EECS 445: Intro to Machine Learning

11/30/2016, Deep Learning (Part 2)

Training Neural Networks

Mini-batch Gradient descent

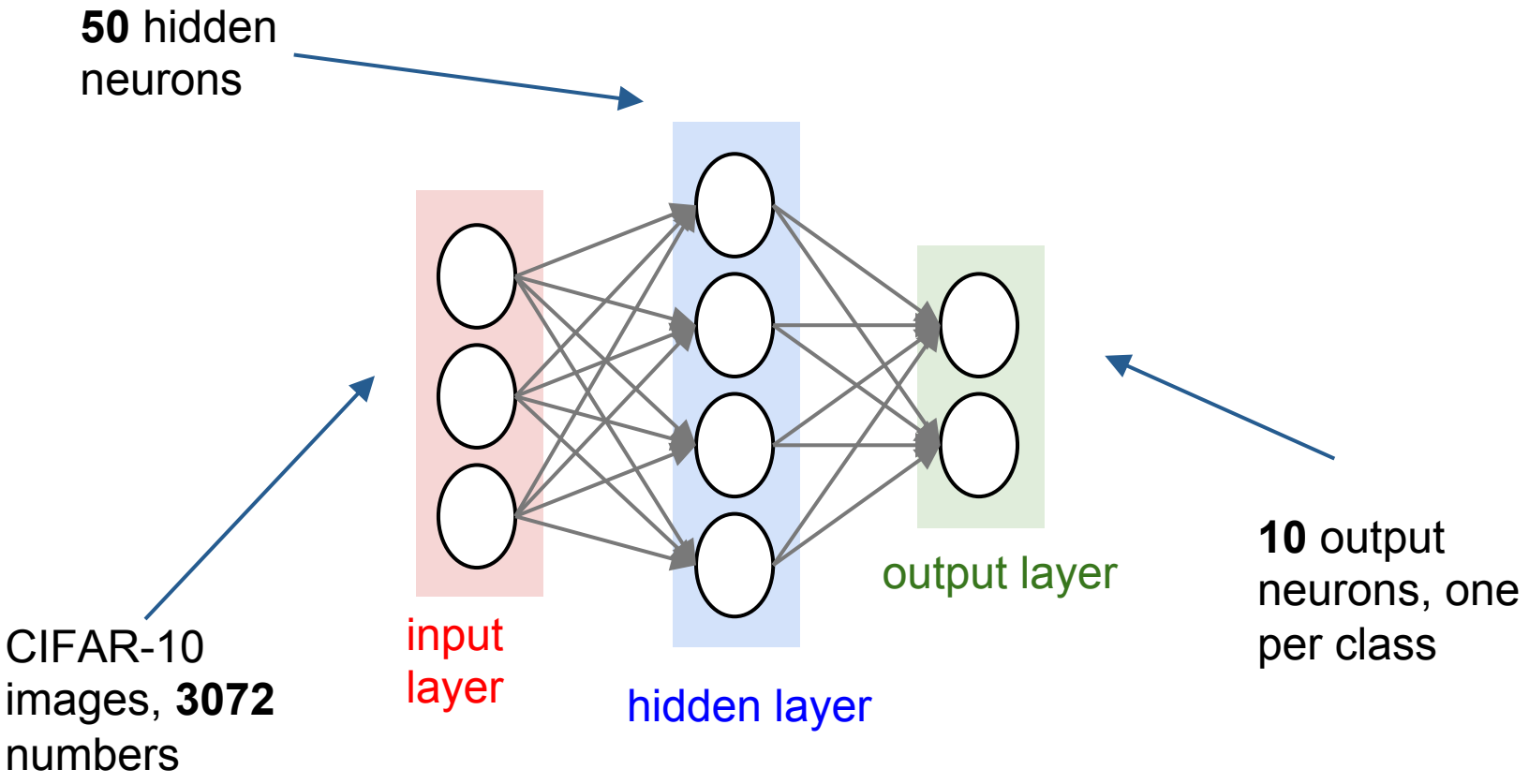
Loop:

1. Sample a batch of data
2. Backprop to calculate the analytic gradient
3. Perform a parameter update

Source: Andrej Karpathy & Fei-Fei Li

Choose the architecture:

say we start with one hidden layer of 50 neurons:



Source: Andrej Karpathy & Fei-Fei Li

Use cross-validation to choose hyper-parameters:
e.g. learning rate, initialization, strength of regularization, …

Source: Andrej Karpathy & Fei-Fei Li

Learning rates and updates:

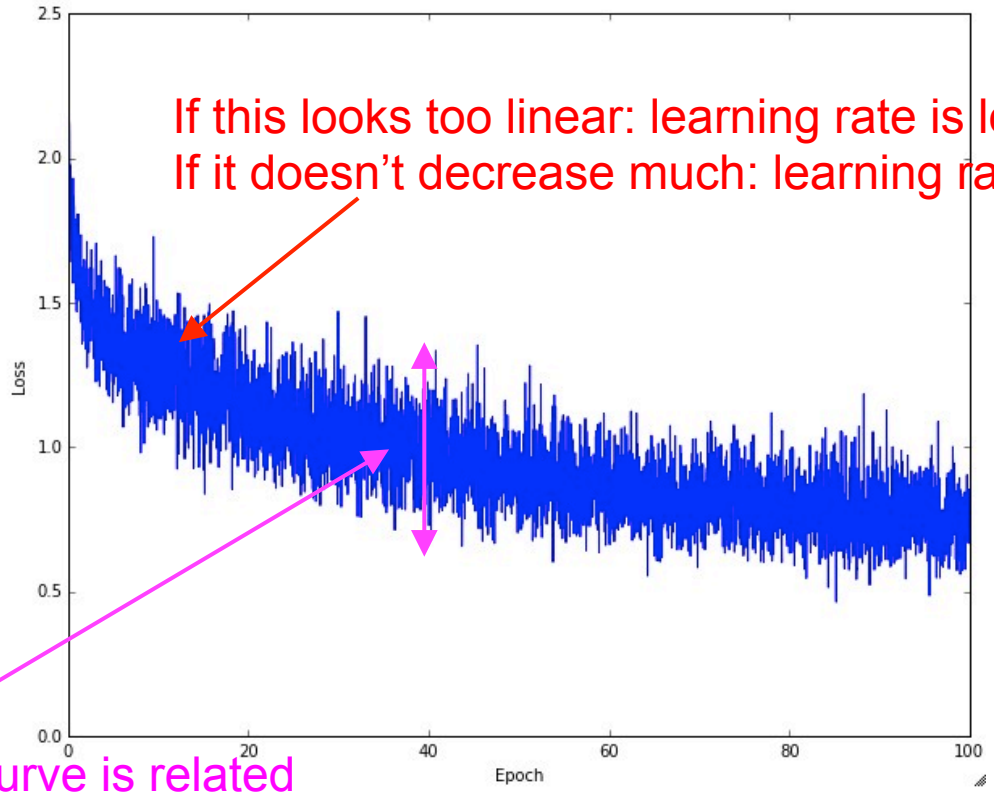
- SGD + Momentum > SGD
- Momentum 0.9 usually works well
- Decrease the learning rate over time

  (people use 1/t, exp(-t), or steps)

  simplest: learning_rate *= 0.97 every epoch (or so)

Source: Andrej Karpathy & Fei-Fei Li

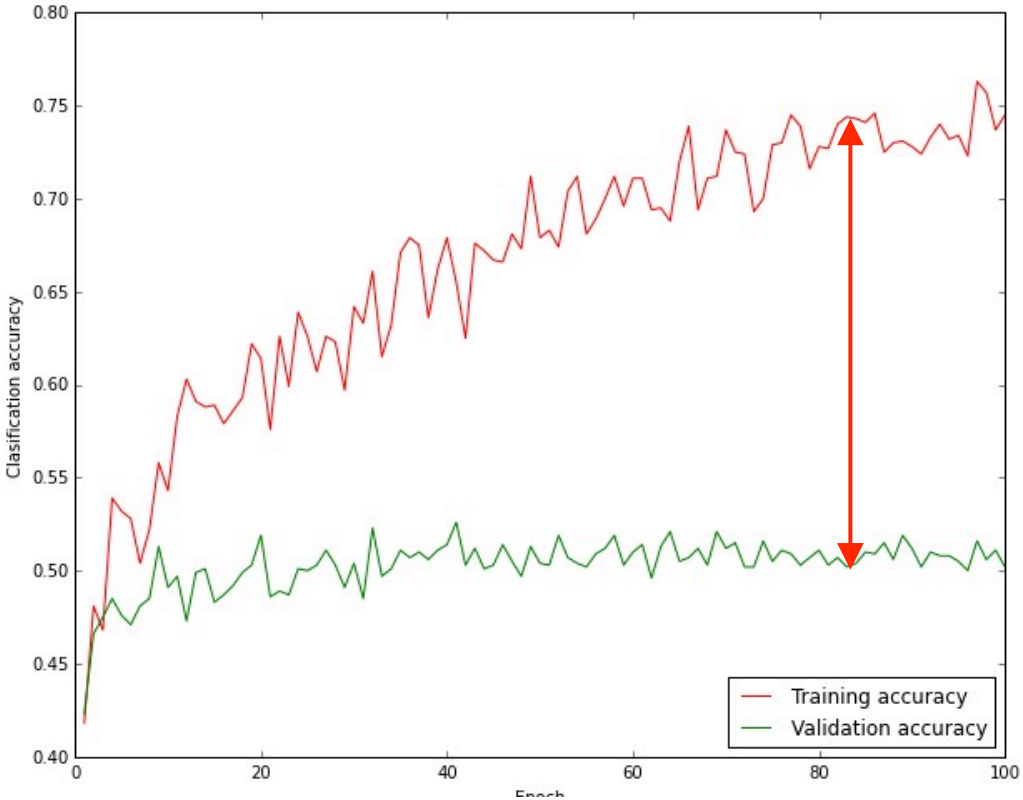
Monitor and visualize the loss curve



the "width" of the curve is related
to the batch size. This one looks too wide (noisy)
=> might want to increase batch size

Source: Andrej Karpathy & Fei-Fei Li

Monitor and visualize the accuracy:



big gap = overfitting

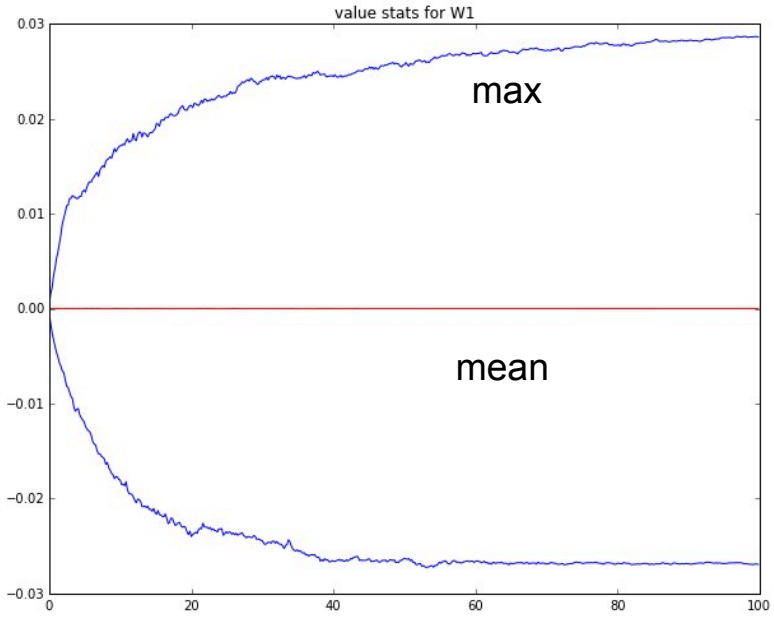
=> increase regularization strength

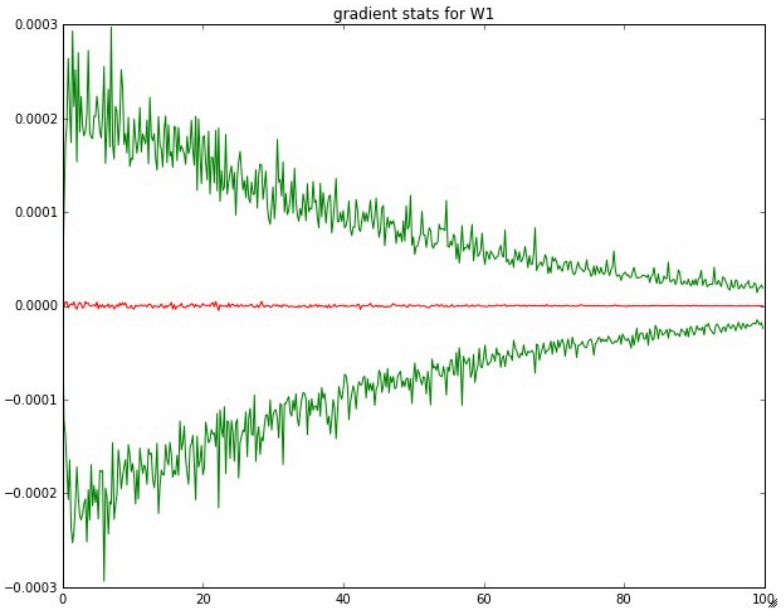
no gap

=> increase model capacity

Source: Andrej Karpathy & Fei-Fei Li

Track the ratio of weight updates / weight magnitudes:

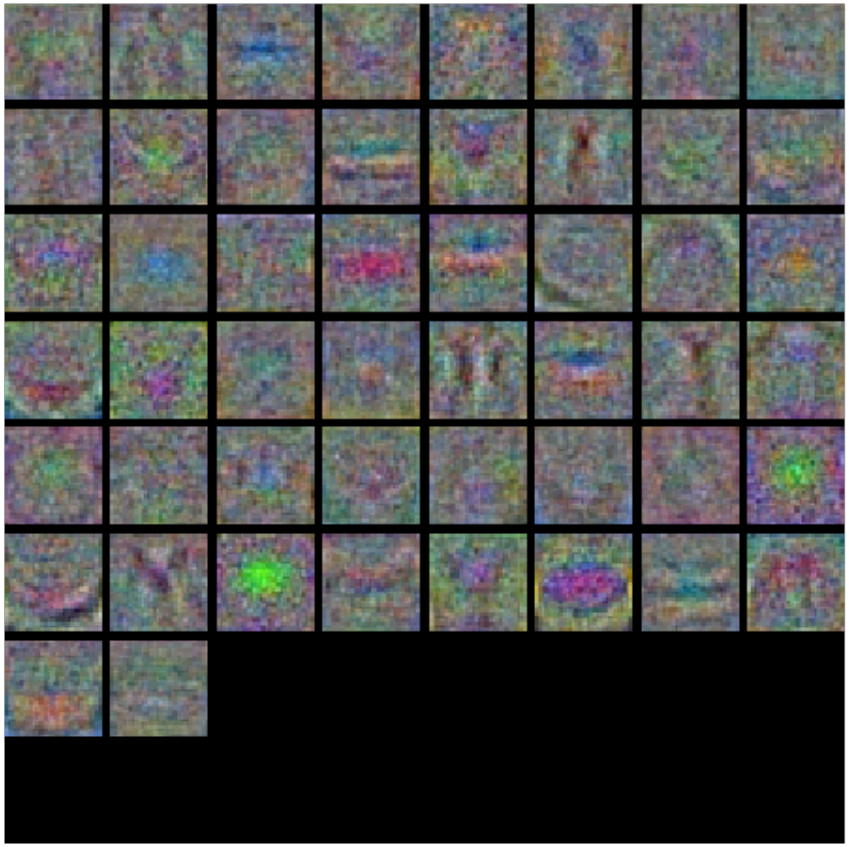




ratio between the values and updates: ~ 0.0002 / 0.02 = 0.01 (about okay)
want this to be somewhere around 0.01 - 0.001 or so

Source: Andrej Karpathy & Fei-Fei Li

Visualizing first-layer weights:

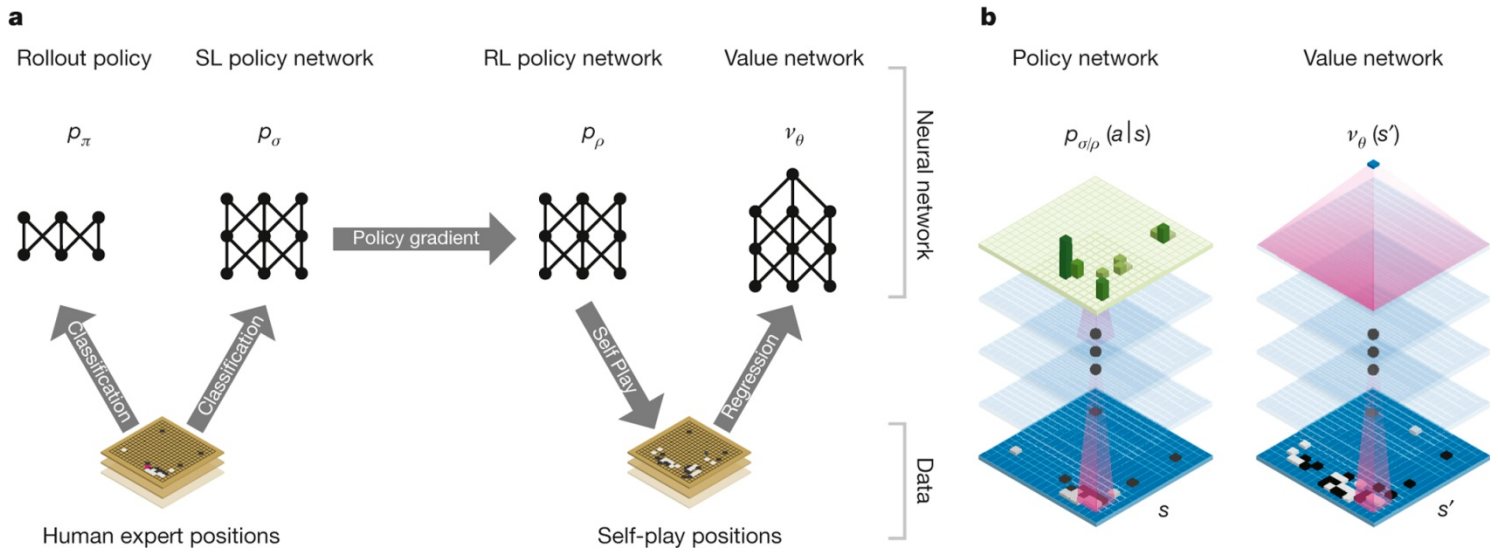


Noisy weights => Regularization maybe not strong enough

Source: Andrej Karpathy & Fei-Fei Li

AI machine takes 2-0 lead against South Korea's Lee Sedol, putting its owners one victory away from $1m prize

