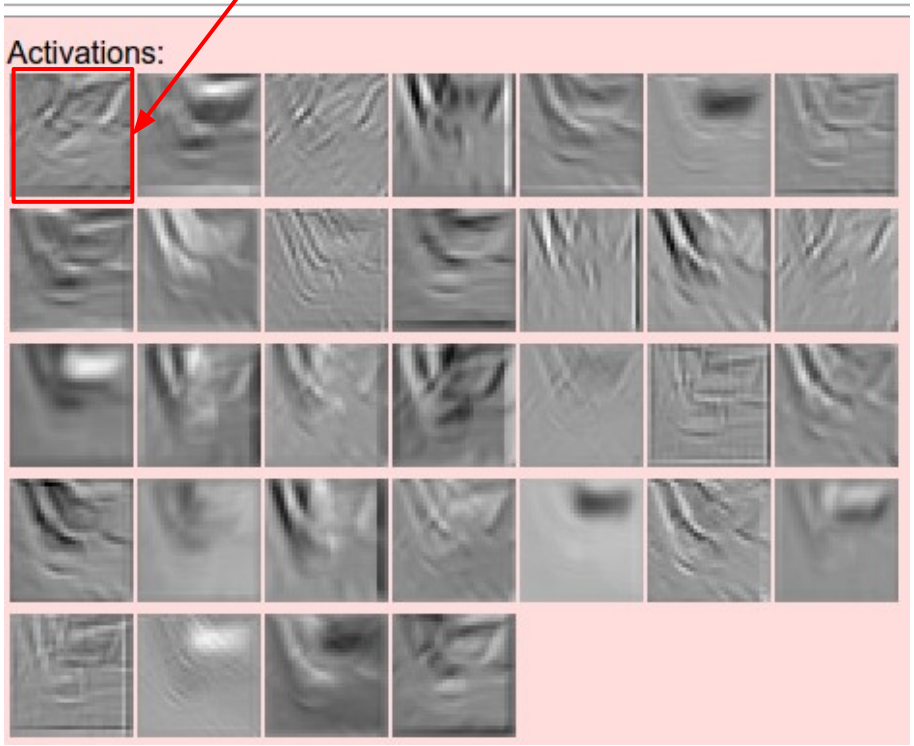
1. Connect neurons only to local receptive fields
2. Use the same neuron weight parameters for neurons in each “depth slice” (i.e. across spatial positions)



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”

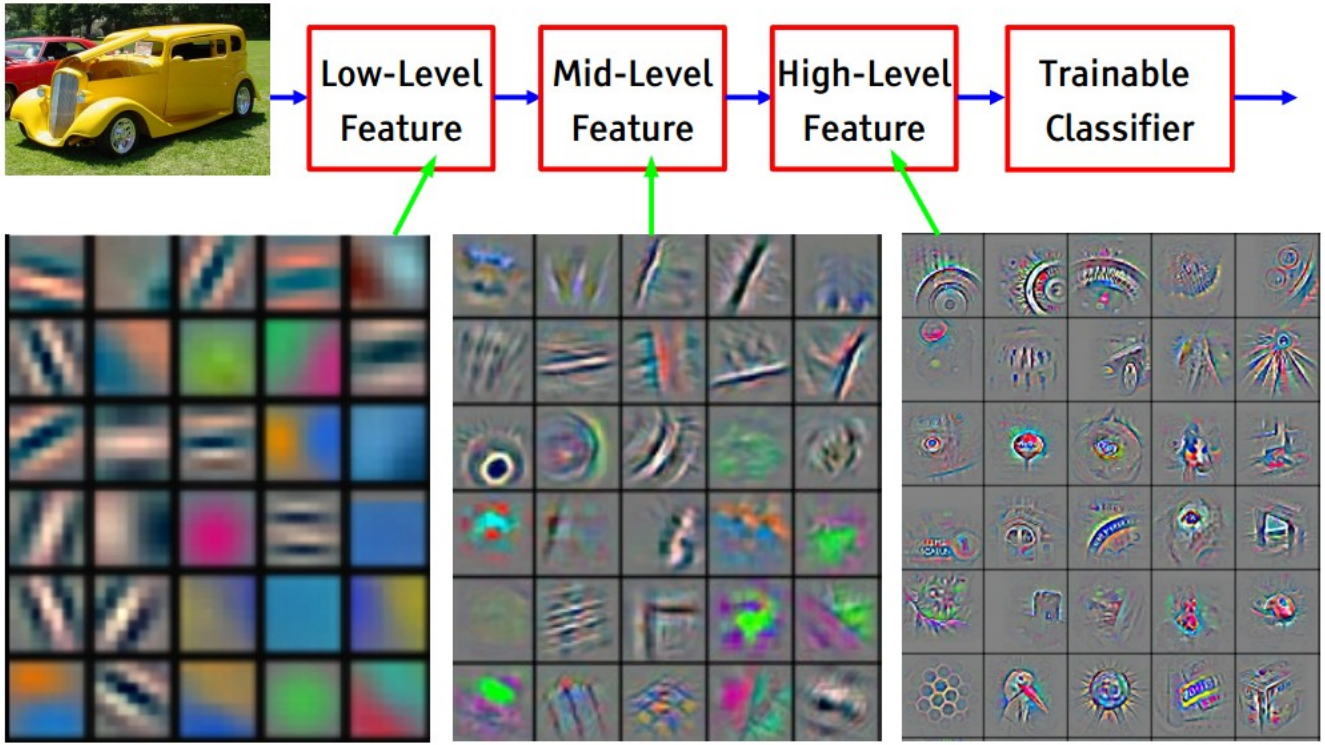
We call the layer convolutional because it is related to convolution of two signals (kind of):