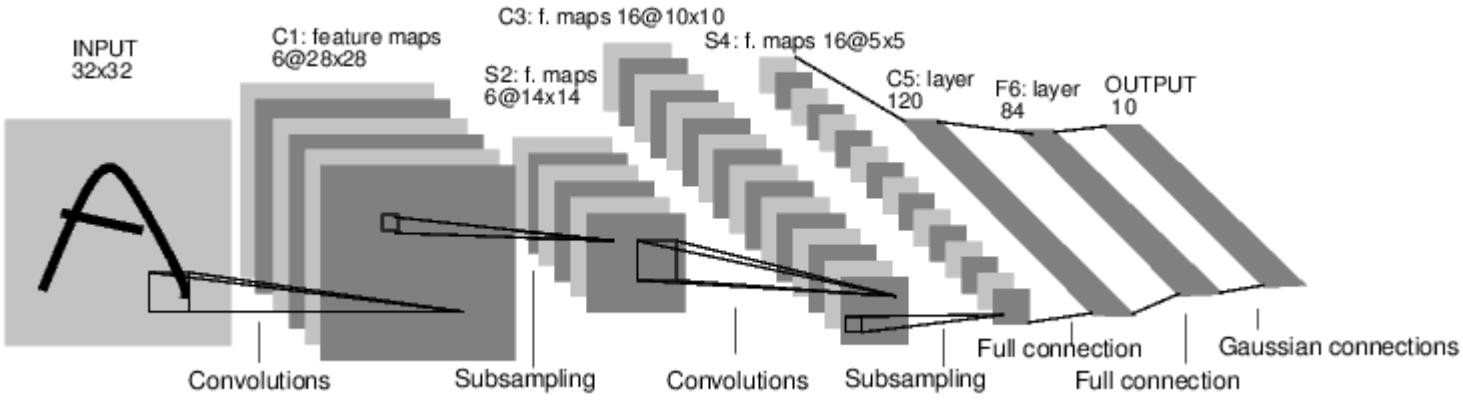
Neural network training pipeline and architecture



D Silver *et al. Nature* **529,** 484–489 (2016) doi:10.1038/nature16961

nature

Convolutional Neural Networks



[LeNet-5, LeCun 1980]

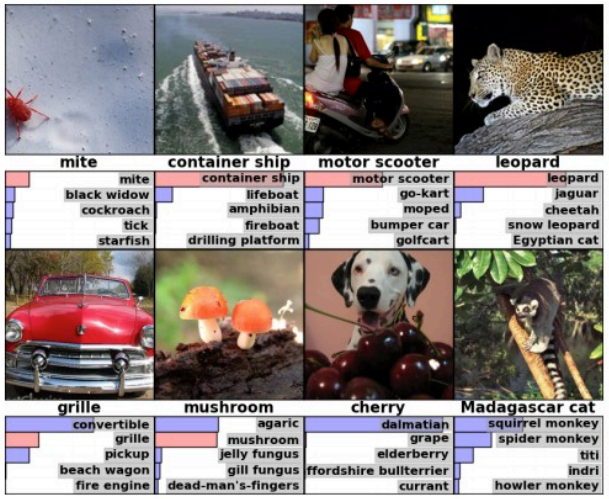
Source: Andrej Karpathy & Fei-Fei Li

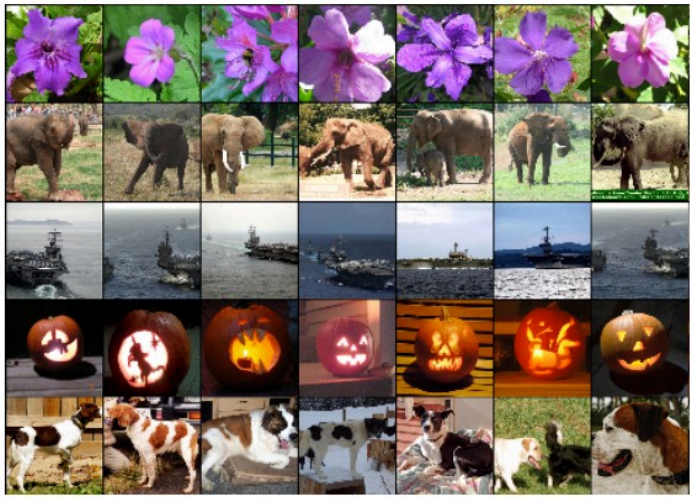
ConvNets are everywhere

[Goodfellow 2014]

Classification

Retrieval

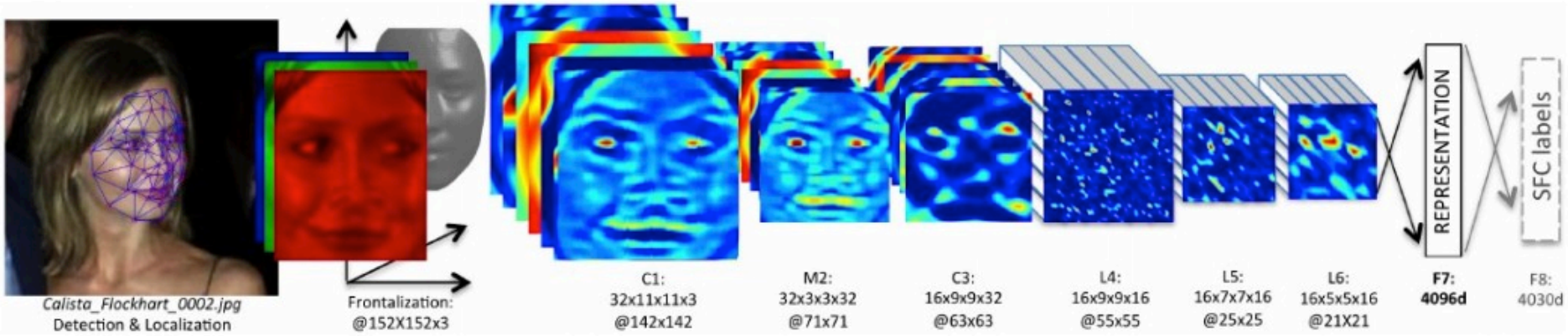




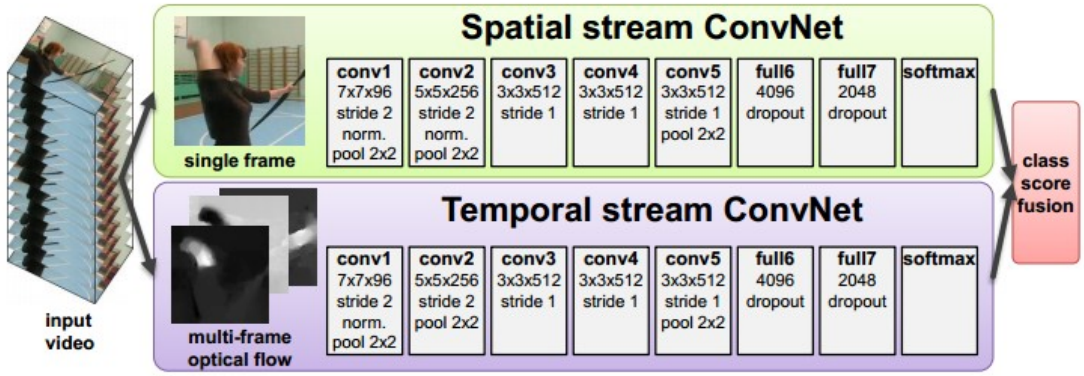
[Krizhevsky 2012]

Source: Andrej Karpathy & Fei-Fei Li

ConvNets are everywhere



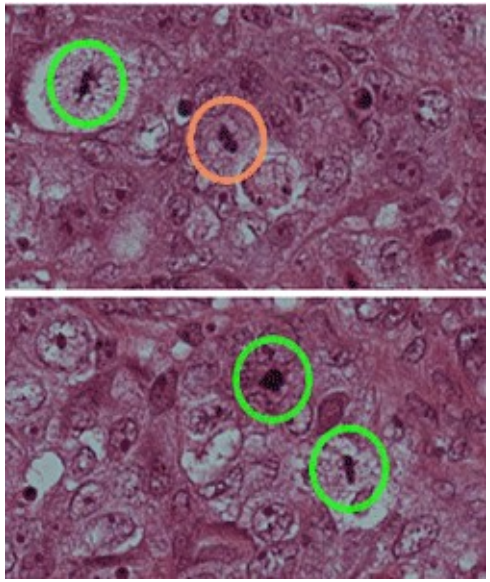
[Taigman et al. 2014]



[Simonyan et al. 2014]

Source: Andrej Karpathy & Fei-Fei Li

ConvNets are everywhere



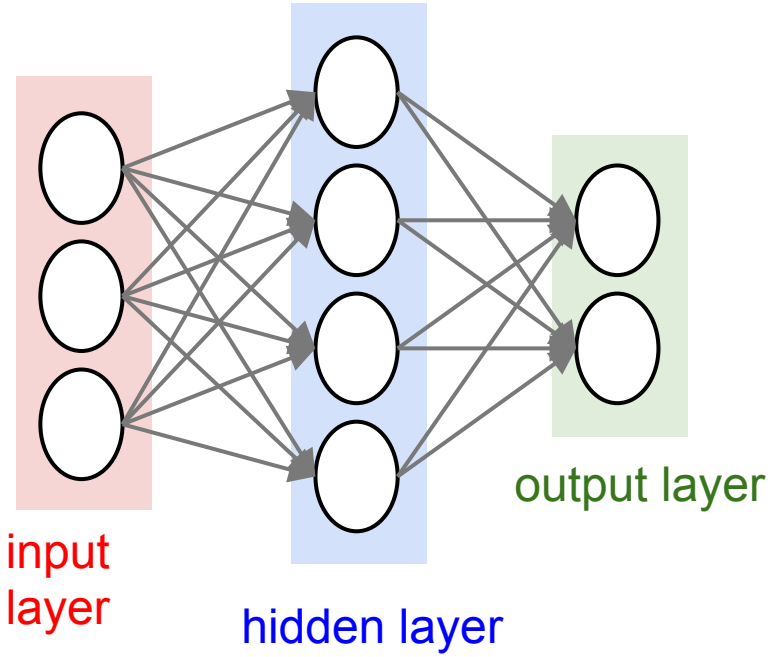
[Ciresan et al. 2013]

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

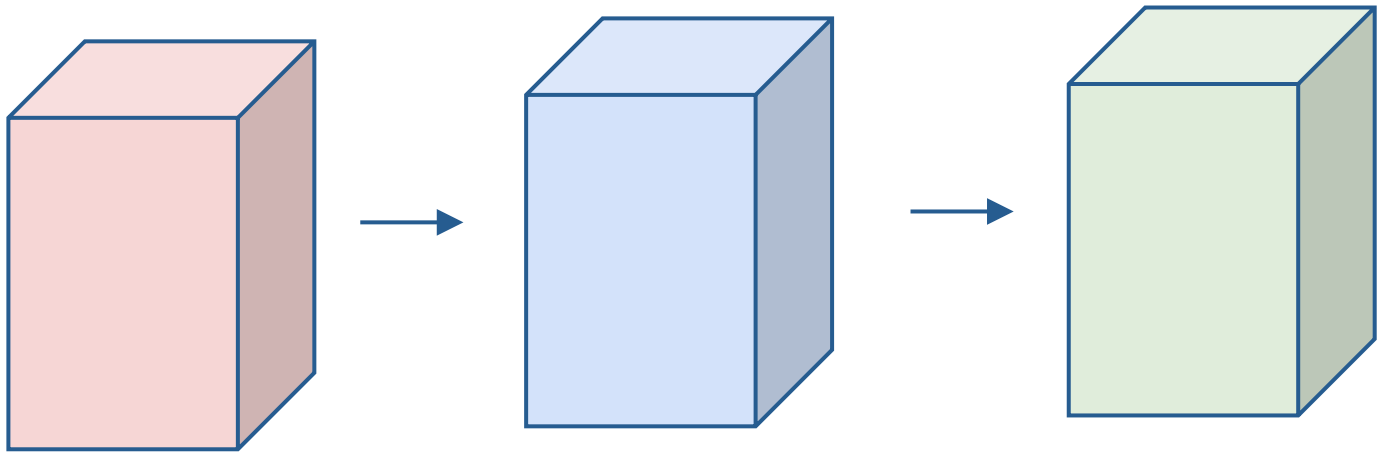
Source: Andrej Karpathy & Fei-Fei Li

Source: Andrej Karpathy & Fei-Fei Li

before:

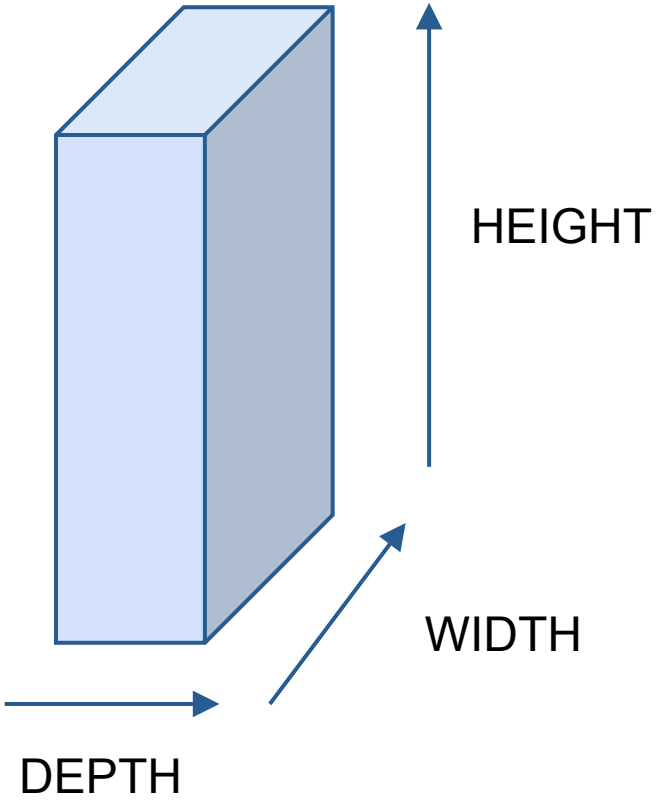


now:



Source: Andrej Karpathy & Fei-Fei Li

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

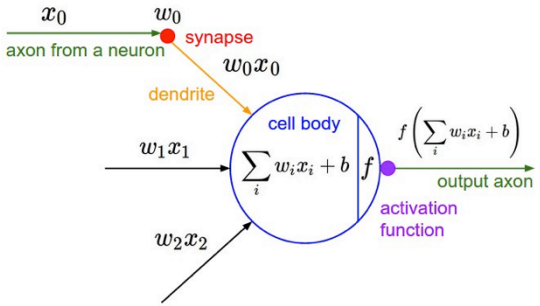


For example, a CIFAR-10 image is a 32x32x3 volume
32 width, 32 height, 3 depth (RGB channels)

Source: Andrej Karpathy & Fei-Fei Li

Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity



nothing changes

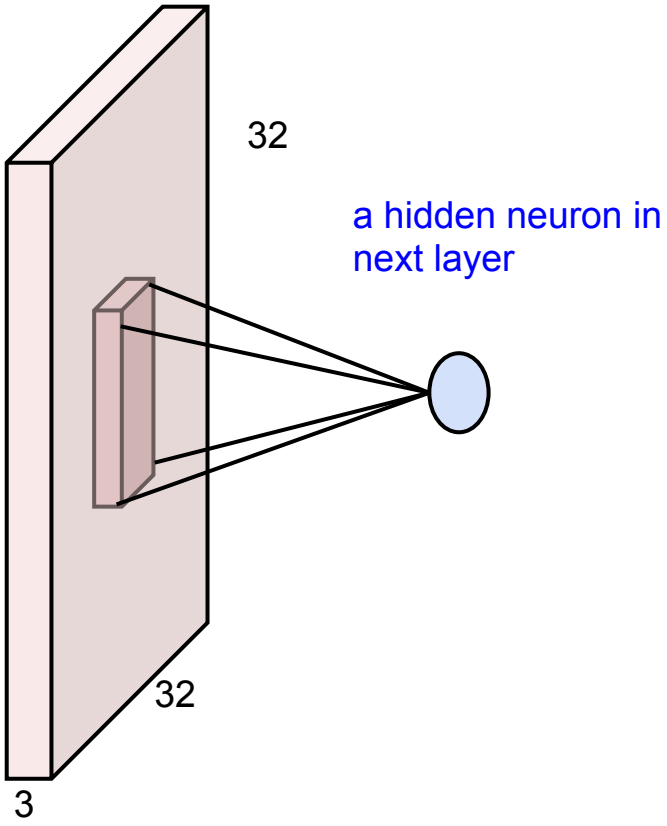


image: 32x32x3 volume
before: full connectivity: 32x32x3 weights

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

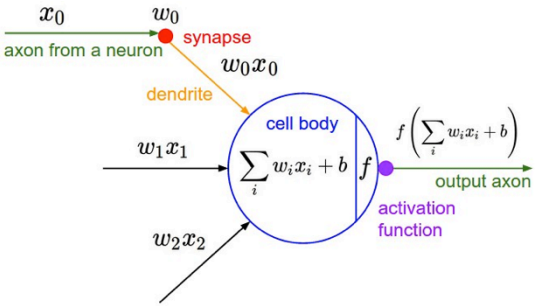
note that connectivity is:

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

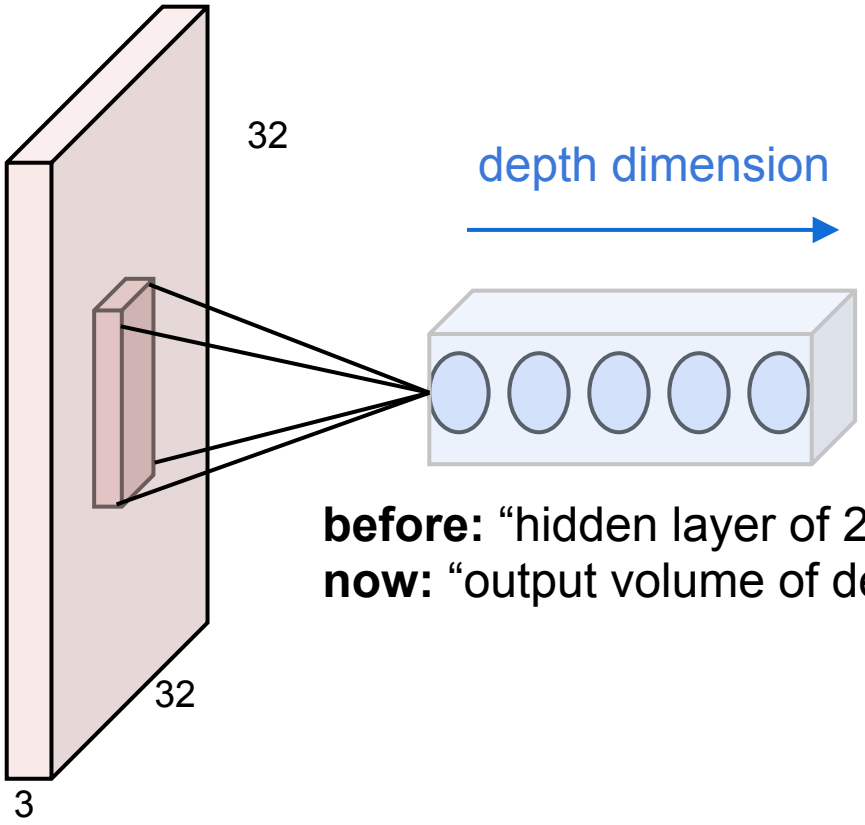
Source: Andrej Karpathy & Fei-Fei Li

Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity



nothing changes



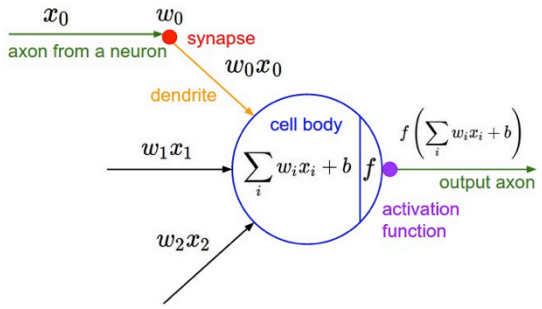
before: “hidden layer of 200 neurons”
now: “output volume of depth 200”

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

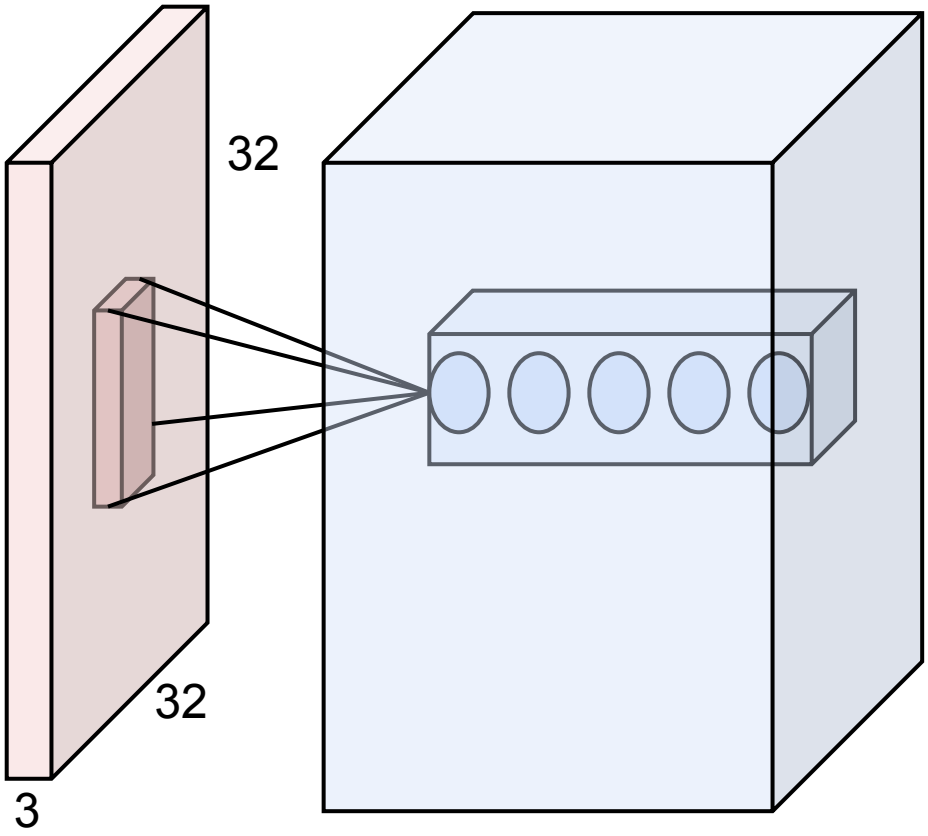
Source: Andrej Karpathy & Fei-Fei Li

Convolutional Neural Networks are just Neural Networks BUT:

1. Local connectivity



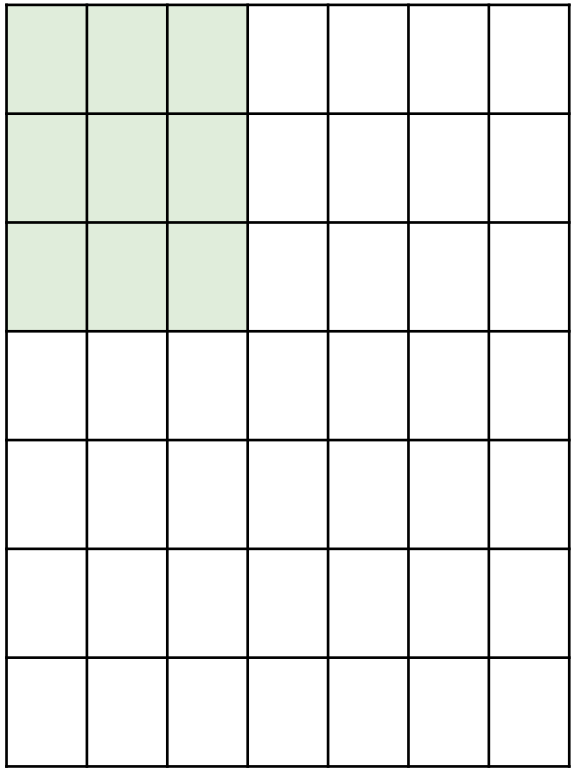
nothing changes



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

Source: Andrej Karpathy & Fei-Fei Li

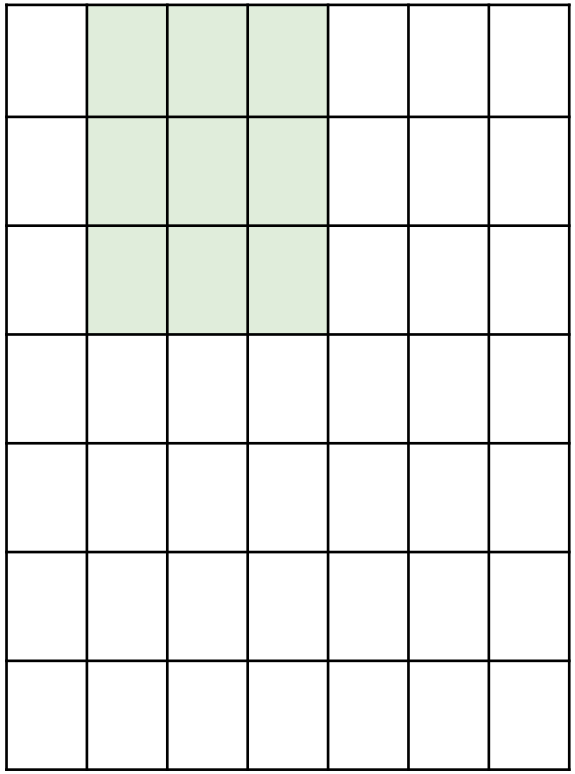
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

Source: Andrej Karpathy & Fei-Fei Li

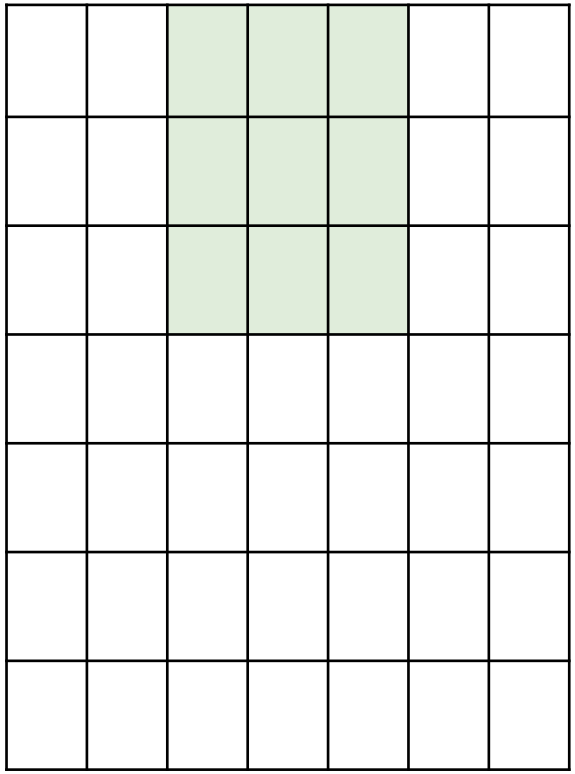
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

Source: Andrej Karpathy & Fei-Fei Li

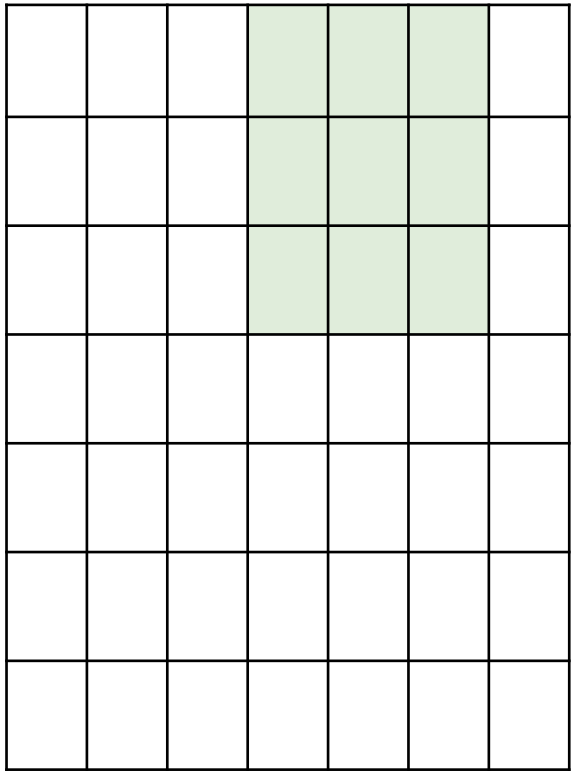
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

Source: Andrej Karpathy & Fei-Fei Li

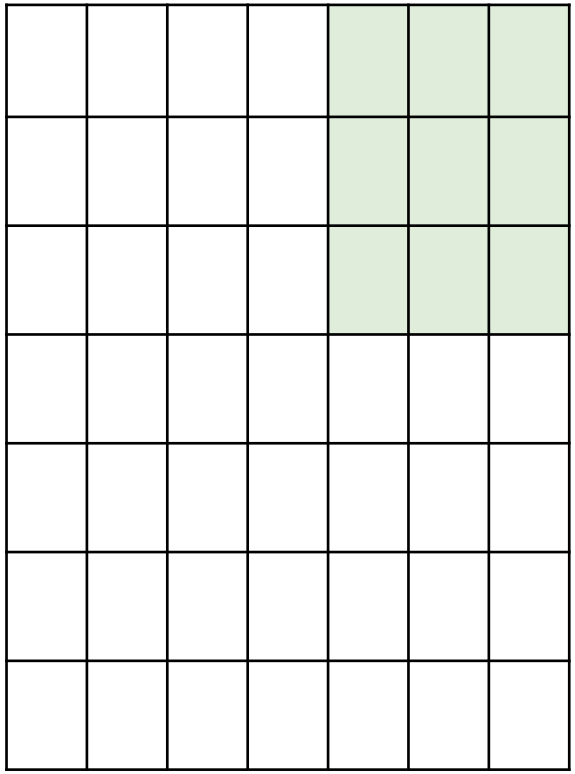
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

Source: Andrej Karpathy & Fei-Fei Li

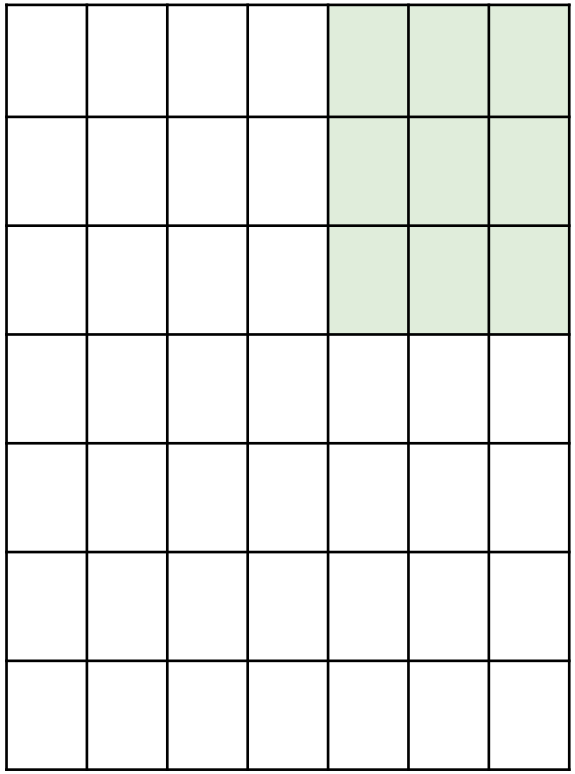
Replicate this column of hidden neurons across space, with some **stride.**



7x7 input
assume 3x3 connectivity, stride 1

Source: Andrej Karpathy & Fei-Fei Li

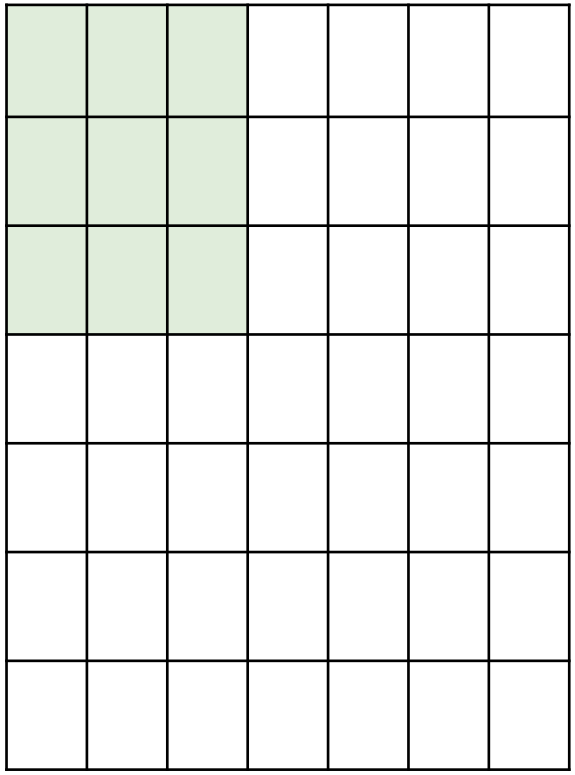
Replicate this column of hidden neurons across space, with some **stride.**