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

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

Fast-forward to today

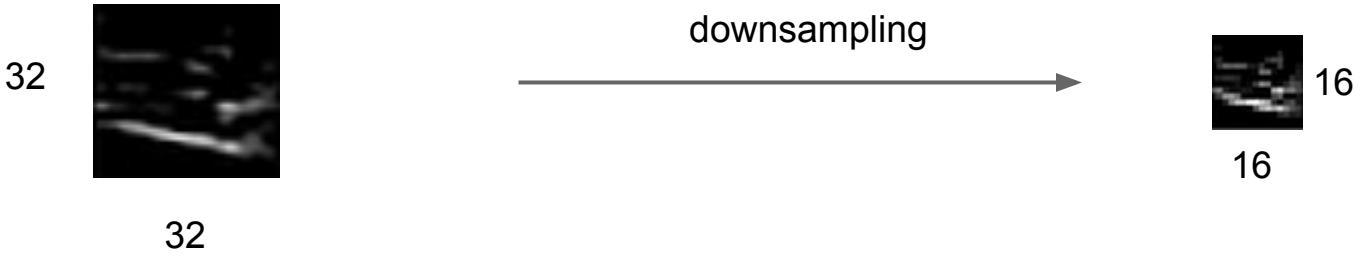
[From recent Yann LeCun slides]



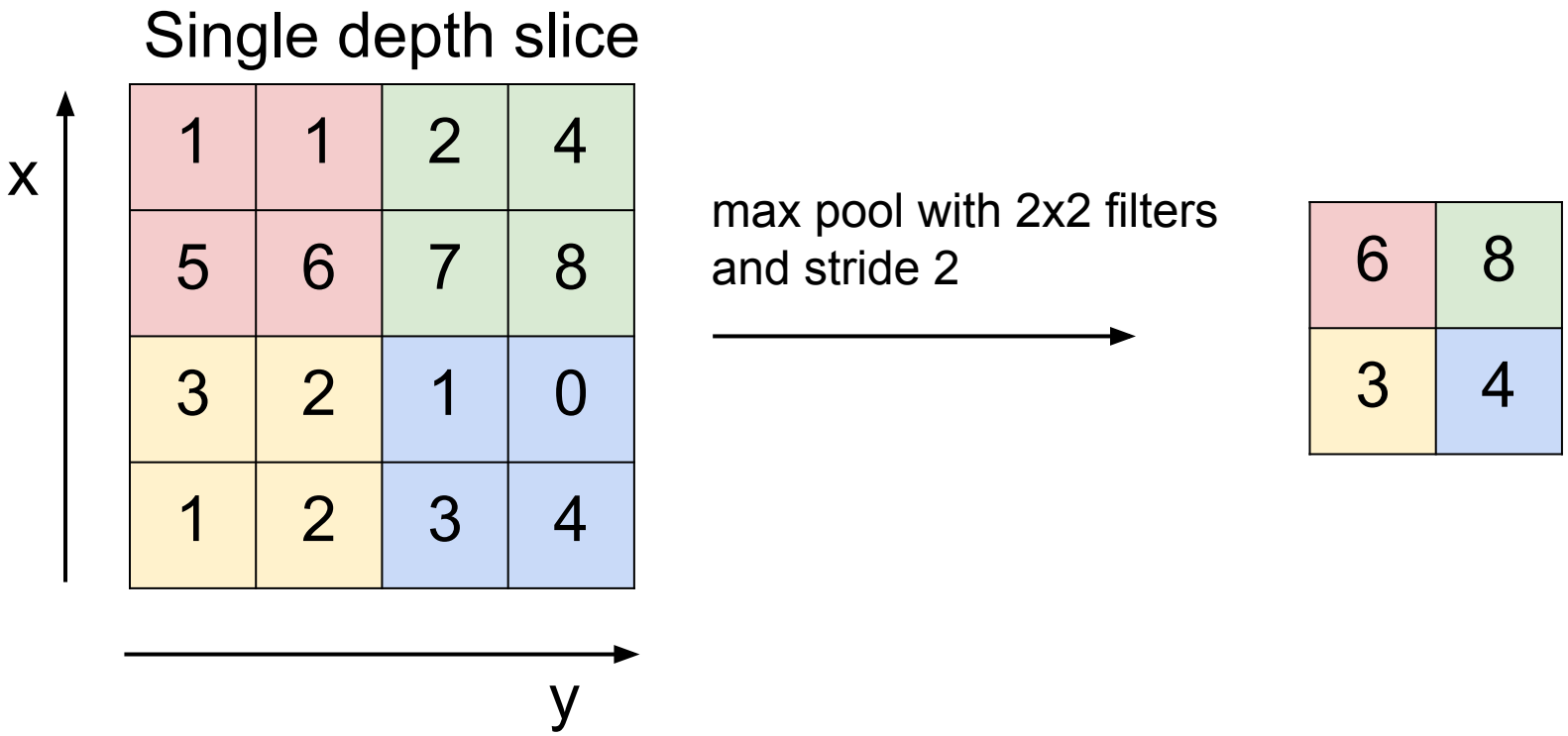
Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