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

Source: Andrej Karpathy & Fei-Fei Li

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

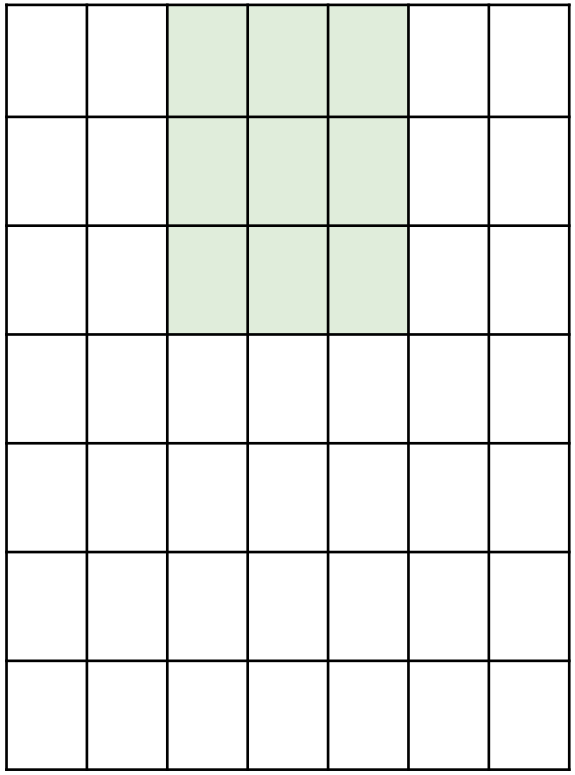


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

Source: Andrej Karpathy & Fei-Fei Li

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

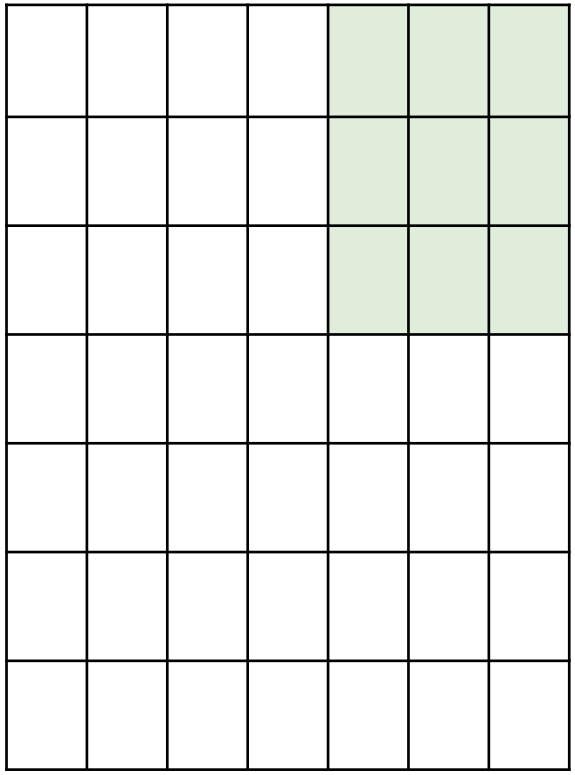


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

Source: Andrej Karpathy & Fei-Fei Li

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

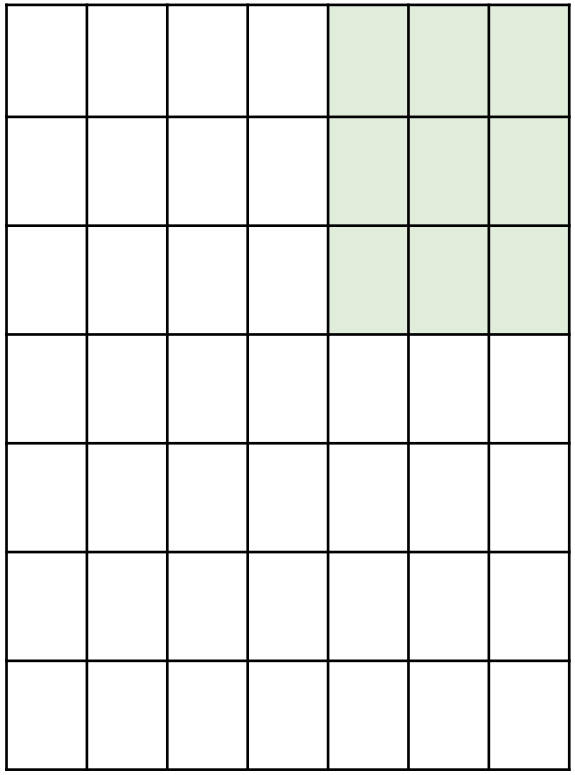


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

what about stride 2?

Source: Andrej Karpathy & Fei-Fei Li

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

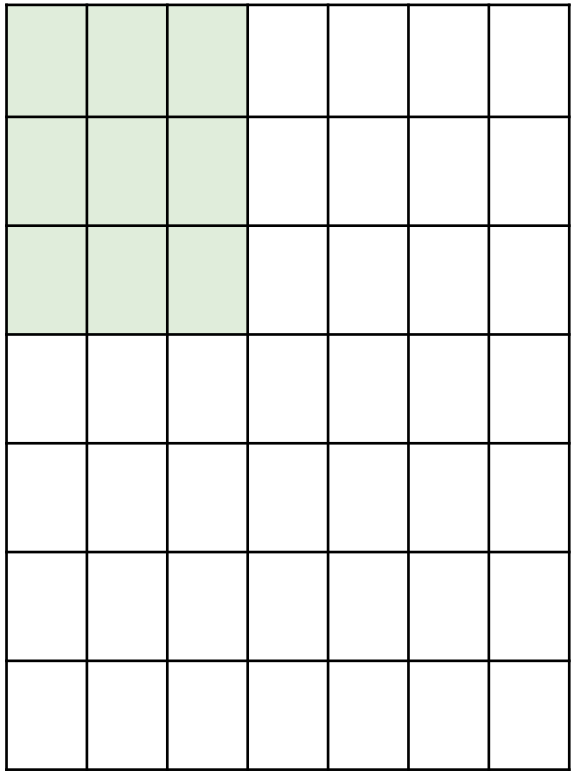


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

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

Source: Andrej Karpathy & Fei-Fei Li

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



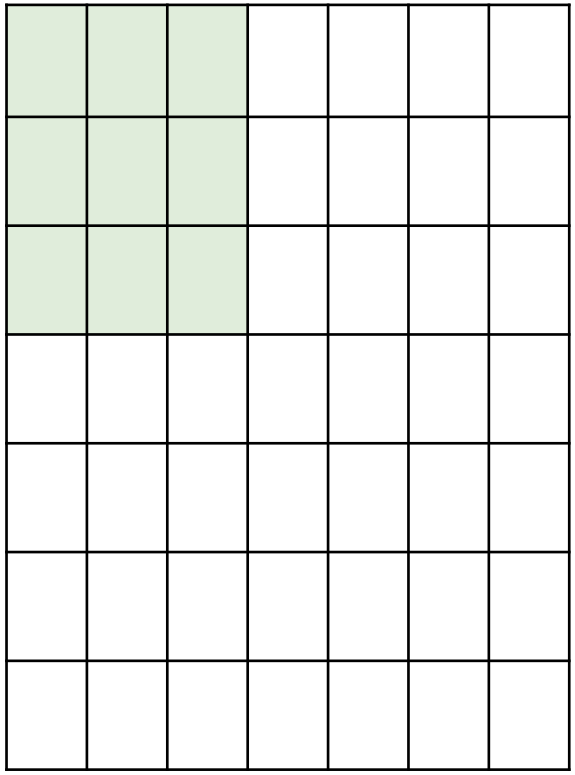
7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

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

what about stride 3?

Source: Andrej Karpathy & Fei-Fei Li

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

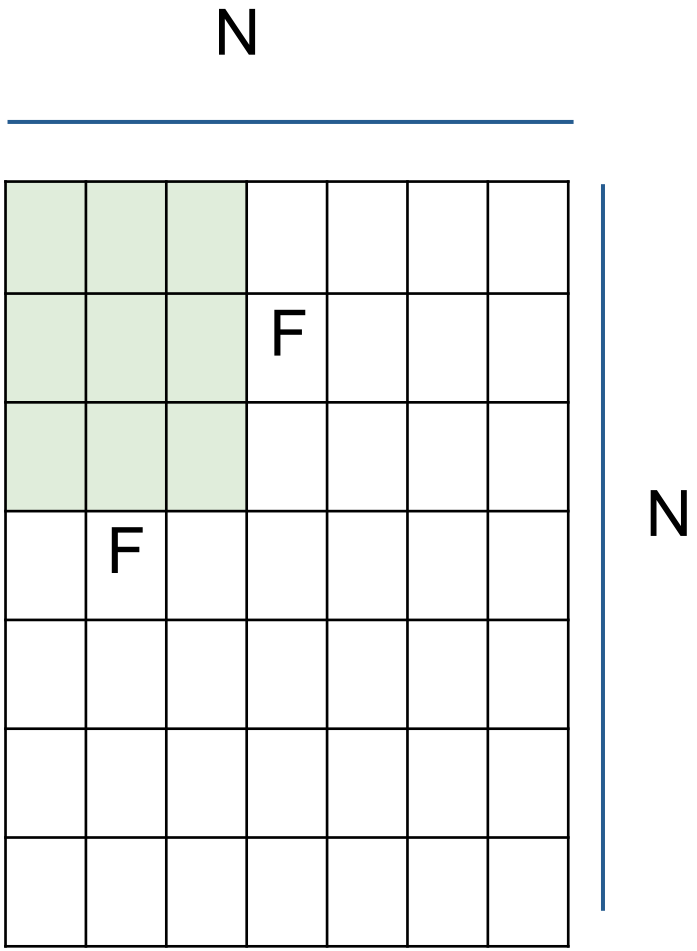


7x7 input
assume 3x3 connectivity, stride 1
=> **5x5 output**

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

what about stride 3? Cannot.

Source: Andrej Karpathy & Fei-Fei Li

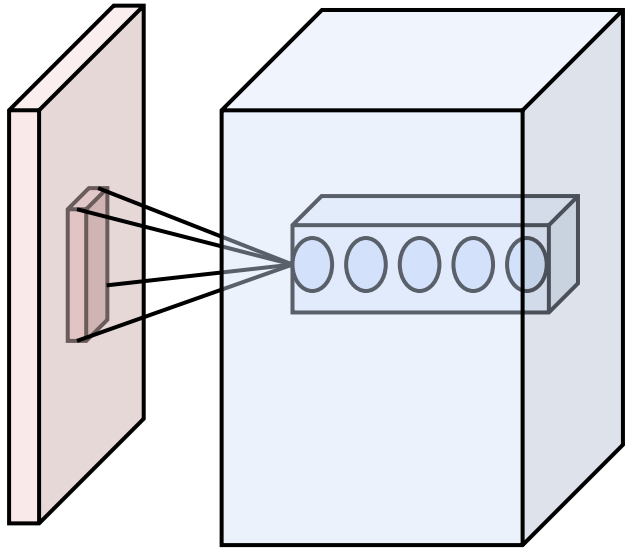


Output size:
(N - F) / stride + 1

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 = … :\

Source: Andrej Karpathy & Fei-Fei Li

Examples time:

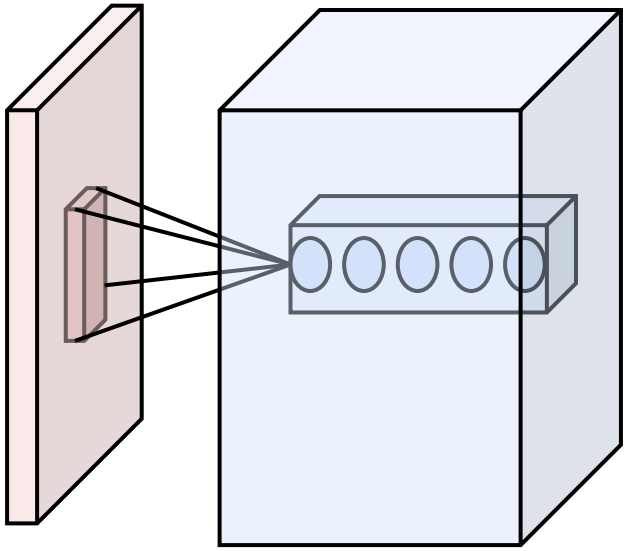


Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

Output volume: ?

Source: Andrej Karpathy & Fei-Fei Li

Examples time:

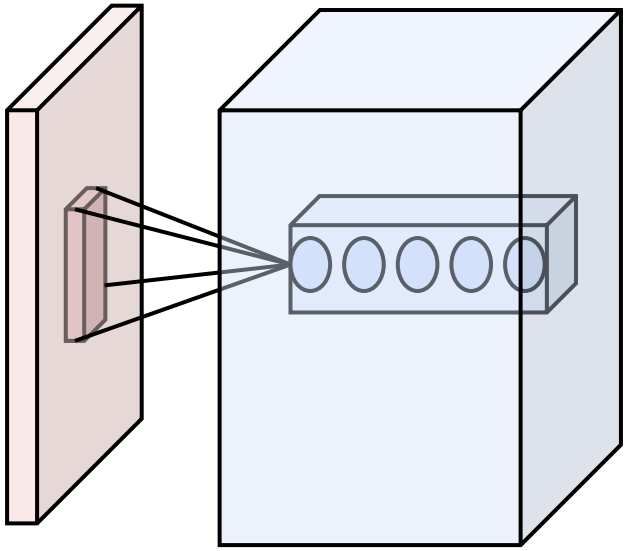


Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 1**
Number of neurons: **5**

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

Source: Andrej Karpathy & Fei-Fei Li

Examples time:

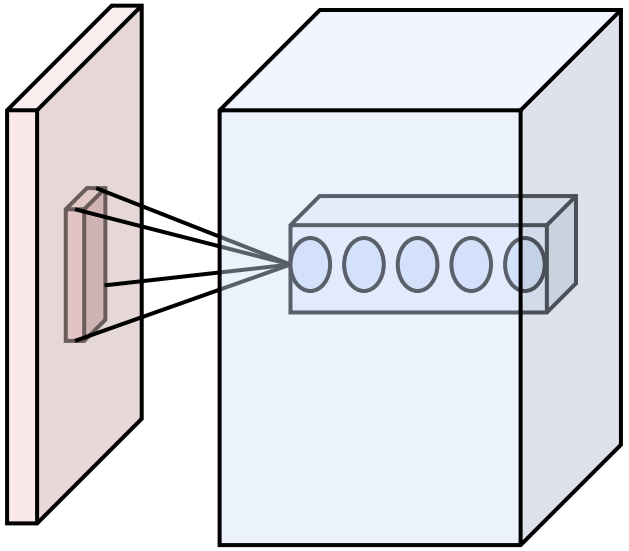


Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 3**
Number of neurons: **5**

Output volume: ?

Source: Andrej Karpathy & Fei-Fei Li

Examples time:

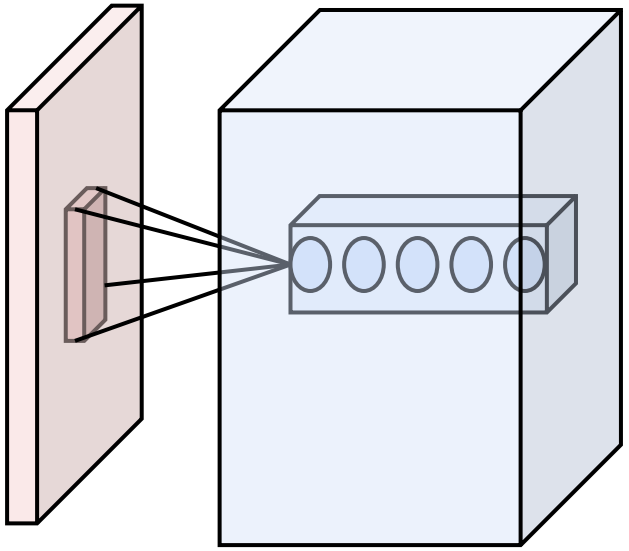


Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 3**
Number of neurons: **5**

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

Source: Andrej Karpathy & Fei-Fei Li

Examples time:



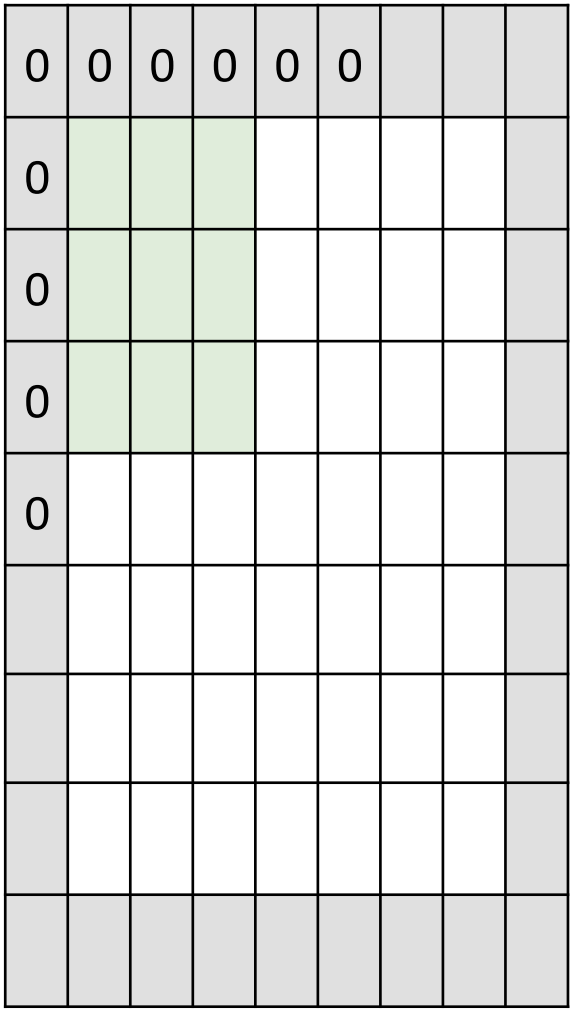
Input volume: **32x32x3**
Receptive fields: **5x5**, **stride 3**
Number of neurons: **5**