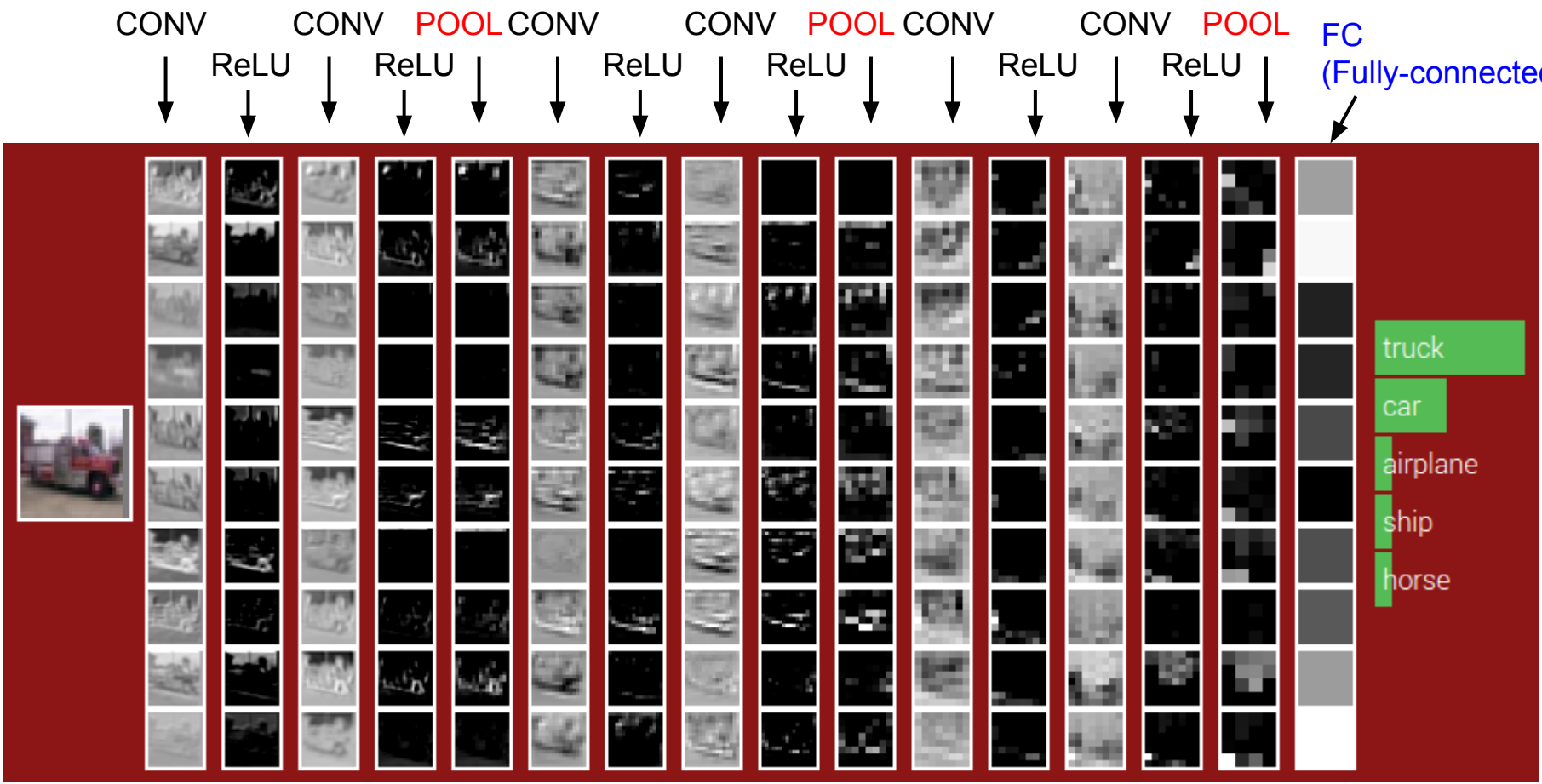
In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

- 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


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:

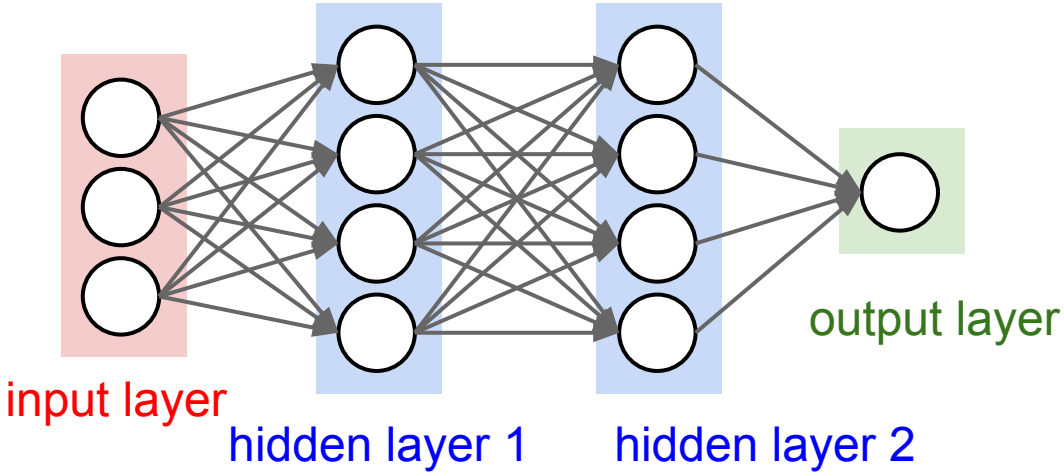
- use **filter sizes of 3x3** (maybe even 2x2 or 1x1!)
- use **pooling sizes of 2x2** (maybe even less - e.g. fractional pooling!)
- **stride 1**
- **very deep**

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

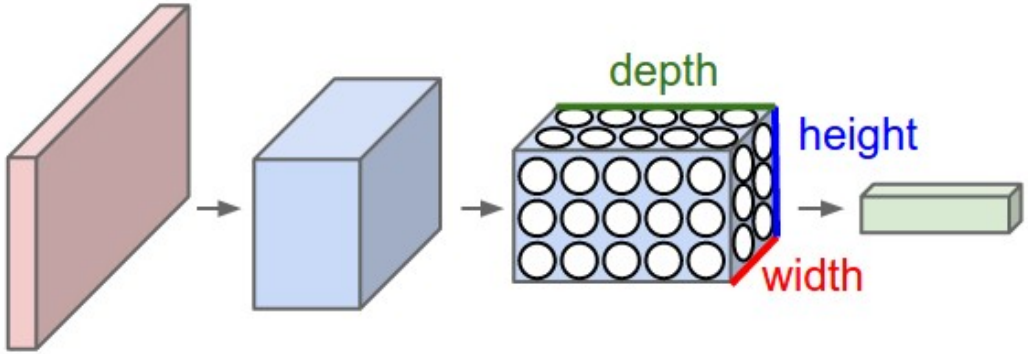
Summary:

- 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

before:


input layer
hidden layer 1
hidden layer 2
output layer

now:


depth
height
width

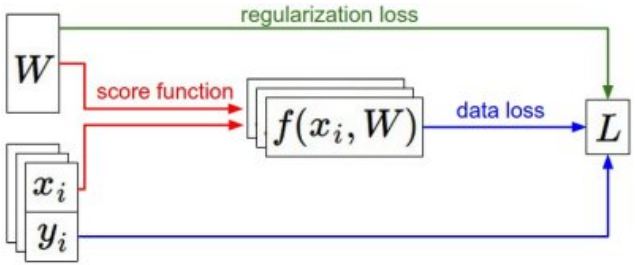
Q: What images maximize the score of some class in a ConvNet?

1. Find images that maximize some class score:

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2$$

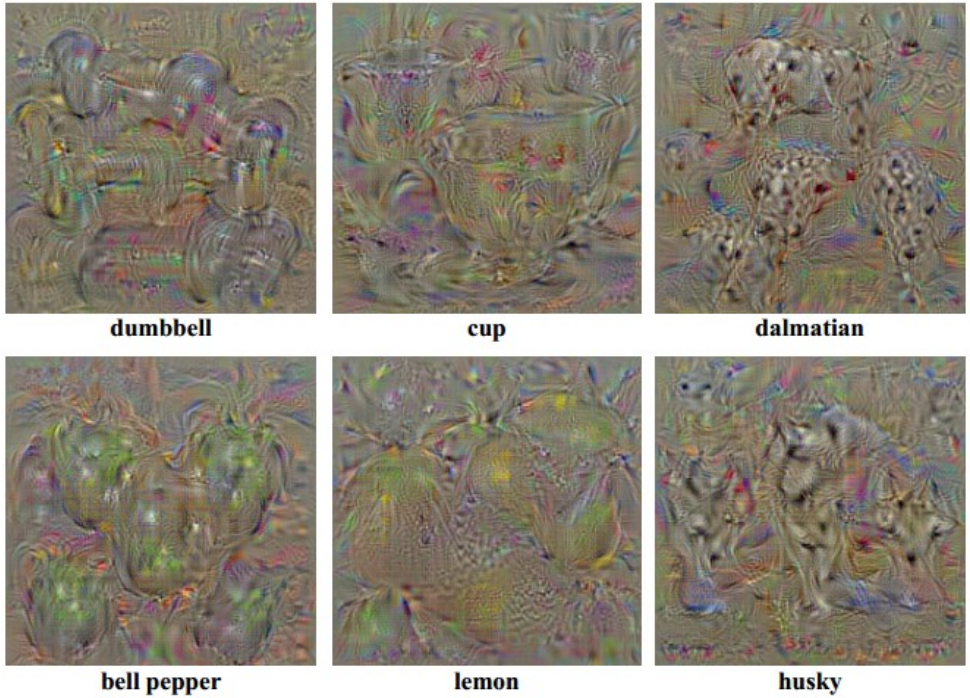
Score for class c
(before Softmax)