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

Source: Andrej Karpathy & Fei-Fei Li

In practice: Common to zero pad the border

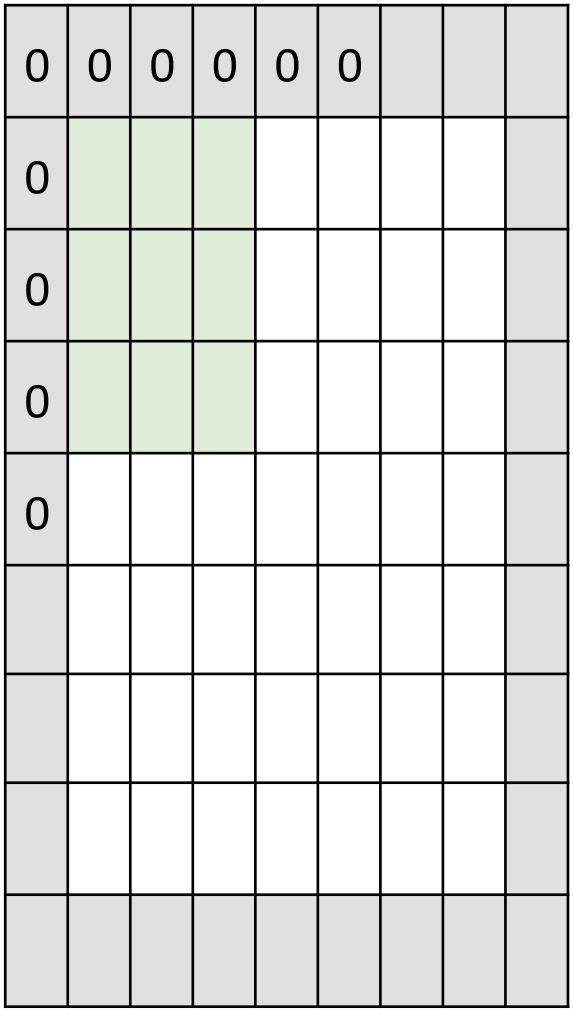
(in each channel)



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

Source: Andrej Karpathy & Fei-Fei Li

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)

Source: Andrej Karpathy & Fei-Fei Li

“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

Source: Andrej Karpathy & Fei-Fei Li

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

Source: Andrej Karpathy & Fei-Fei Li

There's one more problem…

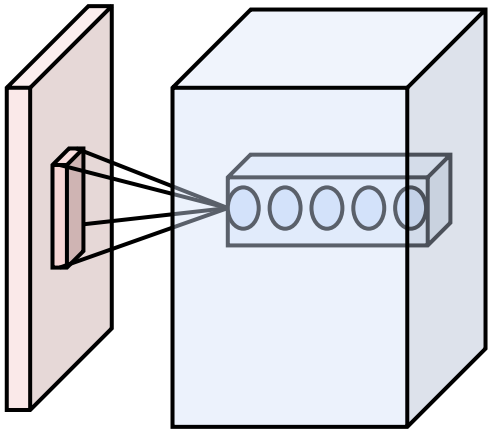
Assume input **[32 x 32 x3]**

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

Source: Andrej Karpathy & Fei-Fei Li

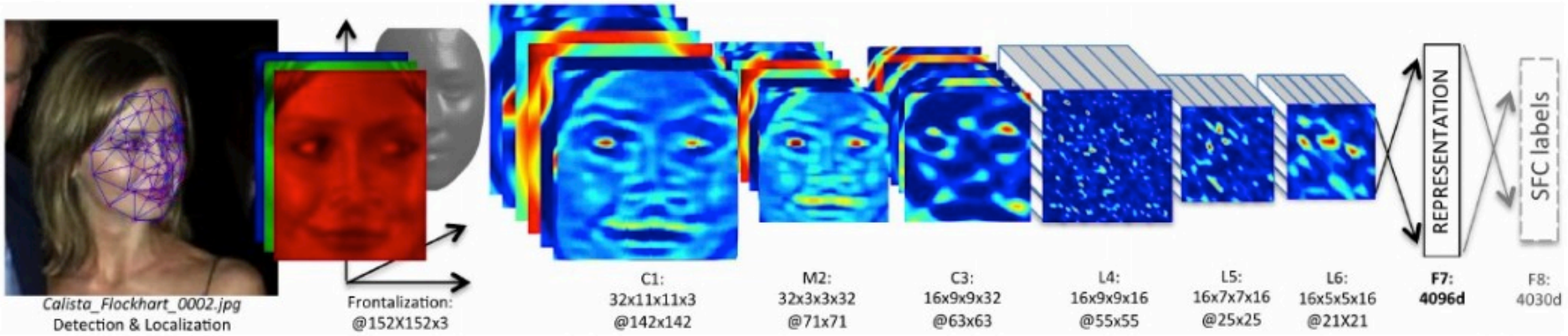
Our first ConvNet layer had size **[32 x 32 x3]**
If we had **30** neurons with receptive fields **5x5**, **stride 1, pad 1**
Output volume: [32 x 32 x 30] (32*32*30 = 30720 neurons)
Each neuron has 5*5*3 (=75) weights

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

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

Source: Andrej Karpathy & Fei-Fei Li

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

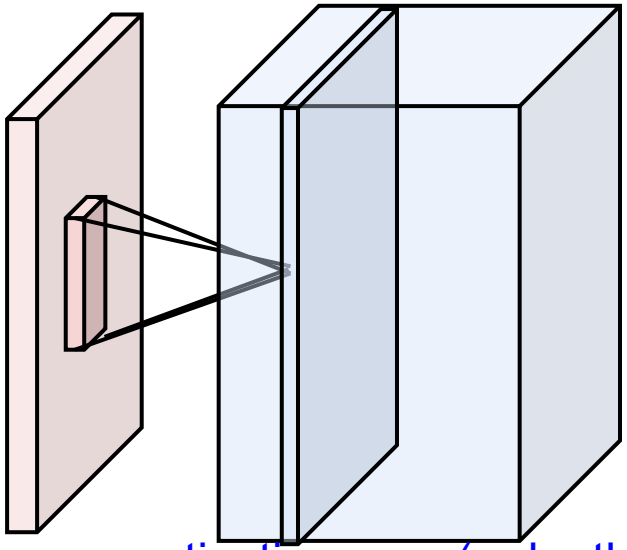


We'd like to be able to learn different things at different spatial positions

Source: Andrej Karpathy & Fei-Fei Li

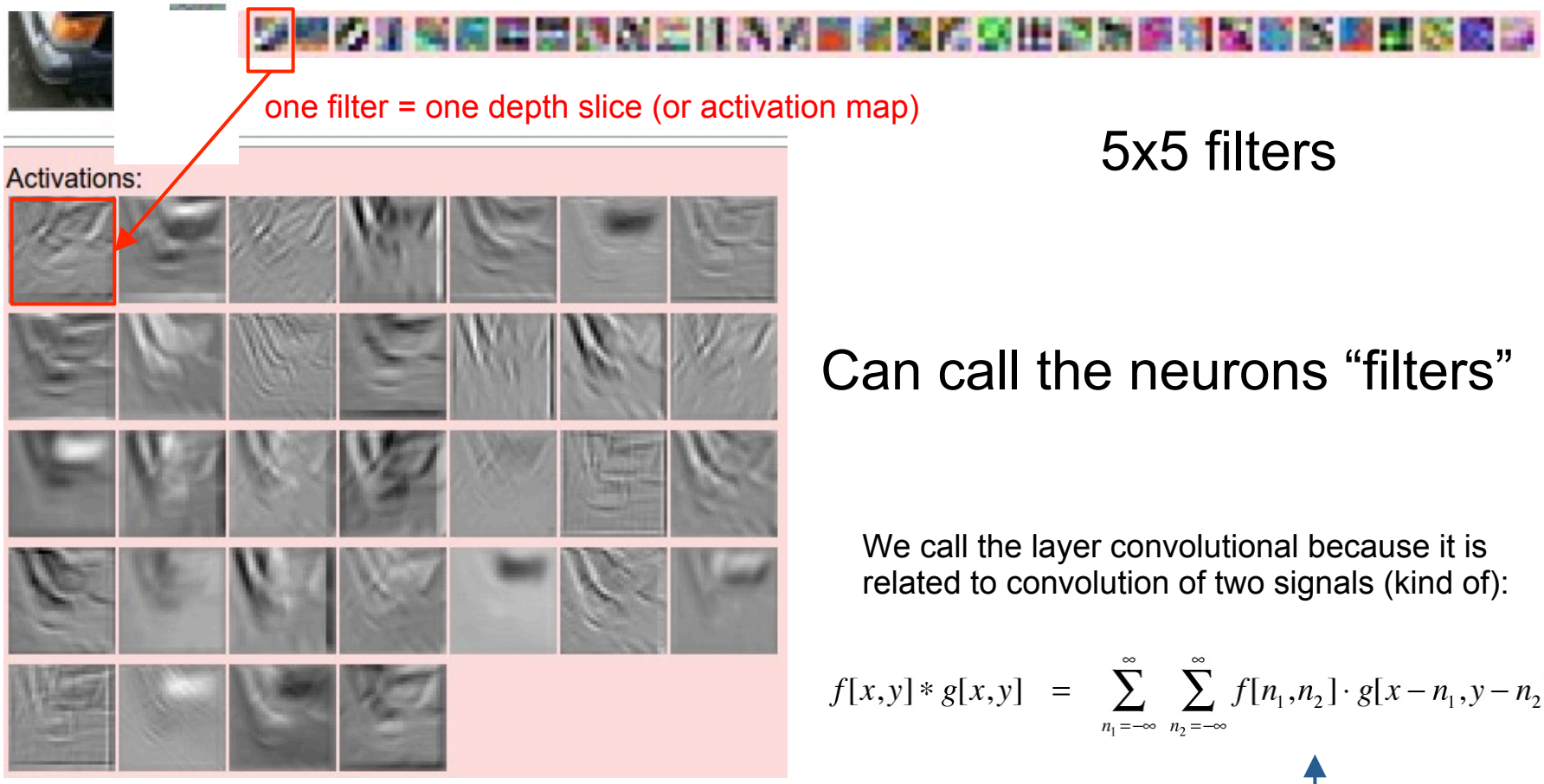
These layers are called **Convolutional Layers**

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

Source: Andrej Karpathy & Fei-Fei Li



5x5 filters

Can call the neurons “filters”

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

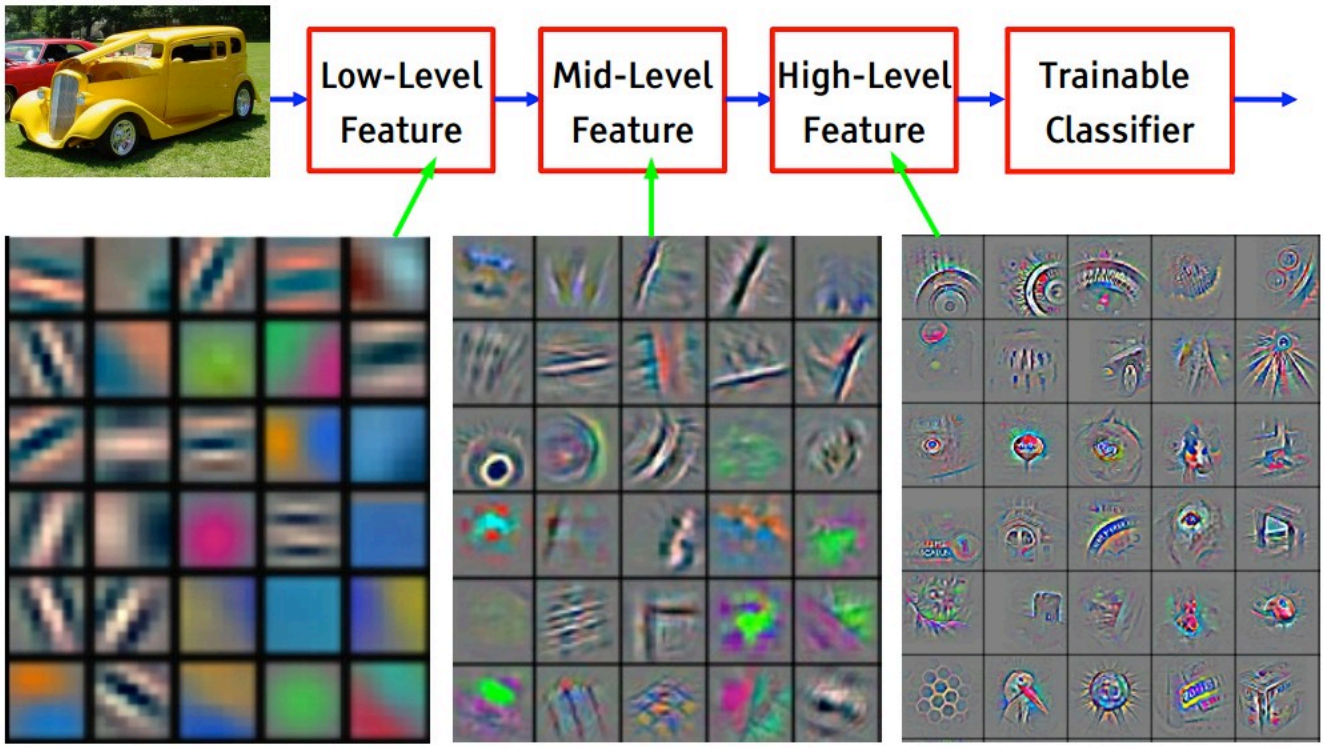
$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

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

Source: Andrej Karpathy & Fei-Fei Li

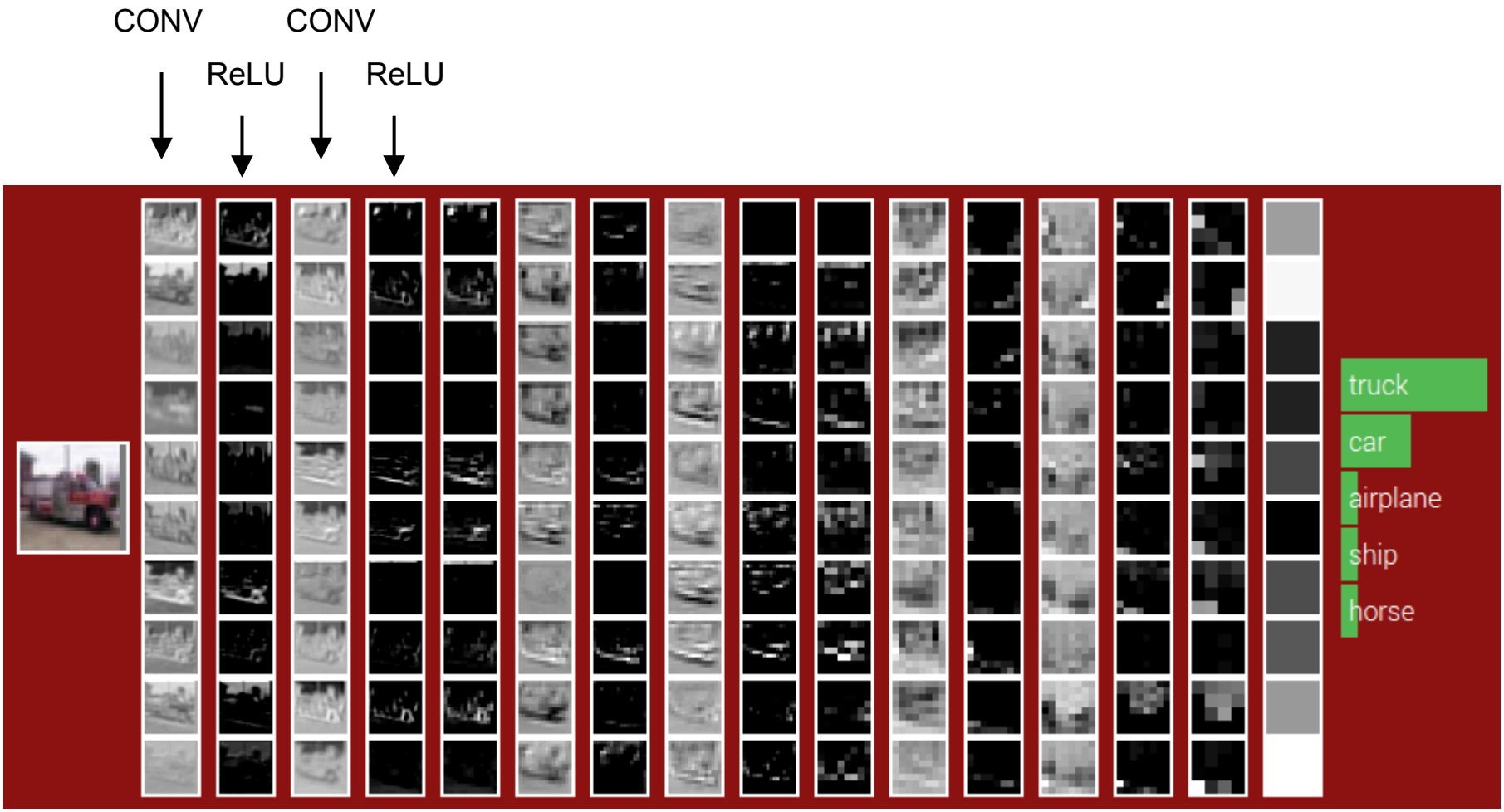
Fast-forward to today

[From recent Yann LeCun slides]