Remember:



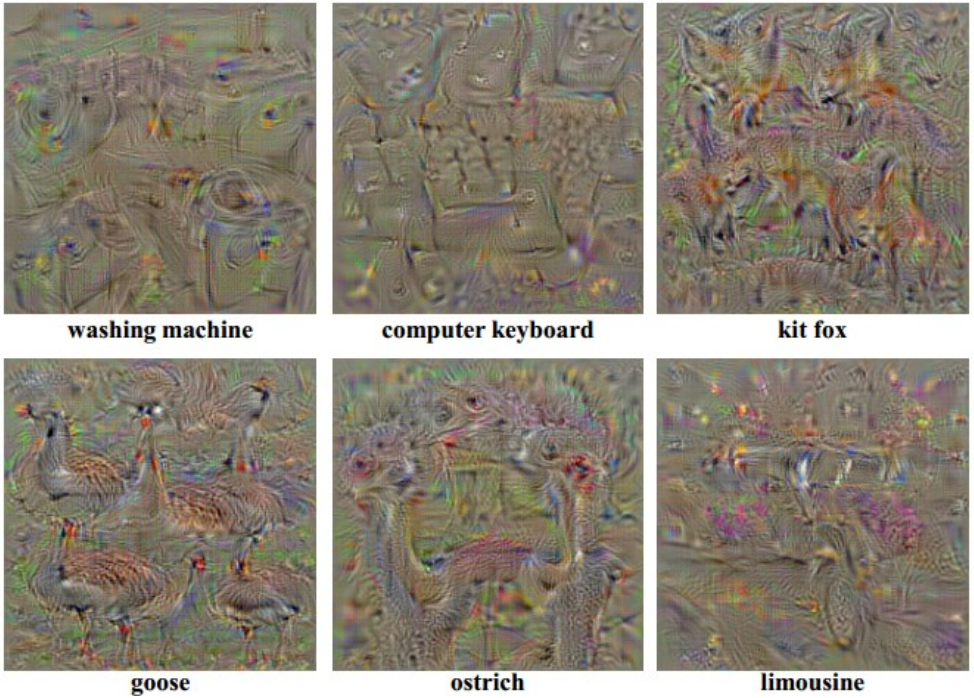
Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

1. Find images that maximize some class score:



Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps
Karen Simonyan, Andrea Vedaldi, Andrew Zisserman, 2014

1. Find images that maximize some class score:



	GoogLeNet correct	GoogLeNet wrong
Human correct	1352/1500 	72/1500 • Objects very small or thin • Abstract representations • Image filters
Human wrong	46/1500 • Fine-grained recognition • Class unawareness • Insufficient training data	30/1500 • Multiple objects • Incorrect annotations

GoogLeNet: 6.8%
Andrej: 5.1% phew...

rule, ruler	king crab, Alaska crab	sidewinder	saltshaker, salt shaker	reel	hatchet	schipperke
pencil box, pencil case	pizza, pizza pie	maze, labyrinth	pill bottle	stethoscope	vase	schipperke
rubber eraser, rubber	strawberry	gar, garfish	water bottle	whistle	pitcher, ewer	groenendael
ballpoint, ballpoint pen	orange	valley, vale	lotion	ice lolly, lolly	coffeepot	doormat, welcome mat
pencil sharpener	fig	hammerhead	hair spray	hair spray	mask	teddy, teddy bear
carpenter's kit, tool kit	ice cream, icecream	sea snake	beer bottle	maypole	cup	jigsaw puzzle

Slide References

- Stanford CS 231n
- Univ of Notre Dame: CSE 60647, Spring 2014
- Bayesian Behavior Lab, UNW
- Princeton, COS598 Spring 2015: The Unreasonable Effectiveness of Big Visual Data
- Slides from Deep Learning tutorials
- Slides from DeepMind