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

Source: Andrej Karpathy & Fei-Fei Li



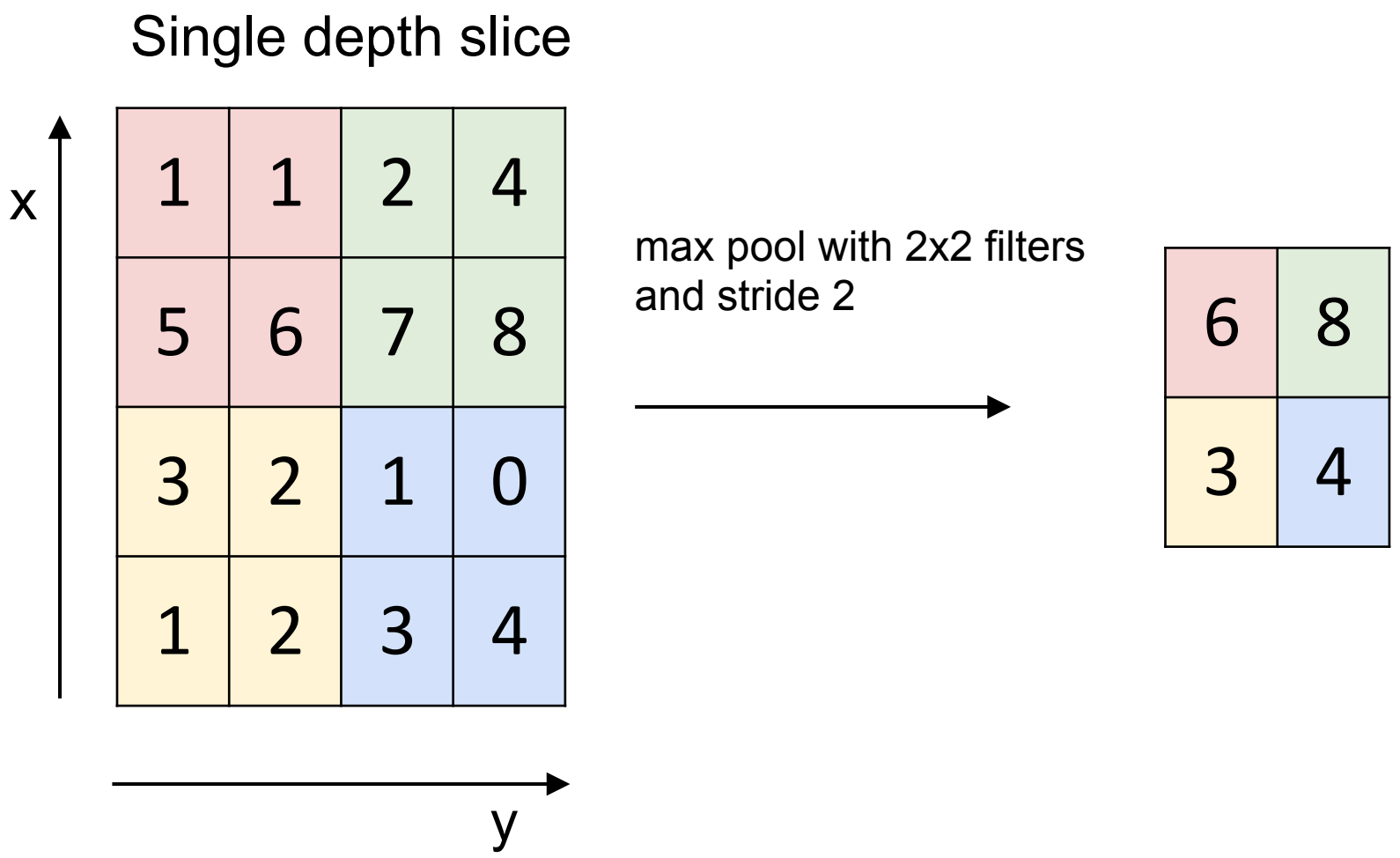
Source: Andrej Karpathy & Fei-Fei Li

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)

Source: Andrej Karpathy & Fei-Fei Li

MAX POOLING



Source: Andrej Karpathy & Fei-Fei Li

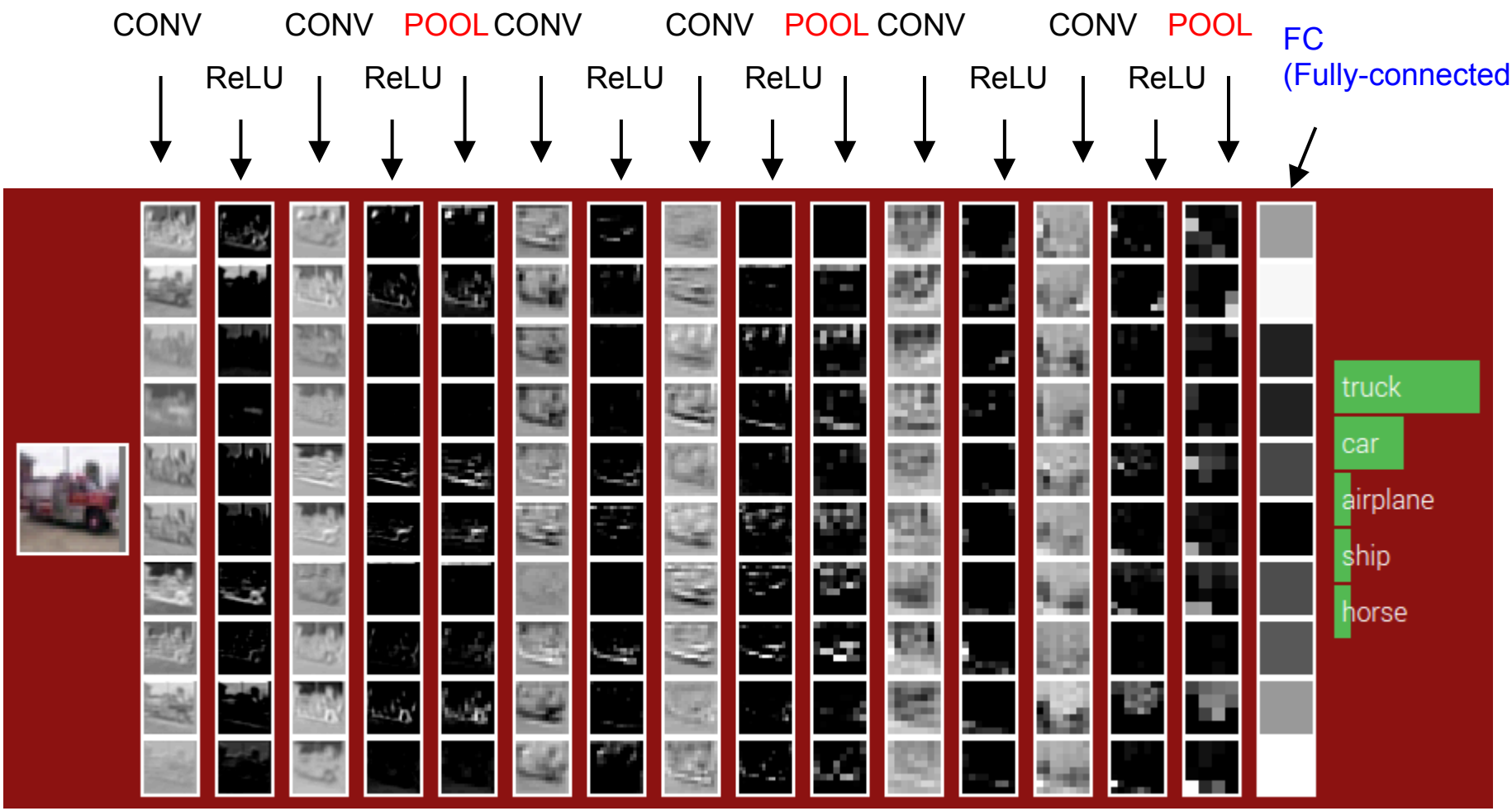
In ConvNet architectures, **Conv** layers are often followed by **Pool** layers

- 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.

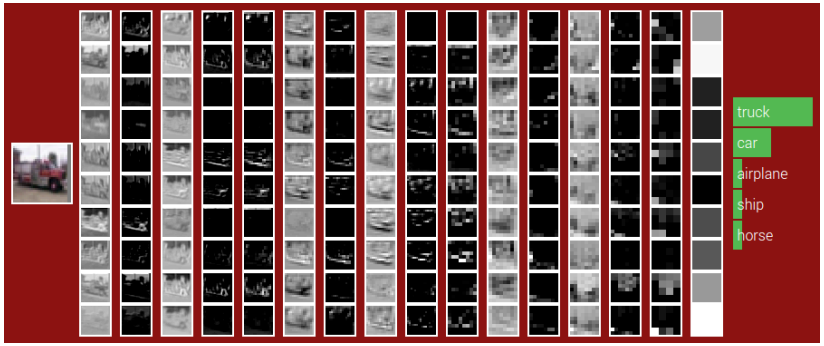
Source: Andrej Karpathy & Fei-Fei Li



Source: Andrej Karpathy & Fei-Fei Li

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



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

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

Source: Andrej Karpathy & Fei-Fei Li

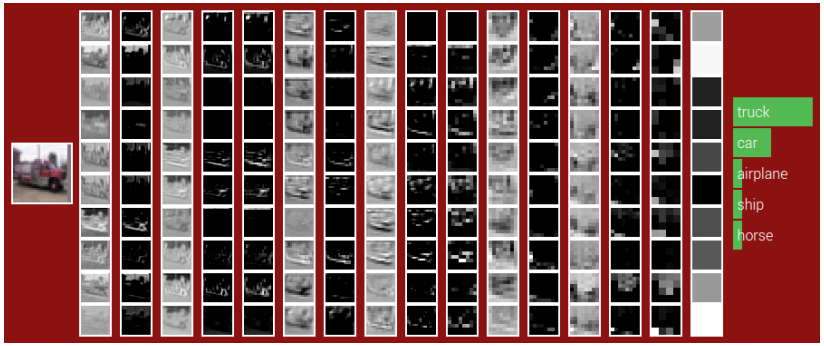
Example:

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!



Source: Andrej Karpathy & Fei-Fei Li

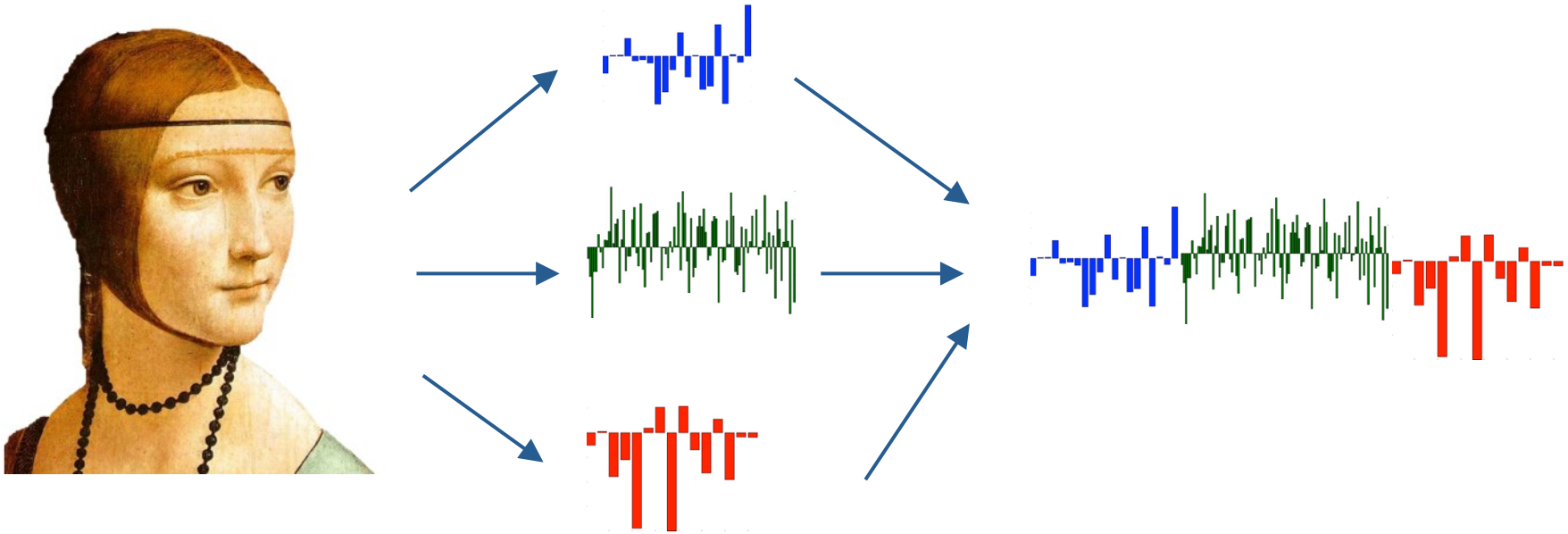
Summary:

- 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)

Source: Andrej Karpathy & Fei-Fei Li

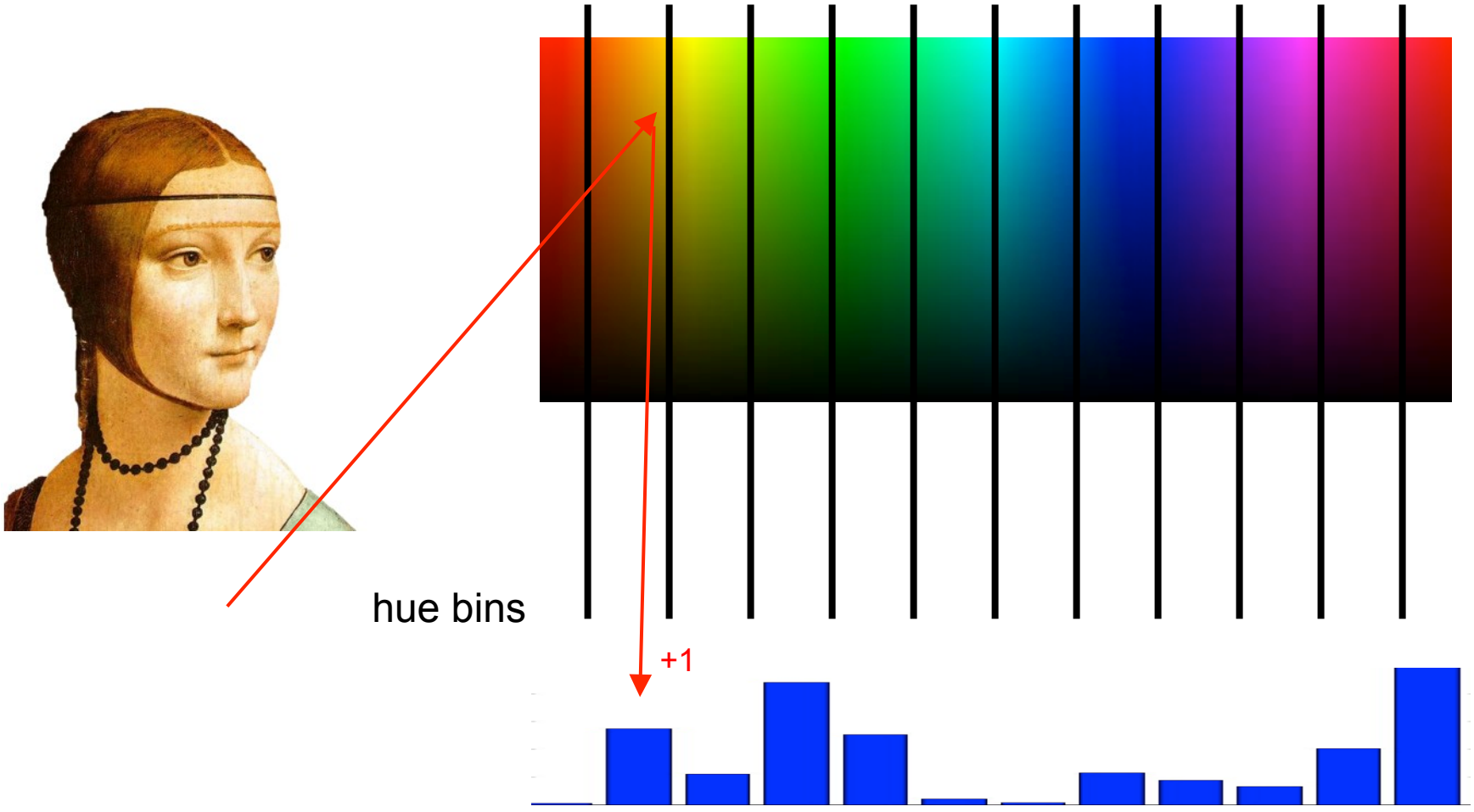
Brief aside: Image Features

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



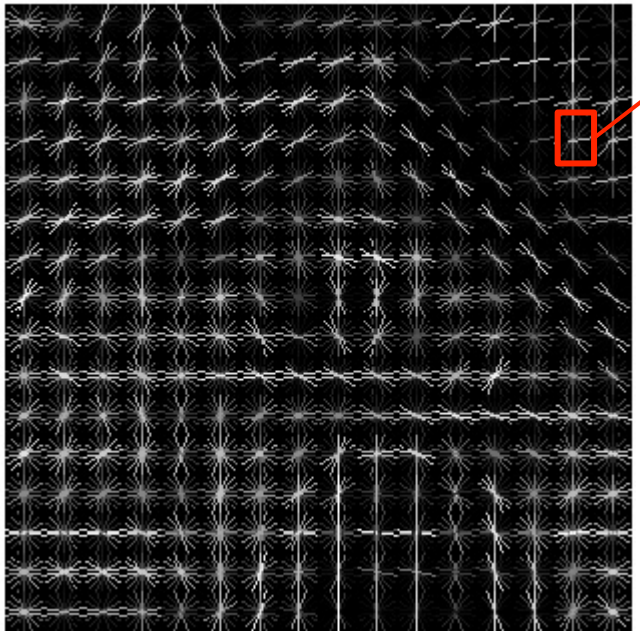
Source: Andrej Karpathy & Fei-Fei Li

Example: Color (Hue) Histogram



Source: Andrej Karpathy & Fei-Fei Li

Example: HOG features

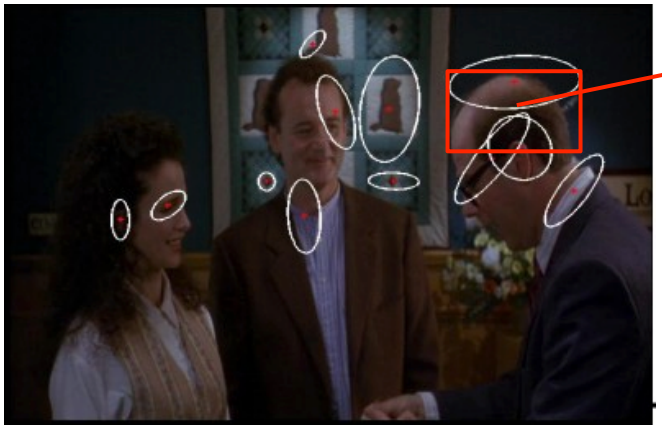


8x8 pixel region, quantize the edge orientation into 9 bins

(images from vlfeat.org)

Source: Andrej Karpathy & Fei-Fei Li

Example: Bag of Words



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

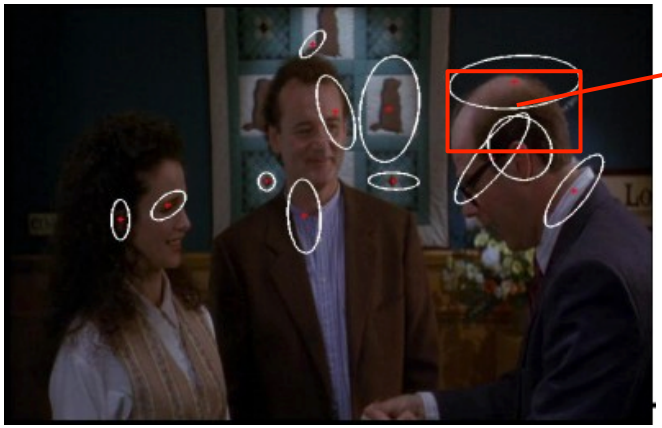
repeat for each detected feature

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

Source: Andrej Karpathy & Fei-Fei Li

Example: Bag of Words



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

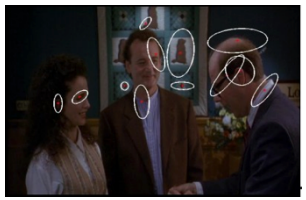
repeat for each detected feature

gives a matrix of size
[number_of_features x 144]

Source: Andrej Karpathy & Fei-Fei Li

Example: Bag of Words

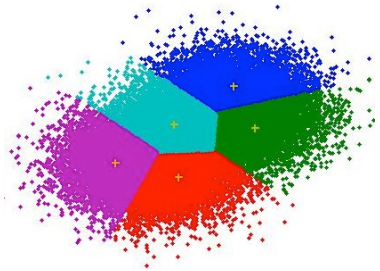
histogram of visual words



visual word vectors

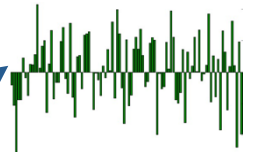
144

learn k-means centroids
"vocabulary of visual words

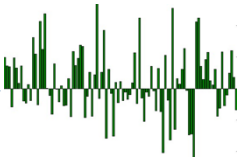


e.g. 1000 centroids

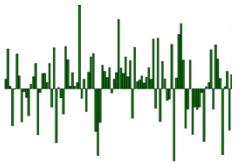
1000-d vector



1000-d vector

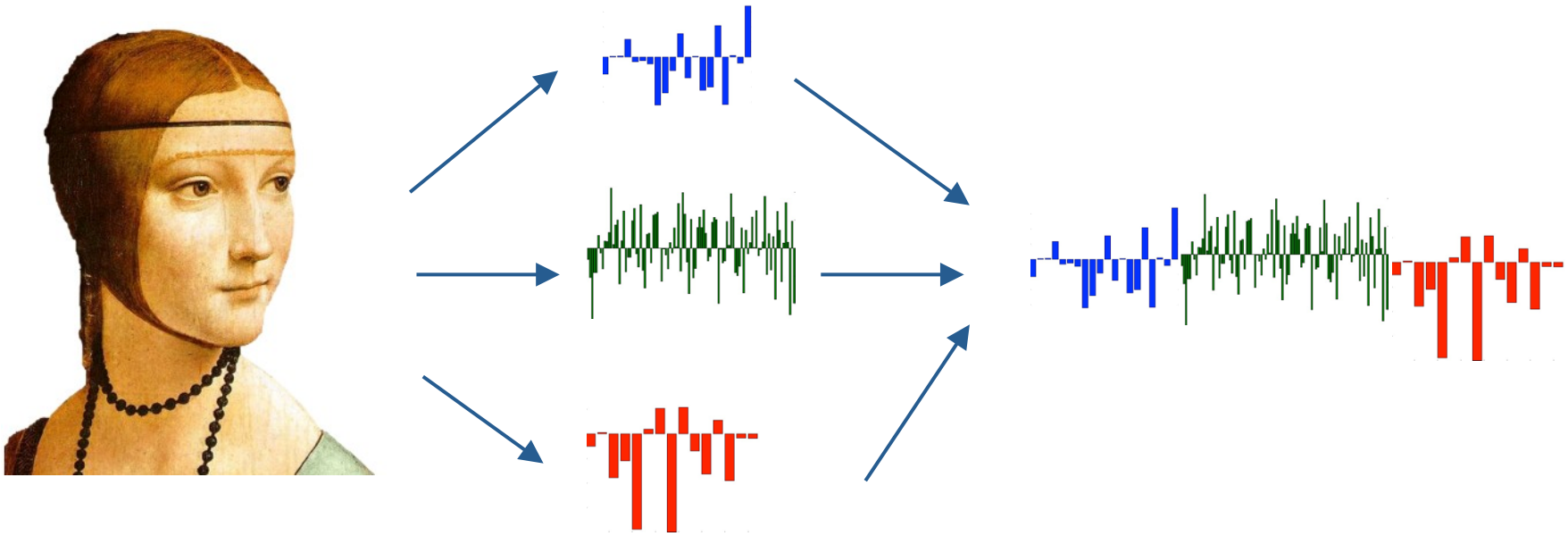


1000-d vector



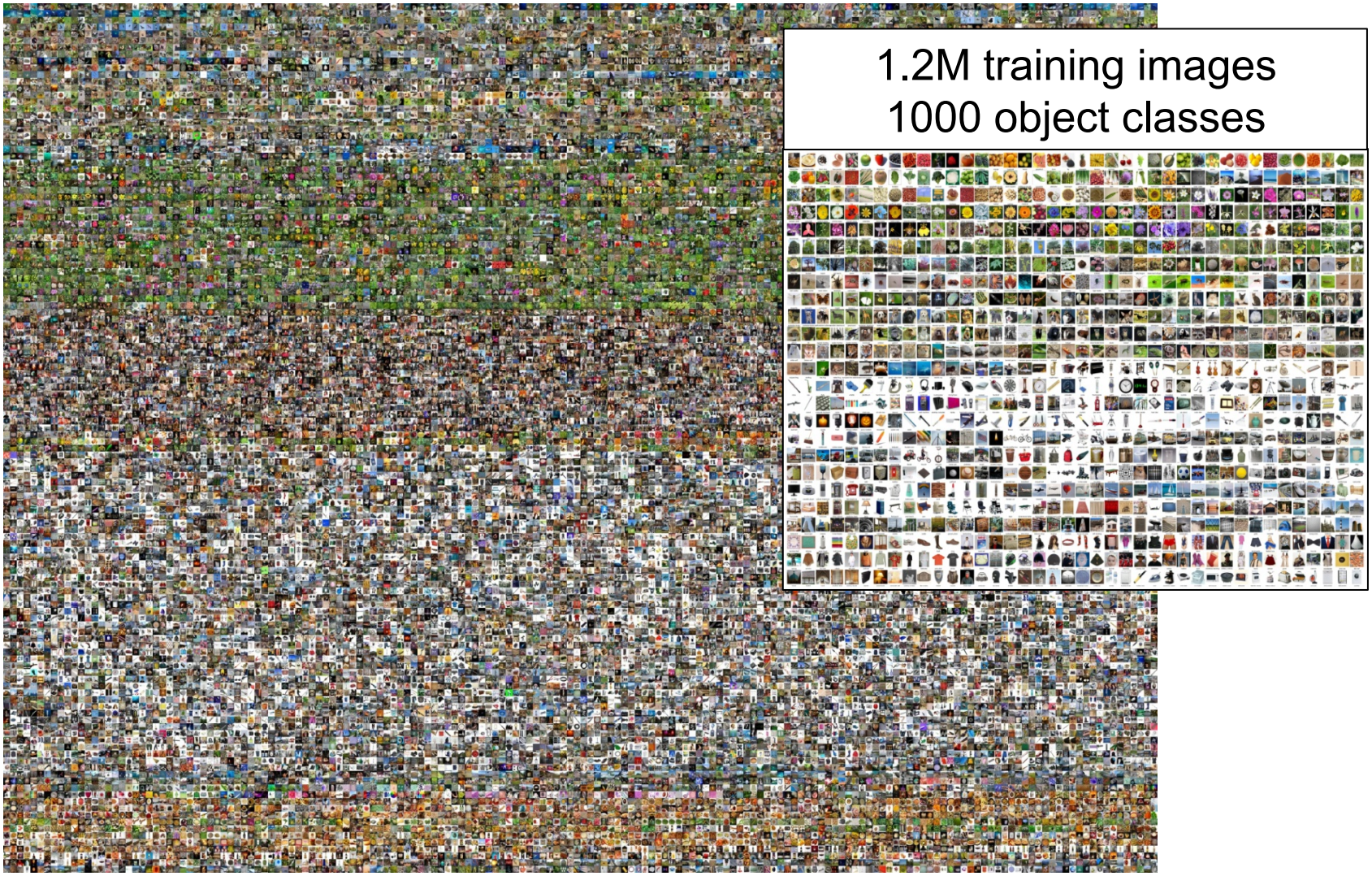
Source: Andrej Karpathy & Fei-Fei Li

Brief aside: Image Features



Source: Andrej Karpathy & Fei-Fei Li

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

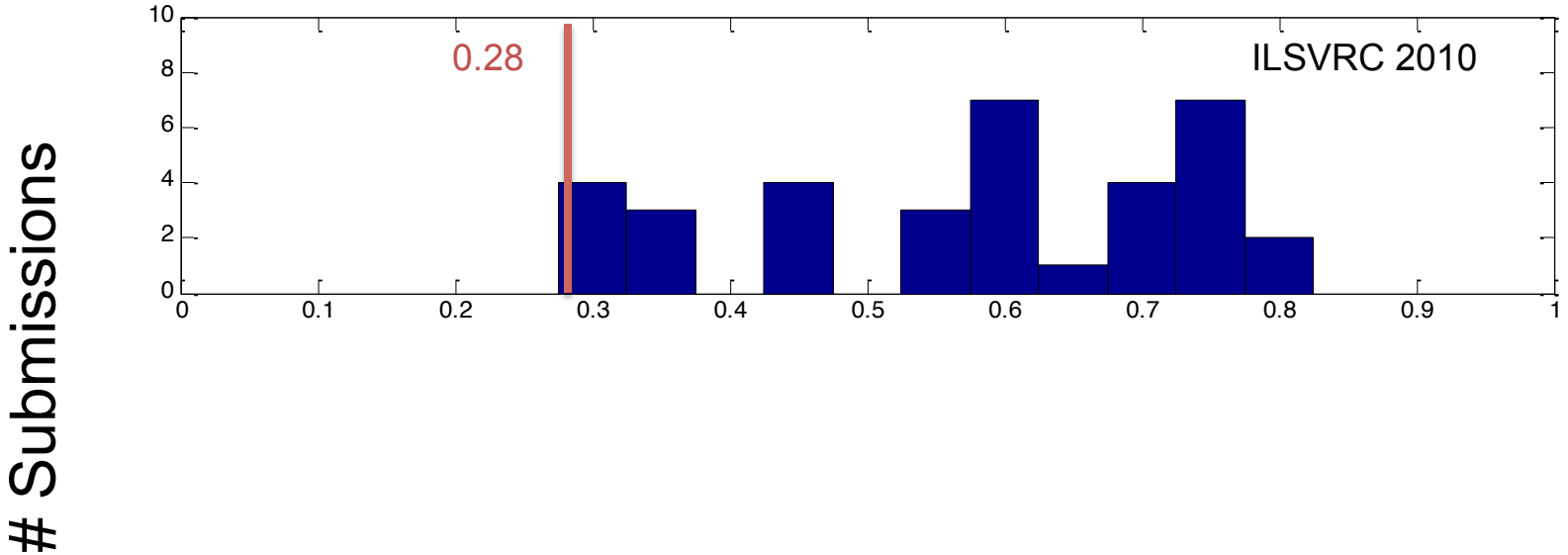


ILSVRC 2010-2014

Task: Classification

Car

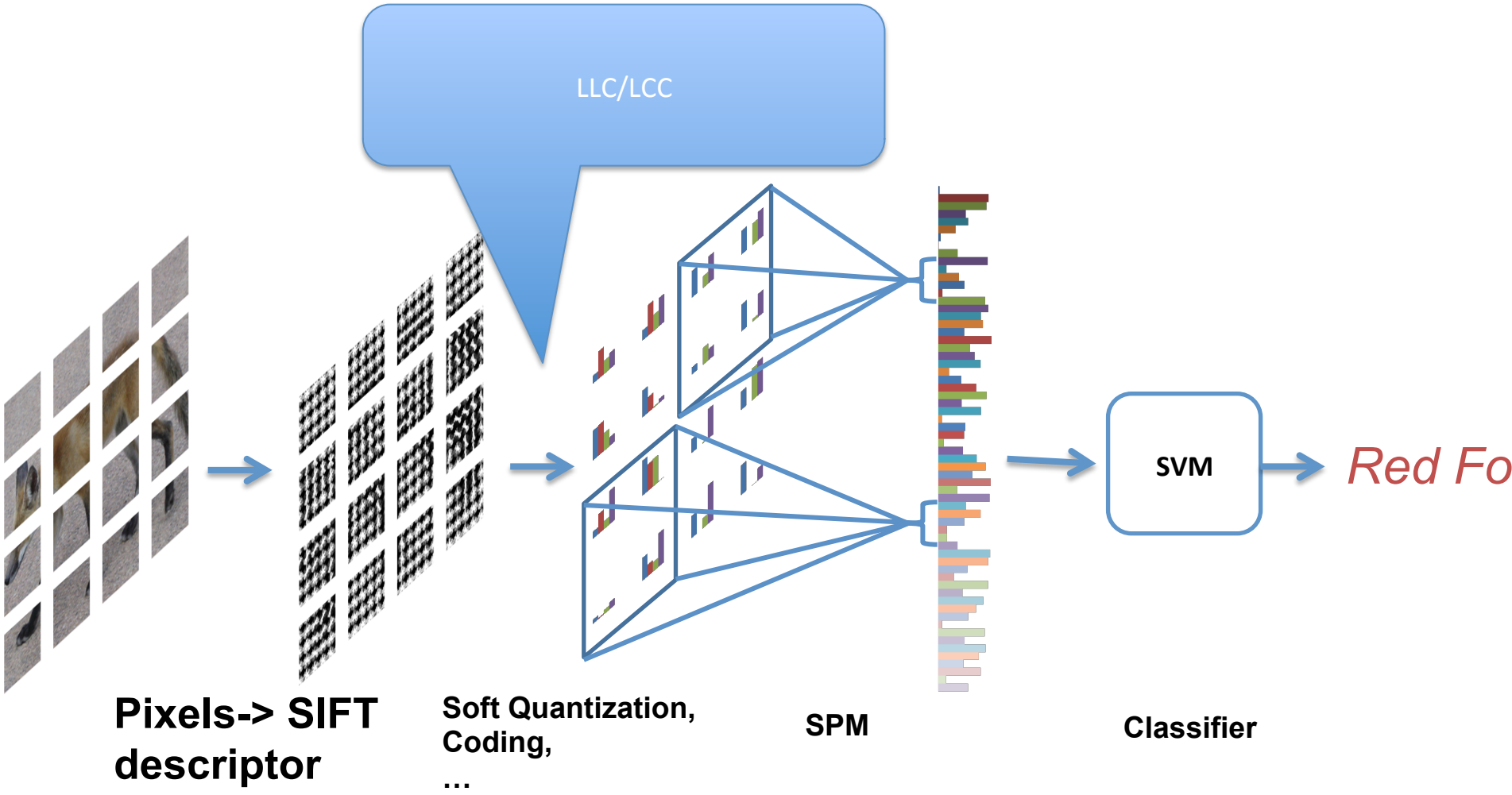
- Predict a class label
- 5 predictions / image
- 1000 classes
- 1,200 images per class for training

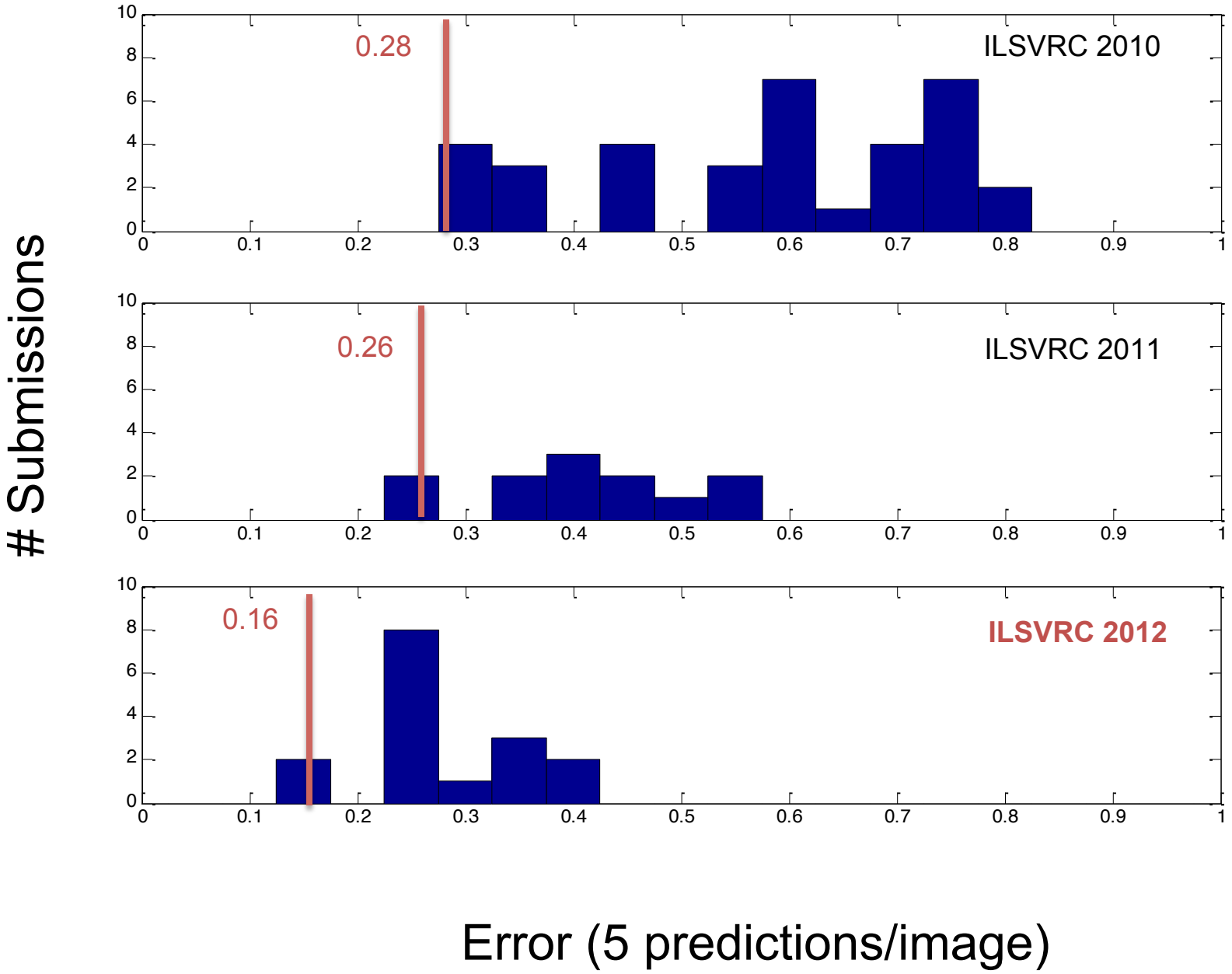

0.28
ILSVRC 2010
Submissions
Error (5 predictions/image)
0
2
4
6
8
10
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Winning Method, ILSVRC 2010:

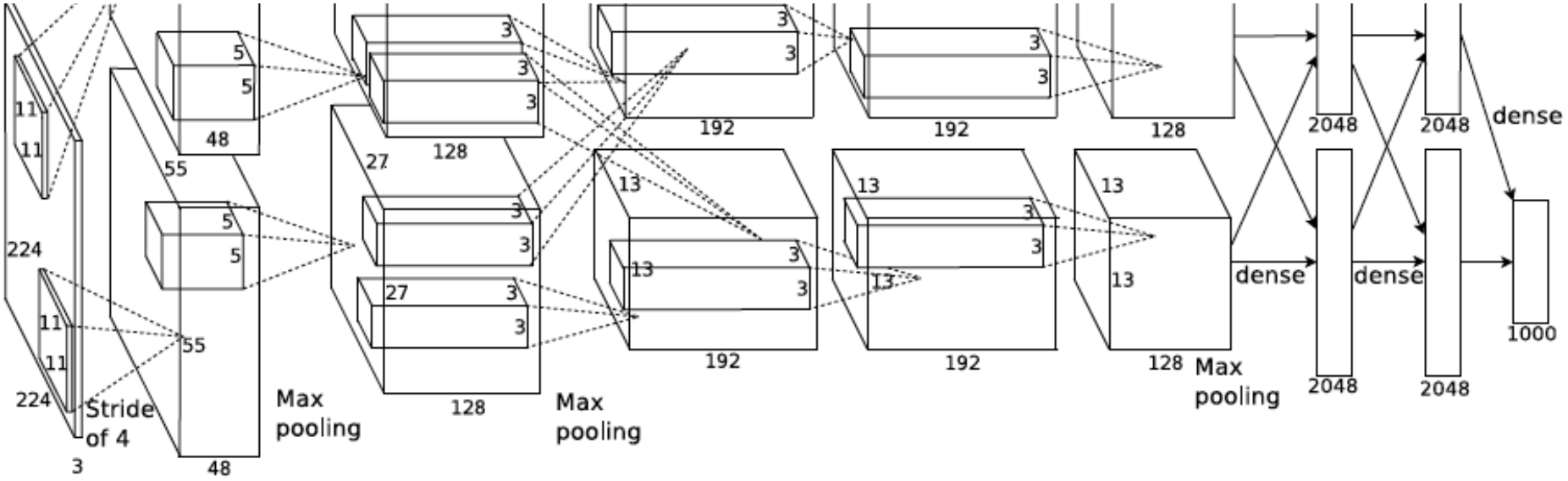
Local coordinate coding (LCC):

K. Yu et.al, NIPS2009; J. Wang et. al, CVPR 2010



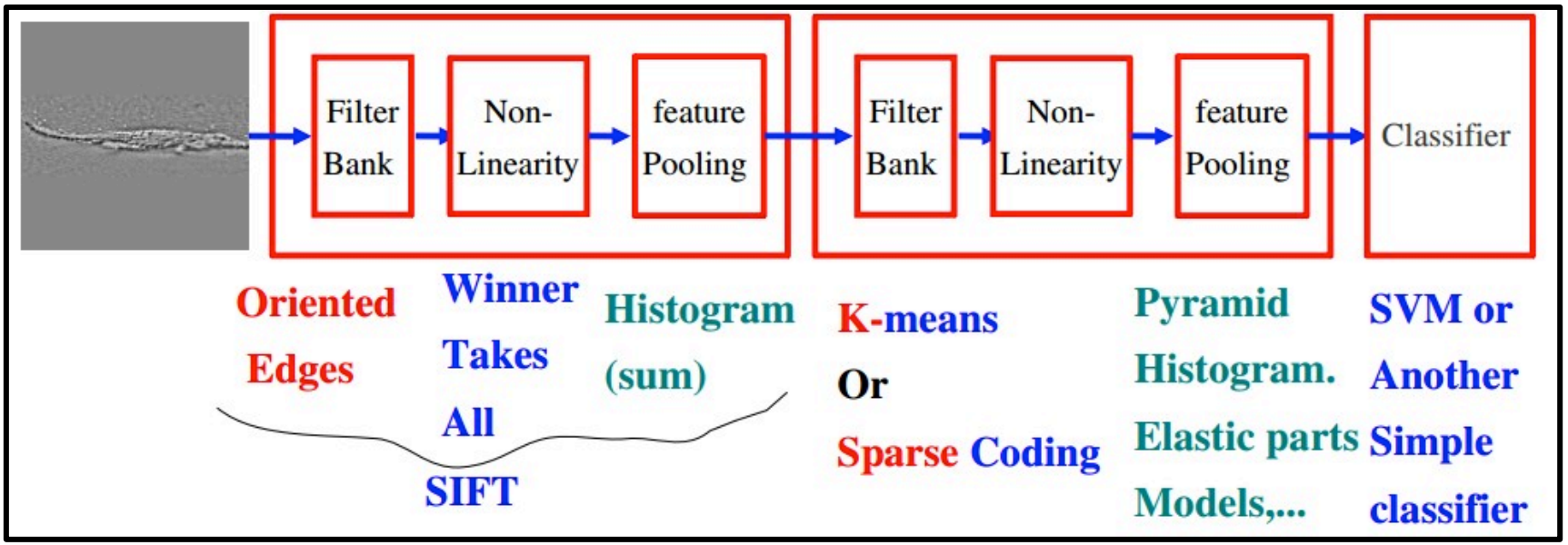

Submissions
0.28
ILSVRC 2010
0.26
ILSVRC 2011
0.16
ILSVRC 2012
Error (5 predictions/image)

Convolutional Neural Networks



A. Krizhevsky, I. Sutskever, and G. Hinton,
ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012

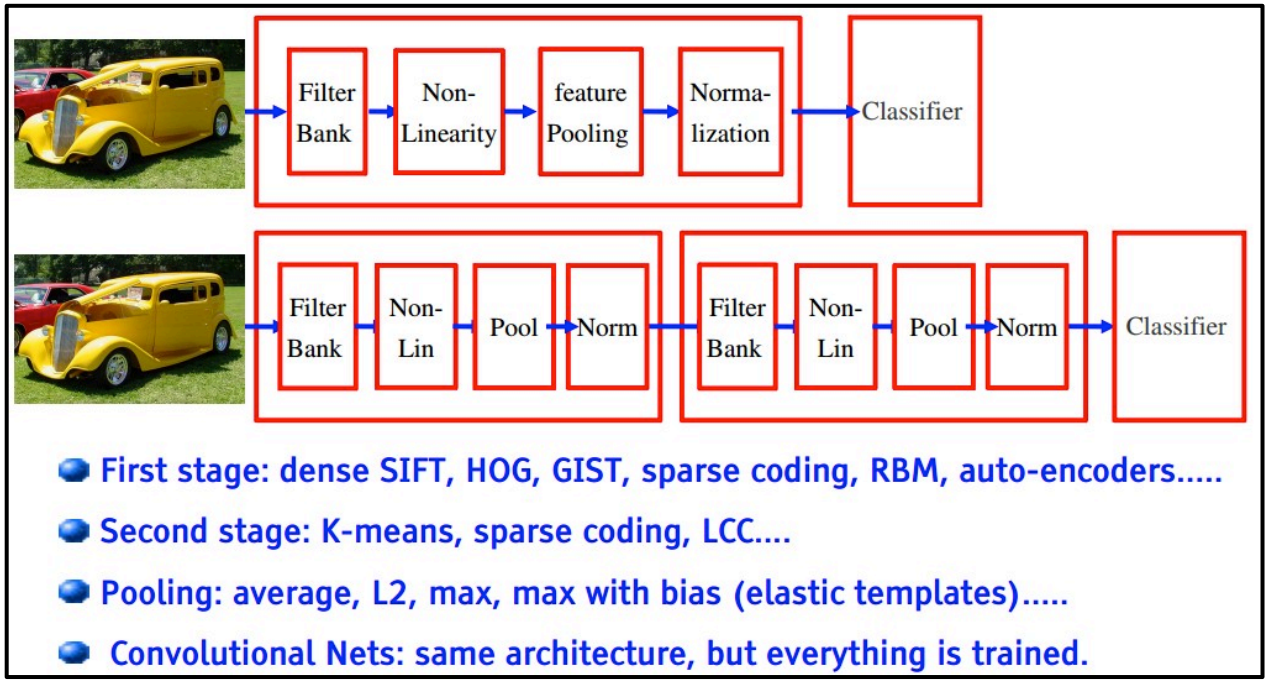
Most recognition systems are build on the same Architecture



(slide from Yann LeCun)

Source: Andrej Karpathy & Fei-Fei Li

Most recognition systems are build on the same Architecture

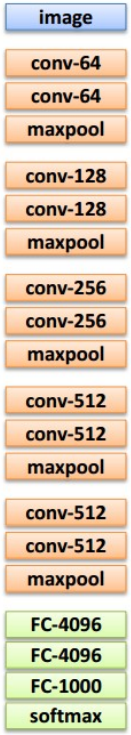


(slide from Yann LeCun)

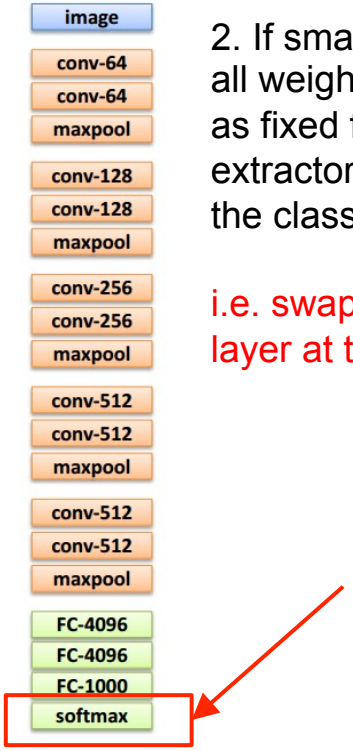
Source: Andrej Karpathy & Fei-Fei Li

Transfer Learning

Transfer Learning with CNNs

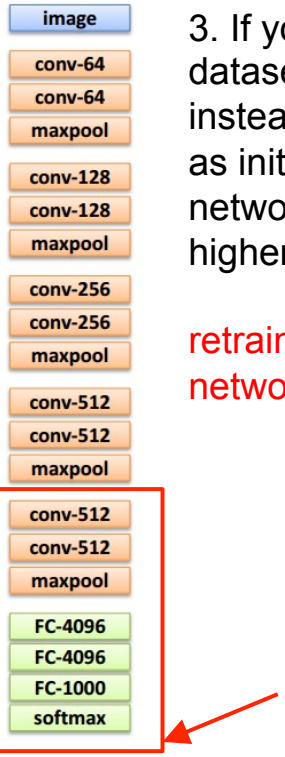


1. Train on Imagenet



2. If small dataset: fix all weights (treat CNN as fixed feature extractor), retrain only the classifier

i.e. swap the Softmax layer at the end



3. If you have medium sized dataset, **"finetune"** instead: use the old weights as initialization, train the full network or only some of the higher layers

retrain bigger portion of the network, or even all of it.

Source: Andrej Karpathy & Fei-Fei Li

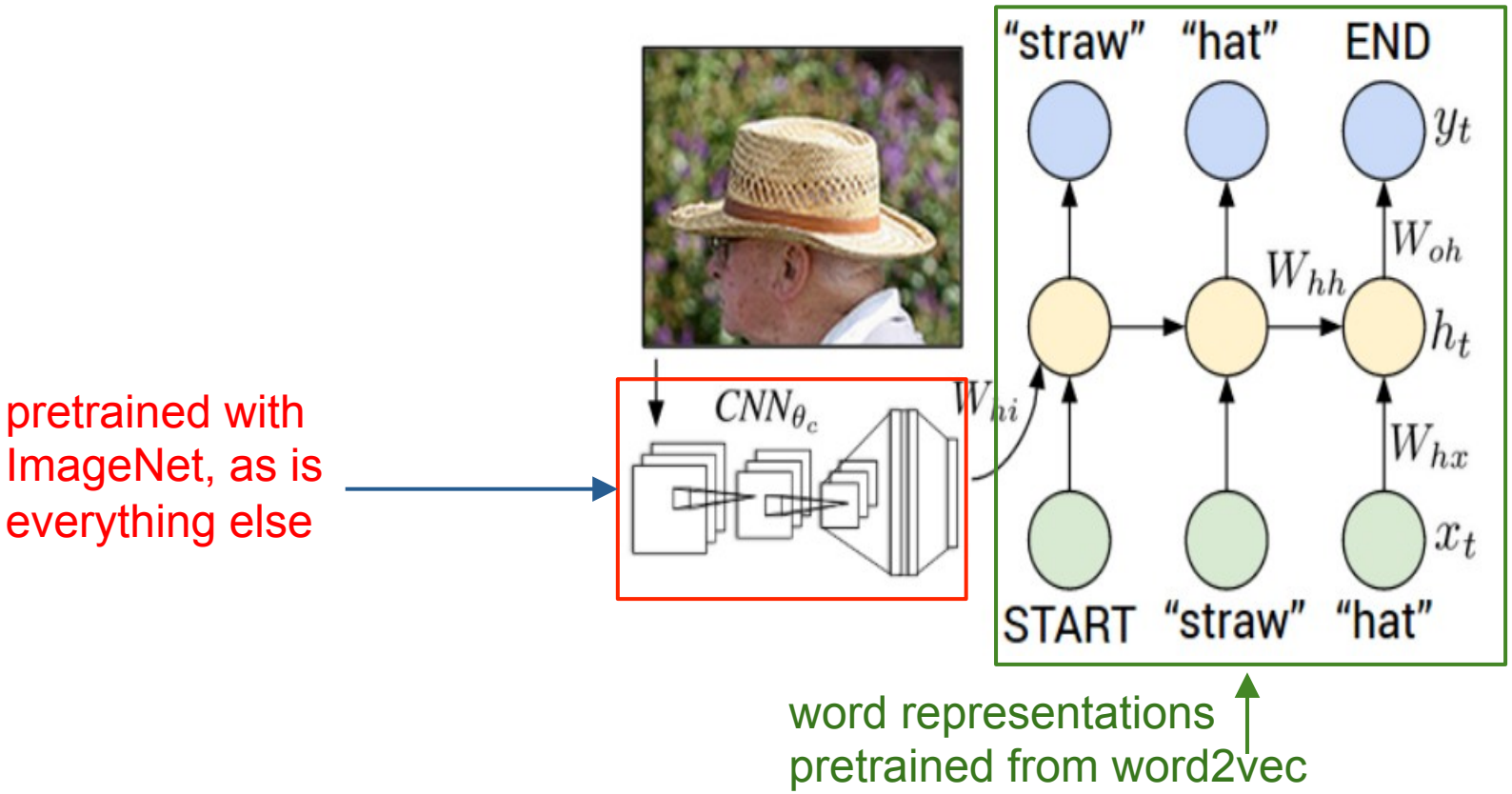
more generic

more specific

	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Source: Andrej Karpathy & Fei-Fei Li

Transfer learning with CNNs is pervasive…
(it's the norm, not an exception)



Source: Andrej Karpathy & Fei-Fei Li